Gromov hyperbolic spaces in Isabelle

Sebastien Gouezel

Abstract

A geodesic metric space is Gromov hyperbolic if all its geodesic triangles are thin, i.e., every side is contained in a fixed thickening of the two other sides. While this definition looks innocuous, it has proved extremely important and versatile in modern geometry since its introduction by Gromov. We formalize the basic classical properties of Gromov hyperbolic spaces, notably the Morse lemma asserting that quasigeodesics are close to geodesics, the invariance of hyperbolicity under quasi-isometries, we define and study the Gromov boundary and its associated distance, and prove that a quasi-isometry between Gromov hyperbolic spaces extends to a homeomorphism of the boundaries. We also classify the isometries of hyperbolic spaces into elliptic, parabolic and loxodromic ones, both in terms of translation length and of fixed points at infinity. We also prove a less classical theorem, by Bonk and Schramm, asserting that a Gromov hyperbolic space embeds isometrically in a geodesic Gromov-hyperbolic space. As the original proof uses a transfinite sequence of Cauchy completions, this is an interesting formalization exercise. Along the way, we introduce basic material on isometries, quasi-isometries, geodesic spaces, the Hausdorff distance, the Cauchy completion of a metric space, and the exponential on extended real numbers.

Contents

1 Additions to the library

theory *Library-Complements*
imports *HOL−Analysis.Analysis HOL−Cardinals.Cardinal-Order-Relation*
begin

1.1 Mono intros

We have a lot of (large) inequalities to prove. It is very convenient to have a set of introduction rules for this purpose (a lot should be added to it, I have put here all the ones I needed).

The typical use case is when one wants to prove some inequality, say $\exp(x * x) \leq y + \exp(1 + z * z + y)$, assuming $y \geq 0$ and $0 \leq x \leq z$. One would write it has

```
have "0 + \exp(0 + x * x + 0) < = y + \exp(1 + z * z + y)"
using ‘y > = 0‘ ‘x < = z‘ by (intro mono_intros)
```

When the left and right hand terms are written in completely analogous ways as above, then the introduction rules (that contain monotonicity of addition, of the exponential, and so on) reduce this to comparison of elementary terms in the formula. This is a very naive strategy, that fails in many situations, but that is very efficient when used correctly.

named-theorems *mono-intros structural introduction rules to prove inequalities*
declare *le-imp-neg-le* [*mono-intros*]
declare *add-left-mono* [*mono-intros*]
declare *add-right-mono* [*mono-intros*]
declare *add-strict-left-mono* [*mono-intros*]
declare *add-strict-right-mono* [*mono-intros*]
declare *add-mono* [*mono-intros*]
declare *add-less-le-mono* [*mono-intros*]
declare *diff-right-mono* [*mono-intros*]
declare *diff-left-mono* [*mono-intros*]
declare *diff-mono* [*mono-intros*]
declare *mult-left-mono* [*mono-intros*]
declare *mult-right-mono* [*mono-intros*]
declare *mult-mono* [*mono-intros*]
declare *max.mono* [*mono-intros*]
declare *min.mono* [*mono-intros*]
declare *power-mono* [*mono-intros*]
declare *ln-ge-zero* [*mono-intros*]
declare *ln-le-minus-one* [*mono-intros*]
declare *ennreal-minus-mono* [*mono-intros*]
declare *ennreal-leI* [*mono-intros*]
declare *e2ennreal-mono* [*mono-intros*]
declare *enn2ereal-nonneg* [*mono-intros*]
declare *zero-le* [*mono-intros*]
declare *top-greatest* [*mono-intros*]
declare *bot-least* [*mono-intros*]
declare *dist-triangle* [*mono-intros*]
declare *dist-triangle2* [*mono-intros*]
declare *dist-triangle3* [*mono-intros*]
declare *exp-ge-add-one-self* [*mono-intros*]
declare *exp-gt-one* [*mono-intros*]
declare *exp-less-mono* [*mono-intros*]
declare *dist-triangle* [*mono-intros*]
declare *abs-triangle-ineq* [*mono-intros*]
declare *abs-triangle-ineq2* [*mono-intros*]
declare *abs-triangle-ineq2-sym* [*mono-intros*]
declare *abs-triangle-ineq3* [*mono-intros*]
declare *abs-triangle-ineq4* [*mono-intros*]
declare *Liminf-le-Limsup* [*mono-intros*]
declare *ereal-liminf-add-mono* [*mono-intros*]
declare *le-of-int-ceiling* [*mono-intros*]
declare *ereal-minus-mono* [*mono-intros*]
declare *infdist-triangle* [*mono-intros*]
declare *divide-right-mono* [*mono-intros*]
declare *self-le-power* [*mono-intros*]

```
lemma ln-le-cancelI [mono-intros]:
  assumes (0::real) < x x ≤ y
  shows ln x ≤ ln y
using assms by auto

lemma exp-le-cancelI [mono-intros]:
  assumes x ≤ (y::real)
  shows exp x ≤ exp y
using assms by simp

lemma mult-ge1-mono [mono-intros]:
  assumes a ≥ (0::'a::linordered-idom) b ≥ 1
  shows a ≤ a * b a ≤ b * a
using assms mult-le-cancel-left1 mult-le-cancel-right1 by force+
```

A few convexity inequalities we will need later on.

```
lemma xy-le-uxx-vyy [mono-intros]:
  assumes u > 0 u * v = (1::real)
  shows x * y ≤ u * x^2/2 + v * y^2/2
proof −
  have v > 0 using assms
   by (metis (full-types) dual-order.strict-implies-order le-less-linear mult-nonneg-nonpos
not-one-le-zero)
  then have *: sqrt u * sqrt v = 1
    using assms by (metis real-sqrt-mult real-sqrt-one)
  have (sqrt u * x − sqrt v * y)^2 ≥ 0 by auto
  then have u * x^2 + v * y^2 − 2 * 1 * x * y ≥ 0
     unfolding power2-eq-square *[symmetric] using ‹u > 0› ‹v > 0› by (auto
simp add: algebra-simps)
  then show ?thesis by (auto simp add: algebra-simps divide-simps)
qed

lemma xy-le-xx-yy [mono-intros]:
  x * y ≤ x^2/2 + y^2/2 for x y::real
using xy-le-uxx-vyy[of 1 1] by auto

lemma ln-squared-bound [mono-intros]:
  (ln x)^2 ≤ 2 * x − 2 if x ≥ 1 for x::real
proof −
  define f where f = (λx::real. 2 * x − 2 − ln x * ln x)
  have *: DERIV f x :> 2 − 2 * ln x / x if x > 0 for x::real
    unfolding f-def using that by (auto intro!: derivative-eq-intros)
  have f 1 ≤ f x if x ≥ 1 for x
  proof (rule DERIV-nonneg-imp-nondecreasing[OF that])
    fix t::real assume t ≥ 1
    show ∃y. (f has-real-derivative y) (at t) ∧ 0 ≤ y
      apply (rule exI[of - 2 − 2 * ln t / t])
      using *[of t] ‹t ≥ 1› by (auto simp add: divide-simps ln-bound)
  qed
```

then show *?thesis* **unfolding** *f-def power2-eq-square* **using** *that* **by** *auto*
qed

In the next lemma, the assumptions are too strong (negative numbers less than -1 also work well to have a square larger than 1), but in practice one proves inequalities with nonnegative numbers, so this version is really the useful one for `mono_intros`.

lemma *mult-ge1-powers* [*mono-intros*]:
  assumes $a \geq (1::'a::linordered\text{-}idom)$
  shows $1 \leq a * a$ $1 \leq a * a * a$ $1 \leq a * a * a * a$
using *assms* **by** (*meson assms dual-order.trans mult-ge1-mono*(*1*) *zero-le-one*)+

lemmas [*mono-intros*] = *ln-bound*

lemma *mono-cSup*:
  fixes f :: $'a::conditionally\text{-}complete\text{-}lattice \Rightarrow 'b::conditionally\text{-}complete\text{-}lattice$
  assumes *bdd-above* A $A \neq \{\}$ *mono f*
  shows $Sup\ (f`A) \leq f\ (Sup\ A)$
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *cSUP-least cSup-upper mono-def*)

lemma *mono-cSup-bij*:
  fixes f :: $'a::conditionally\text{-}complete\text{-}linorder \Rightarrow 'b::conditionally\text{-}complete\text{-}linorder$
  assumes *bdd-above* A $A \neq \{\}$ *mono f bij f*
  shows $Sup\ (f`A) = f(Sup\ A)$
proof −
  have $Sup\ ((inv\ f)`(f`A)) \leq (inv\ f)\ (Sup\ (f`A))$
    apply (*rule mono-cSup*)
    using *mono-inv*[*OF assms*(*3*) *assms*(*4*)] *assms*(*2*) *bdd-above-image-mono*[*OF assms*(*3*) *assms*(*1*)] **by** *auto*
  then have $f\ (Sup\ ((inv\ f)`(f`A))) \leq Sup\ (f`A)$
    using *assms mono-def* **by** (*metis* (*no-types*, *opaque-lifting*) *bij-betw-imp-surj-on surj-f-inv-f*)
  moreover have $f\ (Sup\ ((inv\ f)`(f`A))) = f(Sup\ A)$
    using *assms* **by** (*simp add*: *bij-is-inj*)
  ultimately show *?thesis* **using** *mono-cSup*[*OF assms*(*1*) *assms*(*2*) *assms*(*3*)] **by** *auto*
qed

1.2 More topology

In situations of interest to us later on, convergence is well controlled only for sequences living in some dense subset of the space (but the limit can be anywhere). This is enough to establish continuity of the function, if the target space is well enough separated.

The statement we give below is very general, as we do not assume that the function is continuous inside the original set S, it will typically only be continuous at a set T contained in the closure of S. In many applications, T will be the closure of S, but we are also thinking of the case where one constructs an extension of a function inside a space, to its boundary, and the behaviour at the boundary is better than inside the space. The example we have in mind is the extension of a quasi-isometry to the boundary of a Gromov hyperbolic space.

In the following criterion, we assume that if u_n inside S converges to a point at the boundary T, then $f(u_n)$ converges (where f is some function inside). Then, we can extend the function f at the boundary, by picking the limit value of $f(u_n)$ for some sequence converging to u_n. Then the lemma asserts that f is continuous at every point b on the boundary.

The proof is done in two steps:

1. First, if v_n is another inside sequence tending to the same point b on the boundary, then $f(v_n)$ converges to the same value as $f(u_n)$: this is proved by considering the sequence w equal to u at even times and to v at odd times, and saying that $f(w_n)$ converges. Its limit is equal to the limit of $f(u_n)$ and of $f(v_n)$, so they have to coincide.

2. Now, consider a general sequence v (in the space or the boundary) converging to b. We want to show that $f(v_n)$ tends to $f(b)$. If v_n is inside S, we have already done it in the first step. If it is on the boundary, on the other hand, we can approximate it by an inside point w_n for which $f(w_n)$ is very close to $f(v_n)$. Then w_n is an inside sequence converging to b, hence $f(w_n)$ converges to $f(b)$ by the first step, and then $f(v_n)$ also converges to $f(b)$. The precise argument is more conveniently written by contradiction. It requires good separation properties of the target space.

First, we introduce the material to interpolate between two sequences, one at even times and the other one at odd times.

definition *even-odd-interpolate*::$(nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a)$
where *even-odd-interpolate u v n* $=$ (*if even n then u* (*n div 2*) *else v* (*n div 2*)))

lemma *even-odd-interpolate-compose*:
even-odd-interpolate (*f o u*) (*f o v*) $=$ *f o* (*even-odd-interpolate u v*)
unfolding *even-odd-interpolate-def comp-def* **by** *auto*

lemma *even-odd-interpolate-filterlim*:
filterlim u F sequentially $\wedge$ *filterlim v F sequentially* $\longleftrightarrow$ *filterlim* (*even-odd-interpolate u v*) *F sequentially*
proof (*auto*)
assume *H*: *filterlim* (*even-odd-interpolate u v*) *F sequentially*
define *r*::*nat* $\Rightarrow$ *nat* **where** *r* $=$ ($\lambda n.$ *2* $*$ *n*)
have *strict-mono r* **unfolding** *r-def strict-mono-def* **by** *auto*
then have *filterlim r sequentially sequentially*
by (*simp add*: *filterlim-subseq*)

```
  have filterlim (λn. (even-odd-interpolate u v) (r n)) F sequentially
    by (rule filterlim-compose[OF H filterlim-subseq[OF ‹strict-mono r›]])
  moreover have even-odd-interpolate u v (r n) = u n for n
    unfolding r-def even-odd-interpolate-def by auto
  ultimately show filterlim u F sequentially by auto
  define r::nat ⇒ nat where r = (λn. 2 ∗ n + 1)
  have strict-mono r unfolding r-def strict-mono-def by auto
  then have filterlim r sequentially sequentially
    by (simp add: filterlim-subseq)
  have filterlim (λn. (even-odd-interpolate u v) (r n)) F sequentially
    by (rule filterlim-compose[OF H filterlim-subseq[OF ‹strict-mono r›]])
  moreover have even-odd-interpolate u v (r n) = v n for n
    unfolding r-def even-odd-interpolate-def by auto
  ultimately show filterlim v F sequentially by auto
next
  assume H: filterlim u F sequentially filterlim v F sequentially
  show filterlim (even-odd-interpolate u v) F sequentially
  unfolding filterlim-iff eventually-sequentially proof (auto)
    fix P assume ∗: eventually P F
    obtain N1 where N1: ⋀n. n ≥ N1 ⟹ P (u n)
      using H(1) unfolding filterlim-iff eventually-sequentially using ∗ by auto
    obtain N2 where N2: ⋀n. n ≥ N2 ⟹ P (v n)
      using H(2) unfolding filterlim-iff eventually-sequentially using ∗ by auto
    have P (even-odd-interpolate u v n) if n ≥ 2 ∗ N1 + 2 ∗ N2 for n
    proof (cases even n)
      case True
      have n div 2 ≥ N1 using that by auto
       then show ?thesis unfolding even-odd-interpolate-def using True N1 by
auto
    next
      case False
      have n div 2 ≥ N2 using that by auto
       then show ?thesis unfolding even-odd-interpolate-def using False N2 by
auto
    qed
    then show ∃N. ∀n ≥ N. P (even-odd-interpolate u v n) by auto
  qed
qed
```

Then, we prove the continuity criterion for extensions of functions to the boundary T of a set S. The first assumption is that $f(u_n)$ converges when f converges to the boundary, and the second one that the extension of f to the boundary has been defined using the limit along some sequence tending to the point under consideration. The following criterion is the most general one, but this is not the version that is most commonly applied so we use a prime in its name.

```
lemma continuous-at-extension-sequentially':
  fixes f :: 'a::{first-countable-topology, t2-space} ⇒ 'b::t3-space
```

assumes $b \in T$
$\bigwedge u\ b.\ (\forall n.\ u\ n \in S) \Longrightarrow b \in T \Longrightarrow u \longrightarrow b \Longrightarrow convergent\ (\lambda n.\ f\ (u\ n))$
$\bigwedge b.\ b \in T \Longrightarrow \exists u.\ (\forall n.\ u\ n \in S) \wedge u \longrightarrow b \wedge ((\lambda n.\ f\ (u\ n)) \longrightarrow f\ b)$
shows *continuous* $(at\ b\ within\ (S \cup T))\ f$
proof $-$
have *first-step*: $(\lambda n.\ f\ (u\ n)) \longrightarrow f\ c$ **if** $\bigwedge n.\ u\ n \in S\ u \longrightarrow c\ c \in T$ **for** $u\ c$
proof $-$
obtain v **where** v: $\bigwedge n.\ v\ n \in S\ v \longrightarrow c\ (\lambda n.\ f\ (v\ n)) \longrightarrow f\ c$
using *assms*(*3*)[*OF* ‹$c \in T$›] **by** *blast*
then have A: *even-odd-interpolate* $u\ v \longrightarrow c$
unfolding *even-odd-interpolate-filterlim*[*symmetric*] **using** ‹$u \longrightarrow c$› **by** *auto*
moreover have B: $\forall n.$ *even-odd-interpolate* $u\ v\ n \in S$
using ‹$\bigwedge n.\ u\ n \in S$› ‹$\bigwedge n.\ v\ n \in S$› **unfolding** *even-odd-interpolate-def* **by** *auto*
have *convergent* $(\lambda n.\ f$ (*even-odd-interpolate* $u\ v\ n))$
by (*rule assms*(*2*)[*OF* B ‹$c \in T$› A])
then obtain m **where** $(\lambda n.\ f$ (*even-odd-interpolate* $u\ v\ n)) \longrightarrow m$
unfolding *convergent-def* **by** *auto*
then have *even-odd-interpolate* $(f\ o\ u)\ (f\ o\ v) \longrightarrow m$
unfolding *even-odd-interpolate-compose* **unfolding** *comp-def* **by** *auto*
then have $(f\ o\ u) \longrightarrow m\ (f\ o\ v) \longrightarrow m$
unfolding *even-odd-interpolate-filterlim*[*symmetric*] **by** *auto*
then have $m = f\ c$ **using** $v(3)$ **unfolding** *comp-def* **using** *LIMSEQ-unique* **by** *auto*
then show *?thesis* **using** ‹$(f\ o\ u) \longrightarrow m$› **unfolding** *comp-def* **by** *auto*
qed
show *continuous* $(at\ b\ within\ (S \cup T))\ f$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then obtain U **where** U: *open* $U\ f\ b \in U\ \neg(\forall_F\ x\ in\ at\ b\ within\ S \cup T.\ f\ x \in U)$
unfolding *continuous-within tendsto-def*[**where** $l = f\ b$] **using** *sequentially-imp-eventually-nhds-within* **by** *auto*
have $\exists\ V\ W.$ *open* $V \wedge$ *open* $W \wedge f\ b \in V \wedge (UNIV - U) \subseteq W \wedge V \cap W = \{\}$
apply (*rule t3-space*) **using** U **by** *auto*
then obtain $V\ W$ **where** VW: *open* V *open* $W\ f\ b \in V\ UNIV - U \subseteq W\ V \cap W = \{\}$
by *auto*

obtain $A :: nat \Rightarrow 'a\ set$ **where** $*$:
$\bigwedge i.$ *open* $(A\ i)$
$\bigwedge i.\ b \in A\ i$
$\bigwedge F.\ \forall n.\ F\ n \in A\ n \Longrightarrow F \longrightarrow b$
by (*rule first-countable-topology-class.countable-basis*) *blast*

with $*\ U(3)$ **have** $\exists F.\ \forall n.\ F\ n \in S \cup T \wedge F\ n \in A\ n \wedge \neg\ (f(F\ n) \in U)$
unfolding *at-within-def eventually-inf-principal eventually-nhds*
by (*intro choice*) (*meson DiffE*)
then obtain F **where** F: $\bigwedge n.\ F\ n \in S \cup T\ \bigwedge n.\ F\ n \in A\ n\ \bigwedge n.\ f(F\ n) \notin U$
by *auto*

have $\exists y.\ y \in S \wedge y \in A\ n \wedge f\ y \in W$ **for** n
proof (*cases* $F\ n \in S$)
case *True*
show *?thesis* **apply** (*rule exI*[*of* - $F\ n$]) **using** $F\ VW$ *True* **by** *auto*
next
case *False*
then have $F\ n \in T$ **using** ‹$F\ n \in S \cup T$› **by** *auto*
obtain u **where** u: $\bigwedge p.\ u\ p \in S\ u \longrightarrow F\ n\ (\lambda p.\ f\ (u\ p)) \longrightarrow f(F\ n)$
using *assms*(*3*)[*OF* ‹$F\ n \in T$›] **by** *auto*
moreover have $f(F\ n) \in W$ **using** $F\ VW$ **by** *auto*
ultimately have *eventually* $(\lambda p.\ f\ (u\ p) \in W)$ *sequentially*
using ‹*open* W› **by** (*simp add*: *tendsto-def*)
moreover have *eventually* $(\lambda p.\ u\ p \in A\ n)$ *sequentially*
using ‹$F\ n \in A\ n$› u ‹*open* $(A\ n)$› **by** (*simp add*: *tendsto-def*)
ultimately have $\exists p.\ f(u\ p) \in W \wedge u\ p \in A\ n$
using *eventually-False-sequentially eventually-elim2* **by** *blast*
then show *?thesis* **using** $u(1)$ **by** *auto*
qed
then have $\exists u.\ \forall n.\ u\ n \in S \wedge u\ n \in A\ n \wedge f\ (u\ n) \in W$
by (*auto intro*: *choice*)
then obtain u **where** u: $\bigwedge n.\ u\ n \in S\ \bigwedge n.\ u\ n \in A\ n\ \bigwedge n.\ f\ (u\ n) \in W$
by *blast*
then have $u \longrightarrow b$ **using** $*(3)$ **by** *auto*
then have $(\lambda n.\ f\ (u\ n)) \longrightarrow f\ b$ **using** *first-step assms* u **by** *auto*
then have *eventually* $(\lambda n.\ f\ (u\ n) \in V)$ *sequentially*
using VW **by** (*simp add*: *tendsto-def*)
then have $\exists n.\ f\ (u\ n) \in V$
using *eventually-False-sequentially eventually-elim2* **by** *blast*
then show *False*
using $u(3)$ ‹$V \cap W = \{\}$› **by** *auto*
qed
qed

We can specialize the previous statement to the common case where one already knows the sequential continuity of f along sequences in S converging to a point in T. This will be the case in most –but not all– applications. This is a straightforward application of the above criterion.

proposition *continuous-at-extension-sequentially*:
fixes f :: *'a*::{*first-countable-topology*, *t2-space*} $\Rightarrow$ *'b*::*t3-space*
assumes $a \in T$
$T \subseteq$ *closure* S
$\bigwedge u\ b.\ (\forall n.\ u\ n \in S) \Longrightarrow b \in T \Longrightarrow u \longrightarrow b \Longrightarrow (\lambda n.\ f\ (u\ n)) \longrightarrow f\ b$

shows *continuous* (*at a within* ($S \cup T$)) *f*
apply (*rule continuous-at-extension-sequentially*′[*OF* ‹*a* ∈ *T*›])
using *assms*(*3*) *convergent-def* **apply** *blast*
by (*metis assms*(*2*) *assms*(*3*) *closure-sequential subset-iff*)

We also give global versions. We can only express the continuity on T, so this is slightly weaker than the previous statements since we are not saying anything on inside sequences tending to T – but in cases where T contains S these statements contain all the information.

lemma *continuous-on-extension-sequentially*′:
fixes *f* :: ′*a*::{*first-countable-topology*, *t2-space*} ⇒ ′*b*::*t3-space*
assumes $\bigwedge u\ b.\ (\forall n.\ u\ n \in S) \Longrightarrow b \in T \Longrightarrow u \longrightarrow b \Longrightarrow$ *convergent* ($\lambda n.\ f$ $(u\ n)$)
$\bigwedge b.\ b \in T \Longrightarrow \exists u.\ (\forall n.\ u\ n \in S) \wedge u \longrightarrow b \wedge ((\lambda n.\ f\ (u\ n)) \longrightarrow$ $f\ b)$
shows *continuous-on T f*
unfolding *continuous-on-eq-continuous-within* **apply** (*auto intro*!: *continuous-within-subset*[*of* - $S \cup T$ *f T*])
by (*intro continuous-at-extension-sequentially*′[*OF* - *assms*], *auto*)

lemma *continuous-on-extension-sequentially*:
fixes *f* :: ′*a*::{*first-countable-topology*, *t2-space*} ⇒ ′*b*::*t3-space*
assumes $T \subseteq$ *closure S*
$\bigwedge u\ b.\ (\forall n.\ u\ n \in S) \Longrightarrow b \in T \Longrightarrow u \longrightarrow b \Longrightarrow (\lambda n.\ f\ (u\ n)) \longrightarrow$ $f\ b$
shows *continuous-on T f*
unfolding *continuous-on-eq-continuous-within* **apply** (*auto intro*!: *continuous-within-subset*[*of* - $S \cup T$ *f T*])
by (*intro continuous-at-extension-sequentially*[*OF* - *assms*], *auto*)

1.2.1 Homeomorphisms

A variant around the notion of homeomorphism, which is only expressed in terms of the function and not of its inverse.

definition *homeomorphism-on*::′*a set* ⇒ (′*a*::*topological-space* ⇒ ′*b*::*topological-space*) ⇒ *bool*
where *homeomorphism-on S f* = (∃ *g*. *homeomorphism S* (*f*‘*S*) *f g*)

lemma *homeomorphism-on-continuous*:
assumes *homeomorphism-on S f*
shows *continuous-on S f*
using *assms* **unfolding** *homeomorphism-on-def homeomorphism-def* **by** *auto*

lemma *homeomorphism-on-bij*:
assumes *homeomorphism-on S f*
shows *bij-betw f S* (*f*‘*S*)
using *assms* **unfolding** *homeomorphism-on-def homeomorphism-def* **by** *auto* (*metis inj-on-def inj-on-imp-bij-betw*)

lemma *homeomorphism-on-homeomorphic*:
  assumes *homeomorphism-on S f*
  shows *S homeomorphic* (*f*‘*S*)
using *assms* **unfolding** *homeomorphism-on-def homeomorphic-def* **by** *auto*

lemma *homeomorphism-on-compact*:
  fixes f::$'a$::*topological-space* ⇒ $'b$::*t2-space*
  assumes *continuous-on S f*
    compact S
    inj-on f S
  shows *homeomorphism-on S f*
unfolding *homeomorphism-on-def* **using** *homeomorphism-compact*[*OF assms*(*2*) *assms*(*1*) - *assms*(*3*)] **by** *auto*

lemma *homeomorphism-on-subset*:
  assumes *homeomorphism-on S f*
    $T \subseteq S$
  shows *homeomorphism-on T f*
using *assms homeomorphism-of-subsets* **unfolding** *homeomorphism-on-def* **by** *blast*

lemma *homeomorphism-on-empty* [*simp*]:
  homeomorphism-on {} *f*
unfolding *homeomorphism-on-def* **using** *homeomorphism-empty*[*of f*] **by** *auto*

lemma *homeomorphism-on-cong*:
  assumes *homeomorphism-on X f*
    $X' = X$ $\bigwedge x.\ x \in X \Longrightarrow f'\ x = f\ x$
  shows *homeomorphism-on* X' f'
proof −
  obtain *g* **where** *g*:*homeomorphism X* (*f*‘*X*) *f g*
    using *assms* **unfolding** *homeomorphism-on-def* **by** *auto*
  have *homeomorphism* X' (f'‘X') f' *g*
    apply (*rule homeomorphism-cong*[*OF g*]) **using** *assms* **by** (*auto simp add*: *rev-image-eqI*)
  then show *?thesis*
    unfolding *homeomorphism-on-def* **by** *auto*
qed

lemma *homeomorphism-on-inverse*:
  fixes f::$'a$::*topological-space* ⇒ $'b$::*topological-space*
  assumes *homeomorphism-on X f*
  shows *homeomorphism-on* (*f*‘*X*) (*inv-into X f*)
proof −
  obtain *g* **where** *g*: *homeomorphism X* (*f*‘*X*) *f g*
    using *assms* **unfolding** *homeomorphism-on-def* **by** *auto*
  then have *g*‘*f*‘*X* = *X*
    by (*simp add*: *homeomorphism-def*)
  then have *homeomorphism-on* (*f*‘*X*) *g*

unfolding *homeomorphism-on-def* **using** *homeomorphism-symD*[*OF g*] **by** *auto*
moreover have $g\ x = \textit{inv-into}\ X\ f\ x$ **if** $x \in f`X$ **for** x
using *g that* **unfolding** *homeomorphism-def* **by** (*auto*, *metis f-inv-into-f inv-into-into that*)
ultimately show *?thesis*
using *homeomorphism-on-cong* **by** *force*
qed

Characterization of homeomorphisms in terms of sequences: a map is a homeomorphism if and only if it respects convergent sequences.

lemma *homeomorphism-on-compose*:
assumes *homeomorphism-on S f*
$x \in S$
eventually $(\lambda n.\ u\ n \in S)\ F$
shows $(u \longrightarrow x)\ F \longleftrightarrow ((\lambda n.\ f\ (u\ n)) \longrightarrow f\ x)\ F$
proof
assume $(u \longrightarrow x)\ F$
then show $((\lambda n.\ f\ (u\ n)) \longrightarrow f\ x)\ F$
using *continuous-on-tendsto-compose*[*OF homeomorphism-on-continuous*[*OF assms*(*1*)] - *assms*(*2*) *assms*(*3*)] **by** *simp*
next
assume $*$: $((\lambda n.\ f\ (u\ n)) \longrightarrow f\ x)\ F$
have *I*: $\textit{inv-into}\ S\ f\ (f\ y) = y$ **if** $y \in S$ **for** y
using *homeomorphism-on-bij*[*OF assms*(*1*)] **by** (*meson bij-betw-inv-into-left that*)
then have *A*: *eventually* $(\lambda n.\ u\ n = \textit{inv-into}\ S\ f\ (f\ (u\ n)))\ F$
using *assms eventually-mono* **by** *force*
have $((\lambda n.\ (\textit{inv-into}\ S\ f)\ (f\ (u\ n))) \longrightarrow (\textit{inv-into}\ S\ f)\ (f\ x))\ F$
apply (*rule continuous-on-tendsto-compose*[*OF homeomorphism-on-continuous*[*OF homeomorphism-on-inverse*[*OF assms*(*1*)]] $*$])
using *assms eventually-mono* **by** (*auto*) *fastforce*
then show $(u \longrightarrow x)\ F$
unfolding *tendsto-cong*[*OF A*] *I*[*OF* ‹$x \in S$›] **by** *simp*
qed

lemma *homeomorphism-on-sequentially*:
fixes f::'a::{*first-countable-topology*, *t2-space*} ⇒ 'b::{*first-countable-topology*, *t2-space*}
assumes $\bigwedge x\ u.\ x \in S \Longrightarrow (\forall n.\ u\ n \in S) \Longrightarrow u \longrightarrow x \longleftrightarrow (\lambda n.\ f\ (u\ n)) \longrightarrow f\ x$
shows *homeomorphism-on S f*
proof −
have $x = y$ **if** $f\ x = f\ y\ x \in S\ y \in S$ **for** $x\ y$
proof −
have $(\lambda n.\ f\ x) \longrightarrow f\ y$ **using** *that* **by** *auto*
then have $(\lambda n.\ x) \longrightarrow y$ **using** *assms*(*1*) *that* **by** *auto*
then show $x = y$ **using** *LIMSEQ-unique* **by** *auto*
qed
then have *inj-on f S* **by** (*simp add*: *inj-on-def*)

have *Cf*: *continuous-on S f*
apply (*rule continuous-on-sequentiallyI*) **using** *assms* **by** *auto*
define g **where** $g = $ *inv-into S f*
have *Cg*: *continuous-on* (f‘S) g
proof (*rule continuous-on-sequentiallyI*)
fix v b **assume** H: $\forall n.\ v\ n \in f\ `\ S\ b \in f\ `\ S\ v \longlongrightarrow b$
define u **where** $u = (\lambda n.\ g\ (v\ n))$
define a **where** $a = g\ b$
have $u\ n \in S\ f\ (u\ n) = v\ n$ **for** n
unfolding *u-def g-def* **using** $H(1)$ **by** (*auto simp add*: *inv-into-into f-inv-into-f*)
have $a \in S\ f\ a = b$
unfolding *a-def g-def* **using** $H(2)$ **by** (*auto simp add*: *inv-into-into f-inv-into-f*)
show $(\lambda n.\ g(v\ n)) \longlongrightarrow g\ b$
unfolding *u-def*[*symmetric*] *a-def*[*symmetric*] **apply** (*rule iffD2*[*OF assms*])
using ‹$\bigwedge n.\ u\ n \in S$› ‹$a \in S$› ‹$v \longlongrightarrow b$›
unfolding ‹$\bigwedge n.\ f\ (u\ n) = v\ n$› ‹$f\ a = b$› **by** *auto*
qed
have *homeomorphism S* (f‘S) f g
apply (*rule homeomorphismI*[*OF Cf Cg*]) **unfolding** *g-def* **using** ‹*inj-on f S*›
by *auto*
then show *?thesis*
unfolding *homeomorphism-on-def* **by** *auto*
qed

lemma *homeomorphism-on-UNIV-sequentially*:
fixes f::$'a$::{*first-countable-topology*, *t2-space*} $\Rightarrow$ $'b$::{*first-countable-topology*, *t2-space*}
assumes $\bigwedge x\ u.\ u \longlongrightarrow x \longleftrightarrow (\lambda n.\ f\ (u\ n)) \longlongrightarrow f\ x$
shows *homeomorphism-on UNIV f*
using *assms* **by** (*auto intro*!: *homeomorphism-on-sequentially*)

Now, we give similar characterizations in terms of sequences living in a dense subset. As in the sequential continuity criteria above, we first give a very general criterion, where the map does not have to be continuous on the approximating set S, only on the limit set T, without any a priori identification of the limit. Then, we specialize this statement to a less general but often more usable version.

lemma *homeomorphism-on-extension-sequentially-precise*:
fixes f::$'a$::{*first-countable-topology*, *t3-space*} $\Rightarrow$ $'b$::{*first-countable-topology*, *t3-space*}
assumes $\bigwedge u\ b.\ (\forall n.\ u\ n \in S) \Longrightarrow b \in T \Longrightarrow u \longlongrightarrow b \Longrightarrow$ *convergent* $(\lambda n.\ f$
$(u\ n))$
$\bigwedge u\ c.\ (\forall n.\ u\ n \in S) \Longrightarrow c \in f`T \Longrightarrow (\lambda n.\ f\ (u\ n)) \longlongrightarrow c \Longrightarrow$ *convergent*
u
$\bigwedge b.\ b \in T \Longrightarrow \exists u.\ (\forall n.\ u\ n \in S) \wedge u \longlongrightarrow b \wedge ((\lambda n.\ f\ (u\ n)) \longlongrightarrow$
$f\ b)$
$\bigwedge n.\ u\ n \in S \cup T\ l \in T$
shows $u \longlongrightarrow l \longleftrightarrow (\lambda n.\ f\ (u\ n)) \longlongrightarrow f\ l$
proof
assume H: $u \longlongrightarrow l$
have *continuous* (*at* l *within* $(S \cup T)$) f

apply (*rule continuous-at-extension-sequentially*′[*OF* ‹$l \in T$›]) **using** *assms*(*1*) *assms*(*3*) **by** *auto*
then show ($\lambda n.\ f\ (u\ n)$) $\longrightarrow f\ l$
apply (*rule continuous-within-tendsto-compose*) **using** *H assms*(*4*) **by** *auto*
next

For the reverse implication, we would like to use the continuity criterion `continuous_at_extension_sequentially'` applied to the inverse of f. Unfortunately, this inverse is only well defined on T, while our sequence takes values in $S \cup T$. So, instead, we redo by hand the proof of the continuity criterion, but in the opposite direction.

assume H: ($\lambda n.\ f\ (u\ n)$) $\longrightarrow f\ l$
show $u \longrightarrow l$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then obtain U **where** U: *open* U $l \in U$ $\neg(\forall_F\ n$ *in sequentially*. $u\ n \in U)$
unfolding *continuous-within tendsto-def*[**where** $l = l$] **using** *sequentially-imp-eventually-nhds-within* **by** *auto*
obtain A :: *nat* $\Rightarrow$ ′*b set* **where** $*$:
$\bigwedge i.$ *open* $(A\ i)$
$\bigwedge i.\ f\ l \in A\ i$
$\bigwedge F.\ \forall n.\ F\ n \in A\ n \Longrightarrow F \longrightarrow f\ l$
by (*rule first-countable-topology-class.countable-basis*) *blast*
have B: *eventually* ($\lambda n.\ f\ (u\ n) \in A\ i$) *sequentially* **for** i
using ‹*open* $(A\ i)$› ‹$f\ l \in A\ i$› *H topological-tendstoD* **by** *fastforce*
have M: $\exists r.\ r \geq N \wedge (u\ r \notin U) \wedge f\ (u\ r) \in A\ i$ **for** $N\ i$
using U(*3*) B[*of i*] **unfolding** *eventually-sequentially* **by** (*meson dual-order.trans le-cases*)
have $\exists r.\ \forall n.\ (u\ (r\ n) \notin U \wedge f\ (u\ (r\ n)) \in A\ n) \wedge r\ (Suc\ n) \geq r\ n + 1$
apply (*rule dependent-nat-choice*) **using** M **by** *auto*
then obtain r **where** r: $\bigwedge n.\ u\ (r\ n) \notin U$ $\bigwedge n.\ f\ (u\ (r\ n)) \in A\ n$ $\bigwedge n.\ r\ (Suc\ n) \geq r\ n + 1$
by *auto*
then have *strict-mono r*
by (*metis Suc-eq-plus1 Suc-le-lessD strict-monoI-Suc*)

have $\exists V\ W.$ *open* $V \wedge$ *open* $W \wedge l \in V \wedge (UNIV - U) \subseteq W \wedge V \cap W = \{\}$
apply (*rule t3-space*) **using** U **by** *auto*
then obtain $V\ W$ **where** VW: *open* V *open* W $l \in V$ $UNIV - U \subseteq W$ $V \cap W = \{\}$
by *auto*

have $\exists z.\ z \in S \wedge f\ z \in A\ n \wedge z \in W$ **for** n
proof −
define z **where** $z = u\ (r\ n)$
have $f\ z \in A\ n$ **unfolding** *z-def* **using** r(*2*) **by** *auto*
have $z \in S \cup T$ $z \notin U$
unfolding *z-def* **using** r(*1*) *assms*(*4*) **by** *auto*

```
      then have z ∈ W using VW by auto
      show ?thesis
      proof (cases z ∈ T)
        case True
        obtain u::nat ⇒ 'a where u: ⋀p. u p ∈ S u ⟶ z (λp. f (u p)) ⟶
f z
          using assms(3)[OF ‹z ∈ T›] by auto
        then have eventually (λp. f (u p) ∈ A n) sequentially
          using ‹open (A n)› ‹f z ∈ A n› unfolding tendsto-def by simp
        moreover have eventually (λp. u p ∈ W) sequentially
          using ‹open W› ‹z ∈ W› u unfolding tendsto-def by simp
        ultimately have ∃p. u p ∈ W ∧ f (u p) ∈ A n
          using eventually-False-sequentially eventually-elim2 by blast
        then show ?thesis using u(1) by auto
      next
        case False
        then have z ∈ S using ‹z ∈ S ∪ T› by auto
        then show ?thesis using ‹f z ∈ A n› ‹z ∈ W› by auto
      qed
    qed
    then have ∃v. ∀n. v n ∈ S ∧ f (v n) ∈ A n ∧ v n ∈ W
      by (auto intro: choice)
    then obtain v where v: ⋀n. v n ∈ S ⋀n. f (v n) ∈ A n ⋀n. v n ∈ W
      by blast
    then have I: (λn. f (v n)) ⟶ f l using *(3) by auto

    obtain w where w: ⋀n. w n ∈ S w ⟶ l ((λn. f (w n)) ⟶ f l)
      using assms(3)[OF ‹l ∈ T›] by auto
    have even-odd-interpolate (f o v) (f o w) ⟶ f l
     unfolding even-odd-interpolate-filterlim[symmetric] comp-def using v w I by
auto
    then have *: (λn. f (even-odd-interpolate v w n)) ⟶ f l
      unfolding even-odd-interpolate-compose unfolding comp-def by auto
    have convergent (even-odd-interpolate v w)
      apply (rule assms(2)[OF - - *])
      unfolding even-odd-interpolate-def using v(1) w(1) ‹l ∈ T› by auto
    then obtain z where even-odd-interpolate v w ⟶ z
      unfolding convergent-def by auto
   then have *: v ⟶ z w ⟶ z unfolding even-odd-interpolate-filterlim[symmetric]
by auto
    then have z = l using v(2) w(2) LIMSEQ-unique by auto
    then have v ⟶ l using * by simp
    then have eventually (λn. v n ∈ V) sequentially
      using VW by (simp add: tendsto-def)
    then have ∃n. v n ∈ V
      using eventually-False-sequentially eventually-elim2 by blast
    then show False
      using v(3) ‹V ∩ W = {}› by auto
  qed
```

qed

lemma *homeomorphism-on-extension-sequentially′*:
fixes $f::'a::\{first\text{-}countable\text{-}topology, t3\text{-}space\} \Rightarrow 'b::\{first\text{-}countable\text{-}topology, t3\text{-}space\}$
assumes $\bigwedge u\ b.\ (\forall n.\ u\ n \in S) \Longrightarrow b \in T \Longrightarrow u \longrightarrow b \Longrightarrow convergent\ (\lambda n.\ f\ (u\ n))$
$\bigwedge u\ c.\ (\forall n.\ u\ n \in S) \Longrightarrow c \in f`T \Longrightarrow (\lambda n.\ f\ (u\ n)) \longrightarrow c \Longrightarrow convergent\ u$
$\bigwedge b.\ b \in T \Longrightarrow \exists u.\ (\forall n.\ u\ n \in S) \wedge u \longrightarrow b \wedge ((\lambda n.\ f\ (u\ n)) \longrightarrow f\ b)$
shows *homeomorphism-on T f*
apply (*rule homeomorphism-on-sequentially*, *rule homeomorphism-on-extension-sequentially-precise*[*of S T*])
using *assms* **by** *auto*

proposition *homeomorphism-on-extension-sequentially*:
fixes $f::'a::\{first\text{-}countable\text{-}topology, t3\text{-}space\} \Rightarrow 'b::\{first\text{-}countable\text{-}topology, t3\text{-}space\}$
assumes $\bigwedge u\ b.\ (\forall n.\ u\ n \in S) \Longrightarrow u \longrightarrow b \longleftrightarrow (\lambda n.\ f\ (u\ n)) \longrightarrow f\ b$
$T \subseteq closure\ S$
shows *homeomorphism-on T f*
apply (*rule homeomorphism-on-extension-sequentially′*[*of S*])
using *assms*(*1*) *convergent-def* **apply** *fastforce*
using *assms*(*1*) *convergent-def* **apply** *blast*
by (*metis assms*(*1*) *assms*(*2*) *closure-sequential subsetCE*)

lemma *homeomorphism-on-UNIV-extension-sequentially*:
fixes $f::'a::\{first\text{-}countable\text{-}topology, t3\text{-}space\} \Rightarrow 'b::\{first\text{-}countable\text{-}topology, t3\text{-}space\}$
assumes $\bigwedge u\ b.\ (\forall n.\ u\ n \in S) \Longrightarrow u \longrightarrow b \longleftrightarrow (\lambda n.\ f\ (u\ n)) \longrightarrow f\ b$
$closure\ S = UNIV$
shows *homeomorphism-on UNIV f*
apply (*rule homeomorphism-on-extension-sequentially*[*of S*]) **using** *assms* **by** *auto*

1.2.2 Proper spaces

Proper spaces, i.e., spaces in which every closed ball is compact – or, equivalently, any closed bounded set is compact.

definition $proper::('a::metric\text{-}space)\ set \Rightarrow bool$
where $proper\ S \equiv (\forall\ x\ r.\ compact\ (cball\ x\ r \cap S))$

lemma *properI*:
assumes $\bigwedge x\ r.\ compact\ (cball\ x\ r \cap S)$
shows *proper S*
using *assms* **unfolding** *proper-def* **by** *auto*

lemma *proper-compact-cball*:
assumes $proper\ (UNIV::'a::metric\text{-}space\ set)$
shows $compact\ (cball\ (x::'a)\ r)$
using *assms* **unfolding** *proper-def* **by** *auto*

lemma *proper-compact-bounded-closed*:
  assumes *proper* (*UNIV*::′a::*metric-space set*) *closed* (*S*::′*a set*) *bounded S*
  shows *compact S*
proof −
  obtain *x r* **where** $S \subseteq$ *cball x r*
    using ‹*bounded S*› *bounded-subset-cball* **by** *blast*
  then have ∗: $S = S \cap$ *cball x r*
    by *auto*
  show *?thesis*
    apply (*subst* ∗, *rule closed-Int-compact*) **using** *assms* **unfolding** *proper-def* **by** *auto*
qed

lemma *proper-real* [*simp*]:
  proper (*UNIV*::*real set*)
unfolding *proper-def* **by** *auto*

lemma *complete-of-proper*:
  assumes *proper S*
  shows *complete S*
proof −
  have $\exists\, l{\in}S.\ u \longrightarrow l$ **if** *Cauchy u* $\bigwedge n.\ u\ n \in S$ **for** *u*
  proof −
    have *bounded* (*range u*)
      using ‹*Cauchy u*› *cauchy-imp-bounded* **by** *auto*
    then obtain *x r* **where** ∗: $\bigwedge n.$ *dist x* $(u\ n) \le r$
      unfolding *bounded-def* **by** *auto*
    then have $u\ n \in$ (*cball x r*) $\cap S$ **for** *n* **using** ‹$u\ n \in S$› **by** *auto*
    moreover have *complete* ((*cball x r*) $\cap S$)
      apply (*rule compact-imp-complete*) **using** *assms* **unfolding** *proper-def* **by** *auto*
    ultimately show *?thesis*
      unfolding *complete-def* **using** ‹*Cauchy u*› **by** *auto*
  qed
  then show *?thesis*
    unfolding *complete-def* **by** *auto*
qed

lemma *proper-of-compact*:
  assumes *compact S*
  shows *proper S*
using *assms* **by** (*auto intro*: *properI*)

lemma *proper-Un*:
  assumes *proper A proper B*
  shows *proper* $(A \cup B)$
using *assms* **unfolding** *proper-def* **by** (*auto simp add*: *compact-Un inf-sup-distrib1*)

1.2.3 Miscellaneous topology

When manipulating the triangle inequality, it is very frequent to deal with 4 points (and automation has trouble doing it automatically). Even sometimes with 5 points...

lemma *dist-triangle4* [*mono-intros*]:
  dist x t $\leq$ *dist x y* + *dist y z* + *dist z t*
using *dist-triangle*[*of x z y*] *dist-triangle*[*of x t z*] **by** *auto*

lemma *dist-triangle5* [*mono-intros*]:
  dist x u $\leq$ *dist x y* + *dist y z* + *dist z t* + *dist t u*
using *dist-triangle4* [*of x u y z*] *dist-triangle*[*of z u t*] **by** *auto*

A thickening of a compact set is closed.

lemma *compact-has-closed-thickening*:
  assumes *compact C*
    continuous-on C f
  shows *closed* ($\bigcup x \in C$. *cball x* (*f x*))
proof (*auto simp add*: *closed-sequential-limits*)
  fix *u l* **assume** ∗: $\forall n::nat. \exists x\in C.$ *dist x* (*u n*) $\leq$ *f x u* $\longrightarrow$ *l*
  have $\exists x::nat \Rightarrow 'a. \forall n.\ x\ n \in C \wedge$ *dist* (*x n*) (*u n*) $\leq$ *f* (*x n*)
    apply (*rule choice*) **using** ∗ **by** *auto*
  then obtain *x::nat* $\Rightarrow$ *'a* **where** *x*: $\bigwedge n.\ x\ n \in C$ $\bigwedge n.$ *dist* (*x n*) (*u n*) $\leq$ *f* (*x n*)
    by *blast*
  obtain *r c* **where** *strict-mono r c* $\in$ *C* (*x o r*) $\longrightarrow$ *c*
   using *x*(*1*) ‹*compact C*› **by** (*meson compact-eq-seq-compact-metric seq-compact-def*)
  then have *c* $\in$ *C* **using** *x*(*1*) ‹*compact C*› **by** *auto*
  have *lim*: ($\lambda n.$ *f* (*x* (*r n*)) − *dist* (*x* (*r n*)) (*u* (*r n*))) $\longrightarrow$ *f c* − *dist c l*
    apply (*intro tendsto-intros*, *rule continuous-on-tendsto-compose*[*of C f*])
    using ∗(*2*) *x*(*1*) ‹(*x o r*) $\longrightarrow$ *c*› ‹*continuous-on C f*› ‹*c* $\in$ *C*› ‹*strict-mono r*› *LIMSEQ-subseq-LIMSEQ*
    unfolding *comp-def* **by** *auto*
  have *f c* − *dist c l* $\geq$ *0* **apply** (*rule LIMSEQ-le-const*[*OF lim*]) **using** *x*(*2*) **by** *auto*
  then show $\exists x\in C.$ *dist x l* $\leq$ *f x* **using** ‹*c* $\in$ *C*› **by** *auto*
qed

congruence rule for continuity. The assumption that $fy = gy$ is necessary since `at x` is the pointed neighborhood at x.

lemma *continuous-within-cong*:
  assumes *continuous* (*at y within S*) *f*
    eventually ($\lambda x.$ *f x* = *g x*) (*at y within S*)
    f y = *g y*
  shows *continuous* (*at y within S*) *g*
  using *assms continuous-within filterlim-cong* **by** *fastforce*

A function which tends to infinity at infinity, on a proper set, realizes its infimum

```
lemma continuous-attains-inf-proper:
  fixes f :: 'a::metric-space ⇒ 'b::linorder-topology
  assumes proper s a ∈ s
          continuous-on s f
          ⋀z. z ∈ s − cball a r ⟹ f z ≥ f a
  shows ∃x∈s. ∀y∈s. f x ≤ f y
proof (cases r ≥ 0)
  case True
  have ∃x∈cball a r ∩ s. ∀y ∈ cball a r ∩ s. f x ≤ f y
     apply (rule continuous-attains-inf) using assms True unfolding proper-def
apply (auto simp add: continuous-on-subset)
    using centre-in-cball by blast
  then obtain x where x: x ∈ cball a r ∩ s ⋀y. y ∈ cball a r ∩ s ⟹ f x ≤ f y
    by auto
  have f x ≤ f y if y ∈ s for y
  proof (cases y ∈ cball a r)
    case True
    then show ?thesis using x(2) that by auto
  next
    case False
    have f x ≤ f a
      apply (rule x(2)) using assms True by auto
    then show ?thesis using assms(4)[of y] that False by auto
  qed
  then show ?thesis using x(1) by auto
next
  case False
  show ?thesis
    apply (rule bexI[of - a]) using assms False by auto
qed
```

1.2.4 Measure of balls

The image of a ball by an affine map is still a ball, with explicit center and radius. (Now unused)

```
lemma affine-image-ball [simp]:
  (λy. R *R y + x) ‘ cball 0 1 = cball (x::('a::real-normed-vector)) |R|
proof
  have dist x (R *R y + x) ≤ |R| if dist 0 y ≤ 1 for y
  proof −
   have dist x (R *R y + x) = norm ((R *R y + x) − x) by (simp add: dist-norm)
    also have ... = |R| * norm y by auto
    finally show ?thesis using that by (simp add: mult-left-le)
  qed
  then show (λy. R *R y + x) ‘ cball 0 1 ⊆ cball x |R| by auto

  show cball x |R| ⊆ (λy. R *R y + x) ‘ cball 0 1
  proof (cases |R| = 0)
    case True
```

then have *cball* x $|R|$ = $\{x\}$ **by** *auto*
moreover have $x = R *_R 0 + x \wedge 0 \in$ *cball 0 1* **by** *auto*
ultimately show *?thesis* **by** *auto*
next
case *False*
have $z \in (\lambda y.\ R *_R y + x)$ ' *cball 0 1* **if** $z \in$ *cball* x $|R|$ **for** z
proof −
define y **where** $y = (z - x) /_R R$
have $R *_R y + x = z$ **unfolding** *y-def* **using** *False* **by** *auto*
moreover have $y \in$ *cball 0 1*
using ‹$z \in$ *cball* x $|R|$› *False* **unfolding** *y-def* **by** (*auto simp add*: *dist-norm*[*symmetric*] *divide-simps dist-commute*)
ultimately show *?thesis* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed
qed

From the rescaling properties of Lebesgue measure in a euclidean space, it follows that the measure of any ball can be expressed in terms of the measure of the unit ball.

lemma *lebesgue-measure-ball*:
assumes $R \geq 0$
shows *measure lborel* (*cball* (*x*::('*a*::*euclidean-space*)) *R*) = *R*^(*DIM*('*a*)) ∗ *measure lborel* (*cball* (*0*::'*a*) *1*)
emeasure lborel (*cball* (*x*::('*a*::*euclidean-space*)) *R*) = *R*^(*DIM*('*a*)) ∗ *emeasure lborel* (*cball* (*0*::'*a*) *1*)
apply (*simp add*: *assms content-cball*)
by (*simp add*: *assms emeasure-cball ennreal-mult' ennreal-power mult.commute*)

We show that the unit ball has positive measure – this is obvious, but useful. We could show it by arguing that it contains a box, whose measure can be computed, but instead we say that if the measure vanished then the measure of any ball would also vanish, contradicting the fact that the space has infinite measure. This avoids all computations.

lemma *lebesgue-measure-ball-pos*:
emeasure lborel (*cball* (*0*::'*a*::*euclidean-space*) *1*) > *0*
measure lborel (*cball* (*0*::'*a*::*euclidean-space*) *1*) > *0*
proof −
show *emeasure lborel* (*cball* (*0*::'*a*::*euclidean-space*) *1*) > *0*
proof (*rule ccontr*)
assume ¬(*emeasure lborel* (*cball* (*0*::'*a*::*euclidean-space*) *1*) > *0*)
then have *emeasure lborel* (*cball* (*0*::'*a*) *1*) = *0* **by** *auto*
then have *emeasure lborel* (*cball* (*0*::'*a*) *n*) = *0* **for** *n*::*nat*
using *lebesgue-measure-ball*(*2*)[*of real n 0*] **by** (*metis mult-zero-right of-nat-0-le-iff*)
then have *emeasure lborel* (⋃*n*. *cball* (*0*::'*a*) (*real n*)) = *0*
by (*metis* (*mono-tags*, *lifting*) *borel-closed closed-cball emeasure-UN-eq-0 imageE sets-lborel subsetI*)

moreover have $(\bigcup n.\ cball\ (0::'a)\ (real\ n)) = UNIV$ **by** (*auto simp add*: *real-arch-simple*)
ultimately show *False*
by *simp*
qed
moreover have *emeasure lborel* $(cball\ (0::'a::euclidean\text{-}space)\ 1) < \infty$
by (*rule emeasure-bounded-finite*, *auto*)
ultimately show *measure lborel* $(cball\ (0::'a::euclidean\text{-}space)\ 1) > 0$
by (*metis borel-closed closed-cball ennreal-0 has-integral-iff-emeasure-lborel has-integral-measure-lborel less-irrefl order-refl zero-less-measure-iff*)
qed

1.2.5 infdist and closest point projection

The distance to a union of two sets is the minimum of the distance to the two sets.

lemma *infdist-union-min* [*mono-intros*]:
assumes $A \neq \{\}$ $B \neq \{\}$
shows $infdist\ x\ (A \cup B) = min\ (infdist\ x\ A)\ (infdist\ x\ B)$
using *assms* **by** (*simp add*: *infdist-def cINF-union inf-real-def*)

The distance to a set is non-increasing with the set.

lemma *infdist-mono* [*mono-intros*]:
assumes $A \subseteq B$ $A \neq \{\}$
shows $infdist\ x\ B \leq infdist\ x\ A$
by (*simp add*: *assms infdist-eq-setdist setdist-subset-right*)

If a set is proper, then the infimum of the distances to this set is attained.

lemma *infdist-proper-attained*:
assumes *proper* C $C \neq \{\}$
shows $\exists c \in C.\ infdist\ x\ C = dist\ x\ c$
proof −
obtain a **where** $a \in C$ **using** *assms* **by** *auto*
have ∗: $dist\ x\ a \leq dist\ x\ z$ **if** $dist\ a\ z \geq 2 * dist\ a\ x$ **for** z
proof −
have $2 * dist\ a\ x \leq dist\ a\ z$ **using** *that* **by** *simp*
also have ... $\leq dist\ a\ x + dist\ x\ z$ **by** (*intro mono-intros*)
finally show *?thesis* **by** (*simp add*: *dist-commute*)
qed
have $\exists c \in C.\ \forall d \in C.\ dist\ x\ c \leq dist\ x\ d$
apply (*rule continuous-attains-inf-proper*[*OF assms*(*1*) ‹$a \in C$›, *of* - $2 * dist\ a\ x$])
using ∗ **by** (*auto intro*: *continuous-intros*)
then show *?thesis* **unfolding** *infdist-def* **using** ‹$C \neq \{\}$›
by (*metis antisym bdd-below-image-dist cINF-lower le-cINF-iff*)
qed

lemma *infdist-almost-attained*:

assumes *infdist x X* $<$ *a X* $\neq$ $\{\}$
shows $\exists\, y{\in}X.$ *dist x y* $<$ *a*
using *assms* **using** *cInf-less-iff*[*of* (*dist x*)'*X*] **unfolding** *infdist-def* **by** *auto*

lemma *infdist-triangle-abs* [*mono-intros*]:
$|$*infdist x A* $-$ *infdist y A*$|$ $\leq$ *dist x y*
by (*metis* (*full-types*) *abs-diff-le-iff diff-le-eq dist-commute infdist-triangle*)

The next lemma is missing in the library, contrary to its cousin `continuous_infdist`.

The infimum of the distance to a singleton set is simply the distance to the unique member of the set.

The closest point projection of x on A. It is not unique, so we choose one point realizing the minimal distance. And if there is no such point, then we use x, to make some statements true without any assumption.

definition *proj-set*::'*a*::*metric-space* $\Rightarrow$ '*a set* $\Rightarrow$ '*a set*
where *proj-set x A* $=$ $\{$*y* $\in$ *A*. *dist x y* $=$ *infdist x A*$\}$

definition *distproj*::'*a*::*metric-space* $\Rightarrow$ '*a set* $\Rightarrow$ '*a*
where *distproj x A* $=$ (*if proj-set x A* $\neq$ $\{\}$ *then SOME y. y* $\in$ *proj-set x A else x*)

lemma *proj-setD*:
assumes *y* $\in$ *proj-set x A*
shows *y* $\in$ *A dist x y* $=$ *infdist x A*
using *assms* **unfolding** *proj-set-def* **by** *auto*

lemma *proj-setI*:
assumes *y* $\in$ *A dist x y* $\leq$ *infdist x A*
shows *y* $\in$ *proj-set x A*
using *assms infdist-le*[*OF* ‹*y* $\in$ *A*›, *of x*] **unfolding** *proj-set-def* **by** *auto*

lemma *proj-setI'*:
assumes *y* $\in$ *A* $\bigwedge$*z. z* $\in$ *A* $\Longrightarrow$ *dist x y* $\leq$ *dist x z*
shows *y* $\in$ *proj-set x A*
proof (*rule proj-setI*[*OF* ‹*y* $\in$ *A*›])
show *dist x y* $\leq$ *infdist x A*
apply (*subst infdist-notempty*)
using *assms* **by** (*auto intro*!: *cInf-greatest*)
qed

lemma *distproj-in-proj-set*:
assumes *proj-set x A* $\neq$ $\{\}$
shows *distproj x A* $\in$ *proj-set x A*
distproj x A $\in$ *A*
dist x (*distproj x A*) $=$ *infdist x A*
proof $-$
show *distproj x A* $\in$ *proj-set x A*

```
    using assms unfolding distproj-def using some-in-eq by auto
  then show distproj x A ∈ A dist x (distproj x A) = infdist x A
    unfolding proj-set-def by auto
qed

lemma proj-set-nonempty-of-proper:
  assumes proper A A ≠ {}
  shows proj-set x A ≠ {}
proof −
  have ∃ y. y ∈ A ∧ dist x y = infdist x A
    using infdist-proper-attained[OF assms, of x] by auto
  then show proj-set x A ≠ {} unfolding proj-set-def by auto
qed

lemma distproj-self [simp]:
  assumes x ∈ A
  shows proj-set x A = {x}
        distproj x A = x
proof −
  show proj-set x A = {x}
    unfolding proj-set-def using assms by auto
  then show distproj x A = x
    unfolding distproj-def by auto
qed

lemma distproj-closure [simp]:
  assumes x ∈ closure A
  shows distproj x A = x
proof (cases proj-set x A ≠ {})
  case True
  show ?thesis
    using distproj-in-proj-set(3)[OF True] assms
    by (metis closure-empty dist-eq-0-iff distproj-self(2) in-closure-iff-infdist-zero)
next
  case False
  then show ?thesis unfolding distproj-def by auto
qed

lemma distproj-le:
  assumes y ∈ A
  shows dist x (distproj x A) ≤ dist x y
proof (cases proj-set x A ≠ {})
  case True
  show ?thesis using distproj-in-proj-set(3)[OF True] infdist-le[OF assms] by auto
next
  case False
  then show ?thesis unfolding distproj-def by auto
qed
```

lemma *proj-set-dist-le*:
  assumes $y \in A$ $p \in$ *proj-set* x A
  shows *dist* x $p \leq$ *dist* x y
  using *assms infdist-le* **unfolding** *proj-set-def* **by** *auto*

1.3 Material on ereal and ennreal

We add the simp rules that we needed to make all computations become more or less automatic.

lemma *ereal-of-real-of-ereal-iff* [*simp*]:
  ereal(*real-of-ereal* x) $= x \longleftrightarrow x \neq \infty \wedge x \neq -\infty$
  $x =$ *ereal*(*real-of-ereal* x) $\longleftrightarrow x \neq \infty \wedge x \neq -\infty$
by (*metis MInfty-neq-ereal*(*1*) *PInfty-neq-ereal*(*2*) *real-of-ereal.elims*)+

declare *ereal-inverse-eq-0* [*simp*]
declare *ereal-0-gt-inverse* [*simp*]
declare *ereal-inverse-le-0-iff* [*simp*]
declare *ereal-divide-eq-0-iff* [*simp*]
declare *ereal-mult-le-0-iff* [*simp*]
declare *ereal-zero-le-0-iff* [*simp*]
declare *ereal-mult-less-0-iff* [*simp*]
declare *ereal-zero-less-0-iff* [*simp*]
declare *ereal-uminus-eq-reorder* [*simp*]
declare *ereal-minus-le-iff* [*simp*]

lemma *ereal-inverse-noteq-minus-infinity* [*simp*]:
  $1/(x::ereal) \neq -\infty$
by (*simp add*: *divide-ereal-def*)

lemma *ereal-inverse-positive-iff-nonneg-not-infinity* [*simp*]:
  $0 < 1/(x::ereal) \longleftrightarrow (x \geq 0 \wedge x \neq \infty)$
by (*cases* x, *auto simp add*: *one-ereal-def*)

lemma *ereal-inverse-negative-iff-nonpos-not-infinity'* [*simp*]:
  $0 >$ *inverse* $(x::ereal) \longleftrightarrow (x < 0 \wedge x \neq -\infty)$
by (*cases* x, *auto simp add*: *one-ereal-def*)

lemma *ereal-divide-pos-iff* [*simp*]:
  $0 < x/(y::ereal) \longleftrightarrow (y \neq \infty \wedge y \neq -\infty) \wedge ((x > 0 \wedge y > 0) \vee (x < 0 \wedge y < 0) \vee (y = 0 \wedge x > 0))$
unfolding *divide-ereal-def* **by** *auto*

lemma *ereal-divide-neg-iff* [*simp*]:
  $0 > x/(y::ereal) \longleftrightarrow (y \neq \infty \wedge y \neq -\infty) \wedge ((x > 0 \wedge y < 0) \vee (x < 0 \wedge y > 0) \vee (y = 0 \wedge x < 0))$
unfolding *divide-ereal-def* **by** *auto*

More additions to `mono_intros`.

lemma *ereal-leq-imp-neg-leq* [*mono-intros*]:

fixes *x y*::*ereal*
assumes $x \leq y$
shows $-y \leq -x$
using *assms* **by** *auto*

lemma *ereal-le-imp-neg-le* [*mono-intros*]:
fixes *x y*::*ereal*
assumes $x < y$
shows $-y < -x$
using *assms* **by** *auto*

declare *ereal-mult-left-mono* [*mono-intros*]
declare *ereal-mult-right-mono* [*mono-intros*]
declare *ereal-mult-strict-right-mono* [*mono-intros*]
declare *ereal-mult-strict-left-mono* [*mono-intros*]

Monotonicity of basic inclusions.

lemma *ennreal-mono'*:
mono ennreal
by (*simp add*: *ennreal-leI monoI*)

lemma *enn2ereal-mono'*:
mono enn2ereal
by (*simp add*: *less-eq-ennreal.rep-eq mono-def*)

lemma *e2ennreal-mono'*:
mono e2ennreal
by (*simp add*: *e2ennreal-mono mono-def*)

lemma *enn2ereal-mono* [*mono-intros*]:
assumes $x \leq y$
shows *enn2ereal* $x \leq$ *enn2ereal* y
using *assms less-eq-ennreal.rep-eq* **by** *auto*

lemma *ereal-mono*:
mono ereal
unfolding *mono-def* **by** *auto*

lemma *ereal-strict-mono*:
strict-mono ereal
unfolding *strict-mono-def* **by** *auto*

lemma *ereal-mono2* [*mono-intros*]:
assumes $x \leq y$
shows *ereal* $x \leq$ *ereal* y
by (*simp add*: *assms*)

lemma *ereal-strict-mono2* [*mono-intros*]:
assumes $x < y$

shows *ereal x* < *ereal y*
using *assms* **by** *auto*

lemma *enn2ereal-a-minus-b-plus-b* [*mono-intros*]:
enn2ereal a ≤ *enn2ereal* (*a* − *b*) + *enn2ereal b*
by (*metis diff-add-self-ennreal less-eq-ennreal.rep-eq linear plus-ennreal.rep-eq*)

The next lemma follows from the same assertion in ereals.

lemma *enn2ereal-strict-mono* [*mono-intros*]:
assumes *x* < *y*
shows *enn2ereal x* < *enn2ereal y*
using *assms less-ennreal.rep-eq* **by** *auto*

declare *ennreal-mult-strict-left-mono* [*mono-intros*]
declare *ennreal-mult-strict-right-mono* [*mono-intros*]

lemma *ennreal-ge-0* [*mono-intros*]:
assumes *0* < *x*
shows *0* < *ennreal x*
by (*simp add*: *assms*)

The next lemma is true and useful in ereal. Note that variants such as $a + b \leq c + d$ implies $a - d \leq c - b$ are not true – take $a = c = \infty$ and $b = d = 0$...

lemma *ereal-minus-le-minus-plus* [*mono-intros*]:
fixes *a b c*::*ereal*
assumes *a* ≤ *b* + *c*
shows −*b* ≤ −*a* + *c*
using *assms* **apply** (*cases a*, *cases b*, *cases c*, *auto*)
using *ereal-infty-less-eq2*(*2*) *ereal-plus-1*(*4*) **by** *fastforce*

lemma *tendsto-ennreal-0* [*tendsto-intros*]:
assumes (*u* ⟶ *0*) *F*
shows ((λ*n*. *ennreal*(*u n*)) ⟶ *0*) *F*
unfolding *ennreal-0*[*symmetric*] **by** (*intro tendsto-intros assms*)

lemma *tendsto-ennreal-1* [*tendsto-intros*]:
assumes (*u* ⟶ *1*) *F*
shows ((λ*n*. *ennreal*(*u n*)) ⟶ *1*) *F*
unfolding *ennreal-1*[*symmetric*] **by** (*intro tendsto-intros assms*)

1.4 Miscellaneous

lemma *lim-ceiling-over-n* [*tendsto-intros*]:
assumes (λ*n*. *u n*/*n*) ⟶ *l*
shows (λ*n*. *ceiling*(*u n*)/*n*) ⟶ *l*
proof (*rule tendsto-sandwich*[*of* λ*n*. *u n*/*n* - - λ*n*. *u n*/*n* + *1*/*n*])
show $\forall_F$ *n in sequentially*. *u n* / *real n* ≤ *real-of-int* ⌈*u n*⌉ / *real n*

unfolding *eventually-sequentially* **by** (*rule exI*[*of* - *1*], *auto simp add*: *divide-simps*)
show $\forall_F$ *n in sequentially*. *real-of-int* ⌈*u n*⌉ / *real n* ≤ *u n* / *real n* + *1* / *real n*
unfolding *eventually-sequentially* **by** (*rule exI*[*of* - *1*], *auto simp add*: *divide-simps*)
have (λ*n*. *u n* / *real n* + *1* / *real n*) ⟶ *l* + *0*
by (*intro tendsto-intros assms*)
then show (λ*n*. *u n* / *real n* + *1* / *real n*) ⟶ *l* **by** *auto*
qed (*simp add*: *assms*)

1.4.1 Liminfs and Limsups

More facts on liminfs and limsups

lemma *Limsup-obtain′*:
fixes *u*::′*a* ⇒ ′*b*::*complete-linorder*
assumes *Limsup F u* > *c eventually P F*
shows ∃ *n*. *P n* ∧ *u n* > *c*
proof −
have ∗: (*INF P*∈{*P*. *eventually P F*}. *SUP x*∈{*x*. *P x*}. *u x*) > *c* **using** *assms* **by** (*simp add*: *Limsup-def*)
have ∗∗: *c* < (*SUP x*∈{*x*. *P x*}. *u x*) **using** *less-INF-D*[*OF* ∗, *of P*] *assms* **by** *auto*
then show *?thesis* **by** (*simp add*: *less-SUP-iff*)
qed

lemma *limsup-obtain*:
fixes *u*::*nat* ⇒ ′*a* :: *complete-linorder*
assumes *limsup u* > *c*
shows ∃ *n* ≥ *N*. *u n* > *c*
using *Limsup-obtain′*[*OF assms*, *of* λ*n*. *n* ≥ *N*] **unfolding** *eventually-sequentially* **by** *auto*

lemma *Liminf-obtain′*:
fixes *u*::′*a* ⇒ ′*b*::*complete-linorder*
assumes *Liminf F u* < *c eventually P F*
shows ∃ *n*. *P n* ∧ *u n* < *c*
proof −
have ∗: (*SUP P*∈{*P*. *eventually P F*}. *INF x*∈{*x*. *P x*}. *u x*) < *c* **using** *assms* **by** (*simp add*: *Liminf-def*)
have ∗∗: (*INF x*∈{*x*. *P x*}. *u x*) < *c* **using** *SUP-lessD*[*OF* ∗, *of P*] *assms* **by** *auto*
then show *?thesis* **by** (*simp add*: *INF-less-iff*)
qed

lemma *liminf-obtain*:
fixes *u*::*nat* ⇒ ′*a* :: *complete-linorder*
assumes *liminf u* < *c*
shows ∃ *n* ≥ *N*. *u n* < *c*
using *Liminf-obtain′*[*OF assms*, *of* λ*n*. *n* ≥ *N*] **unfolding** *eventually-sequentially*

by *auto*

The Liminf of a minimum is the minimum of the Liminfs.

lemma *Liminf-min-eq-min-Liminf*:
  fixes $u\ v$::$nat \Rightarrow 'a$::*complete-linorder*
  shows *Liminf F* $(\lambda n.\ min\ (u\ n)\ (v\ n)) = min\ (Liminf\ F\ u)\ (Liminf\ F\ v)$
proof (*rule order-antisym*)
  show *Liminf F* $(\lambda n.\ min\ (u\ n)\ (v\ n)) \leq min\ (Liminf\ F\ u)\ (Liminf\ F\ v)$
    by (*auto simp add*: *Liminf-mono*)

  have *Liminf F* $(\lambda n.\ min\ (u\ n)\ (v\ n)) > w$ **if** H: $min\ (Liminf\ F\ u)\ (Liminf\ F\ v)$
$> w$ **for** w
  proof (*cases* $\{w<..<min\ (Liminf\ F\ u)\ (Liminf\ F\ v)\} = \{\}$)
    case *True*
    have *eventually* $(\lambda n.\ u\ n > w)\ F$ *eventually* $(\lambda n.\ v\ n > w)\ F$
      using *H le-Liminf-iff* **by** *fastforce+*
    then have *eventually* $(\lambda n.\ min\ (u\ n)\ (v\ n) > w)\ F$
      apply *auto* **using** *eventually-elim2* **by** *fastforce*
    moreover have $z > w \Longrightarrow z \geq min\ (Liminf\ F\ u)\ (Liminf\ F\ v)$ **for** z
      using *H True not-le-imp-less* **by** *fastforce*
    ultimately have *eventually* $(\lambda n.\ min\ (u\ n)\ (v\ n) \geq min\ (Liminf\ F\ u)\ (Liminf$
$F\ v))\ F$
      by (*simp add*: *eventually-mono*)
    then have $min\ (Liminf\ F\ u)\ (Liminf\ F\ v) \leq Liminf\ F\ (\lambda n.\ min\ (u\ n)\ (v\ n))$
      by (*metis Liminf-bounded*)
    then show *?thesis* **using** *H less-le-trans* **by** *blast*
  next
    case *False*
    then obtain z **where** $z \in \{w<..<min\ (Liminf\ F\ u)\ (Liminf\ F\ v)\}$
      by *blast*
    then have H: $w < z\ z < min\ (Liminf\ F\ u)\ (Liminf\ F\ v)$
      by *auto*
    then have *eventually* $(\lambda n.\ u\ n > z)\ F$ *eventually* $(\lambda n.\ v\ n > z)\ F$
      using *le-Liminf-iff* **by** *fastforce+*
    then have *eventually* $(\lambda n.\ min\ (u\ n)\ (v\ n) > z)\ F$
      apply *auto* **using** *eventually-elim2* **by** *fastforce*
    then have *Liminf F* $(\lambda n.\ min\ (u\ n)\ (v\ n)) \geq z$
      by (*simp add*: *Liminf-bounded eventually-mono less-imp-le*)
    then show *?thesis* **using** $H(1)$
      by *auto*
  qed
  then show $min\ (Liminf\ F\ u)\ (Liminf\ F\ v) \leq Liminf\ F\ (\lambda n.\ min\ (u\ n)\ (v\ n))$
    using *not-le-imp-less* **by** *blast*
qed

The Limsup of a maximum is the maximum of the Limsups.

lemma *Limsup-max-eq-max-Limsup*:
  fixes u::$'a \Rightarrow 'b$::*complete-linorder*
  shows *Limsup F* $(\lambda n.\ max\ (u\ n)\ (v\ n)) = max\ (Limsup\ F\ u)\ (Limsup\ F\ v)$

proof (*rule order-antisym*)
  show *max* (*Limsup F u*) (*Limsup F v*) $\leq$ *Limsup F* ($\lambda n.$ *max* (*u n*) (*v n*))
    by (*auto intro*: *Limsup-mono*)

  have *Limsup F* ($\lambda n.$ *max* (*u n*) (*v n*)) $<$ *e* **if** *max* (*Limsup F u*) (*Limsup F v*) $<$ *e* **for** *e*
  proof (*cases* $\exists$ *t*. *max* (*Limsup F u*) (*Limsup F v*) $<$ *t* $\wedge$ *t* $<$ *e*)
    case *True*
    then obtain *t* **where** *t*: *t* $<$ *e* *max* (*Limsup F u*) (*Limsup F v*) $<$ *t* **by** *auto*
    then have *Limsup F u* $<$ *t* *Limsup F v* $<$ *t* **using** *that max-def* **by** *auto*
    then have $*$: *eventually* ($\lambda n.$ *u n* $<$ *t*) *F* *eventually* ($\lambda n.$ *v n* $<$ *t*) *F*
      by (*auto simp*: *Limsup-lessD*)
    have *eventually* ($\lambda n.$ *max* (*u n*) (*v n*) $<$ *t*) *F*
      using *eventually-mono*[*OF eventually-conj*[*OF* $*$]] **by** *auto*
    then have *Limsup F* ($\lambda n.$ *max* (*u n*) (*v n*)) $\leq$ *t*
      by (*meson Limsup-obtain′ not-le-imp-less order.asym*)
    then show *?thesis*
      using *t* **by** *auto*
  next
    case *False*
    have *Limsup F u* $<$ *e* *Limsup F v* $<$ *e* **using** *that max-def* **by** *auto*
    then have $*$: *eventually* ($\lambda n.$ *u n* $<$ *e*) *F* *eventually* ($\lambda n.$ *v n* $<$ *e*) *F*
      by (*auto simp*: *Limsup-lessD*)
    have *eventually* ($\lambda n.$ *max* (*u n*) (*v n*) $\leq$ *max* (*Limsup F u*) (*Limsup F v*)) *F*
    apply (*rule eventually-mono*[*OF eventually-conj*[*OF* $*$]]) **using** *False not-le-imp-less* **by** *force*
    then have *Limsup F* ($\lambda n.$ *max* (*u n*) (*v n*)) $\leq$ *max* (*Limsup F u*) (*Limsup F v*)
      by (*meson Limsup-obtain′ leD leI*)
    then show *?thesis* **using** *that le-less-trans* **by** *blast*
  qed
  then show *Limsup F* ($\lambda n.$ *max* (*u n*) (*v n*)) $\leq$ *max* (*Limsup F u*) (*Limsup F v*)
    using *not-le-imp-less* **by** *blast*
qed

1.4.2 Bounding the cardinality of a finite set

A variation with real bounds.

lemma *finite-finite-subset-caract′*:
  fixes *C*::*real*
  assumes $\bigwedge G.$ *G* $\subseteq$ *F* $\Longrightarrow$ *finite G* $\Longrightarrow$ *card G* $\leq$ *C*
  shows *finite F* $\wedge$ *card F* $\leq$ *C*
by (*meson assms finite-if-finite-subsets-card-bdd le-nat-floor order-refl*)

To show that a set has cardinality at most one, it suffices to show that any two of its elements coincide.

lemma *finite-at-most-singleton*:
  assumes $\bigwedge x\ y.$ *x* $\in$ *F* $\Longrightarrow$ *y* $\in$ *F* $\Longrightarrow$ *x* $=$ *y*

shows *finite* $F \wedge$ *card* $F \leq 1$
proof (*cases* $F = \{\}$)
case *True*
then show *?thesis* **by** *auto*
next
case *False*
then obtain x **where** $x \in F$ **by** *auto*
then have $F = \{x\}$ **using** *assms* **by** *auto*
then show *?thesis* **by** *auto*
qed

Bounded sets of integers are finite.

lemma *finite-real-int-interval* [*simp*]:
finite (*range real-of-int* $\cap$ $\{a..b\}$)
proof −
have *range real-of-int* $\cap$ $\{a..b\} \subseteq$ *real-of-int*‘{*floor a..ceiling b*}
by (*auto*, *metis atLeastAtMost-iff ceiling-mono ceiling-of-int floor-mono floor-of-int image-eqI*)
then show *?thesis* **using** *finite-subset* **by** *blast*
qed

Well separated sets of real numbers are finite, with controlled cardinality.

lemma *separated-in-real-card-bound*:
assumes $T \subseteq \{a..(b::real)\}$ $d > 0$ $\bigwedge x\ y.\ x \in T \Longrightarrow y \in T \Longrightarrow y > x \Longrightarrow y \geq x + d$
shows *finite* T *card* $T \leq$ *nat* (*floor* $((b-a)/d) + 1$)
proof −
define f **where** $f = (\lambda x.$ *floor* $((x-a) / d))$
have f‘$\{a..b\} \subseteq \{0..$*floor* $((b-a)/d)\}$
unfolding *f-def* **using** ‹$d > 0$› **by** (*auto simp add*: *floor-mono frac-le*)
then have ∗: f‘$T \subseteq \{0..$*floor* $((b-a)/d)\}$ **using** ‹$T \subseteq \{a..b\}$› **by** *auto*
then have *finite* (f‘T) **by** (*rule finite-subset*, *auto*)
have *card* (f‘T) $\leq$ *card* $\{0..$*floor* $((b-a)/d)\}$ **apply** (*rule card-mono*) **using** ∗ **by** *auto*
then have *card-le*: *card* (f‘T) $\leq$ *nat* (*floor* $((b-a)/d) + 1$) **using** *card-atLeastAtMost-int* **by** *auto*

have ∗: $f\ x \neq f\ y$ **if** $y \geq x + d$ **for** $x\ y$
proof −
have $(y-a)/d \geq (x-a)/d + 1$ **using** ‹$d > 0$› *that* **by** (*auto simp add*: *divide-simps*)
then show *?thesis* **unfolding** *f-def* **by** *linarith*
qed
have *inj-on* $f\ T$
unfolding *inj-on-def* **using** ∗ *assms*(*3*) **by** (*auto*, *metis not-less-iff-gr-or-eq*)
show *finite* T
using ‹*finite* (f‘T)› ‹*inj-on* $f\ T$› *finite-image-iff* **by** *blast*
have *card* $T =$ *card* (f‘T)
using ‹*inj-on* $f\ T$› **by** (*simp add*: *card-image*)

then show $card\ T \leq nat\ (floor\ ((b-a)/d) + 1)$
  using *card-le* **by** *auto*
qed

1.5 Manipulating finite ordered sets

We will need below to construct finite sets of real numbers with good properties expressed in terms of consecutive elements of the set. We introduce tools to manipulate such sets, expressing in particular the next and the previous element of the set and controlling how they evolve when one inserts a new element in the set. It works in fact in any linorder, and could also prove useful to construct sets of integer numbers.

Manipulating the next and previous elements work well, except at the top (respectively bottom). In our constructions, these will be fixed and called b and a.

Notations for the next and the previous elements.

definition $next\text{-}in::'a\ set \Rightarrow 'a \Rightarrow ('a::linorder)$
  where $next\text{-}in\ A\ u = Min\ (A \cap \{u<..\})$

definition $prev\text{-}in::'a\ set \Rightarrow 'a \Rightarrow ('a::linorder)$
  where $prev\text{-}in\ A\ u = Max\ (A \cap \{..<u\})$

context
  fixes $A::('a::linorder)\ set$ **and** $a\ b::'a$
  assumes A: $finite\ A\ A \subseteq \{a..b\}\ a \in A\ b \in A\ a < b$
begin

Basic properties of the next element, when one starts from an element different from top.

lemma *next-in-basics*:
  assumes $u \in \{a..<b\}$
  shows $next\text{-}in\ A\ u \in A$
    $next\text{-}in\ A\ u > u$
    $A \cap \{u<..<next\text{-}in\ A\ u\} = \{\}$
proof $-$
  have *next-in-A*: $next\text{-}in\ A\ u \in A \cap \{u<..\}$
    unfolding *next-in-def* **apply** (*rule Min-in*)
    using *assms* ‹*finite A*› ‹$b \in A$› **by** *auto*
  then show $next\text{-}in\ A\ u \in A\ next\text{-}in\ A\ u > u$ **by** *auto*
  show $A \cap \{u<..<next\text{-}in\ A\ u\} = \{\}$
    unfolding *next-in-def* **using** A **by** (*auto simp add*: *leD*)
qed

lemma *next-inI*:
  assumes $u \in \{a..<b\}$
    $v \in A$

$v > u$
$\{u<..<v\} \cap A = \{\}$
shows *next-in* $A\ u = v$
using *assms next-in-basics*[*OF* ‹$u \in \{a..<b\}$›] **by** *fastforce*

Basic properties of the previous element, when one starts from an element different from bottom.

lemma *prev-in-basics*:
assumes $u \in \{a<..b\}$
shows *prev-in* $A\ u \in A$
prev-in $A\ u < u$
$A \cap \{$*prev-in* $A\ u<..<u\} = \{\}$
proof −
have *prev-in-A*: *prev-in* $A\ u \in A \cap \{..<u\}$
unfolding *prev-in-def* **apply** (*rule Max-in*)
using *assms* ‹*finite* A› ‹$a \in A$› **by** *auto*
then show *prev-in* $A\ u \in A$ *prev-in* $A\ u < u$ **by** *auto*
show $A \cap \{$*prev-in* $A\ u<..<u\} = \{\}$
unfolding *prev-in-def* **using** A **by** (*auto simp add*: *leD*)
qed

lemma *prev-inI*:
assumes $u \in \{a<..b\}$
$v \in A$
$v < u$
$\{v<..<u\} \cap A = \{\}$
shows *prev-in* $A\ u = v$
using *assms prev-in-basics*[*OF* ‹$u \in \{a<..b\}$›]
by (*meson disjoint-iff-not-equal greaterThanLessThan-iff less-linear*)

The interval $[a, b]$ is covered by the intervals between the consecutive elements of A.

lemma *intervals-decomposition*:
$(\bigcup\ U \in \{\{u..$*next-in* $A\ u\} \mid u.\ u \in A - \{b\}\}.\ U) = \{a..b\}$
proof
show $(\bigcup U \in \{\{u..$*next-in* $A\ u\} \mid u.\ u \in A - \{b\}\}.\ U) \subseteq \{a..b\}$
using ‹$A \subseteq \{a..b\}$› *next-in-basics*(*1*) **apply** *auto* **apply** *fastforce*
by (*metis* ‹$A \subseteq \{a..b\}$› *atLeastAtMost-iff eq-iff le-less-trans less-eq-real-def not-less subset-eq subset-iff-psubset-eq*)

have $x \in (\bigcup U \in \{\{u..$*next-in* $A\ u\} \mid u.\ u \in A - \{b\}\}.\ U)$ **if** $x \in \{a..b\}$ **for** x
proof −
consider $x = b \mid x \in A - \{b\} \mid x \notin A$ **by** *blast*
then show *?thesis*
proof(*cases*)
case *1*
define u **where** $u =$ *prev-in* $A\ b$
have $b \in \{a<..b\}$ **using** ‹$a < b$› **by** *simp*

have *u* ∈ *A* − {*b*} **unfolding** *u-def* **using** *prev-in-basics*[*OF* ‹*b* ∈ {*a*<..*b*}›] **by** *simp*
then have *u* ∈ {*a*..<*b*} **using** ‹*A* ⊆ {*a*..*b*}› ‹*a* < *b*› **by** *fastforce*
have *next-in A u* = *b*
using *prev-in-basics*[*OF* ‹*b* ∈ {*a*<..*b*}›] *next-in-basics*[*OF* ‹*u* ∈ {*a*..<*b*}›] ‹*A* ⊆ {*a*..*b*}› **unfolding** *u-def* **by** *force*
then have *x* ∈ {*u*..*next-in A u*} **unfolding** *1* **using** *prev-in-basics*[*OF* ‹*b* ∈ {*a*<..*b*}›] *u-def* **by** *auto*
then show *?thesis* **using** ‹*u* ∈ *A* − {*b*}› **by** *auto*
next
case *2*
then have *x* ∈ {*a*..<*b*} **using** ‹*A* ⊆ {*a*..*b*}› ‹*a* < *b*› **by** *fastforce*
have *x* ∈ {*x*.. *next-in A x*} **using** *next-in-basics*[*OF* ‹*x* ∈ {*a*..<*b*}›] **by** *auto*
then show *?thesis* **using** *2* **by** *auto*
next
case *3*
then have *x* ∈ {*a*<..*b*} **using** *that* ‹*a* ∈ *A*› *leI* **by** *fastforce*
define *u* **where** *u* = *prev-in A x*
have *u* ∈ *A* − {*b*} **unfolding** *u-def* **using** *prev-in-basics*[*OF* ‹*x* ∈ {*a*<..*b*}›] *that* **by** *auto*
then have *u* ∈ {*a*..<*b*} **using** ‹*A* ⊆ {*a*..*b*}› ‹*a* < *b*› **by** *fastforce*
have *x* ∈ {*u*..*next-in A u*}
using *prev-in-basics*[*OF* ‹*x* ∈ {*a*<..*b*}›] *next-in-basics*[*OF* ‹*u* ∈ {*a*..<*b*}›] **unfolding** *u-def* **by** *auto*
then show *?thesis* **using** ‹*u* ∈ *A* − {*b*}› **by** *auto*
qed
qed
then show {*a*..*b*} ⊆ (⋃ *U*∈{{*u*..*next-in A u*} |*u*. *u* ∈ *A* − {*b*}}. *U*) **by** *auto*
qed
end

If one inserts an additional element, then next and previous elements are not modified, except at the location of the insertion.

lemma *next-in-insert*:
assumes *A*: *finite A A* ⊆ {*a*..*b*} *a* ∈ *A* *b* ∈ *A* *a* < *b*
and *x* ∈ {*a*..*b*} − *A*
shows ⋀*u*. *u* ∈ *A* − {*b*, *prev-in A x*} ⟹ *next-in* (*insert x A*) *u* = *next-in A u*
next-in (*insert x A*) *x* = *next-in A x*
next-in (*insert x A*) (*prev-in A x*) = *x*
proof −
define *B* **where** *B* = *insert x A*
have *B*: *finite B B* ⊆ {*a*..*b*} *a* ∈ *B* *b* ∈ *B* *a* < *b*
using *assms* **unfolding** *B-def* **by** *auto*
have *x*: *x* ∈ {*a*..<*b*} *x* ∈ {*a*<..*b*} **using** *assms leI* **by** *fastforce+*
show *next-in B x* = *next-in A x*
unfolding *B-def* **by** (*auto simp add*: *next-in-def*)

show *next-in B* (*prev-in A x*) = *x*
apply (*rule next-inI*[*OF B*])

unfolding *B-def* **using** *prev-in-basics*[*OF A* ‹*x* ∈ {*a*<..*b*}›] ‹*A* ⊆ {*a*..*b*}› *x* **by** *auto*

fix *u* **assume** *u* ∈ *A* − {*b*, *prev-in A x*}
then have *u* ∈ {*a*..<*b*} **using** *assms* **by** *fastforce*
have *x* ∉ {*u*<..<*next-in A u*}
proof (*rule ccontr*)
assume ¬(*x* ∉ {*u*<..<*next-in A u*})
then have ∗: *x* ∈ {*u*<..<*next-in A u*} **by** *auto*
have *prev-in A x* = *u*
apply (*rule prev-inI*[*OF A* ‹*x* ∈ {*a*<..*b*}›])
using ‹*u* ∈ *A* − {*b*, *prev-in A x*}› ∗ *next-in-basics*[*OF A* ‹*u* ∈ {*a*..<*b*}›] **apply** *auto*
by (*meson disjoint-iff-not-equal greaterThanLessThan-iff less-trans*)
then show *False* **using** ‹*u* ∈ *A* − {*b*, *prev-in A x*}› **by** *auto*
qed
show *next-in B u* = *next-in A u*
apply (*rule next-inI*[*OF B* ‹*u* ∈ {*a*..<*b*}›]) **unfolding** *B-def*
using *next-in-basics*[*OF A* ‹*u* ∈ {*a*..<*b*}›] ‹*x* ∉ {*u*<..<*next-in A u*}› **by** *auto*
qed

If consecutive elements are enough separated, this implies a simple bound on the cardinality of the set.

lemma *separated-in-real-card-bound2*:
fixes *A*::*real set*
assumes *A*: *finite A A* ⊆ {*a*..*b*} *a* ∈ *A b* ∈ *A a* < *b*
and *B*: ⋀*u*. *u* ∈ *A* − {*b*} ⟹ *next-in A u* ≥ *u* + *d d* > *0*
shows *card A* ≤ *nat* (*floor* ((*b*−*a*)/*d*) + *1*)
proof (*rule separated-in-real-card-bound*[*OF* ‹*A* ⊆ {*a*..*b*}› ‹*d* > *0*›])
fix *x y* **assume** *x* ∈ *A y* ∈ *A y* > *x*
then have *x* ∈ *A* − {*b*} *x* ∈ {*a*..<*b*} **using** ‹*A* ⊆ {*a*..*b*}› **by** *auto*
have *y* ≥ *next-in A x*
using *next-in-basics*[*OF A* ‹*x* ∈ {*a*..<*b*}›] ‹*y* ∈ *A*› ‹*y* > *x*› **by** *auto*
then show *y* ≥ *x* + *d* **using** *B*(*1*)[*OF* ‹*x* ∈ *A* − {*b*}›] **by** *auto*
qed

1.6 Well-orders

In this subsection, we give additional lemmas on well-orders or cardinals or whatever, that would well belong to the library, and will be needed below.

lemma (**in** *wo-rel*) *max2-underS* [*simp*]:
assumes *x* ∈ *underS z y* ∈ *underS z*
shows *max2 x y* ∈ *underS z*
using *assms max2-def* **by** *auto*

lemma (**in** *wo-rel*) *max2-underS′* [*simp*]:
assumes *x* ∈ *underS y*
shows *max2 x y* = *y max2 y x* = *y*

apply (*simp add*: *underS-E assms max2-def*)
using *assms max2-def ANTISYM antisym-def underS-E* **by** *fastforce*

lemma (**in** *wo-rel*) *max2-xx* [*simp*]:
  max2 x x = *x*
using *max2-def* **by** *auto*

declare *underS-notIn* [*simp*]

The abbrevation $= o$ is used both in `Set_Algebras` and Cardinals. We disable the one from `Set_Algebras`.

no-notation *elt-set-eq* (**infix** ‹=*o*› *50*)

lemma *regularCard-ordIso*:
  assumes *Card-order r regularCard r s =o r*
  shows *regularCard s*
unfolding *regularCard-def*
proof (*auto*)
  fix *K* **assume** *K*: *K* ⊆ *Field s cofinal K s*
  obtain *f* **where** *f*: *bij-betw f* (*Field s*) (*Field r*) *embed s r f* **using** ‹*s =o r*› **unfolding** *ordIso-def iso-def* **by** *auto*
  have *f*‘*K* ⊆ *Field r* **using** *K*(*1*) *f*(*1*) *bij-betw-imp-surj-on* **by** *blast*
  have *cofinal* (*f*‘*K*) *r* **unfolding** *cofinal-def*
  proof
    fix *a* **assume** *a* ∈ *Field r*
    then obtain *a′* **where** *a*: *a′* ∈ *Field s f a′* = *a* **using** *f*
      by (*metis bij-betw-imp-surj-on imageE*)
    then obtain *b′* **where** *b*: *b′* ∈ *K a′* ≠ *b′* ∧ (*a′*, *b′*) ∈ *s*
      using ‹*cofinal K s*› **unfolding** *cofinal-def* **by** *auto*
    have *P1*: *f b′* ∈ *f*‘*K* **using** *b*(*1*) **by** *auto*
    have *a′* ≠ *b′ a′* ∈ *Field s b′* ∈ *Field s* **using** *a*(*1*) *b K*(*1*) **by** *auto*
    then have *P2*: *a* ≠ *f b′* **unfolding** *a*(*2*)[*symmetric*] **using** *f*(*1*) **unfolding** *bij-betw-def inj-on-def* **by** *auto*
    have (*a′*, *b′*) ∈ *s* **using** *b* **by** *auto*
    then have *P3*: (*a*, *f b′*) ∈ *r* **unfolding** *a*(*2*)[*symmetric*] **using** *f*
    by (*meson FieldI1 FieldI2 Card-order-ordIso*[*OF assms*(*1*) *assms*(*3*)] *card-order-on-def iso-defs*(*1*) *iso-iff2*)
    show ∃ *b*∈*f* ‘ *K*. *a* ≠ *b* ∧ (*a*, *b*) ∈ *r*
      using *P1 P2 P3* **by** *blast*
  qed
  then have |*f*‘*K*| =*o r*
    using ‹*regularCard r*› ‹*f*‘*K* ⊆ *Field r*› **unfolding** *regularCard-def* **by** *auto*
  moreover have |*f*‘*K*| =*o* |*K*| **using** *f*(*1*) *K*(*1*)
    by (*meson bij-betw-subset card-of-ordIsoI ordIso-symmetric*)
  ultimately show |*K*| =*o s*
    using ‹*s =o r*› **by** (*meson ordIso-symmetric ordIso-transitive*)
qed

lemma *AboveS-not-empty-in-regularCard*:

assumes $|S| <o\ r$ $S \subseteq$ *Field r*
assumes *r*: *Card-order r regularCard r* $\neg$*finite* (*Field r*)
shows *AboveS r S* $\neq$ {}
proof −
have $\neg$(*cofinal S r*)
using *assms not-ordLess-ordIso* **unfolding** *regularCard-def* **by** *auto*
then obtain *a* **where** *a*: $a \in$ *Field r* $\forall b \in S.\ \neg(a \neq b \wedge (a,b) \in r)$
unfolding *cofinal-def* **by** *auto*
have $*$: $a = b \vee (b,\ a) \in r$ **if** $b \in S$ **for** *b*
proof −
have $a = b \vee (a,b) \notin r$ **using** *a that* **by** *auto*
then show *?thesis*
using ‹*Card-order r*› ‹$a \in$ *Field r*› ‹$b \in S$› ‹$S \subseteq$ *Field r*› **unfolding** *card-order-on-def well-order-on-def linear-order-on-def total-on-def*
by *auto*
qed
obtain *c* **where** $c \in$ *Field r* $c \neq a$ $(a,\ c) \in r$
using *a*(*1*) *r infinite-Card-order-limit* **by** *fastforce*
then have $c \in$ *AboveS r S*
unfolding *AboveS-def* **apply** *simp* **using** *Card-order-trans*[*OF r*(*1*)] **by** (*metis* $*$)
then show *?thesis* **by** *auto*
qed

lemma *AboveS-not-empty-in-regularCard′*:
assumes $|S| <o\ r$ *f*‘$S \subseteq$ *Field r* $T \subseteq S$
assumes *r*: *Card-order r regularCard r* $\neg$*finite* (*Field r*)
shows *AboveS r* (*f*‘*T*) $\neq$ {}
proof −
have $|f`T| \leq o\ |T|$ **by** *simp*
moreover have $|T| \leq o\ |S|$ **using** ‹$T \subseteq S$› **by** *simp*
ultimately have $*$: $|f`T| <o\ r$ **using** ‹$|S| <o\ r$› **by** (*meson ordLeq-ordLess-trans*)
show *?thesis* **using** *AboveS-not-empty-in-regularCard*[*OF* $*$ - *r*] ‹$T \subseteq S$› ‹*f*‘$S \subseteq$ *Field r*› **by** *auto*
qed

lemma *Well-order-extend*:
assumes *WELL*: *well-order-on A r* **and** *SUB*: $A \subseteq B$
shows $\exists r'.$ *well-order-on B r′* $\wedge$ $r \subseteq r'$
proof−
have *r*: *Well-order r* $\wedge$ *Field r* $= A$ **using** *WELL well-order-on-Well-order* **by** *blast*
let *?C* $= B - A$
obtain *r′′* **where** *well-order-on ?C r′′* **using** *well-order-on* **by** *blast*
then have *r′′*: *Well-order r′′* $\wedge$ *Field r′′* $=$ *?C*
using *well-order-on-Well-order* **by** *blast*
let *?r′* $=$ *r Osum r′′*
have *Field r Int Field r′′* $=$ {} **using** *r r′′* **by** *auto*
then have $r \leq o$ *?r′* **using** *Osum-ordLeq*[*of r r′′*] *r r′′* **by** *blast*

then have *Well-order ?r′* **unfolding** *ordLeq-def* **by** *auto*
moreover have *Field ?r′ = B* **using** *r r″ SUB* **by** (*auto simp add*: *Field-Osum*)
ultimately have *well-order-on B ?r′* **by** *auto*
moreover have $r \subseteq$ *?r′* **by** (*simp add*: *Osum-def subrelI*)
ultimately show *?thesis* **by** *blast*
qed

The next lemma shows that, if the range of a function is endowed with a wellorder, then one can pull back this wellorder by the function, and then extend it in the fibers of the function in order to keep the wellorder property.

The proof is done by taking an arbitrary family of wellorders on each of the fibers, and using the lexicographic order: one has $x < y$ if $fx < fy$, or if $fx = fy$ and, in the corresponding fiber of f, one has $x < y$.

To formalize it, it is however more efficient to use one single wellorder, and restrict it to each fiber.

lemma *Well-order-pullback*:
assumes *Well-order r*
shows $\exists s.$ *Well-order s* $\wedge$ *Field s = UNIV* $\wedge$ $(\forall x\ y.\ (f\ x, f\ y) \in (r - Id) \longrightarrow (x, y) \in s)$
proof −
obtain *r2* **where** *r2*: *Well-order r2 Field r2 = UNIV* $r \subseteq r2$
using *Well-order-extend*[*OF assms*, *of UNIV*] *well-order-on-Well-order* **by** *auto*
obtain *s2* **where** *s2*: *Well-order s2 Field s2 =* (*UNIV*::′*b set*)
by (*meson well-ordering*)

have *r2s2*:
$\bigwedge x\ y\ z.\ (x, y) \in s2 \Longrightarrow (y, z) \in s2 \Longrightarrow (x, z) \in s2$
$\bigwedge x.\ (x, x) \in s2$
$\bigwedge x\ y.\ (x, y) \in s2 \vee (y, x) \in s2$
$\bigwedge x\ y.\ (x, y) \in s2 \Longrightarrow (y, x) \in s2 \Longrightarrow x = y$
$\bigwedge x\ y\ z.\ (x, y) \in r2 \Longrightarrow (y, z) \in r2 \Longrightarrow (x, z) \in r2$
$\bigwedge x.\ (x, x) \in r2$
$\bigwedge x\ y.\ (x, y) \in r2 \vee (y, x) \in r2$
$\bigwedge x\ y.\ (x, y) \in r2 \Longrightarrow (y, x) \in r2 \Longrightarrow x = y$
using *r2 s2* **unfolding** *well-order-on-def linear-order-on-def partial-order-on-def total-on-def preorder-on-def antisym-def refl-on-def trans-def*
by (*metis UNIV-I*)+

define *s* **where** $s = \{(x,y).\ (f\ x, f\ y) \in r2 \wedge (f\ x = f\ y \longrightarrow (x, y) \in s2)\}$
have *linear-order s*
unfolding *linear-order-on-def partial-order-on-def preorder-on-def*
proof (*auto*)
show *total-on UNIV s*
unfolding *s-def* **apply** (*rule total-onI*, *auto*) **using** *r2s2* **by** *metis*+
show *refl-on UNIV s*
unfolding *s-def* **apply** (*rule refl-onI*, *auto*) **using** *r2s2* **by** *blast*+
show *trans s*
unfolding *s-def* **apply** (*rule transI*, *auto*) **using** *r2s2* **by** *metis*+

show *antisym s*
unfolding *s-def* **apply** (*rule antisymI*, *auto*) **using** *r2s2* **by** *metis+*
qed
moreover have *wf* (*s* − *Id*)
proof (*rule wfI-min*)
fix $x::'b$ **and** *Q* **assume** $x \in Q$
obtain z' **where** z': $z' \in f`Q \bigwedge y. (y, z') \in r2 - Id \longrightarrow y \notin f`Q$
proof (*rule wfE-min*[*of r2−Id f x f‘Q*], *auto*)
show *wf*(*r2*−*Id*) **using** ‹*Well-order r2*› **unfolding** *well-order-on-def* **by** *auto*
show $f\ x \in f`Q$ **using** ‹$x \in Q$› **by** *auto*
qed
define *Q2* **where** $Q2 = Q \cap f -` \{z'\}$
obtain *z* **where** *z*: $z \in Q2 \bigwedge y. (y, z) \in s2 - Id \longrightarrow y \notin Q2$
proof (*rule wfE-min'*[*of s2−Id Q2*], *auto*)
show *wf*(*s2*−*Id*) **using** ‹*Well-order s2*› **unfolding** *well-order-on-def* **by** *auto*
assume $Q2 = \{\}$
then show *False* **unfolding** *Q2-def* **using** ‹$z' \in f`Q$› **by** *blast*
qed
have $(y, z) \in (s-Id) \Longrightarrow y \notin Q$ **for** *y*
unfolding *s-def* **using** z' *z Q2-def* **by** *auto*
then show $\exists z\in Q. \forall y. (y, z) \in s - Id \longrightarrow y \notin Q$
using ‹$z \in Q2$› *Q2-def* **by** *auto*
qed
ultimately have *well-order-on UNIV s* **unfolding** *well-order-on-def* **by** *simp*
moreover have $(f\ x, f\ y) \in (r-Id) \longrightarrow (x, y) \in s$ **for** *x y*
unfolding *s-def* **using** ‹$r \subseteq r2$› **by** *auto*
ultimately show *?thesis* **using** *well-order-on-Well-order* **by** *metis*
qed

end

2 The exponential on extended real numbers.

theory *Eexp-Eln*
imports *Library-Complements*
begin

To define the distance on the Gromov completion of hyperbolic spaces, we need to use the exponential on extended real numbers. We can not use the symbol `exp`, as this symbol is already used in Banach algebras, so we use `ennexp` instead. We prove its basic properties (together with properties of the logarithm) here. We also use it to define the square root on ennreal. Finally, we also define versions from ereal to ereal.

function *ennexp*::*ereal* ⇒ *ennreal* **where**
ennexp (*ereal r*) = *ennreal* (*exp r*)
| *ennexp* (∞) = ∞
| *ennexp* (−∞) = *0*
by (*auto intro*: *ereal-cases*)

termination by *standard* (*rule wf-empty*)

lemma *ennexp-0* [*simp*]:
  ennexp 0 = 1
by (*auto simp add*: *zero-ereal-def one-ennreal-def*)

function *eln*::*ennreal* ⇒ *ereal* **where**
eln (*ennreal r*) = (*if r* ≤ *0 then* −∞ *else ereal* (*ln r*))
| *eln* (∞) = ∞
by (*auto intro*: *ennreal-cases*, *metis ennreal-eq-0-iff*, *simp add*: *ennreal-neg*)
termination by *standard* (*rule wf-empty*)

lemma *eln-simps* [*simp*]:
  eln 0 = −∞
  eln 1 = *0*
  eln top = ∞
apply (*simp only*: *eln.simps ennreal-0*[*symmetric*], *simp*)
apply (*simp only*: *eln.simps ennreal-1*[*symmetric*], *simp*)
using *eln.simps*(*2*) **by** *auto*

lemma *eln-real-pos*:
  assumes *r* > *0*
  shows *eln* (*ennreal r*) = *ereal* (*ln r*)
using *eln.simps assms* **by** *auto*

lemma *eln-ennexp* [*simp*]:
  eln (*ennexp x*) = *x*
apply (*cases x*) **using** *eln.simps* **by** *auto*

lemma *ennexp-eln* [*simp*]:
  ennexp (*eln x*) = *x*
apply (*cases x*) **using** *eln.simps* **by** *auto*

lemma *ennexp-strict-mono*:
  strict-mono ennexp
proof −
  have *ennexp x* < *ennexp y* **if** *x* < *y* **for** *x y*
    apply (*cases x*, *cases y*)
    using *that* **apply** (*auto simp add*: *ennreal-less-iff*)
    by (*cases y*, *auto*)
  then show *?thesis* **unfolding** *strict-mono-def* **by** *auto*
qed

lemma *ennexp-mono*:
  mono ennexp
using *ennexp-strict-mono* **by** (*simp add*: *strict-mono-mono*)

lemma *ennexp-strict-mono2* [*mono-intros*]:
  assumes *x* < *y*

shows *ennexp* x < *ennexp* y
using *ennexp-strict-mono assms* **unfolding** *strict-mono-def* **by** *auto*

lemma *ennexp-mono2* [*mono-intros*]:
assumes $x \leq y$
shows *ennexp* $x \leq$ *ennexp* y
using *ennexp-mono assms* **unfolding** *mono-def* **by** *auto*

lemma *ennexp-le1* [*simp*]:
ennexp $x \leq 1 \longleftrightarrow x \leq 0$
by (*metis ennexp-0 ennexp-mono2 ennexp-strict-mono eq-iff le-cases strict-mono-eq*)

lemma *ennexp-ge1* [*simp*]:
ennexp $x \geq 1 \longleftrightarrow x \geq 0$
by (*metis ennexp-0 ennexp-mono2 ennexp-strict-mono eq-iff le-cases strict-mono-eq*)

lemma *eln-strict-mono*:
strict-mono eln
by (*metis ennexp-eln strict-monoI ennexp-strict-mono strict-mono-less*)

lemma *eln-mono*:
mono eln
using *eln-strict-mono* **by** (*simp add*: *strict-mono-mono*)

lemma *eln-strict-mono2* [*mono-intros*]:
assumes $x < y$
shows *eln* x < *eln* y
using *eln-strict-mono assms* **unfolding** *strict-mono-def* **by** *auto*

lemma *eln-mono2* [*mono-intros*]:
assumes $x \leq y$
shows *eln* $x \leq$ *eln* y
using *eln-mono assms* **unfolding** *mono-def* **by** *auto*

lemma *eln-le0* [*simp*]:
eln $x \leq 0 \longleftrightarrow x \leq 1$
by (*metis ennexp-eln ennexp-le1*)

lemma *eln-ge0* [*simp*]:
eln $x \geq 0 \longleftrightarrow x \geq 1$
by (*metis ennexp-eln ennexp-ge1*)

lemma *bij-ennexp*:
bij ennexp
by (*auto intro*!: *bij-betw-byWitness*[*of - eln*])

lemma *bij-eln*:
bij eln
by (*auto intro*!: *bij-betw-byWitness*[*of - ennexp*])

lemma *ennexp-continuous*:
continuous-on UNIV ennexp
apply (*rule continuous-onI-mono*)
using *ennexp-mono* **unfolding** *mono-def* **by** (*auto simp add*: *bij-ennexp bij-is-surj*)

lemma *ennexp-tendsto* [*tendsto-intros*]:
assumes $((\lambda n.\ u\ n) \longrightarrow l)\ F$
shows $((\lambda n.\ ennexp(u\ n)) \longrightarrow ennexp\ l)\ F$
using *ennexp-continuous assms* **by** (*metis UNIV-I continuous-on tendsto-compose*)

lemma *eln-continuous*:
continuous-on UNIV eln
apply (*rule continuous-onI-mono*)
using *eln-mono* **unfolding** *mono-def* **by** (*auto simp add*: *bij-eln bij-is-surj*)

lemma *eln-tendsto* [*tendsto-intros*]:
assumes $((\lambda n.\ u\ n) \longrightarrow l)\ F$
shows $((\lambda n.\ eln(u\ n)) \longrightarrow eln\ l)\ F$
using *eln-continuous assms* **by** (*metis UNIV-I continuous-on tendsto-compose*)

lemma *ennexp-special-values* [*simp*]:
$ennexp\ x = 0 \longleftrightarrow x = -\infty$
$ennexp\ x = 1 \longleftrightarrow x = 0$
$ennexp\ x = \infty \longleftrightarrow x = \infty$
$ennexp\ x = top \longleftrightarrow x = \infty$
by *auto* (*metis eln-ennexp eln-simps*)+

lemma *eln-special-values* [*simp*]:
$eln\ x = -\infty \longleftrightarrow x = 0$
$eln\ x = 0 \longleftrightarrow x = 1$
$eln\ x = \infty \longleftrightarrow x = \infty$
apply *auto*
apply (*metis ennexp.simps ennexp-eln ennexp-0*)+
by (*metis ennexp.simps*(*2*) *ennexp-eln infinity-ennreal-def*)

lemma *ennexp-add-mult*:
assumes $\neg((a = \infty \wedge b = -\infty) \vee (a = -\infty \wedge b = \infty))$
shows $ennexp(a+b) = ennexp\ a * ennexp\ b$
apply (*cases a*, *cases b*)
using *assms* **by** (*auto simp add*: *ennreal-mult''* *exp-add ennreal-top-eq-mult-iff*)

lemma *eln-mult-add*:
assumes $\neg((a = \infty \wedge b = 0) \vee (a = 0 \wedge b = \infty))$
shows $eln(a * b) = eln\ a + eln\ b$
by (*smt assms ennexp.simps*(*2*) *ennexp.simps*(*3*) *ennexp-add-mult ennexp-eln eln-ennexp*)

We can also define the square root on ennreal using the above exponential.

definition *ennsqrt*::*ennreal* ⇒ *ennreal*

where *ennsqrt x* = *ennexp*(*eln x*/*2*)

lemma *ennsqrt-square* [*simp*]:
(*ennsqrt x*) ∗ (*ennsqrt x*) = *x*
proof −
have $y/2 + y/2 = y$ **for** *y*::*ereal*
by (*cases y*, *auto*)
then show *?thesis*
unfolding *ennsqrt-def* **by** (*subst ennexp-add-mult*[*symmetric*], *auto*)
qed

lemma *ennsqrt-simps* [*simp*]:
ennsqrt 0 = *0*
ennsqrt 1 = *1*
ennsqrt $\infty = \infty$
ennsqrt top = *top*
unfolding *ennsqrt-def* **by** *auto*

lemma *ennsqrt-mult*:
ennsqrt(*a* ∗ *b*) = *ennsqrt a* ∗ *ennsqrt b*
proof −
have [*simp*]: *z*/*ereal 2* = $-\infty \longleftrightarrow$ *z* = $-\infty$ **for** *z*
by (*auto simp add*: *ereal-divide-eq*)

consider *a* = *0* | *b* = *0* | *a* > *0* ∧ *b* > *0*
using *zero-less-iff-neq-zero* **by** *auto*
then show *?thesis*
apply (*cases*, *auto*)
apply (*cases a*, *cases b*, *auto simp add*: *ennreal-mult-top ennreal-top-mult*)
unfolding *ennsqrt-def* **apply** (*subst ennexp-add-mult*[*symmetric*], *auto*)
apply (*subst eln-mult-add*, *auto*)
done
qed

lemma *ennsqrt-square2* [*simp*]:
ennsqrt (*x* ∗ *x*) = *x*
unfolding *ennsqrt-mult* **by** *auto*

lemma *ennsqrt-eq-iff-square*:
ennsqrt x = *y* $\longleftrightarrow$ *x* = *y* ∗ *y*
by *auto*

lemma *ennsqrt-bij*:
bij ennsqrt
by (*rule bij-betw-byWitness*[*of* - λ*x*. *x* ∗ *x*], *auto*)

lemma *ennsqrt-strict-mono*:
strict-mono ennsqrt
unfolding *ennsqrt-def*

apply (*rule strict-mono-compose*[*OF ennexp-strict-mono*])
apply (*rule strict-mono-compose*[*OF - eln-strict-mono*])
by (*auto simp add*: *ereal-less-divide-pos ereal-mult-divide strict-mono-def*)

lemma *ennsqrt-mono*:
mono ennsqrt
using *ennsqrt-strict-mono* **by** (*simp add*: *strict-mono-mono*)

lemma *ennsqrt-mono2* [*mono-intros*]:
assumes $x \leq y$
shows *ennsqrt* $x \leq$ *ennsqrt* y
using *ennsqrt-mono assms* **unfolding** *mono-def* **by** *auto*

lemma *ennsqrt-continuous*:
continuous-on UNIV ennsqrt
apply (*rule continuous-onI-mono*)
using *ennsqrt-mono* **unfolding** *mono-def* **by** (*auto simp add*: *ennsqrt-bij bij-is-surj*)

lemma *ennsqrt-tendsto* [*tendsto-intros*]:
assumes $((\lambda n.\ u\ n) \longlongrightarrow l)\ F$
shows $((\lambda n.\ ennsqrt(u\ n)) \longlongrightarrow ennsqrt\ l)\ F$
using *ennsqrt-continuous assms* **by** (*metis UNIV-I continuous-on tendsto-compose*)

lemma *ennsqrt-ennreal-ennreal-sqrt* [*simp*]:
assumes $t \geq (0::real)$
shows *ennsqrt* (*ennreal t*) = *ennreal* (*sqrt t*)
proof −
have *ennreal t* = *ennreal* (*sqrt t*) ∗ *ennreal*(*sqrt t*)
apply (*subst ennreal-mult*[*symmetric*]) **using** *assms* **by** *auto*
then show *?thesis*
by *auto*
qed

lemma *ennreal-sqrt2*:
ennreal (*sqrt 2*) = *ennsqrt 2*
using *ennsqrt-ennreal-ennreal-sqrt*[*of 2*] **by** *auto*

lemma *ennsqrt-4* [*simp*]:
ennsqrt 4 = *2*
by (*metis ennreal-numeral ennsqrt-ennreal-ennreal-sqrt real-sqrt-four zero-le-numeral*)

lemma *ennsqrt-le* [*simp*]:
ennsqrt $x \leq$ *ennsqrt* $y \longleftrightarrow x \leq y$
proof
assume *ennsqrt* $x \leq$ *ennsqrt* y
then have *ennsqrt* x ∗ *ennsqrt* $x \leq$ *ennsqrt* y ∗ *ennsqrt* y
by (*intro mult-mono*, *auto*)
then show $x \leq y$ **by** *auto*
qed (*auto intro*: *mono-intros*)

We can also define the square root on ereal using the square root on ennreal, and 0 for negative numbers.

definition *esqrt*::*ereal* ⇒ *ereal*
  where *esqrt* x = *enn2ereal*(*ennsqrt* (*e2ennreal* x))

lemma *esqrt-square* [*simp*]:
  assumes $x \geq 0$
  shows (*esqrt* x) ∗ (*esqrt* x) = x
unfolding *esqrt-def times-ennreal.rep-eq*[*symmetric*] *ennsqrt-square*[*of e2ennreal x*]
using *assms enn2ereal-e2ennreal* **by** *auto*

lemma *esqrt-of-neg* [*simp*]:
  assumes $x \leq 0$
  shows *esqrt* x = *0*
 unfolding *esqrt-def e2ennreal-neg*[*OF assms*] **by** (*auto simp add*: *zero-ennreal.rep-eq*)

lemma *esqrt-nonneg* [*simp*]:
  esqrt $x \geq 0$
unfolding *esqrt-def* **by** *auto*

lemma *esqrt-eq-iff-square* [*simp*]:
  assumes $x \geq 0$ $y \geq 0$
  shows *esqrt* $x = y \longleftrightarrow x = y * y$
using *esqrt-def esqrt-square assms* **apply** *auto*
by (*metis e2ennreal-enn2ereal ennsqrt-square2 eq-onp-same-args ereal-ennreal-cases leD times-ennreal.abs-eq*)

lemma *esqrt-simps* [*simp*]:
  esqrt 0 = *0*
  esqrt 1 = *1*
  esqrt ∞ = ∞
  esqrt top = *top*
  esqrt (−∞) = *0*
by (*auto simp*: *top-ereal-def*)

lemma *esqrt-mult*:
  assumes $a \geq 0$
  shows *esqrt*($a * b$) = *esqrt* a ∗ *esqrt* b
proof (*cases* $b \geq 0$)
  case *True*
  show *?thesis*
    unfolding *esqrt-def* **apply** (*subst times-ennreal.rep-eq*[*symmetric*])
    apply (*subst ennsqrt-mult*[*of e2ennreal a e2ennreal b*, *symmetric*])
    apply (*subst times-ennreal.abs-eq*)
    using *assms True* **by** (*auto simp add*: *eq-onp-same-args*)
next
  case *False*
  then have $a * b \leq 0$ **using** *assms ereal-mult-le-0-iff* **by** *auto*
  then have *esqrt*($a * b$) = *0* **by** *auto*

moreover have *esqrt b = 0* **using** *False* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *esqrt-square2* [*simp*]:
esqrt(*x* ∗ *x*) = *abs*(*x*)
proof −
have *esqrt*(*x* ∗ *x*) = *esqrt*(*abs x* ∗ *abs x*)
by (*metis* (*no-types*, *opaque-lifting*) *abs-ereal-ge0 ereal-abs-mult ereal-zero-le-0-iff linear*)
also have ... = *abs x*
by (*auto simp add*: *esqrt-mult*)
finally show *?thesis* **by** *auto*
qed

lemma *esqrt-mono*:
mono esqrt
unfolding *esqrt-def mono-def* **by** (*auto intro*: *mono-intros*)

lemma *esqrt-mono2* [*mono-intros*]:
assumes $x \leq y$
shows *esqrt x* ≤ *esqrt y*
using *esqrt-mono assms* **unfolding** *mono-def* **by** *auto*

lemma *esqrt-continuous*:
continuous-on UNIV esqrt
unfolding *esqrt-def* **apply** (*rule continuous-on-compose2*[*of UNIV enn2ereal*], *intro continuous-on-enn2ereal*)
by (*rule continuous-on-compose2*[*of UNIV ennsqrt*], *auto intro*!: *ennsqrt-continuous continuous-on-e2ennreal*)

lemma *esqrt-tendsto* [*tendsto-intros*]:
assumes ((λ*n*. *u n*) ⟶ *l*) *F*
shows ((λ*n*. *esqrt*(*u n*)) ⟶ *esqrt l*) *F*
using *esqrt-continuous assms* **by** (*metis UNIV-I continuous-on tendsto-compose*)

lemma *esqrt-ereal-ereal-sqrt* [*simp*]:
assumes *t* ≥ (*0*::*real*)
shows *esqrt* (*ereal t*) = *ereal* (*sqrt t*)
proof −
have *ereal t* = *ereal* (*sqrt t*) ∗ *ereal*(*sqrt t*)
using *assms* **by** *auto*
then show *?thesis*
using *assms ereal-less-eq*(*5*) *esqrt-mult esqrt-square real-sqrt-ge-zero* **by** *presburger*
qed

lemma *ereal-sqrt2*:
ereal (*sqrt 2*) = *esqrt 2*

using *esqrt-ereal-ereal-sqrt*[*of 2*] **by** *auto*

lemma *esqrt-4* [*simp*]:
  $esqrt\ 4 = 2$
by *auto*

lemma *esqrt-le* [*simp*]:
  $esqrt\ x \le esqrt\ y \longleftrightarrow (x \le 0 \lor x \le y)$
apply (*auto simp add*: *esqrt-mono2*)
by (*metis eq-iff ereal-zero-times esqrt-mono2 esqrt-square le-cases*)

Finally, we define eexp, as the composition of ennexp and the injection of ennreal in ereal.

definition *eexp*::*ereal* ⇒ *ereal* **where**
  $eexp\ x = enn2ereal\ (ennexp\ x)$

lemma *eexp-special-values* [*simp*]:
  $eexp\ 0 = 1$
  $eexp\ (\infty) = \infty$
  $eexp(-\infty) = 0$
unfolding *eexp-def* **by** (*auto simp add*: *zero-ennreal.rep-eq one-ennreal.rep-eq*)

lemma *eexp-strict-mono*:
  strict-mono eexp
unfolding *eexp-def* **using** *ennexp-strict-mono* **unfolding** *strict-mono-def* **by** (*auto intro*: *mono-intros*)

lemma *eexp-mono*:
  mono eexp
using *eexp-strict-mono* **by** (*simp add*: *strict-mono-mono*)

lemma *eexp-strict-mono2* [*mono-intros*]:
  assumes $x < y$
  shows $eexp\ x < eexp\ y$
using *eexp-strict-mono assms* **unfolding** *strict-mono-def* **by** *auto*

lemma *eexp-mono2* [*mono-intros*]:
  assumes $x \le y$
  shows $eexp\ x \le eexp\ y$
using *eexp-mono assms* **unfolding** *mono-def* **by** *auto*

lemma *eexp-le-eexp-iff-le*:
  $eexp\ x \le eexp\ y \longleftrightarrow x \le y$
using *eexp-strict-mono2 not-le* **by** (*auto intro*: *mono-intros*)

lemma *eexp-lt-eexp-iff-lt*:
  $eexp\ x < eexp\ y \longleftrightarrow x < y$
using *eexp-mono2 not-le* **by** (*auto intro*: *mono-intros*)

lemma *eexp-special-values-iff* [*simp*]:
  $eexp\ x = 0 \longleftrightarrow x = -\infty$
  $eexp\ x = 1 \longleftrightarrow x = 0$
  $eexp\ x = \infty \longleftrightarrow x = \infty$
  $eexp\ x = top \longleftrightarrow x = \infty$
unfolding *eexp-def* **apply** (*auto simp add*: *zero-ennreal.rep-eq one-ennreal.rep-eq top-ereal-def*)
apply (*metis e2ennreal-enn2ereal ennexp.simps*(*3*) *ennexp-strict-mono strict-mono-eq zero-ennreal-def*)
by (*metis e2ennreal-enn2ereal eln-ennexp eln-simps*(*2*) *one-ennreal-def*)

lemma *eexp-ineq-iff* [*simp*]:
  $eexp\ x \le 1 \longleftrightarrow x \le 0$
  $eexp\ x \ge 1 \longleftrightarrow x \ge 0$
  $eexp\ x > 1 \longleftrightarrow x > 0$
  $eexp\ x < 1 \longleftrightarrow x < 0$
  $eexp\ x \ge 0$
  $eexp\ x > 0 \longleftrightarrow x \ne -\infty$
  $eexp\ x < \infty \longleftrightarrow x \ne \infty$
apply (*metis eexp-le-eexp-iff-le eexp-lt-eexp-iff-lt eexp-special-values*)+
apply (*simp add*: *eexp-def*)
using *eexp-strict-mono2* **apply** (*force*)
by *simp*

lemma *eexp-ineq* [*mono-intros*]:
  $x \le 0 \Longrightarrow eexp\ x \le 1$
  $x < 0 \Longrightarrow eexp\ x < 1$
  $x \ge 0 \Longrightarrow eexp\ x \ge 1$
  $x > 0 \Longrightarrow eexp\ x > 1$
  $eexp\ x \ge 0$
  $x > -\infty \Longrightarrow eexp\ x > 0$
  $x < \infty \Longrightarrow eexp\ x < \infty$
by *auto*

lemma *eexp-continuous*:
  continuous-on UNIV eexp
unfolding *eexp-def* **by** (*rule continuous-on-compose2*[*of UNIV enn2ereal*], *auto simp*: *continuous-on-enn2ereal ennexp-continuous*)

lemma *eexp-tendsto'* [*simp*]:
  $((\lambda n.\ eexp(u\ n)) \longrightarrow eexp\ l)\ F \longleftrightarrow ((\lambda n.\ u\ n) \longrightarrow l)\ F$
proof
  assume *H*: $((\lambda n.\ eexp\ (u\ n)) \longrightarrow eexp\ l)\ F$
  have $((\lambda n.\ eln\ (e2ennreal\ (eexp\ (u\ n)))) \longrightarrow eln\ (e2ennreal\ (eexp\ l)))\ F$
    by (*intro tendsto-intros H*)
  then show $(u \longrightarrow l)\ F$
    unfolding *eexp-def* **by** *auto*
next

assume $(u \longrightarrow l)$ F
then show $((\lambda n.\ eexp(u\ n)) \longrightarrow eexp\ l)$ F
using *eexp-continuous* **by** (*metis UNIV-I continuous-on tendsto-compose*)
qed

lemma *eexp-tendsto* [*tendsto-intros*]:
assumes $((\lambda n.\ u\ n) \longrightarrow l)$ F
shows $((\lambda n.\ eexp(u\ n)) \longrightarrow eexp\ l)$ F
using *assms* **by** *auto*

lemma *eexp-add-mult*:
assumes $\neg((a = \infty \wedge b = -\infty) \vee (a = -\infty \wedge b = \infty))$
shows $eexp(a+b) = eexp\ a * eexp\ b$
using *ennexp-add-mult*[*OF assms*] **unfolding** *eexp-def* **by** (*simp add*: *times-ennreal.rep-eq*)

lemma *eexp-ereal* [*simp*]:
$eexp(ereal\ x) = ereal(exp\ x)$
by (*simp add*: *eexp-def*)

end

3 Hausdorff distance

theory *Hausdorff-Distance*
imports *Library-Complements*
begin

3.1 Preliminaries

3.2 Hausdorff distance

The Hausdorff distance between two subsets of a metric space is the minimal M such that each set is included in the M-neighborhood of the other. For nonempty bounded sets, it satisfies the triangular inequality, it is symmetric, but it vanishes on sets that have the same closure. In particular, it defines a distance on closed bounded nonempty sets. We establish all these properties below.

definition *hausdorff-distance*::($'a$::*metric-space*) *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *real*
where *hausdorff-distance* $A\ B = (if\ A = \{\} \vee B = \{\} \vee (\neg(bounded\ A)) \vee (\neg(bounded\ B))\ then\ 0$
$else\ max\ (SUP\ x{\in}A.\ infdist\ x\ B)\ (SUP\ x{\in}B.\ infdist\ x\ A))$

lemma *hausdorff-distance-self* [*simp*]:
hausdorff-distance $A\ A = 0$
unfolding *hausdorff-distance-def* **by** *auto*

lemma *hausdorff-distance-sym*:

hausdorff-distance A B = hausdorff-distance B A
unfolding *hausdorff-distance-def* **by** *auto*

lemma *hausdorff-distance-points* [*simp*]:
hausdorff-distance $\{x\}$ $\{y\}$ = *dist x y*
unfolding *hausdorff-distance-def* **by** (*auto*, *metis dist-commute max.idem*)

The Hausdorff distance is expressed in terms of a supremum. To use it, one needs again and again to show that this is the supremum of a set which is bounded from above.

lemma *bdd-above-infdist-aux*:
assumes *bounded A bounded B*
shows *bdd-above* ((λx. *infdist x B*)‘*A*)
proof (*cases B* = {})
case *True*
then show *?thesis* **unfolding** *infdist-def* **by** *auto*
next
case *False*
then obtain *y* **where** $y \in B$ **by** *auto*
then have *infdist x B* $\leq$ *dist x y* **if** $x \in A$ **for** *x*
by (*simp add*: *infdist-le*)
then show *?thesis* **unfolding** *bdd-above-def*
by (*auto*, *metis assms*(*1*) *bounded-any-center dist-commute order-trans*)
qed

lemma *hausdorff-distance-nonneg* [*simp*, *mono-intros*]:
hausdorff-distance A B $\geq$ *0*
proof (*cases A* = {} $\lor$ *B* = {} $\lor$ ($\neg$(*bounded A*)) $\lor$ ($\neg$(*bounded B*)))
case *True*
then show *?thesis* **unfolding** *hausdorff-distance-def* **by** *auto*
next
case *False*
then have *A* $\neq$ {} *B* $\neq$ {} *bounded A bounded B* **by** *auto*
have (*SUP x*$\in$*A*. *infdist x B*) $\geq$ *0*
using *bdd-above-infdist-aux*[*OF* ‹*bounded A*› ‹*bounded B*›] *infdist-nonneg*
by (*metis* ‹*A* $\neq$ {}› *all-not-in-conv cSUP-upper2*)
moreover have (*SUP x*$\in$*B*. *infdist x A*) $\geq$ *0*
using *bdd-above-infdist-aux*[*OF* ‹*bounded B*› ‹*bounded A*›] *infdist-nonneg*
by (*metis* ‹*B* $\neq$ {}› *all-not-in-conv cSUP-upper2*)
ultimately show *?thesis* **unfolding** *hausdorff-distance-def* **by** *auto*
qed

lemma *hausdorff-distanceI*:
assumes $\bigwedge x.\ x \in A \Longrightarrow$ *infdist x B* $\leq D$
$\bigwedge x.\ x \in B \Longrightarrow$ *infdist x A* $\leq D$
$D \geq 0$
shows *hausdorff-distance A B* $\leq D$
proof (*cases A* = {} $\lor$ *B* = {} $\lor$ ($\neg$(*bounded A*)) $\lor$ ($\neg$(*bounded B*)))
case *True*

then show *?thesis* **unfolding** *hausdorff-distance-def* **using** ‹$D \geq 0$› **by** *auto*
next
case *False*
then have $A \neq \{\}$ $B \neq \{\}$ *bounded A bounded B* **by** *auto*
have (*SUP x*∈*A. infdist x B*) $\leq D$
apply (*rule cSUP-least*, *simp add*: ‹$A \neq \{\}$›) **using** *assms*(*1*) **by** *blast*
moreover have (*SUP x*∈*B. infdist x A*) $\leq D$
apply (*rule cSUP-least*, *simp add*: ‹$B \neq \{\}$›) **using** *assms*(*2*) **by** *blast*
ultimately show *?thesis* **unfolding** *hausdorff-distance-def* **using** *False* **by** *auto*
qed

lemma *hausdorff-distanceI2*:
assumes $\bigwedge x.\ x \in A \Longrightarrow \exists y{\in}B.\ dist\ x\ y \leq D$
$\bigwedge x.\ x \in B \Longrightarrow \exists y{\in}A.\ dist\ x\ y \leq D$
$D \geq 0$
shows *hausdorff-distance A B* $\leq D$
proof (*rule hausdorff-distanceI*[*OF - -* ‹$D \geq 0$›])
fix *x* **assume** $x \in A$ **show** *infdist x B* $\leq D$ **using** *assms*(*1*)[*OF* ‹$x \in A$›] *infdist-le2* **by** *fastforce*
next
fix *x* **assume** $x \in B$ **show** *infdist x A* $\leq D$ **using** *assms*(*2*)[*OF* ‹$x \in B$›] *infdist-le2* **by** *fastforce*
qed

lemma *infdist-le-hausdorff-distance* [*mono-intros*]:
assumes $x \in A$ *bounded A bounded B*
shows *infdist x B* $\leq$ *hausdorff-distance A B*
proof (*cases* $B = \{\}$)
case *True*
then have *infdist x B* $= 0$ **unfolding** *infdist-def* **by** *auto*
then show *?thesis* **using** *hausdorff-distance-nonneg* **by** *auto*
next
case *False*
have *infdist x B* $\leq$ (*SUP y*∈*A. infdist y B*)
using *bdd-above-infdist-aux*[*OF* ‹*bounded A*› ‹*bounded B*›] **by** (*meson assms*(*1*) *cSUP-upper*)
then show *?thesis* **unfolding** *hausdorff-distance-def* **using** *assms False* **by** *auto*
qed

lemma *hausdorff-distance-infdist-triangle* [*mono-intros*]:
assumes $B \neq \{\}$ *bounded B bounded C*
shows *infdist x C* $\leq$ *infdist x B* + *hausdorff-distance B C*
proof (*cases* $C = \{\}$)
case *True*
then have *infdist x C* $= 0$ **unfolding** *infdist-def* **by** *auto*
then show *?thesis* **using** *infdist-nonneg*[*of x B*] *hausdorff-distance-nonneg*[*of B C*] **by** *auto*
next
case *False*

have *infdist x C* − *hausdorff-distance B C* ≤ *dist x b* **if** *b* ∈ *B* **for** *b*
proof −
have *infdist x C* ≤ *infdist b C* + *dist x b* **by** (*rule infdist-triangle*)
also have ... ≤ *dist x b* + *hausdorff-distance B C*
using *infdist-le-hausdorff-distance*[*OF* ‹*b* ∈ *B*› ‹*bounded B*› ‹*bounded C*›] **by** *auto*
finally show *?thesis* **by** *auto*
qed
then have *infdist x C* − *hausdorff-distance B C* ≤ *infdist x B*
unfolding *infdist-def* **using** ‹*B* ≠ {}› **by** (*simp add*: *le-cINF-iff*)
then show *?thesis* **by** *auto*
qed

lemma *hausdorff-distance-triangle* [*mono-intros*]:
assumes *B* ≠ {} *bounded B*
shows *hausdorff-distance A C* ≤ *hausdorff-distance A B* + *hausdorff-distance B C*
proof (*cases A* = {} ∨ *C* = {} ∨ (¬(*bounded A*)) ∨ (¬(*bounded C*)))
case *True*
then have *hausdorff-distance A C* = *0* **unfolding** *hausdorff-distance-def* **by** *auto*
then show *?thesis*
using *hausdorff-distance-nonneg*[*of A B*] *hausdorff-distance-nonneg*[*of B C*] **by** *auto*
next
case *False*
then have ∗: *A* ≠ {} *C* ≠ {} *bounded A bounded C* **by** *auto*
define *M* **where** *M* = *hausdorff-distance A B* + *hausdorff-distance B C*
have *infdist x C* ≤ *M* **if** *x* ∈ *A* **for** *x*
using *hausdorff-distance-infdist-triangle*[*OF* ‹*B* ≠ {}› ‹*bounded B* › ‹*bounded C*›, *of x*]
infdist-le-hausdorff-distance[*OF* ‹*x* ∈ *A*› ‹*bounded A*› ‹*bounded B*›] **by** (*auto simp add*: *M-def*)
moreover have *infdist x A* ≤ *M* **if** *x* ∈ *C* **for** *x*
using *hausdorff-distance-infdist-triangle*[*OF* ‹*B* ≠ {}› ‹*bounded B* › ‹*bounded A*›, *of x*]
infdist-le-hausdorff-distance[*OF* ‹*x* ∈ *C*› ‹*bounded C*› ‹*bounded B*›]
by (*auto simp add*: *hausdorff-distance-sym M-def*)
ultimately have *hausdorff-distance A C* ≤ *M*
unfolding *hausdorff-distance-def* **using** ∗ *bdd-above-infdist-aux* **by** (*auto simp add*: *cSUP-least*)
then show *?thesis* **unfolding** *M-def* **by** *auto*
qed

lemma *hausdorff-distance-subset*:
assumes *A* ⊆ *B A* ≠ {} *bounded B*
shows *hausdorff-distance A B* = (*SUP x*∈*B*. *infdist x A*)
proof −
have *H*: *B* ≠ {} *bounded A* **using** *assms bounded-subset* **by** *auto*

have (*SUP x∈A. infdist x B*) = *0* **using** *assms* **by** (*simp add*: *subset-eq*)
moreover have (*SUP x∈B. infdist x A*) ≥ *0*
using *bdd-above-infdist-aux*[*OF* ‹*bounded B*› ‹*bounded A*›] *infdist-nonneg*[*of* - *A*]
by (*meson H*(*1*) *cSUP-upper2 ex-in-conv*)
ultimately show *?thesis* **unfolding** *hausdorff-distance-def* **using** *assms H* **by** *auto*
qed

lemma *hausdorff-distance-closure* [*simp*]:
hausdorff-distance A (*closure A*) = *0*
proof (*cases A* = {} ∨ (¬(*bounded A*)))
case *True*
then show *?thesis* **unfolding** *hausdorff-distance-def* **by** *auto*
next
case *False*
then have *A* ≠ {} *bounded A* **by** *auto*
then have *closure A* ≠ {} *bounded* (*closure A*) *A* ⊆ *closure A*
using *closure-subset* **by** *auto*
have *infdist x A* = *0* **if** *x* ∈ *closure A* **for** *x*
using *in-closure-iff-infdist-zero*[*OF* ‹*A* ≠ {}›] *that* **by** *auto*
then have (*SUP x∈closure A. infdist x A*) = *0*
using ‹*closure A* ≠ {}› **by** *auto*
then show *?thesis*
unfolding *hausdorff-distance-subset*[*OF* ‹*A* ⊆ *closure A*› ‹*A* ≠ {}› ‹*bounded* (*closure A*)›] **by** *simp*
qed

lemma *hausdorff-distance-closures* [*simp*]:
hausdorff-distance (*closure A*) (*closure B*) = *hausdorff-distance A B*
proof (*cases A* = {} ∨ *B* = {} ∨ (¬(*bounded A*)) ∨ (¬(*bounded B*)))
case *True*
then have ∗: *hausdorff-distance A B* = *0* **unfolding** *hausdorff-distance-def* **by** *auto*
have *closure A* = {} ∨ (¬(*bounded* (*closure A*))) ∨ *closure B* = {} ∨ (¬(*bounded* (*closure B*)))
using *True bounded-subset closure-subset* **by** *auto*
then have *hausdorff-distance* (*closure A*) (*closure B*) = *0*
unfolding *hausdorff-distance-def* **by** *auto*
then show *?thesis* **using** ∗ **by** *simp*
next
case *False*
then have *H*: *A* ≠ {} *B* ≠ {} *bounded A bounded B* **by** *auto*
then have *H2*: *closure A* ≠ {} *closure B* ≠ {} *bounded* (*closure A*) *bounded* (*closure B*)
by *auto*
have *hausdorff-distance A B* ≤ *hausdorff-distance A* (*closure A*) + *hausdorff-distance* (*closure A*) *B*
apply (*rule hausdorff-distance-triangle*) **using** *H H2* **by** *auto*

also have ... = *hausdorff-distance* (*closure A*) *B*
using *hausdorff-distance-closure* **by** *auto*
also have ... $\leq$ *hausdorff-distance* (*closure A*) (*closure B*) + *hausdorff-distance* (*closure B*) *B*
apply (*rule hausdorff-distance-triangle*) **using** *H H2* **by** *auto*
also have ... = *hausdorff-distance* (*closure A*) (*closure B*)
using *hausdorff-distance-closure* **by** (*auto simp add*: *hausdorff-distance-sym*)
finally have $*$: *hausdorff-distance A B* $\leq$ *hausdorff-distance* (*closure A*) (*closure B*) **by** *simp*

have *hausdorff-distance* (*closure A*) (*closure B*) $\leq$ *hausdorff-distance* (*closure A*) *A* + *hausdorff-distance A* (*closure B*)
apply (*rule hausdorff-distance-triangle*) **using** *H H2* **by** *auto*
also have ... = *hausdorff-distance A* (*closure B*)
using *hausdorff-distance-closure* **by** (*auto simp add*: *hausdorff-distance-sym*)
also have ... $\leq$ *hausdorff-distance A B* + *hausdorff-distance B* (*closure B*)
apply (*rule hausdorff-distance-triangle*) **using** *H H2* **by** *auto*
also have ... = *hausdorff-distance A B*
using *hausdorff-distance-closure* **by** (*auto simp add*: *hausdorff-distance-sym*)
finally have *hausdorff-distance* (*closure A*) (*closure B*) $\leq$ *hausdorff-distance A B* **by** *simp*
then show *?thesis* **using** $*$ **by** *auto*
qed

lemma *hausdorff-distance-zero*:
assumes $A \neq \{\}$ *bounded A* $B \neq \{\}$ *bounded B*
shows *hausdorff-distance A B* = *0* $\longleftrightarrow$ *closure A* = *closure B*
proof
assume *H*: *hausdorff-distance A B* = *0*
have $A \subseteq$ *closure B*
proof
fix *x* **assume** $x \in A$
have *infdist x B* = *0*
using *infdist-le-hausdorff-distance*[*OF* ‹$x \in A$› ‹*bounded A*› ‹*bounded B*›] *H* *infdist-nonneg*[*of x B*] **by** *auto*
then show $x \in$ *closure B* **using** *in-closure-iff-infdist-zero*[*OF* ‹$B \neq \{\}$›] **by** *auto*
qed
then have *A*: *closure A* $\subseteq$ *closure B* **by** (*simp add*: *closure-minimal*)

have $B \subseteq$ *closure A*
proof
fix *x* **assume** $x \in B$
have *infdist x A* = *0*
using *infdist-le-hausdorff-distance*[*OF* ‹$x \in B$› ‹*bounded B*› ‹*bounded A*›] *H* *infdist-nonneg*[*of x A*]
by (*auto simp add*: *hausdorff-distance-sym*)
then show $x \in$ *closure A* **using** *in-closure-iff-infdist-zero*[*OF* ‹$A \neq \{\}$›] **by** *auto*

qed
then have *closure B* $\subseteq$ *closure A* **by** (*simp add*: *closure-minimal*)
then show *closure A* = *closure B* **using** *A* **by** *auto*
next
assume *closure A* = *closure B*
then show *hausdorff-distance A B* = *0*
using *hausdorff-distance-closures*[*of A B*] **by** *auto*
qed

lemma *hausdorff-distance-vimage*:
assumes $\bigwedge x.\ x \in A \Longrightarrow$ *dist* (*f x*) (*g x*) $\leq$ *C*
$C \geq 0$
shows *hausdorff-distance* (*f'A*) (*g'A*) $\leq$ *C*
apply (*rule hausdorff-distanceI2*[*OF - - ‹C ≥ 0›*]) **using** *assms* **by** (*auto simp add*: *dist-commute*, *auto*)

lemma *hausdorff-distance-union* [*mono-intros*]:
assumes $A \neq \{\}$ $B \neq \{\}$ $C \neq \{\}$ $D \neq \{\}$
shows *hausdorff-distance* ($A \cup B$) ($C \cup D$) $\leq$ *max* (*hausdorff-distance A C*) (*hausdorff-distance B D*)
proof (*cases bounded A* $\wedge$ *bounded B* $\wedge$ *bounded C* $\wedge$ *bounded D*)
case *False*
then have *hausdorff-distance* ($A \cup B$) ($C \cup D$) = *0*
unfolding *hausdorff-distance-def* **by** *auto*
then show *?thesis*
by (*simp add*: *hausdorff-distance-nonneg le-max-iff-disj*)
next
case *True*
show *?thesis*
proof (*rule hausdorff-distanceI*, *auto*)
fix *x* **assume** *H*: $x \in A$
have *infdist x* ($C \cup D$) $\leq$ *infdist x C*
by (*simp add*: *assms infdist-union-min*)
also have ... $\leq$ *hausdorff-distance A C*
apply (*rule infdist-le-hausdorff-distance*) **using** *H True* **by** *auto*
also have ... $\leq$ *max* (*hausdorff-distance A C*) (*hausdorff-distance B D*)
by *auto*
finally show *infdist x* ($C \cup D$) $\leq$ *max* (*hausdorff-distance A C*) (*hausdorff-distance B D*)
by *simp*
next
fix *x* **assume** *H*: $x \in B$
have *infdist x* ($C \cup D$) $\leq$ *infdist x D*
by (*simp add*: *assms infdist-union-min*)
also have ... $\leq$ *hausdorff-distance B D*
apply (*rule infdist-le-hausdorff-distance*) **using** *H True* **by** *auto*
also have ... $\leq$ *max* (*hausdorff-distance A C*) (*hausdorff-distance B D*)
by *auto*
finally show *infdist x* ($C \cup D$) $\leq$ *max* (*hausdorff-distance A C*) (*hausdorff-distance*

```
B D)
      by simp
  next
    fix x assume H: x ∈ C
    have infdist x (A ∪ B) ≤ infdist x A
      by (simp add: assms infdist-union-min)
    also have ... ≤ hausdorff-distance C A
      apply (rule infdist-le-hausdorff-distance) using H True by auto
    also have ... ≤ max (hausdorff-distance A C) (hausdorff-distance B D)
      using hausdorff-distance-sym[of A C] by auto
   finally show infdist x (A ∪ B) ≤ max (hausdorff-distance A C) (hausdorff-distance
B D)
      by simp
  next
    fix x assume H: x ∈ D
    have infdist x (A ∪ B) ≤ infdist x B
      by (simp add: assms infdist-union-min)
    also have ... ≤ hausdorff-distance D B
      apply (rule infdist-le-hausdorff-distance) using H True by auto
    also have ... ≤ max (hausdorff-distance A C) (hausdorff-distance B D)
      using hausdorff-distance-sym[of B D] by auto
   finally show infdist x (A ∪ B) ≤ max (hausdorff-distance A C) (hausdorff-distance
B D)
      by simp
  qed (simp add: le-max-iff-disj)
qed

end
```

4 Isometries

```
theory Isometries
  imports Library-Complements Hausdorff-Distance
begin
```

Isometries, i.e., functions that preserve distances, show up very often in mathematics. We introduce a dedicated definition, and show its basic properties.

```
definition isometry-on::('a::metric-space) set ⇒ ('a ⇒ ('b::metric-space)) ⇒ bool
  where isometry-on X f = (∀ x ∈ X. ∀ y ∈ X. dist (f x) (f y) = dist x y)

definition isometry :: ('a::metric-space ⇒ 'b::metric-space) ⇒ bool
  where isometry f ≡ isometry-on UNIV f ∧ range f = UNIV

lemma isometry-on-subset:
  assumes isometry-on X f
          Y ⊆ X
  shows isometry-on Y f
```

using *assms* **unfolding** *isometry-on-def* **by** *auto*

lemma *isometry-onI* [*intro?*]:
  assumes $\bigwedge x\ y.\ x \in X \Longrightarrow y \in X \Longrightarrow dist\ (f\ x)\ (f\ y) = dist\ x\ y$
  shows *isometry-on X f*
using *assms* **unfolding** *isometry-on-def* **by** *auto*

lemma *isometry-onD*:
  assumes *isometry-on X f*
    $x \in X$ $y \in X$
  shows $dist\ (f\ x)\ (f\ y) = dist\ x\ y$
using *assms* **unfolding** *isometry-on-def* **by** *auto*

lemma *isometryI* [*intro?*]:
  assumes $\bigwedge x\ y.\ dist\ (f\ x)\ (f\ y) = dist\ x\ y$
    $range\ f = UNIV$
  shows *isometry f*
unfolding *isometry-def isometry-on-def* **using** *assms* **by** *auto*

lemma
  assumes *isometry-on X f*
  shows *isometry-on-lipschitz*: *1−lipschitz-on X f*
    and *isometry-on-uniformly-continuous*: *uniformly-continuous-on X f*
    and *isometry-on-continuous*: *continuous-on X f*
proof −
  show *1−lipschitz-on X f* **apply** (*rule lipschitz-onI*) **using** *isometry-onD*[*OF assms*] **by** *auto*
  then show *uniformly-continuous-on X f continuous-on X f*
    using *lipschitz-on-uniformly-continuous lipschitz-on-continuous-on* **by** *auto*
qed

lemma *isometryD*:
  assumes *isometry f*
  shows *isometry-on UNIV f*
    $dist\ (f\ x)\ (f\ y) = dist\ x\ y$
    $range\ f = UNIV$
    1−lipschitz-on UNIV f
    uniformly-continuous-on UNIV f
    continuous-on UNIV f
using *assms* **unfolding** *isometry-def isometry-on-def* **apply** *auto*
using *isometry-on-lipschitz isometry-on-uniformly-continuous isometry-on-continuous assms* **unfolding** *isometry-def* **by** *blast+*

lemma *isometry-on-injective*:
  assumes *isometry-on X f*
  shows *inj-on f X*
using *assms inj-on-def isometry-on-def* **by** *force*

lemma *isometry-on-compose*:

assumes *isometry-on* X f
isometry-on $(f`X)$ g
shows *isometry-on* X $(\lambda x.\ g(f\ x))$
using *assms* **unfolding** *isometry-on-def* **by** *auto*

lemma *isometry-on-cong*:
assumes *isometry-on* X f
$\bigwedge x.\ x \in X \Longrightarrow g\ x = f\ x$
shows *isometry-on* X g
using *assms* **unfolding** *isometry-on-def* **by** *auto*

lemma *isometry-on-inverse*:
assumes *isometry-on* X f
shows *isometry-on* $(f`X)$ $($*inv-into* X $f)$
$\bigwedge x.\ x \in X \Longrightarrow ($*inv-into* $X\ f)\ (f\ x) = x$
$\bigwedge y.\ y \in f`X \Longrightarrow f\ ($*inv-into* $X\ f\ y) = y$
bij-betw f X $(f`X)$
proof −
show ∗: *bij-betw* f X $(f`X)$
using *assms* **unfolding** *bij-betw-def inj-on-def isometry-on-def* **by** *force*
show *isometry-on* $(f`X)$ $($*inv-into* X $f)$
using *assms* **unfolding** *isometry-on-def*
by (*auto*) (*metis* (*mono-tags*, *lifting*) *dist-eq-0-iff inj-on-def inv-into-f-f*)
fix x **assume** $x \in X$
then show $($*inv-into* $X\ f)\ (f\ x) = x$
using ∗ **by** (*simp add*: *bij-betw-def*)
next
fix y **assume** $y \in f`X$
then show $f\ ($*inv-into* $X\ f\ y) = y$
by (*simp add*: *f-inv-into-f*)
qed

lemma *isometry-inverse*:
assumes *isometry* f
shows *isometry* (*inv* f)
bij f
using *isometry-on-inverse*[*OF isometryD*(*1*)[*OF assms*]] *isometryD*(*3*)[*OF assms*]
unfolding *isometry-def* **by** (*auto simp add*: *bij-imp-bij-inv bij-is-surj*)

lemma *isometry-on-homeomorphism*:
assumes *isometry-on* X f
shows *homeomorphism* X $(f`X)$ f $($*inv-into* X $f)$
homeomorphism-on X f
X *homeomorphic* $f`X$
proof −
show ∗: *homeomorphism* X $(f`X)$ f $($*inv-into* X $f)$
apply (*rule homeomorphismI*) **using** *uniformly-continuous-imp-continuous*[*OF isometry-on-uniformly-continuous*]
isometry-on-inverse[*OF assms*] *assms* **by** *auto*

then show *X homeomorphic f‘X*
  unfolding *homeomorphic-def* **by** *auto*
show *homeomorphism-on X f*
  unfolding *homeomorphism-on-def* **using** ∗ **by** *auto*
qed

lemma *isometry-homeomorphism*:
fixes *f*::(′*a*::*metric-space*) ⇒ (′*b*::*metric-space*)
assumes *isometry f*
shows *homeomorphism UNIV UNIV f* (*inv f*)
  (*UNIV*::′*a set*) *homeomorphic* (*UNIV*::′*b set*)
using *isometry-on-homeomorphism*[*OF isometryD*(*1*)[*OF assms*]] *isometryD*(*3*)[*OF assms*] **by** *auto*

lemma *isometry-on-closure*:
assumes *isometry-on X f*
  continuous-on (*closure X*) *f*
shows *isometry-on* (*closure X*) *f*
proof (*rule isometry-onI*)
fix *x y* **assume** *x* ∈ *closure X y* ∈ *closure X*
obtain *u v*::*nat* ⇒ ′*a* **where** ∗: ⋀*n*. *u n* ∈ *X u* ⟶ *x*
  ⋀*n*. *v n* ∈ *X v* ⟶ *y*
  using ‹*x* ∈ *closure X*› ‹*y* ∈ *closure X*› **unfolding** *closure-sequential* **by** *blast*
have (λ*n*. *f* (*u n*)) ⟶ *f x*
  using ∗(*1*) ∗(*2*) ‹*x* ∈ *closure X*› ‹*continuous-on* (*closure X*) *f*›
  unfolding *comp-def continuous-on-closure-sequentially*[*of X f*] **by** *auto*
moreover have (λ*n*. *f* (*v n*)) ⟶ *f y*
  using ∗(*3*) ∗(*4*) ‹*y* ∈ *closure X*› ‹*continuous-on* (*closure X*) *f*›
  unfolding *comp-def continuous-on-closure-sequentially*[*of X f*] **by** *auto*
ultimately have (λ*n*. *dist* (*f* (*u n*)) (*f* (*v n*))) ⟶ *dist* (*f x*) (*f y*)
  by (*simp add*: *tendsto-dist*)
then have (λ*n*. *dist* (*u n*) (*v n*)) ⟶ *dist* (*f x*) (*f y*)
  using *assms*(*1*) ∗(*1*) ∗(*3*) **unfolding** *isometry-on-def* **by** *auto*
moreover have (λ*n*. *dist* (*u n*) (*v n*)) ⟶ *dist x y*
  using ∗(*2*) ∗(*4*) **by** (*simp add*: *tendsto-dist*)
ultimately show *dist* (*f x*) (*f y*) = *dist x y* **using** *LIMSEQ-unique* **by** *auto*
qed

lemma *isometry-extend-closure*:
fixes *f*::(′*a*::*metric-space*) ⇒ (′*b*::*complete-space*)
assumes *isometry-on X f*
shows ∃ *g*. *isometry-on* (*closure X*) *g* ∧ (∀ *x*∈*X*. *g x* = *f x*)
proof −
obtain *g* **where** *g*: ⋀*x*. *x* ∈ *X* ⟹ *g x* = *f x uniformly-continuous-on* (*closure X*) *g*
  using *uniformly-continuous-on-extension-on-closure*[*OF isometry-on-uniformly-continuous*[*OF assms*]] **by** *metis*
have *isometry-on* (*closure X*) *g*
  apply (*rule isometry-on-closure*, *rule isometry-on-cong*[*OF assms*])

using *g uniformly-continuous-imp-continuous*[*OF g*(*2*)] **by** *auto*
then show *?thesis* **using** *g*(*1*) **by** *auto*
qed

lemma *isometry-on-complete-image*:
assumes *isometry-on X f*
complete X
shows *complete* (*f'X*)
proof (*rule completeI*)
fix *u* :: *nat* ⇒ *'b* **assume** *u*: ∀ *n*. *u n* ∈ *f'X Cauchy u*
define *v* **where** *v* = (λ*n*. *inv-into X f* (*u n*))
have *v n* ∈ *X* **for** *n*
unfolding *v-def* **by** (*simp add*: *inv-into-into u*(*1*))
have *dist* (*v n*) (*v m*) = *dist* (*u n*) (*u m*) **for** *m n*
using *u*(*1*) *isometry-on-inverse*[*OF* ‹*isometry-on X f*›] **unfolding** *isometry-on-def v-def* **by** (*auto simp add*: *inv-into-into*)
then have *Cauchy v*
using *u*(*2*) **unfolding** *Cauchy-def* **by** *auto*
obtain *l* **where** *l* ∈ *X v* ⟶ *l*
apply (*rule completeE*[*OF* ‹*complete X*› - ‹*Cauchy v*›]) **using** ‹⋀*n*. *v n* ∈ *X*› **by** *auto*
have (λ*n*. *f* (*v n*)) ⟶ *f l*
apply (*rule continuous-on-tendsto-compose*[*OF isometry-on-continuous*[*OF* ‹*isometry-on X f*›]])
using ‹⋀*n*. *v n* ∈ *X*› ‹*l* ∈ *X*› ‹*v* ⟶ *l*› **by** *auto*
moreover have *f*(*v n*) = *u n* **for** *n*
unfolding *v-def* **by** (*simp add*: *f-inv-into-f u*(*1*))
ultimately have *u* ⟶ *f l* **by** *auto*
then show ∃ *m* ∈ *f'X*. *u* ⟶ *m* **using** ‹*l* ∈ *X*› **by** *auto*
qed

lemma *isometry-on-id* [*simp*]:
isometry-on A (λ*x*. *x*)
isometry-on A id
unfolding *isometry-on-def* **by** *auto*

lemma *isometry-on-add* [*simp*]:
isometry-on A (λ*x*. *x* + (*t*::*'a*::*real-normed-vector*))
unfolding *isometry-on-def* **by** *auto*

lemma *isometry-on-minus* [*simp*]:
isometry-on A (λ(*x*::*'a*::*real-normed-vector*). −*x*)
unfolding *isometry-on-def* **by** (*auto simp add*: *dist-minus*)

lemma *isometry-on-diff* [*simp*]:
isometry-on A (λ*x*. (*t*::*'a*::*real-normed-vector*) − *x*)
unfolding *isometry-on-def* **by** (*auto*, *metis add-uminus-conv-diff dist-add-cancel dist-minus*)

lemma *isometry-preserves-bounded*:
  assumes *isometry-on X f*
  $A \subseteq X$
  shows *bounded* $(f`A) \longleftrightarrow$ *bounded A*
unfolding *bounded-two-points* **using** *assms(2) isometry-onD[OF assms(1)]* **by** *auto (metis assms(2) rev-subsetD)+*

lemma *isometry-preserves-infdist*:
  infdist $(f\ x)\ (f`A) =$ *infdist x A*
  if *isometry-on X f* $A \subseteq X$ $x \in X$
  using *that* **by** *(simp add: infdist-def image-comp isometry-on-def subset-iff)*

lemma *isometry-preserves-hausdorff-distance*:
  hausdorff-distance $(f`A)\ (f`B) =$ *hausdorff-distance A B*
  if *isometry-on X f* $A \subseteq X$ $B \subseteq X$
  using *that isometry-preserves-infdist [OF that(1) that(2)]*
  isometry-preserves-infdist [OF that(1) that(3)]
  isometry-preserves-bounded [OF that(1) that(2)]
  isometry-preserves-bounded [OF that(1) that(3)]
  by *(simp add: hausdorff-distance-def image-comp subset-eq)*

lemma *isometry-on-UNIV-iterates*:
  fixes f::('a::*metric-space*) $\Rightarrow$ 'a
  assumes *isometry-on UNIV f*
  shows *isometry-on UNIV* $(f\widehat{\ \ }\widehat{\ \ }n)$
by *(induction n, auto, rule isometry-on-compose[of - - f], auto intro: isometry-on-subset[OF assms])*

lemma *isometry-iterates*:
  fixes f::('a::*metric-space*) $\Rightarrow$ 'a
  assumes *isometry f*
  shows *isometry* $(f\widehat{\ \ }\widehat{\ \ }n)$
using *isometry-on-UNIV-iterates[OF isometryD(1)[OF assms], of n] bij-fn[OF isometry-inverse(2)[OF assms], of n]*
unfolding *isometry-def* **by** *(simp add: bij-is-surj)*

5 Geodesic spaces

A geodesic space is a metric space in which any pair of points can be joined by a geodesic segment, i.e., an isometrically embedded copy of a segment in the real line. Most spaces in geometry are geodesic. We introduce in this section the corresponding class of metric spaces. First, we study properties of general geodesic segments in metric spaces.

5.1 Geodesic segments in general metric spaces

definition *geodesic-segment-between*::('a::*metric-space*) *set* $\Rightarrow$ 'a $\Rightarrow$ 'a $\Rightarrow$ *bool*

where *geodesic-segment-between G x y* = (∃ *g*::(*real* ⇒ *'a*). *g 0* = *x* ∧ *g* (*dist x y*) = *y* ∧ *isometry-on* {*0..dist x y*} *g* ∧ *G* = *g*'{*0..dist x y*})

definition *geodesic-segment*::(*'a*::*metric-space*) *set* ⇒ *bool*
where *geodesic-segment G* = (∃ *x y*. *geodesic-segment-between G x y*)

We also introduce the parametrization of a geodesic segment. It is convenient to use the following definition, which guarantees that the point is on G even without checking that G is a geodesic segment or that the parameter is in the reasonable range: this shortens some arguments below.

definition *geodesic-segment-param*::(*'a*::*metric-space*) *set* ⇒ *'a* ⇒ *real* ⇒ *'a*
where *geodesic-segment-param G x t* = (*if* ∃ *w*. *w* ∈ *G* ∧ *dist x w* = *t then SOME w*. *w* ∈ *G* ∧ *dist x w* = *t else SOME w*. *w* ∈ *G*)

lemma *geodesic-segment-betweenI*:
assumes *g 0* = *x g* (*dist x y*) = *y isometry-on* {*0..dist x y*} *g G* = *g*'{*0..dist x y*}
shows *geodesic-segment-between G x y*
unfolding *geodesic-segment-between-def* **apply** (*rule exI*[*of - g*]) **using** *assms* **by** *auto*

lemma *geodesic-segmentI* [*intro*, *simp*]:
assumes *geodesic-segment-between G x y*
shows *geodesic-segment G*
unfolding *geodesic-segment-def* **using** *assms* **by** *auto*

lemma *geodesic-segmentI2* [*intro*]:
assumes *isometry-on* {*a..b*} *g a* ≤ (*b*::*real*)
shows *geodesic-segment-between* (*g*'{*a..b*}) (*g a*) (*g b*)
geodesic-segment (*g*'{*a..b*})
proof −
define *h* **where** *h* = (λ*t*. *g* (*t*+*a*))
have ∗: *isometry-on* {*0..b*−*a*} *h*
apply (*rule isometry-onI*)
using ‹*isometry-on* {*a..b*} *g*› ‹*a* ≤ *b*› **by** (*auto simp add*: *isometry-on-def h-def*)
have ∗∗: *dist* (*h 0*) (*h* (*b*−*a*)) = *b*−*a*
using *isometry-onD*[*OF* ‹*isometry-on* {*0..b*−*a*} *h*›, *of 0 b*−*a*] ‹*a* ≤ *b*› **unfolding** *dist-real-def* **by** *auto*
have *geodesic-segment-between* (*h*'{*0..b*−*a*}) (*h 0*) (*h* (*b*−*a*))
unfolding *geodesic-segment-between-def* **apply** (*rule exI*[*of - h*]) **unfolding** ∗∗ **using** ∗ **by** *auto*
moreover have *g*'{*a..b*} = *h*'{*0..b*−*a*}
unfolding *h-def* **apply** (*auto simp add*: *image-iff*)
by (*metis add.commute atLeastAtMost-iff diff-ge-0-iff-ge diff-right-mono le-add-diff-inverse*)
moreover have *h 0* = *g a h* (*b*−*a*) = *g b* **unfolding** *h-def* **by** *auto*
ultimately show *geodesic-segment-between* (*g*'{*a..b*}) (*g a*) (*g b*) **by** *auto*
then show *geodesic-segment* (*g*'{*a..b*}) **unfolding** *geodesic-segment-def* **by** *auto*
qed

lemma *geodesic-segmentD*:
assumes *geodesic-segment-between G x y*
shows $\exists$ *g*::(*real* $\Rightarrow$ -). (*g t* = *x* $\wedge$ *g* (*t* + *dist x y*) = *y* $\wedge$ *isometry-on* {*t*..*t*+*dist x y*} *g* $\wedge$ *G* = *g*'{*t*..*t*+*dist x y*})
proof −
obtain *h* **where** *h*: *h 0* = *x h* (*dist x y*) = *y isometry-on* {*0*..*dist x y*} *h G* = *h*'{*0*..*dist x y*}
by (*meson* ‹*geodesic-segment-between G x y*› *geodesic-segment-between-def*)
have ∗ [*simp*]: (λ*x*. *x* − *t*) ' {*t*..*t* + *dist x y*} = {*0*..*dist x y*} **by** *auto*
define *g* **where** *g* = (λ*s*. *h* (*s* − *t*))
have *g t* = *x g* (*t* + *dist x y*) = *y* **using** *h assms*(*1*) **unfolding** *g-def* **by** *auto*
moreover have *isometry-on* {*t*..*t*+*dist x y*} *g*
unfolding *g-def* **apply** (*rule isometry-on-compose*[*of* - - *h*])
by (*simp add*: *dist-real-def isometry-on-def*, *simp add*: *h*(*3*))
moreover have *g*' {*t*..*t* + *dist x y*} = *G* **unfolding** *g-def h*(*4*) **using** ∗ **by** (*metis image-image*)
ultimately show *?thesis* **by** *auto*
qed

lemma *geodesic-segment-endpoints* [*simp*]:
assumes *geodesic-segment-between G x y*
shows *x* $\in$ *G y* $\in$ *G G* $\neq$ {}
using *assms* **unfolding** *geodesic-segment-between-def*
by (*auto*, *metis atLeastAtMost-iff image-eqI less-eq-real-def zero-le-dist*)

lemma *geodesic-segment-commute*:
assumes *geodesic-segment-between G x y*
shows *geodesic-segment-between G y x*
proof −
obtain *g*::*real*$\Rightarrow$'*a* **where** *g*: *g 0* = *x g* (*dist x y*) = *y isometry-on* {*0*..*dist x y*} *g G* = *g*'{*0*..*dist x y*}
by (*meson* ‹*geodesic-segment-between G x y*› *geodesic-segment-between-def*)
define *h*::*real*$\Rightarrow$'*a* **where** *h* = (λ*t*. *g*(*dist x y*−*t*))
have (λ*t*. *dist x y* −*t*)'{*0*..*dist x y*} = {*0*..*dist x y*} **by** *auto*
then have *h*'{*0*..*dist x y*} = *G* **unfolding** *g*(*4*) *h-def* **by** (*metis image-image*)
moreover have *h 0* = *y h* (*dist x y*) = *x* **unfolding** *h-def* **using** *g* **by** *auto*
moreover have *isometry-on* {*0*..*dist x y*} *h*
unfolding *h-def* **apply** (*rule isometry-on-compose*[*of* - - *g*]) **using** *g*(*3*) **by** *auto*
ultimately show *?thesis*
unfolding *geodesic-segment-between-def* **by** (*auto simp add*: *dist-commute*)
qed

lemma *geodesic-segment-dist*:
assumes *geodesic-segment-between G x y a* $\in$ *G*
shows *dist x a* + *dist a y* = *dist x y*
proof −
obtain *g* **where** *g*: *g 0* = *x g* (*dist x y*) = *y isometry-on* {*0*..*dist x y*} *g G* = *g*'{*0*..*dist x y*}

by (*meson* ‹*geodesic-segment-between G x y*› *geodesic-segment-between-def*)
obtain *t* **where** *t*: *t* ∈ {*0..dist x y*} *a* = *g t*
using *g*(*4*) *assms* **by** *auto*
have *dist x a* = *t* **using** *isometry-onD*[*OF g*(*3*) - *t*(*1*), *of 0*]
unfolding *g*(*1*) *dist-real-def t*(*2*) **using** *t*(*1*) **by** *auto*
moreover have *dist a y* = *dist x y* − *t* **using** *isometry-onD*[*OF g*(*3*) - *t*(*1*), *of dist x y*]
unfolding *g*(*2*) *dist-real-def t*(*2*) **using** *t*(*1*) **by** (*auto simp add*: *dist-commute*)
ultimately show *?thesis* **by** *auto*
qed

lemma *geodesic-segment-dist-unique*:
assumes *geodesic-segment-between G x y a* ∈ *G b* ∈ *G dist x a* = *dist x b*
shows *a* = *b*
proof −
obtain *g* **where** *g*: *g 0* = *x g* (*dist x y*) = *y isometry-on* {*0..dist x y*} *g G* = *g*‘{*0..dist x y*}
by (*meson* ‹*geodesic-segment-between G x y*› *geodesic-segment-between-def*)
obtain *ta* **where** *ta*: *ta* ∈ {*0..dist x y*} *a* = *g ta*
using *g*(*4*) *assms* **by** *auto*
have ∗: *dist x a* = *ta*
unfolding *g*(*1*)[*symmetric*] *ta*(*2*) **using** *isometry-onD*[*OF g*(*3*), *of 0 ta*]
unfolding *dist-real-def* **using** *ta*(*1*) **by** *auto*
obtain *tb* **where** *tb*: *tb* ∈ {*0..dist x y*} *b* = *g tb*
using *g*(*4*) *assms* **by** *auto*
have *dist x b* = *tb*
unfolding *g*(*1*)[*symmetric*] *tb*(*2*) **using** *isometry-onD*[*OF g*(*3*), *of 0 tb*]
unfolding *dist-real-def* **using** *tb*(*1*) **by** *auto*
then have *ta* = *tb* **using** ∗ ‹*dist x a* = *dist x b*› **by** *auto*
then show *a* = *b* **using** *ta*(*2*) *tb*(*2*) **by** *auto*
qed

lemma *geodesic-segment-union*:
assumes *dist x z* = *dist x y* + *dist y z*
geodesic-segment-between G x y geodesic-segment-between H y z
shows *geodesic-segment-between* (*G* ∪ *H*) *x z*
G ∩ *H* = {*y*}
proof −
obtain *g* **where** *g*: *g 0* = *x g* (*dist x y*) = *y isometry-on* {*0..dist x y*} *g G* = *g*‘{*0..dist x y*}
by (*meson* ‹*geodesic-segment-between G x y*› *geodesic-segment-between-def*)
obtain *h* **where** *h*: *h* (*dist x y*) = *y h* (*dist x z*) = *z isometry-on* {*dist x y..dist x z*} *h H* = *h*‘{*dist x y..dist x z*}
unfolding ‹*dist x z* = *dist x y* + *dist y z*›
using *geodesic-segmentD*[*OF* ‹*geodesic-segment-between H y z*›, *of dist x y*] **by** *auto*
define *f* **where** *f* = (λ*t*. *if t* ≤ *dist x y then g t else h t*)
have *fg*: *f t* = *g t* **if** *t* ≤ *dist x y* **for** *t*
unfolding *f-def* **using** *that* **by** *auto*

have *fh*: $f\ t = h\ t$ **if** $t \geq dist\ x\ y$ **for** t
unfolding *f-def* **apply** (*cases* $t > dist\ x\ y$) **using** *that* $g(2)\ h(1)$ **by** *auto*

have $f\ 0 = x\ f\ (dist\ x\ z) = z$ **using** *fg fh* $g(1)\ h(2)$ *assms*(1) **by** *auto*

have $f`\{0..dist\ x\ z\} = f`\{0..dist\ x\ y\} \cup f`\{dist\ x\ y..dist\ x\ z\}$
unfolding *assms*(1) *image-Un*[*symmetric*] **by** (*simp add*: *ivl-disj-un-two-touch*(4))
moreover have $f`\{0..dist\ x\ y\} = G$
unfolding $g(4)$ **using** *fg image-cong* **by** *force*
moreover have $f`\{dist\ x\ y..dist\ x\ z\} = H$
unfolding $h(4)$ **using** *fh image-cong* **by** *force*
ultimately have $f`\{0..dist\ x\ z\} = G \cup H$ **by** *simp*

have *Ifg*: $dist\ (f\ s)\ (f\ t) = s-t$ **if** $0 \leq t\ t \leq s\ s \leq dist\ x\ y$ **for** $s\ t$
using *that fg*[*of s*] *fg*[*of t*] *isometry-onD*[*OF* $g(3)$, *of s t*] **unfolding** *dist-real-def* **by** *auto*
have *Ifh*: $dist\ (f\ s)\ (f\ t) = s-t$ **if** $dist\ x\ y \leq t\ t \leq s\ s \leq dist\ x\ z$ **for** $s\ t$
using *that fh*[*of s*] *fh*[*of t*] *isometry-onD*[*OF* $h(3)$, *of s t*] **unfolding** *dist-real-def* **by** *auto*

have *I*: $dist\ (f\ s)\ (f\ t) = s-t$ **if** $0 \leq t\ t \leq s\ s \leq dist\ x\ z$ **for** $s\ t$
proof −
consider $t \leq dist\ x\ y \wedge s \geq dist\ x\ y \mid s \leq dist\ x\ y \mid t \geq dist\ x\ y$ **by** *fastforce*
then show *?thesis*
proof (*cases*)
case *1*
have $dist\ (f\ t)\ (f\ s) \leq dist\ (f\ t)\ (f\ (dist\ x\ y)) + dist\ (f\ (dist\ x\ y))\ (f\ s)$
using *dist-triangle* **by** *auto*
also have ... $\leq (dist\ x\ y - t) + (s - dist\ x\ y)$
using *that 1 Ifg*[*of t dist x y*] *Ifh*[*of dist x y s*] **by** (*auto simp add*: *dist-commute intro*: *mono-intros*)
finally have $*$: $dist\ (f\ t)\ (f\ s) \leq s - t$ **by** *simp*

have $dist\ x\ z \leq dist\ (f\ 0)\ (f\ t) + dist\ (f\ t)\ (f\ s) + dist\ (f\ s)\ (f\ (dist\ x\ z))$
unfolding ‹$f\ 0 = x$› ‹$f\ (dist\ x\ z) = z$› **using** *dist-triangle4* **by** *auto*
also have ... $\leq t + dist\ (f\ t)\ (f\ s) + (dist\ x\ z - s)$
using *that 1 Ifg*[*of 0 t*] *Ifh*[*of s dist x z*] **by** (*auto simp add*: *dist-commute intro*: *mono-intros*)
finally have $s - t \leq dist\ (f\ t)\ (f\ s)$ **by** *auto*
then show $dist\ (f\ s)\ (f\ t) = s-t$ **using** $*$ *dist-commute* **by** *auto*
next
case *2*
then show *?thesis* **using** *Ifg that* **by** *auto*
next
case *3*
then show *?thesis* **using** *Ifh that* **by** *auto*
qed
qed
have *isometry-on* $\{0..dist\ x\ z\}\ f$

unfolding *isometry-on-def dist-real-def* **using** *I*
by (*auto*, *metis abs-of-nonneg dist-commute dist-real-def le-cases zero-le-dist*)
then show *geodesic-segment-between* $(G \cup H)$ *x z*
unfolding *geodesic-segment-between-def*
using ‹$f\ 0 = x$› ‹$f\ (dist\ x\ z) = z$› ‹$f`\{0..dist\ x\ z\} = G \cup H$› **by** *auto*
have $G \cap H \subseteq \{y\}$
proof (*auto*)
fix *a* **assume** *a*: $a \in G$ $a \in H$
obtain *s* **where** *s*: $s \in \{0..dist\ x\ y\}$ $a = g\ s$ **using** *a g*(*4*) **by** *auto*
obtain *t* **where** *t*: $t \in \{dist\ x\ y..dist\ x\ z\}$ $a = h\ t$ **using** *a h*(*4*) **by** *auto*
have $a = f\ s$ **using** *fg s* **by** *auto*
moreover have $a = f\ t$ **using** *fh t* **by** *auto*
ultimately have $s = t$ **using** *isometry-onD*[*OF* ‹*isometry-on* $\{0..dist\ x\ z\}$ *f*›, *of s t*] *s*(*1*) *t*(*1*) **by** *auto*
then have $s = dist\ x\ y$ **using** *s t* **by** *auto*
then show $a = y$ **using** *s*(*2*) *g* **by** *auto*
qed
then show $G \cap H = \{y\}$ **using** *assms* **by** *auto*
qed

lemma *geodesic-segment-dist-le*:
assumes *geodesic-segment-between G x y* $a \in G$ $b \in G$
shows $dist\ a\ b \leq dist\ x\ y$
proof –
obtain *g* **where** *g*: $g\ 0 = x$ $g\ (dist\ x\ y) = y$ *isometry-on* $\{0..dist\ x\ y\}$ *g* $G = g`\{0..dist\ x\ y\}$
by (*meson* ‹*geodesic-segment-between G x y*› *geodesic-segment-between-def*)
obtain *s t* **where** *st*: $s \in \{0..dist\ x\ y\}$ $t \in \{0..dist\ x\ y\}$ $a = g\ s$ $b = g\ t$
using *g*(*4*) *assms* **by** *auto*
have $dist\ a\ b = abs(s-t)$ **using** *isometry-onD*[*OF g*(*3*) *st*(*1*) *st*(*2*)]
unfolding *st*(*3*) *st*(*4*) *dist-real-def* **by** *simp*
then show $dist\ a\ b \leq dist\ x\ y$ **using** *st*(*1*) *st*(*2*) **unfolding** *dist-real-def* **by** *auto*
qed

lemma *geodesic-segment-param* [*simp*]:
assumes *geodesic-segment-between G x y*
shows *geodesic-segment-param G x* $0 = x$
geodesic-segment-param G x $(dist\ x\ y) = y$
$t \in \{0..dist\ x\ y\} \Longrightarrow$ *geodesic-segment-param G x t* $\in G$
isometry-on $\{0..dist\ x\ y\}$ (*geodesic-segment-param G x*)
(*geodesic-segment-param G x*)$`\{0..dist\ x\ y\} = G$
$t \in \{0..dist\ x\ y\} \Longrightarrow dist\ x$ (*geodesic-segment-param G x t*) $= t$
$s \in \{0..dist\ x\ y\} \Longrightarrow t \in \{0..dist\ x\ y\} \Longrightarrow dist$ (*geodesic-segment-param G x s*) (*geodesic-segment-param G x t*) $= abs(s-t)$
$z \in G \Longrightarrow z =$ *geodesic-segment-param G x* $(dist\ x\ z)$
proof –
obtain $g{::}real \Rightarrow 'a$ **where** *g*: $g\ 0 = x$ $g\ (dist\ x\ y) = y$ *isometry-on* $\{0..dist\ x\ y\}$ *g* $G = g`\{0..dist\ x\ y\}$
by (*meson* ‹*geodesic-segment-between G x y*› *geodesic-segment-between-def*)

have ∗: *g t* ∈ *G* ∧ *dist x* (*g t*) = *t* **if** *t* ∈ {*0..dist x y*} **for** *t*
using *isometry-onD*[*OF g*(*3*), *of 0 t*] *that g*(*1*) *g*(*4*) **unfolding** *dist-real-def* **by** *auto*
have *G*: *geodesic-segment-param G x t* = *g t* **if** *t* ∈ {*0..dist x y*} **for** *t*
proof −
have *A*: *geodesic-segment-param G x t* ∈ *G* ∧ *dist x* (*geodesic-segment-param G x t*) = *t*
using ∗[*OF that*] **unfolding** *geodesic-segment-param-def* **apply** *auto*
using ∗[*OF that*] **by** (*metis* (*mono-tags*, *lifting*) *someI*)+
obtain *s* **where** *s*: *geodesic-segment-param G x t* = *g s s* ∈ {*0..dist x y*}
using *A g*(*4*) **by** *auto*
have *s* = *t* **using** ∗[*OF* ‹*s* ∈ {*0..dist x y*}›] *A* **unfolding** *s*(*1*) **by** *auto*
then show *?thesis* **using** *s* **by** *auto*
qed
show *geodesic-segment-param G x 0* = *x*
geodesic-segment-param G x (*dist x y*) = *y*
t ∈ {*0..dist x y*} ⟹ *geodesic-segment-param G x t* ∈ *G*
isometry-on {*0..dist x y*} (*geodesic-segment-param G x*)
(*geodesic-segment-param G x*)‘{*0..dist x y*} = *G*
t ∈ {*0..dist x y*} ⟹ *dist x* (*geodesic-segment-param G x t*) = *t*
s ∈ {*0..dist x y*} ⟹ *t* ∈ {*0..dist x y*} ⟹ *dist* (*geodesic-segment-param G x s*) (*geodesic-segment-param G x t*) = *abs*(*s*−*t*)
z ∈ *G* ⟹ *z* = *geodesic-segment-param G x* (*dist x z*)
using *G g* **apply** (*auto simp add*: *rev-image-eqI*)
using *G isometry-on-cong* ∗ *atLeastAtMost-iff* **apply** *blast*
using *G isometry-on-cong* ∗ *atLeastAtMost-iff* **apply** *blast*
by (*auto simp add*: ∗ *dist-real-def isometry-onD*)
qed

lemma *geodesic-segment-param-in-segment*:
assumes *G* ≠ {}
shows *geodesic-segment-param G x t* ∈ *G*
unfolding *geodesic-segment-param-def*
apply (*auto*, *metis* (*mono-tags*, *lifting*) *someI*)
using *assms some-in-eq* **by** *fastforce*

lemma *geodesic-segment-reverse-param*:
assumes *geodesic-segment-between G x y*
t ∈ {*0..dist x y*}
shows *geodesic-segment-param G y* (*dist x y* − *t*) = *geodesic-segment-param G x t*
proof −
have ∗ [*simp*]: *geodesic-segment-between G y x*
using *geodesic-segment-commute*[*OF assms*(*1*)] **by** *simp*
have *geodesic-segment-param G y* (*dist x y* − *t*) ∈ *G*
apply (*rule geodesic-segment-param*(*3*)[*of - - x*])
using *assms*(*2*) **by** (*auto simp add*: *dist-commute*)
moreover have *dist* (*geodesic-segment-param G y* (*dist x y* − *t*)) *x* = *t*
using *geodesic-segment-param*(*2*)[*OF* ∗] *geodesic-segment-param*(*7*)[*OF* ∗, *of*

dist x y −t dist x y] *assms*(*2*) **by** (*auto simp add*: *dist-commute*)
moreover have *geodesic-segment-param G x t* ∈ *G*
apply (*rule geodesic-segment-param*(*3*)[*OF assms*(*1*)])
using *assms*(*2*) **by** *auto*
moreover have *dist* (*geodesic-segment-param G x t*) *x* = *t*
using *geodesic-segment-param*(*6*)[*OF assms*] **by** (*simp add*: *dist-commute*)
ultimately show *?thesis*
using *geodesic-segment-dist-unique*[*OF assms*(*1*)] **by** (*simp add*: *dist-commute*)
qed

lemma *dist-along-geodesic-wrt-endpoint*:
assumes *geodesic-segment-between G x y*
u ∈ *G v* ∈ *G*
shows *dist u v* = *abs*(*dist u x* − *dist v x*)
proof −
have ∗: *u* = *geodesic-segment-param G x* (*dist x u*) *v* = *geodesic-segment-param G x* (*dist x v*)
using *assms* **by** *auto*
have *dist u v* = *dist* (*geodesic-segment-param G x* (*dist x u*)) (*geodesic-segment-param G x* (*dist x v*))
using ∗ **by** *auto*
also have ... = *abs*(*dist x u* − *dist x v*)
apply (*rule geodesic-segment-param*(*7*)[*OF assms*(*1*)]) **using** *assms* **apply** *auto*
using *geodesic-segment-dist-le geodesic-segment-endpoints*(*1*) **by** *blast+*
finally show *?thesis* **by** (*simp add*: *dist-commute*)
qed

One often needs to restrict a geodesic segment to a subsegment. We introduce the tools to express this conveniently.

definition *geodesic-subsegment*::(*'a*::*metric-space*) *set* ⇒ *'a* ⇒ *real* ⇒ *real* ⇒ *'a set*
where *geodesic-subsegment G x s t* = *G* ∩ {*z*. *dist x z* ≥ *s* ∧ *dist x z* ≤ *t*}

A subsegment is always contained in the original segment.

lemma *geodesic-subsegment-subset*:
geodesic-subsegment G x s t ⊆ *G*
unfolding *geodesic-subsegment-def* **by** *simp*

A subsegment is indeed a geodesic segment, and its endpoints and parametrization can be expressed in terms of the original segment.

lemma *geodesic-subsegment*:
assumes *geodesic-segment-between G x y*
0 ≤ *s s* ≤ *t t* ≤ *dist x y*
shows *geodesic-subsegment G x s t* = (*geodesic-segment-param G x*)‘{*s*..*t*}
geodesic-segment-between (*geodesic-subsegment G x s t*) (*geodesic-segment-param G x s*) (*geodesic-segment-param G x t*)
⋀*u*. *s* ≤ *u* ⟹ *u* ≤ *t* ⟹ *geodesic-segment-param* (*geodesic-subsegment G x s t*) (*geodesic-segment-param G x s*) (*u* − *s*) = *geodesic-segment-param G x u*

```
proof −
  show A: geodesic-subsegment G x s t = (geodesic-segment-param G x)‘{s..t}
  proof (auto)
    fix y assume y: y ∈ geodesic-subsegment G x s t
    have y = geodesic-segment-param G x (dist x y)
      apply (rule geodesic-segment-param(8)[OF assms(1)])
      using y geodesic-subsegment-subset by force
    moreover have dist x y ≥ s ∧ dist x y ≤ t
      using y unfolding geodesic-subsegment-def by auto
    ultimately show y ∈ geodesic-segment-param G x ‘ {s..t} by auto
  next
    fix u assume H: s ≤ u u ≤ t
    have ∗: dist x (geodesic-segment-param G x u) = u
       apply (rule geodesic-segment-param(6)[OF assms(1)]) using H assms by
auto
    show geodesic-segment-param G x u ∈ geodesic-subsegment G x s t
      unfolding geodesic-subsegment-def
     using geodesic-segment-param-in-segment[OF geodesic-segment-endpoints(3)[OF
assms(1)]] by (auto simp add: ∗ H)
  qed

  have ∗: isometry-on {s..t} (geodesic-segment-param G x)
    by (rule isometry-on-subset[of {0..dist x y}]) (auto simp add: assms)
 show B: geodesic-segment-between (geodesic-subsegment G x s t) (geodesic-segment-param
G x s) (geodesic-segment-param G x t)
    unfolding A apply (rule geodesic-segmentI2) using ∗ assms by auto

  fix u assume u: s ≤ u u ≤ t
 show geodesic-segment-param (geodesic-subsegment G x s t) (geodesic-segment-param
G x s) (u − s) = geodesic-segment-param G x u
  proof (rule geodesic-segment-dist-unique[OF B])
   show geodesic-segment-param (geodesic-subsegment G x s t) (geodesic-segment-param
G x s) (u − s) ∈ geodesic-subsegment G x s t
     by (rule geodesic-segment-param-in-segment[OF geodesic-segment-endpoints(3)[OF
B]])
    show geodesic-segment-param G x u ∈ geodesic-subsegment G x s t
      unfolding A using u by auto
   have dist (geodesic-segment-param G x s) (geodesic-segment-param (geodesic-subsegment
G x s t) (geodesic-segment-param G x s) (u − s)) = u − s
      using B assms u by auto
    moreover have dist (geodesic-segment-param G x s) (geodesic-segment-param
G x u) = u −s
      using assms u by auto
   ultimately show dist (geodesic-segment-param G x s) (geodesic-segment-param
(geodesic-subsegment G x s t) (geodesic-segment-param G x s) (u − s)) =
        dist (geodesic-segment-param G x s) (geodesic-segment-param G x u)
      by simp
  qed
qed
```

The parameterizations of a segment and a subsegment sharing an endpoint coincide where defined.

lemma *geodesic-segment-subparam*:
assumes *geodesic-segment-between G x z geodesic-segment-between H x y* $H \subseteq G$ $t \in \{0..dist\ x\ y\}$
shows *geodesic-segment-param G x t = geodesic-segment-param H x t*
proof −
have *geodesic-segment-param H x t* $\in G$
using *assms*(*3*) *geodesic-segment-param*(*3*)[*OF assms*(*2*) *assms*(*4*)] **by** *auto*
then have *geodesic-segment-param H x t = geodesic-segment-param G x* (*dist x* (*geodesic-segment-param H x t*))
using *geodesic-segment-param*(*8*)[*OF assms*(*1*)] **by** *auto*
then show *?thesis* **using** *geodesic-segment-param*(*6*)[*OF assms*(*2*) *assms*(*4*)] **by** *auto*
qed

A segment contains a subsegment between any of its points

lemma *geodesic-subsegment-exists*:
assumes *geodesic-segment G* $x \in G$ $y \in G$
shows $\exists H.\ H \subseteq G \wedge$ *geodesic-segment-between H x y*
proof −
obtain *a0 b0* **where** *Ga0b0*: *geodesic-segment-between G a0 b0*
using *assms*(*1*) **unfolding** *geodesic-segment-def* **by** *auto*

Permuting the endpoints if necessary, we can ensure that the first endpoint a is closer to x than y.

have $\exists$ *a b. geodesic-segment-between G a b* $\wedge$ *dist x a* $\leq$ *dist y a*
proof (*cases dist x a0* $\leq$ *dist y a0*)
case *True*
show *?thesis*
apply (*rule exI*[*of - a0*], *rule exI*[*of - b0*]) **using** *True Ga0b0* **by** *auto*
next
case *False*
show *?thesis*
apply (*rule exI*[*of - b0*], *rule exI*[*of - a0*])
using *Ga0b0 geodesic-segment-commute geodesic-segment-dist*[*OF Ga0b0* ‹$x \in G$›] *geodesic-segment-dist*[*OF Ga0b0* ‹$y \in G$›] *False*
by (*auto simp add*: *dist-commute*)
qed
then obtain *a b* **where** *Gab*: *geodesic-segment-between G a b dist x a* $\leq$ *dist y a*
by *auto*
have $*$: $0 \leq$ *dist x a dist x a* $\leq$ *dist y a dist y a* $\leq$ *dist a b*
using *Gab assms* **by** (*meson geodesic-segment-dist-le geodesic-segment-endpoints*(*1*) *zero-le-dist*)+
have $**$: $x =$ *geodesic-segment-param G a* (*dist x a*) $y =$ *geodesic-segment-param G a* (*dist y a*)
using *Gab* ‹$x \in G$› ‹$y \in G$› **by** (*metis dist-commute geodesic-segment-param*(*8*))+
define *H* **where** $H =$ *geodesic-subsegment G a* (*dist x a*) (*dist y a*)

have *H* ⊆ *G*
unfolding *H-def* **by** (*rule geodesic-subsegment-subset*)
moreover have *geodesic-segment-between H x y*
unfolding *H-def* **using** *geodesic-subsegment*(*2*)[*OF Gab*(*1*) ∗] ∗∗ **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

A geodesic segment is homeomorphic to an interval.

lemma *geodesic-segment-homeo-interval*:
assumes *geodesic-segment-between G x y*
shows {*0..dist x y*} *homeomorphic G*
proof −
obtain *g* **where** *g*: *g 0* = *x g* (*dist x y*) = *y isometry-on* {*0..dist x y*} *g G* = *g*'{*0..dist x y*}
by (*meson* ‹*geodesic-segment-between G x y*› *geodesic-segment-between-def*)
show *?thesis* **using** *isometry-on-homeomorphism*(*3*)[*OF g*(*3*)] **unfolding** *g*(*4*) **by** *simp*
qed

Just like an interval, a geodesic segment is compact, connected, path connected, bounded, closed, nonempty, and proper.

lemma *geodesic-segment-topology*:
assumes *geodesic-segment G*
shows *compact G connected G path-connected G bounded G closed G G* ≠ {} *proper G*
proof −
show *compact G*
using *assms geodesic-segment-homeo-interval homeomorphic-compactness*
unfolding *geodesic-segment-def* **by** *force*
show *path-connected G*
using *assms is-interval-path-connected geodesic-segment-homeo-interval homeomorphic-path-connectedness*
unfolding *geodesic-segment-def*
by (*metis is-interval-cc*)
then show *connected G*
using *path-connected-imp-connected* **by** *auto*
show *bounded G*
by (*rule compact-imp-bounded*, *fact*)
show *closed G*
by (*rule compact-imp-closed*, *fact*)
show *G* ≠ {}
using *assms geodesic-segment-def geodesic-segment-endpoints*(*3*) **by** *auto*
show *proper G*
using *proper-of-compact* ‹*compact G*› **by** *auto*
qed

lemma *geodesic-segment-between-x-x* [*simp*]:
geodesic-segment-between {*x*} *x x*
geodesic-segment {*x*}

geodesic-segment-between G x x $\longleftrightarrow$ *G* = {*x*}
proof −
show ∗: *geodesic-segment-between* {*x*} *x x*
unfolding *geodesic-segment-between-def* **apply** (*rule exI*[*of* - λ-. *x*]) **unfolding** *isometry-on-def* **by** *auto*
then show *geodesic-segment* {*x*} **by** *auto*
show *geodesic-segment-between G x x* $\longleftrightarrow$ *G* = {*x*}
using *geodesic-segment-dist-le geodesic-segment-endpoints*(*2*) ∗ **by** *fastforce*
qed

lemma *geodesic-segment-disconnection*:
assumes *geodesic-segment-between G x y z* ∈ *G*
shows (*connected* (*G* − {*z*})) = (*z* = *x* ∨ *z* = *y*)
proof −
obtain *g* **where** *g*: *g 0* = *x g* (*dist x y*) = *y isometry-on* {*0*..*dist x y*} *g G* = *g*‘{*0*..*dist x y*}
by (*meson* ‹*geodesic-segment-between G x y*› *geodesic-segment-between-def*)
obtain *t* **where** *t*: *t* ∈ {*0*..*dist x y*} *z* = *g t* **using** ‹*z* ∈ *G*› *g*(*4*) **by** *auto*
have ({*0*..*dist x y*} − {*t*}) *homeomorphic* (*G* − {*g t*})
proof −
have ∗: *isometry-on* ({*0*..*dist x y*} − {*t*}) *g*
apply (*rule isometry-on-subset*[*OF g*(*3*)]) **by** *auto*
have ({*0*..*dist x y*} − {*t*}) *homeomorphic g*‘({*0*..*dist x y*} − {*t*})
by (*rule isometry-on-homeomorphism*(*3*)[*OF* ∗])
moreover have *g*‘({*0*..*dist x y*} − {*t*}) = *G* − {*g t*}
unfolding *g*(*4*) **using** *isometry-on-injective*[*OF g*(*3*)] *t* **by** (*auto simp add*: *inj-onD*)
ultimately show *?thesis* **by** *auto*
qed
moreover have *connected*({*0*..*dist x y*} − {*t*}) = (*t* = *0* ∨ *t* = *dist x y*)
using *t*(*1*) **by** (*auto simp add*: *connected-iff-interval*, *fastforce*)
ultimately have *connected* (*G* − {*z*}) = (*t* = *0* ∨ *t* = *dist x y*)
unfolding ‹*z* = *g t*›[*symmetric*]**using** *homeomorphic-connectedness* **by** *blast*
moreover have (*t* = *0* ∨ *t* = *dist x y*) = (*z* = *x* ∨ *z* = *y*)
using *t g* **apply** *auto*
by (*metis atLeastAtMost-iff isometry-on-inverse*(*2*) *order-refl zero-le-dist*)+
ultimately show *?thesis* **by** *auto*
qed

lemma *geodesic-segment-unique-endpoints*:
assumes *geodesic-segment-between G x y*
geodesic-segment-between G a b
shows {*x*, *y*} = {*a*, *b*}
by (*metis geodesic-segment-disconnection assms*(*1*) *assms*(*2*) *doubleton-eq-iff geodesic-segment-endpoints*(*1*) *geodesic-segment-endpoints*(*2*))

lemma *geodesic-segment-subsegment*:
assumes *geodesic-segment G H* ⊆ *G compact H connected H H* ≠ {}
shows *geodesic-segment H*

proof −
  obtain *x y* **where** *geodesic-segment-between G x y*
    using *assms* **unfolding** *geodesic-segment-def* **by** *auto*
  obtain *g* **where** *g*: *g 0 = x g (dist x y) = y isometry-on {0..dist x y} g G = g‘{0..dist x y}*
    by (*meson ‹geodesic-segment-between G x y› geodesic-segment-between-def*)
  define *L* **where** *L = (inv-into {0..dist x y} g)‘H*
  have *L ⊆ {0..dist x y}*
    unfolding *L-def* **using** *isometry-on-inverse*[*OF ‹isometry-on {0..dist x y} g›*] *assms*(*2*) *g*(*4*) **by** *auto*
  have *isometry-on G (inv-into {0..dist x y} g)*
    using *isometry-on-inverse*[*OF ‹isometry-on {0..dist x y} g›*] *g*(*4*) **by** *auto*
  then have *isometry-on H (inv-into {0..dist x y} g)*
    using *‹H ⊆ G› isometry-on-subset* **by** *auto*
  then have *H homeomorphic L* **unfolding** *L-def* **using** *isometry-on-homeomorphism*(*3*) **by** *auto*
  then have *compact L ∧ connected L*
    using *assms homeomorphic-compactness homeomorphic-connectedness* **by** *blast*
  then obtain *a b* **where** *L = {a..b}*
    using *connected-compact-interval-1*[*of L*] **by** *auto*
  have *a ≤ b* **using** *‹H ≠ {}› ‹L = {a..b}›* **unfolding** *L-def* **by** *auto*
  then have *0 ≤ a b ≤ dist x y* **using** *‹L ⊆ {0..dist x y}› ‹L = {a..b}›* **by** *auto*
  have ∗: *H = g‘{a..b}*
    by (*metis L-def ‹L = {a..b}› assms*(*2*) *g*(*4*) *image-inv-into-cancel*)
  show *geodesic-segment H*
    unfolding ∗ **apply** (*rule geodesic-segmentI2*[*OF - ‹a ≤ b›*])
    apply (*rule isometry-on-subset*[*OF g*(*3*)]) **using** *‹0 ≤ a› ‹b ≤ dist x y›* **by** *auto*
qed

The image under an isometry of a geodesic segment is still obviously a geodesic segment.

lemma *isometry-preserves-geodesic-segment-between*:
  assumes *isometry-on X f*
      G ⊆ X geodesic-segment-between G x y
  shows *geodesic-segment-between (f‘G) (f x) (f y)*
proof −
  obtain *g* **where** *g*: *g 0 = x g (dist x y) = y isometry-on {0..dist x y} g G = g‘{0..dist x y}*
    by (*meson ‹geodesic-segment-between G x y› geodesic-segment-between-def*)
  then have ∗: *f‘G = (f o g) ‘{0..dist x y} f x = (f o g) 0 f y = (f o g) (dist x y)*
    by *auto*
  show *?thesis*
    unfolding ∗ **apply** (*intro geodesic-segmentI2*(*1*))
    unfolding *comp-def* **apply** (*rule isometry-on-compose*[*of - g*])
    using *g*(*3*) *g*(*4*) *assms* **by** (*auto intro*: *isometry-on-subset*)
qed

The sum of distances $d(w, x) + d(w, y)$ can be controlled using the distance from w to a geodesic segment between x and y.

```
lemma geodesic-segment-distance:
  assumes geodesic-segment-between G x y
  shows dist w x + dist w y ≤ dist x y + 2 ∗ infdist w G
proof −
  have ∃ z ∈ G. infdist w G = dist w z
   apply (rule infdist-proper-attained) using assms by (auto simp add: geodesic-segment-topology)
  then obtain z where z: z ∈ G infdist w G = dist w z by auto
  have dist w x + dist w y ≤ (dist w z + dist z x) + (dist w z + dist z y)
    by (intro mono-intros)
  also have ... = dist x z + dist z y + 2 ∗ dist w z
    by (auto simp add: dist-commute)
  also have ... = dist x y + 2 ∗ infdist w G
    using z(1) assms geodesic-segment-dist unfolding z(2) by auto
  finally show ?thesis by auto
qed
```

If a point y is on a geodesic segment between x and its closest projection p on a set A, then p is also a closest projection of y, and the closest projection set of y is contained in that of x.

```
lemma proj-set-geodesic-same-basepoint:
  assumes p ∈ proj-set x A geodesic-segment-between G p x y ∈ G
  shows p ∈ proj-set y A
proof (rule proj-setI)
  show p ∈ A
    using assms proj-setD by auto
  have ∗: dist y p ≤ dist y q if q ∈ A for q
  proof −
    have dist p y + dist y x = dist p x
      using assms geodesic-segment-dist by blast
    also have ... ≤ dist q x
      using proj-set-dist-le[OF ‹q ∈ A› assms(1)] by (simp add: dist-commute)
    also have ... ≤ dist q y + dist y x
      by (intro mono-intros)
    finally show ?thesis
      by (simp add: dist-commute)
  qed
  have dist y p ≤ Inf (dist y ‘ A)
    apply (rule cINF-greatest) using ‹p ∈ A› ∗ by auto
  then show dist y p ≤ infdist y A
    unfolding infdist-def using ‹p ∈ A› by auto
qed

lemma proj-set-subset:
  assumes p ∈ proj-set x A geodesic-segment-between G p x y ∈ G
  shows proj-set y A ⊆ proj-set x A
proof −
  have z ∈ proj-set x A if z ∈ proj-set y A for z
  proof (rule proj-setI)
    show z ∈ A using that proj-setD by auto
```

```
    have dist x z ≤ dist x y + dist y z
      by (intro mono-intros)
    also have ... ≤ dist x y + dist y p
      using proj-set-dist-le[OF proj-setD(1)[OF ‹p ∈ proj-set x A›] that] by auto
    also have ... = dist x p
      using assms geodesic-segment-commute geodesic-segment-dist by blast
    also have ... = infdist x A
      using proj-setD(2)[OF assms(1)] by simp
    finally show dist x z ≤ infdist x A
      by simp
  qed
  then show ?thesis by auto
qed

lemma proj-set-thickening:
  assumes p ∈ proj-set x Z
          0 ≤ D
          D ≤ dist p x
          geodesic-segment-between G p x
  shows geodesic-segment-param G p D ∈ proj-set x (⋃z∈Z. cball z D)
proof (rule proj-setI')
  have dist p (geodesic-segment-param G p D) = D
    using geodesic-segment-param(7)[OF assms(4), of 0 D]
    unfolding geodesic-segment-param(1)[OF assms(4)] using assms by simp
  then show geodesic-segment-param G p D ∈ (⋃z∈Z. cball z D)
    using proj-setD(1)[OF ‹p ∈ proj-set x Z›] by force
  show dist x (geodesic-segment-param G p D) ≤ dist x y if y ∈ (⋃z∈Z. cball z
D) for y
  proof −
     obtain z where y: y ∈ cball z D z ∈ Z using ‹y ∈ (⋃z∈Z. cball z D)› by
auto
    have dist (geodesic-segment-param G p D) x + D = dist p x
      using geodesic-segment-param(7)[OF assms(4), of D dist p x]
      unfolding geodesic-segment-param(2)[OF assms(4)] using assms by simp
    also have ... ≤ dist z x
       using proj-setD(2)[OF ‹p ∈ proj-set x Z›] infdist-le[OF ‹z ∈ Z›, of x] by
(simp add: dist-commute)
    also have ... ≤ dist z y + dist y x
      by (intro mono-intros)
    also have ... ≤ D + dist y x
      using y by simp
    finally show ?thesis by (simp add: dist-commute)
  qed
qed

lemma proj-set-thickening':
  assumes p ∈ proj-set x Z
          0 ≤ D
          D ≤ E
```

E ≤ *dist p x*
geodesic-segment-between G p x
shows *geodesic-segment-param G p D* ∈ *proj-set* (*geodesic-segment-param G p E*) (⋃ *z*∈*Z*. *cball z D*)
proof −
define *H* **where** *H* = *geodesic-subsegment G p D* (*dist p x*)
have *H1*: *geodesic-segment-between H* (*geodesic-segment-param G p D*) *x*
apply (*subst geodesic-segment-param*(*2*)[*OF* ‹*geodesic-segment-between G p x*›, *symmetric*])
unfolding *H-def* **apply** (*rule geodesic-subsegment*(*2*)) **using** *assms* **by** *auto*
have *H2*: *geodesic-segment-param G p E* ∈ *H*
unfolding *H-def* **using** *assms geodesic-subsegment*(*1*) **by** *force*
have *geodesic-segment-param G p D* ∈ *proj-set x* (⋃ *z*∈*Z*. *cball z D*)
apply (*rule proj-set-thickening*) **using** *assms* **by** *auto*
then show *?thesis*
by (*rule proj-set-geodesic-same-basepoint*[*OF* - *H1 H2*])
qed

It is often convenient to use *one* geodesic between x and y, even if it is not unique. We introduce a notation for such a choice of a geodesic, denoted `{x--S--y}` for such a geodesic that moreover remains in the set S. We also enforce the condition `{x--S--y} = {y--S--x}`. When there is no such geodesic, we simply take `{x--S--y} = {x, y}` for definiteness. It would be even better to enforce that, if a is on `{x--S--y}`, then `{x--S--y}` is the union of `{x--S--a}` and `{a--S--y}`, but I do not know if such a choice is always possible – such a choice of geodesics is called a geodesic bicombing. We also write `{x--y}` for `{x--UNIV--y}`.

definition *some-geodesic-segment-between*::′*a*::*metric-space* ⇒ ′*a set* ⇒ ′*a* ⇒ ′*a set* (‹(*1*{-−−−−-})›)
where *some-geodesic-segment-between* = (*SOME f*. ∀ *x y S*. *f x S y* = *f y S x*
∧ (*if* (∃ *G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S*) *then* (*geodesic-segment-between* (*f x S y*) *x y* ∧ (*f x S y* ⊆ *S*))
else f x S y = {*x*, *y*}))

abbreviation *some-geodesic-segment-between-UNIV*::′*a*::*metric-space* ⇒ ′*a* ⇒ ′*a set* (‹(*1*{-−−-})›)
where *some-geodesic-segment-between-UNIV x y* ≡ {*x*−−*UNIV*−−*y*}

We prove that there is such a choice of geodesics, compatible with direction reversal. What we do is choose arbitrarily a geodesic between x and y if it exists, and then use the geodesic between $\min(x, y)$ and $\max(x, y)$, for any total order on the space, to ensure that we get the same result from x to y or from y to x.

lemma *some-geodesic-segment-between-exists*:
∃ *f*. ∀ *x y S*. *f x S y* = *f y S x*
∧ (*if* (∃ *G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S*) *then* (*geodesic-segment-between* (*f x S y*) *x y* ∧ (*f x S y* ⊆ *S*))

else f x S y = $\{x, y\}$)
proof −
define *g*::′*a* ⇒ ′*a set* ⇒ ′*a* ⇒ ′*a set* **where**
g = (λ*x S y*. *if* (∃ *G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S*) *then* (*SOME G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S*) *else* $\{x, y\}$)
have *g1*: *geodesic-segment-between* (*g x S y*) *x y* ∧ (*g x S y* ⊆ *S*) **if** ∃ *G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S* **for** *x y S*
unfolding *g-def* **using** *someI-ex*[*OF that*] **by** *auto*
have *g2*: *g x S y* = $\{x, y\}$ **if** ¬(∃ *G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S*) **for** *x y S*
unfolding *g-def* **using** *that* **by** *auto*
obtain *r*::′*a rel* **where** *r*: *well-order-on UNIV r*
using *well-order-on* **by** *auto*
have *A*: *x* = *y* **if** (*x*, *y*) ∈ *r* (*y*, *x*) ∈ *r* **for** *x y*
using *r that* **unfolding** *well-order-on-def linear-order-on-def partial-order-on-def antisym-def* **by** *auto*
have *B*: (*x*, *y*) ∈ *r* ∨ (*y*, *x*) ∈ *r* **for** *x y*
using *r* **unfolding** *well-order-on-def linear-order-on-def total-on-def partial-order-on-def preorder-on-def refl-on-def* **by** *force*

define *f* **where** *f* = (λ*x S y*. *if* (*x*, *y*) ∈ *r then g x S y else g y S x*)
have *f x S y* = *f y S x* **for** *x y S* **unfolding** *f-def* **using** *r A B* **by** *auto*
moreover have *geodesic-segment-between* (*f x S y*) *x y* ∧ (*f x S y* ⊆ *S*) **if** ∃ *G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S* **for** *x y S*
unfolding *f-def* **using** *g1 geodesic-segment-commute that* **by** *smt*
moreover have *f x S y* = $\{x, y\}$ **if** ¬(∃ *G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S*) **for** *x y S*
unfolding *f-def* **using** *g2 that geodesic-segment-commute doubleton-eq-iff* **by** *metis*
ultimately show *?thesis* **by** *metis*
qed

lemma *some-geodesic-commute*:
$\{x{-}{-}S{-}{-}y\} = \{y{-}{-}S{-}{-}x\}$
unfolding *some-geodesic-segment-between-def* **by** (*auto simp add*: *someI-ex*[*OF some-geodesic-segment-between-exists*])

lemma *some-geodesic-segment-description*:
(∃ *G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S*) ⟹ *geodesic-segment-between* $\{x{-}{-}S{-}{-}y\}$ *x y*
(¬(∃ *G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S*)) ⟹ $\{x{-}{-}S{-}{-}y\} = \{x, y\}$
unfolding *some-geodesic-segment-between-def* **by** (*simp add*: *someI-ex*[*OF some-geodesic-segment-between-exi*

Basic topological properties of our chosen set of geodesics.

lemma *some-geodesic-compact* [*simp*]:
compact $\{x{-}{-}S{-}{-}y\}$
apply (*cases* ∃ *G*. *geodesic-segment-between G x y* ∧ *G* ⊆ *S*)
using *some-geodesic-segment-description*[*of x y*] *geodesic-segment-topology*[*of* $\{x{-}{-}S{-}{-}y\}$] *geodesic-segment-def* **apply** *auto*

by *blast*

lemma *some-geodesic-closed* [*simp*]:
closed $\{x{-}{-}S{-}{-}y\}$
by (*rule compact-imp-closed*[*OF some-geodesic-compact*[*of x S y*]])

lemma *some-geodesic-bounded* [*simp*]:
bounded $\{x{-}{-}S{-}{-}y\}$
by (*rule compact-imp-bounded*[*OF some-geodesic-compact*[*of x S y*]])

lemma *some-geodesic-endpoints* [*simp*]:
$x \in \{x{-}{-}S{-}{-}y\}$ $y \in \{x{-}{-}S{-}{-}y\}$ $\{x{-}{-}S{-}{-}y\} \neq \{\}$
apply (*cases* $\exists$ *G. geodesic-segment-between G x y* $\wedge$ $G \subseteq S$) **using** *some-geodesic-segment-description*[*of x y S*] **apply** *auto*
apply (*cases* $\exists$ *G. geodesic-segment-between G x y* $\wedge$ $G \subseteq S$) **using** *some-geodesic-segment-description*[*of x y S*] **apply** *auto*
apply (*cases* $\exists$ *G. geodesic-segment-between G x y* $\wedge$ $G \subseteq S$) **using** *geodesic-segment-endpoints*(*3*)
by (*auto*, *blast*)

lemma *some-geodesic-subsegment*:
assumes $H \subseteq \{x{-}{-}S{-}{-}y\}$ *compact H connected H* $H \neq \{\}$
shows *geodesic-segment H*
apply (*cases* $\exists$ *G. geodesic-segment-between G x y* $\wedge$ $G \subseteq S$)
using *some-geodesic-segment-description*[*of x y*] *geodesic-segment-subsegment*[*OF - assms*] *geodesic-segment-def* **apply** *auto*[*1*]
using *some-geodesic-segment-description*[*of x y*] *assms*
by (*metis connected-finite-iff-sing finite.emptyI finite.insertI finite-subset geodesic-segment-between-x-x*(*2*))

lemma *some-geodesic-in-subset*:
assumes $x \in S$ $y \in S$
shows $\{x{-}{-}S{-}{-}y\} \subseteq S$
apply (*cases* $\exists$ *G. geodesic-segment-between G x y* $\wedge$ $G \subseteq S$)
unfolding *some-geodesic-segment-between-def* **by** (*simp add*: *assms someI-ex*[*OF some-geodesic-segment-between-exists*])+

lemma *some-geodesic-same-endpoints* [*simp*]:
$\{x{-}{-}S{-}{-}x\} = \{x\}$
apply (*cases* $\exists$ *G. geodesic-segment-between G x x* $\wedge$ $G \subseteq S$)
apply (*meson geodesic-segment-between-x-x*(*3*) *some-geodesic-segment-description*(*1*))
by (*simp add*: *some-geodesic-segment-description*(*2*))

5.2 Geodesic subsets

A subset is *geodesic* if any two of its points can be joined by a geodesic segment. We prove basic properties of such a subset in this paragraph – notably connectedness. A basic example is given by convex subsets of vector spaces, as closed segments are geodesic.

definition *geodesic-subset*::(*'a*::*metric-space*) *set* $\Rightarrow$ *bool*

where *geodesic-subset* $S = (\forall x{\in}S.\ \forall y{\in}S.\ \exists G.\ \textit{geodesic-segment-between}\ G\ x\ y \wedge G \subseteq S)$

lemma *geodesic-subsetD*:
assumes *geodesic-subset* S $x \in S$ $y \in S$
shows *geodesic-segment-between* $\{x{-}{-}S{-}{-}y\}$ x y
using *assms some-geodesic-segment-description*(*1*) **unfolding** *geodesic-subset-def*
by *blast*

lemma *geodesic-subsetI*:
assumes $\bigwedge x\ y.\ x \in S \Longrightarrow y \in S \Longrightarrow \exists G.\ \textit{geodesic-segment-between}\ G\ x\ y \wedge G \subseteq S$
shows *geodesic-subset* S
using *assms* **unfolding** *geodesic-subset-def* **by** *auto*

lemma *geodesic-subset-empty*:
geodesic-subset $\{\}$
using *geodesic-subsetI* **by** *auto*

lemma *geodesic-subset-singleton*:
geodesic-subset $\{x\}$
by (*auto intro*!: *geodesic-subsetI geodesic-segment-between-x-x*(*1*))

lemma *geodesic-subset-path-connected*:
assumes *geodesic-subset* S
shows *path-connected* S
proof −
have $\exists g.\ \textit{path}\ g \wedge \textit{path-image}\ g \subseteq S \wedge \textit{pathstart}\ g = x \wedge \textit{pathfinish}\ g = y$ **if** $x \in S$ $y \in S$ **for** x y
proof −
define G **where** $G = \{x{-}{-}S{-}{-}y\}$
have $*$: *geodesic-segment-between* G x y $G \subseteq S$ $x \in G$ $y \in G$
using *assms that* **by** (*auto simp add*: *G-def geodesic-subsetD some-geodesic-in-subset that*(*1*) *that*(*2*))
then have *path-connected* G
using *geodesic-segment-topology*(*3*) **unfolding** *geodesic-segment-def* **by** *auto*
then have $\exists g.\ \textit{path}\ g \wedge \textit{path-image}\ g \subseteq G \wedge \textit{pathstart}\ g = x \wedge \textit{pathfinish}\ g = y$
using $*$ **unfolding** *path-connected-def* **by** *auto*
then show *?thesis* **using** ‹$G \subseteq S$› **by** *auto*
qed
then show *?thesis*
unfolding *path-connected-def* **by** *auto*
qed

To show that a segment in a normed vector space is geodesic, we will need to use its length parametrization, which is given in the next lemma.

lemma *closed-segment-as-isometric-image*:
$((\lambda t.\ x + (t/\textit{dist}\ x\ y) *_R (y - x))\ `\{0..\textit{dist}\ x\ y\}) = \textit{closed-segment}\ x\ y$

proof (*auto simp add*: *closed-segment-def image-iff*)
  fix *t* **assume** *H*: $0 \leq t$ $t \leq$ *dist x y*
  show $\exists u.\ x + (t \ /\ dist\ x\ y) *_R (y - x) = (1 - u) *_R x + u *_R y \wedge 0 \leq u \wedge u \leq 1$
    apply (*rule exI*[*of - t/dist x y*])
    using *H* **apply** (*auto simp add*: *algebra-simps divide-simps*)
   apply (*metis add-diff-cancel-left′ add-diff-eq add-divide-distrib dist-eq-0-iff scaleR-add-left vector-fraction-eq-iff*)
    done
next
  fix *u*::*real* **assume** *H*: $0 \leq u$ $u \leq 1$
  show $\exists t\in\{0..dist\ x\ y\}.\ (1 - u) *_R x + u *_R y = x + (t \ /\ dist\ x\ y) *_R (y - x)$
    apply (*rule bexI*[*of - u * dist x y*])
    using *H* **by** (*auto simp add*: *algebra-simps mult-left-le-one-le*)
qed

proposition *closed-segment-is-geodesic*:
  fixes *x y*::*′a*::*real-normed-vector*
  shows *isometry-on* $\{0..dist\ x\ y\}$ ($\lambda t.\ x + (t/dist\ x\ y) *_R (y - x)$)
        geodesic-segment-between (*closed-segment x y*) *x y*
        geodesic-segment (*closed-segment x y*)
proof −
  show ∗: *isometry-on* $\{0..dist\ x\ y\}$ ($\lambda t.\ x + (t/dist\ x\ y) *_R (y - x)$)
    unfolding *isometry-on-def dist-norm*
    apply (*cases* $x = y$)
      by (*auto simp add*: *scaleR-diff-left*[*symmetric*] *diff-divide-distrib*[*symmetric*] *norm-minus-commute*)
  show *geodesic-segment-between* (*closed-segment x y*) *x y*
    unfolding *closed-segment-as-isometric-image*[*symmetric*]
    apply (*rule geodesic-segment-betweenI*[*OF - - ∗*]) **by** *auto*
  then show *geodesic-segment* (*closed-segment x y*)
    by *auto*
qed

We deduce that a convex set is geodesic.

proposition *convex-is-geodesic*:
  assumes *convex* (*S*::*′a*::*real-normed-vector set*)
  shows *geodesic-subset S*
proof (*rule geodesic-subsetI*)
  fix *x y* **assume** *H*: $x \in S$ $y \in S$
  show $\exists G.$ *geodesic-segment-between G x y* $\wedge G \subseteq S$
    apply (*rule exI*[*of - closed-segment x y*])
    apply (*auto simp add*: *closed-segment-is-geodesic*)
    using *H assms convex-contains-segment* **by** *blast*
qed

5.3 Geodesic spaces

In this subsection, we define geodesic spaces (metric spaces in which there is a geodesic segment joining any pair of points). We specialize the previous statements on geodesic segments to these situations.

class *geodesic-space* = *metric-space* +
assumes *geodesic*: *geodesic-subset* (*UNIV*::('*a*::*metric-space*) *set*)

The simplest example of a geodesic space is a real normed vector space. Significant examples also include graphs (with the graph distance), Riemannian manifolds, and $CAT(\kappa)$ spaces.

instance *real-normed-vector* ⊆ *geodesic-space*
by (*standard*, *simp add*: *convex-is-geodesic*)

lemma (**in** *geodesic-space*) *some-geodesic-is-geodesic-segment* [*simp*]:
geodesic-segment-between $\{x{-}{-}y\}$ x $(y::'a)$
geodesic-segment $\{x{-}{-}y\}$
using *some-geodesic-segment-description*(*1*)[*of x y*] *geodesic-subsetD*[*OF geodesic*]
by (*auto*, *blast*)

lemma (**in** *geodesic-space*) *some-geodesic-connected* [*simp*]:
connected $\{x{-}{-}y\}$ *path-connected* $\{x{-}{-}y\}$
by (*auto intro*!: *geodesic-segment-topology*)

In geodesic spaces, we restate as simp rules all properties of the geodesic segment parametrizations.

lemma (**in** *geodesic-space*) *geodesic-segment-param-in-geodesic-spaces* [*simp*]:
geodesic-segment-param $\{x{-}{-}y\}$ x $0 = x$
geodesic-segment-param $\{x{-}{-}y\}$ x (*dist* x y) $= y$
$t \in \{0..\mathit{dist}\ x\ y\} \Longrightarrow$ *geodesic-segment-param* $\{x{-}{-}y\}$ x $t \in \{x{-}{-}y\}$
isometry-on $\{0..\mathit{dist}\ x\ y\}$ (*geodesic-segment-param* $\{x{-}{-}y\}$ x)
(*geodesic-segment-param* $\{x{-}{-}y\}$ x)‘$\{0..\mathit{dist}\ x\ y\} = \{x{-}{-}y\}$
$t \in \{0..\mathit{dist}\ x\ y\} \Longrightarrow$ *dist* x (*geodesic-segment-param* $\{x{-}{-}y\}$ x t) $= t$
$s \in \{0..\mathit{dist}\ x\ y\} \Longrightarrow t \in \{0..\mathit{dist}\ x\ y\} \Longrightarrow$ *dist* (*geodesic-segment-param* $\{x{-}{-}y\}$ x s) (*geodesic-segment-param* $\{x{-}{-}y\}$ x t) $=$ *abs*$(s-t)$
$z \in \{x{-}{-}y\} \Longrightarrow z =$ *geodesic-segment-param* $\{x{-}{-}y\}$ x (*dist* x z)
using *geodesic-segment-param*[*OF some-geodesic-is-geodesic-segment*(*1*)[*of x y*]] **by** *auto*

5.4 Uniquely geodesic spaces

In this subsection, we define uniquely geodesic spaces, i.e., geodesic spaces in which, additionally, there is a unique geodesic between any pair of points.

class *uniquely-geodesic-space* = *geodesic-space* +
assumes *uniquely-geodesic*: $\bigwedge x\ y\ G\ H.$ *geodesic-segment-between* G x $y \Longrightarrow$ *geodesic-segment-between* H x $y \Longrightarrow G = H$

To prove that a geodesic space is uniquely geodesic, it suffices to show that there is no loop, i.e., if two geodesic segments intersect only at their endpoints, then they coincide.

Indeed, assume this holds, and consider two geodesics with the same endpoints. If they differ at some time t, then consider the last time a before t where they coincide, and the first time b after t where they coincide. Then the restrictions of the two geodesics to $[a, b]$ give a loop, and a contradiction.

lemma (**in** *geodesic-space*) *uniquely-geodesic-spaceI*:
 assumes $\bigwedge G\ H\ x\ (y::'a).$ *geodesic-segment-between* $G\ x\ y \Longrightarrow$ *geodesic-segment-between* $H\ x\ y \Longrightarrow G \cap H = \{x, y\} \Longrightarrow x = y$
  geodesic-segment-between $G\ x\ y$ *geodesic-segment-between* $H\ x\ (y::'a)$
 shows $G = H$
proof −
 obtain g **where** g: $g\ 0 = x$ $g\ (dist\ x\ y) = y$ *isometry-on* $\{0..dist\ x\ y\}$ g $G = g`\{0..dist\ x\ y\}$
  by (*meson* ‹*geodesic-segment-between* $G\ x\ y$› *geodesic-segment-between-def*)
 obtain h **where** h: $h\ 0 = x$ $h\ (dist\ x\ y) = y$ *isometry-on* $\{0..dist\ x\ y\}$ h $H = h`\{0..dist\ x\ y\}$
  by (*meson* ‹*geodesic-segment-between* $H\ x\ y$› *geodesic-segment-between-def*)
 have $g\ t = h\ t$ **if** $t \in \{0..dist\ x\ y\}$ **for** t
 proof (*rule ccontr*)
  assume $g\ t \neq h\ t$
  define Z **where** $Z = \{s \in \{0..dist\ x\ y\}.\ g\ s = h\ s\}$
  have $0 \in Z$ $dist\ x\ y \in Z$ **unfolding** *Z-def* **using** g h **by** *auto*
  have $t \notin Z$ **unfolding** *Z-def* **using** ‹$g\ t \neq h\ t$› **by** *auto*
  have [*simp*]: *closed* Z
  proof −
   have ∗: $Z = (\lambda s.\ dist\ (g\ s)\ (h\ s))-`\{0\} \cap \{0..dist\ x\ y\}$
    unfolding *Z-def* **by** *auto*
   show *?thesis*
    unfolding ∗ **apply** (*rule closed-vimage-Int*)
     using *isometry-on-continuous*[*OF* $g(3)$] *isometry-on-continuous*[*OF* $h(3)$] *continuous-on-dist* **by** *auto*
  qed
  define a **where** $a = Sup\ (Z \cap \{0..t\})$
  have a: $a \in Z \cap \{0..t\}$
   unfolding *a-def* **apply** (*rule closed-contains-Sup*, *auto*)
   using ‹$0 \in Z$› *that* **by** *auto*
  then have $h\ a = g\ a$ **unfolding** *Z-def* **by** *auto*
  define b **where** $b = Inf\ (Z \cap \{t..dist\ x\ y\})$
  have b: $b \in Z \cap \{t..dist\ x\ y\}$
   unfolding *b-def* **apply** (*rule closed-contains-Inf*, *auto*)
   using ‹$dist\ x\ y \in Z$› *that* **by** *auto*
  then have $h\ b = g\ b$ **unfolding** *Z-def* **by** *auto*
  have *notZ*: $s \notin Z$ **if** $s \in \{a<..<b\}$ **for** s
  proof (*rule ccontr*, *auto*, *cases* $s \leq t$)
   case *True*
   assume $s \in Z$

then have ∗: $s \in Z \cap \{0..t\}$ **using** *that a True* **by** *auto*
have $s \leq a$ **unfolding** *a-def* **apply** (*rule cSup-upper*) **using** ∗ **by** *auto*
then show *False* **using** *that* **by** *auto*
next
case *False*
assume $s \in Z$
then have ∗: $s \in Z \cap \{t..dist\ x\ y\}$ **using** *that b False* **by** *auto*
have $s \geq b$ **unfolding** *b-def* **apply** (*rule cInf-lower*) **using** ∗ **by** *auto*
then show *False* **using** *that* **by** *auto*
qed
have $t \in \{a<..<b\}$ **using** *a b* ‹$t \notin Z$› *less-eq-real-def* **by** *auto*
then have $a \leq b$ **by** *auto*
then have $dist\ (h\ a)\ (h\ b) = b-a$
using *isometry-onD*[*OF h*(*3*), *of a b*] *a b that* **unfolding** *dist-real-def* **by** *auto*
then have $dist\ (h\ a)\ (h\ b) > 0$ **using** ‹$t \in \{a<..<b\}$› **by** *auto*
then have $h\ a \neq h\ b$ **by** *auto*

define *G2* **where** $G2 = g`\{a..b\}$
define *H2* **where** $H2 = h`\{a..b\}$
have $G2 \cap H2 \subseteq \{h\ a,\ h\ b\}$
proof
fix *z* **assume** *z*: $z \in G2 \cap H2$
obtain *sg* **where** *sg*: $z = g\ sg\ sg \in \{a..b\}$ **using** *z* **unfolding** *G2-def* **by** *auto*
obtain *sh* **where** *sh*: $z = h\ sh\ sh \in \{a..b\}$ **using** *z* **unfolding** *H2-def* **by** *auto*
have $sg = dist\ x\ z$
using *isometry-onD*[*OF g*(*3*), *of 0 sg*] *a b sg*(*2*) **unfolding** *sg*(*1*) *g*(*1*)[*symmetric*] *dist-real-def* **by** *auto*
moreover have $sh = dist\ x\ z$
using *isometry-onD*[*OF h*(*3*), *of 0 sh*] *a b sh*(*2*) **unfolding** *sh*(*1*) *h*(*1*)[*symmetric*] *dist-real-def* **by** *auto*
ultimately have $sg = sh$ **by** *auto*
then have $sh \in Z$ **using** *sg*(*1*) *sh*(*1*) *a b sh*(*2*) **unfolding** *Z-def* **by** *auto*
then have $sh \in \{a,\ b\}$ **using** *notZ sh*(*2*)
by (*metis IntD2 atLeastAtMost-iff atLeastAtMost-singleton greaterThanLessThan-iff inf-bot-left insertI2 insert-inter-insert not-le*)
then show $z \in \{h\ a,\ h\ b\}$ **using** *sh*(*1*) **by** *auto*
qed
then have $G2 \cap H2 = \{h\ a,\ h\ b\}$
using ‹$h\ a = g\ a$› ‹$h\ b = g\ b$› ‹$a \leq b$› **unfolding** *H2-def G2-def* **apply** *auto*
unfolding ‹$h\ a = g\ a$›[*symmetric*] ‹$h\ b = g\ b$›[*symmetric*] **by** *auto*
moreover have *geodesic-segment-between G2* (*h a*) (*h b*)
unfolding *G2-def* ‹$h\ a = g\ a$› ‹$h\ b = g\ b$›
apply (*rule geodesic-segmentI2*) **apply** (*rule isometry-on-subset*[*OF g*(*3*)])
using *a b that* **by** *auto*
moreover have *geodesic-segment-between H2* (*h a*) (*h b*)
unfolding *H2-def* **apply** (*rule geodesic-segmentI2*) **apply** (*rule isometry-on-subset*[*OF h*(*3*)])
using *a b that* **by** *auto*

ultimately have $h\ a = h\ b$ **using** *assms*(*1*) **by** *auto*
then show *False* **using** ‹$h\ a \neq h\ b$› **by** *simp*
qed
then show $G = H$ **using** *g*(*4*) *h*(*4*) **by** (*simp add*: *image-def*)
qed

context *uniquely-geodesic-space*
begin

lemma *geodesic-segment-unique*:
geodesic-segment-between $G\ x\ y = (G = \{x{-}{-}(y::'a)\})$
using *uniquely-geodesic*[*of - x y*] **by** (*meson some-geodesic-is-geodesic-segment*)

lemma *geodesic-segment-dist'*:
assumes *dist* $x\ z =$ *dist* $x\ y +$ *dist* $y\ z$
shows $y \in \{x{-}{-}z\}$ $\{x{-}{-}z\} = \{x{-}{-}y\} \cup \{y{-}{-}z\}$
proof −
have *geodesic-segment-between* $(\{x{-}{-}y\} \cup \{y{-}{-}z\})\ x\ z$
using *geodesic-segment-union*[*OF assms*] **by** *auto*
then show $\{x{-}{-}z\} = \{x{-}{-}y\} \cup \{y{-}{-}z\}$
using *geodesic-segment-unique* **by** *auto*
then show $y \in \{x{-}{-}z\}$ **by** *auto*
qed

lemma *geodesic-segment-expression*:
$\{x{-}{-}z\} = \{y.$ *dist* $x\ z =$ *dist* $x\ y +$ *dist* $y\ z\}$
using *geodesic-segment-dist'*(*1*) *geodesic-segment-dist*[*OF some-geodesic-is-geodesic-segment*(*1*)]
by *auto*

lemma *geodesic-segment-split*:
assumes $(y::'a) \in \{x{-}{-}z\}$
shows $\{x{-}{-}z\} = \{x{-}{-}y\} \cup \{y{-}{-}z\}$
$\{x{-}{-}y\} \cap \{y{-}{-}z\} = \{y\}$
apply (*metis assms geodesic-segment-dist geodesic-segment-dist'*(*2*) *some-geodesic-is-geodesic-segment*(*1*))
apply (*rule geodesic-segment-union*(*2*)[*of x z*], *auto simp add*: *assms*)
using *assms geodesic-segment-expression* **by** *blast*

lemma *geodesic-segment-subparam'*:
assumes $y \in \{x{-}{-}z\}$ $t \in \{0..$*dist* $x\ y\}$
shows *geodesic-segment-param* $\{x{-}{-}z\}\ x\ t =$ *geodesic-segment-param* $\{x{-}{-}y\}$
$x\ t$
apply (*rule geodesic-segment-subparam*[*of - - z - y*]) **using** *assms* **apply** *auto*
using *geodesic-segment-split*(*1*)[*OF assms*(*1*)] **by** *auto*

end

5.5 A complete metric space with middles is geodesic.

A complete space in which every pair of points has a middle (i.e., a point m which is half distance of x and y) is geodesic: to construct a geodesic between x_0 and y_0, first choose a middle m, then middles of the pairs (x_0, m) and (m, y_0), and so on. This will define the geodesic on dyadic points (and this is indeed an isometry on these dyadic points. Then, extend it by uniform continuity to the whole segment $[0, dist x0 y0]$.

The formal proof will be done in a locale where x_0 and y_0 are fixed, for notational simplicity. We define inductively the sequence of middles, in a function `geod` of two natural variables: *geod n m* corresponds to the image of the dyadic point $m/2^n$. It is defined inductively, by $geod (n+1) (2m) = geod n m$, and $geod (n+1) (2m+1)$ is a middle of *geod n m* and $geod n (m+1)$. This is not a completely classical inductive definition, so one has to use `function` to define it. Then, one checks inductively that it has all the properties we want, and use it to define the geodesic segment on dyadic points. We will not use a canonical representative for a dyadic point, but any representative (i.e., numerator and denominator will not have to be coprime) – this will not create problems as *geod* does not depend on the choice of the representative, by construction.

locale *complete-space-with-middle* =
  fixes *x0 y0*::*'a*::*complete-space*
  assumes *middles*: $\bigwedge x\ y$::*'a*. $\exists$ *z*. *dist x z* = (*dist x y*)/*2* $\wedge$ *dist z y* = (*dist x y*)/*2*
begin

definition *middle*::*'a* ⇒ *'a* ⇒ *'a*
  where *middle x y* = (*SOME z*. *dist x z* = (*dist x y*)/*2* $\wedge$ *dist z y* = (*dist x y*)/*2*)

lemma *middle*:
  dist x (*middle x y*) = (*dist x y*)/*2*
  dist (*middle x y*) *y* = (*dist x y*)/*2*
unfolding *middle-def* **using** *middles*[*of x y*] **by** (*metis* (*mono-tags*, *lifting*) *someI-ex*)+

function *geod*::*nat* ⇒ *nat* ⇒ *'a* **where**
 geod 0 0 = *x0*
|*geod 0* (*Suc m*) = *y0*
|*geod* (*Suc n*) (*2* ∗ *m*) = *geod n m*
|*geod* (*Suc n*) (*Suc* (*2*∗*m*)) = *middle* (*geod n m*) (*geod n* (*Suc m*))
apply (*auto simp add*: *double-not-eq-Suc-double*)
by (*metis One-nat-def dvd-mult-div-cancel list-decode.cases odd-Suc-minus-one odd-two-times-div-two-nat*)
termination by *lexicographic-order*

By induction, the distance between successive points is $D/2^n$.

lemma *geod-distance-successor*:
  ∀ *a* < *2*^*n*. *dist* (*geod n a*) (*geod n* (*Suc a*)) = *dist x0 y0* / *2*^*n*
proof (*induction n*)
  case *0*

```
  show ?case by auto
next
  case (Suc n)
  show ?case
  proof (auto)
    fix a::nat assume a: a < 2 * 2^n
    obtain m where m: a = 2 * m ∨ a = Suc (2 * m) by (metis geod.elims)
    then have m < 2^n using a by auto
    consider a = 2 * m | a = Suc(2*m) using m by auto
    then show dist (geod (Suc n) a) (geod (Suc n) (Suc a)) = dist x0 y0 / (2 * 2
^ n)
    proof (cases)
      case 1
      show ?thesis
        unfolding 1 apply auto
        unfolding middle using Suc.IH ‹m < 2^n› by auto
    next
      case 2
      have *: Suc (Suc (2 * m)) = 2 * (Suc m) by auto
      show ?thesis
        unfolding 2 apply auto
        unfolding * geod.simps(3) middle using Suc.IH ‹m < 2^n› by auto
    qed
  qed
qed

lemma geod-mult:
  geod n a = geod (n + k) (a * 2^k)
apply (induction k, auto) using geod.simps(3) by (metis mult.left-commute)

lemma geod-0:
  geod n 0 = x0
by (induction n, auto, metis geod.simps(3) semiring-normalization-rules(10))

lemma geod-end:
  geod n (2^n) = y0
by (induction n, auto)
```

By the triangular inequality, the distance between points separated by $(b-a)/2^n$ is at most $D*(b-a)/2^n$.

```
lemma geod-upper:
  assumes a ≤ b b ≤ 2^n
  shows dist (geod n a) (geod n b) ≤ (b−a) * dist x0 y0 / 2^n
proof −
  have *: a+k > 2^n ∨ dist (geod n a) (geod n (a+k)) ≤ k * dist x0 y0 / 2^n for
k
  proof (induction k)
    case 0 then show ?case by auto
  next
```

case (*Suc k*)
show *?case*
proof (*cases* $2 \,\hat{}\, n < a + Suc\ k$)
case *True* **then show** *?thesis* **by** *auto*
next
case *False*
then have $*$: $a + k < 2 \,\hat{}\, n$ **by** *auto*
have $dist\ (geod\ n\ a)\ (geod\ n\ (a + Suc\ k)) \leq dist\ (geod\ n\ a)\ (geod\ n\ (a{+}k)) + dist\ (geod\ n\ (a{+}k))\ (geod\ n\ (a{+}Suc\ k))$
using *dist-triangle* **by** *auto*
also have ... $\leq k * dist\ x0\ y0\ /\ 2\hat{}n + dist\ x0\ y0\ /\ 2\hat{}n$
using *Suc.IH* $*$ *geod-distance-successor* **by** *auto*
finally show *?thesis*
by (*simp add*: *add-divide-distrib distrib-left mult.commute*)
qed
qed
show *?thesis* **using** $*[of\ b{-}a]$ *assms* **by** (*simp add*: *of-nat-diff*)
qed

In fact, the distance is exactly $D * (b - a)/2^n$, otherwise the extremities of the interval would be closer than D, a contradiction.

lemma *geod-dist*:
assumes $a \leq b$ $b \leq 2\hat{}n$
shows $dist\ (geod\ n\ a)\ (geod\ n\ b) = (b{-}a) * dist\ x0\ y0\ /\ 2\hat{}n$
proof $-$
have $dist\ (geod\ n\ a)\ (geod\ n\ b) \leq (real\ b{-}a) * dist\ x0\ y0\ /\ 2\hat{}n$
using *geod-upper*[*of a b n*] *assms* **by** *auto*
moreover have $\neg\ (dist\ (geod\ n\ a)\ (geod\ n\ b) < (real\ b{-}a) * dist\ x0\ y0\ /\ 2\hat{}n)$
proof (*rule ccontr*, *simp*)
assume $*$: $dist\ (geod\ n\ a)\ (geod\ n\ b) < (real\ b{-}a) * dist\ x0\ y0\ /\ 2\hat{}n$
have $dist\ x0\ y0 = dist\ (geod\ n\ 0)\ (geod\ n\ (2\hat{}n))$
using *geod-0 geod-end* **by** *auto*
also have ... $\leq dist\ (geod\ n\ 0)\ (geod\ n\ a) + dist\ (geod\ n\ a)\ (geod\ n\ b) + dist\ (geod\ n\ b)\ (geod\ n\ (2\hat{}n))$
using *dist-triangle4* **by** *auto*
also have ... $< a * dist\ x0\ y0\ /\ 2\hat{}n + (real\ b{-}a) * dist\ x0\ y0\ /\ 2\hat{}n + (2\hat{}n - real\ b) * dist\ x0\ y0\ /\ 2\hat{}n$
using $*$ *assms geod-upper*[*of 0 a n*] *geod-upper*[*of b* $2\hat{}n$ *n*] **by** (*auto intro*: *mono-intros*)
also have ... $= dist\ x0\ y0$
using *assms* **by** (*auto simp add*: *algebra-simps divide-simps*)
finally show *False* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

We deduce the same statement but for points that are not on the same level, by putting them on a common multiple level.

lemma *geod-dist2*:

assumes $a \leq 2\hat{}n$ $b \leq 2\hat{}p$ $a/2\hat{}n \leq b\ /\ 2\hat{}p$
shows $dist\ (geod\ n\ a)\ (geod\ p\ b) = (b/2\hat{}p - a/2\hat{}n) * dist\ x0\ y0$
proof –
define *r* **where** $r = max\ n\ p$
define *ar* **where** $ar = a * 2\hat{}(r - n)$
have *a*: $ar\ /\ 2\hat{}r = a\ /\ 2\hat{}n$
unfolding *ar-def r-def* **by** (*auto simp add*: *divide-simps semiring-normalization-rules*(*26*))
have *A*: $geod\ r\ ar = geod\ n\ a$
unfolding *ar-def r-def* **using** *geod-mult*[*of n a max n p* – *n*] **by** *auto*
define *br* **where** $br = b * 2\hat{}(r - p)$
have *b*: $br\ /\ 2\hat{}r = b\ /\ 2\hat{}p$
unfolding *br-def r-def* **by** (*auto simp add*: *divide-simps semiring-normalization-rules*(*26*))
have *B*: $geod\ r\ br = geod\ p\ b$
unfolding *br-def r-def* **using** *geod-mult*[*of p b max n p* – *p*] **by** *auto*

have $dist\ (geod\ n\ a)\ (geod\ p\ b) = dist\ (geod\ r\ ar)\ (geod\ r\ br)$
using *A B* **by** *auto*
also have ... $= (real\ br - ar) * dist\ x0\ y0\ /\ 2\ \hat{}r$
apply (*rule geod-dist*)
using ‹$a/2\hat{}n \leq b\ /\ 2\hat{}p$› **unfolding** *a*[*symmetric*] *b*[*symmetric*] **apply** (*auto simp add*: *divide-simps*)
using ‹$b \leq 2\hat{}p$› *b* **apply** (*auto simp add*: *divide-simps*)
by (*metis br-def le-add-diff-inverse2 max.cobounded2 mult.commute mult-le-mono2 r-def semiring-normalization-rules*(*26*))
also have ... $= (real\ br\ /\ 2\hat{}r - real\ ar\ /\ 2\hat{}r) * dist\ x0\ y0$
by (*auto simp add*: *algebra-simps divide-simps*)
finally show *?thesis* **using** *a b* **by** *auto*
qed

Same thing but without a priori ordering of the points.

lemma *geod-dist3*:
assumes $a \leq 2\hat{}n$ $b \leq 2\hat{}p$
shows $dist\ (geod\ n\ a)\ (geod\ p\ b) = abs(b/2\hat{}p - a/2\hat{}n) * dist\ x0\ y0$
apply (*cases* $a\ /2\hat{}n \leq b/2\hat{}p$, *auto*)
apply (*rule geod-dist2*[*OF assms*], *auto*)
apply (*subst dist-commute*, *rule geod-dist2*[*OF assms*(*2*) *assms*(*1*)], *auto*)
done

Finally, we define a geodesic by extending what we have already defined on dyadic points, thanks to the result of isometric extension of isometries taking their values in complete spaces.

lemma *geod*:
shows $\exists g.\ isometry\text{-}on\ \{0..dist\ x0\ y0\}\ g \wedge g\ 0 = x0 \wedge g\ (dist\ x0\ y0) = y0$
proof (*cases* $x0 = y0$)
case *True*
show *?thesis* **apply** (*rule exI*[*of* - λ-. *x0*]) **unfolding** *isometry-on-def* **using** *True* **by** *auto*
next
case *False*

define *A* **where** *A* = {(*real k*/*2*^*n*) ∗ *dist x0 y0* |*k n*. *k* ≤ *2*^*n*}
have {*0*..*dist x0 y0*} ⊆ *closure A*
proof (*auto simp add*: *closure-approachable dist-real-def*)
fix *t*::*real* **assume** *t*: *0* ≤ *t t* ≤ *dist x0 y0*
fix *e*:: *real* **assume** *e* > *0*
then obtain *n*::*nat* **where** *n*: *dist x0 y0*/*e* < *2*^*n*
using *one-less-numeral-iff real-arch-pow semiring-norm*(*76*) **by** *blast*
define *k* **where** *k* = *floor* (*2*^*n* ∗ *t*/ *dist x0 y0*)
have *k* ≤ *2*^*n* ∗ *t*/ *dist x0 y0* **unfolding** *k-def* **by** *auto*
also have ... ≤ *2*^*n* **using** *t False* **by** (*auto simp add*: *algebra-simps divide-simps*)
finally have *k* ≤ *2*^*n* **by** *auto*
have *k* ≥ *0* **using** *t False* **unfolding** *k-def* **by** *auto*
define *l* **where** *l* = *nat k*
have *k* = *int l l* ≤ *2*^*n* **using** ‹*k* ≥ *0*› ‹*k* ≤ *2*^*n*› *nat-le-iff* **unfolding** *l-def* **by** *auto*

have *abs* (*2*^*n* ∗ *t*/*dist x0 y0* − *k*) ≤ *1* **unfolding** *k-def* **by** *linarith*
then have *abs*(*t* − *k*/*2*^*n* ∗ *dist x0 y0*) ≤ *dist x0 y0* / *2*^*n*
by (*auto simp add*: *algebra-simps divide-simps False*)
also have ... < *e* **using** *n* ‹*e* > *0*› **by** (*auto simp add*: *algebra-simps divide-simps*)
finally have *abs*(*t* − *k*/*2*^*n* ∗ *dist x0 y0*) < *e* **by** *auto*
then have *abs*(*t* − *l*/*2*^*n* ∗ *dist x0 y0*) < *e* **using** ‹*k* = *int l*› **by** *auto*
moreover have *l*/*2*^*n* ∗ *dist x0 y0* ∈ *A* **unfolding** *A-def* **using** ‹*l* ≤ *2*^*n*› **by** *auto*
ultimately show ∃ *u*∈*A*. *abs*(*u* − *t*) < *e* **by** *force*
qed

For each dyadic point, we choose one representation of the form $K/2^N$, it is not important for us that it is the minimal one.

define *index* **where** *index* = (λ*t*. *SOME i*. *t* = *real* (*fst i*)/*2*^(*snd i*) ∗ *dist x0 y0* ∧ (*fst i*) ≤ *2*^(*snd i*))
define *K* **where** *K* = (λ*t*. *fst* (*index t*))
define *N* **where** *N* = (λ*t*. *snd* (*index t*))
have *t*: *t* = *K t*/ *2*^(*N t*) ∗ *dist x0 y0* ∧ *K t* ≤ *2*^(*N t*) **if** *t* ∈ *A* **for** *t*
proof −
obtain *n k*::*nat* **where** *t* = *k*/*2*^*n* ∗ *dist x0 y0 k* ≤ *2*^*n* **using** ‹*t*∈ *A*› **unfolding** *A-def* **by** *auto*
then have ∗: ∃ *i*. *t* = *real* (*fst i*)/*2*^(*snd i*) ∗ *dist x0 y0* ∧ (*fst i*) ≤ *2*^(*snd i*) **by** *auto*
show *?thesis* **unfolding** *K-def N-def index-def* **using** *someI-ex*[*OF* ∗] **by** *auto*
qed

We can now define our function on dyadic points.

define *f* **where** *f* = (λ*t*. *geod* (*N t*) (*K t*))
have *0* ∈ *A* **unfolding** *A-def* **by** *auto*
have *f 0* = *x0*
proof −

```
    have 0 = K 0 /2^(N 0) * dist x0 y0 using t ‹0 ∈ A› by auto
    then have K 0 = 0 using False by auto
    then show ?thesis unfolding f-def using geod-0 by auto
  qed
  have dist x0 y0 = (real 1/2^0) * dist x0 y0 by auto
  then have dist x0 y0 ∈ A unfolding A-def by force
  have f (dist x0 y0) = y0
  proof -
    have dist x0 y0 = K (dist x0 y0) / 2^(N (dist x0 y0)) * dist x0 y0
      using t ‹dist x0 y0 ∈ A› by auto
   then have K (dist x0 y0) = 2^(N(dist x0 y0)) using False by (auto simp add:
divide-simps)
    then show ?thesis unfolding f-def using geod-end by auto
  qed
```

By construction, it is an isometry on dyadic points.

```
  have isometry-on A f
  proof (rule isometry-onI)
    fix s t assume inA: s ∈ A t ∈ A
    have dist (f s) (f t) = abs (K t/2^(N t) − K s/2^(N s)) * dist x0 y0
      unfolding f-def apply (rule geod-dist3) using t inA by auto
    also have ... = abs(K t/2^(N t) * dist x0 y0 − K s/2^(N s) * dist x0 y0)
      by (auto simp add: abs-mult-pos left-diff-distrib)
    also have ... = abs(t − s)
      using t inA by auto
    finally show dist (f s) (f t) = dist s t unfolding dist-real-def by auto
  qed
```

We can thus extend it to an isometry on the closure of dyadic points. It is the desired geodesic.

```
  then obtain g where g: isometry-on (closure A) g ⋀t. t ∈ A ⟹ g t = f t
    using isometry-extend-closure by metis
  have isometry-on {0..dist x0 y0} g
   by (rule isometry-on-subset[OF ‹isometry-on (closure A) g› ‹{0..dist x0 y0} ⊆
closure A›])
  moreover have g 0 = x0
    using g(2)[OF ‹0 ∈ A›] ‹f 0 = x0› by simp
  moreover have g (dist x0 y0) = y0
    using g(2)[OF ‹dist x0 y0 ∈ A›] ‹f (dist x0 y0) = y0› by simp
  ultimately show ?thesis by auto
qed

end
```

We can now complete the proof that a complete space with middles is in fact geodesic: all the work has been done in the locale `complete_space_with_middle`, in Lemma `geod`.

theorem *complete-with-middles-imp-geodesic*:

assumes $\bigwedge x\ y{::}('a{::}complete\text{-}space).\ \exists\, m.\ dist\ x\ m = dist\ x\ y\ /2 \wedge dist\ m\ y = dist\ x\ y\ /2$
shows *OFCLASS*('a, *geodesic-space-class*)
proof (*standard*, *rule geodesic-subsetI*)
fix *x0 y0*::'a
interpret *complete-space-with-middle x0 y0*
apply *standard* **using** *assms* **by** *auto*
have $\exists\, g.\ g\ 0 = x0 \wedge g\ (dist\ x0\ y0) = y0 \wedge isometry\text{-}on\ \{0..dist\ x0\ y0\}\ g$
using *geod* **by** *auto*
then show $\exists\, G.\ geodesic\text{-}segment\text{-}between\ G\ x0\ y0 \wedge G \subseteq UNIV$
unfolding *geodesic-segment-between-def* **by** *auto*
qed

6 Quasi-isometries

A (λ, C) quasi-isometry is a function which behaves like an isometry, up to an additive error C and a multiplicative error λ. It can be very different from an isometry on small scales (for instance, the function integer part is a quasi-isometry between $\mathbb{R}$ and $\mathbb{Z}$), but on large scales it captures many important features of isometries.

When the space is unbounded, one checks easily that $C \geq 0$ and $\lambda \geq 1$. As this is the only case of interest (any two bounded sets are quasi-isometric), we incorporate this requirement in the definition.

definition *quasi-isometry-on*::*real* ⇒ *real* ⇒ ('a::*metric-space*) *set* ⇒ ('a ⇒ ('b::*metric-space*)) ⇒ *bool*
(‹- - −*quasi*'-*isometry*'-*on*› [*1000*, *999*])
where $lambda\ C{-}quasi\text{-}isometry\text{-}on\ X\ f = ((lambda \geq 1) \wedge (C \geq 0) \wedge (\forall\, x \in X.\ \forall\, y \in X.\ (dist\ (f\ x)\ (f\ y) \leq lambda * dist\ x\ y + C \wedge dist\ (f\ x)\ (f\ y) \geq (1/lambda) * dist\ x\ y - C)))$

abbreviation *quasi-isometry* :: *real* ⇒ *real* ⇒ ('a::*metric-space* ⇒ 'b::*metric-space*) ⇒ *bool*
(‹- - −*quasi*'-*isometry*› [*1000*, *999*])
where $quasi\text{-}isometry\ lambda\ C\ f \equiv lambda\ C{-}quasi\text{-}isometry\text{-}on\ UNIV\ f$

6.1 Basic properties of quasi-isometries

lemma *quasi-isometry-onD*:
assumes $lambda\ C{-}quasi\text{-}isometry\text{-}on\ X\ f$
shows $\bigwedge x\ y.\ x \in X \Longrightarrow y \in X \Longrightarrow dist\ (f\ x)\ (f\ y) \leq lambda * dist\ x\ y + C$
$\bigwedge x\ y.\ x \in X \Longrightarrow y \in X \Longrightarrow dist\ (f\ x)\ (f\ y) \geq (1/lambda) * dist\ x\ y - C$
$lambda \geq 1\ C \geq 0$
using *assms* **unfolding** *quasi-isometry-on-def* **by** *auto*

lemma *quasi-isometry-onI* [*intro*]:
assumes $\bigwedge x\ y.\ x \in X \Longrightarrow y \in X \Longrightarrow dist\ (f\ x)\ (f\ y) \leq lambda * dist\ x\ y + C$
$\bigwedge x\ y.\ x \in X \Longrightarrow y \in X \Longrightarrow dist\ (f\ x)\ (f\ y) \geq (1/lambda) * dist\ x\ y - C$

        lambda $\geq$ *1 C* $\geq$ *0*
  shows *lambda C−quasi-isometry-on X f*
using *assms* **unfolding** *quasi-isometry-on-def* **by** *auto*

lemma *isometry-quasi-isometry-on*:
  assumes *isometry-on X f*
  shows *1 0−quasi-isometry-on X f*
using *assms* **unfolding** *isometry-on-def quasi-isometry-on-def* **by** *auto*

lemma *quasi-isometry-on-change-params*:
  assumes *lambda C−quasi-isometry-on X f mu* $\geq$ *lambda D* $\geq$ *C*
  shows *mu D−quasi-isometry-on X f*
proof (*rule quasi-isometry-onI*)
  have *P1*: *lambda* $\geq$ *1 C* $\geq$ *0* **using** *quasi-isometry-onD*[*OF assms*(*1*)] **by** *auto*
  then show *P2*: *mu* $\geq$ *1 D* $\geq$ *0* **using** *assms* **by** *auto*
  fix *x y* **assume** *inX*: *x* $\in$ *X y* $\in$ *X*
  have *dist* (*f x*) (*f y*) $\leq$ *lambda* $*$ *dist x y* + *C*
    using *quasi-isometry-onD*[*OF assms*(*1*)] *inX* **by** *auto*
  also have ... $\leq$ *mu* $*$ *dist x y* + *D*
    using *assms* **by** (*auto intro*!: *mono-intros*)
  finally show *dist* (*f x*) (*f y*) $\leq$ *mu* $*$ *dist x y* + *D* **by** *simp*
  have *dist* (*f x*) (*f y*) $\geq$ (*1*/*lambda*) $*$ *dist x y* − *C*
    using *quasi-isometry-onD*[*OF assms*(*1*)] *inX* **by** *auto*
  moreover have (*1*/*lambda*) $*$ *dist x y* + (− *C*) $\geq$ (*1*/*mu*) $*$ *dist x y* + (− *D*)
    apply (*intro mono-intros*)
    using *P1 P2 assms* **by** (*auto simp add*: *divide-simps*)
  ultimately show *dist* (*f x*) (*f y*) $\geq$ (*1*/*mu*) $*$ *dist x y* − *D* **by** *simp*
qed

lemma *quasi-isometry-on-subset*:
  assumes *lambda C−quasi-isometry-on X f*
        Y $\subseteq$ *X*
  shows *lambda C−quasi-isometry-on Y f*
using *assms* **unfolding** *quasi-isometry-on-def* **by** *auto*

lemma *quasi-isometry-on-perturb*:
  assumes *lambda C−quasi-isometry-on X f*
        D $\geq$ *0*
        $\bigwedge$*x*. *x* $\in$ *X* $\Longrightarrow$ *dist* (*f x*) (*g x*) $\leq$ *D*
  shows *lambda* (*C* + *2* $*$ *D*)*−quasi-isometry-on X g*
proof (*rule quasi-isometry-onI*)
  show *lambda* $\geq$ *1 C* + *2* $*$ *D* $\geq$ *0* **using** ‹*D* $\geq$ *0*› *quasi-isometry-onD*[*OF assms*(*1*)] **by** *auto*
  fix *x y* **assume** $*$: *x* $\in$ *X y* $\in$ *X*
  have *dist* (*g x*) (*g y*) $\leq$ *dist* (*f x*) (*f y*) + *2* $*$ *D*
    using *assms*(*3*)[*OF* $*$(*1*)] *assms*(*3*)[*OF* $*$(*2*)] *dist-triangle4*[*of g x g y f x f y*] **by** (*simp add*: *dist-commute*)
  then show *dist* (*g x*) (*g y*) $\leq$ *lambda* $*$ *dist x y* + (*C* + *2* $*$ *D*)
    using *quasi-isometry-onD*(*1*)[*OF assms*(*1*) $*$] **by** *auto*

have *dist* $(g\ x)\ (g\ y) \geq$ *dist* $(f\ x)\ (f\ y) - 2 * D$
using *assms*(*3*)[*OF* ∗(*1*)] *assms*(*3*)[*OF* ∗(*2*)] *dist-triangle4*[*of f x f y g x g y*]
by (*simp add*: *dist-commute*)
then show *dist* $(g\ x)\ (g\ y) \geq (1/lambda) *$ *dist* $x\ y - (C + 2 * D)$
using *quasi-isometry-onD*(*2*)[*OF assms*(*1*) ∗] **by** *auto*
qed

lemma *quasi-isometry-on-compose*:
assumes *lambda C*−*quasi-isometry-on X f*
mu D−*quasi-isometry-on Y g*
f‘X ⊆ *Y*
shows (*lambda* ∗ *mu*) (*C* ∗ *mu* + *D*)−*quasi-isometry-on X* (*g o f*)
proof (*rule quasi-isometry-onI*)
have *I*: *lambda* ≥ *1 C* ≥ *0 mu* ≥ *1 D* ≥ *0*
using *quasi-isometry-onD*[*OF assms*(*1*)] *quasi-isometry-onD*[*OF assms*(*2*)] **by** *auto*
then show *lambda* ∗ *mu* ≥ *1 C* ∗ *mu* + *D* ≥ *0*
by (*auto*, *metis dual-order.order-iff-strict le-numeral-extra*(*2*) *mult-le-cancel-right1 order.strict-trans1*)
fix *x y* **assume** *inX*: *x* ∈ *X y* ∈ *X*
then have *inY*: *f x* ∈ *Y f y* ∈ *Y* **using** ‹*f‘X* ⊆ *Y*› **by** *auto*
have *dist* ((*g o f*) *x*) ((*g o f*) *y*) ≤ *mu* ∗ *dist* (*f x*) (*f y*) + *D*
using *quasi-isometry-onD*(*1*)[*OF assms*(*2*) *inY*] **by** *simp*
also have ... ≤ *mu* ∗ (*lambda* ∗ *dist x y* + *C*) + *D*
using ‹*mu* ≥ *1*› *quasi-isometry-onD*(*1*)[*OF assms*(*1*) *inX*] **by** *auto*
finally show *dist* ((*g o f*) *x*) ((*g o f*) *y*) ≤ (*lambda* ∗ *mu*) ∗ *dist x y* + (*C* ∗ *mu* + *D*)
by (*auto simp add*: *algebra-simps*)

have (*1*/(*lambda* ∗ *mu*)) ∗ *dist x y* − (*C* ∗ *mu* + *D*) ≤ (*1*/(*lambda* ∗ *mu*)) ∗ *dist x y* − (*C*/*mu* + *D*)
using ‹*mu* ≥ *1*› ‹*C* ≥ *0*› **apply** (*auto*, *auto simp add*: *divide-simps*)
by (*metis eq-iff less-eq-real-def mult.commute mult-eq-0-iff mult-le-cancel-right1 order.trans*)
also have ... = (*1*/*mu*) ∗ ((*1*/*lambda*) ∗ *dist x y* − *C*) − *D*
by (*auto simp add*: *algebra-simps*)
also have ... ≤ (*1*/*mu*) ∗ *dist* (*f x*) (*f y*) − *D*
using ‹*mu* ≥ *1*› *quasi-isometry-onD*(*2*)[*OF assms*(*1*) *inX*] **by** (*auto simp add*: *divide-simps*)
also have ... ≤ *dist* ((*g o f*) *x*) ((*g o f*) *y*)
using *quasi-isometry-onD*(*2*)[*OF assms*(*2*) *inY*] **by** *auto*
finally show *1* / (*lambda* ∗ *mu*) ∗ *dist x y* − (*C* ∗ *mu* + *D*) ≤ *dist* ((*g* ∘ *f*) *x*) ((*g* ∘ *f*) *y*)
by *auto*
qed

lemma *quasi-isometry-on-bounded*:
assumes *lambda C*−*quasi-isometry-on X f*
bounded X

shows *bounded* $(f`X)$
proof (*cases* $X = \{\}$)
case *True*
then show *?thesis* **by** *auto*
next
case *False*
obtain x **where** $x \in X$ **using** *False* **by** *auto*
obtain e **where** e: $\bigwedge z.\ z \in X \Longrightarrow dist\ x\ z \leq e$
using *bounded-any-center assms(2)* **by** *metis*
have $dist\ (f\ x)\ y \leq C + lambda * e$ **if** $y \in f`X$ **for** y
proof −
obtain z **where** $*$: $z \in X\ y = f\ z$ **using** ‹$y \in f`X$› **by** *auto*
have $dist\ (f\ x)\ y \leq lambda * dist\ x\ z + C$
unfolding ‹$y = f\ z$› **using** $*$ *quasi-isometry-onD(1)[OF assms(1)* ‹$x \in X$› ‹$z \in X$›*]* **by** (*auto simp add: add-mono*)
also have ... $\leq C + lambda * e$ **using** *e[OF* ‹$z \in X$›*] quasi-isometry-onD(3)[OF assms(1)]* **by** *auto*
finally show *?thesis* **by** *simp*
qed
then show *?thesis* **unfolding** *bounded-def* **by** *auto*
qed

lemma *quasi-isometry-on-empty*:
assumes $C \geq 0$ $lambda \geq 1$
shows *lambda C−quasi-isometry-on* $\{\}$ f
using *assms* **unfolding** *quasi-isometry-on-def* **by** *auto*

Quasi-isometries change the distance to a set by at most $\lambda \cdot +C$, this follows readily from the fact that this inequality holds pointwise.

lemma *quasi-isometry-on-infdist*:
assumes *lambda C−quasi-isometry-on X f*
$w \in X$
$S \subseteq X$
shows $infdist\ (f\ w)\ (f`S) \leq lambda * infdist\ w\ S + C$
$infdist\ (f\ w)\ (f`S) \geq (1/lambda) * infdist\ w\ S - C$
proof −
have $lambda \geq 1$ $C \geq 0$ **using** *quasi-isometry-onD[OF assms(1)]* **by** *auto*
show $infdist\ (f\ w)\ (f`S) \leq lambda * infdist\ w\ S + C$
proof (*cases* $S = \{\}$)
case *True*
then show *?thesis*
using ‹$C \geq 0$› **unfolding** *infdist-def* **by** *auto*
next
case *False*
then have $(INF\ x{\in}S.\ dist\ (f\ w)\ (f\ x)) \leq (INF\ x{\in}S.\ lambda * dist\ w\ x + C)$
apply (*rule cINF-superset-mono*)
apply (*meson bdd-belowI2 zero-le-dist*) **using** *assms* **by** (*auto intro!: quasi-isometry-onD(1)[OF assms(1)]*)
also have ... $= (INF\ t{\in}(dist\ w)`S.\ lambda * t + C)$

by (*auto simp add*: *image-comp*)
also have ... = *lambda* ∗ *Inf* ((*dist w*)‘*S*) + *C*
apply (*rule continuous-at-Inf-mono*[*symmetric*])
unfolding *mono-def* **using** ‹*lambda* ≥ *1*› *False* **by** (*auto intro*!: *continuous-intros*)
finally show *?thesis* **unfolding** *infdist-def* **using** *False* **by** (*auto simp add*: *image-comp*)
qed
show *1* / *lambda* ∗ *infdist w S* − *C* ≤ *infdist* (*f w*) (*f* ‘ *S*)
proof (*cases S* = {})
case *True*
then show *?thesis*
using ‹*C* ≥ *0*› **unfolding** *infdist-def* **by** *auto*
next
case *False*
then have (*1*/*lambda*) ∗ *infdist w S* − *C* = (*1*/*lambda*) ∗ *Inf* ((*dist w*)‘*S*) − *C*
unfolding *infdist-def* **by** *auto*
also have ... = (*INF t*∈(*dist w*)‘*S*. (*1*/*lambda*) ∗ *t* − *C*)
apply (*rule continuous-at-Inf-mono*)
unfolding *mono-def* **using** ‹*lambda* ≥ *1*› *False* **by** (*auto simp add*: *divide-simps intro*!: *continuous-intros*)
also have ... = (*INF x*∈*S*. (*1*/*lambda*) ∗ *dist w x* − *C*)
by (*auto simp add*: *image-comp*)
also have ... ≤ (*INF x*∈*S*. *dist* (*f w*) (*f x*))
apply (*rule cINF-superset-mono*[*OF False*]) **apply** (*rule bdd-belowI2*[*of* - −*C*])
using *assms* ‹*lambda* ≥ *1*› **apply** *simp* **apply** *simp* **apply** (*rule quasi-isometry-onD*(*2*)[*OF assms*(*1*)])
using *assms* **by** *auto*
finally show *?thesis* **unfolding** *infdist-def* **using** *False* **by** (*auto simp add*: *image-comp*)
qed
qed

6.2 Quasi-isometric isomorphisms

The notion of isomorphism for quasi-isometries is not that it should be a bijection, as it is a coarse notion, but that it is a bijection up to a bounded displacement. For instance, the inclusion of ℤ in ℝ is a quasi-isometric isomorphism between these spaces, whose (quasi)-inverse (which is non-unique) is given by the function integer part. This is formalized in the next definition.

definition *quasi-isometry-between*::*real* ⇒ *real* ⇒ (′*a*::*metric-space*) *set* ⇒ (′*b*::*metric-space*) *set* ⇒ (′*a* ⇒ ′*b*) ⇒ *bool*
(‹- - −*quasi*′-*isometry*′-*between*› [*1000*, *999*])
where *lambda C*−*quasi-isometry-between X Y f* = ((*lambda C*−*quasi-isometry-on X f*) ∧ (*f*‘*X* ⊆ *Y*) ∧ (∀ *y*∈*Y*. ∃ *x*∈*X*. *dist* (*f x*) *y* ≤ *C*))

definition *quasi-isometric*::($'a$::*metric-space*) *set* $\Rightarrow$ ($'b$::*metric-space*) *set* $\Rightarrow$ *bool*
where *quasi-isometric X Y* = ($\exists$ *lambda C f*. *lambda C*−*quasi-isometry-between X Y f*)

lemma *quasi-isometry-betweenD*:
assumes *lambda C*−*quasi-isometry-between X Y f*
shows *lambda C*−*quasi-isometry-on X f*
$f`X \subseteq Y$
$\bigwedge y.\ y \in Y \Longrightarrow \exists x{\in}X.\ dist\ (f\ x)\ y \le C$
$\bigwedge x\ y.\ x \in X \Longrightarrow y \in X \Longrightarrow dist\ (f\ x)\ (f\ y) \le lambda * dist\ x\ y + C$
$\bigwedge x\ y.\ x \in X \Longrightarrow y \in X \Longrightarrow dist\ (f\ x)\ (f\ y) \ge (1/lambda) * dist\ x\ y - C$
$lambda \ge 1\ C \ge 0$
using *assms* **unfolding** *quasi-isometry-between-def quasi-isometry-on-def* **by** *auto*

lemma *quasi-isometry-betweenI*:
assumes *lambda C*−*quasi-isometry-on X f*
$f`X \subseteq Y$
$\bigwedge y.\ y \in Y \Longrightarrow \exists x{\in}X.\ dist\ (f\ x)\ y \le C$
shows *lambda C*−*quasi-isometry-between X Y f*
using *assms* **unfolding** *quasi-isometry-between-def* **by** *auto*

lemma *quasi-isometry-on-between*:
assumes *lambda C*−*quasi-isometry-on X f*
shows *lambda C*−*quasi-isometry-between X* ($f`X$) *f*
using *assms* **unfolding** *quasi-isometry-between-def quasi-isometry-on-def* **by** *force*

lemma *quasi-isometry-between-change-params*:
assumes *lambda C*−*quasi-isometry-between X Y f* $mu \ge lambda\ D \ge C$
shows *mu D*−*quasi-isometry-between X Y f*
proof (*rule quasi-isometry-betweenI*)
show *mu D*−*quasi-isometry-on X f*
by (*rule quasi-isometry-on-change-params*[*OF quasi-isometry-betweenD*(*1*)[*OF assms*(*1*)] *assms*(*2*) *assms*(*3*)])
show $f`X \subseteq Y$ **using** *quasi-isometry-betweenD*[*OF assms*(*1*)] **by** *auto*
fix *y* **assume** $y \in Y$
show $\exists x{\in}X.\ dist\ (f\ x)\ y \le D$ **using** *quasi-isometry-betweenD*(*3*)[*OF assms*(*1*) ‹$y \in Y$›] ‹$D \ge C$› **by** *force*
qed

lemma *quasi-isometry-subset*:
assumes $X \subseteq Y\ \bigwedge y.\ y \in Y \Longrightarrow \exists x{\in}X.\ dist\ x\ y \le C\ C \ge 0$
shows *1 C*−*quasi-isometry-between X Y* ($\lambda x.\ x$)
unfolding *quasi-isometry-between-def* **using** *assms* **by** *auto*

lemma *isometry-quasi-isometry-between*:
assumes *isometry f*
shows *1 0*−*quasi-isometry-between UNIV UNIV f*
using *assms* **unfolding** *quasi-isometry-between-def quasi-isometry-on-def isometry-def isometry-on-def surj-def* **by** (*auto*) *metis*

```
proposition quasi-isometry-inverse:
  assumes lambda C−quasi-isometry-between X Y f
  shows ∃ g. lambda (3 ∗ C ∗ lambda)−quasi-isometry-between Y X g
          ∧ (∀ x∈X. dist x (g (f x)) ≤ 3 ∗ C ∗ lambda)
          ∧ (∀ y∈Y. dist y (f (g y)) ≤ 3 ∗ C ∗ lambda)
proof −
  define g where g = (λy. SOME x. x ∈ X ∧ dist (f x) y ≤ C)
  have ∗: g y ∈ X ∧ dist (f (g y)) y ≤ C if y ∈ Y for y
    unfolding g-def using quasi-isometry-betweenD(3)[OF assms that] by (metis
(no-types, lifting) someI-ex)
  have lambda ≥ 1 C ≥ 0 using quasi-isometry-betweenD[OF assms] by auto

  have C ≤ 3 ∗ C ∗ lambda using ‹lambda ≥ 1› ‹C ≥ 0›
    by (simp add: algebra-simps mult-ge1-mono)
  then have A: dist y (f (g y)) ≤ 3 ∗ C ∗ lambda if y ∈ Y for y
    using ∗[OF that] by (simp add: dist-commute)

  have B: dist x (g (f x)) ≤ 3 ∗ C ∗ lambda if x ∈ X for x
  proof −
    have f x ∈ Y using that quasi-isometry-betweenD(2)[OF assms] by auto
    have (1/lambda) ∗ dist x (g (f x)) − C ≤ dist (f x) (f (g (f x)))
      apply (rule quasi-isometry-betweenD(5)[OF assms]) using that ∗[OF ‹f x ∈
Y›] by auto
    also have ... ≤ C using ∗[OF ‹f x ∈ Y›] by (simp add: dist-commute)
    finally have dist x (g (f x)) ≤ 2 ∗ C ∗ lambda
      using ‹lambda ≥ 1› ‹C ≥ 0› by (simp add: divide-simps)
    also have ... ≤ 3 ∗ C ∗ lambda
      using ‹lambda ≥ 1› ‹C ≥ 0› by (simp add: divide-simps)
    finally show ?thesis by auto
  qed

  have lambda (3 ∗ C ∗ lambda)−quasi-isometry-on Y g
  proof (rule quasi-isometry-onI)
    show lambda ≥ 1 3 ∗ C ∗ lambda ≥ 0 using ‹lambda ≥ 1› ‹C ≥ 0› by auto
    fix y1 y2 assume inY: y1 ∈ Y y2 ∈ Y
    then have inX: g y1 ∈ X g y2 ∈ X using ∗ by auto
    have dist y1 y2 ≤ dist y1 (f (g y1)) + dist (f (g y1)) (f (g y2)) + dist (f (g
y2)) y2
      using dist-triangle4 by auto
    also have ... ≤ C + dist (f (g y1)) (f (g y2)) + C
       using ∗[OF inY(1)] ∗[OF inY(2)] by (auto simp add: dist-commute intro:
add-mono)
    also have ... ≤ C + (lambda ∗ dist (g y1) (g y2) + C) + C
      using quasi-isometry-betweenD(4)[OF assms inX] by (auto intro: add-mono)
    finally have dist y1 y2 − 3 ∗ C ≤ lambda ∗ dist (g y1) (g y2) by auto
    then have dist (g y1) (g y2) ≥ (1/lambda) ∗ dist y1 y2 − 3 ∗ C / lambda
      using ‹lambda ≥ 1› by (auto simp add: divide-simps mult.commute)
    moreover have 3 ∗ C / lambda ≤ 3 ∗ C ∗ lambda
```

using ‹$lambda \geq 1$› ‹$C \geq 0$› **apply** (*auto simp add*: *divide-simps mult-le-cancel-left1*)
by (*metis dual-order.order-iff-strict less-1-mult mult.left-neutral*)
ultimately show $dist\ (g\ y1)\ (g\ y2) \geq (1/lambda) * dist\ y1\ y2 - 3 * C * lambda$
by *auto*

have $(1/lambda) * dist\ (g\ y1)\ (g\ y2) - C \leq dist\ (f\ (g\ y1))\ (f\ (g\ y2))$
using *quasi-isometry-betweenD*(5)[*OF assms inX*] **by** *auto*
also have ... $\leq dist\ (f\ (g\ y1))\ y1 + dist\ y1\ y2 + dist\ y2\ (f\ (g\ y2))$
using *dist-triangle4* **by** *auto*
also have ... $\leq C + dist\ y1\ y2 + C$
using $*$[*OF inY*(1)] $*$[*OF inY*(2)] **by** (*auto simp add*: *dist-commute intro*: *add-mono*)
finally show $dist\ (g\ y1)\ (g\ y2) \leq lambda * dist\ y1\ y2 + 3 * C * lambda$
using ‹$lambda \geq 1$› **by** (*auto simp add*: *divide-simps algebra-simps*)
qed
then have $lambda\ (3 * C * lambda)-quasi\text{-}isometry\text{-}between\ Y\ X\ g$
proof (*rule quasi-isometry-betweenI*)
show $g\ `\ Y \subseteq X$ **using** $*$ **by** *auto*
fix x **assume** $x \in X$
have $f\ x \in Y\ dist\ (g\ (f\ x))\ x \leq 3 * C * lambda$
using *B*[*OF* ‹$x \in X$›] *quasi-isometry-betweenD*(2)[*OF assms*] ‹$x \in X$› **by** (*auto simp add*: *dist-commute*)
then show $\exists y \in Y.\ dist\ (g\ y)\ x \leq 3 * C * lambda$ **by** *blast*
qed
then show *?thesis* **using** *A B* **by** *blast*
qed

proposition *quasi-isometry-compose*:
assumes $lambda\ C-quasi\text{-}isometry\text{-}between\ X\ Y\ f$
$mu\ D-quasi\text{-}isometry\text{-}between\ Y\ Z\ g$
shows $(lambda * mu)\ (C * mu + 2 * D)-quasi\text{-}isometry\text{-}between\ X\ Z\ (g\ o\ f)$
proof (*rule quasi-isometry-betweenI*)
have $(lambda * mu)\ (C * mu + D)-quasi\text{-}isometry\text{-}on\ X\ (g \circ f)$
by (*rule quasi-isometry-on-compose*[*OF quasi-isometry-betweenD*(1)[*OF assms*(1)] *quasi-isometry-betweenD*(1)[*OF assms*(2)] *quasi-isometry-betweenD*(2)[*OF assms*(1)]]])
then show $(lambda * mu)\ (C * mu + 2 * D)-quasi\text{-}isometry\text{-}on\ X\ (g \circ f)$
apply (*rule quasi-isometry-on-change-params*) **using** *quasi-isometry-betweenD*(7)[*OF assms*(2)] **by** *auto*

show $(g \circ f)\ `\ X \subseteq Z$
using *quasi-isometry-betweenD*(2)[*OF assms*(1)] *quasi-isometry-betweenD*(2)[*OF assms*(2)]
by *auto*
fix z **assume** $z \in Z$
obtain y **where** y: $y \in Y\ dist\ (g\ y)\ z \leq D$
using *quasi-isometry-betweenD*(3)[*OF assms*(2) ‹$z \in Z$›] **by** *auto*
obtain x **where** x: $x \in X\ dist\ (f\ x)\ y \leq C$

using *quasi-isometry-betweenD*(*3*)[*OF assms*(*1*) ‹*y* ∈ *Y*›] **by** *auto*
have *dist* ((*g o f*) *x*) *z* ≤ *dist* (*g* (*f x*)) (*g y*) + *dist* (*g y*) *z*
using *dist-triangle* **by** *auto*
also have ... ≤ (*mu* ∗ *dist* (*f x*) *y* + *D*) + *D*
apply (*rule add-mono*, *rule quasi-isometry-betweenD*(*4*)[*OF assms*(*2*)])
using *x y quasi-isometry-betweenD*(*2*)[*OF assms*(*1*)] **by** *auto*
also have ... ≤ *C* ∗ *mu* + *2* ∗ *D*
using *x*(*2*) *quasi-isometry-betweenD*(*6*)[*OF assms*(*2*)] **by** *auto*
finally show ∃ *x*∈*X*. *dist* ((*g* ∘ *f*) *x*) *z* ≤ *C* ∗ *mu* + *2* ∗ *D*
using *x*(*1*) **by** *auto*
qed

theorem *quasi-isometric-equiv-rel*:
quasi-isometric X X
quasi-isometric X Y ⟹ *quasi-isometric Y Z* ⟹ *quasi-isometric X Z*
quasi-isometric X Y ⟹ *quasi-isometric Y X*
proof −
show *quasi-isometric X X*
unfolding *quasi-isometric-def* **using** *quasi-isometry-subset*[*of X X 0*] **by** *auto*
assume *H*: *quasi-isometric X Y*
then show *quasi-isometric Y X*
unfolding *quasi-isometric-def* **using** *quasi-isometry-inverse* **by** *blast*
assume *quasi-isometric Y Z*
then show *quasi-isometric X Z*
using *H* **unfolding** *quasi-isometric-def* **using** *quasi-isometry-compose* **by** *blast*
qed

Many interesting properties in geometric group theory are invariant under quasi-isometry. We prove the most basic ones here.

lemma *quasi-isometric-empty*:
assumes *X* = {} *quasi-isometric X Y*
shows *Y* = {}
using *assms* **unfolding** *quasi-isometric-def quasi-isometry-between-def quasi-isometry-on-def*
by *blast*

lemma *quasi-isometric-bounded*:
assumes *bounded X quasi-isometric X Y*
shows *bounded Y*
proof (*cases X* = {})
case *True*
show *?thesis* **using** *quasi-isometric-empty*[*OF True assms*(*2*)] **by** *auto*
next
case *False*
obtain *lambda C f* **where** *QI*: *lambda C*−*quasi-isometry-between X Y f*
using *assms*(*2*) **unfolding** *quasi-isometric-def* **by** *auto*
obtain *x* **where** *x* ∈ *X* **using** *False* **by** *auto*
obtain *e* **where** *e*: ⋀*z*. *z* ∈ *X* ⟹ *dist x z* ≤ *e*
using *bounded-any-center assms*(*1*) **by** *metis*
have *dist* (*f x*) *y* ≤ *2* ∗ *C* + *lambda* ∗ *e* **if** *y* ∈ *Y* **for** *y*

```
  proof -
    obtain z where *: z ∈ X dist (f z) y ≤ C
      using quasi-isometry-betweenD(3)[OF QI ‹y ∈ Y›] by auto
    have dist (f x) y ≤ dist (f x) (f z) + dist (f z) y using dist-triangle by auto
    also have ... ≤ (lambda * dist x z + C) + C
      using * quasi-isometry-betweenD(4)[OF QI ‹x ∈ X› ‹z ∈ X›] by (auto simp
add: add-mono)
    also have ... ≤ 2 * C + lambda * e
      using quasi-isometry-betweenD(6)[OF QI] e[OF ‹z ∈ X›] by (auto simp add:
algebra-simps)
    finally show ?thesis by simp
  qed
  then show ?thesis unfolding bounded-def by auto
qed

lemma quasi-isometric-bounded-iff:
  assumes bounded X X ≠ {} bounded Y Y ≠ {}
  shows quasi-isometric X Y
proof -
  obtain x y where x ∈ X y ∈ Y using assms by auto
  obtain C where C: ⋀z. z ∈ Y ⟹ dist y z ≤ C
    using ‹bounded Y› bounded-any-center by metis
  have C ≥ 0 using C[OF ‹y ∈ Y›] by auto
  obtain D where D: ⋀z. z ∈ X ⟹ dist x z ≤ D
    using ‹bounded X› bounded-any-center by metis
  have D ≥ 0 using D[OF ‹x ∈ X›] by auto

  define f::'a ⇒ 'b where f = (λ-. y)
  have 1 (C + 2 * D)-quasi-isometry-between X Y f
  proof (rule quasi-isometry-betweenI)
    show f`X ⊆ Y unfolding f-def using ‹y ∈ Y› by auto
    show 1 (C + 2 * D)-quasi-isometry-on X f
    proof (rule quasi-isometry-onI, auto simp add: ‹C ≥ 0› ‹D ≥ 0› f-def)
      fix a b assume a ∈ X b ∈ X
      have dist a b ≤ dist a x + dist x b
        using dist-triangle by auto
      also have ... ≤ D + D
        using D[OF ‹a ∈ X›] D[OF ‹b ∈ X›] by (auto simp add: dist-commute)
      finally show dist a b ≤ C + 2 * D using ‹C ≥ 0› by auto
    qed
    show ∃ a∈X. dist (f a) z ≤ C + 2 * D if z ∈ Y for z
      unfolding f-def using ‹x ∈ X› C[OF ‹z ∈ Y›] ‹D ≥ 0› by auto
  qed
  then show ?thesis unfolding quasi-isometric-def by auto
qed
```

6.3 Quasi-isometries of Euclidean spaces.

A less trivial fact is that the dimension of euclidean spaces is invariant under quasi-isometries. It is proved below using growth argument, as quasi-isometries preserve the growth rate.

The growth of the space is asymptotic behavior of the number of well-separated points that fit in a ball of radius R, when R tends to infinity. Up to a suitable equivalence, it is clearly a quasi-isometry invariance. We show below that, in a Euclidean space of dimension d, the growth is like R^d: the upper bound is obtained by using the fact that we have disjoint balls inside a big ball, hence volume controls conclude the argument, while the lower bound is obtained by considering integer points.

First, we show that the growth rate of a Euclidean space of dimension d is bounded from above by R^d, using the control on measure of disjoint balls and a volume argument.

```
proposition growth-rate-euclidean-above:
  fixes D::real
  assumes D > (0::real)
    and H: F ⊆ cball (0::'a::euclidean-space) R R ≥ 0
        ⋀x y. x ∈ F ⟹ y ∈ F ⟹ x ≠ y ⟹ dist x y ≥ D
  shows finite F ∧ card F ≤ 1 + ((6/D)^(DIM('a))) ∗ R^(DIM('a))
proof −
  define C::real where C = ((6/D)^(DIM('a)))
  have C ≥ 0 unfolding C-def using ‹D > 0› by auto
  have D/3 ≥ 0 using assms by auto
  have finite F ∧ card F ≤ 1 + C ∗ R^(DIM('a))
  proof (cases R < D/2)
    case True
    have x = y if x ∈ F y ∈ F for x y
    proof (rule ccontr)
      assume ¬(x = y)
      then have D ≤ dist x y using H ‹x ∈ F› ‹y ∈ F› by auto
      also have ... ≤ dist x 0 + dist 0 y by (rule dist-triangle)
      also have ... ≤ R + R
        using H(1) ‹x ∈ F› ‹y ∈ F› by (intro add-mono, auto)
      also have ... < D using ‹R < D/2› by auto
      finally show False by simp
    qed
    then have finite F ∧ card F ≤ 1 using finite-at-most-singleton by auto
    moreover have 1 + 0 ∗ R^(DIM('a)) ≤ 1 + C ∗ R^(DIM('a))
      using ‹C ≥ 0› ‹R ≥ 0› by (auto intro: mono-intros)
    ultimately show ?thesis by auto
  next
    case False
    have card G ≤ 1 + C ∗ R^(DIM('a)) if G ⊆ F finite G for G
    proof −
      have norm y ≤ 2∗R if y ∈ cball x (D/3) x ∈ G for x y
```

proof −
  have *norm y* = *dist 0 y* **by** *auto*
  also have ... ≤ *dist 0 x* + *dist x y* **by** (*rule dist-triangle*)
  also have ... ≤ *R* + *D/3*
    using ‹*x* ∈ *G*› ‹*G* ⊆ *F*› ‹*y* ∈ *cball x* (*D/3*)› ‹*F* ⊆ *cball 0 R*› **by** (*auto intro*: *add-mono*)
  finally show *?thesis* **using** *False* ‹*D* > *0*› **by** *auto*
qed
then have *I*: (⋃*x*∈*G*. *cball x* (*D/3*)) ⊆ *cball 0* (*2*∗*R*)
  by *auto*
have *disjoint-family-on* (λ*x*. *cball x* (*D/3*)) *G*
  unfolding *disjoint-family-on-def* **proof** (*auto*)
  fix *a b x* **assume** ∗: *a* ∈ *G b* ∈ *G a* ≠ *b dist a x* ∗ *3* ≤ *D dist b x* ∗ *3* ≤ *D*
  then have *D* ≤ *dist a b* **using** *H* ‹*G* ⊆ *F*› **by** *auto*
  also have ... ≤ *dist a x* + *dist x b* **by** (*rule dist-triangle*)
  also have ... ≤ *D/3* + *D/3*
    using ∗ **by** (*auto simp add*: *dist-commute intro*: *mono-intros*)
  also have ... < *D* **using** ‹*D* > *0*› **by** *auto*
  finally show *False* **by** *simp*
qed

have *2* ∗ *R* ≥ *0* **using** ‹*R* ≥ *0*› **by** *auto*
define *A* **where** *A* = *measure lborel* (*cball* (*0*::′*a*) *1*)
have *A* > *0* **unfolding** *A-def* **using** *lebesgue-measure-ball-pos* **by** *auto*
have *card G* ∗ ((*D/3*) ^(*DIM*(′*a*)) ∗ *A*) = (∑*x*∈*G*. ((*D/3*) ^(*DIM*(′*a*)) ∗ *A*))
  by *auto*
also have ... = (∑*x*∈*G*. *measure lborel* (*cball x* (*D/3*)))
  unfolding *lebesgue-measure-ball*[*OF* ‹*D/3* ≥ *0*›] *A-def* **by** *auto*
also have ... = *measure lborel* (⋃*x*∈*G*. *cball x* (*D/3*))
apply (*rule measure-finite-Union*[*symmetric*, *OF* ‹*finite G*› - ‹*disjoint-family-on* (λ*x*. *cball x* (*D/3*)) *G*›])
  apply *auto* **using** *emeasure-bounded-finite less-imp-neq* **by** *auto*
also have ... ≤ *measure lborel* (*cball* (*0*::′*a*) (*2*∗*R*))
apply (*rule measure-mono-fmeasurable*) **using** *I* ‹*finite G*› *emeasure-bounded-finite*
  unfolding *fmeasurable-def* **by** *auto*
also have ... = (*2*∗*R*) ^(*DIM*(′*a*)) ∗ *A*
  unfolding *A-def* **using** *lebesgue-measure-ball*[*OF* ‹*2*∗*R* ≥ *0*›] **by** *auto*
finally have *card G* ∗ (*D/3*) ^(*DIM*(′*a*)) ≤ (*2*∗*R*) ^(*DIM*(′*a*))
  using ‹*A* > *0*› **by** (*auto simp add*: *divide-simps*)
then have *card G* ≤ *C* ∗ *R* ^(*DIM*(′*a*))
    unfolding *C-def* **using** ‹*D* > *0*› **apply** (*auto simp add*: *algebra-simps divide-simps*)
    by (*metis numeral-times-numeral power-mult-distrib semiring-norm*(*12*) *semiring-norm*(*14*))
then show *?thesis* **by** *auto*
qed
then show *finite F* ∧ *card F* ≤ *1* + *C* ∗ *R* ^(*DIM*(′*a*))
  by (*rule finite-finite-subset-caract*′)
qed

then show *?thesis* **unfolding** *C-def* **by** *blast*
qed

Then, we show that the growth rate of a Euclidean space of dimension d is bounded from below by R^d, using integer points.

proposition *growth-rate-euclidean-below*:
fixes D::*real*
assumes $R \geq 0$
shows $\exists F.\ (F \subseteq cball\ (0::'a::euclidean\text{-}space)\ R$
$\wedge (\forall x{\in}F.\ \forall y{\in}F.\ x = y \vee dist\ x\ y \geq D) \wedge finite\ F \wedge card\ F \geq (1/((max\ D\ 1) * DIM('a)))\widehat{\ }(DIM('a)) * R\widehat{\ }(DIM('a)))$
proof −
define E **where** $E = max\ D\ 1$
have $E > 0$ **unfolding** *E-def* **by** *auto*
define c **where** $c = (1/(E * DIM('a)))\widehat{\ }(DIM('a))$
have $c > 0$ **unfolding** *c-def* **using** ‹$E > 0$› **by** *auto*

define n **where** $n = nat\ (floor\ (R/(E * DIM('a)))) + 1$
then have $n > 0$ **using** ‹$R \geq 0$› **by** *auto*

have $R/(E * DIM('a)) \leq n$ **unfolding** *n-def* **by** *linarith*
then have $c * R\widehat{\ }(DIM('a)) \leq n\widehat{\ }(DIM('a))$
unfolding *c-def power-mult-distrib*[*symmetric*] **by** (*auto simp add*: ‹$0 < E$› ‹$0 \leq R$› *less-imp-le power-mono*)
have $n-1 \leq R/(E * DIM('a))$
unfolding *n-def* **using** ‹$R \geq 0$› ‹$E > 0$› **by** *auto*
then have $E * DIM('a) * (n-1) \leq R$
using ‹$R \geq 0$› ‹$E > 0$› **by** (*simp add*: *mult.commute pos-le-divide-eq*)

We want to consider the set of linear combinations of basis elements with integer coefficients bounded by n (multiplied by E to guarantee the D separation). The formal way to write these elements is to consider all the functions from the basis to $\{0, \dots, n-1\}$, and associate to such a function f the point $\sum Ef(i) \cdot i$ where the sum is over all basis elements i. This is what the next definition does.

define F::*'a set* **where** $F = (\lambda f.\ (\sum i{\in}Basis.\ (E * real\ (f\ i)) *_R i))\text{`}((Basis::('a\ set)) \rightarrow_E \{0..{<}n\})$

have $f = g$ **if** $f \in (Basis::('a\ set)) \rightarrow_E \{0..{<}n\}$ $g \in Basis \rightarrow_E \{0..{<}n\}$
$(\sum i{\in}Basis.\ (E * real\ (f\ i)) *_R i) = (\sum i{\in}Basis.\ (E * real\ (g\ i)) *_R i)$ **for** $f\ g$
proof (*rule ext*)
fix i **show** $f\ i = g\ i$
proof (*cases* $i \in Basis$)
case *True*
then have $E * real(f\ i) = E * real(g\ i)$
using *inner-sum-left-Basis*[*OF True*, *of* $\lambda i.\ E * real(f\ i)$] *inner-sum-left-Basis*[*OF True*, *of* $\lambda i.\ E * real(g\ i)$] *that*(*3*)

by *auto*
then show $f\ i = g\ i$ **using** ‹$E > 0$› **by** *auto*
next
case *False*
then have $f\ i = undefined\ g\ i = undefined$ **using** *that* **by** *auto*
then show $f\ i = g\ i$ **by** *auto*
qed
qed
then have $inj\text{-}on\ (\lambda f.\ (\sum i{\in}Basis.\ (E * real\ (f\ i)) *_R\ i))\ ((Basis::('a\ set)) \rightarrow_E \{0..<n\})$
by (*simp add*: *inj-onI*)
then have $card\ F = card\ ((Basis::('a\ set)) \rightarrow_E \{0..<n\})$ **unfolding** *F-def*
using *card-image* **by** *blast*
also have $... = n\widehat{\ }(DIM('a))$
unfolding *card-PiE*[*OF finite-Basis*] **by** (*auto simp add*: *prod-constant*)
finally have $card\ F = n\widehat{\ }(DIM('a))$ **by** *auto*
then have *finite F* **using** ‹$n > 0$›
using *card.infinite* **by** *force*
have $card\ F \geq c * R\widehat{\ }(DIM('a))$
using ‹$c * R\widehat{\ }(DIM('a)) \leq n\widehat{\ }(DIM('a))$› ‹$card\ F = n\widehat{\ }(DIM('a))$› **by** *auto*

have *separation*: $dist\ x\ y \geq D$ **if** $x \in F\ y \in F\ x \neq y$ **for** $x\ y$
proof −
obtain f **where** x: $f \in (Basis::('a\ set)) \rightarrow_E \{0..<n\}\ x = (\sum i{\in}Basis.\ (E * real\ (f\ i)) *_R\ i)$
using ‹$x \in F$› **unfolding** *F-def* **by** *auto*
obtain g **where** y: $g \in (Basis::('a\ set)) \rightarrow_E \{0..<n\}\ y = (\sum i{\in}Basis.\ (E * real\ (g\ i)) *_R\ i)$
using ‹$y \in F$› **unfolding** *F-def* **by** *auto*
obtain i **where** $f\ i \neq g\ i$ **using** $x\ y$ ‹$x \neq y$› **by** *force*
moreover have $f\ j = g\ j$ **if** $j \notin Basis$ **for** j
using $x(1)\ y(1)$ *that* **by** *fastforce*
ultimately have $i \in Basis$ **by** *auto*
have $D \leq E$ **unfolding** *E-def* **by** *auto*
also have $... \leq abs(E * (real\ (f\ i) - real\ (g\ i)))$ **using** ‹$E > 0$›
using ‹$f\ i \neq g\ i$› **by** (*auto simp add*: *divide-simps abs-mult*)
also have $... = abs(inner\ x\ i - inner\ y\ i)$
unfolding $x(2)\ y(2)$ *inner-sum-left-Basis*[*OF* ‹$i \in Basis$›] **by** (*auto simp add*: *algebra-simps*)
also have $... = abs(inner\ (x-y)\ i)$
by (*simp add*: *inner-diff-left*)
also have $... \leq norm\ (x-y)$ **using** *Basis-le-norm*[*OF* ‹$i \in Basis$›] **by** *blast*
finally show $dist\ x\ y \geq D$ **by** (*simp add*: *dist-norm*)
qed

have $norm\ x \leq R$ **if** $x \in F$ **for** x
proof −
obtain f **where** x: $f \in (Basis::('a\ set)) \rightarrow_E \{0..<n\}\ x = (\sum i{\in}Basis.\ (E * real\ (f\ i)) *_R\ i)$

using ‹$x \in F$› **unfolding** *F-def* **by** *auto*
then have $norm\ x = norm\ (\sum i{\in}Basis.\ (E * real\ (f\ i)) *_R\ i)$ **by** *simp*
also have ... $\leq (\sum i{\in}Basis.\ norm((E * real\ (f\ i)) *_R\ i))$
by (*rule norm-sum*)
also have ... $= (\sum i{\in}Basis.\ abs(E * real\ (f\ i)))$ **by** *auto*
also have ... $= (\sum i{\in}Basis.\ E * real\ (f\ i))$ **using** ‹$E > 0$› **by** *auto*
also have ... $\leq (\sum i{\in}(Basis{::}'a\ set).\ E * (n-1))$
apply (*rule sum-mono*) **using** *PiE-mem*[*OF x*(*1*)] ‹$E > 0$› **apply** (*auto simp add*: *divide-simps*)
using ‹$n > 0$› **by** *fastforce*
also have ... $= DIM('a) * E * (n-1)$
by *auto*
finally show $norm\ x \leq R$ **using** ‹$E * DIM('a) * (n-1) \leq R$› **by** (*auto simp add*: *algebra-simps*)
qed
then have $F \subseteq cball\ 0\ R$ **by** *auto*
then show *?thesis* **using** ‹$card\ F \geq c * R\,\widehat{}\,(DIM('a))$› ‹*finite F*› *separation c-def E-def* **by** *blast*
qed

As the growth is invariant under quasi-isometries, we deduce that it is impossible to map quasi-isometrically a Euclidean space in a space of strictly smaller dimension.

proposition *quasi-isometry-on-euclidean*:
fixes $f{::}'a{::}euclidean\text{-}space \Rightarrow 'b{::}euclidean\text{-}space$
assumes *lambda C−quasi-isometry-on UNIV f*
shows $DIM('a) \leq DIM('b)$
proof −
have C: $lambda \geq 1\ C \geq 0$ **using** *quasi-isometry-onD*[*OF assms*] **by** *auto*
define D **where** $D = lambda * (C+1)$
define Ca **where** $Ca = (1/((max\ D\ 1) * DIM('a)))\,\widehat{}\,(DIM('a))$
have $Ca > 0$ **unfolding** *Ca-def* **by** *auto*
have A: $\bigwedge R{::}real.\ R \geq 0 \Longrightarrow (\exists F.\ (F \subseteq cball\ (0{::}'a{::}euclidean\text{-}space)\ R \wedge (\forall x{\in}F.\ \forall y{\in}F.\ x = y \vee dist\ x\ y \geq D) \wedge finite\ F \wedge card\ F \geq Ca * R\,\widehat{}\,(DIM('a))))$
using *growth-rate-euclidean-below*[*of - D*] **unfolding** *Ca-def* **by** *blast*
define $Cb{::}real$ **where** $Cb = ((6/1)\,\widehat{}\,(DIM('b)))$
have B: $\bigwedge F\ (R{::}real).\ (F \subseteq cball\ (0{::}'b{::}euclidean\text{-}space)\ R \Longrightarrow R \geq 0 \Longrightarrow (\forall x{\in}F.\ \forall y{\in}F.\ x = y \vee dist\ x\ y \geq 1) \Longrightarrow (finite\ F \wedge card\ F \leq 1 + Cb * R\,\widehat{}\,(DIM('b))))$
using *growth-rate-euclidean-above*[*of 1*] **unfolding** *Cb-def* **by** *fastforce*

have M: $Ca * R\,\widehat{}\,(DIM('a)) \leq 1 + Cb * (lambda * R + C + norm(f\ 0))\,\widehat{}\,(DIM('b))$ **if** $R \geq 0$ **for** $R{::}real$
proof −
obtain $F{::}'a\ set$ **where** F: $F \subseteq cball\ 0\ R\ \forall x{\in}F.\ \forall y{\in}F.\ x = y \vee dist\ x\ y \geq D$
$finite\ F\ card\ F \geq Ca * R\,\widehat{}\,(DIM('a))$
using A[*OF* ‹$R \geq 0$›] **by** *auto*
define G **where** $G = f\,`F$

have ∗: *dist* (*f x*) (*f y*) ≥ *1* **if** *x* ≠ *y x* ∈ *F y* ∈ *F* **for** *x y*
proof −
have *dist x y* ≥ *D* **using** *that F*(*2*) **by** *auto*
have *1* = (*1*/*lambda*) ∗ *D* − *C* **using** ‹*lambda* ≥ *1*› **unfolding** *D-def* **by** *auto*
also have ... ≤ (*1*/*lambda*) ∗ *dist x y* − *C*
using ‹*dist x y* ≥ *D*› ‹*lambda* ≥ *1*› **by** (*auto simp add*: *divide-simps*)
also have ... ≤ *dist* (*f x*) (*f y*)
using *quasi-isometry-onD*[*OF assms*] **by** *auto*
finally show *?thesis* **by** *simp*
qed
then have *inj-on f F* **unfolding** *inj-on-def* **by** *force*
then have *card G* = *card F* **unfolding** *G-def* **by** (*simp add*: *card-image*)
then have *card G* ≥ *Ca* ∗ *R*⁀(*DIM*(′*a*)) **using** *F* **by** *auto*

moreover have *finite G* ∧ *card G* ≤ *1* + *Cb* ∗ (*lambda* ∗ *R* + *C* + *norm*(*f 0*))⁀(*DIM*(′*b*))
proof (*rule B*)
show *0* ≤ *lambda* ∗ *R* + *C* + *norm* (*f 0*) **using** ‹*R* ≥ *0*› ‹*C* ≥ *0*› ‹*lambda* ≥ *1*› **by** *auto*
show ∀*x*∈*G*. ∀*y*∈*G*. *x* = *y* ∨ *1* ≤ *dist x y* **using** ∗ **unfolding** *G-def* **by** (*auto*, *metis*)
show *G* ⊆ *cball 0* (*lambda* ∗ *R* + *C* + *norm* (*f 0*))
unfolding *G-def* **proof** (*auto*)
fix *x* **assume** *x* ∈ *F*
have *norm* (*f x*) ≤ *norm* (*f 0*) + *dist* (*f x*) (*f 0*)
by (*metis dist-0-norm dist-triangle2*)
also have ... ≤ *norm* (*f 0*) + (*lambda* ∗ *dist x 0* + *C*)
by (*intro mono-intros quasi-isometry-onD*(*1*)[*OF assms*]) *auto*
also have ... ≤ *norm* (*f 0*) + *lambda* ∗ *R* + *C*
using ‹*x* ∈ *F*› ‹*F* ⊆ *cball 0 R*› ‹*lambda* ≥ *1*› **by** *auto*
finally show *norm* (*f x*) ≤ *lambda* ∗ *R* + *C* + *norm* (*f 0*) **by** *auto*
qed
qed
ultimately show *Ca* ∗ *R*⁀(*DIM*(′*a*)) ≤ *1* + *Cb* ∗ (*lambda* ∗ *R* + *C* + *norm*(*f 0*))⁀(*DIM*(′*b*))
by *auto*
qed
define *CB* **where** *CB* = *max Cb 0*
have *CB* ≥ *0 CB* ≥ *Cb* **unfolding** *CB-def* **by** *auto*
define *D*::*real* **where** *D* = (*1* + *CB* ∗ (*lambda* + *C* + *norm*(*f 0*))⁀(*DIM*(′*b*)))/*Ca*
have *Rineq*: *R*⁀(*DIM*(′*a*)) ≤ *D* ∗ *R*⁀(*DIM*(′*b*)) **if** *R* ≥ *1* **for** *R*::*real*
proof −
have *Ca* ∗ *R*⁀(*DIM*(′*a*)) ≤ *1* + *Cb* ∗ (*lambda* ∗ *R* + *C* + *norm*(*f 0*))⁀(*DIM*(′*b*))
using *M* ‹*R* ≥ *1*› **by** *auto*
also have ... ≤ *1* + *CB* ∗ (*lambda* ∗ *R* + *C* + *norm*(*f 0*))⁀(*DIM*(′*b*))
using ‹*CB* ≥ *Cb*› ‹*lambda* ≥ *1*› ‹*R* ≥ *1*› ‹*C* ≥ *0*› **by** (*auto intro*!: *mult-right-mono*)
also have ... ≤ *R*⁀(*DIM*(′*b*)) + *CB* ∗ (*lambda* ∗ *R* + *C* ∗ *R* + *norm*(*f 0*) ∗ *R*)⁀(*DIM*(′*b*))

using ‹$lambda \geq 1$› ‹$R \geq 1$› ‹$C \geq 0$› ‹$CB \geq 0$› **by** (*auto intro*!: *mono-intros*)
also have ... $= (1 + CB * (lambda + C + norm(f\ 0))) \widehat{\ }(DIM('b))) * R \widehat{\ }(DIM('b))$
by (*auto simp add*: *algebra-simps power-mult-distrib*[*symmetric*])
finally show *?thesis*
using ‹$Ca > 0$› **unfolding** *D-def* **by** (*auto simp add*: *divide-simps algebra-simps*)
qed
show $DIM('a) \leq DIM('b)$
proof (*rule ccontr*)
assume $\neg(DIM('a) \leq DIM('b))$
then obtain n **where** $DIM('a) = DIM('b) + n$ $n > 0$
by (*metis less-imp-add-positive not-le*)
have $D \geq 1$ **using** *Rineq*[*of 1*] **by** *auto*
define R **where** $R = 2 * D$
then have $R \geq 1$ **using** ‹$D \geq 1$› **by** *auto*
have $R \widehat{\ } n * R \widehat{\ }(DIM('b)) = R \widehat{\ }(DIM('a))$
unfolding ‹$DIM('a) = DIM('b) + n$› **by** (*auto simp add*: *power-add*)
also have ... $\leq D * R \widehat{\ }(DIM('b))$ **using** *Rineq*[*OF* ‹$R \geq 1$›] **by** *auto*
finally have $R \widehat{\ } n \leq D$ **using** ‹$R \geq 1$› **by** *auto*
moreover have $2 * D \leq R \widehat{\ } n$ **unfolding** *R-def* **using** ‹$D \geq 1$› ‹$n > 0$›
by (*metis One-nat-def Suc-leI* ‹$1 \leq R$› ‹$R \equiv 2 * D$› *less-eq-real-def power-increasing-iff power-one power-one-right*)
ultimately show *False* **using** ‹$D \geq 1$› **by** *auto*
qed
qed

As a particular case, we deduce that two quasi-isometric Euclidean spaces have the same dimension.

theorem *quasi-isometric-euclidean*:
assumes *quasi-isometric* (*UNIV*::'*a*::*euclidean-space set*) (*UNIV*::'*b*::*euclidean-space set*)
shows $DIM('a) = DIM('b)$
proof −
obtain *lambda C* **and** $f::'a \Rightarrow 'b$ **where** *lambda C*−*quasi-isometry-on UNIV f*
using *assms* **unfolding** *quasi-isometric-def quasi-isometry-between-def* **by** *auto*
then have $*$: $DIM('a) \leq DIM('b)$ **using** *quasi-isometry-on-euclidean* **by** *auto*

have *quasi-isometric* (*UNIV*::'*b*::*euclidean-space set*) (*UNIV*::'*a*::*euclidean-space set*)
using *quasi-isometric-equiv-rel*(*3*)[*OF assms*] **by** *auto*
then obtain *lambda C* **and** $f::'b \Rightarrow 'a$ **where** *lambda C*−*quasi-isometry-on UNIV f*
unfolding *quasi-isometric-def quasi-isometry-between-def* **by** *auto*
then have $DIM('b) \leq DIM('a)$ **using** *quasi-isometry-on-euclidean* **by** *auto*
then show *?thesis* **using** $*$ **by** *auto*
qed

A different (and important) way to prove the above statement would be to use asymptotic cones. Here, it can be done in an elementary way: start with a quasi-isometric map f, and consider a limit (defined with a ultrafilter) of $x \mapsto f(nx)/n$. This is a map which contracts and expands the distances by at most λ. In particular, it is a homeomorphism on its image. No such map exists if the dimension of the target is smaller than the dimension of the source (invariance of domain theorem, already available in the library). The above argument using growth is more elementary to write, though.

6.4 Quasi-geodesics

A quasi-geodesic is a quasi-isometric embedding of a real segment into a metric space. As the embedding need not be continuous, a quasi-geodesic does not have to be compact, nor connected, which can be a problem. However, in a geodesic space, it is always possible to deform a quasi-geodesic into a continuous one (at the price of worsening the quasi-isometry constants). This is the content of the proposition `quasi_geodesic_made_lipschitz` below, which is a variation around Lemma III.H.1.11 in [BH99]. The strategy of the proof is simple: assume that the quasi-geodesic c is defined on $[a, b]$. Then, on the points a, $a + C/\lambda$, $\cdots$, $a + N \cdot C/\lambda$, b, take d equal to c, where N is chosen so that the distance between the last point and b is in $[C/\lambda, 2C/\lambda)$. In the intervals, take d to be geodesic.

proposition (**in** *geodesic-space*) *quasi-geodesic-made-lipschitz*:
  fixes *c*::*real* ⇒ *'a*
  assumes *lambda C−quasi-isometry-on* {*a*..*b*} *c dist* (*c a*) (*c b*) ≥ *2* ∗ *C*
  shows ∃ *d*. *continuous-on* {*a*..*b*} *d* ∧ *d a* = *c a* ∧ *d b* = *c b*
    ∧ (∀ *x*∈{*a*..*b*}. *dist* (*c x*) (*d x*) ≤ *4* ∗ *C*)
    ∧ *lambda* (*4* ∗ *C*)−*quasi-isometry-on* {*a*..*b*} *d*
    ∧ (*2* ∗ *lambda*)−*lipschitz-on* {*a*..*b*} *d*
    ∧ *hausdorff-distance* (*c*'{*a*..*b*}) (*d*'{*a*..*b*}) ≤ *2* ∗ *C*
proof −
  consider *C* = *0* | *C* > *0* ∧ *b* ≤ *a* | *C* > *0* ∧ *a* < *b* ∧ *b* ≤ *a* + *2* ∗ *C*/*lambda* | *C* > *0* ∧ *a* +*2* ∗ *C*/*lambda* < *b*
    using *quasi-isometry-onD*(*4*)[*OF assms*(*1*)] **by** *fastforce*
  then show *?thesis*
  proof (*cases*)

If the original function is Lipschitz, we can use it directly.

    case *1*
    have *lambda−lipschitz-on* {*a*..*b*} *c*
      apply (*rule lipschitz-onI*) **using** *1 quasi-isometry-onD*[*OF assms*(*1*)] **by** *auto*
    then have *a*: (*2* ∗ *lambda*)−*lipschitz-on* {*a*..*b*} *c*
      apply (*rule lipschitz-on-mono*) **using** *quasi-isometry-onD*[*OF assms*(*1*)] *assms* **by** (*auto simp add*: *divide-simps*)
    then have *b*: *continuous-on* {*a*..*b*} *c*
      using *lipschitz-on-continuous-on* **by** *blast*
    have *continuous-on* {*a*..*b*} *c* ∧ *c a* = *c a* ∧ *c b* = *c b*
      ∧ (∀ *x*∈{*a*..*b*}. *dist* (*c x*) (*c x*) ≤ *4* ∗ *C*)

∧ *lambda* (*4* ∗ *C*)−*quasi-isometry-on* {*a*..*b*} *c*
∧ (*2* ∗ *lambda*)−*lipschitz-on* {*a*..*b*} *c*
∧ *hausdorff-distance* (*c*‘{*a*..*b*}) (*c*‘{*a*..*b*}) ≤ *2* ∗ *C*
using *1 a b assms*(*1*) **by** *auto*
then show *?thesis* **by** *blast*
next

If the original interval is empty, anything will do.

case *2*
then have *b* < *a* **using** *assms*(*2*) *less-eq-real-def* **by** *auto*
then have ∗: {*a*..*b*} = {} **by** *auto*
have *a*: (*2* ∗ *lambda*)−*lipschitz-on* {*a*..*b*} *c*
unfolding ∗ **apply** (*rule lipschitz-intros*) **using** *quasi-isometry-onD*[*OF assms*(*1*)] *assms* **by** (*auto simp add*: *divide-simps*)
then have *b*: *continuous-on* {*a*..*b*} *c*
using *lipschitz-on-continuous-on* **by** *blast*
have *continuous-on* {*a*..*b*} *c* ∧ *c a* = *c a* ∧ *c b* = *c b*
∧ (∀ *x*∈{*a*..*b*}. *dist* (*c x*) (*c x*) ≤ *4* ∗ *C*)
∧ *lambda* (*4* ∗ *C*)−*quasi-isometry-on* {*a*..*b*} *c*
∧ (*2* ∗ *lambda*)−*lipschitz-on* {*a*..*b*} *c*
∧ *hausdorff-distance* (*c*‘{*a*..*b*}) (*c*‘{*a*..*b*}) ≤ *2* ∗ *C*
using *a b quasi-isometry-on-empty assms*(*1*) *quasi-isometry-onD*[*OF assms*(*1*)] ∗ *assms* **by** *auto*
then show *?thesis* **by** *blast*
next

If the original interval is short, we can use a direct geodesic interpolation between its endpoints

case *3*
then have *C*: *C* > *0 lambda* ≥ *1* **using** *quasi-isometry-onD*[*OF assms*(*1*)] **by** *auto*
have [*mono-intros*]: *1*/*lambda* ≤ *lambda* **using** *C* **by** (*simp add*: *divide-simps mult-ge1-powers*(*1*))
have *a* < *b* **using** *3* **by** *simp*
have *2* ∗ *C* ≤ *dist* (*c a*) (*c b*) **using** *assms* **by** *auto*
also have ... ≤ *lambda* ∗ *dist a b* + *C*
using *quasi-isometry-onD*[*OF assms*(*1*)] ‹*a* < *b*› **by** *auto*
also have ... = *lambda* ∗ (*b*−*a*) + *C*
using ‹*a* < *b*› *dist-real-def* **by** *auto*
finally have ∗: *C* ≤ (*b*−*a*) ∗ *lambda* **by** (*auto simp add*: *algebra-simps*)
define *d* **where** *d* = (λ*x*. *geodesic-segment-param* {(*c a*)−−(*c b*)} (*c a*) ((*dist* (*c a*) (*c b*) /(*b*−*a*)) ∗ (*x*−*a*)))
have *dend*: *d a* = *c a d b* = *c b* **unfolding** *d-def* **using** ‹*a* < *b*› **by** *auto*

have *Lip*: (*2* ∗ *lambda*)−*lipschitz-on* {*a*..*b*} *d*
proof −
have (*1* ∗ (((*2* ∗ *lambda*)) ∗ (*1*+*0*)))−*lipschitz-on* {*a*..*b*} (λ*x*. *geodesic-segment-param* {(*c a*)−−(*c b*)} (*c a*) ((*dist* (*c a*) (*c b*) /(*b*−*a*)) ∗ (*x*−*a*)))

```
      proof (rule lipschitz-on-compose2[of - - λx. ((dist (c a) (c b) /(b−a)) ∗
(x−a))], intro lipschitz-intros)
       have (λx. dist (c a) (c b) / (b−a) ∗ (x − a)) ‘ {a..b} ⊆ {0..dist (c a) (c b)}
         apply auto using ‹a < b› by (auto simp add: algebra-simps divide-simps
intro: mult-right-mono)
       moreover have 1−lipschitz-on {0..dist (c a) (c b)} (geodesic-segment-param
{c a−−c b} (c a))
         by (rule isometry-on-lipschitz, simp)
       ultimately show 1−lipschitz-on ((λx. dist (c a) (c b) / (b−a) ∗ (x − a))
‘ {a..b}) (geodesic-segment-param {c a−−c b} (c a))
         using lipschitz-on-subset by auto

      have dist (c a) (c b) ≤ lambda ∗ dist a b + C
        apply (rule quasi-isometry-onD(1)[OF assms(1)])
        using ‹a < b› by auto
      also have ... = lambda ∗ (b − a) + C
        unfolding dist-real-def using ‹a < b› by auto
      also have ... ≤ 2 ∗ lambda ∗ (b−a)
        using ∗ by (auto simp add: algebra-simps)
      finally show |dist (c a) (c b) / (b − a)| ≤ 2 ∗ lambda
        using ‹a < b› by (auto simp add: divide-simps)
    qed
    then show ?thesis unfolding d-def by auto
  qed
  have dist-c-d: dist (c x) (d x) ≤ 4 ∗ C if H: x ∈ {a..b} for x
  proof −
    have (x−a) + (b − x) ≤ 2 ∗ C/lambda
      using that 3 by auto
    then consider x−a ≤ C/lambda | b − x ≤ C/lambda by linarith
    then have ∃ v∈{a,b}. dist x v ≤ C/lambda
    proof (cases)
      case 1
      show ?thesis
        apply (rule bexI[of - a]) using 1 H by (auto simp add: dist-real-def)
    next
      case 2
      show ?thesis
        apply (rule bexI[of - b]) using 2 H by (auto simp add: dist-real-def)
    qed
    then obtain v where v: v ∈ {a,b} dist x v ≤ C/lambda by auto
    have dist (c x) (d x) ≤ dist (c x) (c v) + dist (c v) (d v) + dist (d v) (d x)
      by (intro mono-intros)
    also have ... ≤ (lambda ∗ dist x v + C) + 0 + ((2 ∗ lambda) ∗ dist v x)
       apply (intro mono-intros quasi-isometry-onD(1)[OF assms(1)] that lips-
chitz-onD[OF Lip])
      using v ‹a < b› dend by auto
       also have ... ≤ (lambda ∗ (C/lambda) + C) + 0 + ((2 ∗ lambda) ∗
(C/lambda))
    apply (intro mono-intros) using C v by (auto simp add: metric-space-class.dist-commute)
```

```
  finally show ?thesis
    using C by (auto simp add: algebra-simps divide-simps)
qed
```

A similar argument shows that the Hausdorff distance between the images is bounded by $2C$.

```
have hausdorff-distance (c‘{a..b}) (d‘{a..b}) ≤ 2 ∗ C
proof (rule hausdorff-distanceI2)
  show 0 ≤ 2 ∗ C using C by auto
  fix z assume z ∈ c‘{a..b}
  then obtain x where x: x ∈ {a..b} z = c x by auto
  have (x−a) + (b − x) ≤ 2 ∗ C/lambda
    using x 3 by auto
  then consider x−a ≤ C/lambda | b − x ≤ C/lambda by linarith
  then have ∃ v∈{a,b}. dist x v ≤ C/lambda
  proof (cases)
    case 1
    show ?thesis
      apply (rule bexI[of - a]) using 1 x by (auto simp add: dist-real-def)
  next
    case 2
    show ?thesis
      apply (rule bexI[of - b]) using 2 x by (auto simp add: dist-real-def)
  qed
  then obtain v where v: v ∈ {a,b} dist x v ≤ C/lambda by auto
  have dist z (d v) = dist (c x) (c v) unfolding x(2) using v dend by auto
  also have ... ≤ lambda ∗ dist x v + C
   apply (rule quasi-isometry-onD(1)[OF assms(1)]) using v(1) x(1) by auto
  also have ... ≤ lambda ∗ (C/lambda) + C
    apply (intro mono-intros) using C v(2) by auto
  also have ... = 2 ∗ C
    using C by (simp add: divide-simps)
  finally have ∗: dist z (d v) ≤ 2 ∗ C by simp
  show ∃ y∈d ‘ {a..b}. dist z y ≤ 2 ∗ C
    apply (rule bexI[of - d v]) using ∗ v(1) ‹a < b› by auto
next
  fix z assume z ∈ d‘{a..b}
  then obtain x where x: x ∈ {a..b} z = d x by auto
  have (x−a) + (b − x) ≤ 2 ∗ C/lambda
    using x 3 by auto
  then consider x−a ≤ C/lambda | b − x ≤ C/lambda by linarith
  then have ∃ v∈{a,b}. dist x v ≤ C/lambda
  proof (cases)
    case 1
    show ?thesis
      apply (rule bexI[of - a]) using 1 x by (auto simp add: dist-real-def)
  next
    case 2
    show ?thesis
```

```
      apply (rule bexI[of - b]) using 2 x by (auto simp add: dist-real-def)
    qed
    then obtain v where v: v ∈ {a,b} dist x v ≤ C/lambda by auto
    have dist z (c v) = dist (d x) (d v) unfolding x(2) using v dend by auto
    also have ... ≤ 2 ∗ lambda ∗ dist x v
      apply (rule lipschitz-onD(1)[OF Lip]) using v(1) x(1) by auto
    also have ... ≤ 2 ∗ lambda ∗ (C/lambda)
      apply (intro mono-intros) using C v(2) by auto
    also have ... = 2 ∗ C
      using C by (simp add: divide-simps)
    finally have ∗: dist z (c v) ≤ 2 ∗ C by simp
    show ∃y∈c‘{a..b}. dist z y ≤ 2 ∗ C
      apply (rule bexI[of - c v]) using ∗ v(1) ‹a < b› by auto
  qed
  have lambda (4 ∗ C)−quasi-isometry-on {a..b} d
  proof
    show 1 ≤ lambda using C by auto
    show 0 ≤ 4 ∗ C using C by auto
    show dist (d x) (d y) ≤ lambda ∗ dist x y + 4 ∗ C if x ∈ {a..b} y ∈ {a..b}
for x y
    proof −
      have dist (d x) (d y) ≤ 2 ∗ lambda ∗ dist x y
        apply (rule lipschitz-onD[OF Lip]) using that by auto
      also have ... = lambda ∗ dist x y + lambda ∗ dist x y
        by auto
      also have ... ≤ lambda ∗ dist x y + lambda ∗ (2 ∗ C/lambda)
       apply (intro mono-intros) using 3 that C unfolding dist-real-def by auto
      also have ... = lambda ∗ dist x y + 2 ∗ C
        using C by (simp add: algebra-simps divide-simps)
      finally show ?thesis using C by auto
    qed
     show 1 / lambda ∗ dist x y − 4 ∗ C ≤ dist (d x) (d y) if x ∈ {a..b} y ∈
{a..b} for x y
    proof −
      have 1/lambda ∗ dist x y − 4 ∗ C ≤ lambda ∗ dist x y − 2 ∗ C
        apply (intro mono-intros) using C by auto
      also have ... ≤ lambda ∗ (2 ∗ C/lambda) − 2 ∗ C
       apply (intro mono-intros) using that 3 C unfolding dist-real-def by auto
      also have ... = 0
        using C by (auto simp add: algebra-simps divide-simps)
      also have ... ≤ dist (d x) (d y) by auto
      finally show ?thesis by simp
    qed
  qed

  then have continuous-on {a..b} d ∧ d a = c a ∧ d b = c b
        ∧ lambda (4 ∗ C)−quasi-isometry-on {a..b} d
        ∧ (∀x∈{a..b}. dist (c x) (d x) ≤ 4 ∗C)
        ∧ (2∗lambda)−lipschitz-on {a..b} d
```

$\wedge$ *hausdorff-distance* (*c*‘{*a..b*}) (*d*‘{*a..b*}) ≤ *2* ∗ *C*
using *dist-c-d* ‹*d a* = *c a*› ‹*d b* = *c b*› ‹(*2*∗*lambda*)−*lipschitz-on* {*a..b*} *d*›
‹*hausdorff-distance* (*c*‘{*a..b*}) (*d*‘{*a..b*}) ≤ *2* ∗ *C*› *lipschitz-on-continuous-on*
by *auto*
then show *?thesis* **by** *auto*
next

Now, for the only nontrivial case, we use geodesic interpolation between the points a, $a + C/\lambda$, $\cdots$, $a + N \cdot C/\lambda$, b', b where N is chosen so that the distance between $a + NC/\lambda$ and b belongs to $[2C/\lambda, 3C/\lambda)$, and b' is the middle of this interval. This gives a decomposition into intervals of length at most $3/2 \cdot C/\lambda$.

case *4*
then have *C*: *C* > *0* *lambda* ≥ *1* **using** *quasi-isometry-onD*[*OF assms*(*1*)] **by** *auto*
have *a* < *b* **using** *4* *C* **by** (*smt divide-pos-pos*)

have [*mono-intros*]: *1*/*lambda* ≤ *lambda* **using** *C* **by** (*simp add*: *divide-simps* *mult-ge1-powers*(*1*))
define *N* **where** *N* = *floor*((*b*−*a*)/(*C*/*lambda*)) − *2*
have *N*: *N* ≤ (*b*−*a*)/(*C*/*lambda*)−*2* (*b*−*a*)/(*C*/*lambda*) ≤ *N* + (*3*::*real*)
unfolding *N-def* **by** *linarith*+

have *2* < (*b*−*a*)/(*C*/*lambda*)
using *C* *4* **by** (*auto simp add*: *divide-simps algebra-simps*)
then have *N0* : *0* ≤ *N* **unfolding** *N-def* **by** *auto*
define *p* **where** *p* = (λ*t*::*int*. *a* + (*C*/*lambda*) ∗ *t*)
have *pmono*: *p i* ≤ *p j* **if** *i* ≤ *j* **for** *i j*
unfolding *p-def* **using** *that C* **by** (*auto simp add*: *algebra-simps divide-simps*)
have *pmono*′: *p i* < *p j* **if** *i* < *j* **for** *i j*
unfolding *p-def* **using** *that C* **by** (*auto simp add*: *algebra-simps divide-simps*)
have *p* (*N*+*1*) ≤ *b*
unfolding *p-def* **using** *C N* **by** (*auto simp add*: *algebra-simps divide-simps*)
then have *pb*: *p i* ≤ *b* **if** *i* ∈ {*0..N*} **for** *i*
using *that pmono* **by** (*meson atLeastAtMost-iff linear not-le order-trans* *zle-add1-eq-le*)
have *bpN*: *b* − *p N* ∈ {*2* ∗ *C*/*lambda* .. *3* ∗ *C*/*lambda*}
unfolding *p-def* **using** *C N* **apply** (*auto simp add*: *divide-simps*)
by (*auto simp add*: *algebra-simps*)
have *p N* < *b* **using** *pmono*′[*of N N*+*1*] ‹*p* (*N*+*1*) ≤ *b*› **by** *auto*
define *b*′ **where** *b*′ = (*b* + *p N*)/*2*
have *b*′: *p N* < *b*′ *b*′ < *b* **using** ‹*p N* < *b*› **unfolding** *b*′*-def* **by** *auto*
have *pb*′: *p i* ≤ *b*′ **if** *i* ∈ {*0..N*} **for** *i*
using *pmono*[*of i N*] *b*′ *that* **by** *auto*

Introduce the set A along which one will discretize.

define *A* **where** *A* = *p*‘{*0..N*} ∪ {*b*′, *b*}
have *finite A* **unfolding** *A-def* **by** *auto*
have *b* ∈ *A* **unfolding** *A-def* **by** *auto*

have $p\ 0 \in A$ **unfolding** *A-def* **using** ‹$0 \le N$› **by** *auto*
moreover have *pa*: $p\ 0 = a$ **unfolding** *p-def* **by** *auto*
ultimately have $a \in A$ **by** *auto*
have $A \subseteq \{a..b\}$
unfolding *A-def* **using** ‹$a < b$› b' *pa pb pmono N0* **by** *fastforce*
then have $b' \in \{a..{<}b\}$ **unfolding** *A-def* **using** ‹$b' < b$› **by** *auto*

have A : *finite* A $A \subseteq \{a..b\}$ $a \in A$ $b \in A$ $a < b$ **by** *fact+*

have *nx*: *next-in* $A\ x = x + C/lambda$ **if** $x \in A$ $x \neq b$ $x \neq b'$ $x \neq p\ N$ **for** x
proof (*rule next-inI*[*OF A*])
show $x \in \{a..{<}b\}$ **using** ‹$x \in A$› ‹$A \subseteq \{a..b\}$› ‹$x \neq b$› **by** *auto*
obtain i **where** i: $x = p\ i$ $i \in \{0..N\}$
using ‹$x \in A$› ‹$x \neq b$› ‹$x \neq b'$› **unfolding** *A-def* **by** *auto*
have $*$: $p\ (i{+}1) = x + C/lambda$ **unfolding** $i(1)$ *p-def* **by** (*auto simp add*: *algebra-simps*)
have $i \neq N$ **using** *that* i **by** *auto*
then have $i + 1 \in \{0..N\}$ **using** ‹$i \in \{0..N\}$› **by** *auto*
then have $p\ (i{+}1) \in A$ **unfolding** *A-def* **by** *fastforce*
then show $x + C/lambda \in A$ **unfolding** $*$ **by** *auto*
show $x < x + C\ /\ lambda$ **using** C **by** *auto*
show $\{x{<}..{<}x + C\ /\ lambda\} \cap A = \{\}$
proof (*auto*)
fix y **assume** y: $y \in A$ $x < y$ $y < x + C/lambda$
consider $y = b$ | $y = b'$ | $\exists j{\le}i.\ y = p\ j$ | $\exists j{>}i.\ y = p\ j$
using ‹$y \in A$› *not-less* **unfolding** *A-def* **by** *auto*
then show *False*
proof (*cases*)
case *1*
have $x + C/lambda \le b$ **unfolding** $*$[*symmetric*] **using** ‹$i + 1 \in \{0..N\}$› *pb* **by** *auto*
then show *False* **using** $y(3)$ **unfolding** *1* $i(1)$ **by** *auto*
next
case *2*
have $x + C/lambda \le b'$ **unfolding** $*$[*symmetric*] **using** ‹$i + 1 \in \{0..N\}$› *pb'* **by** *auto*
then show *False* **using** $y(3)$ **unfolding** *2* $i(1)$ **by** *auto*
next
case *3*
then obtain j **where** j: $j \le i$ $y = p\ j$ **by** *auto*
have $y \le x$ **unfolding** $j(2)$ $i(1)$ **using** *pmono*[*OF* ‹$j \le i$›] **by** *simp*
then show *False* **using** ‹$x < y$› **by** *auto*
next
case *4*
then obtain j **where** j: $j > i$ $y = p\ j$ **by** *auto*
then have $i{+}1 \le j$ **by** *auto*
have $x + C/lambda \le y$ **unfolding** $j(2)$ $*$[*symmetric*] **using** *pmono*[*OF* ‹$i{+}1 \le j$›] **by** *auto*
then show *False* **using** ‹$y < x + C/lambda$› **by** *auto*

qed
qed
qed
have *npN*: *next-in A* $(p\ N) = b'$
proof (*rule next-inI*[*OF A*])
show $p\ N \in \{a..<b\}$ **using** *pa pmono* ‹$0 \le N$› ‹$p\ N < b$› **by** *auto*
show $p\ N < b'$ **by** *fact*
show $b' \in A$ **unfolding** *A-def* **by** *auto*
show $\{p\ N<..<b'\} \cap A = \{\}$
unfolding *A-def* **using** *pmono* b' **by** *force*
qed
have nb': *next-in A* $(b') = b$
proof (*rule next-inI*[*OF A*])
show $b' \in \{a..<b\}$ **using** *A-def A* ‹$b' < b$› **by** *auto*
show $b' < b$ **by** *fact*
show $b \in A$ **by** *fact*
show $\{b'<..<b\} \cap A = \{\}$
unfolding *A-def* **using** *pmono* b' **by** *force*
qed
have *gap*: *next-in A* $x - x \in \{C/lambda..\ 3/2 * C/lambda\}$ **if** $x \in A - \{b\}$ **for** x
proof (*cases* $x = p\ N \vee x = b'$)
case *True*
then show *?thesis* **using** *npN* nb' *bpN* b'*-def* **by** *force*
next
case *False*
have $*$: *next-in A* $x = x + C/lambda$
apply (*rule nx*) **using** *that False* **by** *auto*
show *?thesis* **unfolding** $*$ **using** C **by** (*auto simp add*: *algebra-simps divide-simps*)
qed

We can now define the function d, by geodesic interpolation between points in A.

define d **where** $d\ x = ($*if* $x \in A$ *then* $c\ x$
else geodesic-segment-param $\{c$ (*prev-in A x*) $--$ c (*next-in A x*)$\}$ $(c$ (*prev-in A x*)$)$
$((x -$ *prev-in A x*$)/($*next-in A x* $-$ *prev-in A x*$) *$ *dist* $(c($*prev-in A x*$))$ $(c($*next-in A x*$))))$ **for** x
have $d\ a = c\ a$ $d\ b = c\ b$ **unfolding** *d-def* **using** ‹$a \in A$› ‹$b \in A$› **by** *auto*

To prove the Lipschitz continuity, we argue that d is Lipschitz on finitely many intervals, that cover the interval $[a, b]$, the intervals between points in A. There is a formula for d on them (the nontrivial point is that the above formulas for d match at the boundaries).

have $*$: $d\ x =$ *geodesic-segment-param* $\{(c\ u)--(c\ v)\}$ $(c\ u)$ $(($*dist* $(c\ u)$ $(c\ v)$ $/(v-u)) * (x-u))$
if $u \in A - \{b\}$ $v =$ *next-in A u* $x \in \{u..v\}$ **for** $x\ u\ v$
proof −

have $u \in \{a..<b\}$ **using** *that* ‹$A \subseteq \{a..b\}$› **by** *fastforce*
have H: $u \in A$ $v \in A$ $u < v$ $A \cap \{u<..<v\} = \{\}$ **using** *that next-in-basics*[*OF* A ‹$u \in \{a..<b\}$›] **by** *auto*
consider $x = u \mid x = v \mid x \in \{u<..<v\}$ **using** ‹$x \in \{u..v\}$› **by** *fastforce*
then show *?thesis*
proof (*cases*)
case *1*
then have $d\ x = c\ u$ **unfolding** *d-def* **using** ‹$u \in A-\ \{b\}$› ‹$A \subseteq \{a..b\}$› **by** *auto*
then show *?thesis* **unfolding** *1* **by** *auto*
next
case *2*
then have $d\ x = c\ v$ **unfolding** *d-def* **using** ‹$v \in A$› ‹$A \subseteq \{a..b\}$› **by** *auto*
then show *?thesis* **unfolding** *2* **using** ‹$u < v$› **by** *auto*
next
case *3*
have $*$: *prev-in* $A\ x = u$
apply (*rule prev-inI*[*OF A*]) **using** *3 H* ‹$A \subseteq \{a..b\}$› **by** *auto*
have $**$: *next-in* $A\ x = v$
apply (*rule next-inI*[*OF A*]) **using** *3 H* ‹$A \subseteq \{a..b\}$› **by** *auto*
show *?thesis* **unfolding** *d-def* $*$ $**$ **using** *3 H* ‹$A \cap \{u<..<v\} = \{\}$› ‹$A \subseteq \{a..b\}$›
by (*auto simp add*: *algebra-simps*)
qed
qed

From the above formula, we deduce that d is Lipschitz on those intervals.

have *lip0*: (*lambda* + C / (*next-in* $A\ u - u$))$-$*lipschitz-on* $\{u..$*next-in* $A\ u\}$ d **if** $u \in A - \{b\}$ **for** u
proof $-$
define v **where** $v =$ *next-in* $A\ u$
have $u \in \{a..<b\}$ **using** *that* ‹$A \subseteq \{a..b\}$› **by** *fastforce*
have $u \in A$ $v \in A$ $u < v$ $A \cap \{u<..<v\} = \{\}$
unfolding *v-def* **using** *that next-in-basics*[*OF* A ‹$u \in \{a..<b\}$›] **by** *auto*

have (*1* $*$ (((*lambda* + C / (*next-in* $A\ u - u$))) $*$ (*1*+*0*)))$-$*lipschitz-on* $\{u..v\}$ ($\lambda x.$ *geodesic-segment-param* $\{(c\ u)--(c\ v)\}$ $(c\ u)$ ((*dist* $(c\ u)$ $(c\ v)$ $/(v-u)$) $*$ $(x-u)$))
proof (*rule lipschitz-on-compose2*[*of - -* $\lambda x.$ ((*dist* $(c\ u)$ $(c\ v)$ $/(v-u)$) $*$ $(x-u)$)], *intro lipschitz-intros*)
have ($\lambda x.$ *dist* $(c\ u)$ $(c\ v)$ / $(v - u) * (x - u)$) ‘ $\{u..v\} \subseteq \{$*0*..*dist* $(c\ u)$ $(c\ v)\}$
apply *auto* **using** ‹$u < v$› **by** (*auto simp add*: *algebra-simps divide-simps intro*: *mult-right-mono*)
moreover have *1*$-$*lipschitz-on* $\{$*0*..*dist* $(c\ u)$ $(c\ v)\}$ (*geodesic-segment-param* $\{c\ u--c\ v\}$ $(c\ u)$)
by (*rule isometry-on-lipschitz*, *simp*)
ultimately show *1*$-$*lipschitz-on* (($\lambda x.$ *dist* $(c\ u)$ $(c\ v)$ / $(v - u) * (x - u)$) ‘ $\{u..v\}$) (*geodesic-segment-param* $\{c\ u--c\ v\}$ $(c\ u)$)

```
      using lipschitz-on-subset by auto

    have dist (c u) (c v) ≤ lambda ∗ dist u v + C
      apply (rule quasi-isometry-onD(1)[OF assms(1)])
      using ‹u ∈ A› ‹v ∈ A› ‹A ⊆ {a..b}› by auto
    also have ... = lambda ∗ (v − u) + C
      unfolding dist-real-def using ‹u < v› by auto
    finally show |dist (c u) (c v) / (v − u)| ≤ lambda + C / (next-in A u − u)
      using ‹u < v› unfolding v-def by (auto simp add: divide-simps)
  qed
  then show ?thesis
    using ∗[OF ‹u ∈ A −{b}› ‹v = next-in A u›] unfolding v-def
    by (auto intro: lipschitz-on-transform)
qed
have lip: (2 ∗ lambda)−lipschitz-on {u..next-in A u} d if u ∈ A − {b} for u
proof (rule lipschitz-on-mono[OF lip0[OF that]], auto)
  define v where v = next-in A u
  have u ∈ {a..<b} using that ‹A ⊆ {a..b}› by fastforce
  have u ∈ A v ∈ A u < v A ∩ {u<..<v} = {}
    unfolding v-def using that next-in-basics[OF A ‹u ∈ {a..<b}›] by auto
  have Duv: v − u ∈ {C/lambda .. 2 ∗ C/lambda}
    unfolding v-def using gap[OF ‹u ∈ A − {b}›] by simp
  then show  C / (next-in A u − u) ≤ lambda
     using ‹u < v› C unfolding v-def by (auto simp add: algebra-simps
divide-simps)
qed
```

The Lipschitz continuity of d now follows from its Lipschitz continuity on each subinterval in I.

```
have Lip: (2 ∗ lambda)−lipschitz-on {a..b} d
  apply (rule lipschitz-on-closed-Union[of {{u..next-in A u} |u. u ∈ A − {b}}
- λx. x])
  using lip ‹finite A› C intervals-decomposition[OF A] using assms by auto
then have continuous-on {a..b} d
  using lipschitz-on-continuous-on by auto
```

d has good upper controls on each basic interval.

```
have QI0: dist (d x) (d y) ≤ lambda ∗ dist x y + C
  if H: u ∈ A − {b} x ∈ {u..next-in A u} y ∈ {u..next-in A u} for u x y
proof −
  have u < next-in A u using H(1) A next-in-basics(2)[OF A] by auto
  moreover have dist x y ≤ next-in A u − u unfolding dist-real-def using H
by auto
  ultimately have ∗: dist x y / (next-in A u − u) ≤ 1 by (simp add: di-
vide-simps)
  have dist (d x) (d y) ≤ (lambda + C / (next-in A u − u)) ∗ dist x y
    by (rule lipschitz-onD[OF lip0[OF H(1)] H(2) H(3)])
  also have ... = lambda ∗ dist x y + C ∗ (dist x y / (next-in A u − u))
    by (simp add: algebra-simps)
```

also have ... $\leq$ *lambda* $*$ *dist x y* + C $*$ *1*
  apply (*intro mono-intros*) **using** C $*$ **by** *auto*
 finally show *?thesis* **by** *simp*
qed

We can now show that c and d are pointwise close. This follows from the fact that they coincide on A and are well controlled in between (for c, this is a consequence of the choice of A. For d, it follows from the fact that it is geodesic in the intervals).

have *dist-c-d*: *dist* (c x) (d x) $\leq$ *4* $*$ C **if** $x \in \{a..b\}$ **for** x
proof −
 obtain u **where** u: $u \in A - \{b\}$ $x \in \{u..$*next-in* A $u\}$
  using ‹$x \in \{a..b\}$› *intervals-decomposition*[*OF A*] **by** *blast*
 have ($x-u$) + (*next-in* A u − x) $\leq$ *2* $*$ C/*lambda*
  using *gap*[*OF u*(*1*)] **by** *auto*
 then consider $x-u \leq C$/*lambda* | *next-in* A u − $x \leq C$/*lambda* **by** *linarith*
 then have $\exists\, v \in A$. *dist x v* $\leq$ C/*lambda*
 proof (*cases*)
  case *1*
  show *?thesis*
   apply (*rule bexI*[*of* - *u*]) **using** *1* u **by** (*auto simp add*: *dist-real-def*)
 next
  case *2*
  show *?thesis*
   apply (*rule bexI*[*of* - *next-in A u*]) **using** *2* u A(*2*)
   by (*auto simp add*: *dist-real-def intro*!:*next-in-basics*[*OF A*])
 qed
 then obtain v **where** v: $v \in A$ *dist x v* $\leq$ C/*lambda* **by** *auto*
 have *dist* (c x) (d x) $\leq$ *dist* (c x) (c v) + *dist* (c v) (d v) + *dist* (d v) (d x)
  by (*intro mono-intros*)
 also have ... $\leq$ (*lambda* $*$ *dist x v* + C) + *0* + ((*2* $*$ *lambda*) $*$ *dist v x*)
   apply (*intro mono-intros quasi-isometry-onD*(*1*)[*OF assms*(*1*)] *that lipschitz-onD*[*OF Lip*])
  using A(*2*) ‹$v \in A$› **apply** *blast*
  using ‹$v \in A$› *d-def* **apply** *auto*[*1*]
  using A(*2*) ‹$v \in A$› **by** *blast*
 also have ... $\leq$ (*lambda* $*$ (C/*lambda*) + C) + *0* + ((*2* $*$ *lambda*) $*$ (C/*lambda*))
  apply (*intro mono-intros*) **using** v(*2*) C **by** (*auto simp add*: *metric-space-class.dist-commute*)
 finally show *?thesis*
  using C **by** (*auto simp add*: *algebra-simps divide-simps*)
qed

A similar argument shows that the Hausdorff distance between the images is bounded by $2C$.

have *hausdorff-distance* (c‘$\{a..b\}$) (d‘$\{a..b\}$) $\leq$ *2* $*$ C
proof (*rule hausdorff-distanceI2*)
 show *0* $\leq$ *2* $*$ C **using** C **by** *auto*
 fix z **assume** $z \in c$‘$\{a..b\}$

```
    then obtain x where x: x ∈ {a..b} z = c x by auto
    then obtain u where u: u ∈ A − {b} x ∈ {u..next-in A u}
      using intervals-decomposition[OF A] by blast
    have (x−u) + (next-in A u − x) ≤ 2 ∗ C/lambda
      using gap[OF u(1)] by auto
    then consider x−u ≤ C/lambda | next-in A u − x ≤ C/lambda by linarith
    then have ∃ v∈A. dist x v ≤ C/lambda
    proof (cases)
      case 1
      show ?thesis
        apply (rule bexI[of - u]) using 1 u by (auto simp add: dist-real-def)
    next
      case 2
      show ?thesis
        apply (rule bexI[of - next-in A u]) using 2 u A(2)
        by (auto simp add: dist-real-def intro!:next-in-basics[OF A])
    qed
    then obtain v where v: v ∈ A dist x v ≤ C/lambda by auto
    have dist z (d v) = dist (c x) (c v) unfolding x(2) d-def using ‹v ∈ A› by
auto
    also have ... ≤ lambda ∗ dist x v + C
       apply (rule quasi-isometry-onD(1)[OF assms(1)]) using v(1) A(2) x(1)
by auto
    also have ... ≤ lambda ∗ (C/lambda) + C
      apply (intro mono-intros) using C v(2) by auto
    also have ... = 2 ∗ C
      using C by (simp add: divide-simps)
    finally have ∗: dist z (d v) ≤ 2 ∗ C by simp
    show ∃ y∈d ‘ {a..b}. dist z y ≤ 2 ∗ C
      apply (rule bexI[of - d v]) using ∗ v(1) A(2) by auto
  next
    fix z assume z ∈ d‘{a..b}
    then obtain x where x: x ∈ {a..b} z = d x by auto
    then obtain u where u: u ∈ A − {b} x ∈ {u..next-in A u}
      using intervals-decomposition[OF A] by blast
    have (x−u) + (next-in A u − x) ≤ 2 ∗ C/lambda
      using gap[OF u(1)] by auto
    then consider x−u ≤ C/lambda | next-in A u − x ≤ C/lambda by linarith
    then have ∃ v∈A. dist x v ≤ C/lambda
    proof (cases)
      case 1
      show ?thesis
        apply (rule bexI[of - u]) using 1 u by (auto simp add: dist-real-def)
    next
      case 2
      show ?thesis
        apply (rule bexI[of - next-in A u]) using 2 u A(2)
        by (auto simp add: dist-real-def intro!:next-in-basics[OF A])
    qed
```

```
    then obtain v where v: v ∈ A dist x v ≤ C/lambda by auto
    have dist z (c v) = dist (d x) (d v) unfolding x(2) d-def using ‹v ∈ A› by
auto
    also have ... ≤ 2 ∗ lambda ∗ dist x v
      apply (rule lipschitz-onD(1)[OF Lip]) using v(1) A(2) x(1) by auto
    also have ... ≤ 2 ∗ lambda ∗ (C/lambda)
      apply (intro mono-intros) using C v(2) by auto
    also have ... = 2 ∗ C
      using C by (simp add: divide-simps)
    finally have ∗: dist z (c v) ≤ 2 ∗ C by simp
    show ∃ y∈c‘{a..b}. dist z y ≤ 2 ∗ C
      apply (rule bexI[of - c v]) using ∗ v(1) A(2) by auto
  qed
```

From the above controls, we check that d is a quasi-isometry, with explicit constants.

```
  have lambda (4 ∗ C)−quasi-isometry-on {a..b} d
  proof
    show 1 ≤ lambda using C by auto
    show 0 ≤ 4 ∗ C using C by auto
    have I : dist (d x) (d y) ≤ lambda ∗ dist x y + 4 ∗ C if H: x ∈ {a..b} y ∈
{a..b} x < y for x y
    proof −
      obtain u where u: u ∈ A − {b} x ∈ {u..next-in A u}
        using intervals-decomposition[OF A] H(1) by force
      have u ∈ {a..<b} using u(1) A by auto
      have next-in A u ∈ A using next-in-basics(1)[OF A ‹u ∈ {a..<b}›] by auto
      obtain v where v: v ∈ A − {b} y ∈ {v..next-in A v}
        using intervals-decomposition[OF A] H(2) by force
      have v ∈ {a..<b} using v(1) A by auto
      have u < next-in A v using H(3) u(2) v(2) by auto
      then have u ≤ v
        using u(1) next-in-basics(3)[OF A, OF ‹v ∈ {a..<b}›] by auto
      show ?thesis
      proof (cases u = v)
        case True
        have dist (d x) (d y) ≤ lambda ∗ dist x y + C
          apply (rule QI0[OF u]) using v(2) True by auto
        also have ... ≤ lambda ∗ dist x y + 4 ∗ C
          using C by auto
        finally show ?thesis by simp
      next
        case False
        then have u < v using ‹u ≤ v› by auto
        then have next-in A u ≤ v using v(1) next-in-basics(3)[OF A, OF ‹u ∈
{a..<b}›] by auto
        have d1: d (next-in A u) = c (next-in A u)
          using ‹next-in A u ∈ A› unfolding d-def by auto
        have d2: d v = c v
```

```
          using v(1) unfolding d-def by auto
        have dist (d x) (d y) ≤ dist (d x) (d (next-in A u)) + dist (d (next-in A
u)) (d v) + dist (d v) (d y)
          by (intro mono-intros)
          also have ... ≤ (lambda ∗ dist x (next-in A u) + C) + (lambda ∗ dist
(next-in A u) v + C)
                        + (lambda ∗ dist v y + C)
          apply (intro mono-intros)
            apply (rule QI0[OF u]) using u(2) apply simp
              apply (simp add: d1 d2) apply (rule quasi-isometry-onD(1)[OF
assms(1)])
          using ‹next-in A u ∈ A› ‹A ⊆ {a..b}› apply auto[1]
          using ‹v ∈ A − {b}› ‹A ⊆ {a..b}› apply auto[1]
          apply (rule QI0[OF v(1)]) using v(2) by auto
        also have ... = lambda ∗ dist x y + 3 ∗ C
          unfolding dist-real-def
          using ‹x ∈ {u..next-in A u}› ‹y ∈ {v..next-in A v}› ‹x < y› ‹next-in A
u ≤ v›
          by (auto simp add: algebra-simps)
        finally show ?thesis using C by simp
      qed
    qed
    show dist (d x) (d y) ≤ lambda ∗ dist x y + 4 ∗ C if H: x ∈ {a..b} y ∈
{a..b} for x y
    proof −
      consider x < y | x = y | x > y by linarith
      then show ?thesis
      proof (cases)
        case 1
        then show ?thesis using I[OF H(1) H(2) 1] by simp
      next
        case 2
        show ?thesis unfolding 2 using C by auto
      next
        case 3
     show ?thesis using I [OF H(2) H(1) 3] by (simp add: metric-space-class.dist-commute)
      qed
    qed
```

The lower bound is more tricky. We separate the case where x and y are in the same interval, when they are in different nearby intervals, and when they are in different separated intervals. The latter case is more difficult. In this case, one of the intervals has length C/λ and the other one has length at most $3/2 \cdot C/\lambda$. There, we approximate $dist(dx)(dy)$ by $dist(du')(dv')$ where u' and v' are suitable endpoints of the intervals containing respectively x and y. We use the inner endpoint (between x and y) if the distance between x or y and this point is less than $2/5$ of the length of the interval, and the outer endpoint otherwise. The reason is that, with the outer endpoints, we get right away an upper bound for the distance between x and y, while this is not the case with the inner endpoints where there is an additional error. The equilibrium is reached at proportion 2/5.

have J : *dist* (*d x*) (*d y*) $\geq$ (*1/lambda*) $*$ *dist x y* $-$ *4* $*$ *C* **if** *H*: $x \in \{a..b\}$ $y \in \{a..b\}$ $x < y$ **for** *x y*
proof $-$
obtain *u* **where** *u*: $u \in A - \{b\}$ $x \in \{u..next\text{-}in\ A\ u\}$
using *intervals-decomposition*[*OF A*] *H*(*1*) **by** *force*
have $u \in \{a..<b\}$ **using** *u*(*1*) *A* **by** *auto*
have *next-in A u* $\in A$ **using** *next-in-basics*(*1*)[*OF A* ‹$u \in \{a..<b\}$›] **by** *auto*
obtain *v* **where** *v*: $v \in A - \{b\}$ $y \in \{v..next\text{-}in\ A\ v\}$
using *intervals-decomposition*[*OF A*] *H*(*2*) **by** *force*
have $v \in \{a..<b\}$ **using** *v*(*1*) *A* **by** *auto*
have *next-in A v* $\in A$ **using** *next-in-basics*(*1*)[*OF A* ‹$v \in \{a..<b\}$›] **by** *auto*
have $u <$ *next-in A v* **using** *H*(*3*) *u*(*2*) *v*(*2*) **by** *auto*
then have $u \leq v$
using *u*(*1*) *next-in-basics*(*3*)[*OF A*, *OF* ‹$v \in \{a..<b\}$›] **by** *auto*
consider $v = u$ | $v =$ *next-in A u* | $v \neq u \wedge v \neq$ *next-in A u* **by** *auto*
then show *?thesis*
proof (*cases*)
case *1*
have (*1/lambda*) $*$ *dist x y* $-$ *4* $*$ *C* $\leq$ *lambda* $*$ *dist x y* $-$ *4* $*$ *C*
apply (*intro mono-intros*) **by** *auto*
also have ... $\leq$ *lambda* $*$ (*3/2* $*$ *C/lambda*) $-$ *3/2* $*$ *C*
apply (*intro mono-intros*)
using *u*(*2*) *v*(*2*) **unfolding** *1* **using** *C gap*[*OF u*(*1*)] *dist-real-def* ‹$x < y$› **by** *auto*
also have ... $=$ *0*
using *C* **by** *auto*
also have ... $\leq$ *dist* (*d x*) (*d y*)
by *auto*
finally show *?thesis* **by** *simp*
next
case *2*
have *dist x y* $\leq$ *dist x* (*next-in A u*) $+$ *dist v y*
unfolding *2* **by** (*intro mono-intros*)
also have ... $\leq$ *3/2* $*$ *C/lambda* $+$ *3/2* $*$ *C/lambda*
apply (*intro mono-intros*)
unfolding *dist-real-def* **using** *u*(*2*) *v*(*2*) *gap*[*OF u*(*1*)] *gap*[*OF v*(*1*)] **by** *auto*
finally have $*$: *dist x y* $\leq$ *3* $*$ *C/lambda* **by** *auto*
have (*1/lambda*) $*$ *dist x y* $-$ *4* $*$ *C* $\leq$ *lambda* $*$ *dist x y* $-$ *4* $*$ *C*
apply (*intro mono-intros*) **by** *auto*
also have ... $\leq$ *lambda* $*$ (*3* $*$ *C/lambda*) $-$ *3* $*$ *C*
apply (*intro mono-intros*)
using $*$ *C* **by** *auto*
also have ... $=$ *0*
using *C* **by** *auto*
also have ... $\leq$ *dist* (*d x*) (*d y*)

by *auto*
finally show *?thesis* **by** *simp*
next
case *3*
then have $u < v$ **using** ‹$u \leq v$› **by** *auto*
then have $*$: *next-in* A $u < v$ **using** $v(1)$ *next-in-basics*(3)[*OF* A ‹$u \in \{a..<b\}$›] *3* **by** *auto*
have *nu*: *next-in* A $u = u + C/lambda$
proof (*rule nx*)
show $u \in A$ **using** $u(1)$ **by** *auto*
show $u \neq b$ **using** $u(1)$ **by** *auto*
show $u \neq b'$
proof
assume H: $u = b'$
have $b < v$ **using** $*$ **unfolding** H nb' **by** *simp*
then show *False* **using** ‹$v \in \{a..<b\}$› **by** *auto*
qed
show $u \neq p\ N$
proof
assume H: $u = p\ N$
have $b' < v$ **using** $*$ **unfolding** H *npN* **by** *simp*
then have *next-in* A $b' \leq v$ **using** *next-in-basics*(3)[*OF* A ‹$b' \in \{a..<b\}$›] v **by** *force*
then show *False* **unfolding** nb' **using** ‹$v \in \{a..<b\}$› **by** *auto*
qed
qed
have *nv*: *next-in* A $v \leq v + 3/2 * C/lambda$ **using** *gap*[*OF* $v(1)$] **by** *auto*

have d: $d\ u = c\ u$ d (*next-in* A u) $= c$ (*next-in* A u) $d\ v = c\ v$ d (*next-in* A v) $= c$ (*next-in* A v)
using ‹$u \in A - \{b\}$› ‹*next-in* A $u \in A$› ‹$v \in A - \{b\}$› ‹*next-in* A $v \in A$› **unfolding** *d-def* **by** *auto*

The interval containing x has length C/λ, while the interval containing y has length at most $\leq 3/2C/\lambda$. Therefore, x is at proportion $2/5$ of the inner point if $x > u + (3/5)C/\lambda$, and y is at proportion $2/5$ of the inner point if $y < v + (2/5) \cdot 3/2 \cdot C/\lambda = v + (3/5)C/\lambda$.

consider $x \leq u + (3/5) * C/lambda \wedge y \leq v + (3/5) * C/lambda$
$\mid x \geq u + (3/5) * C/lambda \wedge y \leq v + (3/5) * C/lambda$
$\mid x \leq u + (3/5) * C/lambda \wedge y \geq v + (3/5) * C/lambda$
$\mid x \geq u + (3/5) * C/lambda \wedge y \geq v + (3/5) * C/lambda$
by *linarith*
then show *?thesis*
proof (*cases*)
case *1*
have $(1/lambda) * dist\ u\ v - C \leq dist\ (c\ u)\ (c\ v)$
apply (*rule quasi-isometry-onD*(2)[*OF assms*(1)])
using ‹$u \in A - \{b\}$› ‹$v \in A - \{b\}$› ‹$A \subseteq \{a..b\}$› **by** *auto*
also have ... $= dist\ (d\ u)\ (d\ v)$

```
      using d by auto
    also have ... ≤ dist (d u) (d x) + dist (d x) (d y) + dist (d y) (d v)
      by (intro mono-intros)
    also have ... ≤ (2 * lambda * dist u x) + dist (d x) (d y) + (2 * lambda
* dist y v)
      apply (intro mono-intros)
     apply (rule lipschitz-onD[OF lip[OF u(1)]]) using u(2) apply auto[1]
using u(2) apply auto[1]
      apply (rule lipschitz-onD[OF lip[OF v(1)]]) using v(2) by auto
    also have ... ≤ (2 * lambda * (3/5 * C/lambda)) + dist (d x) (d y) +
(2 * lambda * (3/5 * C/lambda))
      apply (intro mono-intros)
      unfolding dist-real-def using 1 u v C by auto
    also have ... = 12/5 * C + dist (d x) (d y)
      using C by (auto simp add: algebra-simps divide-simps)
    finally have *: (1/lambda) * dist u v ≤ dist (d x) (d y) + 17/5 * C by
auto

    have (1/lambda) * dist x y ≤ (1/lambda) * (dist u v + dist v y)
      apply (intro mono-intros)
      unfolding dist-real-def using C u(2) v(2) ‹x < y› by auto
    also have ... ≤ (1/lambda) * (dist u v + 3/5 * C/lambda)
      apply (intro mono-intros)
      unfolding dist-real-def using 1 v(2) C by auto
   also have ... = (1/lambda) * dist u v + 3/5 * C * (1/(lambda * lambda))
      using C by (auto simp add: algebra-simps divide-simps)
    also have ... ≤ (1/lambda) * dist u v + 3/5 * C * 1
      apply (intro mono-intros)
   using C by (auto simp add: divide-simps algebra-simps mult-ge1-powers(1))
    also have ... ≤ (dist (d x) (d y) + 17/5 * C) + 3/5 * C * 1
      using * by auto
    finally show ?thesis by auto
  next
    case 2
    have (1/lambda) * dist (next-in A u) v − C ≤ dist (c (next-in A u)) (c
v)
      apply (rule quasi-isometry-onD(2)[OF assms(1)])
      using ‹next-in A u ∈ A› ‹v ∈ A − {b}› ‹A ⊆ {a..b}› by auto
    also have ... = dist (d (next-in A u)) (d v)
      using d by auto
    also have ... ≤ dist (d (next-in A u)) (d x) + dist (d x) (d y) + dist (d
y) (d v)
      by (intro mono-intros)
    also have ... ≤ (2 * lambda * dist (next-in A u) x) + dist (d x) (d y) +
(2 * lambda * dist y v)
      apply (intro mono-intros)
     apply (rule lipschitz-onD[OF lip[OF u(1)]]) using u(2) apply auto[1]
using u(2) apply auto[1]
      apply (rule lipschitz-onD[OF lip[OF v(1)]]) using v(2) by auto
```

also have ... ≤ (*2* ∗ *lambda* ∗ (*2*/*5* ∗ *C*/*lambda*)) + *dist* (*d x*) (*d y*) + (*2* ∗ *lambda* ∗ (*3*/*5* ∗ *C*/*lambda*))
apply (*intro mono-intros*)
unfolding *dist-real-def* **using** *2 u v C nu* **by** *auto*
also have ... = *2* ∗ *C* + *dist* (*d x*) (*d y*)
using *C* **by** (*auto simp add*: *algebra-simps divide-simps*)
finally have ∗: (*1*/*lambda*) ∗ *dist* (*next-in A u*) *v* ≤ *dist* (*d x*) (*d y*) + *3* ∗ *C* **by** *auto*

have (*1*/*lambda*) ∗ *dist x y* ≤ (*1*/*lambda*) ∗ (*dist x* (*next-in A u*) + *dist* (*next-in A u*) *v* + *dist v y*)
apply (*intro mono-intros*)
unfolding *dist-real-def* **using** *C u*(*2*) *v*(*2*) ‹*x* < *y*› **by** *auto*
also have ... ≤ (*1*/*lambda*) ∗ ((*2*/*5* ∗ *C*/*lambda*) + *dist* (*next-in A u*) *v* + (*3*/*5* ∗ *C*/*lambda*))
apply (*intro mono-intros*)
unfolding *dist-real-def* **using** *2 u*(*2*) *v*(*2*) *C nu* **by** *auto*
also have ... = (*1*/*lambda*) ∗ *dist* (*next-in A u*) *v* + *C* ∗ (*1*/(*lambda* ∗ *lambda*))
using *C* **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... ≤ (*1*/*lambda*) ∗ *dist* (*next-in A u*) *v* + *C* ∗ *1*
apply (*intro mono-intros*)
using *C* **by** (*auto simp add*: *divide-simps algebra-simps mult-ge1-powers*(*1*))
also have ... ≤ (*dist* (*d x*) (*d y*) + *3* ∗ *C*) + *C* ∗ *1*
using ∗ **by** *auto*
finally show *?thesis* **by** *auto*
next
case *3*
have (*1*/*lambda*) ∗ *dist u* (*next-in A v*) − *C* ≤ *dist* (*c u*) (*c* (*next-in A v*))
apply (*rule quasi-isometry-onD*(*2*)[*OF assms*(*1*)])
using ‹*u* ∈ *A* − {*b*}› ‹*next-in A v* ∈ *A*› ‹*A* ⊆ {*a*..*b*}› **by** *auto*
also have ... = *dist* (*d u*) (*d* (*next-in A v*))
using *d* **by** *auto*
also have ... ≤ *dist* (*d u*) (*d x*) + *dist* (*d x*) (*d y*) + *dist* (*d y*) (*d* (*next-in A v*))
by (*intro mono-intros*)
also have ... ≤ (*2* ∗ *lambda* ∗ *dist u x*) + *dist* (*d x*) (*d y*) + (*2* ∗ *lambda* ∗ *dist y* (*next-in A v*))
apply (*intro mono-intros*)
apply (*rule lipschitz-onD*[*OF lip*[*OF u*(*1*)]]) **using** *u*(*2*) **apply** *auto*[*1*] **using** *u*(*2*) **apply** *auto*[*1*]
apply (*rule lipschitz-onD*[*OF lip*[*OF v*(*1*)]]) **using** *v*(*2*) **by** *auto*
also have ... ≤ (*2* ∗ *lambda* ∗ (*3*/*5* ∗ *C*/*lambda*)) + *dist* (*d x*) (*d y*) + (*2* ∗ *lambda* ∗ (*9*/*10* ∗ *C*/*lambda*))
apply (*intro mono-intros*)
unfolding *dist-real-def* **using** *3 u v C nv* **by** *auto*
also have ... = *3* ∗ *C* + *dist* (*d x*) (*d y*)
using *C* **by** (*auto simp add*: *algebra-simps divide-simps*)

finally have ∗: (*1/lambda*) ∗ *dist u* (*next-in A v*) ≤ *dist* (*d x*) (*d y*) + *4* ∗ *C* **by** *auto*

have (*1/lambda*) ∗ *dist x y* ≤ (*1/lambda*) ∗ *dist u* (*next-in A v*)
apply (*intro mono-intros*)
unfolding *dist-real-def* **using** *C u*(*2*) *v*(*2*) ‹*x* < *y*› **by** *auto*
also have ... ≤ *dist* (*d x*) (*d y*) + *4* ∗ *C*
using ∗ **by** *auto*
finally show *?thesis* **by** *auto*
next
case *4*
have (*1/lambda*) ∗ *dist* (*next-in A u*) (*next-in A v*) − *C* ≤ *dist* (*c* (*next-in A u*)) (*c* (*next-in A v*))
apply (*rule quasi-isometry-onD*(*2*)[*OF assms*(*1*)])
using ‹*next-in A u* ∈ *A*› ‹*next-in A v* ∈ *A*› ‹*A* ⊆ {*a..b*}› **by** *auto*
also have ... = *dist* (*d* (*next-in A u*)) (*d* (*next-in A v*))
using *d* **by** *auto*
also have ... ≤ *dist* (*d* (*next-in A u*)) (*d x*) + *dist* (*d x*) (*d y*) + *dist* (*d y*) (*d* (*next-in A v*))
by (*intro mono-intros*)
also have ... ≤ (*2* ∗ *lambda* ∗ *dist* (*next-in A u*) *x*) + *dist* (*d x*) (*d y*) + (*2* ∗ *lambda* ∗ *dist y* (*next-in A v*))
apply (*intro mono-intros*)
apply (*rule lipschitz-onD*[*OF lip*[*OF u*(*1*)]]) **using** *u*(*2*) **apply** *auto*[*1*] **using** *u*(*2*) **apply** *auto*[*1*]
apply (*rule lipschitz-onD*[*OF lip*[*OF v*(*1*)]]) **using** *v*(*2*) **by** *auto*
also have ... ≤ (*2* ∗ *lambda* ∗ (*2/5* ∗ *C/lambda*)) + *dist* (*d x*) (*d y*) + (*2* ∗ *lambda* ∗ (*9/10* ∗ *C/lambda*))
apply (*intro mono-intros*)
unfolding *dist-real-def* **using** *4 u v C nu nv* **by** *auto*
also have ... = *13/5* ∗ *C* + *dist* (*d x*) (*d y*)
using *C* **by** (*auto simp add*: *algebra-simps divide-simps*)
finally have ∗: (*1/lambda*) ∗ *dist* (*next-in A u*) (*next-in A v*) ≤ *dist* (*d x*) (*d y*) + *18/5* ∗ *C* **by** *auto*

have (*1/lambda*) ∗ *dist x y* ≤ (*1/lambda*) ∗ (*dist x* (*next-in A u*) + *dist* (*next-in A u*) (*next-in A v*))
apply (*intro mono-intros*)
unfolding *dist-real-def* **using** *C u*(*2*) *v*(*2*) ‹*x* < *y*› **by** *auto*
also have ... ≤ (*1/lambda*) ∗ ((*2/5* ∗*C/lambda*) + *dist* (*next-in A u*) (*next-in A v*))
apply (*intro mono-intros*)
unfolding *dist-real-def* **using** *4 u*(*2*) *v*(*2*) *C nu* **by** *auto*
also have ... = (*1/lambda*) ∗ *dist* (*next-in A u*) (*next-in A v*) + *2/5* ∗ *C* ∗ (*1/*(*lambda* ∗ *lambda*))
using *C* **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... ≤ (*1/lambda*) ∗ *dist* (*next-in A u*) (*next-in A v*) + *2/5* ∗ *C* ∗ *1*
apply (*intro mono-intros*)

using *C* **by** (*auto simp add*: *divide-simps algebra-simps mult-ge1-powers*(*1*))
also have ... $\leq$ (*dist* (*d x*) (*d y*) + *18/5* $*$ *C*) + *2/5* $*$ *C* $*$ *1*
using $*$ **by** *auto*
finally show *?thesis* **by** *auto*
qed
qed
qed
show *dist* (*d x*) (*d y*) $\geq$ (*1/lambda*) $*$ *dist x y* $-$ *4* $*$ *C* **if** *H*: *x* $\in$ {*a..b*} *y* $\in$ {*a..b*} **for** *x y*
proof $-$
consider *x* < *y* | *x* = *y* | *x* > *y* **by** *linarith*
then show *?thesis*
proof (*cases*)
case *1*
then show *?thesis* **using** *J*[*OF H*(*1*) *H*(*2*) *1*] **by** *simp*
next
case *2*
show *?thesis* **unfolding** *2* **using** *C* **by** *auto*
next
case *3*
show *?thesis* **using** *J*[*OF H*(*2*) *H*(*1*) *3*] **by** (*simp add*: *metric-space-class.dist-commute*)
qed
qed
qed

We have proved that *d* has all the properties we wanted.

then have *continuous-on* {*a..b*} *d* $\wedge$ *d a* = *c a* $\wedge$ *d b* = *c b*
$\wedge$ *lambda* (*4* $*$ *C*)$-$*quasi-isometry-on* {*a..b*} *d*
$\wedge$ ($\forall$ *x*$\in${*a..b*}. *dist* (*c x*) (*d x*) $\leq$ *4* $*$*C*)
$\wedge$ (*2*$*$*lambda*)$-$*lipschitz-on* {*a..b*} *d*
$\wedge$ *hausdorff-distance* (*c*‘{*a..b*}) (*d*‘{*a..b*}) $\leq$ *2* $*$ *C*
using *dist-c-d* ‹*continuous-on* {*a..b*} *d*› ‹*d a* = *c a*› ‹*d b* = *c b*› ‹(*2*$*$*lambda*)$-$*lipschitz-on* {*a..b*} *d*›
‹*hausdorff-distance* (*c*‘{*a..b*}) (*d*‘{*a..b*}) $\leq$ *2* $*$ *C*› **by** *auto*
then show *?thesis* **by** *auto*
qed
qed

end

7 The metric completion of a metric space

theory *Metric-Completion*
imports *Isometries*
begin

Any metric space can be completed, by adding the missing limits of Cauchy sequences. Formally, there exists an isometric embedding of the space in a complete space, with dense image. In this paragraph, we construct this metric completion. This is exactly the same construction as the way in which real numbers are constructed from rational numbers.

7.1 Definition of the metric completion

quotient-type (**overloaded**) *'a metric-completion* =
  nat ⇒ (*'a::metric-space*) / *partial*: λ*u v*. (*Cauchy u*) ∧ (*Cauchy v*) ∧ (λ*n*. *dist* (*u n*) (*v n*)) ⟶ *0*
unfolding *part-equivp-def* **proof**(*auto intro*!: *ext*)
  show ∃ *x*. *Cauchy x*
   by (*rule exI*[*of* - λ-. *undefined*]) (*simp add*: *convergent-Cauchy convergent-const*)
  fix *x y z*::*nat* ⇒ *'a* **assume** *H*: (λ*n*. *dist* (*x n*) (*y n*)) ⟶ *0*
                 (λ*n*. *dist* (*x n*) (*z n*)) ⟶ *0*
  have ∗: (λ*n*. *dist* (*x n*) (*y n*) + *dist* (*x n*) (*z n*)) ⟶ *0* + *0*
    by (*rule tendsto-add*) (*auto simp add*: *H*)
  show (λ*n*. *dist* (*y n*) (*z n*)) ⟶ *0*
    apply (*rule tendsto-sandwich*[*of* λ-. *0* - - λ*n*. *dist* (*x n*) (*y n*) + *dist* (*x n*) (*z n*)])
    using ∗ **by** (*auto simp add*: *dist-triangle3*)
next
  fix *x y z*::*nat* ⇒ *'a* **assume** *H*: (λ*n*. *dist* (*x n*) (*y n*)) ⟶ *0*
                 (λ*n*. *dist* (*y n*) (*z n*)) ⟶ *0*
  have ∗: (λ*n*. *dist* (*x n*) (*y n*) + *dist* (*y n*) (*z n*)) ⟶ *0* + *0*
    by (*rule tendsto-add*) (*auto simp add*: *H*)
  show (λ*n*. *dist* (*x n*) (*z n*)) ⟶ *0*
    apply (*rule tendsto-sandwich*[*of* λ-. *0* - - λ*n*. *dist* (*x n*) (*y n*) + *dist* (*y n*) (*z n*)])
    using ∗ **by** (*auto simp add*: *dist-triangle*)
next
  fix *x y*::*nat* ⇒ *'a* **assume** *H*: *Cauchy x*
    (λ*v*. *Cauchy v* ∧ (λ*n*. *dist* (*x n*) (*v n*)) ⟶ *0*) = (λ*v*. *Cauchy v* ∧ (λ*n*. *dist* (*y n*) (*v n*)) ⟶ *0*)
  have *Cauchy x* ∧ (λ*n*. *dist* (*x n*) (*x n*)) ⟶ *0* **using** *H* **by** *auto*
  then have (λ*n*. *dist* (*y n*) (*x n*))⟶ *0* **using** *H* **by** *meson*
  moreover have *dist* (*x n*) (*y n*) = *dist* (*y n*) (*x n*) **for** *n* **using** *dist-commute* **by** *auto*
  ultimately show (λ*n*. *dist* (*x n*) (*y n*)) ⟶ *0* **by** *auto*
qed

We have to show that the metric completion is indeed a metric space, that the original space embeds isometrically into it, and that it is complete. Before we prove these statements, we start with two simple lemmas that will be needed later on.

lemma *convergent-Cauchy-dist*:
  fixes *u v*::*nat* ⇒ (*'a::metric-space*)
  assumes *Cauchy u Cauchy v*
  shows *convergent* (λ*n*. *dist* (*u n*) (*v n*))
proof (*rule real-Cauchy-convergent*, *intro CauchyI*)

```
fix e::real assume e > 0
 obtain Nu where Nu: ∀ n ≥ Nu. ∀ m ≥ Nu. dist (u n) (u m) < e/2 using
assms(1)
   by (metis ‹0 < e› less-divide-eq-numeral1(1) metric-CauchyD mult-zero-left)
 obtain Nv where Nv: ∀ n ≥ Nv. ∀ m ≥ Nv. dist (v n) (v m) < e/2 using
assms(2)
   by (metis ‹0 < e› less-divide-eq-numeral1(1) metric-CauchyD mult-zero-left)
 define M where M = max Nu Nv
 {
   fix n m assume H: n ≥ M m ≥ M
   have ∗: dist (u n) (u m) < e/2 dist (v n) (v m) < e/2
     using Nu Nv H unfolding M-def by auto
   have dist (u m) (v m) − dist (u n) (v n) ≤ dist (u m) (u n) + dist (v n) (v m)
    by (simp add: algebra-simps) (metis add-le-cancel-left dist-commute dist-triangle2
dist-triangle-le)
   also have ... < e/2 + e/2
     using ∗ by (simp add: dist-commute)
   finally have A: dist (u m) (v m) − dist (u n) (v n) < e by simp

   have dist (u n) (v n) − dist (u m) (v m) ≤ dist (u m) (u n) + dist (v n) (v m)
    by (simp add: algebra-simps) (metis add-le-cancel-left dist-commute dist-triangle2
dist-triangle-le)
   also have ... < e/2 + e/2
     using ∗ by (simp add: dist-commute)
   finally have dist (u n) (v n) − dist (u m) (v m) < e by simp
   then have norm(dist (u m) (v m) − dist (u n) (v n)) < e using A by auto
 }
 then show ∃ M. ∀ m ≥ M. ∀ n ≥ M. norm (dist (u m) (v m) − dist (u n) (v
n)) < e
   by auto
qed

lemma convergent-add-null:
 fixes u v::nat ⇒ ('a::real-normed-vector)
 assumes convergent u
        (λn. v n − u n) ⟶ 0
 shows convergent v lim v = lim u
proof −
 have (λn. u n + (v n − u n)) ⟶ lim u + 0
   apply (rule tendsto-add) using assms convergent-LIMSEQ-iff by auto
 then have ∗: v ⟶ lim u by auto
 show convergent v using ∗ by (simp add: Lim-def convergentI)
 show lim v = lim u using ∗ by (simp add: limI)
qed
```

Let us now prove that the metric completion is a metric space: the distance between two Cauchy sequences is the limit of the distances of points in the sequence. The convergence follows from Lemma `convergent_Cauchy_dist` above.

instantiation *metric-completion* :: (*metric-space*) *metric-space*
begin

lift-definition *dist-metric-completion*::(*'a*::*metric-space*) *metric-completion* ⇒ *'a metric-completion* ⇒ *real*
is *λx y. lim* (*λn. dist* (*x n*) (*y n*))
proof −
fix *x y z t*::*nat* ⇒ *'a* **assume** *H*: *Cauchy x* ∧ *Cauchy y* ∧ (*λn. dist* (*x n*) (*y n*)) ⟶ *0*
Cauchy z ∧ *Cauchy t* ∧ (*λn. dist* (*z n*) (*t n*)) ⟶ *0*
show *lim* (*λn. dist* (*x n*) (*z n*)) = *lim* (*λn. dist* (*y n*) (*t n*))
proof (*rule convergent-add-null*(*2*))
show *convergent* (*λn. dist* (*y n*) (*t n*))
apply (*rule convergent-Cauchy-dist*) **using** *H* **by** *auto*

have *a*: (*λn.* − *dist* (*t n*) (*z n*) − *dist* (*x n*) (*y n*)) ⟶ −*0* −*0*
apply (*intro tendsto-intros*) **using** *H* **by** (*auto simp add*: *dist-commute*)
have *b*:(*λn. dist* (*t n*) (*z n*) + *dist* (*x n*) (*y n*)) ⟶ *0* + *0*
apply (*rule tendsto-add*) **using** *H* **by** (*auto simp add*: *dist-commute*)
have *I*: *dist* (*x n*) (*z n*) ≤ *dist* (*t n*) (*y n*) + (*dist* (*t n*) (*z n*) + *dist* (*x n*) (*y n*)) **for** *n*
using *dist-triangle*[*of x n z n y n*] *dist-triangle*[*of y n z n t n*]
by (*auto simp add*: *dist-commute add.commute*)
show (*λn. dist* (*x n*) (*z n*) − *dist* (*y n*) (*t n*)) ⟶ *0*
apply (*rule tendsto-sandwich*[*of λn.* −(*dist* (*x n*) (*y n*) + *dist* (*z n*) (*t n*)) - - *λn. dist* (*x n*) (*y n*) + *dist* (*z n*) (*t n*)])
apply (*auto intro*!: *always-eventually simp add*: *algebra-simps dist-commute I*)
apply (*meson add-left-mono dist-triangle3 dist-triangle-le*)
using *a b* **by** *auto*
qed
qed

lemma *dist-metric-completion-limit*:
fixes *x y*::*'a metric-completion*
shows (*λn. dist* (*rep-metric-completion x n*) (*rep-metric-completion y n*)) ⟶ *dist x y*
proof −
have *C*: *Cauchy* (*rep-metric-completion x*) *Cauchy* (*rep-metric-completion y*)
using *Quotient3-metric-completion Quotient3-rep-reflp* **by** *fastforce*+
show *?thesis*
unfolding *dist-metric-completion-def* **using** *C* **apply** *auto*
using *convergent-Cauchy-dist*[*OF C*] *convergent-LIMSEQ-iff* **by** *force*
qed

lemma *dist-metric-completion-limit'*:
fixes *x y*::*nat* ⇒ *'a*
assumes *Cauchy x Cauchy y*
shows (*λn. dist* (*x n*) (*y n*)) ⟶ *dist* (*abs-metric-completion x*) (*abs-metric-completion*

y)
apply (*subst dist-metric-completion.abs-eq*)
using *assms convergent-Cauchy-dist*[*OF assms*] **by** (*auto simp add*: *convergent-LIMSEQ-iff*)

To define a metric space in the current library of Isabelle/HOL, one should also introduce a uniformity structure and a topology, as follows (they are prescribed by the distance):

definition *uniformity-metric-completion*::((′*a metric-completion*) × (′*a metric-completion*)) *filter*
  where *uniformity-metric-completion* = (*INF e*∈{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open-metric-completion* :: ′*a metric-completion set* ⇒ *bool*
  where *open-metric-completion U* = (∀ *x*∈*U*. *eventually* (λ(*x*′, *y*). *x*′ = *x* ⟶ *y* ∈ *U*) *uniformity*)

instance proof
  fix *x y*::′*a metric-completion*
  have *C*: *Cauchy* (*rep-metric-completion x*) *Cauchy* (*rep-metric-completion y*)
    using *Quotient3-metric-completion Quotient3-rep-reflp* **by** *fastforce*+
  show (*dist x y* = *0*) = (*x* = *y*)
    apply (*subst Quotient3-rel-rep*[*OF Quotient3-metric-completion*, *symmetric*])
    unfolding *dist-metric-completion-def* **using** *C* **apply** *auto*
    using *convergent-Cauchy-dist*[*OF C*] *convergent-LIMSEQ-iff* **apply** *force*
    by (*simp add*: *limI*)
next
  fix *x y z*::′*a metric-completion*
  have *a*: (λ*n*. *dist* (*rep-metric-completion x n*) (*rep-metric-completion y n*)) ⟶ *dist x y*
    using *dist-metric-completion-limit* **by** *auto*
  have *b*: (λ*n*. *dist* (*rep-metric-completion x n*) (*rep-metric-completion z n*) + *dist* (*rep-metric-completion y n*) (*rep-metric-completion z n*))
      ⟶ *dist x z* + *dist y z*
    apply (*rule tendsto-add*) **using** *dist-metric-completion-limit* **by** *auto*
  show *dist x y* ≤ *dist x z* + *dist y z*
    by (*rule LIMSEQ-le*[*OF a b*], *rule exI*[*of - 0*], *auto simp add*: *dist-triangle2*)
qed (*auto simp add*: *uniformity-metric-completion-def open-metric-completion-def*)
end

Let us now show that the distance thus defined on the metric completion is indeed complete. This is essentially by design.

instance *metric-completion* :: (*metric-space*) *complete-space*
proof
  fix *X*::*nat* ⇒ ′*a metric-completion* **assume** *Cauchy X*
  have ∗: ∃ *N*. ∀ *n* ≥ *N*. *dist* (*rep-metric-completion* (*X k*) *N*) (*rep-metric-completion* (*X k*) *n*) < (*1*/*Suc k*) **for** *k*
  proof −
    have *Cauchy* (*rep-metric-completion* (*X k*))
      using *Quotient3-metric-completion Quotient3-rep-reflp* **by** *fastforce*+

then have $\exists N. \forall m \geq N. \forall n \geq N.$ *dist* (*rep-metric-completion* (*X k*) *m*) (*rep-metric-completion* (*X k*) *n*) < (*1/Suc k*)
unfolding *Cauchy-def* **by** *auto*
then show *?thesis* **by** *auto*
qed
have $\exists N. \forall k. \forall n \geq N\, k.$ *dist* (*rep-metric-completion* (*X k*) (*N k*)) (*rep-metric-completion* (*X k*) *n*) < (*1/Suc k*)
apply (*rule choice*) **using** ∗ **by** *auto*
then obtain *N*::*nat* ⇒ *nat* **where**
N: *dist* (*rep-metric-completion* (*X k*) (*N k*)) (*rep-metric-completion* (*X k*) *n*) < (*1/Suc k*) **if** $n \geq N\, k$ **for** *n k*
by *auto*
define *u* **where** *u* = (λ*k*. *rep-metric-completion* (*X k*) (*N k*))

have *Cauchy u*
proof (*rule metric-CauchyI*)
fix *e*::*real* **assume** *e* > *0*
obtain *K*::*nat* **where** *K* > *4/e* **using** *reals-Archimedean2* **by** *blast*
obtain *L*::*nat* **where** *L*: $\forall m \geq L. \forall n \geq L.$ *dist* (*X m*) (*X n*) < *e/2*
using *metric-CauchyD*[*OF* ‹*Cauchy X*›, *of e/2*] ‹*e* > *0*› **by** *auto*
{
fix *m n* **assume** *m* ≥ *max K L* *n* ≥ *max K L*
then have *dist* (*X m*) (*X n*) < *e/2* **using** *L* **by** *auto*
then have *eventually* (λ*p*. *dist* (*rep-metric-completion* (*X m*) *p*) (*rep-metric-completion* (*X n*) *p*) < *e/2*) *sequentially*
using *dist-metric-completion-limit*[*of X m X n*] **by** (*metis order-tendsto-iff*)
then obtain *p* **where** *p*: *p* ≥ *max* (*N m*) (*N n*) *dist* (*rep-metric-completion* (*X m*) *p*) (*rep-metric-completion* (*X n*) *p*) < *e/2*
using *eventually-False-sequentially eventually-elim2 eventually-ge-at-top* **by** *blast*
have *dist* (*u m*) (*rep-metric-completion* (*X m*) *p*) < *1* / *real* (*Suc m*)
unfolding *u-def* **using** *N*[*of m p*] *p*(*1*) **by** *auto*
also have ... < *e/4*
using ‹*m* ≥ *max K L*› ‹*K* > *4/e*› ‹*e* > *0*› **apply** (*auto simp add*: *divide-simps algebra-simps*)
by (*metis leD le-less-trans less-add-same-cancel2 linear of-nat-le-iff mult-le-cancel-left-pos*)
finally have *Im*: *dist* (*u m*) (*rep-metric-completion* (*X m*) *p*) < *e/4* **by** *simp*
have *dist* (*u n*) (*rep-metric-completion* (*X n*) *p*) < *1* / *real* (*Suc n*)
unfolding *u-def* **using** *N*[*of n p*] *p*(*1*) **by** *auto*
also have ... < *e/4*
using ‹*n* ≥ *max K L*› ‹*K* > *4/e*› ‹*e* > *0*› **apply** (*auto simp add*: *divide-simps algebra-simps*)
by (*metis leD le-less-trans less-add-same-cancel2 linear of-nat-le-iff mult-le-cancel-left-pos*)
finally have *In*: *dist* (*u n*) (*rep-metric-completion* (*X n*) *p*) < *e/4* **by** *simp*

have *dist* (*u m*) (*u n*) ≤ *dist* (*u m*) (*rep-metric-completion* (*X m*) *p*)
+ *dist* (*rep-metric-completion* (*X m*) *p*) (*rep-metric-completion* (*X n*) *p*) + *dist* (*rep-metric-completion* (*X n*) *p*) (*u n*)
by (*metis add.commute add-left-mono dist-commute dist-triangle-le dist-triangle*)

```
      also have ... < e/4 + e/2 + e/4
        using In Im p(2) by (simp add: dist-commute)
      also have ... = e by auto
      finally have dist (u m) (u n) < e by auto
    }
    then show ∃M. ∀m ≥ M. ∀n ≥ M. dist (u m) (u n) < e by meson
  qed
  have ∗: (λn. dist (abs-metric-completion u) (X n)) ⟶ 0
  proof (rule order-tendstoI, auto simp add: less-le-trans eventually-sequentially)
    fix e::real assume e > 0
    obtain K::nat where K > 4/e using reals-Archimedean2 by blast
    obtain L::nat where L: ∀m ≥ L. ∀n ≥ L. dist (u m) (u n) < e/4
      using metric-CauchyD[OF ‹Cauchy u›, of e/4] ‹e > 0› by auto
    {
      fix n assume n: n ≥ max K L
      {
        fix p assume p: p ≥ max (N n) L
        have dist (u n) (rep-metric-completion (X n) p) < 1/(Suc n)
          unfolding u-def using N p by simp
        also have ... < e/4
            using ‹n ≥ max K L› ‹K > 4/e› ‹e > 0› apply (auto simp add:
divide-simps algebra-simps)
             by (metis leD le-less-trans less-add-same-cancel2 linear of-nat-le-iff
mult-le-cancel-left-pos)
        finally have ∗: dist (u n) (rep-metric-completion (X n) p) < e/4
          by fastforce

        have dist (u p) (rep-metric-completion (X n) p) ≤ dist (u p) (u n) + dist
(u n) (rep-metric-completion (X n) p)
          using dist-triangle by auto
        also have ... < e/4 + e/4 using ∗ L n p by force
        finally have dist (u p) (rep-metric-completion (X n) p) ≤ e/2 by auto
      }
      then have A: eventually (λp. dist (u p) (rep-metric-completion (X n) p) ≤
e/2) sequentially
        using eventually-at-top-linorder by blast
    have B: (λp. dist (u p) (rep-metric-completion (X n) p)) ⟶ dist (abs-metric-completion
u) (X n)
      using dist-metric-completion-limit'[OF ‹Cauchy u›, of rep-metric-completion
(X n)]
        unfolding Quotient3-abs-rep[OF Quotient3-metric-completion, of X n]
        using Quotient3-rep-reflp[OF Quotient3-metric-completion] by auto
      have dist (abs-metric-completion u) (X n) ≤ e/2
      apply (rule LIMSEQ-le-const2[OF B]) using A unfolding eventually-sequentially
by auto
      then have dist (abs-metric-completion u) (X n) < e using ‹e > 0› by auto
    }
    then show ∃N. ∀n ≥ N. dist (abs-metric-completion u) (X n) < e
      by blast
```

qed
have $X \longrightarrow$ *abs-metric-completion u*
apply (*rule tendstoI*) **using** $*$ **by** (*auto simp add*: *order-tendsto-iff dist-commute*)
then show *convergent X* **unfolding** *convergent-def* **by** *auto*
qed

7.2 Isometric embedding of a space in its metric completion

The canonical embedding of a space into its metric completion is obtained by taking the Cauchy sequence which is constant, equal to the given point. This is indeed an isometric embedding with dense image, as we prove in the lemmas below.

definition *to-metric-completion*::($'a$::*metric-space*) $\Rightarrow$ $'a$ *metric-completion*
where *to-metric-completion x* = *abs-metric-completion* ($\lambda n.\ x$)

lemma *to-metric-completion-isometry*:
isometry-on UNIV to-metric-completion
proof (*rule isometry-onI*)
fix *x y*::$'a$
have ($\lambda n.$ *dist* (x) (y)) $\longrightarrow$ *dist* (*to-metric-completion x*) (*to-metric-completion y*)
unfolding *to-metric-completion-def* **apply** (*rule dist-metric-completion-limit'*)
unfolding *Cauchy-def* **by** *auto*
then show *dist* (*to-metric-completion x*) (*to-metric-completion y*) = *dist x y*
by (*simp add*: *LIMSEQ-const-iff*)
qed

lemma *to-metric-completion-dense*:
assumes *open U* $U \neq \{\}$
shows $\exists x.$ *to-metric-completion x* $\in U$
proof −
obtain *y* **where** $y \in U$ **using** ‹$U \neq \{\}$› **by** *auto*
obtain *e*::*real* **where** *e*: $e > 0$ $\bigwedge z.$ *dist z y* $< e \Longrightarrow z \in U$
using ‹$y \in U$› ‹*open U*› **by** (*metis open-dist*)
have $*$: *Cauchy* (*rep-metric-completion y*)
using *Quotient3-metric-completion Quotient3-rep-reflp* **by** *fastforce*
then obtain *N* **where** *N*: $\forall n \geq N.\ \forall m \geq N.$ *dist* (*rep-metric-completion y n*) (*rep-metric-completion y m*) $< e/2$
using ‹$e > 0$› **unfolding** *Cauchy-def* **by** (*meson divide-pos-pos zero-less-numeral*)
define *x* **where** *x* = *rep-metric-completion y N*
have ($\lambda n.$ *dist x* (*rep-metric-completion y n*)) $\longrightarrow$ *dist* (*to-metric-completion x*) (*abs-metric-completion* (*rep-metric-completion y*))
unfolding *to-metric-completion-def* **apply** (*rule dist-metric-completion-limit'*)
using $*$ **unfolding** *Cauchy-def* **by** *auto*
then have ($\lambda n.$ *dist x* (*rep-metric-completion y n*)) $\longrightarrow$ *dist* (*to-metric-completion x*) *y*
unfolding *Quotient3-abs-rep*[*OF Quotient3-metric-completion*] **by** *simp*
moreover have *eventually* ($\lambda n.$ *dist x* (*rep-metric-completion y n*) $\leq e/2$) *se-*

quentially
  unfolding *eventually-sequentially x-def* **apply** (*rule exI*[*of - N*]) **using** *N less-imp-le* **by** *auto*
  ultimately have *dist* (*to-metric-completion x*) $y \leq e/2$
    using *LIMSEQ-le-const2* **unfolding** *eventually-sequentially* **by** *metis*
  then have *to-metric-completion* $x \in U$
    using *e* **by** *auto*
  then show *?thesis* **by** *auto*
qed

lemma *to-metric-completion-dense'*:
  closure (*range to-metric-completion*) = *UNIV*
apply (*auto simp add*: *closure-iff-nhds-not-empty*) **using** *to-metric-completion-dense*
by *fastforce*

The main feature of the completion is that a uniformly continuous function on the original space can be extended to a uniformly continuous function on the completion, i.e., it can be written as the composition of a new function and of the inclusion `to_metric_completion`.

lemma *lift-to-metric-completion*:
  fixes *f*::(*'a*::*metric-space*) ⇒ (*'b*::*complete-space*)
  assumes *uniformly-continuous-on UNIV f*
  shows ∃ *g*. (*uniformly-continuous-on UNIV g*)
        ∧ (*f* = *g o to-metric-completion*)
        ∧ (∀ x ∈ *range to-metric-completion*. *g x* = *f* (*inv to-metric-completion x*))
proof −
  define *I*::*'a metric-completion* ⇒ *'a* **where** *I* = *inv to-metric-completion*
  have *uniformly-continuous-on* (*range to-metric-completion*) *I*
   using *isometry-on-uniformly-continuous*[*OF isometry-on-inverse*(*1*)[*OF to-metric-completion-isometry*]] *I-def*
    by *auto*
  then have *UC*: *uniformly-continuous-on* (*range to-metric-completion*) (λ*x*. *f* (*I x*))
    using *assms uniformly-continuous-on-compose*
     by (*metis I-def bij-betw-imp-surj-on bij-betw-inv-into isometry-on-inverse*(*4*) *to-metric-completion-isometry*)
 obtain *g* **where** *g*: *uniformly-continuous-on* (*closure*(*range to-metric-completion*)) *g*
          $\bigwedge x$. x ∈ *range to-metric-completion* ⟹ *f* (*I x*) = *g x*
    using *uniformly-continuous-on-extension-on-closure*[*OF UC*] **by** *metis*
  have *uniformly-continuous-on UNIV g*
    using *to-metric-completion-dense' g*(*1*) **by** *metis*
  moreover have *f x* = *g* (*to-metric-completion x*) **for** *x*
   using *g*(*2*) **by** (*metis I-def UNIV-I isometry-on-inverse*(*2*) *range-eqI to-metric-completion-isometry*)
 moreover have *g x* = *f* (*inv to-metric-completion x*) **if** x ∈ *range to-metric-completion* **for** *x*
    using *I-def g*(*2*) *that* **by** *auto*
  ultimately show *?thesis* **unfolding** *comp-def* **by** *auto*

qed

When the function is an isometry, the lifted function is also an isometry (and its range is the closure of the range of the original function). This shows that the metric completion is unique, up to isometry:

lemma *lift-to-metric-completion-isometry*:
  fixes *f*::(*'a*::*metric-space*) ⇒ (*'b*::*complete-space*)
  assumes *isometry-on UNIV f*
  shows ∃ *g*. *isometry-on UNIV g*
          ∧ *range g* = *closure*(*range f*)
          ∧ *f* = *g o to-metric-completion*
          ∧ (∀ *x* ∈ *range to-metric-completion*. *g x* = *f* (*inv to-metric-completion x*))
proof −
 have ∗: *uniformly-continuous-on UNIV f* **using** *assms isometry-on-uniformly-continuous* **by** *force*
  obtain *g* **where** *g*: *uniformly-continuous-on UNIV g*
                    f = *g o to-metric-completion*
                    ⋀*x*. *x* ∈ *range to-metric-completion* ⟹ *g x* = *f* (*inv to-metric-completion x*)
    using *lift-to-metric-completion*[*OF* ∗] **by** *blast*
  have ∗: *isometry-on* (*range to-metric-completion*) *g*
    apply (*rule isometry-on-cong*[*OF* - *g*(*3*)], *rule isometry-on-compose*[*of* - - *f*])
      using *assms isometry-on-inverse*[*OF to-metric-completion-isometry*] *isometry-on-subset* **by** (*auto*) (*fastforce*)
  then have *isometry-on UNIV g*
   unfolding *to-metric-completion-dense'*[*symmetric*] **apply** (*rule isometry-on-closure*)
   using *continuous-on-subset*[*OF uniformly-continuous-imp-continuous*[*OF g*(*1*)]] **by** *auto*

  have *g'*(*range to-metric-completion*) ⊆ *range f*
    using *g* **unfolding** *comp-def* **by** *auto*
   moreover have *g'*(*closure* (*range to-metric-completion*)) ⊆ *closure* (*g'*(*range to-metric-completion*))
    using *uniformly-continuous-imp-continuous*[*OF g*(*1*)]
   by (*meson closed-closure closure-subset continuous-on-subset image-closure-subset top-greatest*)
  ultimately have *range g* ⊆ *closure* (*range f*)
    unfolding *to-metric-completion-dense'* **by** (*simp add*: *g*(*2*) *image-comp*)

  have *range f* ⊆ *range g*
    using *g*(*2*) **by** *auto*
  moreover have *closed* (*range g*)
     using *isometry-on-complete-image*[*OF* ‹*isometry-on UNIV g*›] **by** (*simp add*: *complete-eq-closed*)
  ultimately have *closure* (*range f*) ⊆ *range g*
    by (*simp add*: *closure-minimal*)
  then have *range g* = *closure* (*range f*)
    using ‹*range g* ⊆ *closure* (*range f*)› **by** *auto*
  then show *?thesis* **using** ‹*isometry-on UNIV g*› *g* **by** *metis*

qed

7.3 The metric completion of a second countable space is second countable

We want to show that the metric completion of a second countable space is still second countable. This is most easily expressed using the fact that a metric space is second countable if and only if there exists a dense countable subset. We prove the equivalence in the next lemma, and use it then to prove that the metric completion is still second countable.

```
lemma second-countable-iff-dense-countable-subset:
  (∃ B::'a::metric-space set set. countable B ∧ topological-basis B)
    ⟷ (∃ A::'a set. countable A ∧ closure A = UNIV)
proof
  assume ∃ B::'a set set. countable B ∧ topological-basis B
  then obtain B::'a set set where countable B topological-basis B by auto
  define A where A = (λU. SOME x. x ∈ U)`B
  have countable A unfolding A-def using ‹countable B› by auto
  moreover have closure A = UNIV
  proof (auto simp add: closure-approachable)
    fix x::'a and e::real assume e > 0
    obtain U where U ∈ B x ∈ U U ⊆ ball x e
      by (rule topological-basisE[OF ‹topological-basis B›, of ball x e x], auto simp
add: ‹e > 0›)
    define y where y = (λU. SOME x. x ∈ U) U
    have y ∈ U unfolding y-def using ‹x ∈ U› some-in-eq by fastforce
    then have dist y x < e
      using ‹U ⊆ ball x e› by (metis dist-commute mem-ball subset-iff)
    moreover have y ∈ A unfolding A-def y-def using ‹U ∈ B› by auto
    ultimately show ∃ y∈A. dist y x < e by auto
  qed
  ultimately show ∃ A::'a set. countable A ∧ closure A = UNIV by auto
next
  assume ∃ A::'a set. countable A ∧ closure A = UNIV
  then obtain A::'a set where countable A closure A = UNIV by auto
  define B where B = (λ(x, (n::nat)). ball x (1/n))`(A × UNIV)
  have countable B unfolding B-def using ‹countable A› by auto
  moreover have topological-basis B
  proof (rule topological-basisI)
    fix x::'a and U assume x ∈ U open U
    then obtain e where e > 0 ball x e ⊆ U
      using openE by blast
    obtain n::nat where n > 2/e using reals-Archimedean2 by auto
    then have n > 0 using ‹e > 0› not-less by fastforce
    then have 1/n > 0 using zero-less-divide-iff by fastforce
    then obtain y where y: y ∈ A dist x y < 1/n
      by (metis ‹closure A = UNIV› UNIV-I closure-approachable dist-commute)
    then have ball y (1/n) ∈ B unfolding B-def by auto
```

moreover have $x \in$ *ball* y $(1/n)$ **using** $y(2)$ **by** (*auto simp add*: *dist-commute*)
moreover have *ball* y $(1/n) \subseteq U$
proof (*auto*)
fix z **assume** z: *dist* y $z < 1/n$
have *dist* z $x \leq$ *dist* z y + *dist* y x **using** *dist-triangle* **by** *auto*
also have ... $< 1/n + 1/n$ **using** z $y(2)$ **by** (*auto simp add*: *dist-commute*)
also have ... $< e$
using ‹$n > 2/e$› ‹$e > 0$› ‹$n > 0$› **by** (*auto simp add*: *divide-simps mult.commute*)
finally have $z \in$ *ball* x e **by** (*auto simp add*: *dist-commute*)
then show $z \in U$ **using** ‹*ball* x $e \subseteq U$› **by** *auto*
qed
ultimately show $\exists V \in B.\ x \in V \wedge V \subseteq U$ **by** *metis*
qed (*auto simp add*: *B-def*)
ultimately show $\exists B$::*'a set set*. *countable* $B \wedge$ *topological-basis* B **by** *auto*
qed

lemma *second-countable-metric-dense-subset*:
$\exists A$::*'a*::{*metric-space*, *second-countable-topology*} *set*. *countable* $A \wedge$ *closure* A = *UNIV*
using *ex-countable-basis* **by** (*auto simp add*: *second-countable-iff-dense-countable-subset*[*symmetric*])

instance *metric-completion*::({*metric-space*, *second-countable-topology*}) *second-countable-topology*
proof
obtain A::*'a set* **where** *countable* A *closure* A = *UNIV*
using *second-countable-metric-dense-subset* **by** *auto*
define *Ab* **where** *Ab* = *to-metric-completion*‘A
have *range to-metric-completion* $\subseteq$ *closure Ab*
unfolding *Ab-def*
by (*metis* ‹*closure* A = *UNIV*› *isometry-on-continuous*[*OF to-metric-completion-isometry*] *closed-closure closure-subset image-closure-subset*)
then have *closure Ab* = *UNIV*
by (*metis* (*no-types*) *to-metric-completion-dense'*[*symmetric*] ‹*range to-metric-completion* $\subseteq$ *closure Ab*› *closure-closure closure-mono top.extremum-uniqueI*)
moreover have *countable Ab* **unfolding** *Ab-def* **using** ‹*countable* A› **by** *auto*
ultimately have $\exists$ *Ab*::*'a metric-completion set*. *countable Ab* $\wedge$ *closure Ab* = *UNIV*
by *auto*
then show $\exists B$::*'a metric-completion set set*. *countable* $B \wedge$ *open* = *generate-topology* B
using *second-countable-iff-dense-countable-subset topological-basis-imp-subbasis*
by *auto*
qed

instance *metric-completion*::({*metric-space*, *second-countable-topology*}) *polish-space*
by *standard*

end

8 Gromov hyperbolic spaces

theory *Gromov-Hyperbolicity*
 imports *Isometries Metric-Completion*
begin

8.1 Definition, basic properties

Although we will mainly work with type classes later on, we introduce the definition of hyperbolicity on subsets of a metric space.

A set is δ-hyperbolic if it satisfies the following inequality. It is very obscure at first sight, but we will see several equivalent characterizations later on. For instance, a space is hyperbolic (maybe for a different constant δ) if all geodesic triangles are thin, i.e., every side is close to the union of the two other sides. This definition captures the main features of negative curvature at a large scale, and has proved extremely fruitful and influential.

Two important references on this topic are [GdlH90] and [BH99]. We will sometimes follow them, sometimes depart from them.

definition *Gromov-hyperbolic-subset*::*real* ⇒ (*'a*::*metric-space*) *set* ⇒ *bool*
 where *Gromov-hyperbolic-subset delta A* = (∀ *x*∈*A*. ∀ *y*∈*A*. ∀ *z*∈*A*. ∀ *t*∈*A*. *dist x y* + *dist z t* ≤ *max* (*dist x z* + *dist y t*) (*dist x t* + *dist y z*) + *2* ∗ *delta*)

lemma *Gromov-hyperbolic-subsetI* [*intro*]:
 assumes ⋀*x y z t*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *z* ∈ *A* ⟹ *t* ∈ *A* ⟹ *dist x y* + *dist z t* ≤ *max* (*dist x z* + *dist y t*) (*dist x t* + *dist y z*) + *2* ∗ *delta*
 shows *Gromov-hyperbolic-subset delta A*
using *assms* **unfolding** *Gromov-hyperbolic-subset-def* **by** *auto*

When the four points are not all distinct, the above inequality is always satisfied for $\delta = 0$.

lemma *Gromov-hyperbolic-ineq-not-distinct*:
 assumes *x* = *y* ∨ *x* = *z* ∨ *x* = *t* ∨ *y* = *z* ∨ *y* = *t* ∨ *z* = (*t*::*'a*::*metric-space*)
 shows *dist x y* + *dist z t* ≤ *max* (*dist x z* + *dist y t*) (*dist x t* + *dist y z*)
using *assms* **by** (*auto simp add*: *dist-commute*, *simp add*: *dist-triangle add.commute*, *simp add*: *dist-triangle3*)

It readily follows from the definition that hyperbolicity passes to the closure of the set.

lemma *Gromov-hyperbolic-closure*:
 assumes *Gromov-hyperbolic-subset delta A*
 shows *Gromov-hyperbolic-subset delta* (*closure A*)
unfolding *Gromov-hyperbolic-subset-def* **proof** (*auto*)
 fix *x y z t* **assume** *H*: *x* ∈ *closure A y* ∈ *closure A z* ∈ *closure A t* ∈ *closure A*
 obtain *X*::*nat* ⇒ *'a* **where** *X*: ⋀*n*. *X n* ∈ *A X* ⟶ *x*
  using *H closure-sequential* **by** *blast*
 obtain *Y*::*nat* ⇒ *'a* **where** *Y*: ⋀*n*. *Y n* ∈ *A Y* ⟶ *y*
  using *H closure-sequential* **by** *blast*

obtain Z::*nat* ⇒ *'a* **where** Z: $\bigwedge n.\ Z\ n \in A\ Z \longrightarrow z$
using H *closure-sequential* **by** *blast*
obtain T::*nat* ⇒ *'a* **where** T: $\bigwedge n.\ T\ n \in A\ T \longrightarrow t$
using H *closure-sequential* **by** *blast*
have ∗: *max* (*dist* (X n) (Z n) + *dist* (Y n) (T n)) (*dist* (X n) (T n) + *dist* (Y n) (Z n)) + *2* ∗ *delta* − *dist* (X n) (Y n) − *dist* (Z n) (T n) ≥ *0* **for** n
using *assms* $X(1)[of\ n]$ $Y(1)[of\ n]$ $Z(1)[of\ n]$ $T(1)[of\ n]$ **unfolding** *Gromov-hyperbolic-subset-def*
by (*auto simp add*: *algebra-simps*)
have ∗∗: (λn. *max* (*dist* (X n) (Z n) + *dist* (Y n) (T n)) (*dist* (X n) (T n) + *dist* (Y n) (Z n)) + *2* ∗ *delta* − *dist* (X n) (Y n) − *dist* (Z n) (T n))
⟶ *max* (*dist x z* + *dist y t*) (*dist x t* + *dist y z*) + *2* ∗ *delta* − *dist x y* − *dist z t*
apply (*auto intro*!: *tendsto-intros*) **using** X Y Z T **by** *auto*
have *max* (*dist x z* + *dist y t*) (*dist x t* + *dist y z*) + *2* ∗ *delta* − *dist x y* − *dist z t* ≥ *0*
apply (*rule LIMSEQ-le-const*[*OF* ∗∗]) **using** ∗ **by** *auto*
then show *dist x y* + *dist z t* ≤ *max* (*dist x z* + *dist y t*) (*dist x t* + *dist y z*) + *2* ∗ *delta*
by *auto*
qed

A good formulation of hyperbolicity is in terms of Gromov products. Intuitively, the Gromov product of x and y based at e is the distance between e and the geodesic between x and y. It is also the time after which the geodesics from e to x and from e to y stop travelling together.

definition *Gromov-product-at*::(*'a*::*metric-space*) ⇒ *'a* ⇒ *'a* ⇒ *real*
where *Gromov-product-at e x y* = (*dist e x* + *dist e y* − *dist x y*) / *2*

lemma *Gromov-hyperbolic-subsetI2*:
fixes *delta*::*real*
assumes $\bigwedge e\ x\ y\ z.\ e \in A \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow$ *Gromov-product-at* (*e*::*'a*::*metric-space*) *x z* ≥ *min* (*Gromov-product-at e x y*) (*Gromov-product-at e y z*) − *delta*
shows *Gromov-hyperbolic-subset delta A*
proof (*rule Gromov-hyperbolic-subsetI*)
fix *x y z t* **assume** H: $x \in A$ $z \in A$ $y \in A$ $t \in A$
show *dist x y* + *dist z t* ≤ *max* (*dist x z* + *dist y t*) (*dist x t* + *dist y z*) + *2* ∗ *delta*
using *assms*[*OF H*] **unfolding** *Gromov-product-at-def min-def max-def*
by (*auto simp add*: *divide-simps algebra-simps dist-commute*)
qed

lemma *Gromov-product-nonneg* [*simp*, *mono-intros*]:
Gromov-product-at e x y ≥ *0*
unfolding *Gromov-product-at-def* **by** (*simp add*: *dist-triangle3*)

lemma *Gromov-product-commute*:
Gromov-product-at e x y = *Gromov-product-at e y x*

unfolding *Gromov-product-at-def* **by** (*auto simp add*: *dist-commute*)

lemma *Gromov-product-le-dist* [*simp*, *mono-intros*]:
  Gromov-product-at e x y ≤ *dist e x*
  Gromov-product-at e x y ≤ *dist e y*
unfolding *Gromov-product-at-def* **by** (*auto simp add*: *diff-le-eq dist-triangle dist-triangle2*)

lemma *Gromov-product-le-infdist* [*mono-intros*]:
  assumes *geodesic-segment-between G x y*
  shows *Gromov-product-at e x y* ≤ *infdist e G*
proof −
  have [*simp*]: *G* ≠ {} **using** *assms* **by** *auto*
  have *Gromov-product-at e x y* ≤ *dist e z* **if** *z* ∈ *G* **for** *z*
  proof −
    have *dist e x* + *dist e y* ≤ (*dist e z* + *dist z x*) + (*dist e z* + *dist z y*)
      by (*intro add-mono dist-triangle*)
    also have ... = *2* ∗ *dist e z* + *dist x y*
        apply (*auto simp add*: *dist-commute*) **using** ‹*z* ∈ *G*› *assms* **by** (*metis dist-commute geodesic-segment-dist*)
    finally show *?thesis* **unfolding** *Gromov-product-at-def* **by** *auto*
  qed
  then show *?thesis*
    apply (*subst infdist-notempty*) **by** (*auto intro*: *cINF-greatest*)
qed

lemma *Gromov-product-add*:
  Gromov-product-at e x y + *Gromov-product-at x e y* = *dist e x*
unfolding *Gromov-product-at-def* **by** (*auto simp add*: *algebra-simps divide-simps dist-commute*)

lemma *Gromov-product-geodesic-segment*:
  assumes *geodesic-segment-between G x y t* ∈ {*0..dist x y*}
  shows *Gromov-product-at x y* (*geodesic-segment-param G x t*) = *t*
proof −
  have *dist x* (*geodesic-segment-param G x t*) = *t*
    using *assms*(*1*) *assms*(*2*) *geodesic-segment-param*(*6*) **by** *auto*
  moreover have *dist y* (*geodesic-segment-param G x t*) = *dist x y* − *t*
     by (*metis* ‹*dist x* (*geodesic-segment-param G x t*) = *t*› *add-diff-cancel-left′ assms*(*1*) *assms*(*2*) *dist-commute geodesic-segment-dist geodesic-segment-param*(*3*))
  ultimately show *?thesis* **unfolding** *Gromov-product-at-def* **by** *auto*
qed

lemma *Gromov-product-e-x-x* [*simp*]:
  Gromov-product-at e x x = *dist e x*
unfolding *Gromov-product-at-def* **by** *auto*

lemma *Gromov-product-at-diff*:
  |*Gromov-product-at x y z* − *Gromov-product-at a b c*| ≤ *dist x a* + *dist y b* + *dist z c*

unfolding *Gromov-product-at-def abs-le-iff* **apply** (*auto simp add*: *divide-simps*)
by (*smt dist-commute dist-triangle4*)+

lemma *Gromov-product-at-diff1*:
  $|$*Gromov-product-at a x y* $-$ *Gromov-product-at b x y*$|$ $\leq$ *dist a b*
using *Gromov-product-at-diff*[*of a x y b x y*] **by** *auto*

lemma *Gromov-product-at-diff2*:
  $|$*Gromov-product-at e x z* $-$ *Gromov-product-at e y z*$|$ $\leq$ *dist x y*
using *Gromov-product-at-diff*[*of e x z e y z*] **by** *auto*

lemma *Gromov-product-at-diff3*:
  $|$*Gromov-product-at e x y* $-$ *Gromov-product-at e x z*$|$ $\leq$ *dist y z*
using *Gromov-product-at-diff*[*of e x y e x z*] **by** *auto*

The Gromov product is continuous in its three variables. We formulate it in terms of sequences, as it is the way it will be used below (and moreover continuity for functions of several variables is very poor in the library).

lemma *Gromov-product-at-continuous*:
  assumes ($u \longrightarrow x$) F ($v \longrightarrow y$) F ($w \longrightarrow z$) F
  shows ((λn. *Gromov-product-at* (u n) (v n) (w n)) $\longrightarrow$ *Gromov-product-at x y z*) F
proof $-$
  have ((λn. *abs*(*Gromov-product-at* (u n) (v n) (w n) $-$ *Gromov-product-at x y z*)) $\longrightarrow$ *0* + *0* + *0*) F
    apply (*rule tendsto-sandwich*[*of* λn. *0* - - λn. *dist* (u n) x + *dist* (v n) y + *dist* (w n) z, *OF always-eventually always-eventually*])
    apply (*simp*, *simp add*: *Gromov-product-at-diff*, *simp*, *intro tendsto-intros*)
    using *assms tendsto-dist-iff* **by** *auto*
  then show *?thesis*
    apply (*subst tendsto-dist-iff*) **unfolding** *dist-real-def* **by** *auto*
qed

8.2 Typeclass for Gromov hyperbolic spaces

We could (should?) just derive `Gromov_hyperbolic_space` from `metric_space`. However, in this case, properties of metric spaces are not available when working in the locale! It is more efficient to ensure that we have a metric space by putting a type class restriction in the definition. The δ in Gromov-hyperbolicity type class is called `deltaG` to avoid name clashes.

class *metric-space-with-deltaG* = *metric-space* +
  fixes *deltaG*::('*a*::*metric-space*) *itself* $\Rightarrow$ *real*

class *Gromov-hyperbolic-space* = *metric-space-with-deltaG* +
  assumes *hyperb-quad-ineq0*: *Gromov-hyperbolic-subset* (*deltaG*(*TYPE*('*a*::*metric-space*))) (*UNIV*::'*a set*)

class *Gromov-hyperbolic-space-geodesic* = *Gromov-hyperbolic-space* + *geodesic-space*

lemma (**in** *Gromov-hyperbolic-space*) *hyperb-quad-ineq* [*mono-intros*]:
  shows *dist x y* + *dist z t* ≤ *max* (*dist x z* + *dist y t*) (*dist x t* + *dist y z*) + *2* ∗ *deltaG*(*TYPE*(′*a*))
using *hyperb-quad-ineq0* **unfolding** *Gromov-hyperbolic-subset-def* **by** *auto*

It readily follows from the definition that the completion of a δ-hyperbolic space is still δ-hyperbolic.

instantiation *metric-completion* :: (*Gromov-hyperbolic-space*) *Gromov-hyperbolic-space*
begin
definition *deltaG-metric-completion*::(′*a metric-completion*) *itself* ⇒ *real* **where**
  deltaG-metric-completion - = *deltaG*(*TYPE*(′*a*))

instance proof (*standard*, *rule Gromov-hyperbolic-subsetI*)
 have *Gromov-hyperbolic-subset* (*deltaG*(*TYPE*(′*a*))) (*range* (*to-metric-completion*::′*a* ⇒ -))
    unfolding *Gromov-hyperbolic-subset-def*
    apply (*auto simp add*: *isometry-onD*[*OF to-metric-completion-isometry*])
    by (*metis hyperb-quad-ineq*)
 then have *Gromov-hyperbolic-subset* (*deltaG TYPE*(′*a metric-completion*)) (*UNIV*::′*a metric-completion set*)
   unfolding *deltaG-metric-completion-def to-metric-completion-dense*′[*symmetric*]
    using *Gromov-hyperbolic-closure* **by** *auto*
  then show *dist x y* + *dist z t* ≤ *max* (*dist x z* + *dist y t*) (*dist x t* + *dist y z*) + *2* ∗ *deltaG TYPE*(′*a metric-completion*)
      for *x y z t*::′*a metric-completion*
    unfolding *Gromov-hyperbolic-subset-def* **by** *auto*
qed
end

context *Gromov-hyperbolic-space*
begin

lemma *delta-nonneg* [*simp*, *mono-intros*]:
  deltaG(*TYPE*(′*a*)) ≥ *0*
proof −
  obtain *x*::′*a* **where** *True* **by** *auto*
  show *?thesis* **using** *hyperb-quad-ineq*[*of x x x x*] **by** *auto*
qed

lemma *hyperb-ineq* [*mono-intros*]:
 Gromov-product-at (*e*::′*a*) *x z* ≥ *min* (*Gromov-product-at e x y*) (*Gromov-product-at e y z*) − *deltaG*(*TYPE*(′*a*))
using *hyperb-quad-ineq*[*of e y x z*] **unfolding** *Gromov-product-at-def min-def max-def*
by (*auto simp add*: *divide-simps algebra-simps metric-space-class.dist-commute*)

lemma *hyperb-ineq*′ [*mono-intros*]:
  Gromov-product-at (*e*::′*a*) *x z* + *deltaG*(*TYPE*(′*a*)) ≥ *min* (*Gromov-product-at e*

x y) (*Gromov-product-at e y z*)
using *hyperb-ineq*[*of e x y z*] **by** *auto*

lemma *hyperb-ineq-4-points* [*mono-intros*]:
  Min {*Gromov-product-at* (*e*::*'a*) *x y*, *Gromov-product-at e y z*, *Gromov-product-at e z t*} − *2* ∗ *deltaG*(*TYPE*(*'a*)) ≤ *Gromov-product-at e x t*
using *hyperb-ineq*[*of e x y z*] *hyperb-ineq*[*of e x z t*] **apply** *auto* **using** *delta-nonneg*
by *linarith*

lemma *hyperb-ineq-4-points'* [*mono-intros*]:
  Min {*Gromov-product-at* (*e*::*'a*) *x y*, *Gromov-product-at e y z*, *Gromov-product-at e z t*} ≤ *Gromov-product-at e x t* + *2* ∗ *deltaG*(*TYPE*(*'a*))
using *hyperb-ineq-4-points*[*of e x y z t*] **by** *auto*

In Gromov-hyperbolic spaces, geodesic triangles are thin, i.e., a point on one side of a geodesic triangle is close to the union of the two other sides (where the constant in "close" is 4δ, independent of the size of the triangle). We prove this basic property (which, in fact, is a characterization of Gromov-hyperbolic spaces: a geodesic space in which triangles are thin is hyperbolic).

lemma *thin-triangles1*:
  assumes *geodesic-segment-between G x y geodesic-segment-between H x* (*z*::*'a*)
        t ∈ {*0*..*Gromov-product-at x y z*}
  shows *dist* (*geodesic-segment-param G x t*) (*geodesic-segment-param H x t*) ≤ *4* ∗ *deltaG*(*TYPE*(*'a*))
proof −
  have ∗: *Gromov-product-at x z* (*geodesic-segment-param H x t*) = *t*
    apply (*rule Gromov-product-geodesic-segment*[*OF assms*(*2*)]) **using** *assms*(*3*) *Gromov-product-le-dist*(*2*)
    by (*metis atLeastatMost-subset-iff subset-iff*)
  have *Gromov-product-at x y* (*geodesic-segment-param H x t*)
      ≥ *min* (*Gromov-product-at x y z*) (*Gromov-product-at x z* (*geodesic-segment-param H x t*)) − *deltaG*(*TYPE*(*'a*))
    by (*rule hyperb-ineq*)
  then have *I*: *Gromov-product-at x y* (*geodesic-segment-param H x t*) ≥ *t* − *deltaG*(*TYPE*(*'a*))
    using *assms*(*3*) **unfolding** ∗ **by** *auto*

  have ∗: *Gromov-product-at x* (*geodesic-segment-param G x t*) *y* = *t*
    apply (*subst Gromov-product-commute*)
    apply (*rule Gromov-product-geodesic-segment*[*OF assms*(*1*)]) **using** *assms*(*3*) *Gromov-product-le-dist*(*1*)
    by (*metis atLeastatMost-subset-iff subset-iff*)
  have *t* − *2* ∗ *deltaG*(*TYPE*(*'a*)) = *min t* (*t*− *deltaG*(*TYPE*(*'a*))) − *deltaG*(*TYPE*(*'a*))
    unfolding *min-def* **using** *antisym* **by** *fastforce*
  also have ... ≤ *min* (*Gromov-product-at x* (*geodesic-segment-param G x t*) *y*) (*Gromov-product-at x y* (*geodesic-segment-param H x t*)) − *deltaG*(*TYPE*(*'a*))
    using *I* ∗ **by** (*simp add*: *algebra-simps*)
  also have ... ≤ *Gromov-product-at x* (*geodesic-segment-param G x t*) (*geodesic-segment-param H x t*)

```
    by (rule hyperb-ineq)
  finally have I: Gromov-product-at x (geodesic-segment-param G x t) (geodesic-segment-param
H x t) ≥ t − 2 ∗ deltaG(TYPE('a))
    by simp

  have A: dist x (geodesic-segment-param G x t) = t
   by (meson assms(1) assms(3) atLeastatMost-subset-iff geodesic-segment-param(6)
Gromov-product-le-dist(1) subset-eq)
  have B: dist x (geodesic-segment-param H x t) = t
   by (meson assms(2) assms(3) atLeastatMost-subset-iff geodesic-segment-param(6)
Gromov-product-le-dist(2) subset-eq)
  show ?thesis
    using I unfolding Gromov-product-at-def A B by auto
qed

theorem thin-triangles:
  assumes geodesic-segment-between Gxy x y
          geodesic-segment-between Gxz x z
          geodesic-segment-between Gyz y z
          (w::'a) ∈ Gyz
  shows infdist w (Gxy ∪ Gxz) ≤ 4 ∗ deltaG(TYPE('a))
proof −
  obtain t where w: t ∈ {0..dist y z} w = geodesic-segment-param Gyz y t
    using geodesic-segment-param[OF assms(3)] assms(4) by (metis imageE)
  show ?thesis
  proof (cases t ≤ Gromov-product-at y x z)
    case True
     have ∗: dist w (geodesic-segment-param Gxy y t) ≤ 4 ∗ deltaG(TYPE('a))
unfolding w(2)
      apply (rule thin-triangles1[of - - z - x])
     using True assms(1) assms(3) w(1) by (auto simp add: geodesic-segment-commute
Gromov-product-commute)
    show ?thesis
      apply (rule infdist-le2[OF - ∗])
     by (metis True assms(1) box-real(2) geodesic-segment-commute geodesic-segment-param(3)
Gromov-product-le-dist(1) mem-box-real(2) order-trans subset-eq sup.cobounded1
w(1))
  next
    case False
    define s where s = dist y z − t
    have s: s ∈ {0..Gromov-product-at z y x}
       unfolding s-def using Gromov-product-add[of y z x] w(1) False by (auto
simp add: Gromov-product-commute)
    have w2: w = geodesic-segment-param Gyz z s
     unfolding s-def w(2) apply (rule geodesic-segment-reverse-param[symmetric])
using assms(3) w(1) by auto
     have ∗: dist w (geodesic-segment-param Gxz z s) ≤ 4 ∗ deltaG(TYPE('a))
unfolding w2
      apply (rule thin-triangles1[of - - y - x])
```

using *s assms* **by** (*auto simp add*: *geodesic-segment-commute*)
show *?thesis*
apply (*rule infdist-le2*[*OF* - ∗])
by (*metis Un-iff assms*(*2*) *atLeastAtMost-iff geodesic-segment-commute geodesic-segment-param*(*3*) *Gromov-product-commute Gromov-product-le-dist*(*1*) *order-trans s*)
qed
qed

A consequence of the thin triangles property is that, although the geodesic between two points is in general not unique in a Gromov-hyperbolic space, two such geodesics are within $O(\delta)$ of each other.

lemma *geodesics-nearby*:
assumes *geodesic-segment-between G x y geodesic-segment-between H x y*
(*z*::′*a*) ∈ *G*
shows *infdist z H* ≤ *4* ∗ *deltaG*(*TYPE*(′*a*))
using *thin-triangles*[*OF geodesic-segment-between-x-x*(*1*) *assms*(*2*) *assms*(*1*) *assms*(*3*)]
geodesic-segment-endpoints(*1*)[*OF assms*(*2*)] *insert-absorb* **by** *fastforce*

A small variant of the property of thin triangles is that triangles are slim, i.e., there is a point which is close to the three sides of the triangle (a "center" of the triangle, but only defined up to $O(\delta)$). And one can take it on any side, and its distance to the corresponding vertices is expressed in terms of a Gromov product.

lemma *slim-triangle*:
assumes *geodesic-segment-between Gxy x y*
geodesic-segment-between Gxz x z
geodesic-segment-between Gyz y (*z*::′*a*)
shows ∃ *w*. *infdist w Gxy* ≤ *4* ∗ *deltaG*(*TYPE*(′*a*)) ∧
infdist w Gxz ≤ *4* ∗ *deltaG*(*TYPE*(′*a*)) ∧
infdist w Gyz ≤ *4* ∗ *deltaG*(*TYPE*(′*a*)) ∧
dist w x = (*Gromov-product-at x y z*) ∧ *w* ∈ *Gxy*
proof −
define *w* **where** *w* = *geodesic-segment-param Gxy x* (*Gromov-product-at x y z*)
have *w* ∈ *Gxy* **unfolding** *w-def*
by (*rule geodesic-segment-param*(*3*)[*OF assms*(*1*)], *auto*)
then have *xy*: *infdist w Gxy* ≤ *4* ∗ *deltaG*(*TYPE*(′*a*)) **by** *simp*
have ∗: *dist w x* = (*Gromov-product-at x y z*)
unfolding *w-def* **using** *assms*(*1*)
by (*metis Gromov-product-le-dist*(*1*) *Gromov-product-nonneg atLeastAtMost-iff geodesic-segment-param*(*6*) *metric-space-class.dist-commute*)

define *w2* **where** *w2* = *geodesic-segment-param Gxz x* (*Gromov-product-at x y z*)
have *w2* ∈ *Gxz* **unfolding** *w2-def*
by (*rule geodesic-segment-param*(*3*)[*OF assms*(*2*)], *auto*)
moreover have *dist w w2* ≤ *4* ∗ *deltaG*(*TYPE*(′*a*))
unfolding *w-def w2-def* **by** (*rule thin-triangles1*[*OF assms*(*1*) *assms*(*2*)], *auto*)
ultimately have *xz*: *infdist w Gxz* ≤ *4* ∗ *deltaG*(*TYPE*(′*a*))

```
    using infdist-le2 by blast

  have w = geodesic-segment-param Gxy y (dist x y − Gromov-product-at x y z)
    unfolding w-def by (rule geodesic-segment-reverse-param[OF assms(1), symmetric], auto)
  then have w: w = geodesic-segment-param Gxy y (Gromov-product-at y x z)
    using Gromov-product-add[of x y z] by (metis add-diff-cancel-left′)

  define w3 where w3 = geodesic-segment-param Gyz y (Gromov-product-at y x z)
  have w3 ∈ Gyz unfolding w3-def
    by (rule geodesic-segment-param(3)[OF assms(3)], auto)
  moreover have dist w w3 ≤ 4 ∗ deltaG(TYPE(′a))
    unfolding w w3-def by (rule thin-triangles1[OF geodesic-segment-commute[OF assms(1)] assms(3)], auto)
  ultimately have yz: infdist w Gyz ≤ 4 ∗ deltaG(TYPE(′a))
    using infdist-le2 by blast

  show ?thesis using xy xz yz ∗ ‹w ∈ Gxy› by force
qed
```

The distance of a vertex of a triangle to the opposite side is essentially given by the Gromov product, up to 2δ.

```
lemma dist-triangle-side-middle:
  assumes geodesic-segment-between G x (y::′a)
  shows dist z (geodesic-segment-param G x (Gromov-product-at x z y)) ≤ Gromov-product-at z x y + 2 ∗ deltaG(TYPE(′a))
proof −
  define m where m = geodesic-segment-param G x (Gromov-product-at x z y)
  have m ∈ G
    unfolding m-def using assms(1) by auto
  have A: dist x m = Gromov-product-at x z y
    unfolding m-def by (rule geodesic-segment-param(6)[OF assms(1)], auto)
  have B: dist y m = dist x y − dist x m
    using geodesic-segment-dist[OF assms ‹m ∈ G›] by (auto simp add: metric-space-class.dist-commute)
  have ∗: dist x z + dist y m = Gromov-product-at z x y + dist x y
          dist x m + dist y z = Gromov-product-at z x y + dist x y
    unfolding B A Gromov-product-at-def by (auto simp add: metric-space-class.dist-commute divide-simps)

  have dist x y + dist z m ≤ max (dist x z + dist y m) (dist x m + dist y z) + 2 ∗ deltaG(TYPE(′a))
    by (rule hyperb-quad-ineq)
  then have dist z m ≤ Gromov-product-at z x y + 2 ∗ deltaG(TYPE(′a))
    unfolding ∗ by auto
  then show ?thesis
    unfolding m-def by auto
qed
```

lemma *infdist-triangle-side* [*mono-intros*]:
assumes *geodesic-segment-between G x* (*y*::′*a*)
shows *infdist z G* ≤ *Gromov-product-at z x y* + *2* ∗ *deltaG*(*TYPE*(′*a*))
proof −
have *infdist z G* ≤ *dist z* (*geodesic-segment-param G x* (*Gromov-product-at x z y*))
using *assms* **by** (*auto intro*!: *infdist-le*)
then show *?thesis*
using *dist-triangle-side-middle*[*OF assms*, *of z*] **by** *auto*
qed

The distance of a point on a side of triangle to the opposite vertex is controlled by the length of the opposite sides, up to δ.

lemma *dist-le-max-dist-triangle*:
assumes *geodesic-segment-between G x y*
m ∈ *G*
shows *dist m z* ≤ *max* (*dist x z*) (*dist y z*) + *deltaG*(*TYPE*(′*a*))
proof −
consider *dist m x* ≤ *deltaG*(*TYPE*(′*a*)) | *dist m y* ≤ *deltaG*(*TYPE*(′*a*)) |
dist m x ≥ *deltaG*(*TYPE*(′*a*)) ∧ *dist m y* ≥ *deltaG*(*TYPE*(′*a*)) ∧ *Gromov-product-at z x m* ≤ *Gromov-product-at z m y* |
dist m x ≥ *deltaG*(*TYPE*(′*a*)) ∧ *dist m y* ≥ *deltaG*(*TYPE*(′*a*)) ∧ *Gromov-product-at z m y* ≤ *Gromov-product-at z x m*
by *linarith*
then show *?thesis*
proof (*cases*)
case *1*
have *dist m z* ≤ *dist m x* + *dist x z*
by (*intro mono-intros*)
then show *?thesis* **using** *1* **by** *auto*
next
case *2*
have *dist m z* ≤ *dist m y* + *dist y z*
by (*intro mono-intros*)
then show *?thesis* **using** *2* **by** *auto*
next
case *3*
then have *Gromov-product-at z x m* = *min* (*Gromov-product-at z x m*) (*Gromov-product-at z m y*)
by *auto*
also have ... ≤ *Gromov-product-at z x y* + *deltaG*(*TYPE*(′*a*))
by (*intro mono-intros*)
finally have *dist z m* ≤ *dist z y* + *dist x m* − *dist x y* + *2* ∗ *deltaG*(*TYPE*(′*a*))
unfolding *Gromov-product-at-def* **by** (*auto simp add*: *divide-simps algebra-simps*)
also have ... = *dist z y* − *dist m y* + *2* ∗ *deltaG*(*TYPE*(′*a*))
using *geodesic-segment-dist*[*OF assms*] **by** *auto*
also have ... ≤ *dist z y* + *deltaG*(*TYPE*(′*a*))

```
      using 3 by auto
    finally show ?thesis
      by (simp add: metric-space-class.dist-commute)
  next
    case 4
   then have Gromov-product-at z m y = min (Gromov-product-at z x m) (Gromov-product-at
z m y)
      by auto
    also have ... ≤ Gromov-product-at z x y + deltaG(TYPE('a))
      by (intro mono-intros)
   finally have dist z m ≤ dist z x + dist m y − dist x y + 2 ∗ deltaG(TYPE('a))
        unfolding Gromov-product-at-def by (auto simp add: divide-simps alge-
bra-simps)
    also have ... = dist z x − dist x m + 2 ∗ deltaG(TYPE('a))
      using geodesic-segment-dist[OF assms] by auto
    also have ... ≤ dist z x + deltaG(TYPE('a))
      using 4 by (simp add: metric-space-class.dist-commute)
    finally show ?thesis
      by (simp add: metric-space-class.dist-commute)
  qed
qed

end
```

A useful variation around the previous properties is that quadrilaterals are thin, in the following sense: if one has a union of three geodesics from x to t, then a geodesic from x to t remains within distance 8δ of the union of these 3 geodesics. We formulate the statement in geodesic hyperbolic spaces as the proof requires the construction of an additional geodesic, but in fact the statement is true without this assumption, thanks to the Bonk-Schramm extension theorem.

```
lemma (in Gromov-hyperbolic-space-geodesic) thin-quadrilaterals:
  assumes geodesic-segment-between Gxy x y
          geodesic-segment-between Gyz y z
          geodesic-segment-between Gzt z t
          geodesic-segment-between Gxt x t
          (w::'a) ∈ Gxt
  shows infdist w (Gxy ∪ Gyz ∪ Gzt) ≤ 8 ∗ deltaG(TYPE('a))
proof −
  have I: infdist w ({x−−z} ∪ Gzt) ≤ 4 ∗ deltaG(TYPE('a))
    apply (rule thin-triangles[OF - assms(3) assms(4) assms(5)])
    by (simp add: geodesic-segment-commute)
  have ∃ u ∈ {x−−z} ∪ Gzt. infdist w ({x−−z} ∪ Gzt) = dist w u
   apply (rule infdist-proper-attained, auto intro!: proper-Un simp add: geodesic-segment-topology(7))
    by (meson assms(3) geodesic-segmentI geodesic-segment-topology)
  then obtain u where u: u ∈ {x−−z} ∪ Gzt infdist w ({x−−z} ∪ Gzt) = dist
w u
    by auto
```

have *infdist u* $(Gxy \cup Gyz \cup Gzt) \leq 4 * deltaG(TYPE('a))$
proof (*cases* $u \in \{x{-}{-}z\}$)
case *True*
have *infdist u* $(Gxy \cup Gyz \cup Gzt) \leq$ *infdist u* $(Gxy \cup Gyz)$
apply (*intro mono-intros*) **using** *assms*(*1*) **by** *auto*
also have ... $\leq 4 * deltaG(TYPE('a))$
using *thin-triangles*[*OF geodesic-segment-commute*[*OF assms*(*1*)] *assms*(*2*) - *True*] **by** *auto*
finally show *?thesis*
by *auto*
next
case *False*
then have $*$: $u \in Gzt$ **using** *u*(*1*) **by** *auto*
have *infdist u* $(Gxy \cup Gyz \cup Gzt) \leq$ *infdist u Gzt*
apply (*intro mono-intros*) **using** *assms*(*3*) **by** *auto*
also have ... $= 0$ **using** $*$ **by** *auto*
finally show *?thesis*
using *local.delta-nonneg* **by** *linarith*
qed
moreover have *infdist w* $(Gxy \cup Gyz \cup Gzt) \leq$ *infdist u* $(Gxy \cup Gyz \cup Gzt)$ + *dist w u*
by (*intro mono-intros*)
ultimately show *?thesis*
using *I u*(*2*) **by** *auto*
qed

There are converses to the above statements: if triangles are thin, or slim, then the space is Gromov-hyperbolic, for some δ. We prove these criteria here, following the proofs in Ghys (with a simplification in the case of slim triangles.

The basic result we will use twice below is the following: if points on sides of triangles at the same distance of the basepoint are close to each other up to the Gromov product, then the space is hyperbolic. The proof goes as follows. One wants to show that $(x, z)_e \geq \min((x, y)_e, (y, z)_e) - \delta = t - \delta$. On $[ex]$, $[ey]$ and $[ez]$, consider points wx, wy and wz at distance t of e. Then wx and wy are δ-close by assumption, and so are wy and wz. Then wx and wz are 2δ-close. One can use these two points to express $(x, z)_e$, and the result follows readily.

lemma (**in** *geodesic-space*) *controlled-thin-triangles-implies-hyperbolic*:
assumes $\bigwedge(x::'a)$ *y z t Gxy Gxz. geodesic-segment-between Gxy x y* $\Longrightarrow$ *geodesic-segment-between Gxz x z* $\Longrightarrow t \in \{0..$*Gromov-product-at x y z*$\}$
$\Longrightarrow$ *dist* (*geodesic-segment-param Gxy x t*) (*geodesic-segment-param Gxz x t*) $\leq$ *delta*
shows *Gromov-hyperbolic-subset delta* (*UNIV*::*'a set*)
proof (*rule Gromov-hyperbolic-subsetI2*)
fix *e x y z*::*'a*
define *t* **where** $t = min$ (*Gromov-product-at e x y*) (*Gromov-product-at e y z*)

define *wx* **where** *wx* = *geodesic-segment-param* {*e*−−*x*} *e t*
define *wy* **where** *wy* = *geodesic-segment-param* {*e*−−*y*} *e t*
define *wz* **where** *wz* = *geodesic-segment-param* {*e*−−*z*} *e t*
have *dist wx wy* ≤ *delta*
unfolding *wx-def wy-def t-def* **by** (*rule assms*[*of - - x - y*], *auto*)
have *dist wy wz* ≤ *delta*
unfolding *wy-def wz-def t-def* **by** (*rule assms*[*of - - y - z*], *auto*)

have *t* + *dist wy x* = *dist e wx* + *dist wy x*
unfolding *wx-def* **apply** (*auto intro*!: *geodesic-segment-param-in-geodesic-spaces*(*6*)[*symmetric*])
unfolding *t-def* **by** (*auto*, *meson Gromov-product-le-dist*(*1*) *min.absorb-iff2 min.left-idem order.trans*)
also have ... ≤ *dist e wx* + (*dist wy wx* + *dist wx x*)
by (*intro mono-intros*)
also have ... ≤ *dist e wx* + (*delta* + *dist wx x*)
using ‹*dist wx wy* ≤ *delta*› **by** (*auto simp add*: *metric-space-class.dist-commute*)
also have ... = *delta* + *dist e x*
apply *auto* **apply** (*rule geodesic-segment-dist*[*of* {*e*−−*x*}])
unfolding *wx-def t-def* **by** (*auto simp add*: *geodesic-segment-param-in-segment*)
finally have ∗: *t* + *dist wy x* − *delta* ≤ *dist e x* **by** *simp*

have *t* + *dist wy z* = *dist e wz* + *dist wy z*
unfolding *wz-def* **apply** (*auto intro*!: *geodesic-segment-param-in-geodesic-spaces*(*6*)[*symmetric*])
unfolding *t-def* **by** (*auto*, *meson Gromov-product-le-dist*(*2*) *min.absorb-iff1 min.right-idem order.trans*)
also have ... ≤ *dist e wz* + (*dist wy wz* + *dist wz z*)
by (*intro mono-intros*)
also have ... ≤ *dist e wz* + (*delta* + *dist wz z*)
using ‹*dist wy wz* ≤ *delta*› **by** (*auto simp add*: *metric-space-class.dist-commute*)
also have ... = *delta* + *dist e z*
apply *auto* **apply** (*rule geodesic-segment-dist*[*of* {*e*−−*z*}])
unfolding *wz-def t-def* **by** (*auto simp add*: *geodesic-segment-param-in-segment*)
finally have *t* + *dist wy z* − *delta* ≤ *dist e z* **by** *simp*

then have (*t* + *dist wy x* − *delta*) + (*t* + *dist wy z* − *delta*) ≤ *dist e x* + *dist e z*
using ∗ **by** *simp*
also have ... = *dist x z* + *2* ∗ *Gromov-product-at e x z*
unfolding *Gromov-product-at-def* **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... ≤ *dist wy x* + *dist wy z* + *2* ∗ *Gromov-product-at e x z*
using *metric-space-class.dist-triangle*[*of x z wy*] **by** (*auto simp add*: *metric-space-class.dist-commute*)
finally have *2* ∗ *t* − *2* ∗ *delta* ≤ *2* ∗ *Gromov-product-at e x z*
by *auto*
then show *min* (*Gromov-product-at e x y*) (*Gromov-product-at e y z*) − *delta* ≤ *Gromov-product-at e x z*
unfolding *t-def* **by** *auto*
qed

We prove that if triangles are thin, i.e., they satisfy the Rips condition, i.e., every side of a triangle is included in the δ-neighborhood of the union of the other triangles, then the space is hyperbolic. If a point w on $[xy]$ satisfies $d(x,w) < (y,z)_x - \delta$, then its friend on $[xz] \cup [yz]$ has to be on $[xz]$, and roughly at the same distance of the origin. Then it follows that the point on $[xz]$ with $d(x,w') = d(x,w)$ is close to w, as desired. If $d(x,w) \in [(y,z)_x - \delta, (y,z)_x)$, we argue in the same way but for the point which is closer to x by an amount δ. Finally, the last case $d(x,w) = (y,z)_x$ follows by continuity.

proposition (**in** *geodesic-space*) *thin-triangles-implies-hyperbolic*:
  assumes $\bigwedge$(*x*::*'a*) *y z w Gxy Gyz Gxz*. *geodesic-segment-between Gxy x y* ⟹ *geodesic-segment-between Gxz x z* ⟹ *geodesic-segment-between Gyz y z*
    ⟹ *w* ∈ *Gxy* ⟹ *infdist w* (*Gxz* ∪ *Gyz*) ≤ *delta*
  shows *Gromov-hyperbolic-subset* (*4* ∗ *delta*) (*UNIV*::*'a set*)
proof −
  obtain *x0*::*'a* **where** *True* **by** *auto*
  have *infdist x0* ({*x0*} ∪ {*x0*}) ≤ *delta*
    by (*rule assms*[*of* {*x0*} *x0 x0* {*x0*} *x0* {*x0*} *x0*], *auto*)
  then have [*simp*]: *delta* ≥ *0*
    using *infdist-nonneg* **by** *auto*

  have *dist* (*geodesic-segment-param Gxy x t*) (*geodesic-segment-param Gxz x t*) ≤ *4* ∗ *delta*
    if *H*: *geodesic-segment-between Gxy x y geodesic-segment-between Gxz x z t* ∈ {*0*..*Gromov-product-at x y z*}
    for *x y z t Gxy Gxz*
  proof −
    have *Main*: *dist* (*geodesic-segment-param Gxy x u*) (*geodesic-segment-param Gxz x u*) ≤ *4* ∗ *delta*
      if *u* ∈ {*delta*..<*Gromov-product-at x y z*} **for** *u*
    proof −
      define *wy* **where** *wy* = *geodesic-segment-param Gxy x* (*u*−*delta*)
      have *dist wy* (*geodesic-segment-param Gxy x u*) = *abs*((*u*−*delta*) − *u*)
        unfolding *wy-def* **apply** (*rule geodesic-segment-param*(*7*)[*OF H*(*1*)]) **using** *that* **apply** *auto*
        using *Gromov-product-le-dist*(*1*)[*of x y z*] ‹*delta* ≥ *0*› **by** *linarith*+
      then have *I1*: *dist wy* (*geodesic-segment-param Gxy x u*) = *delta* **by** *auto*

      have *infdist wy* (*Gxz* ∪ {*y*−−*z*}) ≤ *delta*
        unfolding *wy-def* **apply** (*rule assms*[*of Gxy x y - z*]) **using** *H* **by** (*auto simp add*: *geodesic-segment-param-in-segment*)
      moreover have ∃ *wz* ∈ *Gxz* ∪ {*y*−−*z*}. *infdist wy* (*Gxz* ∪ {*y*−−*z*}) = *dist wy wz*
        apply (*rule infdist-proper-attained*, *intro proper-Un*)
        using *H*(*2*) **by** (*auto simp add*: *geodesic-segment-topology*)
      ultimately obtain *wz* **where** *wz*: *wz* ∈ *Gxz* ∪ {*y*−−*z*} *dist wy wz* ≤ *delta*
        by *force*

      have *dist wz x* ≤ *dist wz wy* + *dist wy x*

by (*rule metric-space-class.dist-triangle*)
also have ... $\leq$ *delta* + (*u*−*delta*)
apply (*intro add-mono*) **using** *wz*(*2*) **unfolding** *wy-def* **apply** (*auto simp add*: *metric-space-class.dist-commute*)
apply (*intro eq-refl geodesic-segment-param*(*6*)[*OF H*(*1*)])
using *that* **apply** *auto*
by (*metis diff-0-right diff-mono dual-order.trans Gromov-product-le-dist*(*1*) *less-eq-real-def metric-space-class.dist-commute metric-space-class.zero-le-dist wy-def*)
finally have *dist wz x* $\leq$ *u* **by** *auto*
also have ... < *Gromov-product-at x y z*
using *that* **by** *auto*
also have ... $\leq$ *infdist x* {*y*−−*z*}
by (*rule Gromov-product-le-infdist*, *auto*)
finally have *dist x wz* < *infdist x* {*y*−−*z*}
by (*simp add*: *metric-space-class.dist-commute*)
then have *wz* $\notin$ {*y*−−*z*}
by (*metis add.left-neutral infdist-triangle infdist-zero leD*)
then have *wz* $\in$ *Gxz*
using *wz* **by** *auto*

have *u* − *delta* = *dist x wy*
unfolding *wy-def* **apply** (*rule geodesic-segment-param*(*6*)[*symmetric*, *OF H*(*1*)])
using *that* **apply** *auto*
using *Gromov-product-le-dist*(*1*)[*of x y z*] ‹*delta* $\geq$ *0*› **by** *linarith*
also have ... $\leq$ *dist x wz* + *dist wz wy*
by (*rule metric-space-class.dist-triangle*)
also have ... $\leq$ *dist x wz* + *delta*
using *wz*(*2*) **by** (*simp add*: *metric-space-class.dist-commute*)
finally have *dist x wz* $\geq$ *u* − *2* ∗ *delta* **by** *auto*

define *dz* **where** *dz* = *dist x wz*
have ∗: *wz* = *geodesic-segment-param Gxz x dz*
unfolding *dz-def* **using** ‹*wz* $\in$ *Gxz*› *H*(*2*) **by** *auto*
have *dist wz* (*geodesic-segment-param Gxz x u*) = *abs*(*dz* − *u*)
unfolding ∗ **apply** (*rule geodesic-segment-param*(*7*)[*OF H*(*2*)])
unfolding *dz-def* **using** ‹*dist wz x* $\leq$ *u*› *that* **apply** (*auto simp add*: *metric-space-class.dist-commute*)
using *Gromov-product-le-dist*(*2*)[*of x y z*] ‹*delta* $\geq$ *0*› **by** *linarith*+
also have ... $\leq$ *2* ∗ *delta*
unfolding *dz-def* **using** ‹*dist wz x* $\leq$ *u*› ‹*dist x wz* $\geq$ *u* − *2* ∗ *delta*›
by (*auto simp add*: *metric-space-class.dist-commute*)
finally have *I3*: *dist wz* (*geodesic-segment-param Gxz x u*) $\leq$ *2* ∗ *delta*
by *simp*

have *dist* (*geodesic-segment-param Gxy x u*) (*geodesic-segment-param Gxz x u*)
$\leq$ *dist* (*geodesic-segment-param Gxy x u*) *wy* + *dist wy wz* + *dist wz* (*geodesic-segment-param Gxz x u*)

by (*rule dist-triangle4*)
also have ... ≤ *delta* + *delta* + (*2* ∗ *delta*)
using *I1 wz*(*2*) *I3* **by** (*auto simp add*: *metric-space-class.dist-commute*)
finally show *?thesis* **by** *simp*
qed
have *t* ∈ {*0..dist x y*} *t* ∈ {*0..dist x z*} *t* ≥ *0*
using ‹*t* ∈ {*0..Gromov-product-at x y z*}› **apply** *auto*
using *Gromov-product-le-dist*[*of x y z*] **by** *linarith*+
consider *t* ≤ *delta* | *t* ∈ {*delta..*<*Gromov-product-at x y z*} | *t* = *Gromov-product-at x y z* ∧ *t* > *delta*
using ‹*t* ∈ {*0..Gromov-product-at x y z*}› **by** (*auto*, *linarith*)
then show *?thesis*
proof (*cases*)
case *1*
have *dist* (*geodesic-segment-param Gxy x t*) (*geodesic-segment-param Gxz x t*) ≤ *dist x* (*geodesic-segment-param Gxy x t*) + *dist x* (*geodesic-segment-param Gxz x t*)
by (*rule metric-space-class.dist-triangle3*)
also have ... = *t* + *t*
using *geodesic-segment-param*(*6*)[*OF H*(*1*) ‹*t* ∈ {*0..dist x y*}›] *geodesic-segment-param*(*6*)[*OF H*(*2*) ‹*t* ∈ {*0..dist x z*}›]
by *auto*
also have ... ≤ *4* ∗ *delta* **using** *1* ‹*delta* ≥ *0*› **by** *linarith*
finally show *?thesis* **by** *simp*
next
case *2*
show *?thesis* **using** *Main*[*OF 2*] **by** *simp*
next
case *3*

In this case, we argue by approximating t by a slightly smaller parameter, for which the result has already been proved above. We need to argue that all functions are continuous on the sets we are considering, which is straightforward but tedious.

define *u*::*nat* ⇒ *real* **where** *u* = (*λn. t−1/n*)
have *u* ⟶ *t* − *0*
unfolding *u-def* **by** (*intro tendsto-intros*)
then have *u* ⟶ *t* **by** *simp*
then have ∗: *eventually* (*λn. u n* > *delta*) *sequentially*
using *3* **by** (*auto simp add*: *order-tendsto-iff*)
have ∗∗: *eventually* (*λn. u n* ≥ *0*) *sequentially*
apply (*rule eventually-elim2*[*OF* ∗, *of* (*λn. delta* ≥ *0*)]) **apply** *auto*
using ‹*delta* ≥ *0*› **by** *linarith*
have ∗∗∗: *u n* ≤ *t* **for** *n* **unfolding** *u-def* **by** *auto*
have *A*: *eventually* (*λn. u n* ∈ {*delta..*<*Gromov-product-at x y z*}) *sequentially*
apply (*auto intro*!: *eventually-conj*)
apply (*rule eventually-mono*[*OF* ∗], *simp*)
unfolding *u-def* **using** *3* **by** *auto*
have *B*: *eventually* (*λn. dist* (*geodesic-segment-param Gxy x* (*u n*)) (*geodesic-segment-param*

Gxz x (u n)) ≤ 4 ∗ delta) sequentially
by (*rule eventually-mono*[*OF A Main*], *simp*)
have *C*: (*λn. dist* (*geodesic-segment-param Gxy x* (*u n*)) (*geodesic-segment-param Gxz x* (*u n*)))
⟶ *dist* (*geodesic-segment-param Gxy x t*) (*geodesic-segment-param Gxz x t*)
apply (*intro tendsto-intros*)
apply (*rule continuous-on-tendsto-compose*[*OF - ‹u* ⟶ *t› ‹t* ∈ {*0..dist x y*}›])
apply (*simp add*: *isometry-on-continuous H*(*1*))
using ∗∗ ∗∗∗ *‹t* ∈ {*0..dist x y*}› **apply** (*simp*, *intro eventually-conj*, *simp*, *meson dual-order.trans eventually-mono*)
apply (*rule continuous-on-tendsto-compose*[*OF - ‹u* ⟶ *t› ‹t* ∈ {*0..dist x z*}›])
apply (*simp add*: *isometry-on-continuous H*(*2*))
using ∗∗ ∗∗∗ *‹t* ∈ {*0..dist x z*}› **apply** (*simp*, *intro eventually-conj*, *simp*, *meson dual-order.trans eventually-mono*)
done
show *?thesis*
using *B* **unfolding** *eventually-sequentially* **using** *LIMSEQ-le-const2*[*OF C*] **by** *simp*
qed
qed
with *controlled-thin-triangles-implies-hyperbolic*[*OF this*]
show *?thesis* **by** *auto*
qed

Then, we prove that if triangles are slim (i.e., there is a point that is δ-close to all sides), then the space is hyperbolic. Using the previous statement, we should show that points on $[xy]$ and $[xz]$ at the same distance t of the origin are close, if $t \leq (y,z)_x$. There are two steps: - for $t = (y,z)_x$, then the two points are in fact close to the middle of the triangle (as this point satisfies $d(x,y) = d(x,w) + d(w,y) + O(\delta)$, and similarly for the other sides, one gets readily $d(x,w) = (y,z)_w + O(\delta)$ by expanding the formula for the Gromov product). Hence, they are close together. - For $t < (y,z)_x$, we argue that there are points $y' \in [xy]$ and $z' \in [xz]$ for which $t = (y',z')_x$, by a continuity argument and the intermediate value theorem. Then the result follows from the first step in the triangle $xy'z'$.

The proof we give is simpler than the one in [GdlH90], and gives better constants.

proposition (**in** *geodesic-space*) *slim-triangles-implies-hyperbolic*:
assumes ⋀(*x*::′*a*) *y z Gxy Gyz Gxz. geodesic-segment-between Gxy x y* ⟹ *geodesic-segment-between Gxz x z* ⟹ *geodesic-segment-between Gyz y z*
⟹ ∃ *w. infdist w Gxy* ≤ *delta* ∧ *infdist w Gxz* ≤ *delta* ∧ *infdist w Gyz* ≤ *delta*
shows *Gromov-hyperbolic-subset* (*6* ∗ *delta*) (*UNIV*::′*a set*)
proof −

First step: the result is true for $t = (y, z)_x$.

have *Main*: *dist* (*geodesic-segment-param Gxy x* (*Gromov-product-at x y z*)) (*geodesic-segment-param Gxz x* (*Gromov-product-at x y z*)) ≤ *6* ∗ *delta*
if *H*: *geodesic-segment-between Gxy x y geodesic-segment-between Gxz x z*
for *x y z Gxy Gxz*
proof −
obtain *w* **where** *w*: *infdist w Gxy* ≤ *delta infdist w Gxz* ≤ *delta infdist w* {*y*−−*z*} ≤ *delta*
using *assms*[*OF H*, *of* {*y*−−*z*}] **by** *auto*
have ∃ *wxy* ∈ *Gxy*. *infdist w Gxy* = *dist w wxy*
apply (*rule infdist-proper-attained*) **using** *H*(*1*) **by** (*auto simp add*: *geodesic-segment-topology*)
then obtain *wxy* **where** *wxy*: *wxy* ∈ *Gxy dist w wxy* ≤ *delta*
using *w* **by** *auto*
have ∃ *wxz* ∈ *Gxz*. *infdist w Gxz* = *dist w wxz*
apply (*rule infdist-proper-attained*) **using** *H*(*2*) **by** (*auto simp add*: *geodesic-segment-topology*)
then obtain *wxz* **where** *wxz*: *wxz* ∈ *Gxz dist w wxz* ≤ *delta*
using *w* **by** *auto*
have ∃ *wyz* ∈ {*y*−−*z*}. *infdist w* {*y*−−*z*} = *dist w wyz*
apply (*rule infdist-proper-attained*) **by** (*auto simp add*: *geodesic-segment-topology*)
then obtain *wyz* **where** *wyz*: *wyz* ∈ {*y*−−*z*} *dist w wyz* ≤ *delta*
using *w* **by** *auto*

have *I*: *dist wxy wxz* ≤ *2* ∗ *delta dist wxy wyz* ≤ *2* ∗ *delta dist wxz wyz* ≤ *2* ∗ *delta*
using *metric-space-class.dist-triangle*[*of wxy wxz w*] *metric-space-class.dist-triangle*[*of wxy wyz w*] *metric-space-class.dist-triangle*[*of wxz wyz w*]
wxy(*2*) *wyz*(*2*) *wxz*(*2*) **by** (*auto simp add*: *metric-space-class.dist-commute*)

We show that $d(x, wxy)$ is close to the Gromov product of y and z seen from x. This follows from the fact that w is essentially on all geodesics, so that everything simplifies when one writes down the Gromov products, leaving only $d(x, w)$ up to $O(\delta)$. To get the right $O(\delta)$, one has to be a little bit careful, using the triangular inequality when possible. This means that the computations for the upper and lower bounds are different, making them a little bit tedious, although straightforward.

have *dist y wxy* −*4* ∗ *delta* + *dist wxy z* ≤ *dist y wxy* − *dist wxy wyz* + *dist wxy z* − *dist wxy wyz*
using *I* **by** *simp*
also have ... ≤ *dist wyz y* + *dist wyz z*
using *metric-space-class.dist-triangle*[*of y wxy wyz*] *metric-space-class.dist-triangle*[*of wxy z wyz*]
by (*auto simp add*: *metric-space-class.dist-commute*)
also have ... = *dist y z*
using *wyz*(*1*) **by** (*metis geodesic-segment-dist local.some-geodesic-is-geodesic-segment*(*1*) *metric-space-class.dist-commute*)
finally have ∗: *dist y wxy* + *dist wxy z* − *4* ∗ *delta* ≤ *dist y z* **by** *simp*
have *2* ∗ *Gromov-product-at x y z* = *dist x y* + *dist x z* − *dist y z*
unfolding *Gromov-product-at-def* **by** *simp*

also have ... $\leq$ *dist x wxy* + *dist wxy y* + *dist x wxy* + *dist wxy z* − (*dist y wxy* + *dist wxy z* − *4* ∗ *delta*)
using *metric-space-class.dist-triangle*[*of x y wxy*] *metric-space-class.dist-triangle*[*of x z wxy*] ∗
by (*auto simp add*: *metric-space-class.dist-commute*)
also have ... = *2* ∗ *dist x wxy* + *4* ∗ *delta*
by (*auto simp add*: *metric-space-class.dist-commute*)
finally have *A*: *Gromov-product-at x y z* $\leq$ *dist x wxy* + *2* ∗ *delta* **by** *simp*

have *dist x wxy* −*4* ∗ *delta* + *dist wxy z* $\leq$ *dist x wxy* − *dist wxy wxz* + *dist wxy z* − *dist wxy wxz*
using *I* **by** *simp*
also have ... $\leq$ *dist wxz x* + *dist wxz z*
using *metric-space-class.dist-triangle*[*of x wxy wxz*] *metric-space-class.dist-triangle*[*of wxy z wxz*]
by (*auto simp add*: *metric-space-class.dist-commute*)
also have ... = *dist x z*
using *wxz*(*1*) *H*(*2*) **by** (*metis geodesic-segment-dist metric-space-class.dist-commute*)
finally have ∗: *dist x wxy* + *dist wxy z* − *4* ∗ *delta* $\leq$ *dist x z* **by** *simp*
have *2* ∗ *dist x wxy* − *4* ∗ *delta* = (*dist x wxy* + *dist wxy y*) + (*dist x wxy* + *dist wxy z* − *4* ∗ *delta*) − (*dist y wxy* + *dist wxy z*)
by (*auto simp add*: *metric-space-class.dist-commute*)
also have ... $\leq$ *dist x y* + *dist x z* − *dist y z*
using ∗ *metric-space-class.dist-triangle*[*of y z wxy*] *geodesic-segment-dist*[*OF H*(*1*) *wxy*(*1*)] **by** *auto*
also have ... = *2* ∗ *Gromov-product-at x y z*
unfolding *Gromov-product-at-def* **by** *simp*
finally have *B*: *Gromov-product-at x y z* $\geq$ *dist x wxy* − *2* ∗ *delta* **by** *simp*

define *dy* **where** *dy* = *dist x wxy*
have ∗: *wxy* = *geodesic-segment-param Gxy x dy*
unfolding *dy-def* **using** ‹*wxy* ∈ *Gxy*› *H*(*1*) **by** *auto*
have *dist wxy* (*geodesic-segment-param Gxy x* (*Gromov-product-at x y z*)) = *abs*(*dy* − *Gromov-product-at x y z*)
unfolding ∗ **apply** (*rule geodesic-segment-param*(*7*)[*OF H*(*1*)])
unfolding *dy-def* **using** *that geodesic-segment-dist-le*[*OF H*(*1*) *wxy*(*1*), *of x*] **by** (*auto simp add*: *metric-space-class.dist-commute*)
also have ... $\leq$ *2* ∗ *delta*
using *A B* **unfolding** *dy-def* **by** *auto*
finally have *Iy*: *dist wxy* (*geodesic-segment-param Gxy x* (*Gromov-product-at x y z*)) $\leq$ *2* ∗ *delta*
by *simp*

We need the same estimate for *wxz*. The proof is exactly the same, copied and pasted. It would be better to have a separate statement, but since its assumptions would be rather cumbersome I decided to keep the two proofs.

have *dist z wxz* −*4* ∗ *delta* + *dist wxz y* $\leq$ *dist z wxz* − *dist wxz wyz* + *dist wxz y* − *dist wxz wyz*
using *I* **by** *simp*

also have ... ≤ *dist wyz z* + *dist wyz y*
using *metric-space-class.dist-triangle*[*of z wxz wyz*] *metric-space-class.dist-triangle*[*of wxz y wyz*]
by (*auto simp add*: *metric-space-class.dist-commute*)
also have ... = *dist z y*
using ‹*dist wyz y* + *dist wyz z* = *dist y z*› **by** (*auto simp add*: *metric-space-class.dist-commute*)
finally have ∗: *dist z wxz* + *dist wxz y* − *4* ∗ *delta* ≤ *dist z y* **by** *simp*
have *2* ∗ *Gromov-product-at x y z* = *dist x z* + *dist x y* − *dist z y*
unfolding *Gromov-product-at-def* **by** (*simp add*: *metric-space-class.dist-commute*)
also have ... ≤ *dist x wxz* + *dist wxz z* + *dist x wxz* + *dist wxz y* − (*dist z wxz* + *dist wxz y* − *4* ∗ *delta*)
using *metric-space-class.dist-triangle*[*of x z wxz*] *metric-space-class.dist-triangle*[*of x y wxz*] ∗
by (*auto simp add*: *metric-space-class.dist-commute*)
also have ... = *2* ∗ *dist x wxz* + *4* ∗ *delta*
by (*auto simp add*: *metric-space-class.dist-commute*)
finally have *A*: *Gromov-product-at x y z* ≤ *dist x wxz* + *2* ∗ *delta* **by** *simp*

have *dist x wxz* −*4* ∗ *delta* + *dist wxz y* ≤ *dist x wxz* − *dist wxz wxy* + *dist wxz y* − *dist wxz wxy*
using *I* **by** (*simp add*: *metric-space-class.dist-commute*)
also have ... ≤ *dist wxy x* + *dist wxy y*
using *metric-space-class.dist-triangle*[*of x wxz wxy*] *metric-space-class.dist-triangle*[*of wxz y wxy*]
by (*auto simp add*: *metric-space-class.dist-commute*)
also have ... = *dist x y*
using *wxy*(*1*) *H*(*1*) **by** (*metis geodesic-segment-dist metric-space-class.dist-commute*)
finally have ∗: *dist x wxz* + *dist wxz y* − *4* ∗ *delta* ≤ *dist x y* **by** *simp*
have *2* ∗ *dist x wxz* − *4* ∗ *delta* = (*dist x wxz* + *dist wxz z*) + (*dist x wxz* + *dist wxz y* − *4* ∗ *delta*) − (*dist z wxz* + *dist wxz y*)
by (*auto simp add*: *metric-space-class.dist-commute*)
also have ... ≤ *dist x z* + *dist x y* − *dist z y*
using ∗ *metric-space-class.dist-triangle*[*of z y wxz*] *geodesic-segment-dist*[*OF H*(*2*) *wxz*(*1*)] **by** *auto*
also have ... = *2* ∗ *Gromov-product-at x y z*
unfolding *Gromov-product-at-def* **by** (*simp add*: *metric-space-class.dist-commute*)
finally have *B*: *Gromov-product-at x y z* ≥ *dist x wxz* − *2* ∗ *delta* **by** *simp*

define *dz* **where** *dz* = *dist x wxz*
have ∗: *wxz* = *geodesic-segment-param Gxz x dz*
unfolding *dz-def* **using** ‹*wxz* ∈ *Gxz*› *H*(*2*) **by** *auto*
have *dist wxz* (*geodesic-segment-param Gxz x* (*Gromov-product-at x y z*)) = *abs*(*dz* − *Gromov-product-at x y z*)
unfolding ∗ **apply** (*rule geodesic-segment-param*(*7*)[*OF H*(*2*)])
unfolding *dz-def* **using** *that geodesic-segment-dist-le*[*OF H*(*2*) *wxz*(*1*), *of x*] **by** (*auto simp add*: *metric-space-class.dist-commute*)
also have ... ≤ *2* ∗ *delta*
using *A B* **unfolding** *dz-def* **by** *auto*

finally have *Iz*: *dist wxz* (*geodesic-segment-param Gxz x* (*Gromov-product-at x y z*)) $\leq$ *2* $*$ *delta*
by *simp*

have *dist* (*geodesic-segment-param Gxy x* (*Gromov-product-at x y z*)) (*geodesic-segment-param Gxz x* (*Gromov-product-at x y z*))
$\leq$ *dist* (*geodesic-segment-param Gxy x* (*Gromov-product-at x y z*)) *wxy* + *dist wxy wxz* + *dist wxz* (*geodesic-segment-param Gxz x* (*Gromov-product-at x y z*))
by (*rule dist-triangle4*)
also have ... $\leq$ *2* $*$ *delta* + *2* $*$ *delta* + *2* $*$ *delta*
using *Iy Iz I* **by** (*auto simp add*: *metric-space-class.dist-commute*)
finally show *?thesis* **by** *simp*
qed

Second step: the result is true for $t \leq (y, z)_x$, by a continuity argument and a reduction to the first step.

have *dist* (*geodesic-segment-param Gxy x t*) (*geodesic-segment-param Gxz x t*) $\leq$ *6* $*$ *delta*
if *H*: *geodesic-segment-between Gxy x y geodesic-segment-between Gxz x z t* $\in$ {*0..Gromov-product-at x y z*}
for *x y z t Gxy Gxz*
proof −
define *ys* **where** *ys* = (λ*s. geodesic-segment-param Gxy x* (*s* $*$ *dist x y*))
define *zs* **where** *zs* = (λ*s. geodesic-segment-param Gxz x* (*s* $*$ *dist x z*))
define *F* **where** *F* = (λ*s. Gromov-product-at x* (*ys s*) (*zs s*))
have $\exists$*s. 0* $\leq$ *s* $\wedge$ *s* $\leq$ *1* $\wedge$ *F s* = *t*
proof (*rule IVT'*)
show *F 0* $\leq$ *t t* $\leq$ *F 1*
unfolding *F-def* **using** *that* **unfolding** *ys-def zs-def* **by** (*auto simp add*: *Gromov-product-e-x-x*)
show *continuous-on* {*0..1*} *F*
unfolding *F-def Gromov-product-at-def ys-def zs-def*
apply (*intro continuous-intros continuous-on-compose2*[*of* {*0..dist x y*} - - λ*t. t* $*$ *dist x y*] *continuous-on-compose2*[*of* {*0..dist x z*} - - λ*t. t* $*$ *dist x z*])
apply (*auto intro*!: *isometry-on-continuous geodesic-segment-param*(*4*) *that*)
using *metric-space-class.zero-le-dist mult-left-le-one-le* **by** *blast+*
qed (*simp*)
then obtain *s* **where** *s*: *s* $\in$ {*0..1*} *t* = *Gromov-product-at x* (*ys s*) (*zs s*)
unfolding *F-def* **by** *auto*

have *a*: *x* = *geodesic-segment-param Gxy x 0* **using** *H*(*1*) **by** *auto*
have *b*: *x* = *geodesic-segment-param Gxz x 0* **using** *H*(*2*) **by** *auto*
have *dy*: *dist x* (*ys s*) = *s* $*$ *dist x y*
unfolding *ys-def* **apply** (*rule geodesic-segment-param*[*OF H*(*1*)]) **using** *s*(*1*) **by** (*auto simp add*: *mult-left-le-one-le*)
have *dz*: *dist x* (*zs s*) = *s* $*$ *dist x z*
unfolding *zs-def* **apply** (*rule geodesic-segment-param*[*OF H*(*2*)]) **using** *s*(*1*) **by** (*auto simp add*: *mult-left-le-one-le*)

define *Gxy2* **where** *Gxy2* = *geodesic-subsegment Gxy x 0* (*s* ∗ *dist x y*)
define *Gxz2* **where** *Gxz2* = *geodesic-subsegment Gxz x 0* (*s* ∗ *dist x z*)

have *dist* (*geodesic-segment-param Gxy2 x t*) (*geodesic-segment-param Gxz2 x t*) ≤ *6* ∗ *delta*
unfolding *s*(*2*) **proof** (*rule Main*)
show *geodesic-segment-between Gxy2 x* (*ys s*)
apply (*subst a*) **unfolding** *Gxy2-def ys-def* **apply** (*rule geodesic-subsegment*[*OF H*(*1*)])
using *s*(*1*) **by** (*auto simp add*: *mult-left-le-one-le*)
show *geodesic-segment-between Gxz2 x* (*zs s*)
apply (*subst b*) **unfolding** *Gxz2-def zs-def* **apply** (*rule geodesic-subsegment*[*OF H*(*2*)])
using *s*(*1*) **by** (*auto simp add*: *mult-left-le-one-le*)
qed
moreover have *geodesic-segment-param Gxy2 x* (*t*−*0*) = *geodesic-segment-param Gxy x t*
apply (*subst a*) **unfolding** *Gxy2-def* **apply** (*rule geodesic-subsegment*(*3*)[*OF H*(*1*)])
using *s*(*1*) *H*(*3*) **unfolding** *s*(*2*) **apply** (*auto simp add*: *mult-left-le-one-le*)
unfolding *dy*[*symmetric*] **by** (*rule Gromov-product-le-dist*)
moreover have *geodesic-segment-param Gxz2 x* (*t*−*0*) = *geodesic-segment-param Gxz x t*
apply (*subst b*) **unfolding** *Gxz2-def* **apply** (*rule geodesic-subsegment*(*3*)[*OF H*(*2*)])
using *s*(*1*) *H*(*3*) **unfolding** *s*(*2*) **apply** (*auto simp add*: *mult-left-le-one-le*)
unfolding *dz*[*symmetric*] **by** (*rule Gromov-product-le-dist*)
ultimately show *?thesis* **by** *simp*
qed
with *controlled-thin-triangles-implies-hyperbolic*[*OF this*]
show *?thesis* **by** *auto*
qed

9 Metric trees

Metric trees have several equivalent definitions. The simplest one is probably that it is a geodesic space in which the union of two geodesic segments intersecting only at one endpoint is still a geodesic segment.

Metric trees are Gromov hyperbolic, with $\delta = 0$.

class *metric-tree* = *geodesic-space* +
assumes *geod-union*: *geodesic-segment-between G x y* ⟹ *geodesic-segment-between H y z* ⟹ *G* ∩ *H* = {*y*} ⟹ *geodesic-segment-between* (*G* ∪ *H*) *x z*

We will now show that the real line is a metric tree, by identifying its geodesic segments, i.e., the compact intervals.

lemma *geodesic-segment-between-real*:
assumes *x* ≤ (*y*::*real*)

shows *geodesic-segment-between* $(G::real\ set)\ x\ y = (G = \{x..y\})$
proof
assume H: *geodesic-segment-between* $G\ x\ y$
then have *connected* $G\ x \in G\ y \in G$
using *geodesic-segment-topology*(2) *geodesic-segmentI geodesic-segment-endpoints* **by** *auto*
then have $*$: $\{x..y\} \subseteq G$
by (*simp add*: *connected-contains-Icc*)
moreover have $G \subseteq \{x..y\}$
proof
fix s **assume** $s \in G$
have $abs(s-x) + abs(s-y) = abs(x-y)$
using *geodesic-segment-dist*[*OF H* ‹$s \in G$›] **unfolding** *dist-real-def* **by** *auto*
then show $s \in \{x..y\}$ **using** ‹$x \leq y$› **by** *auto*
qed
ultimately show $G = \{x..y\}$ **by** *auto*
next
assume H: $G = \{x..y\}$
define g **where** $g = (\lambda t.\ t + x)$
have $g\ 0 = x \wedge g\ (dist\ x\ y) = y \wedge$ *isometry-on* $\{0..dist\ x\ y\}\ g \wedge G = g\ `\ \{0..dist\ x\ y\}$
unfolding *g-def isometry-on-def H* **using** ‹$x \leq y$› **by** (*auto simp add*: *dist-real-def*)
then have $\exists g.\ g\ 0 = x \wedge g\ (dist\ x\ y) = y \wedge$ *isometry-on* $\{0..dist\ x\ y\}\ g \wedge G = g\ `\ \{0..dist\ x\ y\}$
by *auto*
then show *geodesic-segment-between* $G\ x\ y$ **unfolding** *geodesic-segment-between-def* **by** *auto*
qed

lemma *geodesic-segment-between-real'*:
$\{x--y\} = \{min\ x\ y..max\ x\ (y::real)\}$
by (*metis geodesic-segment-between-real geodesic-segment-commute some-geodesic-is-geodesic-segment*(1) *max-def min.cobounded1 min-def*)

lemma *geodesic-segment-real*:
geodesic-segment $(G::real\ set) = (\exists x\ y.\ x \leq y \wedge G = \{x..y\})$
proof
assume *geodesic-segment* G
then obtain $x\ y$ **where** $*$: *geodesic-segment-between* $G\ x\ y$ **unfolding** *geodesic-segment-def* **by** *auto*
have $(x \leq y \wedge G = \{x..y\}) \vee (y \leq x \wedge G = \{y..x\})$
apply (*rule le-cases*[*of x y*])
using *geodesic-segment-between-real* $*$ *geodesic-segment-commute* **apply** *simp*
using *geodesic-segment-between-real* $*$ *geodesic-segment-commute* **by** *metis*
then show $\exists x\ y.\ x \leq y \wedge G = \{x..y\}$ **by** *auto*
next
assume $\exists x\ y.\ x \leq y \wedge G = \{x..y\}$
then show *geodesic-segment* G
unfolding *geodesic-segment-def* **using** *geodesic-segment-between-real* **by** *metis*

qed

instance *real*::*metric-tree*
proof
fix *G H*::*real set* **and** *x y z*::*real* **assume** *GH*: *geodesic-segment-between G x y geodesic-segment-between H y z* $G \cap H = \{y\}$
have *G*: $G = \{min\ x\ y..max\ x\ y\}$ **using** *GH*
by (*metis geodesic-segment-between-real geodesic-segment-commute inf-real-def inf-sup-ord*(*2*) *max.coboundedI2 max-def min-def*)
have *H*: $H = \{min\ y\ z..max\ y\ z\}$ **using** *GH*
by (*metis geodesic-segment-between-real geodesic-segment-commute inf-real-def inf-sup-ord*(*2*) *max.coboundedI2 max-def min-def*)
have ∗: $(x \leq y \wedge y \leq z) \vee (z \leq y \wedge y \leq x)$
using *G H* ‹$G \cap H = \{y\}$› **unfolding** *min-def max-def*
apply *auto*
apply (*metis* (*mono-tags*, *opaque-lifting*) *min-le-iff-disj order-refl*)
by (*metis* (*full-types*) *less-eq-real-def max-def*)
show *geodesic-segment-between* $(G \cup H)$ *x z*
using ∗ **apply** *rule*
using ‹$G \cap H = \{y\}$› **unfolding** *G H* **apply** (*metis G GH*(*1*) *GH*(*2*) *H geodesic-segment-between-real ivl-disj-un-two-touch*(*4*) *order-trans*)
using ‹$G \cap H = \{y\}$› **unfolding** *G H*
by (*metis* (*full-types*) *Un-commute geodesic-segment-between-real geodesic-segment-commute ivl-disj-un-two-touch*(*4*) *le-max-iff-disj max.absorb-iff2 max.commute min-absorb2*)
qed

context *metric-tree* **begin**

We show that a metric tree is uniquely geodesic.

subclass *uniquely-geodesic-space*
proof
fix *x y G H* **assume** *H*: *geodesic-segment-between G x y geodesic-segment-between H x* $(y::'a)$
show $G = H$
proof (*rule uniquely-geodesic-spaceI*[*OF - H*])
fix *G H x y* **assume** *geodesic-segment-between G x y geodesic-segment-between H x y* $G \cap H = \{x, (y::'a)\}$
show $x = y$
proof (*rule ccontr*)
assume $x \neq y$
then have $dist\ x\ y > 0$ **by** *auto*
obtain *g* **where** *g*: $g\ 0 = x$ $g\ (dist\ x\ y) = y$ *isometry-on* $\{0..dist\ x\ y\}$ *g* $G = g`\{0..dist\ x\ y\}$
by (*meson* ‹*geodesic-segment-between G x y*› *geodesic-segment-between-def*)
define *G2* **where** $G2 = g`\{0..dist\ x\ y/2\}$
have $G2 \subseteq G$ **unfolding** *G2-def g*(*4*) **by** *auto*
define *z* **where** $z = g(dist\ x\ y/2)$
have $dist\ x\ z = dist\ x\ y/2$
using *isometry-onD*[*OF g*(*3*), *of 0 dist x y/2*] *g*(*1*) *z-def* **unfolding**

dist-real-def **by** *auto*
have *dist y z* = *dist x y*/*2*
using *isometry-onD*[*OF g*(*3*), *of dist x y dist x y*/*2*] *g*(*2*) *z-def* **unfolding** *dist-real-def* **by** *auto*

have *G2*: *geodesic-segment-between G2 x z* **unfolding** ‹*g 0* = *x*›[*symmetric*] *z-def G2-def*
apply (*rule geodesic-segmentI2*) **by** (*rule isometry-on-subset*[*OF g*(*3*)], *auto simp add*: ‹*g 0* = *x*›)
have [*simp*]: *x* ∈ *G2 z* ∈ *G2* **using** *geodesic-segment-endpoints G2* **by** *auto*
have *dist x a* ≤ *dist x z* **if** *a* ∈ *G2* **for** *a*
apply (*rule geodesic-segment-dist-le*) **using** *G2 that* **by** *auto*
also have ... < *dist x y* **unfolding** ‹*dist x z* = *dist x y*/*2*› **using** ‹*dist x y* > *0*› **by** *auto*
finally have *y* ∉ *G2* **by** *auto*

then have *G2* ∩ *H* = {*x*}
using ‹*G2* ⊆ *G*› ‹*x* ∈ *G2*› ‹*G* ∩ *H* = {*x*, *y*}› **by** *auto*
have ∗: *geodesic-segment-between* (*G2* ∪ *H*) *z y*
apply (*rule geod-union*[*of - - x*])
using ‹*G2* ∩ *H* = {*x*}› ‹*geodesic-segment-between H x y*› *G2* **by** (*auto simp add*: *geodesic-segment-commute*)
have *dist x y* ≤ *dist z x* + *dist x y* **by** *auto*
also have ... = *dist z y*
apply (*rule geodesic-segment-dist*[*OF* ∗]) **using** ‹*G* ∩ *H* = {*x*, *y*}› **by** *auto*
also have ... = *dist x y* / *2*
by (*simp add*: ‹*dist y z* = *dist x y* / *2*› *metric-space-class.dist-commute*)
finally show *False* **using** ‹*dist x y* > *0*› **by** *auto*
qed
qed
qed

An important property of metric trees is that any geodesic triangle is degenerate, i.e., the three sides intersect at a unique point, the center of the triangle, that we introduce now.

definition *center*::′*a* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a*
where *center x y z* = (*SOME t*. *t* ∈ {*x*−−*y*} ∩ {*x*−−*z*} ∩ {*y*−−*z*})

lemma *center-as-intersection*:
{*x*−−*y*} ∩ {*x*−−*z*} ∩ {*y*−−*z*} = {*center x y z*}
proof −
obtain *g* **where** *g*: *g 0* = *x g* (*dist x y*) = *y isometry-on* {*0*..*dist x y*} *g* {*x*−−*y*} = *g*‘{*0*..*dist x y*}
by (*meson geodesic-segment-between-def some-geodesic-is-geodesic-segment*(*1*))
obtain *h* **where** *h*: *h 0* = *x h* (*dist x z*) = *z isometry-on* {*0*..*dist x z*} *h* {*x*−−*z*} = *h*‘{*0*..*dist x z*}
by (*meson geodesic-segment-between-def some-geodesic-is-geodesic-segment*(*1*))

define *Z* **where** *Z* = {*t* ∈ {*0*..*min* (*dist x y*) (*dist x z*)}. *g t* = *h t*}

have *0* ∈ *Z* **unfolding** *Z-def* **using** *g*(*1*) *h*(*1*) **by** *auto*
have [*simp*]: *closed Z*
proof −
have ∗: *Z* = (λ*s*. *dist* (*g s*) (*h s*))−'{*0*} ∩ {*0*..*min* (*dist x y*) (*dist x z*)}
unfolding *Z-def* **by** *auto*
show *?thesis*
unfolding ∗ **apply** (*rule closed-vimage-Int*)
using *continuous-on-subset*[*OF isometry-on-continuous*[*OF g*(*3*)], *of* {*0*..*min* (*dist x y*) (*dist x z*)}]
continuous-on-subset[*OF isometry-on-continuous*[*OF h*(*3*)], *of* {*0*..*min* (*dist x y*) (*dist x z*)}]
continuous-on-dist **by** *auto*
qed
define *a* **where** *a* = *Sup Z*
have *a* ∈ *Z*
unfolding *a-def* **apply** (*rule closed-contains-Sup*, *auto*) **using** ‹*0* ∈ *Z*› *Z-def* **by** *auto*
define *c* **where** *c* = *h a*
then have *a*: *g a* = *c h a* = *c a* ≥ *0 a* ≤ *dist x y a* ≤ *dist x z*
using ‹*a* ∈ *Z*› **unfolding** *Z-def c-def* **by** *auto*

define *G2* **where** *G2* = *g*'{*a*..*dist x y*}
have *G2*: *geodesic-segment-between G2* (*g a*) (*g* (*dist x y*))
unfolding *G2-def* **apply** (*rule geodesic-segmentI2*)
using *isometry-on-subset*[*OF g*(*3*)] ‹*a* ∈ *Z*› **unfolding** *Z-def* **by** *auto*
define *H2* **where** *H2* = *h*'{*a*..*dist x z*}
have *H2*: *geodesic-segment-between H2* (*h a*) (*h* (*dist x z*))
unfolding *H2-def* **apply** (*rule geodesic-segmentI2*)
using *isometry-on-subset*[*OF h*(*3*)] ‹*a* ∈ *Z*› **unfolding** *Z-def* **by** *auto*
have *G2* ∩ *H2* ⊆ {*c*}
proof
fix *w* **assume** *w*: *w* ∈ *G2* ∩ *H2*
obtain *sg* **where** *sg*: *w* = *g sg sg* ∈ {*a*..*dist x y*} **using** *w* **unfolding** *G2-def* **by** *auto*
obtain *sh* **where** *sh*: *w* = *h sh sh* ∈ {*a*..*dist x z*} **using** *w* **unfolding** *H2-def* **by** *auto*
have *dist w x* = *sg*
unfolding *g*(*1*)[*symmetric*] *sg*(*1*) **using** *isometry-onD*[*OF g*(*3*), *of 0 sg*] *sg*(*2*)
unfolding *dist-real-def* **using** *a* **by** (*auto simp add*: *metric-space-class.dist-commute*)
moreover have *dist w x* = *sh*
unfolding *h*(*1*)[*symmetric*] *sh*(*1*) **using** *isometry-onD*[*OF h*(*3*), *of 0 sh*] *sh*(*2*)
unfolding *dist-real-def* **using** *a* **by** (*auto simp add*: *metric-space-class.dist-commute*)
ultimately have *sg* = *sh* **by** *simp*
have *sh* ∈ *Z* **unfolding** *Z-def* **using** *sg sh* ‹*a* ≥ *0*› **unfolding** ‹*sg* = *sh*› **by** *auto*
then have *sh* ≤ *a*
unfolding *a-def* **apply** (*rule cSup-upper*) **unfolding** *Z-def* **by** *auto*
then have *sh* = *a* **using** *sh*(*2*) **by** *auto*
then show *w* ∈ {*c*} **unfolding** *sh*(*1*) **using** *a*(*2*) **by** *auto*

qed
then have ∗: $G2 \cap H2 = \{c\}$
unfolding *G2-def H2-def* **using** *a* **by** (*auto simp add*: *image-iff*, *force*)
have *geodesic-segment-between* $(G2 \cup H2)$ *y z*
apply (*subst g*(*2*)[*symmetric*], *subst h*(*2*)[*symmetric*]) **apply**(*rule geod-union*[*of* - - *h a*])
using *geodesic-segment-commute G2 H2 a* ∗ **by** *force*+
then have $G2 \cup H2 = \{y{-}{-}z\}$
using *geodesic-segment-unique* **by** *auto*
then have $c \in \{y{-}{-}z\}$ **using** ∗ **by** *auto*
then have ∗: $c \in \{x{-}{-}y\} \cap \{x{-}{-}z\} \cap \{y{-}{-}z\}$
using *g*(*4*) *h*(*4*) *c-def a* **by** *force*
have *center*: *center x y z* $\in \{x{-}{-}y\} \cap \{x{-}{-}z\} \cap \{y{-}{-}z\}$
unfolding *center-def* **using** *someI*[*of* $\lambda p.\ p \in \{x{-}{-}y\} \cap \{x{-}{-}z\} \cap \{y{-}{-}z\}$, *OF* ∗] **by** *blast*
have ∗: *dist x d* = *Gromov-product-at x y z* **if** $d \in \{x{-}{-}y\} \cap \{x{-}{-}z\} \cap \{y{-}{-}z\}$ **for** *d*
proof −
have *dist x y* = *dist x d* + *dist d y*
dist x z = *dist x d* + *dist d z*
dist y z = *dist y d* + *dist d z*
using *that* **by** (*auto simp add*: *geodesic-segment-dist geodesic-segment-unique*)
then show *?thesis* **unfolding** *Gromov-product-at-def* **by** (*auto simp add*: *metric-space-class.dist-commute*)
qed
have *d* = *center x y z* **if** $d \in \{x{-}{-}y\} \cap \{x{-}{-}z\} \cap \{y{-}{-}z\}$ **for** *d*
apply (*rule geodesic-segment-dist-unique*[*of* $\{x{-}{-}y\}$ *x y*])
using ∗[*OF that*] ∗[*OF center*] *that center* **by** *auto*
then show $\{x{-}{-}y\} \cap \{x{-}{-}z\} \cap \{y{-}{-}z\} = \{$*center x y z*$\}$ **using** *center* **by** *blast*
qed

lemma *center-on-geodesic* [*simp*]:
center x y z $\in \{x{-}{-}y\}$
center x y z $\in \{x{-}{-}z\}$
center x y z $\in \{y{-}{-}z\}$
center x y z $\in \{y{-}{-}x\}$
center x y z $\in \{z{-}{-}x\}$
center x y z $\in \{z{-}{-}y\}$
using *center-as-intersection* **by** (*auto simp add*: *some-geodesic-commute*)

lemma *center-commute*:
center x y z = *center x z y*
center x y z = *center y x z*
center x y z = *center y z x*
center x y z = *center z x y*
center x y z = *center z y x*
using *center-as-intersection some-geodesic-commute* **by** *blast*+

lemma *center-dist*:
dist x (*center x y z*) = *Gromov-product-at x y z*
proof −
have *dist x y* = *dist x* (*center x y z*) + *dist* (*center x y z*) *y*
dist x z = *dist x* (*center x y z*) + *dist* (*center x y z*) *z*
dist y z = *dist y* (*center x y z*) + *dist* (*center x y z*) *z*
by (*auto simp add*: *geodesic-segment-dist geodesic-segment-unique*)
then show *?thesis* **unfolding** *Gromov-product-at-def* **by** (*auto simp add*: *metric-space-class.dist-commute*)
qed

lemma *geodesic-intersection*:
{*x*−−*y*} ∩ {*x*−−*z*} = {*x*−−*center x y z*}
proof −
have {*x*−−*y*} = {*x*−−*center x y z*} ∪ {*center x y z*−−*y*}
using *center-as-intersection geodesic-segment-split* **by** *blast*
moreover have {*x*−−*z*} = {*x*−−*center x y z*} ∪ {*center x y z*−−*z*}
using *center-as-intersection geodesic-segment-split* **by** *blast*
ultimately have {*x*−−*y*} ∩ {*x*−−*z*} = {*x*−−*center x y z*} ∪ ({*center x y z*−−*y*} ∩ {*x*−−*center x y z*}) ∪ ({*center x y z*−−*y*} ∩ {*x*−−*center x y z*}) ∪ ({*center x y z*−−*y*} ∩ {*center x y z*−−*z*})
by *auto*
moreover have {*center x y z*−−*y*} ∩ {*x*−−*center x y z*} = {*center x y z*}
using *geodesic-segment-split*(*2*) *center-as-intersection*[*of x y z*] **by** *auto*
moreover have {*center x y z*−−*y*} ∩ {*x*−−*center x y z*} = {*center x y z*}
using *geodesic-segment-split*(*2*) *center-as-intersection*[*of x y z*] **by** *auto*
moreover have {*center x y z*−−*y*} ∩ {*center x y z*−−*z*} = {*center x y z*}
using *geodesic-segment-split*(*2*)[*of center x y z y z*] *center-as-intersection*[*of x y z*] **by** (*auto simp add*: *some-geodesic-commute*)
ultimately show {*x*−−*y*} ∩ {*x*−−*z*} = {*x*−−*center x y z*} **by** *auto*
qed
end

We can now prove that a metric tree is Gromov hyperbolic, for $\delta = 0$. The simplest proof goes through the slim triangles property: it suffices to show that, given a geodesic triangle, there is a point at distance at most 0 of each of its sides. This is the center we have constructed above.

class *metric-tree-with-delta* = *metric-tree* + *metric-space-with-deltaG* +
assumes *delta0*: *deltaG*(*TYPE*(*'a*::*metric-space*)) = *0*

class *Gromov-hyperbolic-space-0* = *Gromov-hyperbolic-space* +
assumes *delta0* [*simp*]: *deltaG*(*TYPE*(*'a*::*metric-space*)) = *0*

class *Gromov-hyperbolic-space-0-geodesic* = *Gromov-hyperbolic-space-0* + *geodesic-space*

Isabelle does not accept cycles in the class graph. So, we will show that `metric_tree_with_delta` is a subclass of `Gromov_hyperbolic_space_0_geodesic`, and conversely that `Gromov_hyperbolic_space_0_geodesic` is a subclass of `metric_tree`.

In a tree, we have already proved that triangles are 0-slim (the center is common to all sides of the triangle). The 0-hyperbolicity follows from one of the equivalent characterizations of hyperbolicity (the other characterizations could be used as well, but the proofs would be less immediate.)

subclass (**in** *metric-tree-with-delta*) *Gromov-hyperbolic-space-0*
proof (*standard*)
  show *deltaG TYPE('a) = 0* **unfolding** *delta0* **by** *auto*
  have *Gromov-hyperbolic-subset (6 ∗ 0) (UNIV::'a set)*
  proof (*rule slim-triangles-implies-hyperbolic*)
    fix *x::'a* **and** *y z Gxy Gyz Gxz*
    define *w* **where** *w = center x y z*
    assume *geodesic-segment-between Gxy x y*
        geodesic-segment-between Gxz x z geodesic-segment-between Gyz y z
    then have *Gxy = {x−−y} Gyz = {y−−z} Gxz = {x−−z}*
      by (*auto simp add: local.geodesic-segment-unique*)
    then have *w ∈ Gxy w ∈ Gyz w ∈ Gxz*
      unfolding *w-def* **by** *auto*
    then have *infdist w Gxy ≤ 0 ∧ infdist w Gxz ≤ 0 ∧ infdist w Gyz ≤ 0*
      by *auto*
    then show *∃ w. infdist w Gxy ≤ 0 ∧ infdist w Gxz ≤ 0 ∧ infdist w Gyz ≤ 0*
      by *blast*
  qed
  then show *Gromov-hyperbolic-subset (deltaG TYPE('a)) (UNIV::'a set)* **unfolding** *delta0* **by** *auto*
qed

To use the fact that reals are Gromov hyperbolic, given that they are a metric tree, we need to instantiate them as `metric_tree_with_delta`.

instantiation *real::metric-tree-with-delta*
begin
definition *deltaG-real::real itself ⇒ real*
  where *deltaG-real - = 0*
instance apply *standard* **unfolding** *deltaG-real-def* **by** *auto*
end

Let us now prove the converse: a geodesic space which is δ-hyperbolic for $\delta = 0$ is a metric tree. For the proof, we consider two geodesic segments $G = [x, y]$ and $H = [y, z]$ with a common endpoint, and we have to show that their union is still a geodesic segment from x to z. For this, introduce a geodesic segment $L = [x, z]$. By the property of thin triangles, G is included in $H \cup L$. In particular, a point Y close to y but different from y on G is on L, and therefore realizes the equality $d(x, z) = d(x, Y) + d(Y, z)$. Passing to the limit, y also satisfies this equality. The conclusion readily follows thanks to Lemma `geodesic_segment_union`.

subclass (**in** *Gromov-hyperbolic-space-0-geodesic*) *metric-tree*
proof

fix *G H x y z* **assume** *A*: *geodesic-segment-between G x y geodesic-segment-between H y z* $G \cap H = \{y::'a\}$
show *geodesic-segment-between* $(G \cup H)$ *x z*
proof (*cases* $x = y$)
case *True*
then show *?thesis*
by (*metis A Un-commute geodesic-segment-between-x-x*(*3*) *inf.commute sup-inf-absorb*)
next
case *False*
define *D*::*nat* ⇒ *real* **where** $D = (\lambda n.\ dist\ x\ y - (dist\ x\ y) * (1/(real(n+1))))$
have *D*: $D\ n \in \{0..< dist\ x\ y\}$ $D\ n \in \{0..dist\ x\ y\}$ **for** *n*
unfolding *D-def* **by** (*auto simp add*: *False divide-simps algebra-simps*)
have *Dlim*: $D \longrightarrow dist\ x\ y - dist\ x\ y * 0$
unfolding *D-def* **by** (*intro tendsto-intros LIMSEQ-ignore-initial-segment*[*OF lim-1-over-n*, *of 1*])

define *Y*::*nat* ⇒ *'a* **where** $Y = (\lambda n.\ geodesic\text{-}segment\text{-}param\ G\ x\ (D\ n))$
have ∗: $Y \longrightarrow y$
unfolding *Y-def* **apply** (*subst geodesic-segment-param*(*2*)[*OF A*(*1*), *symmetric*])
using *isometry-on-continuous*[*OF geodesic-segment-param*(*4*)[*OF A*(*1*)]]
unfolding *continuous-on-sequentially comp-def* **using** *D*(*2*) *Dlim* **by** *auto*

have $dist\ x\ z = dist\ x\ (Y\ n) + dist\ (Y\ n)\ z$ **for** *n*
proof −
obtain *L* **where** *L*: *geodesic-segment-between L x z* **using** *geodesic-subsetD*[*OF geodesic*] **by** *blast*
have $Y\ n \in G$ **unfolding** *Y-def*
apply (*rule geodesic-segment-param*(*3*)[*OF A*(*1*)]) **using** *D*[*of n*] **by** *auto*
have $dist\ x\ (Y\ n) = D\ n$
unfolding *Y-def* **apply** (*rule geodesic-segment-param*[*OF A*(*1*)]) **using** *D*[*of n*] **by** *auto*
then have $Y\ n \neq y$
using *D*[*of n*] **by** *auto*
then have $Y\ n \notin H$ **using** *A*(*3*) ‹$Y\ n \in G$› **by** *auto*
have $infdist\ (Y\ n)\ (H \cup L) \leq 4 * deltaG(TYPE('a))$
apply (*rule thin-triangles*[*OF geodesic-segment-commute*[*OF A*(*2*)] *geodesic-segment-commute*[*OF L*] *geodesic-segment-commute*[*OF A*(*1*)]]])
using ‹$Y\ n \in G$› **by** *simp*
then have $infdist\ (Y\ n)\ (H \cup L) = 0$
using *infdist-nonneg*[*of Y n* $H \cup L$] **unfolding** *delta0* **by** *auto*
have $Y\ n \in H \cup L$
proof (*subst in-closed-iff-infdist-zero*)
have *closed H*
using *A*(*2*) *geodesic-segment-topology geodesic-segment-def* **by** *fastforce*
moreover have *closed L*
using *L geodesic-segment-topology geodesic-segment-def* **by** *fastforce*
ultimately show *closed* $(H \cup L)$ **by** *auto*
show $H \cup L \neq \{\}$ **using** *A*(*2*) *geodesic-segment-endpoints*(*1*) **by** *auto*

qed (*fact*)
then have $Y\ n \in L$ **using** ‹$Y\ n \notin H$› **by** *simp*
show *?thesis* **using** *geodesic-segment-dist*[*OF L* ‹$Y\ n \in L$›] **by** *simp*
qed
moreover have ($\lambda n.\ dist\ x\ (Y\ n) + dist\ (Y\ n)\ z$) $\longrightarrow$ $dist\ x\ y + dist\ y\ z$
by (*intro tendsto-intros* ∗)
ultimately have ($\lambda n.\ dist\ x\ z$) $\longrightarrow$ $dist\ x\ y + dist\ y\ z$
using *filterlim-cong eventually-sequentially* **by** *auto*
then have ∗: $dist\ x\ z = dist\ x\ y + dist\ y\ z$
using *LIMSEQ-unique* **by** *auto*
show *geodesic-segment-between* ($G \cup H$) $x\ z$
by (*rule geodesic-segment-union*[*OF* ∗ *A*(*1*) *A*(*2*)])
qed
qed

end

theory *Morse-Gromov-Theorem*
imports *HOL−Decision-Procs.Approximation Gromov-Hyperbolicity Hausdorff-Distance*
begin

hide-const (**open**) *Approximation.Min*
hide-const (**open**) *Approximation.Max*

10 Quasiconvexity

In a Gromov-hyperbolic setting, convexity is not a well-defined notion as everything should be coarse. The good replacement is quasi-convexity: A set X is C-quasi-convex if any pair of points in X can be joined by a geodesic that remains within distance C of X. One could also require this for all geodesics, up to changing C, as two geodesics between the same endpoints remain within uniformly bounded distance. We use the first definition to ensure that a geodesic is 0-quasi-convex.

definition *quasiconvex*::*real* ⇒ (*'a*::*metric-space*) *set* ⇒ *bool*
where *quasiconvex* $C\ X = (C \geq 0 \wedge (\forall x{\in}X.\ \forall y{\in}X.\ \exists G.$ *geodesic-segment-between* $G\ x\ y \wedge (\forall z{\in}G.\ infdist\ z\ X \leq C)))$

lemma *quasiconvexD*:
assumes *quasiconvex* $C\ X\ x \in X\ y \in X$
shows $\exists G.$ *geodesic-segment-between* $G\ x\ y \wedge (\forall z{\in}G.\ infdist\ z\ X \leq C)$
using *assms* **unfolding** *quasiconvex-def* **by** *auto*

lemma *quasiconvexC*:
assumes *quasiconvex* $C\ X$
shows $C \geq 0$

using *assms* **unfolding** *quasiconvex-def* **by** *auto*

lemma *quasiconvexI*:
assumes $C \geq 0$
$\bigwedge x\ y.\ x \in X \Longrightarrow y \in X \Longrightarrow (\exists\, G.\ \textit{geodesic-segment-between}\ G\ x\ y \wedge (\forall\, z{\in}G.\ \textit{infdist}\ z\ X \leq C))$
shows *quasiconvex C X*
using *assms* **unfolding** *quasiconvex-def* **by** *auto*

lemma *quasiconvex-of-geodesic*:
assumes *geodesic-segment G*
shows *quasiconvex 0 G*
proof (*rule quasiconvexI*, *simp*)
fix *x y* **assume** $*$: $x \in G\ y \in G$
obtain *H* **where** *H*: $H \subseteq G$ *geodesic-segment-between H x y*
using *geodesic-subsegment-exists*[*OF assms*(*1*) $*$] **by** *auto*
have *infdist* $z\ G \leq 0$ **if** $z \in H$ **for** *z*
using *H*(*1*) *that* **by** *auto*
then show $\exists\, H.\ \textit{geodesic-segment-between}\ H\ x\ y \wedge (\forall\, z{\in}H.\ \textit{infdist}\ z\ G \leq 0)$
using *H*(*2*) **by** *auto*
qed

lemma *quasiconvex-empty*:
assumes $C \geq 0$
shows *quasiconvex C* {}
unfolding *quasiconvex-def* **using** *assms* **by** *auto*

lemma *quasiconvex-mono*:
assumes $C \leq D$
quasiconvex C G
shows *quasiconvex D G*
using *assms* **unfolding** *quasiconvex-def* **by** (*auto*, *fastforce*)

The r-neighborhood of a quasi-convex set is still quasi-convex in a hyperbolic space, for a constant that does not depend on r.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *quasiconvex-thickening*:
assumes *quasiconvex C* (*X*::'*a set*) $r \geq 0$
shows *quasiconvex* ($C + 8\ *\textit{deltaG}(\textit{TYPE}('a))$) ($\bigcup x{\in}X.\ \textit{cball}\ x\ r$)
proof (*rule quasiconvexI*)
show $C + 8\ *\textit{deltaG}(\textit{TYPE}('a)) \geq 0$ **using** *quasiconvexC*[*OF assms*(*1*)] **by** *simp*
next
fix *y z* **assume** $*$: $y \in (\bigcup x{\in}X.\ \textit{cball}\ x\ r)\ z \in (\bigcup x{\in}X.\ \textit{cball}\ x\ r)$
have *A*: *infdist* $w\ (\bigcup x{\in}X.\ \textit{cball}\ x\ r) \leq C + 8 * \textit{deltaG}\ \textit{TYPE}('a)$ **if** $w \in \{y{-}{-}z\}$ **for** *w*
proof −
obtain *py* **where** *py*: $py \in X\ y \in \textit{cball}\ py\ r$
using $*$ **by** *auto*
obtain *pz* **where** *pz*: $pz \in X\ z \in \textit{cball}\ pz\ r$

using $*$ **by** *auto*
obtain G **where** G: *geodesic-segment-between* G *py pz* $(\forall\, p{\in}G.\ infdist\ p\ X \leq C)$
using *quasiconvexD*[*OF assms*(*1*) ‹$py \in X$› ‹$pz \in X$›] **by** *auto*
have A: *infdist* w $(\{y{-}{-}py\} \cup G \cup \{pz{-}{-}z\}) \leq 8 * deltaG(TYPE('a))$
by (*rule thin-quadrilaterals*[*OF* - G(*1*) - - ‹$w \in \{y{-}{-}z\}$›, **where** $?x = y$ **and** $?t = z$], *auto*)
have $\exists\, u \in \{y{-}{-}py\} \cup G \cup \{pz{-}{-}z\}.\ infdist\ w\ (\{y{-}{-}py\} \cup G \cup \{pz{-}{-}z\}) = dist\ w\ u$
apply (*rule infdist-proper-attained*, *auto intro*!: *proper-Un simp add*: *geodesic-segment-topology*(*7*))
by (*meson* G(*1*) *geodesic-segmentI geodesic-segment-topology*(*7*))
then obtain u **where** u: $u \in \{y{-}{-}py\} \cup G \cup \{pz{-}{-}z\}$ *infdist* w $(\{y{-}{-}py\} \cup G \cup \{pz{-}{-}z\}) = dist\ w\ u$
by *auto*
then consider $u \in \{y{-}{-}py\} \mid u \in G \mid u \in \{pz{-}{-}z\}$ **by** *auto*
then have *infdist* u $(\bigcup x{\in}X.\ cball\ x\ r) \leq C$
proof (*cases*)
case *1*
then have *dist py u* $\leq$ *dist py y*
using *geodesic-segment-dist-le local.some-geodesic-is-geodesic-segment*(*1*) *some-geodesic-commute some-geodesic-endpoints*(*1*) **by** *blast*
also have ... $\leq r$
using *py*(*2*) **by** *auto*
finally have $u \in$ *cball py r*
by *auto*
then have $u \in (\bigcup x{\in}X.\ cball\ x\ r)$
using *py*(*1*) **by** *auto*
then have *infdist* u $(\bigcup x{\in}X.\ cball\ x\ r) = 0$
by *auto*
then show *?thesis*
using *quasiconvexC*[*OF assms*(*1*)] **by** *auto*
next
case *3*
then have *dist pz u* $\leq$ *dist pz z*
using *geodesic-segment-dist-le local.some-geodesic-is-geodesic-segment*(*1*) *some-geodesic-commute some-geodesic-endpoints*(*1*) **by** *blast*
also have ... $\leq r$
using *pz*(*2*) **by** *auto*
finally have $u \in$ *cball pz r*
by *auto*
then have $u \in (\bigcup x{\in}X.\ cball\ x\ r)$
using *pz*(*1*) **by** *auto*
then have *infdist* u $(\bigcup x{\in}X.\ cball\ x\ r) = 0$
by *auto*
then show *?thesis*
using *quasiconvexC*[*OF assms*(*1*)] **by** *auto*
next
case *2*
have *infdist* u $(\bigcup x{\in}X.\ cball\ x\ r) \leq$ *infdist* u X

apply (*rule infdist-mono*) **using** *assms*(*2*) *py*(*1*) **by** *auto*
then show *?thesis* **using** *2 G*(*2*) **by** *auto*
qed
moreover have *infdist w* ($\bigcup$*x*∈*X*. *cball x r*) ≤ *infdist u* ($\bigcup$*x*∈*X*. *cball x r*) + *dist w u*
by (*intro mono-intros*)
ultimately show *?thesis*
using *A u*(*2*) **by** *auto*
qed
show ∃ *G*. *geodesic-segment-between G y z* ∧ (∀ *w*∈*G*. *infdist w* ($\bigcup$*x*∈*X*. *cball x r*) ≤ *C* + *8* ∗ *deltaG TYPE*(′*a*))
apply (*rule exI*[*of* - {*y*−−*z*}]) **using** *A* **by** *auto*
qed

If x has a projection p on a quasi-convex set G, then all segments from a point in G to x go close to p, i.e., the triangular inequality $d(x,y) \leq d(x,p)+d(p,y)$ is essentially an equality, up to an additive constant.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *dist-along-quasiconvex*:
assumes *quasiconvex C G p* ∈ *proj-set x G y* ∈ *G*
shows *dist x p* + *dist p y* ≤ *dist x y* + *4* ∗ *deltaG*(*TYPE*(′*a*)) + *2* ∗ *C*
proof −
have ∗: *p* ∈ *G*
using *assms proj-setD* **by** *auto*
obtain *H* **where** *H*: *geodesic-segment-between H p y* $\bigwedge$*q*. *q* ∈ *H* ⟹ *infdist q G* ≤ *C*
using *quasiconvexD*[*OF assms*(*1*) ∗ *assms*(*3*)] **by** *auto*
have ∃ *m*∈*H*. *infdist x H* = *dist x m*
apply (*rule infdist-proper-attained*[*of H x*]) **using** *geodesic-segment-topology*[*OF geodesic-segmentI*[*OF H*(*1*)]] **by** *auto*
then obtain *m* **where** *m*: *m* ∈ *H infdist x H* = *dist x m* **by** *auto*
then have *I*: *dist x m* ≤ *Gromov-product-at x p y* + *2* ∗ *deltaG*(*TYPE*(′*a*))
using *infdist-triangle-side*[*OF H*(*1*), *of x*] **by** *auto*
have *dist x p* − *dist x m* − *C* ≤ *e* **if** *e* > *0* **for** *e*
proof −
have ∃ *r*∈*G*. *dist m r* < *infdist m G* + *e*
apply (*rule infdist-almost-attained*) **using** ‹*e* > *0*› *assms*(*3*) **by** *auto*
then obtain *r* **where** *r*: *r* ∈ *G dist m r* < *infdist m G* + *e*
by *auto*
then have ∗: *dist m r* ≤ *C* + *e* **using** *H*(*2*)[*OF* ‹*m* ∈ *H*›] **by** *auto*
have *dist x p* ≤ *dist x r*
using ‹*r* ∈ *G*› *assms*(*2*) *proj-set-dist-le* **by** *blast*
also have ... ≤ *dist x m* + *dist m r*
by (*intro mono-intros*)
finally show *?thesis* **using** ∗ **by** (*auto simp add*: *metric-space-class.dist-commute*)
qed
then have *dist x p* − *dist x m* − *C* ≤ *0*
using *dense-ge* **by** *blast*
then show *?thesis*
using *I* **unfolding** *Gromov-product-at-def* **by** (*auto simp add*: *algebra-simps*

divide-simps)
qed

The next lemma is [CDP90, Proposition 10.2.1] with better constants. It states that the distance between the projections on a quasi-convex set is controlled by the distance of the original points, with a gain given by the distances of the points to the set.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *proj-along-quasiconvex-contraction*:
  assumes *quasiconvex C G px* ∈ *proj-set x G py* ∈ *proj-set y G*
  shows *dist px py* ≤ *max* (*5* ∗ *deltaG*(*TYPE*(*'a*)) + *2* ∗ *C*) (*dist x y* − *dist px x* − *dist py y* + *10* ∗ *deltaG*(*TYPE*(*'a*)) + *4* ∗ *C*)
proof −
  have *px* ∈ *G py* ∈ *G*
    using *assms proj-setD* **by** *auto*
  have (*dist x px* + *dist px py* − *4* ∗ *deltaG*(*TYPE*(*'a*)) − *2* ∗ *C*) + (*dist y py* + *dist py px* − *4* ∗*deltaG*(*TYPE*(*'a*)) − *2* ∗ *C*)
        ≤ *dist x py* + *dist y px*
    apply (*intro mono-intros*)
    using *dist-along-quasiconvex*[*OF assms*(*1*) *assms*(*2*) ‹*py* ∈ *G*›] *dist-along-quasiconvex*[*OF assms*(*1*) *assms*(*3*) ‹*px* ∈ *G*›] **by** *auto*
  also have ... ≤ *max* (*dist x y* + *dist py px*) (*dist x px* + *dist py y*) + *2* ∗ *deltaG*(*TYPE*(*'a*))
    by (*rule hyperb-quad-ineq*)
  finally have ∗: *dist x px* + *dist y py* + *2* ∗ *dist px py*
      ≤ *max* (*dist x y* + *dist py px*) (*dist x px* + *dist py y*) + *10* ∗ *deltaG*(*TYPE*(*'a*)) + *4* ∗ *C*
    by (*auto simp add*: *metric-space-class.dist-commute*)
  show *?thesis*
  proof (*cases dist x y* + *dist py px* ≥ *dist x px* + *dist py y*)
    case *True*
    then have *dist x px* + *dist y py* + *2* ∗ *dist px py* ≤ *dist x y* + *dist py px* + *10* ∗ *deltaG*(*TYPE*(*'a*)) + *4* ∗ *C*
      using ∗ **by** *auto*
    then show *?thesis* **by** (*auto simp add*: *metric-space-class.dist-commute*)
  next
    case *False*
    then have *dist x px* + *dist y py* + *2* ∗ *dist px py* ≤ *dist x px* + *dist py y* + *10* ∗ *deltaG*(*TYPE*(*'a*)) + *4* ∗ *C*
      using ∗ **by** *auto*
    then show *?thesis* **by** (*simp add*: *metric-space-class.dist-commute*)
  qed
qed

The projection on a quasi-convex set is 1-Lipschitz up to an additive error.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *proj-along-quasiconvex-contraction'*:
  assumes *quasiconvex C G px* ∈ *proj-set x G py* ∈ *proj-set y G*
  shows *dist px py* ≤ *dist x y* + *4* ∗ *deltaG*(*TYPE*(*'a*)) + *2* ∗ *C*
proof (*cases dist y py* ≤ *dist x px*)
  case *True*

have *dist x px* + *dist px py* ≤ *dist x py* + *4* ∗ *deltaG*(*TYPE*(′*a*)) + *2* ∗ *C*
by (*rule dist-along-quasiconvex*[*OF assms*(*1*) *assms*(*2*) *proj-setD*(*1*)[*OF assms*(*3*)]])
also have ... ≤ (*dist x y* + *dist y py*) + *4* ∗ *deltaG*(*TYPE*(′*a*)) + *2* ∗ *C*
by (*intro mono-intros*)
finally show *?thesis* **using** *True* **by** *auto*
next
case *False*
have *dist y py* + *dist py px* ≤ *dist y px* + *4* ∗ *deltaG*(*TYPE*(′*a*)) + *2* ∗ *C*
by (*rule dist-along-quasiconvex*[*OF assms*(*1*) *assms*(*3*) *proj-setD*(*1*)[*OF assms*(*2*)]])
also have ... ≤ (*dist y x* + *dist x px*) + *4* ∗ *deltaG*(*TYPE*(′*a*)) + *2* ∗ *C*
by (*intro mono-intros*)
finally show *?thesis* **using** *False* **by** (*auto simp add*: *metric-space-class.dist-commute*)
qed

We can in particular specialize the previous statements to geodesics, which are 0-quasi-convex.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *dist-along-geodesic*:
assumes *geodesic-segment G p* ∈ *proj-set x G y* ∈ *G*
shows *dist x p* + *dist p y* ≤ *dist x y* + *4* ∗ *deltaG*(*TYPE*(′*a*))
using *dist-along-quasiconvex*[*OF quasiconvex-of-geodesic*[*OF assms*(*1*)] *assms*(*2*) *assms*(*3*)] **by** *auto*

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *proj-along-geodesic-contraction*:
assumes *geodesic-segment G px* ∈ *proj-set x G py* ∈ *proj-set y G*
shows *dist px py* ≤ *max* (*5* ∗ *deltaG*(*TYPE*(′*a*))) (*dist x y* − *dist px x* − *dist py y* + *10* ∗ *deltaG*(*TYPE*(′*a*)))
using *proj-along-quasiconvex-contraction*[*OF quasiconvex-of-geodesic*[*OF assms*(*1*)] *assms*(*2*) *assms*(*3*)] **by** *auto*

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *proj-along-geodesic-contraction′*:
assumes *geodesic-segment G px* ∈ *proj-set x G py* ∈ *proj-set y G*
shows *dist px py* ≤ *dist x y* + *4* ∗ *deltaG*(*TYPE*(′*a*))
using *proj-along-quasiconvex-contraction′*[*OF quasiconvex-of-geodesic*[*OF assms*(*1*)] *assms*(*2*) *assms*(*3*)] **by** *auto*

If one projects a continuous curve on a quasi-convex set, the image does not have to be connected (the projection is discontinuous), but since the projections of nearby points are within uniformly bounded distance one can find in the projection a point with almost prescribed distance to the starting point, say. For further applications, we also pick the first such point, i.e., all the previous points are also close to the starting point.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *quasi-convex-projection-small-gaps*:
assumes *continuous-on* {*a*..(*b*::*real*)} *f*
a ≤ *b*
quasiconvex C G
⋀*t*. *t* ∈ {*a*..*b*} ⟹ *p t* ∈ *proj-set* (*f t*) *G*
delta > *deltaG*(*TYPE*(′*a*))
d ∈ {*4* ∗ *delta* + *2* ∗ *C*..*dist* (*p a*) (*p b*)}

shows $\exists\, t \in \{a..b\}.\ (dist\ (p\ a)\ (p\ t) \in \{d - 4 * delta - 2 * C\ ..\ d\})$
$\wedge\ (\forall\, s \in \{a..t\}.\ dist\ (p\ a)\ (p\ s) \leq d)$
proof −
have $delta > 0$
using $assms(5)$ $local.delta\text{-}nonneg$ **by** $linarith$
moreover have $C \geq 0$
using $quasiconvexC[OF\ assms(3)]$ **by** $simp$
ultimately have $d \geq 0$ **using** $assms$ **by** $auto$

The idea is to define the desired point as the last point u for which there is a projection at distance at most d of the starting point. Then the projection can not be much closer to the starting point, or one could point another such point further away by almost continuity, giving a contradiction. The technical implementation requires some care, as the "last point" may not satisfy the property, for lack of continuity. If it does, then fine. Otherwise, one should go just a little bit to its left to find the desired point.

define I **where** $I = \{t \in \{a..b\}.\ \forall\, s \in \{a..t\}.\ dist\ (p\ a)\ (p\ s) \leq d\}$
have $a \in I$
using ‹$a \leq b$› ‹$d \geq 0$› **unfolding** $I\text{-}def$ **by** $auto$
have $bdd\text{-}above\ I$
unfolding $I\text{-}def$ **by** $auto$
define u **where** $u = Sup\ I$
have $a \leq u$
unfolding $u\text{-}def$ **apply** ($rule\ cSup\text{-}upper$) **using** ‹$a \in I$› ‹$bdd\text{-}above\ I$› **by** $auto$
have $u \leq b$
unfolding $u\text{-}def$ **apply** ($rule\ cSup\text{-}least$) **using** ‹$a \in I$› **apply** $auto$ **unfolding** $I\text{-}def$ **by** $auto$
have A: $dist\ (p\ a)\ (p\ s) \leq d$ **if** $s < u$ $a \leq s$ **for** s
proof −
have $\exists\, t{\in}I.\ s < t$
unfolding $u\text{-}def$ **apply** ($subst\ less\text{-}cSup\text{-}iff[symmetric]$)
using ‹$a \in I$› ‹$bdd\text{-}above\ I$› **using** ‹$s < u$› **unfolding** $u\text{-}def$ **by** $auto$
then obtain t **where** t: $t \in I$ $s < t$ **by** $auto$
then have $s \in \{a..t\}$ **using** ‹$a \leq s$› **by** $auto$
then show $?thesis$
using $t(1)$ **unfolding** $I\text{-}def$ **by** $auto$
qed
have $continuous\ (at\ u\ within\ \{a..b\})\ f$
using $assms(1)$ **by** ($simp\ add$: ‹$a \leq u$› ‹$u \leq b$› $continuous\text{-}on\text{-}eq\text{-}continuous\text{-}within$)
then have $\exists\, i > 0.\ \forall\, s{\in}\{a..b\}.\ dist\ u\ s < i \longrightarrow dist\ (f\ u)\ (f\ s) < (delta - deltaG(TYPE('a)))$
unfolding $continuous\text{-}within\text{-}eps\text{-}delta$ **using** ‹$deltaG(TYPE('a)) < delta$› **by** ($auto\ simp\ add$: $metric\text{-}space\text{-}class.dist\text{-}commute$)
then obtain $e0$ **where** $e0$: $e0 > 0$ $\bigwedge s.\ s \in \{a..b\} \Longrightarrow dist\ u\ s < e0 \Longrightarrow dist\ (f\ u)\ (f\ s) < (delta - deltaG(TYPE('a)))$
by $auto$

show $?thesis$

proof (*cases dist* (*p a*) (*p u*) > *d*)

First, consider the case where u does not satisfy the defining property. Then the desired point t is taken slightly to its left.

```
case True
then have u ≠ a
  using ‹d ≥ 0› by auto
then have a < u using ‹a ≤ u› by auto

define e::real where e = min (e0/2) ((u−a)/2)
then have e > 0 using ‹a < u› ‹e0 > 0› by auto
define t where t = u − e
then have t < u using ‹e > 0› by auto
have u − b ≤ e e ≤ u − a
  using ‹e > 0› ‹u ≤ b› unfolding e-def by (auto simp add: min-def)
then have t ∈ {a..b} t ∈ {a..t}
  unfolding t-def by auto
have dist u t < e0
  unfolding t-def e-def dist-real-def using ‹e0 > 0› ‹a ≤ u› by auto
have *: ∀ s ∈ {a..t}. dist (p a) (p s) ≤ d
  using A ‹t < u› by auto
have dist (p t) (p u) ≤ dist (f t) (f u) + 4 ∗ deltaG(TYPE('a)) + 2 ∗ C
  apply (rule proj-along-quasiconvex-contraction'[OF ‹quasiconvex C G›])
  using assms (4) ‹t ∈ {a..b}› ‹a ≤ u› ‹u ≤ b› by auto
also have ... ≤ (delta − deltaG(TYPE('a))) + 4 ∗ deltaG(TYPE('a)) + 2 ∗ C
  apply (intro mono-intros)
    using e0(2)[OF ‹t ∈ {a..b}› ‹dist u t < e0›] by (auto simp add: metric-space-class.dist-commute)
finally have I: dist (p t) (p u) ≤ 4 ∗ delta + 2 ∗ C
  using ‹delta > deltaG(TYPE('a))› by simp

have d ≤ dist (p a) (p u)
  using True by auto
also have ... ≤ dist (p a) (p t) + dist (p t) (p u)
  by (intro mono-intros)
also have ... ≤ dist (p a) (p t) + 4 ∗ delta + 2 ∗ C
  using I by simp
finally have **: d − 4 ∗ delta − 2 ∗ C ≤ dist (p a) (p t)
  by simp
show ?thesis
  apply (rule bexI[OF - ‹t ∈ {a..b}›]) using ∗ ∗∗ ‹t ∈ {a..b}› by auto
next
```

Next, consider the case where u satisfies the defining property. Then we will take $t = u$. The only nontrivial point to check is that the distance of $f(u)$ to the starting point is not too small. For this, we need to separate the case where $u = b$ (in which case one argues directly) and the case where $u < b$, where one can use a point slightly to the right of u which has a projection at distance $> d$ of the starting point, and use almost continuity.

case *False*
have *B*: *dist* (*p a*) (*p s*) $\leq$ *d* **if** $s \in \{a..u\}$ **for** *s*
proof (*cases s* = *u*)
case *True*
show *?thesis*
unfolding *True* **using** *False* **by** *auto*
next
case *False*
then show *?thesis*
using *that A* **by** *auto*
qed
have *C*: *dist* (*p a*) (*p u*) $\geq d - 4 * delta - 2 * C$
proof (*cases u* = *b*)
case *True*
have $d \leq$ *dist* (*p a*) (*p b*)
using *assms* **by** *auto*
also have ... $\leq$ *dist* (*p a*) (*p u*) + *dist* (*p u*) (*p b*)
by (*intro mono-intros*)
also have ... $\leq$ *dist* (*p a*) (*p u*) + (*dist* (*f u*) (*f b*) + *4* ∗ *deltaG TYPE*(′*a*) + *2* ∗ *C*)
apply (*intro mono-intros proj-along-quasiconvex-contraction*′[*OF* ‹*quasiconvex C G*›])
using *assms* ‹$a \leq u$› ‹$u \leq b$› **by** *auto*
finally show *?thesis*
unfolding *True* **using** ‹*deltaG*(*TYPE*(′*a*)) < *delta*› **by** *auto*
next
case *False*
then have $u < b$
using ‹$u \leq b$› **by** *auto*
define *e*::*real* **where** *e* = *min* (*e0*/*2*) ((*b*−*u*)/*2*)
then have $e > 0$ **using** ‹$u < b$› ‹$e0 > 0$› **by** *auto*
define *v* **where** $v = u + e$
then have $u < v$
using ‹$e > 0$› **by** *auto*
have $e \leq b - u$ $a - u \leq e$
using ‹$e > 0$› ‹$a \leq u$› **unfolding** *e-def* **by** (*auto simp add*: *min-def*)
then have $v \in \{a..b\}$
unfolding *v-def* **by** *auto*
moreover have $v \notin I$
using ‹$u < v$› ‹*bdd-above I*› *cSup-upper not-le* **unfolding** *u-def* **by** *auto*
ultimately have $\exists w \in \{a..v\}$. *dist* (*p a*) (*p w*) $> d$
unfolding *I-def* **by** *force*
then obtain *w* **where** *w*: $w \in \{a..v\}$ *dist* (*p a*) (*p w*) $> d$
by *auto*
then have $w \notin \{a..u\}$
using *B* **by** *force*
then have $u < w$
using *w*(*1*) **by** *auto*

have *w* ∈ {*a*..*b*}
using *w*(*1*) ‹*v* ∈ {*a*..*b*}› **by** *auto*
have *dist u w* = *w* − *u*
unfolding *dist-real-def* **using** ‹*u* < *w*› **by** *auto*
also have ... ≤ *v* − *u*
using *w*(*1*) **by** *auto*
also have ... < *e0*
unfolding *v-def e-def min-def* **using** ‹*e0* > *0*› **by** *auto*
finally have *dist u w* < *e0* **by** *simp*
have *dist* (*p u*) (*p w*) ≤ *dist* (*f u*) (*f w*) + *4* ∗ *deltaG*(*TYPE*(*'a*)) + *2* ∗ *C*
apply (*rule proj-along-quasiconvex-contraction'*[*OF* ‹*quasiconvex C G*›])
using *assms* ‹*a* ≤ *u*› ‹*u* ≤ *b*› ‹*w* ∈ {*a*..*b*}› **by** *auto*
also have ... ≤ (*delta* − *deltaG*(*TYPE*(*'a*))) + *4* ∗ *deltaG*(*TYPE*(*'a*)) + *2* ∗ *C*
apply (*intro mono-intros*)
using *e0*(*2*)[*OF* ‹*w* ∈ {*a*..*b*}› ‹*dist u w* < *e0*›] **by** (*auto simp add*: *metric-space-class.dist-commute*)
finally have *I*: *dist* (*p u*) (*p w*) ≤ *4* ∗ *delta* + *2* ∗ *C*
using ‹*delta* > *deltaG*(*TYPE*(*'a*))› **by** *simp*
have *d* ≤ *dist* (*p a*) (*p u*) + *dist* (*p u*) (*p w*)
using *w*(*2*) *metric-space-class.dist-triangle*[*of p a p w p u*] **by** *auto*
also have ... ≤ *dist* (*p a*) (*p u*) + *4* ∗ *delta* + *2* ∗ *C*
using *I* **by** *auto*
finally show *?thesis* **by** *simp*
qed
show *?thesis*
apply (*rule bexI*[*of* - *u*])
using *B* ‹*a* ≤ *u*› ‹*u* ≤ *b*› *C* **by** *auto*
qed
qed

Same lemma, except that one exchanges the roles of the beginning and the end point.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *quasi-convex-projection-small-gaps'*:
assumes *continuous-on* {*a*..(*b*::*real*)} *f*
a ≤ *b*
quasiconvex C G
⋀*x*. *x* ∈ {*a*..*b*} ⟹ *p x* ∈ *proj-set* (*f x*) *G*
delta > *deltaG*(*TYPE*(*'a*))
d ∈ {*4* ∗ *delta* + *2* ∗ *C*..*dist* (*p a*) (*p b*)}
shows ∃ *t* ∈ {*a*..*b*}. *dist* (*p b*) (*p t*) ∈ {*d* − *4* ∗ *delta* − *2* ∗ *C* .. *d*}
∧ (∀ *s* ∈ {*t*..*b*}. *dist* (*p b*) (*p s*) ≤ *d*)
proof −
have ∗: *continuous-on* {−*b*..−*a*} (λ*t*. *f*(−*t*))
using *continuous-on-compose*[*of* {−*b*..−*a*} λ*t*. −*t f*] **using** *assms*(*1*) *continuous-on-minus*[*OF continuous-on-id*] **by** *auto*
define *q* **where** *q* = (λ*t*. *p*(−*t*))
have ∃ *t* ∈ {−*b*..−*a*}. (*dist* (*q* (−*b*)) (*q t*) ∈ {*d* − *4* ∗ *delta* − *2* ∗ *C* .. *d*})
∧ (∀ *s* ∈ {−*b*..*t*}. *dist* (*q* (−*b*)) (*q s*) ≤ *d*)

apply (*rule quasi-convex-projection-small-gaps*[**where** *?f* = $\lambda t.$ $f(-t)$ **and** *?G* = *G*])
unfolding *q-def* **using** *assms* $*$ **by** (*auto simp add*: *metric-space-class.dist-commute*)
then obtain *t* **where** *t*: $t \in \{-b..-a\}$ *dist* (*q* $(-b)$) (*q t*) $\in \{d - 4 * delta - 2 * C\ ..\ d\}$

$$\bigwedge s.\ s \in \{-b..t\} \Longrightarrow dist\ (q\ (-b))\ (q\ s) \leq d$$

by *blast*
have $*$: *dist* (*p b*) (*p s*) $\leq d$ **if** $s \in \{-t..b\}$ **for** *s*
using *t*(*3*)[*of* $-s$] *that q-def* **by** *auto*
show *?thesis*
apply (*rule bexI*[*of* - $-t$]) **using** *t* $*$ *q-def* **by** *auto*
qed

11 The Morse-Gromov Theorem

The goal of this section is to prove a central basic result in the theory of hyperbolic spaces, usually called the Morse Lemma. It is really a theorem, and we add the name Gromov the avoid the confusion with the other Morse lemma on the existence of good coordinates for C^2 functions with non-vanishing hessian.

It states that a quasi-geodesic remains within bounded distance of a geodesic with the same endpoints, the error depending only on δ and on the parameters (λ, C) of the quasi-geodesic, but not on its length.

There are several proofs of this result. We will follow the one of Shchur [Shc13], which gets an optimal dependency in terms of the parameters of the quasi-isometry, contrary to all previous proofs. The price to pay is that the proof is more involved (relying in particular on the fact that the closest point projection on quasi-convex sets is exponentially contracting).

We will also give afterwards for completeness the proof in [BH99], as it brings up interesting tools, although the dependency it gives is worse.

The next lemma (for $C = 0$, Lemma 2 in [Shc13]) asserts that, if two points are not too far apart (at distance at most 10δ), and far enough from a given geodesic segment, then when one moves towards this geodesic segment by a fixed amount (here 5δ), then the two points become closer (the new distance is at most 5δ, gaining a factor of 2). Later, we will iterate this lemma to show that the projection on a geodesic segment is exponentially contracting. For the application, we give a more general version involving an additional constant C.

This lemma holds for δ the hyperbolicity constant. We will want to apply it with $\delta > 0$, so to avoid problems in the case $\delta = 0$ we formulate it not using the hyperbolicity constant of the given type, but any constant which is at least the hyperbolicity constant (this is to work around the fact that one can not say or use easily in Isabelle that a type with hyperbolicity δ is also hyperbolic for any larger constant δ'.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *geodesic-projection-exp-contracting-aux*:
assumes *geodesic-segment G*
px ∈ *proj-set x G*
py ∈ *proj-set y G*
delta ≥ *deltaG*(*TYPE*(*'a*))
dist x y ≤ *10* ∗ *delta* + *C*
M ≥ *15/2* ∗ *delta*
dist px x ≥ *M* + *5* ∗ *delta* + *C/2*
dist py y ≥ *M* + *5* ∗ *delta* + *C/2*
C ≥ *0*
shows *dist* (*geodesic-segment-param* {*px*−−*x*} *px M*)
(*geodesic-segment-param* {*py*−−*y*} *py M*) ≤ *5* ∗ *delta*
proof −
have *dist px x* ≤ *dist py x*
using *proj-setD*(*2*)[*OF assms*(*2*)] *infdist-le*[*OF proj-setD*(*1*)[*OF assms*(*3*)], *of x*] **by** (*simp add*: *metric-space-class.dist-commute*)
have *dist py y* ≤ *dist px y*
using *proj-setD*(*2*)[*OF assms*(*3*)] *infdist-le*[*OF proj-setD*(*1*)[*OF assms*(*2*)], *of y*] **by** (*simp add*: *metric-space-class.dist-commute*)

have *delta* ≥ *0*
using *assms local.delta-nonneg* **by** *linarith*
then have *M*: *M* ≥ *0 M* ≤ *dist px x M* ≤ *dist px y M* ≤ *dist py x M* ≤ *dist py y*
using *assms* ‹*dist px x* ≤ *dist py x*› ‹*dist py y* ≤ *dist px y* ›**by** *auto*
have *px* ∈ *G py* ∈ *G*
using *assms proj-setD* **by** *auto*

define *x'* **where** *x'* = *geodesic-segment-param* {*px*−−*x*} *px M*
define *y'* **where** *y'* = *geodesic-segment-param* {*py*−−*y*} *py M*

First step: the distance between *px* and *py* is at most 5δ.

have *dist px py* ≤ *max* (*5* ∗ *deltaG*(*TYPE*(*'a*))) (*dist x y* − *dist px x* − *dist py y* + *10* ∗ *deltaG*(*TYPE*(*'a*)))
by (*rule proj-along-geodesic-contraction*[*OF assms*(*1*) *assms*(*2*) *assms*(*3*)])
also have ... ≤ *max* (*5* ∗ *deltaG*(*TYPE*(*'a*))) (*5* ∗ *deltaG*(*TYPE*(*'a*)))
apply (*intro mono-intros*) **using** *assms* ‹*delta* ≥ *0*› **by** *auto*
finally have *dist px py* ≤ *5* ∗ *delta*
using ‹*delta* ≥ *deltaG*(*TYPE*(*'a*))› **by** *auto*

Second step: show that all the interesting Gromov products at bounded below by *M*.

have ∗: *x'* ∈ {*px*−−*x*} **unfolding** *x'-def*
by (*simp add*: *geodesic-segment-param-in-segment*)
have *px* ∈ *proj-set x' G*
by (*rule proj-set-geodesic-same-basepoint*[*OF* ‹*px* ∈ *proj-set x G*› - ∗], *auto*)
have *dist px x'* = *M*

unfolding *x′-def* **using** *M* **by** *auto*
have *dist px x′ ≤ dist py x′*
using *proj-setD(2)[OF ‹px ∈ proj-set x′ G›] infdist-le[OF proj-setD(1)[OF assms(3)], of x′]* **by** (*simp add: metric-space-class.dist-commute*)
have ∗∗: *dist px x = dist px x′ + dist x′ x*
using *geodesic-segment-dist[OF - ∗, of px x]* **by** *auto*
have *Ixx*: *Gromov-product-at px x′ x = M*
unfolding *Gromov-product-at-def ∗∗ x′-def* **using** *M* **by** *auto*
have *2 ∗ M = dist px x′ + dist px x − dist x′ x*
unfolding *∗∗ x′-def* **using** *M* **by** *auto*
also have ... *≤ dist py x′ + dist py x − dist x′ x*
apply (*intro mono-intros, auto*) **by** *fact+*
also have ... *= 2 ∗ Gromov-product-at py x x′*
unfolding *Gromov-product-at-def* **by** (*auto simp add: metric-space-class.dist-commute*)
finally have *Iyx*: *Gromov-product-at py x x′ ≥ M* **by** *auto*

have ∗: *y′ ∈ {py−−y}* **unfolding** *y′-def*
by (*simp add: geodesic-segment-param-in-segment*)
have *py ∈ proj-set y′ G*
by (*rule proj-set-geodesic-same-basepoint[OF ‹py ∈ proj-set y G› - ∗], auto*)
have *dist py y′ = M*
unfolding *y′-def* **using** *M* **by** *auto*
have *dist py y′ ≤ dist px y′*
using *proj-setD(2)[OF ‹py ∈ proj-set y′ G›] infdist-le[OF proj-setD(1)[OF assms(2)], of y′]* **by** (*simp add: metric-space-class.dist-commute*)
have ∗∗: *dist py y = dist py y′ + dist y′ y*
using *geodesic-segment-dist[OF - ∗, of py y]* **by** *auto*
have *Iyy*: *Gromov-product-at py y′ y = M*
unfolding *Gromov-product-at-def ∗∗ y′-def* **using** *M* **by** *auto*
have *2 ∗ M = dist py y′ + dist py y − dist y′ y*
unfolding *∗∗ y′-def* **using** *M* **by** *auto*
also have ... *≤ dist px y′ + dist px y − dist y′ y*
apply (*intro mono-intros, auto*) **by** *fact+*
also have ... *= 2 ∗ Gromov-product-at px y y′*
unfolding *Gromov-product-at-def* **by** (*auto simp add: metric-space-class.dist-commute*)
finally have *Ixy*: *Gromov-product-at px y y′ ≥ M* **by** *auto*

have *2 ∗ M ≤ dist px x + dist py y − dist x y*
using *assms* **by** *auto*
also have ... *≤ dist px x + dist px y − dist x y*
by (*intro mono-intros, fact*)
also have ... *= 2 ∗ Gromov-product-at px x y*
unfolding *Gromov-product-at-def* **by** *auto*
finally have *Ix*: *Gromov-product-at px x y ≥ M*
by *auto*
have *2 ∗ M ≤ dist px x + dist py y − dist x y*
using *assms* **by** *auto*
also have ... *≤ dist py x + dist py y − dist x y*
by (*intro mono-intros, fact*)

also have ... $= 2 * $ *Gromov-product-at py x y*
unfolding *Gromov-product-at-def* **by** *auto*
finally have *Iy*: *Gromov-product-at py x y* $\geq M$
by *auto*

Third step: prove the estimate

have $M - 2 * delta \leq Min$ {*Gromov-product-at px x' x*, *Gromov-product-at px x y*, *Gromov-product-at px y y'*} $- 2 * deltaG(TYPE('a))$
using *Ixx Ixy Ix* ‹$delta \geq deltaG(TYPE('a))$› **by** *auto*
also have ... $\leq$ *Gromov-product-at px x' y'*
by (*intro mono-intros*)
finally have *A*: $M - 4 * delta + dist\ x'\ y' \leq dist\ px\ y'$
unfolding *Gromov-product-at-def* ‹$dist\ px\ x' = M$› **by** *auto*

have $M - 2 * delta \leq Min$ {*Gromov-product-at py x' x*, *Gromov-product-at py x y*, *Gromov-product-at py y y'*} $- 2 * deltaG(TYPE('a))$
using *Iyx Iyy Iy* ‹$delta \geq deltaG(TYPE('a))$› **by** (*auto simp add*: *Gromov-product-commute*)
also have ... $\leq$ *Gromov-product-at py x' y'*
by (*intro mono-intros*)
finally have *B*: $M - 4 * delta + dist\ x'\ y' \leq dist\ py\ x'$
unfolding *Gromov-product-at-def* ‹$dist\ py\ y' = M$› **by** *auto*

have $dist\ px\ py \leq 2 * M - 10 * delta$
using *assms* ‹$dist\ px\ py \leq 5 * delta$› **by** *auto*
have $2 * M - 8 * delta + 2 * dist\ x'\ y' \leq dist\ px\ y' + dist\ py\ x'$
using *A B* **by** *auto*
also have ... $\leq max\ (dist\ px\ py + dist\ y'\ x')\ (dist\ px\ x' + dist\ y'\ py) + 2 * deltaG\ TYPE('a)$
by (*rule hyperb-quad-ineq*)
also have ... $\leq max\ (dist\ px\ py + dist\ y'\ x')\ (dist\ px\ x' + dist\ y'\ py) + 2 * delta$
using ‹$deltaG(TYPE('a)) \leq delta$› **by** *auto*
finally have $2 * M - 10 * delta + 2 * dist\ x'\ y' \leq max\ (dist\ px\ py + dist\ y'\ x')\ (dist\ px\ x' + dist\ y'\ py)$
by *auto*
then have $2 * M - 10 * delta + 2 * dist\ x'\ y' \leq dist\ px\ x' + dist\ py\ y'$
apply (*auto simp add*: *metric-space-class.dist-commute*)
using ‹$0 \leq delta$› ‹$dist\ px\ py \leq 2 * M - 10 * delta$› ‹$dist\ px\ x' = M$› ‹$dist\ py\ y' = M$› **by** *auto*
then have $dist\ x'\ y' \leq 5 * delta$
unfolding ‹$dist\ px\ x' = M$› ‹$dist\ py\ y' = M$› **by** *auto*
then show *?thesis*
unfolding *x'-def y'-def* **by** *auto*
qed

The next lemma (Lemma 10 in [Shc13] for $C = 0$) asserts that the projection on a geodesic segment is an exponential contraction. More precisely, if a path of length L is at distance at least D of a geodesic segment G, then the projection of the path on G has diameter at most $CL\exp(-cD/\delta)$, where C and c are universal constants. This is not completely true at one can not go below a fixed size, as always, so the correct bound is $K \max(\delta, L \exp(-cD/\delta))$. For the application, we give a slightly more general statement involving an additional constant C.

This statement follows from the previous lemma: if one moves towards G by 10δ, then the distance between points is divided by 2. Then one iterates this statement as many times as possible, gaining a factor 2 each time and therefore an exponential factor in the end.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *geodesic-projection-exp-contracting*:
assumes *geodesic-segment G*
$\bigwedge x\ y.\ x \in \{a..b\} \Longrightarrow y \in \{a..b\} \Longrightarrow dist\ (f\ x)\ (f\ y) \le lambda * dist\ x\ y + C$
$a \le b$
$pa \in proj\text{-}set\ (f\ a)\ G$
$pb \in proj\text{-}set\ (f\ b)\ G$
$\bigwedge t.\ t \in \{a..b\} \Longrightarrow infdist\ (f\ t)\ G \ge D$
$D \ge 15/2 * delta + C/2$
$delta > deltaG(TYPE('a))$
$C \ge 0$
$lambda \ge 0$
shows $dist\ pa\ pb \le max\ (5 * deltaG(TYPE('a)))\ ((4 * exp(1/2 * ln\ 2)) * lambda * (b-a) * exp(-(D-C/2) * ln\ 2\ /\ (5 * delta)))$
proof −
have $delta > 0$ **using** *assms*
using *local.delta-nonneg* **by** *linarith*
have $exp(15/2/5 * ln\ 2) = exp(ln\ 2) * exp(1/2 * ln\ (2::real))$
unfolding *mult-exp-exp* **by** *simp*
also have ... $= 2 * exp(1/2 * ln\ 2)$
by *auto*
finally have $exp(15/2/5 * ln\ 2) = 2 * exp(1/2 * ln\ (2::real))$
by *simp*

The idea of the proof is to start with a sequence of points separated by $10\delta + C$ along the original path, and push them by a fixed distance towards G to bring them at distance at most 5δ, thanks to the previous lemma. Then, discard half the points, and start again. This is possible while one is far enough from G. In the first step of the proof, we formalize this in the case where the process can be iterated long enough that, at the end, the projections on G are very close together. This is a simple induction, based on the previous lemma.

have *Main*: $\bigwedge c\ g\ p.\ (\forall i \in \{0..2\hat{\ }k\}.\ p\ i \in proj\text{-}set\ (g\ i)\ G)$
$\Longrightarrow (\forall i \in \{0..2\hat{\ }k\}.\ dist\ (p\ i)\ (g\ i) \ge 5 * delta * k + 15/2 * delta + c/2)$
$\Longrightarrow (\forall i \in \{0..<2\hat{\ }k\}.\ dist\ (g\ i)\ (g\ (Suc\ i)) \le 10 * delta + c)$
$\Longrightarrow c \ge 0$
$\Longrightarrow dist\ (p\ 0)\ (p\ (2\hat{\ }k)) \le 5 * deltaG(TYPE('a))$ **for** k
proof (*induction k*)
case 0

then have *H*: *p 0* ∈ *proj-set* (*g 0*) *G*
p 1 ∈ *proj-set* (*g 1*) *G*
dist (*g 0*) (*g 1*) ≤ *10* ∗ *delta* + *c*
dist (*p 0*) (*g 0*) ≥ *15*/*2* ∗ *delta* + *c*/*2*
dist (*p 1*) (*g 1*) ≥ *15*/*2* ∗ *delta* + *c*/*2*
by *auto*
have *dist* (*p 0*) (*p 1*) ≤ *max* (*5* ∗ *deltaG*(*TYPE*(′*a*))) (*dist* (*g 0*) (*g 1*) − *dist* (*p 0*) (*g 0*) − *dist* (*p 1*) (*g 1*) + *10* ∗ *deltaG*(*TYPE*(′*a*)))
by (*rule proj-along-geodesic-contraction*[*OF* ‹*geodesic-segment G*› ‹*p 0* ∈ *proj-set* (*g 0*) *G*› ‹*p 1* ∈ *proj-set* (*g 1*) *G*›])
also have ... ≤ *max* (*5* ∗ *deltaG*(*TYPE*(′*a*))) (*5* ∗ *deltaG*(*TYPE*(′*a*)))
apply (*intro mono-intros*) **using** *H* ‹*delta* > *deltaG*(*TYPE*(′*a*))› **by** *auto*
finally show *dist* (*p 0*) (*p* (*2*^*0*)) ≤ *5* ∗ *deltaG*(*TYPE*(′*a*))
by *auto*
next
case (*Suc k*)
have ∗: *5* ∗ *delta* ∗ *real* (*k* + *1*) + *5* ∗ *delta* = *5* ∗ *delta* ∗ *real* (*Suc k* + *1*)
by (*simp add*: *algebra-simps*)
define *h* **where** *h* = (λ*i*. *geodesic-segment-param* {*p i*−−*g i*} (*p i*) (*5* ∗ *delta* ∗ *k* + *15*/*2* ∗ *delta*))
have *h-dist*: *dist* (*h i*) (*h* (*Suc i*)) ≤ *5* ∗ *delta* **if** *i* ∈ {*0*..<*2*^(*Suc k*)} **for** *i*
unfolding *h-def* **apply** (*rule geodesic-projection-exp-contracting-aux*[*OF* ‹*geodesic-segment G*› - - *less-imp-le*[*OF* ‹*delta* > *deltaG*(*TYPE*(′*a*))›]])
unfolding ∗ **using** *Suc.prems that* ‹*delta* > *0*› **by** (*auto simp add*: *algebra-simps divide-simps*)
define *g*′ **where** *g*′ = (λ*i*. *h* (*2* ∗ *i*))
define *p*′ **where** *p*′ = (λ*i*. *p* (*2* ∗ *i*))
have *dist* (*p*′ *0*) (*p*′ (*2*^*k*)) ≤ *5* ∗ *deltaG*(*TYPE*(′*a*))
proof (*rule Suc.IH*[**where** *?g* = *g*′ **and** *?c* = *0*])
show ∀ *i*∈{*0*..*2* ^ *k*}. *p*′ *i* ∈ *proj-set* (*g*′ *i*) *G*
proof
fix *i*::*nat* **assume** *i* ∈ {*0*..*2*^*k*}
then have ∗: *2* ∗ *i* ∈ {*0*..*2*^(*Suc k*)} **by** *auto*
show *p*′ *i* ∈ *proj-set* (*g*′ *i*) *G*
unfolding *p*′*-def g*′*-def h-def* **apply** (*rule proj-set-geodesic-same-basepoint*[*of* - *g* (*2* ∗ *i*) - {*p*(*2* ∗ *i*)−−*g*(*2* ∗ *i*)}])
using *Suc* ∗ **by** (*auto simp add*: *geodesic-segment-param-in-segment*)
qed
show ∀ *i*∈{*0*..*2* ^ *k*}. *5* ∗ *delta* ∗ *k* + *15*/*2* ∗ *delta* + *0*/*2* ≤ *dist* (*p*′ *i*) (*g*′ *i*)
proof
fix *i*::*nat* **assume** *i* ∈ {*0*..*2*^*k*}
then have ∗: *2* ∗ *i* ∈ {*0*..*2*^(*Suc k*)} **by** *auto*
have *5* ∗ *delta* ∗ *k* + *15*/*2* ∗ *delta* ≤ *5* ∗ *delta* ∗ *Suc k* + *15*/*2* ∗ *delta* + *c*/*2*
using ‹*delta* > *0*› ‹*c* ≥ *0*› **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... ≤ *dist* (*p* (*2* ∗ *i*)) (*g* (*2* ∗ *i*))
using *Suc* ∗ **by** *auto*
finally have ∗: *5* ∗ *delta* ∗ *k* + *15*/*2* ∗ *delta* ≤ *dist* (*p* (*2* ∗ *i*)) (*g* (*2* ∗ *i*)) **by** *simp*

have *dist* $(p'\ i)\ (g'\ i) = 5 * delta * k + 15/2 * delta$
unfolding *p′-def g′-def h-def* **apply** (*rule geodesic-segment-param-in-geodesic-spaces*(*6*))
using $*$ ‹*delta* > *0*› **by** *auto*
then show $5 * delta * k + 15/2 * delta + 0/2 \leq dist\ (p'\ i)\ (g'\ i)$ **by** *simp*
qed
show $\forall i \in \{0..<2 \,\hat{}\, k\}.\ dist\ (g'\ i)\ (g'\ (Suc\ i)) \leq 10 * delta + 0$
proof
fix *i*::*nat* **assume** $*$: $i \in \{0..<2 \,\hat{}\, k\}$
have $dist\ (g'\ i)\ (g'\ (Suc\ i)) = dist\ (h\ (2 * i))\ (h\ (Suc\ (Suc\ (2 * i))))$
unfolding *g′-def* **by** *auto*
also have ... $\leq dist\ (h\ (2 * i))\ (h\ (Suc\ (2 * i))) + dist\ (h\ (Suc\ (2 * i)))$ $(h\ (Suc\ (Suc\ (2 * i))))$
by (*intro mono-intros*)
also have ... $\leq 5 * delta + 5 * delta$
apply (*intro mono-intros h-dist*) **using** $*$ **by** *auto*
finally show $dist\ (g'\ i)\ (g'\ (Suc\ i)) \leq 10 * delta + 0$ **by** *simp*
qed
qed (*simp*)
then show $dist\ (p\ 0)\ (p\ (2 \,\hat{}\, Suc\ k)) \leq 5 * deltaG(TYPE('a))$
unfolding *p′-def* **by** *auto*
qed

Now, we will apply the previous basic statement to points along our original path. We introduce k, the number of steps for which the pushing process can be done – it only depends on the original distance D to G.

define *k* **where** $k = nat(floor((D - C/2 - 15/2 * delta)/(5 * delta)))$
have $int\ k = floor((D - C/2 - 15/2 * delta)/(5 * delta))$
unfolding *k-def* **apply** (*rule nat-0-le*) **using** ‹$D \geq 15/2 * delta + C/2$› ‹$delta > 0$› **by** *auto*
then have $k \leq (D - C/2 - 15/2 * delta)/(5 * delta)$ $(D - C/2 - 15/2 * delta)/(5 * delta) \leq k + 1$
by *linarith+*
then have *k*: $D \geq 5 * delta * k + 15/2 * delta + C/2$ $D \leq 5 * delta * (k+1) + 15/2 * delta + C/2$
using ‹$delta > 0$› **by** (*auto simp add*: *algebra-simps divide-simps*)
have $exp((D-C/2)/(5 * delta) * ln\ 2) * exp(-15/2/5 * ln\ 2) = exp(((D-C/2-15/2 * delta)/(5 * delta)) * ln\ 2)$
unfolding *mult-exp-exp* **using** ‹$delta > 0$› **by** (*simp add*: *algebra-simps divide-simps*)
also have ... $\leq exp((k+1) * ln\ 2)$
apply (*intro mono-intros*) **using** *k*(*2*) ‹$delta > 0$› **by** (*auto simp add*: *divide-simps algebra-simps*)
also have ... $= 2\hat{}(k+1)$
by (*subst powr-realpow*[*symmetric*], *auto simp add*: *powr-def*)
also have ... $= 2 * 2\hat{}k$
by *auto*
finally have *k′*: $1/2\hat{}k \leq 2 * exp(15/2/5 * ln\ 2) * exp(-\ ((D-C/2) * ln\ 2\ /\ (5 * delta)))$
by (*auto simp add*: *algebra-simps divide-simps exp-minus*)

We separate the proof into two cases. If the path is not too long, then it can be covered by 2^k points at distance at most $10\delta + C$. By the basic statement, it follows that the diameter of the projection is at most 5δ. Otherwise, we subdivide the path into 2^N points at distance at most $10\delta + C$, with $N \geq k$, and apply the basic statement to blocks of 2^k consecutive points. It follows that the projections of $g_0, g_{2^k}, g_{2 \cdot 2^k}, \ldots$ are at distances at most 5δ. Hence, the first and last projections are at distance at most $2^{N-k} \cdot 5\delta$, which is the desired bound.

show *?thesis*
proof (*cases lambda* $*$ (*b*−*a*) ≤ *10* $*$ *delta* $*$ *2^k*)

First, treat the case where the path is rather short.

case *True*
define *g*::*nat* ⇒ *'a* **where** *g* = (λ*i*. *f*(*a* + (*b*−*a*) $*$ *i*/*2^k*))
have *g 0* = *f a g*(*2^k*) = *f b*
unfolding *g-def* **by** *auto*
have $*$: *a* + (*b*−*a*) $*$ *i*/*2^k* ∈ {*a*..*b*} **if** *i* ∈ {*0*..*2^k*} **for** *i*::*nat*
proof −
have *a* + (*b* − *a*) $*$ (*real i* / *2* ^ *k*) ≤ *a* + (*b*−*a*) $*$ (*2^k*/*2^k*)
apply (*intro mono-intros*) **using** *that* ‹*a* ≤ *b*› **by** *auto*
then show *?thesis* **using** ‹*a* ≤ *b*› **by** *auto*
qed
have *A*: *dist* (*g i*) (*g* (*Suc i*)) ≤ *10* $*$ *delta* + *C* **if** *i* ∈ {*0*..<*2^k*} **for** *i*
proof −
have *dist* (*g i*) (*g* (*Suc i*)) ≤ *lambda* $*$ *dist* (*a* + (*b*−*a*) $*$ *i*/*2^k*) (*a* + (*b*−*a*) $*$ (*Suc i*)/*2^k*) + *C*
unfolding *g-def* **apply** (*intro assms*(*2*) $*$) **using** *that* **by** *auto*
also have ... = *lambda* $*$ (*b*−*a*)/*2^k* + *C*
unfolding *dist-real-def* **using** ‹*a* ≤ *b*› **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... ≤ *10* $*$ *delta* + *C*
using *True* **by** (*simp add*: *divide-simps algebra-simps*)
finally show *?thesis* **by** *simp*
qed
define *p* **where** *p* = (λ*i*. *if i* = *0 then pa else if i* = *2^k then pb else SOME p*. *p* ∈ *proj-set* (*g i*) *G*)
have *B*: *p i* ∈ *proj-set* (*g i*) *G* **if** *i* ∈ {*0*..*2^k*} **for** *i*
proof (*cases i* = *0* ∨ *i* = *2^k*)
case *True*
then show *?thesis*
using ‹*pa* ∈ *proj-set* (*f a*) *G*› ‹*pb* ∈ *proj-set* (*f b*) *G*› **unfolding** *p-def g-def* **by** *auto*
next
case *False*
then have *p i* = (*SOME p*. *p* ∈ *proj-set* (*g i*) *G*)
unfolding *p-def* **by** *auto*
moreover have *proj-set* (*g i*) *G* ≠ {}
apply (*rule proj-set-nonempty-of-proper*) **using** *geodesic-segment-topology*[*OF*

‹geodesic-segment G›] **by** *auto*
ultimately show *?thesis*
using *some-in-eq* **by** *auto*
qed
have *C*: *dist (p i) (g i)* $\geq$ *5 * delta * k + 15/2 * delta + C/2* **if** *i* $\in$ *{0..2^k}* **for** *i*
proof −
have *5 * delta * k + 15/2 * delta + C/2* $\leq$ *D*
using *k(1)* **by** *simp*
also have ... $\leq$ *infdist (g i) G*
unfolding *g-def* **apply** (*rule* ‹$\bigwedge$*t. t* $\in$ *{a..b}* $\Longrightarrow$ *infdist (f t) G* $\geq$ *D*›) **using** * *that* **by** *auto*
also have ... = *dist (p i) (g i)*
using *that proj-setD(2)[OF B[OF that]]* **by** (*simp add*: *metric-space-class.dist-commute*)
finally show *?thesis* **by** *simp*
qed
have *dist (p 0) (p (2^k))* $\leq$ *5 * deltaG(TYPE('a))*
apply (*rule Main*[**where** *?g = g* **and** *?c = C*]) **using** *A B C* ‹*C* $\geq$ *0*› **by** *auto*
then show *?thesis*
unfolding *p-def* **by** *auto*
next

Now, the case where the path is long. We introduce N such that it is roughly of length $2^N \cdot 10\delta$.

case *False*
have **: *10 * delta * 2^k* $\leq$ *lambda * (b−a)* **using** *False* **by** *simp*
have *lambda * (b−a) > 0*
using ‹*delta > 0*› *False* ‹*0* $\leq$ *lambda*› *assms(3) less-eq-real-def mult-le-0-iff* **by** *auto*
then have *a < b lambda > 0*
using ‹*a* $\leq$ *b*› ‹*lambda* $\geq$ *0*› *less-eq-real-def* **by** *auto*
define *n* **where** *n = nat(floor(log 2 (lambda * (b−a)/(10 * delta))))*
have *log 2 (lambda * (b−a)/(10 * delta))* $\geq$ *log 2 (2^k)*
apply (*subst log-le-cancel-iff*)
using * ‹*delta > 0*› ‹*a < b*› ‹*lambda > 0*› **by** (*auto simp add*: *divide-simps algebra-simps*)
moreover have *log 2 (2^k) = k*
by *simp*
ultimately have *A*: *log 2 (lambda * (b−a)/(10 * delta))* $\geq$ *k* **by** *auto*
have ***: *int n = floor(log 2 (lambda * (b−a)/(10 * delta)))*
unfolding *n-def* **apply** (*rule nat-0-le*) **using** *A* **by** *auto*
then have *log 2 (2^n)* $\leq$ *log 2 (lambda * (b−a)/(10 * delta))*
apply (*subst log-nat-power*, *auto*) **by** *linarith*
then have *I*: *2^n* $\leq$ *lambda * (b−a)/(10 * delta)*
using ‹*0 < lambda * (b − a)*› ‹*0 < delta*›
by (*simp add*: *le-log-iff powr-realpow*)
have *log 2 (lambda * (b−a)/(10 * delta))* $\leq$ *log 2 (2^(n+1))*
apply (*subst log-nat-power*, *auto*) **using** ** **by** *linarith*

then have J: $lambda * (b-a)/(10 * delta) \leq 2\hat{}(n+1)$
using ‹$0 < lambda * (b - a)$› ‹$0 < delta$› **by** *auto*
have K: $k \leq n$ **using** A ** **by** *linarith*
define N **where** $N = n+1$
have N: $k+1 \leq N$ $lambda * (b-a) / 2\hat{}N \leq 10 *delta$ $2 \hat{} N \leq lambda * (b - a) / (5 * delta)$
using I J K ‹$delta > 0$› **unfolding** *N-def* **by** (*auto simp add*: *divide-simps algebra-simps*)
then have $2 \hat{} k \neq (0::real)$ $k \leq N$
by *auto*
then have $(2\hat{}(N-k)::real) = 2\hat{}N/2\hat{}k$
by (*metis* (*no-types*) *add-diff-cancel-left′ le-Suc-ex nonzero-mult-div-cancel-left power-add*)

Define 2^N points along the path, separated by at most 10δ, and their projections.

define $g::nat \Rightarrow 'a$ **where** $g = (\lambda i.\ f(a + (b-a) * i/2\hat{}N))$
have $g\ 0 = f\ a$ $g(2\hat{}N) = f\ b$
unfolding *g-def* **by** *auto*
have $*$: $a + (b-a) * i/2\hat{}N \in \{a..b\}$ **if** $i \in \{0..2\hat{}N\}$ **for** $i::nat$
proof −
have $a + (b - a) * (real\ i / 2 \hat{} N) \leq a + (b-a) * (2\hat{}N/2\hat{}N)$
apply (*intro mono-intros*) **using** *that* ‹$a \leq b$› **by** *auto*
then show *?thesis* **using** ‹$a \leq b$› **by** *auto*
qed
have A: $dist\ (g\ i)\ (g\ (Suc\ i)) \leq 10 * delta + C$ **if** $i \in \{0..<2\hat{}N\}$ **for** i
proof −
have $dist\ (g\ i)\ (g\ (Suc\ i)) \leq lambda * dist\ (a + (b-a) * i/2\hat{}N)\ (a + (b-a) * (Suc\ i)/2\hat{}N) + C$
unfolding *g-def* **apply** (*intro assms*(*2*) $*$)
using *that* **by** *auto*
also have ... $= lambda * (b-a)/2\hat{}N + C$
unfolding *dist-real-def* **using** ‹$a \leq b$› **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... $\leq 10 * delta + C$
using N **by** *simp*
finally show *?thesis* **by** *simp*
qed
define p **where** $p = (\lambda i.\ if\ i = 0\ then\ pa\ else\ if\ i = 2\hat{}N\ then\ pb\ else\ SOME\ p.\ p \in proj\text{-}set\ (g\ i)\ G)$
have B: $p\ i \in proj\text{-}set\ (g\ i)\ G$ **if** $i \in \{0..2\hat{}N\}$ **for** i
proof ($cases\ i = 0 \vee i = 2\hat{}N$)
case *True*
then show *?thesis*
using ‹$pa \in proj\text{-}set\ (f\ a)\ G$› ‹$pb \in proj\text{-}set\ (f\ b)\ G$› **unfolding** *p-def g-def* **by** *auto*
next
case *False*
then have $p\ i = (SOME\ p.\ p \in proj\text{-}set\ (g\ i)\ G)$

unfolding *p-def* **by** *auto*
moreover have *proj-set* $(g\ i)\ G \neq \{\}$
apply (*rule proj-set-nonempty-of-proper*) **using** *geodesic-segment-topology*[*OF* ‹*geodesic-segment G*›] **by** *auto*
ultimately show *?thesis*
using *some-in-eq* **by** *auto*
qed
have *C*: $dist\ (p\ i)\ (g\ i) \geq 5 * delta * k + 15/2 * delta + C/2$ **if** $i \in \{0..2\hat{}N\}$ **for** *i*
proof −
have $5 * delta * k + 15/2 * delta + C/2 \leq D$
using *k*(*1*) **by** *simp*
also have ... $\leq$ *infdist* $(g\ i)\ G$
unfolding *g-def* **apply** (*rule* ‹$\bigwedge t.\ t \in \{a..b\} \Longrightarrow infdist\ (f\ t)\ G \geq D$›) **using** $*$ *that* **by** *auto*
also have ... $=$ *dist* $(p\ i)\ (g\ i)$
using *that proj-setD*(*2*)[*OF B*[*OF that*]] **by** (*simp add*: *metric-space-class.dist-commute*)
finally show *?thesis* **by** *simp*
qed

Use the basic statement to show that, along packets of size 2^k, the projections are within 5δ of each other.

have *I*: $dist\ (p\ (2\hat{}k * j))\ (p\ (2\hat{}k * (Suc\ j))) \leq 5 * delta$ **if** $j \in \{0..<2\hat{}(N-k)\}$ **for** *j*
proof −
have *I*: $i + 2\hat{}k * j \in \{0..2\hat{}N\}$ **if** $i \in \{0..2\hat{}k\}$ **for** *i*
proof −
have $i + 2\ \hat{}\ k * j \leq 2\hat{}k + 2\hat{}k * (2\hat{}(N-k)-1)$
apply (*intro mono-intros*) **using** *that* ‹$j \in \{0..<2\hat{}(N-k)\}$› **by** *auto*
also have ... $= 2\hat{}N$
using ‹$k+1 \leq N$› **by** (*auto simp add*: *algebra-simps semiring-normalization-rules*(*26*))
finally show *?thesis* **by** *auto*
qed
have *I'*: $i + 2\hat{}k * j \in \{0..<2\hat{}N\}$ **if** $i \in \{0..<2\hat{}k\}$ **for** *i*
proof −
have $i + 2\ \hat{}\ k * j < 2\hat{}k + 2\hat{}k * (2\hat{}(N-k)-1)$
apply (*intro mono-intros*) **using** *that* ‹$j \in \{0..<2\hat{}(N-k)\}$› **by** *auto*
also have ... $= 2\hat{}N$
using ‹$k+1 \leq N$› **by** (*auto simp add*: *algebra-simps semiring-normalization-rules*(*26*))
finally show *?thesis* **by** *auto*
qed
define g' **where** $g' = (\lambda i.\ g\ (i + 2\hat{}k * j))$
define p' **where** $p' = (\lambda i.\ p\ (i + 2\hat{}k * j))$
have $dist\ (p'\ 0)\ (p'\ (2\hat{}k)) \leq 5 * deltaG(TYPE('a))$
apply (*rule Main*[**where** $?g = g'$ **and** $?c = C$]) **unfolding** *p'-def g'-def* **using** *A B C I I'* ‹$C \geq 0$› **by** *auto*
also have ... $\leq 5 * delta$
using ‹$deltaG(TYPE('a)) < delta$› **by** *auto*
finally show *?thesis*

unfolding *p′-def* **by** *auto*
qed

Control the total distance by adding the contributions of blocks of size 2^k.

have ∗: *dist* (*p 0*) (*p*(*2^k* ∗ *j*)) ≤ (∑ *i*<*j*. *dist* (*p* (*2^k* ∗ *i*)) (*p* (*2^k* ∗ (*Suc i*)))) **for** *j*
proof (*induction j*)
case (*Suc j*)
have *dist* (*p 0*) (*p*(*2^k* ∗ (*Suc j*))) ≤ *dist* (*p 0*) (*p*(*2^k* ∗ *j*)) + *dist* (*p*(*2^k* ∗ *j*)) (*p*(*2^k* ∗ (*Suc j*)))
by (*intro mono-intros*)
also have ... ≤ (∑ *i*<*j*. *dist* (*p* (*2^k* ∗ *i*)) (*p* (*2^k* ∗ (*Suc i*)))) + *dist* (*p*(*2^k* ∗ *j*)) (*p*(*2^k* ∗ (*Suc j*)))
using *Suc.IH* **by** *auto*
also have ... = (∑ *i*<*Suc j*. *dist* (*p* (*2^k* ∗ *i*)) (*p* (*2^k* ∗ (*Suc i*))))
by *auto*
finally show *?case* **by** *simp*
qed (*auto*)
have *dist pa pb* = *dist* (*p 0*) (*p* (*2^N*))
unfolding *p-def* **by** *auto*
also have ... = *dist* (*p 0*) (*p* (*2^k* ∗ *2^*(*N*−*k*)))
using ‹*k* +*1* ≤ *N*› **by** (*auto simp add*: *semiring-normalization-rules*(*26*))
also have ... ≤ (∑ *i*<*2^*(*N*−*k*). *dist* (*p* (*2^k* ∗ *i*)) (*p* (*2^k* ∗ (*Suc i*))))
using ∗ **by** *auto*
also have ... ≤ (∑ (*i*::*nat*)<*2^*(*N*−*k*). *5* ∗ *delta*)
apply (*rule sum-mono*) **using** *I* **by** *auto*
also have ... = *5* ∗ *delta* ∗ *2^*(*N*−*k*)
by *auto*
also have ... = *5* ∗ *delta* ∗ *2^N* ∗ (*1*/ *2^k*)
unfolding ‹(*2^*(*N*−*k*)::*real*) = *2^N*/*2^k*› **by** *simp*
also have ... ≤ *5* ∗ *delta* ∗ (*2* ∗ *lambda* ∗ (*b*−*a*)/(*10* ∗ *delta*)) ∗ (*2* ∗ *exp*(*15*/*2*/*5* ∗ *ln 2*) ∗ *exp*(− ((*D*−*C*/*2*) ∗ *ln 2* / (*5* ∗ *delta*))))
apply (*intro mono-intros*) **using** ‹*delta* > *0*› ‹*lambda* > *0*› ‹*a* < *b*› *k′ N* **by** *auto*
also have ... = (*2* ∗ *exp*(*15*/*2*/*5* ∗ *ln 2*)) ∗ *lambda* ∗ (*b*−*a*) ∗ *exp*(−(*D*−*C*/*2*) ∗ *ln 2* / (*5* ∗ *delta*))
using ‹*delta* > *0*› **by** (*auto simp add*: *algebra-simps divide-simps*)
finally show *?thesis*
unfolding ‹*exp*(*15*/*2*/*5* ∗ *ln 2*) = *2* ∗ *exp*(*1*/*2* ∗ *ln* (*2*::*real*))› **by** *auto*
qed
qed

We deduce from the previous result that a projection on a quasiconvex set is also exponentially contracting. To do this, one uses the contraction of a projection on a geodesic, and one adds up the additional errors due to the quasi-convexity. In particular, the projections on the original quasiconvex set or the geodesic do not have to coincide, but they are within distance at most $C + 8\delta$.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *quasiconvex-projection-exp-contracting*:

```
  assumes quasiconvex K G
          ⋀x y. x ∈ {a..b} ⟹ y ∈ {a..b} ⟹ dist (f x) (f y) ≤ lambda ∗ dist x y
+ C
          a ≤ b
          pa ∈ proj-set (f a) G
          pb ∈ proj-set (f b) G
          ⋀t. t ∈ {a..b} ⟹ infdist (f t) G ≥ D
          D ≥ 15/2 ∗ delta + K + C/2
          delta > deltaG(TYPE('a))
          C ≥ 0
          lambda ≥ 0
  shows dist pa pb ≤ 2 ∗ K + 8 ∗ delta + max (5 ∗ deltaG(TYPE('a))) ((4 ∗
exp(1/2 ∗ ln 2)) ∗ lambda ∗ (b−a) ∗ exp(−(D − K − C/2) ∗ ln 2 / (5 ∗ delta)))
proof −
  obtain H where H: geodesic-segment-between H pa pb ⋀q. q ∈ H ⟹ infdist q
G ≤ K
     using quasiconvexD[OF assms(1) proj-setD(1)[OF ‹pa ∈ proj-set (f a) G›]
proj-setD(1)[OF ‹pb ∈ proj-set (f b) G›]] by auto
  obtain qa where qa: qa ∈ proj-set (f a) H
   using proj-set-nonempty-of-proper[of H f a] geodesic-segment-topology[OF geodesic-segmentI[OF
H(1)]] by auto
  obtain qb where qb: qb ∈ proj-set (f b) H
   using proj-set-nonempty-of-proper[of H f b] geodesic-segment-topology[OF geodesic-segmentI[OF
H(1)]] by auto

  have I: infdist (f t) H ≥ D − K if t ∈ {a..b} for t
  proof −
    have ∗: D − K ≤ dist (f t) h if h ∈ H for h
    proof −
      have D − K − dist (f t) h ≤ e if e > 0 for e
      proof −
        have ∗: infdist h G < K + e using H(2)[OF ‹h ∈ H›] ‹e > 0› by auto
        obtain g where g: g ∈ G dist h g < K + e
          using infdist-almost-attained[OF ∗] proj-setD(1)[OF ‹pa ∈ proj-set (f a)
G›] by auto
        have D ≤ dist (f t) g
        using ‹⋀t. t ∈ {a..b} ⟹ infdist (f t) G ≥ D›[OF ‹t ∈ {a..b}›] infdist-le[OF
‹g ∈ G›, of f t] by auto
        also have ... ≤ dist (f t) h + dist h g
          by (intro mono-intros)
        also have ... ≤ dist (f t) h + K + e
          using g(2) by auto
        finally show ?thesis by auto
      qed
      then have ∗: D − K − dist (f t) h ≤ 0
        using dense-ge by blast
      then show ?thesis by simp
    qed
    have D − K ≤ Inf (dist (f t) ‘ H)
```

apply (*rule cInf-greatest*) **using** $*$ $H(1)$ **by** *auto*
then show $D - K \leq$ *infdist* (*f t*) *H*
apply (*subst infdist-notempty*) **using** $H(1)$ **by** *auto*
qed
have *Q*: *dist qa qb* $\leq$ *max* ($5 *$ *deltaG*(*TYPE*('*a*))) (($4 *$ *exp*($1/2 *$ *ln* 2)) $*$ *lambda* $*$ ($b-a$) $*$ *exp*($-$(($D - K$)$-C/2$) $*$ *ln* 2 / ($5 *$ *delta*)))
apply (*rule geodesic-projection-exp-contracting*[*OF geodesic-segmentI*[*OF* ‹*geodesic-segment-between H pa pb*›] *assms*(*2*) *assms*(*3*)])
using *qa qb I assms* **by** *auto*

have *A*: *dist pa qa* $\leq 4 *$ *delta* $+ K$
proof $-$
have *dist* (*f a*) *pa* $-$ *dist* (*f a*) *qa* $- K \leq e$ **if** $e > 0$ **for** *e*::*real*
proof $-$
have $*$: *infdist qa G* $< K + e$ **using** $H(2)$[*OF proj-setD*(*1*)[*OF qa*]] ‹$e > 0$› **by** *auto*
obtain *g* **where** *g*: $g \in G$ *dist qa g* $< K + e$
using *infdist-almost-attained*[*OF* $*$] *proj-setD*(*1*)[*OF* ‹*pa* $\in$ *proj-set* (*f a*) *G*›] **by** *auto*
have *dist* (*f a*) *pa* $\leq$ *dist* (*f a*) *g*
unfolding *proj-setD*(*2*)[*OF* ‹*pa* $\in$ *proj-set* (*f a*) *G*›] **using** *infdist-le*[*OF* ‹$g \in G$›, *of f a*] **by** *simp*
also have ... $\leq$ *dist* (*f a*) *qa* $+$ *dist qa g*
by (*intro mono-intros*)
also have ... $\leq$ *dist* (*f a*) *qa* $+ K + e$
using *g*(*2*) **by** *auto*
finally show *?thesis* **by** *simp*
qed
then have *I*: *dist* (*f a*) *pa* $-$ *dist* (*f a*) *qa* $- K \leq 0$
using *dense-ge* **by** *blast*
have *dist* (*f a*) *qa* $+$ *dist qa pa* $\leq$ *dist* (*f a*) *pa* $+ 4 *$ *deltaG*(*TYPE*('*a*))
apply (*rule dist-along-geodesic*[*OF geodesic-segmentI*[*OF H*(*1*)]]) **using** *qa* *H*(*1*) **by** *auto*
also have ... $\leq$ *dist* (*f a*) *qa* $+ K + 4 *$ *delta*
using *I assms* **by** *auto*
finally show *?thesis*
by (*simp add*: *metric-space-class.dist-commute*)
qed
have *B*: *dist qb pb* $\leq 4 *$ *delta* $+ K$
proof $-$
have *dist* (*f b*) *pb* $-$ *dist* (*f b*) *qb* $- K \leq e$ **if** $e > 0$ **for** *e*::*real*
proof $-$
have $*$: *infdist qb G* $< K + e$ **using** $H(2)$[*OF proj-setD*(*1*)[*OF qb*]] ‹$e > 0$› **by** *auto*
obtain *g* **where** *g*: $g \in G$ *dist qb g* $< K + e$
using *infdist-almost-attained*[*OF* $*$] *proj-setD*(*1*)[*OF* ‹*pa* $\in$ *proj-set* (*f a*) *G*›] **by** *auto*
have *dist* (*f b*) *pb* $\leq$ *dist* (*f b*) *g*
unfolding *proj-setD*(*2*)[*OF* ‹*pb* $\in$ *proj-set* (*f b*) *G*›] **using** *infdist-le*[*OF* ‹*g*

```
∈ G›, of f b] by simp
      also have ... ≤ dist (f b) qb + dist qb g
        by (intro mono-intros)
      also have ... ≤ dist (f b) qb + K + e
        using g(2) by auto
      finally show ?thesis by simp
    qed
    then have I: dist (f b) pb − dist (f b) qb − K ≤ 0
      using dense-ge by blast
    have dist (f b) qb + dist qb pb ≤ dist (f b) pb + 4 ∗ deltaG(TYPE('a))
       apply (rule dist-along-geodesic[OF geodesic-segmentI[OF H(1)]]) using qb
H(1) by auto
    also have ... ≤ dist (f b) qb + K + 4 ∗ delta
      using I assms by auto
    finally show ?thesis
      by simp
  qed
  have dist pa pb ≤ dist pa qa + dist qa qb + dist qb pb
    by (intro mono-intros)
  then show ?thesis
    using Q A B by auto
qed
```

The next statement is the main step in the proof of the Morse-Gromov theorem given by Shchur in [Shc13], asserting that a quasi-geodesic and a geodesic with the same endpoints are close. We show that a point on the quasi-geodesic is close to the geodesic – the other inequality will follow easily later on. We also assume that the quasi-geodesic is parameterized by a Lipschitz map – the general case will follow as any quasi-geodesic can be approximated by a Lipschitz map with good controls.

Here is a sketch of the proof. Fix two large constants $L \leq D$ (that we will choose carefully to optimize the values of the constants at the end of the proof). Consider a quasi-geodesic f between two points $f(u^-)$ and $f(u^+)$, and a geodesic segment G between the same points. Fix $f(z)$. We want to find a bound on $d(f(z), G)$. 1 - If this distance is smaller than L, we are done (and the bound is L). 2 - Assume it is larger. Let π_z be a projection of $f(z)$ on G (at distance $d(f(z), G)$ of $f(z)$), and H a geodesic between z and π_z. The idea will be to project the image of f on H, and record how much progress is made towards $f(z)$. In this proof, we will construct several points before and after z. When necessary, we put an exponent $-$ on the points before z, and $+$ on the points after z. To ease the reading, the points are ordered following the alphabetical order, i.e., $u^- \leq v \leq w \leq x \leq y^- \leq z$. One can find two points $f(y^-)$ and $f(y^+)$ on the left and the right of $f(z)$ that project on H roughly at distance L of π_z (up to some $O(\delta)$ – recall that the closest point projection is not uniquely defined, and not continuous, so we make some choice here). Let d^- be the minimal distance of $f([u^-, y^-])$

to H, and let d^+ be the minimal distance of $f([y^+, u^+)]$ to H.

2.1 If the two distances d^- and d^+ are less than D, then the distance between two points realizing the minimum (say $f(c^-)$ and $f(c^+)$) is at most $2D + L$, hence $c^+ - c^-$ is controlled (by $\lambda \cdot (2D+L)+C$) thanks to the quasi-isometry property. Therefore, $f(z)$ is not far away from $f(c^-)$ and $f(c^+)$ (again by the quasi-isometry property). Since the distance from these points to π_z is controlled (by $D + L$), we get a good control on $d(f(z), \pi_z)$, as desired.

2.2 The interesting case is when d^- and d^+ are both $> D$. Assume also for instance $d^- \geq d^+$, as the other case is analogous. We will construct two points $f(v)$ and $f(x)$ with $u^- \leq v \leq x \leq y^-$ with the following property:

$$K_1 e^{K_2 d(f(v),H)} \leq x - v, \tag{1}$$

where K_1 and K_2 are some explicit constants (depending on λ, δ, L and D). Let us show how this will conclude the proof. The distance from $f(v)$ to $f(c^+)$ is at most $d(f(v), H) + L + d^+ \leq 3d(f(v), H)$. Therefore, $c^+ - v$ is also controlled by $K' d(f(v), H)$ by the quasi-isometry property. This gives

$$\begin{aligned} K &\leq K(x-v)e^{-K(c^+-v)} \leq (e^{K(x-v)} - 1) \cdot e^{-K(c^+-v)} \\ &= e^{-K(c^+-x)} - e^{-K(c^+-v)} \leq e^{-K(c^+-x)} - e^{-K(u^+-u^-)}. \end{aligned}$$

This shows that, when one goes from the original quasi-geodesic $f([u^-, u^+])$ to the restricted quasi-geodesic $f([x, c^+])$, the quantity $e^{-K\cdot}$ decreases by a fixed amount. In particular, this process can only happen a uniformly bounded number of times, say n.

Let G' be a geodesic between $f(x)$ and $f(c^+)$. One checks geometrically that $d(f(z), G) \leq d(f(z), G') + (L + O(\delta))$, as both projections of $f(x)$ and $f(c^+)$ on H are within distance L of π_z. Iterating the process n times, one gets finally $d(f(z), G) \leq O(1) + n(L + O(\delta))$. This is the desired bound for $d(f(z), G)$.

To complete the proof, it remains to construct the points $f(v)$ and $f(x)$ satisfying (1). This will be done through an inductive process.

Assume first that there is a point $f(v)$ whose projection on H is close to the projection of $f(u^-)$, and with $d(f(v), H) \leq 2d^-$. Then the projections of $f(v)$ and $f(y^-)$ are far away (at distance at least $L + O(\delta)$). Since the portion of f between v and y^- is everywhere at distance at least d^- of H, the projection on H contracts by a factor e^{-d^-}. It follows that this portion of f has length at least $e^{d^-} \cdot (L + O(\delta))$. Therefore, by the quasi-isometry property, one gets $x - v \geq K e^{d^-}$. On the other hand, $d(v, H)$ is bounded above by $2d^-$ by assumption. This gives the desired inequality (1) with $x = y^-$.

Otherwise, all points $f(v)$ whose projection on H is close to the projection of $f(u^-)$ are such that $d(f(v), H) \geq 2d^-$. Consider $f(w_1)$ a point whose projection on H is at distance roughly 10δ of the projection of $f(u^-)$. Let V_1 be the set of points at distance at most d^- of H, i.e., the d_1-neighborhood of H. Then the distance between the projections of $f(u^-)$ and $f(w_1)$ on V_1 is very large (are there is an additional big contraction to go from V_1 to H). And moreover all the intermediate points $f(v)$ are at distance at least $2d^-$ of H, and therefore at distance at least d^- of H. Then one can play the same game as in the first case, where y^- replaced by w_1 and H replaced by V_1. If there is a point $f(v)$ whose projection on V_1 is close to the projection of $f(u^-)$, then the pair of points v and $x = w_1$ works. Otherwise, one lifts everything to V_2, the neighborhood of size $2d^-$ of V_1, and one argues again in the same way.

The induction goes on like this until one finds a suitable pair of points. The process has indeed to stop at one time, as it can only go on while $f(u^-)$ is outside of V_k, the $(2^k - 1)d^-$ neighborhood of H). This concludes the sketch of the proof, modulo the adjustment of constants.

Comments on the formalization below:

- The proof is written as an induction on $u^+ - u^-$. This makes it possible to either prove the bound directly (in the cases 1 and 2.1 above), or to use the bound on $d(z, G')$ in case 2.2 using the induction assumption, and conclude the proof. Of course, $u^+ - u^-$ is not integer-valued, but in the reduction to G' it decays by a fixed amount, so one can easily write this down as a genuine induction.

- The main difficulty in the proof is to construct the pair (v, x) in case 2.2. This is again written as an induction over k: either the required bound is true, or one can find a point $f(w)$ whose projection on V_k is far enough from the projection of $f(u^-)$. Then, either one can use this point to prove the bound, or one can construct a point with the same property with respect to V_{k+1}, concluding the induction.

- Instead of writing u^- and u^+ (which are not good variable names in Isabelle), we write um and uM. Similarly for other variables.

- The proof only works when $\delta > 0$ (as one needs to divide by δ in the exponential gain). Hence, we formulate it for some δ which is strictly larger than the hyperbolicity constant. In a subsequent application of the lemma, we will deduce the same statement for the hyperbolicity constant by a limiting argument.

- To optimize the value of the constant in the end, there is an additional important trick with respect to the above sketch: in case 2.2, there is an exponential gain. One can spare a fraction $(1 - \alpha)$ of this gain to improve the constants, and spend the remaining fraction α to make the argument work. This makes it possible to reduce the value of the constant roughly from 40000 to 100, so we do it in the proof below. The values of L, D and α can be chosen freely, and have been chosen to get the best possible constant in the end.

- For another optimization, we do not induce in terms of the distance from $f(z)$ to the geodesic G, but rather in terms of the Gromov product $(f(u^-), f(u^+))_{f(z)}$ (which is the same up to $O(\delta)$. And we do not take for H a geodesic from $f(z)$ to its projection on G, but rather a geodesic from $f(z)$ to the point m on $[f(u^-), f(u^+)]$ opposite to $f(z)$ in the tripod, i.e., at distance $(f(z), f(u^+))_{f(u^-)}$ of $f(u^-)$, and at distance $(f(z), f(u^-))_{f(u^+)}$ of $f(u^+)$. Let π_z denote the point on $[f(z), m]$ at distance $(f(u^-), f(u^+)_{f(z)}$ of $f(z)$. (It is within distance 2δ of m). In both approaches, what we want to control by induction is the distance from $f(z)$ to π_z. However, in the first approach, the points $f(u^-)$ and $f(u^+)$ project on H between π_z and $f(z)$, and since the location of their projection is only controlled up to 4δ one loses essentially a 4δ-length of L for the forthcoming argument. In the second approach, the projections on H are on the other side of π_z compared to $f(z)$, so one does not lose anything, and in the end it gives genuinely better bounds (making it possible to gain roughly 10δ in the final estimate).

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *Morse-Gromov-theorem-aux1*:
  fixes *f*::*real* ⇒ *'a*
  assumes *continuous-on* {*a*..*b*} *f*
    lambda C−quasi-isometry-on {*a*..*b*} *f*
    a ≤ *b*
    geodesic-segment-between G (*f a*) (*f b*)
    z ∈ {*a*..*b*}
    delta > *deltaG*(*TYPE*(*'a*))
  shows *infdist* (*f z*) *G* ≤ *lambda*^*2* ∗ (*11*/*2* ∗ *C* + *91* ∗ *delta*)
proof −
  have *C* ≥ *0 lambda* ≥ *1* **using** *quasi-isometry-onD assms* **by** *auto*
  have *delta* > *0* **using** *assms delta-nonneg order-trans* **by** *linarith*

We give their values to the parameters L, D and α that we will use in the proof. We also define two constants K and K_{mult} that appear in the precise formulation of the bounds. Their values have no precise meaning, they are just the outcome of the computation

  define *alpha*::*real* **where** *alpha* = *12*/*100*
  have *alphaaux*:*alpha* > *0 alpha* ≤ *1* **unfolding** *alpha-def* **by** *auto*
  define *L*::*real* **where** *L* = *18* ∗ *delta*
  define *D*::*real* **where** *D* = *55* ∗ *delta*
  define *K* **where** *K* = *alpha* ∗ *ln 2* / (*5* ∗ (*4* + (*L* + *2* ∗ *delta*)/*D*) ∗ *delta* ∗ *lambda*)
  have *K* > *0 L* > *0 D* > *0* **unfolding** *K-def L-def D-def* **using** ‹*delta* > *0*› ‹*lambda* ≥ *1*› *alpha-def* **by** *auto*

have *Laux*: $L \geq 18 * delta$ $D \geq 50 * delta$ $L \leq D$ $D \leq 4 * L$ **unfolding** *L-def* *D-def* **using** ‹$delta > 0$› **by** *auto*
have *Daux*: $8 * delta \leq (1 - alpha) * D$ **unfolding** *alpha-def* *D-def* **using** ‹$delta > 0$› **by** *auto*
define *Kmult* **where** $Kmult = ((L + 4 * delta)/(L - 13 * delta)) * ((4 * exp(1/2 * ln\ 2)) * lambda * exp\ (- (1 - alpha) * D * ln\ 2 / (5 * delta)) / K)$
have $Kmult > 0$ **unfolding** *Kmult-def* **using** *Laux* ‹$delta > 0$› ‹$K > 0$› ‹$lambda \geq 1$› **by** (*auto simp add*: *divide-simps*)

We prove that, for any pair of points to the left and to the right of $f(z)$, the distance from $f(z)$ to a geodesic between these points is controlled. We prove this by reducing to a closer pair of points, i.e., this is an inductive argument over real numbers. However, we formalize it as an artificial induction over natural numbers, as this is how induction works best, and since in our reduction step the new pair of points is always significantly closer than the initial one, at least by an amount δ/λ.

The main inductive bound that we will prove is the following. In this bound, the first term is what comes from the trivial cases 1 and 2.1 in the description of the proof before the statement of the theorem, while the most interesting term is the second term, corresponding to the induction per se.

have *Main*: $\bigwedge um\ uM.\ um \in \{a..z\} \Longrightarrow uM \in \{z..b\}$
$\Longrightarrow uM - um \leq n * (1/4) * delta / lambda$
$\Longrightarrow$ *Gromov-product-at* $(f\ z)\ (f\ um)\ (f\ uM) \leq lambda\hat{\ }2 * (D + (3/2) * L + delta + 11/2 * C) - 2 * delta + Kmult * (1 - exp(- K * (uM - um)))$
for n::*nat*
proof (*induction n*)

Trivial base case of the induction

case *0*
then have $*$: $z = um$ $z = uM$ **by** *auto*
then have *Gromov-product-at* $(f\ z)\ (f\ um)\ (f\ uM) = 0$ **by** *auto*
also have ... $\leq 1 * (D + (3/2) * L + delta + 11/2 * C) - 2 * delta + 0 * (1 - exp(- K * (uM - um)))$
using *Laux* ‹$C \geq 0$› ‹$delta > 0$› **by** *auto*
also have ... $\leq lambda\hat{\ }2 * (D + (3/2) * L + delta + 11/2 * C) - 2 * delta + Kmult * (1 - exp(- K * (uM - um)))$
apply (*intro mono-intros*)
using ‹$C \geq 0$› ‹$delta > 0$› *Laux* ‹$D > 0$› ‹$K > 0$› *0.prems* ‹$lambda \geq 1$› ‹$Kmult > 0$› **by** *auto*
finally show *?case* **by** *auto*
next
case (*Suc n*)
show *?case*
proof (*cases Gromov-product-at* $(f\ z)\ (f\ um)\ (f\ uM) \leq L$)

If $f(z)$ is already close to the geodesic, there is nothing to do, and we do not need the induction assumption. This is case 1 in the description above.

case *True*
have $L \leq 1 * (D + (3/2) * L + delta + 11/2 * C) - 2 * delta + 0 * (1 - exp(- K * (uM - um)))$
using *Laux* ‹$C \geq 0$› ‹$delta > 0$› **by** *auto*
also have ... $\leq lambda\hat{\ }2 * (D + (3/2) * L + delta + 11/2 * C) - 2 * delta + Kmult * (1 - exp(- K * (uM - um)))$
apply (*intro mono-intros*)
using ‹$C \geq 0$› ‹$delta > 0$› *Laux* ‹$D > 0$› *Suc.prems* ‹$K > 0$› ‹$lambda \geq 1$› ‹$Kmult > 0$› **by** *auto*
finally show *?thesis* **using** *True* **by** *auto*
next

We come to the interesting case where $f(z)$ is far away from a geodesic between $f(um)$ and $f(uM)$. Let m be close to a projection of $f(z)$ on such a geodesic (we use the opposite point of $f(z)$ on the corresponding tripod). On a geodesic between $f(z)$ and m, consider the point pi_z at distance $(f(um), f(uM))_{f(z)}$ of $f(z)$. It is very close to m (within distance 2δ). We will push the points $f(um)$ and $f(uM)$ towards $f(z)$ by considering points whose projection on a geodesic H between m and z is roughly at distance L of pi_z.

case *False*
define m **where** m = *geodesic-segment-param* {*f um*−−*f uM*} (*f um*) (*Gromov-product-at* (*f um*) (*f z*) (*f uM*))
have *dist* (*f z*) $m \leq$ *Gromov-product-at* (*f z*) (*f um*) (*f uM*) + 2 ∗ *deltaG*(*TYPE*(′*a*))
unfolding *m-def* **by** (*rule dist-triangle-side-middle*, *auto*)
then have ∗: *dist* (*f z*) $m \leq$ *Gromov-product-at* (*f z*) (*f um*) (*f uM*) + 2 ∗ *delta*
using ‹*deltaG*(*TYPE*(′*a*)) < *delta*› **by** *auto*
have *Gromov-product-at* (*f z*) (*f um*) (*f uM*) ≤ *infdist* (*f z*) {*f um*−−*f uM*}
by (*intro mono-intros*, *auto*)
also have ... ≤ *dist* (*f z*) *m*
apply (*rule infdist-le*) **unfolding** *m-def* **by** *auto*
finally have ∗∗: *Gromov-product-at* (*f z*) (*f um*) (*f uM*) ≤ *dist* (*f z*) *m*
by *auto*

define *H* **where** *H* = {*f z*−−*m*}
define *pi-z* **where** *pi-z* = *geodesic-segment-param H* (*f z*) (*Gromov-product-at* (*f z*) (*f um*) (*f uM*))
have *pi-z* ∈ *H m* ∈ *H f z* ∈ *H*
unfolding *pi-z-def H-def* **by** (*auto simp add*: *geodesic-segment-param-in-segment*)
have *H*: *geodesic-segment-between H* (*f z*) *m*
unfolding *H-def* **by** *auto*
have *Dpi-z*: *dist* (*f z*) *pi-z* = *Gromov-product-at* (*f z*) (*f um*) (*f uM*)
unfolding *pi-z-def H-def* **by** (*rule geodesic-segment-param*(*6*)[**where** *?y* = *m*], *auto simp add*: ∗∗)
moreover have *dist* (*f z*) *m* = *dist* (*f z*) *pi-z* + *dist pi-z m*
apply (*rule geodesic-segment-dist*[*of H*, *symmetric*]) **using** ‹*pi-z* ∈ *H*› **unfolding** *H-def* **by** *auto*

ultimately have *dist pi-z m* $\leq$ *2* $*$ *delta*
using $*$ **by** *auto*

Introduce the notation p for some projection on the geodesic H.

define *p* **where** *p* = (λ*r*. *SOME x*. *x* $\in$ *proj-set* (*f r*) *H*)
have *p*: *p x* $\in$ *proj-set* (*f x*) *H* **for** *x*
unfolding *p-def* **using** *proj-set-nonempty-of-proper*[*of H f x*] *geodesic-segment-topology*[*OF geodesic-segmentI*[*OF H*]]
by (*simp add*: *some-in-eq*)
then have *pH*: *p x* $\in$ *H* **for** *x*
using *proj-setD*(*1*) **by** *auto*
have *pz*: *p z* = *f z*
using *p*[*of z*] *H* **by** *auto*

The projection of $f(um)$ on H is close to pi_z (but it does not have to be exactly pi_z). It is between pi_z and m.

have *dist* (*f um*) (*f z*) $\leq$ *dist* (*f um*) (*p um*) + *dist* (*p um*) (*f z*)
by (*intro mono-intros*)
also have ... $\leq$ *dist* (*f um*) *m* + *dist* (*p um*) (*f z*)
unfolding *proj-setD*(*2*)[*OF p*[*of um*]] *H-def* **by** (*auto intro*!: *infdist-le*)
also have ... = *Gromov-product-at* (*f um*) (*f z*) (*f uM*) + *dist* (*p um*) (*f z*)
unfolding *m-def* **by** *simp*
finally have *A*: *Gromov-product-at* (*f z*) (*f um*) (*f uM*) $\leq$ *dist* (*p um*) (*f z*)
unfolding *Gromov-product-at-def* **by** (*simp add*: *metric-space-class.dist-commute divide-simps*)
have *dist* (*p um*) *pi-z* = *abs*(*dist* (*p um*) (*f z*) − *dist pi-z* (*f z*))
apply (*rule dist-along-geodesic-wrt-endpoint*[*of H - m*]) **using** *pH* ‹*pi-z* $\in$ *H*› *H-def* **by** *auto*
also have ... = *dist* (*p um*) (*f z*) − *dist pi-z* (*f z*)
using *A Dpi-z* **by** (*simp add*: *metric-space-class.dist-commute*)
finally have *Dum*: *dist* (*p um*) (*f z*) = *dist* (*p um*) *pi-z* + *dist pi-z* (*f z*) **by** *auto*

Choose a point $f(ym)$ whose projection on H is roughly at distance L of pi_z.

have $\exists$ *ym* $\in$ {*um..z*}. (*dist* (*p um*) (*p ym*) $\in$ {(*L* + *dist pi-z* (*p um*)) − *4* $*$ *delta* − *2* $*$ *0* .. *L* + *dist pi-z* (*p um*)})
$\wedge$ ($\forall$ *r* $\in$ {*um..ym*}. *dist* (*p um*) (*p r*) $\leq$ *L* + *dist pi-z* (*p um*))
proof (*rule quasi-convex-projection-small-gaps*[**where** *?f* = *f* **and** *?G* = *H*])
show *continuous-on* {*um..z*} *f*
apply (*rule continuous-on-subset*[*OF* ‹*continuous-on* {*a..b*} *f*›])
using ‹*um* $\in$ {*a..z*}› ‹*z* $\in$ {*a..b*}› **by** *auto*
show *um* $\leq$ *z* **using** ‹*um* $\in$ {*a..z*}› **by** *auto*
show *quasiconvex 0 H* **using** *quasiconvex-of-geodesic geodesic-segmentI H* **by** *auto*
show *deltaG TYPE*('*a*) < *delta* **by** *fact*
have *L* + *dist pi-z* (*p um*) $\leq$ *dist* (*f z*) *pi-z* + *dist pi-z* (*p um*)
using *False Dpi-z* **by** (*simp add*: *metric-space-class.dist-commute*)

then have $L + dist\ pi\text{-}z\ (p\ um) \leq dist\ (p\ um)\ (f\ z)$
using *Dum* **by** (*simp add*: *metric-space-class.dist-commute*)
then show $L + dist\ pi\text{-}z\ (p\ um) \in \{4 * delta + 2 * 0..dist\ (p\ um)\ (p\ z)\}$
using ‹*delta* > *0*› *False L-def pz* **by** *auto*
show *p ym* ∈ *proj-set* (*f ym*) *H* **for** *ym* **using** *p* **by** *simp*
qed
then obtain *ym* **where** *ym* : $ym \in \{um..z\}$
$dist\ (p\ um)\ (p\ ym) \in \{(L + dist\ pi\text{-}z\ (p\ um)) - 4 * delta - 2 * 0\ ..\ L + dist\ pi\text{-}z\ (p\ um)\}$
$\bigwedge r.\ r \in \{um..ym\} \Longrightarrow dist\ (p\ um)\ (p\ r) \leq L + dist\ pi\text{-}z\ (p\ um)$
by *blast*
have ∗: *continuous-on* {*um..ym*} (λ*r*. *infdist* (*f r*) *H*)
using *continuous-on-infdist*[*OF continuous-on-subset*[*OF* ‹*continuous-on* {*a..b*} *f*›, *of* {*um..ym*}], *of H*]
‹*ym* ∈ {*um..z*}› ‹*um* ∈ {*a..z*}› ‹*z* ∈ {*a..b*}› **by** *auto*

Choose a point *cm* between $f(um)$ and $f(ym)$ realizing the minimal distance to H. Call this distance *dm*.

have ∃ *closestm* ∈ {*um..ym*}. ∀ *v* ∈ {*um..ym*}. *infdist* (*f closestm*) *H* ≤ *infdist* (*f v*) *H*
apply (*rule continuous-attains-inf*) **using** *ym*(*1*) ∗ **by** *auto*
then obtain *closestm* **where** *closestm*: *closestm* ∈ {*um..ym*} $\bigwedge$*v*. *v* ∈ {*um..ym*} ⟹ *infdist* (*f closestm*) *H* ≤ *infdist* (*f v*) *H*
by *auto*
define *dm* **where** *dm* = *infdist* (*f closestm*) *H*
have [*simp*]: *dm* ≥ *0* **unfolding** *dm-def* **using** *infdist-nonneg* **by** *auto*

Same things but in the interval $[z, uM]$.

have *I*: *dist m* (*f uM*) = *dist* (*f um*) (*f uM*) − *dist* (*f um*) *m*
dist (*f um*) *m* = *Gromov-product-at* (*f um*) (*f z*) (*f uM*)
using *geodesic-segment-dist*[*of* {*f um*−−*f uM*} *f um f uM m*] *m-def* **by** *auto*
have *dist* (*f uM*) (*f z*) ≤ *dist* (*f uM*) (*p uM*) + *dist* (*p uM*) (*f z*)
by (*intro mono-intros*)
also have ... ≤ *dist* (*f uM*) *m* + *dist* (*p uM*) (*f z*)
unfolding *proj-setD*(*2*)[*OF p*[*of uM*]] *H-def* **by** (*auto intro*!: *infdist-le*)
also have ... = *Gromov-product-at* (*f uM*) (*f z*) (*f um*) + *dist* (*p uM*) (*f z*)
using *I* **unfolding** *Gromov-product-at-def* **by** (*simp add*: *divide-simps algebra-simps metric-space-class.dist-commute*)
finally have *A*: *Gromov-product-at* (*f z*) (*f um*) (*f uM*) ≤ *dist* (*p uM*) (*f z*)
unfolding *Gromov-product-at-def* **by** (*simp add*: *metric-space-class.dist-commute divide-simps*)
have *dist* (*p uM*) *pi-z* = *abs*(*dist* (*p uM*) (*f z*) − *dist pi-z* (*f z*))
apply (*rule dist-along-geodesic-wrt-endpoint*[*of H - m*]) **using** *pH* ‹*pi-z* ∈ *H*› *H-def* **by** *auto*
also have ... = *dist* (*p uM*) (*f z*) − *dist pi-z* (*f z*)
using *A Dpi-z* **by** (*simp add*: *metric-space-class.dist-commute*)
finally have *DuM*: *dist* (*p uM*) (*f z*) = *dist* (*p uM*) *pi-z* + *dist pi-z* (*f z*) **by** *auto*

Choose a point $f(yM)$ whose projection on H is roughly at distance L of pi_z.

have $\exists\, yM \in \{z..uM\}$. *dist* (*p uM*) (*p yM*) $\in$ {(*L* + *dist pi-z* (*p uM*)) − *4*∗ *delta* − *2* ∗ *0* .. *L* + *dist pi-z* (*p uM*)}
∧ (∀ *r* ∈ {*yM..uM*}. *dist* (*p uM*) (*p r*) ≤ *L* + *dist pi-z* (*p uM*))
proof (*rule quasi-convex-projection-small-gaps′*[**where** *?f* = *f* **and** *?G* = *H*])
show *continuous-on* {*z..uM*} *f*
apply (*rule continuous-on-subset*[*OF* ‹*continuous-on* {*a..b*} *f*›])
using ‹*uM* ∈ {*z..b*}› ‹*z* ∈ {*a..b*}› **by** *auto*
show *z* ≤ *uM* **using** ‹*uM* ∈ {*z..b*}› **by** *auto*
show *quasiconvex 0 H* **using** *quasiconvex-of-geodesic geodesic-segmentI H* **by** *auto*
show *deltaG TYPE*(*′a*) < *delta* **by** *fact*
have *L* + *dist pi-z* (*p uM*) ≤ *dist* (*f z*) *pi-z* + *dist pi-z* (*p uM*)
using *False Dpi-z* **by** (*simp add*: *metric-space-class.dist-commute*)
then have *L* + *dist pi-z* (*p uM*) ≤ *dist* (*p uM*) (*f z*)
using *DuM* **by** (*simp add*: *metric-space-class.dist-commute*)
then show *L* + *dist pi-z* (*p uM*) ∈ {*4* ∗ *delta* + *2* ∗ *0*..*dist* (*p z*) (*p uM*)}
using ‹*delta* > *0*› *False L-def pz* **by** (*auto simp add*: *metric-space-class.dist-commute*)
show *p yM* ∈ *proj-set* (*f yM*) *H* **for** *yM* **using** *p* **by** *simp*
qed
then obtain *yM* **where** *yM*: *yM* ∈ {*z..uM*}
dist (*p uM*) (*p yM*) ∈ {(*L* + *dist pi-z* (*p uM*)) − *4*∗ *delta* − *2* ∗ *0* .. *L* + *dist pi-z* (*p uM*)}
⋀*r*. *r* ∈ {*yM..uM*} ⟹ *dist* (*p uM*) (*p r*) ≤ *L* + *dist pi-z* (*p uM*)
by *blast*
have ∗: *continuous-on* {*yM..uM*} (λ*r*. *infdist* (*f r*) *H*)
using *continuous-on-infdist*[*OF continuous-on-subset*[*OF* ‹*continuous-on* {*a..b*} *f*›, *of* {*yM..uM*}], *of H*]
‹*yM* ∈ {*z..uM*}› ‹*uM* ∈ {*z..b*}› ‹*z* ∈ {*a..b*}› **by** *auto*
have ∃ *closestM* ∈ {*yM..uM*}. ∀ *v* ∈ {*yM..uM*}. *infdist* (*f closestM*) *H* ≤ *infdist* (*f v*) *H*
apply (*rule continuous-attains-inf*) **using** *yM*(*1*) ∗ **by** *auto*
then obtain *closestM* **where** *closestM*: *closestM* ∈ {*yM..uM*} ⋀*v*. *v* ∈ {*yM..uM*} ⟹ *infdist* (*f closestM*) *H* ≤ *infdist* (*f v*) *H*
by *auto*
define *dM* **where** *dM* = *infdist* (*f closestM*) *H*
have [*simp*]: *dM* ≥ *0* **unfolding** *dM-def* **using** *infdist-nonneg* **by** *auto*

Points between $f(um)$ and $f(ym)$, or between $f(yM)$ and $f(uM)$, project within distance at most L of pi_z by construction.

have *P0*: *dist m* (*p x*) ≤ *dist m pi-z* + *L* **if** *x* ∈ {*um..ym*} ∪ {*yM..uM*} **for** *x*
proof (*cases x* ∈ {*um..ym*})
case *True*
have *dist m* (*f z*) = *dist m* (*p um*) + *dist* (*p um*) *pi-z* + *dist pi-z* (*f z*)
using *geodesic-segment-dist*[*OF H pH*[*of um*]] *Dum* **by** (*simp add*: *metric-space-class.dist-commute*)
moreover have *dist m* (*f z*) = *dist m pi-z* + *dist pi-z* (*f z*)

using *geodesic-segment-dist*[*OF H* ‹*pi-z* ∈ *H*›] **by** (*simp add*: *metric-space-class.dist-commute*)
ultimately have ∗: *dist m pi-z* = *dist m* (*p um*) + *dist* (*p um*) *pi-z* **by** *auto*
have *dist* (*p um*) (*p x*) ≤ *L* + *dist pi-z* (*p um*)
using *ym*(*3*)[*OF* ‹*x* ∈ {*um..ym*}›] **by** *blast*
then show *?thesis*
using *metric-space-class.dist-triangle*[*of m p x p um*] ∗ **by** (*auto simp add*: *metric-space-class.dist-commute*)
next
case *False*
then have *x* ∈ {*yM..uM*} **using** *that* **by** *auto*
have *dist m* (*f z*) = *dist m* (*p uM*) + *dist* (*p uM*) *pi-z* + *dist pi-z* (*f z*)
using *geodesic-segment-dist*[*OF H pH*[*of uM*]] *DuM* **by** (*simp add*: *metric-space-class.dist-commute*)
moreover have *dist m* (*f z*) = *dist m pi-z* + *dist pi-z* (*f z*)
using *geodesic-segment-dist*[*OF H* ‹*pi-z* ∈ *H*›] **by** (*simp add*: *metric-space-class.dist-commute*)
ultimately have ∗: *dist m pi-z* = *dist m* (*p uM*) + *dist* (*p uM*) *pi-z* **by** *auto*
have *dist* (*p uM*) (*p x*) ≤ *L* + *dist pi-z* (*p uM*)
using *yM*(*3*)[*OF* ‹*x* ∈ {*yM..uM*}›] **by** *blast*
then show *?thesis*
using *metric-space-class.dist-triangle*[*of m p x p uM*] ∗ **by** (*auto simp add*: *metric-space-class.dist-commute*)
qed
have *P*: *dist pi-z* (*p x*) ≤ *L* **if** *x* ∈ {*um..ym*} ∪ {*yM..uM*} **for** *x*
proof (*cases dist m* (*p x*) ≤ *dist pi-z m*)
case *True*
have *dist pi-z* (*p x*) ≤ *dist pi-z m* + *dist m* (*p x*)
by (*intro mono-intros*)
also have ... ≤ *2* ∗ *delta* + *2* ∗ *delta*
using ‹*dist pi-z m* ≤ *2* ∗ *delta*› *True* **by** *auto*
finally show *?thesis*
using *Laux* ‹*delta* > *0*› **by** *auto*
next
case *False*
have *dist pi-z* (*p x*) = *abs*(*dist pi-z m* − *dist* (*p x*) *m*)
apply (*rule dist-along-geodesic-wrt-endpoint*[*OF geodesic-segment-commute*[*OF H*]])
using *pH* ‹*pi-z* ∈ *H*› **by** *auto*
also have ... = *dist* (*p x*) *m* − *dist pi-z m*
using *False* **by** (*simp add*: *metric-space-class.dist-commute*)
finally show *?thesis*
using *P0*[*OF that*] **by** (*simp add*: *metric-space-class.dist-commute*)
qed

Auxiliary fact for later use: The distance between two points in $[um, ym]$ and $[yM, uM]$ can be controlled using the distances of their images under f to H, thanks to the quasi-isometry property.

have *D*: *dist rm rM* ≤ *lambda* ∗ (*infdist* (*f rm*) *H* + (*L* + *C* + *2* ∗ *delta*) + *infdist* (*f rM*) *H*)
if *rm* ∈ {*um..ym*} *rM* ∈ {*yM..uM*} **for** *rm rM*
proof −
have ∗: *dist m* (*p rm*) ≤ *L* + *dist m pi-z dist m* (*p rM*) ≤ *L* + *dist m pi-z*
using *P0 that* **by** *force*+
have *dist* (*p rm*) (*p rM*) = *abs*(*dist* (*p rm*) *m* − *dist* (*p rM*) *m*)
apply (*rule dist-along-geodesic-wrt-endpoint*[*OF geodesic-segment-commute*[*OF H*]])
using *pH* **by** *auto*
also have ... ≤ *L* + *dist m pi-z*
unfolding *abs-le-iff* **using** ∗ **apply** (*auto simp add*: *metric-space-class.dist-commute*)
by (*metis diff-add-cancel le-add-same-cancel1 metric-space-class.zero-le-dist order-trans*)+
finally have ∗: *dist* (*p rm*) (*p rM*) ≤ *L* + *2* ∗ *delta*
using ‹*dist pi-z m* ≤ *2* ∗ *delta*› **by** (*simp add*: *metric-space-class.dist-commute*)

have (*1*/*lambda*) ∗ *dist rm rM* − *C* ≤ *dist* (*f rm*) (*f rM*)
apply (*rule quasi-isometry-onD*(*2*)[*OF* ‹*lambda C*−*quasi-isometry-on* {*a..b*} *f*›])
using ‹*rm* ∈ {*um..ym*}› ‹*ym* ∈ {*um..z*}› ‹*um* ∈ {*a..z*}› ‹*z* ∈ {*a..b*}› ‹*rM* ∈ {*yM..uM*}› ‹*yM* ∈ {*z..uM*}› ‹*uM* ∈ {*z..b*}› **by** *auto*
also have ... ≤ *dist* (*f rm*) (*p rm*) + *dist* (*p rm*) (*p rM*) + *dist* (*p rM*) (*f rM*)
by (*intro mono-intros*)
also have ... ≤ *infdist* (*f rm*) *H* + *L* + *2* ∗ *delta* + *infdist* (*f rM*) *H*
using ∗ *proj-setD*(*2*)[*OF p*] **by** (*simp add*: *metric-space-class.dist-commute*)
finally show *?thesis*
using ‹*lambda* ≥ *1*› **by** (*simp add*: *algebra-simps divide-simps*)
qed

Auxiliary fact for later use in the inductive argument: the distance from $f(z)$ to pi_z is controlled by the distance from $f(z)$ to any intermediate geodesic between points in $f[um, ym]$ and $f[yM, uM]$, up to a constant essentially given by L. This is a variation around Lemma 5 in [Shc13].

have *Rec*: *Gromov-product-at* (*f z*) (*f um*) (*f uM*) ≤ *Gromov-product-at* (*f z*) (*f rm*) (*f rM*) + (*L* + *4* ∗ *delta*) **if** *rm* ∈ {*um..ym*} *rM* ∈ {*yM..uM*} **for** *rm rM*
proof −
have ∗: *dist* (*f rm*) (*p rm*) + *dist* (*p rm*) (*f z*) ≤ *dist* (*f rm*) (*f z*) + *4* ∗ *deltaG*(*TYPE*(*'a*))
apply (*rule dist-along-geodesic*[*of H*]) **using** *p H-def* **by** *auto*
have *dist* (*f z*) *pi-z* ≤ *dist* (*f z*) (*p rm*) + *dist* (*p rm*) *pi-z*
by (*intro mono-intros*)
also have ... ≤ (*Gromov-product-at* (*f z*) (*f rm*) (*p rm*) + *2* ∗ *deltaG*(*TYPE*(*'a*))) + *L*
apply (*intro mono-intros*) **using** ∗ *P* ‹*rm* ∈ {*um..ym*}› **unfolding** *Gromov-product-at-def*
by (*auto simp add*: *metric-space-class.dist-commute algebra-simps divide-simps*)

finally have *A*: *dist* (*f z*) *pi-z* − *L* − *2* ∗ *deltaG*(*TYPE*(*'a*)) ≤ *Gromov-product-at* (*f z*) (*f rm*) (*p rm*)
by *simp*
have ∗: *dist* (*f rM*) (*p rM*) + *dist* (*p rM*) (*f z*) ≤ *dist* (*f rM*) (*f z*) + *4* ∗ *deltaG*(*TYPE*(*'a*))
apply (*rule dist-along-geodesic*[*of H*]) **using** *p H-def* **by** *auto*
have *dist* (*f z*) *pi-z* ≤ *dist* (*f z*) (*p rM*) + *dist* (*p rM*) *pi-z*
by (*intro mono-intros*)
also have ... ≤ (*Gromov-product-at* (*f z*) (*p rM*) (*f rM*) + *2* ∗ *deltaG*(*TYPE*(*'a*))) + *L*
apply (*intro mono-intros*) **using** ∗ *P* ‹*rM* ∈ {*yM*..*uM*}› **unfolding** *Gromov-product-at-def*
by (*auto simp add*: *metric-space-class.dist-commute algebra-simps divide-simps*)
finally have *B*: *dist* (*f z*) *pi-z* − *L* − *2* ∗ *deltaG*(*TYPE*(*'a*)) ≤ *Gromov-product-at* (*f z*) (*p rM*) (*f rM*)
by *simp*
have *C*: *dist* (*f z*) *pi-z* − *L* − *2* ∗ *deltaG*(*TYPE*(*'a*)) ≤ *Gromov-product-at* (*f z*) (*p rm*) (*p rM*)
proof (*cases dist* (*f z*) (*p rm*) ≤ *dist* (*f z*) (*p rM*))
case *True*
have *dist* (*p rm*) (*p rM*) = *abs*(*dist* (*f z*) (*p rm*) − *dist* (*f z*) (*p rM*))
using *proj-setD*(*1*)[*OF p*] *dist-along-geodesic-wrt-endpoint*[*OF H*, *of p rm p rM*]
by (*simp add*: *metric-space-class.dist-commute*)
also have ... = *dist* (*f z*) (*p rM*) − *dist* (*f z*) (*p rm*)
using *True* **by** *auto*
finally have ∗: *dist* (*f z*) (*p rm*) = *Gromov-product-at* (*f z*) (*p rm*) (*p rM*)
unfolding *Gromov-product-at-def* **by** *auto*
have *dist* (*f z*) *pi-z* ≤ *dist* (*f z*) (*p rm*) + *dist* (*p rm*) *pi-z*
by (*intro mono-intros*)
also have ... ≤ *Gromov-product-at* (*f z*) (*p rm*) (*p rM*) + *L* + *2* ∗ *deltaG*(*TYPE*(*'a*))
using ∗ *P*[*of rm*] ‹*rm* ∈ {*um*..*ym*}› **apply** (*simp add*: *metric-space-class.dist-commute*)
using *local.delta-nonneg* **by** *linarith*
finally show *?thesis* **by** *simp*
next
case *False*
have *dist* (*p rm*) (*p rM*) = *abs*(*dist* (*f z*) (*p rm*) − *dist* (*f z*) (*p rM*))
using *proj-setD*(*1*)[*OF p*] *dist-along-geodesic-wrt-endpoint*[*OF H*, *of p rm p rM*]
by (*simp add*: *metric-space-class.dist-commute*)
also have ... = *dist* (*f z*) (*p rm*) − *dist* (*f z*) (*p rM*)
using *False* **by** *auto*
finally have ∗: *dist* (*f z*) (*p rM*) = *Gromov-product-at* (*f z*) (*p rm*) (*p rM*)
unfolding *Gromov-product-at-def* **by** *auto*
have *dist* (*f z*) *pi-z* ≤ *dist* (*f z*) (*p rM*) + *dist* (*p rM*) *pi-z*
by (*intro mono-intros*)
also have ... ≤ *Gromov-product-at* (*f z*) (*p rm*) (*p rM*) + *L* + *2* ∗

deltaG(*TYPE*(*'a*))
using ∗ *P*[*of rM*] ‹*rM* ∈ {*yM*..*uM*}› **apply** (*simp add*: *metric-space-class.dist-commute*)
using *local.delta-nonneg* **by** *linarith*
finally show *?thesis* **by** *simp*
qed

have *Gromov-product-at* (*f z*) (*f um*) (*f uM*) − *L* − *2* ∗ *deltaG*(*TYPE*(*'a*)) ≤ *Min* {*Gromov-product-at* (*f z*) (*f rm*) (*p rm*), *Gromov-product-at* (*f z*) (*p rm*) (*p rM*), *Gromov-product-at* (*f z*) (*p rM*) (*f rM*)}
using *A B C* **unfolding** *Dpi-z* **by** *auto*
also have ... ≤ *Gromov-product-at* (*f z*) (*f rm*) (*f rM*) + *2* ∗ *deltaG*(*TYPE*(*'a*))
by (*intro mono-intros*)
finally show *?thesis*
using ‹*deltaG*(*TYPE*(*'a*)) < *delta*› **by** *auto*
qed

We have proved the basic facts we will need in the main argument. This argument starts here. It is divided in several cases.

consider *dm* ≤ *D* + *4* ∗ *C* ∧ *dM* ≤ *D* + *4* ∗ *C* | *dm* ≥ *D* + *4* ∗ *C* ∧ *dM* ≤ *dm* | *dM* ≥ *D* + *4* ∗ *C* ∧ *dm* ≤ *dM*
by *linarith*
then show *?thesis*
proof (*cases*)

Case 2.1 of the description before the statement: there are points in $f[um, ym]$ and in $f[yM, uM]$ which are close to H. Then one can conclude directly, without relying on the inductive argument, thanks to the quasi-isometry property.

case *1*
have *I*: *Gromov-product-at* (*f z*) (*f closestm*) (*f closestM*) ≤ *lambda*^*2* ∗ (*D* + *L* / *2* + *delta* + *11*/*2* ∗ *C*) − *6* ∗ *delta*
proof (*cases dist* (*f closestm*) (*f closestM*) ≤ *12* ∗ *delta*)
case *True*
have *1*/*lambda* ∗ *dist closestm closestM* − *C* ≤ *dist* (*f closestm*) (*f closestM*)
using *quasi-isometry-onD*(*2*)[*OF assms*(*2*)] ‹*closestm* ∈ {*um*..*ym*}› ‹*um* ∈ {*a*..*z*}› ‹*z* ∈ {*a*..*b*}› ‹*ym* ∈ {*um*..*z*}›
‹*closestM* ∈ {*yM*..*uM*}› ‹*uM* ∈ {*z*..*b*}› ‹*z* ∈ {*a*..*b*}› ‹*yM* ∈ {*z*..*uM*}› **by** *auto*
then have *dist closestm closestM* ≤ *lambda* ∗ *dist* (*f closestm*) (*f closestM*) + *lambda* ∗ *C*
using ‹*lambda* ≥ *1*› **by** (*auto simp add*: *divide-simps algebra-simps*)
also have ... ≤ *lambda* ∗ (*12* ∗ *delta*) + *lambda* ∗ *C*
apply (*intro mono-intros True*) **using** ‹*lambda* ≥ *1*› **by** *auto*
finally have *M*: *dist closestm closestM* ≤ *lambda* ∗ (*12* ∗ *delta* + *C*)
by (*auto simp add*: *algebra-simps*)

have *2* ∗ *Gromov-product-at* (*f z*) (*f closestm*) (*f closestM*) ≤ *dist* (*f closestm*) (*f z*) + *dist* (*f z*) (*f* (*closestM*))

unfolding *Gromov-product-at-def* **by** (*auto simp add*: *metric-space-class.dist-commute*)
also have ... $\leq$ (*lambda* $*$ *dist closestm z* + *C*) + (*lambda* $*$ *dist z closestM* + *C*)
apply (*intro mono-intros quasi-isometry-onD*(*1*)[*OF assms*(*2*)])
using ‹*closestm* $\in$ {*um..ym*}› ‹*um* $\in$ {*a..z*}› ‹*z* $\in$ {*a..b*}› ‹*ym* $\in$ {*um..z*}› ‹*closestM* $\in$ {*yM..uM*}› ‹*uM* $\in$ {*z..b*}› ‹*z* $\in$ {*a..b*}› ‹*yM* $\in$ {*z..uM*}› **by** *auto*
also have ... = *lambda* $*$ *dist closestm closestM* + *1* $*$ *2* $*$ *C*
unfolding *dist-real-def* **using** ‹*closestm* $\in$ {*um..ym*}› ‹*um* $\in$ {*a..z*}› ‹*z* $\in$ {*a..b*}› ‹*ym* $\in$ {*um..z*}›
‹*closestM* $\in$ {*yM..uM*}› ‹*uM* $\in$ {*z..b*}› ‹*z* $\in$ {*a..b*}› ‹*yM* $\in$ {*z..uM*}› **by** (*auto simp add*: *algebra-simps*)
also have ... $\leq$ *lambda* $*$ (*lambda* $*$ (*12* $*$ *delta* + *C*)) + *lambda*^*2* $*$ *2* $*$ *C*
apply (*intro mono-intros M*) **using** ‹*lambda* $\geq$ *1*› ‹*C* $\geq$ *0*› **by** *auto*
also have ... = *lambda*^*2* $*$ (*24* $*$ *delta* + *3* $*$ *C*) − *lambda*^*2* $*$ *12* $*$ *delta*
by (*simp add*: *algebra-simps power2-eq-square*)
also have ... $\leq$ *lambda*^*2* $*$ ((*2* $*$ *D* + *L* + *2* $*$ *delta*) + *11* $*$ *C*) − *1* $*$ *12* $*$ *delta*
apply (*intro mono-intros*) **using** *Laux* ‹*lambda* $\geq$ *1*› ‹*C* $\geq$ *0*› ‹*delta* > *0*› **by** *auto*
finally show *?thesis*
by (*auto simp add*: *divide-simps algebra-simps*)
next
case *False*
have *dist closestm closestM* $\leq$ *lambda* $*$ (*dm* + *dM* + *L* + *2* $*$ *delta* + *C*)
using *D*[*OF* ‹*closestm* $\in$ {*um..ym*}› ‹*closestM* $\in$ {*yM..uM*}›] *dm-def dM-def* **by** (*auto simp add*: *algebra-simps*)
also have ... $\leq$ *lambda* $*$ ((*D* + *4* $*$ *C*) + (*D* + *4* $*$ *C*) + *L* + *2* $*$ *delta* + *C*)
apply (*intro mono-intros*) **using** *1* ‹*lambda* $\geq$ *1*› **by** *auto*
also have ... $\leq$ *lambda* $*$ (*2* $*$ *D* + *L* + *2* $*$ *delta* + *9* $*$ *C*)
using ‹*lambda* $\geq$ *1*› ‹*C* $\geq$ *0*› **by** *auto*
finally have *M*: *dist closestm closestM* $\leq$ *lambda* $*$ (*2* $*$ *D* + *L* + *2* $*$ *delta* + *9* $*$ *C*)
by (*auto simp add*: *algebra-simps divide-simps metric-space-class.dist-commute*)

have *dist* (*f closestm*) (*f z*) + *dist* (*f z*) (*f* (*closestM*)) $\leq$ (*lambda* $*$ *dist closestm z* + *C*) + (*lambda* $*$ *dist z closestM* + *C*)
apply (*intro mono-intros quasi-isometry-onD*(*1*)[*OF assms*(*2*)])
using ‹*closestm* $\in$ {*um..ym*}› ‹*um* $\in$ {*a..z*}› ‹*z* $\in$ {*a..b*}› ‹*ym* $\in$ {*um..z*}› ‹*closestM* $\in$ {*yM..uM*}› ‹*uM* $\in$ {*z..b*}› ‹*z* $\in$ {*a..b*}› ‹*yM* $\in$ {*z..uM*}› **by** *auto*
also have ... = *lambda* $*$ *dist closestm closestM* + *1* $*$ *2* $*$ *C*
unfolding *dist-real-def* **using** ‹*closestm* $\in$ {*um..ym*}› ‹*um* $\in$ {*a..z*}› ‹*z* $\in$ {*a..b*}› ‹*ym* $\in$ {*um..z*}›
‹*closestM* $\in$ {*yM..uM*}› ‹*uM* $\in$ {*z..b*}› ‹*z* $\in$ {*a..b*}› ‹*yM* $\in$ {*z..uM*}› **by** (*auto simp add*: *algebra-simps*)
also have ... $\leq$ *lambda* $*$ (*lambda* $*$ (*2* $*$ *D* + *L* + *2* $*$ *delta* + *9* $*$ *C*)) +

lambda^2 ∗ 2 ∗ C
apply (*intro mono-intros M*) **using** ‹*lambda* ≥ *1*› ‹*C* ≥ *0*› **by** *auto*
finally have *dist* (*f closestm*) (*f z*) + *dist* (*f z*) (*f closestM*) ≤ *lambda^2* ∗ (*2* ∗ *D* + *L* + *2* ∗ *delta* + *11* ∗ *C*)
by (*simp add*: *algebra-simps power2-eq-square*)
then show *?thesis*
unfolding *Gromov-product-at-def* **using** *False* **by** (*simp add*: *metric-space-class.dist-commute algebra-simps divide-simps*)
qed
have *Gromov-product-at* (*f z*) (*f um*) (*f uM*) ≤ *Gromov-product-at* (*f z*) (*f closestm*) (*f closestM*) + *1* ∗ *L* + *4* ∗ *delta* + *0* ∗ (*1* − *exp* (− *K* ∗ (*uM* − *um*)))
using *Rec*[*OF* ‹*closestm* ∈ {*um..ym*}› ‹*closestM* ∈ {*yM..uM*}›] **by** *simp*
also have ... ≤ (*lambda^2* ∗ (*D* + *L* / *2* + *delta* + *11*/*2* ∗ *C*) − *6* ∗ *delta*) + *lambda^2* ∗ *L* + *4* ∗ *delta* + *Kmult* ∗ (*1* − *exp* (− *K* ∗ (*uM* − *um*)))
apply (*intro mono-intros I*)
using *Laux* ‹*lambda* ≥ *1*› ‹*delta* > *0*› ‹*Kmult* > *0*› ‹*um* ∈ {*a..z*}› ‹*uM* ∈ {*z..b*}› ‹*K* > *0*› **by** *auto*
finally show *?thesis*
by (*simp add*: *algebra-simps*)

End of the easy case 2.1

next

Case 2.2: dm is large, i.e., all points in $f[um, ym]$ are far away from H. Moreover, assume that $dm \geq dM$. Then we will find a pair of points v and x with $um \leq v \leq x \leq ym$ satisfying the estimate (1). We argue by induction: while we have not found such a pair, we can find a point x_k whose projection on V_k, the neighborhood of size $(2^k - 1)dm$ of H, is far enough from the projection of um, and such that all points in between are far enough from V_k so that the corresponding projection will have good contraction properties.

case *2*
then have *I*: *D* + *4* ∗ *C* ≤ *dm dM* ≤ *dm* **by** *auto*
define *V* **where** *V* = (λ*k*::*nat*. (⋃*g*∈*H*. *cball g* ((*2^k* − *1*) ∗ *dm*)))
define *QC* **where** *QC* = (λ*k*::*nat*. *if k* = *0 then 0 else 8* ∗ *delta*)
have *QC k* ≥ *0* **for** *k* **unfolding** *QC-def* **using** ‹*delta* > *0*› **by** *auto*
have *Q*: *quasiconvex* (*0* + *8* ∗ *deltaG*(*TYPE*(*'a*))) (*V k*) **for** *k*
unfolding *V-def* **apply** (*rule quasiconvex-thickening*) **using** *geodesic-segmentI*[*OF H*]
by (*auto simp add*: *quasiconvex-of-geodesic*)
have *quasiconvex* (*QC k*) (*V k*) **for** *k*
apply (*cases k* = *0*)
apply (*simp add*: *V-def QC-def quasiconvex-of-geodesic geodesic-segmentI*[*OF H*])
apply (*rule quasiconvex-mono*[*OF* - *Q*[*of k*]]) **using** ‹*deltaG*(*TYPE*(*'a*)) < *delta*› *QC-def* **by** *auto*

Define $q(k, x)$ to be the projection of $f(x)$ on V_k.

define *q*::*nat* ⇒ *real* ⇒ *'a* **where** *q* = (λ*k x*. *geodesic-segment-param* {*p* *x−−f x*} (*p x*) ((*2^k* − *1*) ∗ *dm*))

The inductive argument

have *Ind-k*: (*Gromov-product-at* (*f z*) (*f um*) (*f uM*) ≤ *lambda^2* ∗ (*D* + *3/2* ∗ *L* + *delta* + *11/2* ∗ *C*) − *2* ∗ *delta* + *Kmult* ∗ (*1* − *exp*(− *K* ∗ (*uM* − *um*))))
∨ (∃ *x* ∈ {*um..ym*}. (∀ *w* ∈ {*um..x*}. *dist* (*f w*) (*p w*) ≥ (*2^*(*k+1*)−*1*) ∗ *dm*) ∧ *dist* (*q k um*) (*q k x*) ≥ *L* − *4* ∗ *delta* + *7* ∗ *QC k*) **for** *k*
proof (*induction k*)

Base case: there is a point far enough from $q0um$ on H. This is just the point ym, by construction.

case *0*
have ∗: ∃ *x*∈ {*um..ym*}. (∀ *w* ∈ {*um..x*}. *dist* (*f w*) (*p w*) ≥ (*2^*(*0+1*)−*1*) ∗ *dm*) ∧ *dist* (*q 0 um*) (*q 0 x*) ≥ *L* − *4* ∗ *delta* + *7* ∗ *QC 0*
proof (*rule bexI*[*of* - *ym*], *auto simp add*: *V-def q-def QC-def*)
show *um* ≤ *ym* **using** ‹*ym* ∈ {*um..z*}› **by** *auto*
show *L* − *4* ∗ *delta* ≤ *dist* (*p um*) (*p ym*)
using *ym*(*2*) **apply** *auto* **using** *metric-space-class.zero-le-dist*[*of pi-z p um*] **by** *linarith*
show ⋀*y*. *um* ≤ *y* ⟹ *y* ≤ *ym* ⟹ *dm* ≤ *dist* (*f y*) (*p y*)
using *dm-def closestm proj-setD*(*2*)[*OF p*] **by** *auto*
qed
then show *?case*
by *blast*
next

The induction. The inductive assumption claims that, either the desired inequality holds, or one can construct a point with good properties. If the desired inequality holds, there is nothing left to prove. Otherwise, we can start from this point at step k, say x, and either prove the desired inequality or construct a point with the good properties at step $k + 1$.

case *Suck*: (*Suc k*)
show *?case*
proof (*cases Gromov-product-at* (*f z*) (*f um*) (*f uM*) ≤ *lambda*^{2} ∗ (*D* + *3/2* ∗ *L* + *delta* + *11/2* ∗ *C*) − *2* ∗ *delta* + *Kmult* ∗ (*1* − *exp* (− *K* ∗ (*uM* − *um*))))
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then obtain *x* **where** *x*: *x* ∈ {*um..ym*} *dist* (*q k um*) (*q k x*) ≥ *L* − *4* ∗ *delta* + *7* ∗ *QC k*
⋀*w*. *w* ∈ {*um..x*} ⟹ *dist* (*f w*) (*p w*) ≥ (*2^*(*k+1*)−*1*) ∗ *dm*
using *Suck.IH* **by** *auto*

Some auxiliary technical inequalities to be used later on.

have *aux*: $(2 \,\widehat{}\, k - 1) * dm \leq (2{*}2\widehat{}k{-}1) * dm$ $0 \leq 2 * 2 \,\widehat{}\, k - (1::real)$ $dm \leq dm * 2 \,\widehat{}\, k$
apply (*auto simp add*: *algebra-simps*)
apply (*metis power.simps*(*2*) *two-realpow-ge-one*)
using ‹$0 \leq dm$› *less-eq-real-def* **by** *fastforce*
have $L + C = (L/D) * (D + (D/L) * C)$
using ‹$L > 0$› ‹$D > 0$› **by** (*simp add*: *algebra-simps divide-simps*)
also have ... $\leq (L/D) * (D + 4 * C)$
apply (*intro mono-intros*)
using ‹$L > 0$› ‹$D > 0$› ‹$C \geq 0$› ‹$D \leq 4 * L$› **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... $\leq (L/D) * dm$
apply (*intro mono-intros*) **using** *I* ‹$L > 0$› ‹$D > 0$› **by** *auto*
finally have $L + C \leq (L/D) * dm$
by *simp*
moreover have $2 * delta \leq (2 * delta)/D * dm$
using *I* ‹$C \geq 0$› ‹$delta > 0$› ‹$D > 0$› **by** (*auto simp add*: *algebra-simps divide-simps*)
ultimately have *aux2*: $L + C + 2 * delta \leq ((L + 2 * delta)/D) * dm$
by (*auto simp add*: *algebra-simps divide-simps*)
have *aux3*: $(1{-}alpha) * D + alpha * 2\widehat{}k * dm \leq dm * 2\widehat{}k - C/2 - QC\ k$
proof (*cases* $k = 0$)
case *True*
show *?thesis*
using *I* ‹$C \geq 0$› **unfolding** *True QC-def alpha-def* **by** *auto*
next
case *False*
have $C/2 + QC\ k + (1{-}alpha) * D \leq 2 * (1{-}alpha) * dm$
using *I* ‹$C \geq 0$› **unfolding** *QC-def alpha-def* **using** *False Laux* **by** *auto*
also have ... $\leq 2\widehat{}k * (1{-}alpha) * dm$
apply (*intro mono-intros*) **using** *False alphaaux I* ‹$D > 0$› ‹$C \geq 0$› **by** *auto*
finally show *?thesis*
by (*simp add*: *algebra-simps*)
qed

Construct a point w such that its projection on V_k is close to that of um and therefore far away from that of x. This is just the intermediate value theorem (with some care as the closest point projection is not continuous).

have $\exists\, w \in \{um..x\}.\ (dist\ (q\ k\ um)\ (q\ k\ w) \in \{(9 * delta + 4 * QC\ k) - 4 * delta - 2 * QC\ k\ ..\ 9 * delta + 4 * QC\ k\})$
$\wedge\ (\forall\, v \in \{um..w\}.\ dist\ (q\ k\ um)\ (q\ k\ v) \leq 9 * delta + 4 * QC\ k)$
proof (*rule quasi-convex-projection-small-gaps*[**where** *?f* = *f* **and** *?G* = *V k*])
show *continuous-on* $\{um..x\}$ *f*
apply (*rule continuous-on-subset*[*OF* ‹*continuous-on* $\{a..b\}$ *f*›])
using ‹$um \in \{a..z\}$› ‹$z \in \{a..b\}$› ‹$ym \in \{um..z\}$› ‹$x \in \{um..ym\}$›

```
by auto
        show um ≤ x using ‹x ∈ {um..ym}› by auto
        show quasiconvex (QC k) (V k) by fact
        show deltaG TYPE('a) < delta by fact
        show 9 * delta + 4 * QC k ∈ {4 * delta + 2 * QC k..dist (q k um)
(q k x)}
          using x(2) ‹delta > 0› ‹QC k ≥ 0› Laux by auto
        show q k w ∈ proj-set (f w) (V k) if w ∈ {um..x} for w
          unfolding V-def q-def apply (rule proj-set-thickening)
      using aux p x(3)[OF that] by (auto simp add: metric-space-class.dist-commute)
      qed
      then obtain w where w: w ∈ {um..x}
                          dist (q k um) (q k w) ∈ {(9 * delta + 4 * QC k) − 4
* delta − 2 * QC k .. 9 * delta + 4 * QC k}
                          ⋀v. v ∈ {um..w} ⟹ dist (q k um) (q k v) ≤ 9 * delta
+ 4 * QC k
        by auto
```

There are now two cases to be considered: either one can find a point v between um and w which is close enough to H. Then this point will satisfy (1), and we will be able to prove the desired inequality. Or there is no such point, and then w will have the good properties at step $k+1$

```
      show ?thesis
      proof (cases ∃ v ∈ {um..w}. dist (f v) (p v) ≤ (2^(k+2)−1) * dm)
        case True
```

First subcase: there is a good point v between um and w. This is the heart of the argument: we will show that the desired inequality holds.

```
        then obtain v where v: v ∈ {um..w} dist (f v) (p v) ≤ (2^(k+2)−1)
* dm
          by auto
```

Auxiliary basic fact to be used later on.

```
        have aux4: dm * 2 ^ k ≤ infdist (f r) (V k) if r ∈ {v..x} for r
        proof −
          have *: q k r ∈ proj-set (f r) (V k)
            unfolding q-def V-def apply (rule proj-set-thickening)
            using aux p[of r] x(3)[of r] that ‹v ∈ {um..w}› ‹w ∈ {um..x}› by
(auto simp add: metric-space-class.dist-commute)
          have infdist (f r) (V k) = dist (geodesic-segment-param {p r−−f r}
(p r) (dist (p r) (f r))) (geodesic-segment-param {p r−−f r} (p r) ((2 ^ k − 1) *
dm))
            using proj-setD(2)[OF *] unfolding q-def by auto
          also have ... = abs(dist (p r) (f r) − (2 ^ k − 1) * dm)
            apply (rule geodesic-segment-param(7)[where ?y = f r])
            using x(3)[of r] ‹r ∈ {v..x}› ‹v ∈ {um..w}› ‹w ∈ {um..x}› aux by
(auto simp add: metric-space-class.dist-commute)
          also have ... = dist (f r) (p r) − (2 ^ k − 1) * dm
```

using *x(3)[of r]* ‹*r* ∈ {*v..x*}› ‹*v* ∈ {*um..w*}› ‹*w* ∈ {*um..x*}› *aux* **by** (*auto simp add*: *metric-space-class.dist-commute*)
finally have *dist (f r) (p r) = infdist (f r) (V k) + (2 ^ k − 1) ∗ dm* **by** *simp*
moreover have *(2^(k+1) − 1) ∗ dm ≤ dist (f r) (p r)*
apply (*rule x(3)*) **using** ‹*r* ∈ {*v..x*}› ‹*v* ∈ {*um..w*}› ‹*w* ∈ {*um..x*}› **by** *auto*
ultimately have *(2^(k+1) − 1) ∗ dm ≤ infdist (f r) (V k) + (2 ^ k − 1) ∗ dm*
by *simp*
then show *?thesis* **by** (*auto simp add*: *algebra-simps*)
qed

Substep 1: We can control the distance from $f(v)$ to $f(closestM)$ in terms of the distance of the distance of $f(v)$ to H, i.e., by $2^k dm$. The same control follows for $closestM - v$ thanks to the quasi-isometry property. Then, we massage this inequality to put it in the form we will need, as an upper bound on $(x - v)\exp(-2^k dm)$.

have *infdist (f v) H ≤ (2^(k+2)−1) ∗ dm*
using *v proj-setD(2)[OF p[of v]]* **by** *auto*
have *dist v closestM ≤ lambda ∗ (infdist (f v) H + (L + C + 2 ∗ delta) + infdist (f closestM) H)*
apply (*rule D*)
using ‹*v* ∈ {*um..w*}› ‹*w* ∈ {*um..x*}› ‹*x* ∈ {*um..ym*}› ‹*ym* ∈ {*um..z*}› ‹*um* ∈ {*a..z*}› ‹*z* ∈ {*a..b*}› ‹*closestM* ∈ {*yM..uM*}› ‹*yM* ∈ {*z..uM*}› ‹*uM* ∈ {*z..b*}› **by** *auto*
also have ... ≤ *lambda ∗ ((2^(k+2)−1) ∗ dm + 1 ∗ (L + C + 2 ∗ delta) + dM)*
apply (*intro mono-intros* ‹*infdist (f v) H ≤ (2^(k+2)−1) ∗ dm*›)
using *dM-def* ‹*lambda* ≥ *1*› ‹*L* > *0*› ‹*C* ≥ *0*› ‹*delta* > *0*› **by** (*auto simp add*: *metric-space-class.dist-commute*)
also have ... ≤ *lambda ∗ ((2^(k+2)−1) ∗ dm + 2^k ∗ (((L + 2 ∗ delta)/D) ∗ dm) + dm)*
apply (*intro mono-intros*) **using** *I* ‹*lambda* ≥ *1*› ‹*C* ≥ *0*› ‹*delta* > *0*› ‹*L* > *0*› *aux2* **by** *auto*
also have ... = *lambda ∗ 2^k ∗ (4 + (L + 2 ∗ delta)/D) ∗ dm*
by (*simp add*: *algebra-simps*)
finally have ∗: *dist v closestM / (lambda ∗ (4 + (L + 2 ∗ delta)/D)) ≤ 2^k ∗ dm*
using ‹*lambda* ≥ *1*› ‹*L* > *0*› ‹*D* > *0*› ‹*delta* > *0*› **by** (*simp add*: *divide-simps*, *simp add*: *algebra-simps*)

We reformulate this control inside of an exponential, as this is the form we will use later on.

have *exp(− (alpha ∗ (2^k ∗ dm) ∗ ln 2 / (5 ∗ delta))) ≤ exp(−(alpha ∗ (dist v closestM / (lambda ∗ (4 + (L + 2 ∗ delta)/D))) ∗ ln 2 / (5 ∗ delta)))*
apply (*intro mono-intros* ∗) **using** *alphaaux* ‹*delta* > *0*› **by** *auto*
also have ... = *exp(−K ∗ dist v closestM)*

unfolding *K-def* **by** (*simp add*: *divide-simps*)
also have ... = *exp*(−*K* ∗ (*closestM* − *v*))
unfolding *dist-real-def* **using** ‹*v* ∈ {*um*..*w*}› ‹*w* ∈ {*um*..*x*}› ‹*x* ∈ {*um*..*ym*}› ‹*ym* ∈ {*um*..*z*}› ‹*yM* ∈ {*z*..*uM*}› ‹*closestM* ∈ {*yM*..*uM*}› ‹*K* > *0*› **by** *auto*
finally have *exp*(− (*alpha* ∗ (*2*^*k* ∗ *dm*) ∗ *ln 2* / (*5* ∗ *delta*))) ≤ *exp*(−*K* ∗ (*closestM* − *v*))
by *simp*

Plug in $x - v$ to get the final form of this inequality.

then have *K* ∗ (*x* − *v*) ∗ *exp*(− (*alpha* ∗ (*2*^*k* ∗ *dm*) ∗ *ln 2* / (*5* ∗ *delta*))) ≤ *K* ∗ (*x* − *v*) ∗ *exp*(−*K* ∗ (*closestM* − *v*))
apply (*rule mult-left-mono*)
using ‹*delta* > *0*› ‹*lambda* ≥ *1*› ‹*v* ∈ {*um*..*w*}› ‹*w* ∈ {*um*..*x*}› ‹*K* > *0*› **by** *auto*
also have ... = ((*1* + *K* ∗ (*x* − *v*)) − *1*) ∗ *exp*(− *K* ∗ (*closestM* − *v*))
by (*auto simp add*: *algebra-simps*)
also have ... ≤ (*exp* (*K* ∗ (*x* − *v*)) − *1*) ∗ *exp*(−*K* ∗ (*closestM* − *v*))
by (*intro mono-intros*, *auto*)
also have ... = *exp*(−*K* ∗ (*closestM* − *x*)) − *exp*(−*K* ∗ (*closestM* − *v*))
by (*simp add*: *algebra-simps mult-exp-exp*)
also have ... ≤ *exp*(−*K* ∗ (*closestM* − *x*)) − *exp*(−*K* ∗ (*uM* − *um*))
using ‹*K* > *0*› ‹*v* ∈ {*um*..*w*}› ‹*w* ∈ {*um*..*x*}› ‹*x* ∈ {*um*..*ym*}› ‹*ym* ∈ {*um*..*z*}› ‹*yM* ∈ {*z*..*uM*}› ‹*closestM* ∈ {*yM*..*uM*}› **by** *auto*
finally have *B*: (*x* − *v*) ∗ *exp*(− *alpha* ∗ *2*^*k* ∗ *dm* ∗ *ln 2* / (*5* ∗ *delta*)) ≤
(*exp*(−*K* ∗ (*closestM* − *x*)) − *exp*(−*K* ∗ (*uM*−*um*)))/*K*
using ‹*K* > *0*› **by** (*auto simp add*: *divide-simps algebra-simps*)

End of substep 1

Substep 2: The projections of $f(v)$ and $f(x)$ on the cylinder V_k are well separated, by construction. This implies that v and x themselves are well separated, thanks to the exponential contraction property of the projection on the quasi-convex set V_k. This leads to a uniform lower bound for $(x - v)\exp(-2^k dm)$, which has been upper bounded in Substep 1.

have *L* − *4* ∗ *delta* + *7* ∗ *QC k* ≤ *dist* (*q k um*) (*q k x*)
using *x* **by** *simp*
also have ... ≤ *dist* (*q k um*) (*q k v*) + *dist* (*q k v*) (*q k x*)
by (*intro mono-intros*)
also have ... ≤ (*9* ∗ *delta* + *4* ∗ *QC k*) + *dist* (*q k v*) (*q k x*)
using *w*(*3*)[*of v*] ‹*v* ∈ {*um*..*w*}› **by** *auto*
finally have *L* − *13* ∗ *delta* + *3* ∗ *QC k* ≤ *dist* (*q k v*) (*q k x*)
by *simp*
also have ... ≤ *3* ∗ *QC k* + *max* (*5* ∗ *deltaG*(*TYPE*(*'a*))) ((*4* ∗ *exp*(*1*/*2* ∗ *ln 2*)) ∗ *lambda* ∗ (*x* − *v*) ∗ *exp*(−(*dm* ∗ *2*^*k* − *C*/*2* − *QC k*) ∗ *ln 2* / (*5* ∗ *delta*)))
proof (*cases k* = *0*)

We use different statements for the projection in the case $k = 0$ (projection on a geodesic) and $k > 0$ (projection on a quasi-convex set) as the bounds are better in the first case, which is the most important one for the final value of the constant.

case *True*
have *dist* (*q k v*) (*q k x*) ≤ *max* (*5* ∗ *deltaG*(*TYPE*(*'a*))) ((*4* ∗ *exp*(*1*/*2* ∗ *ln 2*)) ∗ *lambda* ∗ (*x* − *v*) ∗ *exp*(−(*dm* ∗ *2*^*k* − *C*/*2*) ∗ *ln 2* / (*5* ∗ *delta*)))
proof (*rule geodesic-projection-exp-contracting*[**where** *?G* = *V k* **and** *?f* = *f*])
show *geodesic-segment* (*V k*) **unfolding** *True V-def* **using** *geodesic-segmentI*[*OF H*] **by** *auto*
show *v* ≤ *x* **using** ‹*v* ∈ {*um..w*}› ‹*w* ∈ {*um..x*}› **by** *auto*
show *q k v* ∈ *proj-set* (*f v*) (*V k*)
unfolding *q-def V-def* **apply** (*rule proj-set-thickening*)
using *aux p*[*of v*] *x*(*3*)[*of v*] ‹*v* ∈ {*um..w*}› ‹*w* ∈ {*um..x*}› **by** (*auto simp add*: *metric-space-class.dist-commute*)
show *q k x* ∈ *proj-set* (*f x*) (*V k*)
unfolding *q-def V-def* **apply** (*rule proj-set-thickening*)
using *aux p*[*of x*] *x*(*3*)[*of x*] ‹*w* ∈ {*um..x*}› **by** (*auto simp add*: *metric-space-class.dist-commute*)
show *15*/*2* ∗ *delta* + *C*/*2* ≤ *dm* ∗ *2*^*k*
apply (*rule order-trans*[*of - dm*])
using *I* ‹*delta* > *0*› ‹*C* ≥ *0*› *Laux* **unfolding** *QC-def* **by** *auto*
show *deltaG TYPE*(*'a*) < *delta* **by** *fact*
show ⋀*t*. *t* ∈ {*v..x*} ⟹ *dm* ∗ *2* ^ *k* ≤ *infdist* (*f t*) (*V k*)
using *aux4* **by** *auto*
show *0* ≤ *C 0* ≤ *lambda* **using** ‹*C* ≥ *0*› ‹*lambda* ≥ *1*› **by** *auto*
show *dist* (*f x1*) (*f x2*) ≤ *lambda* ∗ *dist x1 x2* + *C* **if** *x1* ∈ {*v..x*} *x2* ∈ {*v..x*} **for** *x1 x2*
using *quasi-isometry-onD*(*1*)[*OF assms*(*2*)] *that* ‹*v* ∈ {*um..w*}› ‹*w* ∈ {*um..x*}› ‹*x* ∈ {*um..ym*}› ‹*ym* ∈ {*um..z*}› ‹*um* ∈ {*a..z*}› ‹*z* ∈ {*a..b*}› **by** *auto*
qed
then show *?thesis* **unfolding** *QC-def True* **by** *auto*
next
case *False*
have *dist* (*q k v*) (*q k x*) ≤ *2* ∗ *QC k* + *8* ∗ *delta* + *max* (*5* ∗ *deltaG*(*TYPE*(*'a*))) ((*4* ∗ *exp*(*1*/*2* ∗ *ln 2*)) ∗ *lambda* ∗ (*x* − *v*) ∗ *exp*(−(*dm* ∗ *2*^*k* − *QC k* −*C*/*2*) ∗ *ln 2* / (*5* ∗ *delta*)))
proof (*rule quasiconvex-projection-exp-contracting*[**where** *?G* = *V k* **and** *?f* = *f*])
show *quasiconvex* (*QC k*) (*V k*) **by** *fact*
show *v* ≤ *x* **using** ‹*v* ∈ {*um..w*}› ‹*w* ∈ {*um..x*}› **by** *auto*
show *q k v* ∈ *proj-set* (*f v*) (*V k*)
unfolding *q-def V-def* **apply** (*rule proj-set-thickening*)
using *aux p*[*of v*] *x*(*3*)[*of v*] ‹*v* ∈ {*um..w*}› ‹*w* ∈ {*um..x*}› **by** (*auto simp add*: *metric-space-class.dist-commute*)
show *q k x* ∈ *proj-set* (*f x*) (*V k*)
unfolding *q-def V-def* **apply** (*rule proj-set-thickening*)
using *aux p*[*of x*] *x*(*3*)[*of x*] ‹*w* ∈ {*um..x*}› **by** (*auto simp add*:

metric-space-class.dist-commute)
show *15/2 ∗ delta + QC k + C/2 ≤ dm ∗ 2^k*
apply (*rule order-trans*[*of - dm*])
using *I* ‹*delta > 0*› ‹*C ≥ 0*› *Laux* **unfolding** *QC-def* **by** *auto*
show *deltaG TYPE('a) < delta* **by** *fact*
show ⋀*t. t ∈ {v..x} ⟹ dm ∗ 2 ^ k ≤ infdist (f t) (V k)*
using *aux4* **by** *auto*
show *0 ≤ C 0 ≤ lambda* **using** ‹*C ≥ 0*› ‹*lambda ≥ 1*› **by** *auto*
show *dist (f x1) (f x2) ≤ lambda ∗ dist x1 x2 + C* **if** *x1 ∈ {v..x} x2 ∈ {v..x}* **for** *x1 x2*
using *quasi-isometry-onD(1)[OF assms(2)] that* ‹*v ∈ {um..w}*› ‹*w ∈ {um..x}*› ‹*x ∈ {um..ym}*› ‹*ym ∈ {um..z}*› ‹*um ∈ {a..z}*› ‹*z ∈ {a..b}*› **by** *auto*
qed
then show *?thesis* **unfolding** *QC-def* **using** *False* **by** (*auto simp add*: *algebra-simps*)
qed
finally have *L − 13 ∗ delta ≤ max (5 ∗ deltaG(TYPE('a))) ((4 ∗ exp(1/2 ∗ ln 2)) ∗ lambda ∗ (x − v) ∗ exp(−(dm ∗ 2^k − C/2 − QC k) ∗ ln 2 / (5 ∗ delta)))*
by *auto*
then have *L − 13 ∗ delta ≤ (4 ∗ exp(1/2 ∗ ln 2)) ∗ lambda ∗ (x − v) ∗ exp(−(dm ∗ 2^k − C/2 − QC k) ∗ ln 2 / (5 ∗ delta))*
using ‹*delta > deltaG(TYPE('a))*› *Laux* **by** *auto*

We separate the exponential gain coming from the contraction into two parts, one to be spent to improve the constant, and one for the inductive argument.

also have ... ≤ *(4 ∗ exp(1/2 ∗ ln 2)) ∗ lambda ∗ (x − v) ∗ exp(−((1−alpha) ∗ D + alpha ∗ 2^k ∗ dm) ∗ ln 2 / (5 ∗ delta))*
apply (*intro mono-intros*) **using** *aux3* ‹*delta > 0*› ‹*lambda ≥ 1*› ‹*v ∈ {um..w}*› ‹*w ∈ {um..x}*› **by** *auto*
also have ... = *(4 ∗ exp(1/2 ∗ ln 2)) ∗ lambda ∗ (x − v) ∗ (exp(−(1−alpha) ∗ D ∗ ln 2 / (5 ∗ delta)) ∗ exp(−alpha ∗ 2^k ∗ dm ∗ ln 2 / (5 ∗ delta)))*
unfolding *mult-exp-exp* **by** (*auto simp add*: *algebra-simps divide-simps*)
finally have *A*: *L − 13 ∗ delta ≤ (4 ∗ exp(1/2 ∗ ln 2)) ∗ lambda ∗ exp(−(1−alpha) ∗ D ∗ ln 2 / (5 ∗ delta)) ∗ ((x − v) ∗ exp(−alpha ∗ 2^k ∗ dm ∗ ln 2 / (5 ∗ delta)))*
by (*simp add*: *algebra-simps*)

This is the end of the second substep.

Use the second substep to show that $x - v$ is bounded below, and therefore that $closestM - x$ (the endpoints of the new geodesic we want to consider in the inductive argument) are quantitatively closer than $uM - um$, which means that we will be able to use the inductive assumption over this new geodesic.

also have ... ≤ *(4 ∗ exp(1/2 ∗ ln 2)) ∗ lambda ∗ exp 0 ∗ ((x − v) ∗ exp 0)*

apply (*intro mono-intros*) **using** ‹*delta* > *0*› ‹*lambda* ≥ *1*› ‹*v* ∈ {*um*..*w*}› ‹*w* ∈ {*um*..*x*}› *alphaaux* ‹*D* > *0*› ‹*C* ≥ *0*› *I*
by (*auto simp add*: *divide-simps mult-nonpos-nonneg*)
also have ... = (*4* ∗ *exp*(*1*/*2* ∗ *ln 2*)) ∗ *lambda* ∗ (*x*−*v*)
by *simp*
also have ... ≤ *20* ∗ *lambda* ∗ (*x* − *v*)
apply (*intro mono-intros*, *approximation 10*)
using ‹*delta* > *0*› ‹*lambda* ≥ *1*› ‹*v* ∈ {*um*..*w*}› ‹*w* ∈ {*um*..*x*}› **by** *auto*
finally have *x* − *v* ≥ (*1*/*4*) ∗ *delta* / *lambda*
using ‹*lambda* ≥ *1*› *L-def* ‹*delta* > *0*› **by** (*simp add*: *divide-simps algebra-simps*)
then have *closestM* − *x* + (*1*/*4*) ∗ *delta* / *lambda* ≤ *closestM* − *v*
by *simp*
also have ... ≤ *uM* − *um*
using ‹*closestM* ∈ {*yM*..*uM*}› ‹*v* ∈ {*um*..*w*}› **by** *auto*
also have ... ≤ *Suc n* ∗ (*1*/*4*) ∗ *delta* / *lambda* **by** *fact*
finally have *closestM* − *x* ≤ *n* ∗ (*1*/*4*) ∗ *delta* / *lambda*
unfolding *Suc-eq-plus1* **by** (*auto simp add*: *algebra-simps add-divide-distrib*)

Conclusion of the proof: combine the lower bound of the second substep with the upper bound of the first substep to get a definite gain when one goes from the old geodesic to the new one. Then, apply the inductive assumption to the new one to conclude the desired inequality for the old one.

have *L* + *4* ∗ *delta* = ((*L* + *4* ∗ *delta*)/(*L* − *13* ∗ *delta*)) ∗ (*L* − *13* ∗ *delta*)
using *Laux* ‹*delta* > *0*› **by** (*simp add*: *algebra-simps divide-simps*)
also have ... ≤ ((*L* + *4* ∗ *delta*)/(*L* − *13* ∗ *delta*)) ∗ ((*4* ∗ *exp*(*1*/*2* ∗ *ln 2*)) ∗ *lambda* ∗ *exp* (− (*1* − *alpha*) ∗ *D* ∗ *ln 2* / (*5* ∗ *delta*)) ∗ ((*x* − *v*) ∗ *exp* (− *alpha* ∗ *2* ^ *k* ∗ *dm* ∗ *ln 2* / (*5* ∗ *delta*))))
apply (*rule mult-left-mono*) **using** *A Laux* ‹*delta* > *0*› **by** (*auto simp add*: *divide-simps*)
also have ... ≤ ((*L* + *4* ∗ *delta*)/(*L* − *13* ∗ *delta*)) ∗ ((*4* ∗ *exp*(*1*/*2* ∗ *ln 2*)) ∗ *lambda* ∗ *exp* (− (*1* − *alpha*) ∗ *D* ∗ *ln 2* / (*5* ∗ *delta*)) ∗ ((*exp*(−*K* ∗ (*closestM* − *x*)) − *exp*(−*K* ∗ (*uM* − *um*)))/*K*))
apply (*intro mono-intros B*) **using** *Laux* ‹*delta* > *0*› ‹*lambda* ≥ *1*› **by** (*auto simp add*: *divide-simps*)
finally have *C*: *L* + *4* ∗ *delta* ≤ *Kmult* ∗ (*exp*(−*K* ∗ (*closestM* − *x*)) − *exp*(−*K* ∗ (*uM* − *um*)))
unfolding *Kmult-def* **by** *auto*

have *Gromov-product-at* (*f z*) (*f um*) (*f uM*) ≤ *Gromov-product-at* (*f z*) (*f x*) (*f closestM*) + (*L* + *4* ∗ *delta*)
apply (*rule Rec*) **using** ‹*closestM* ∈ {*yM*..*uM*}› ‹*x* ∈ {*um*..*ym*}› ‹*ym* ∈ {*um*..*z*}› **by** *auto*
also have ... ≤ (*lambda*^*2* ∗ (*D* + *3*/*2* ∗ *L* + *delta* + *11*/*2* ∗ *C*) − *2* ∗ *delta* + *Kmult* ∗ (*1* − *exp*(− *K* ∗ (*closestM* − *x*)))) + (*Kmult* ∗ (*exp*(−*K* ∗ (*closestM* − *x*)) − *exp*(−*K* ∗ (*uM*−*um*))))
apply (*intro mono-intros C Suc.IH*)

using ‹$x \in \{um..ym\}$› ‹$ym \in \{um..z\}$› ‹$um \in \{a..z\}$› ‹$closestM \in \{yM..uM\}$› ‹$yM \in \{z..uM\}$› ‹$uM \in \{z..b\}$› ‹$closestM - x \leq n * (1/4) * delta / lambda$› **by** *auto*
also have ... = $(lambda\hat{}2 * (D + 3/2 * L + delta + 11/2 * C) - 2 * delta + Kmult * (1 - exp(- K * (uM - um))))$
unfolding *K-def* **by** (*simp add*: *algebra-simps*)
finally show *?thesis* **by** *auto*

End of the first subcase, when there is a good point v between um and w.

next
case *False*

Second subcase: between um and w, all points are far away from V_k. We will show that this implies that w is admissible for the step $k+1$.

have $\exists w{\in}\{um..ym\}$. $(\forall v{\in}\{um..w\}$. $(2 \hat{} (Suc\ k + 1) - 1) * dm \leq dist\ (f\ v)\ (p\ v)) \wedge L - 4 * delta + 7 * QC\ (Suc\ k) \leq dist\ (q\ (Suc\ k)\ um)\ (q\ (Suc\ k)\ w)$
proof (*rule bexI*[*of - w*], *auto*)
show $um \leq w$ $w \leq ym$ **using** ‹$w \in \{um..x\}$› ‹$x \in \{um..ym\}$› **by** *auto*
show $(4 * 2 \hat{} k - 1) * dm \leq dist\ (f\ x)\ (p\ x)$ **if** $um \leq x$ $x \leq w$ **for** x
using *False* ‹$dm \geq 0$› *that* **by** *force*

have $dist\ (q\ k\ um)\ (q\ (k{+}1)\ um) = 2\hat{}k * dm$
unfolding *q-def* **apply** (*subst geodesic-segment-param*(*7*)[**where** *?y* = *f um*])
using *x*(*3*)[*of um*] ‹$x \in \{um..ym\}$› *aux* **by** (*auto simp add*: *metric-space-class.dist-commute*, *simp add*: *algebra-simps*)
have $dist\ (q\ k\ w)\ (q\ (k{+}1)\ w) = 2\hat{}k * dm$
unfolding *q-def* **apply** (*subst geodesic-segment-param*(*7*)[**where** *?y* = *f w*])
using *x*(*3*)[*of w*] ‹$w \in \{um..x\}$› ‹$x \in \{um..ym\}$› *aux* **by** (*auto simp add*: *metric-space-class.dist-commute*, *simp add*: *algebra-simps*)
have *i*: $q\ k\ um \in proj\text{-}set\ (q\ (k{+}1)\ um)\ (V\ k)$
unfolding *q-def V-def* **apply** (*rule proj-set-thickening'*[*of - f um*])
using *p x*(*3*)[*of um*] ‹$x \in \{um..ym\}$› *aux* **by** (*auto simp add*: *algebra-simps metric-space-class.dist-commute*)
have *j*: $q\ k\ w \in proj\text{-}set\ (q\ (k{+}1)\ w)\ (V\ k)$
unfolding *q-def V-def* **apply** (*rule proj-set-thickening'*[*of - f w*])
using *p x*(*3*)[*of w*] ‹$x \in \{um..ym\}$› ‹$w \in \{um..x\}$› *aux* **by** (*auto simp add*: *algebra-simps metric-space-class.dist-commute*)

have $5 * delta + 2 * QC\ k \leq dist\ (q\ k\ um)\ (q\ k\ w)$ **using** *w*(*2*) **by** *simp*
also have ... $\leq max\ (5 * deltaG(TYPE('a)) + 2 * QC\ k)$
$(dist\ (q\ (k + 1)\ um)\ (q\ (k + 1)\ w) - dist\ (q\ k\ um)\ (q\ (k + 1)\ um) - dist\ (q\ k\ w)\ (q\ (k + 1)\ w) + 10 * deltaG(TYPE('a)) + 4 * QC\ k)$
by (*rule proj-along-quasiconvex-contraction*[*OF* ‹*quasiconvex* (*QC k*) (*V k*)› *i j*])

finally have $5 * delta + 2 * QC\ k \leq dist\ (q\ (k + 1)\ um)\ (q\ (k + 1)\ w) - dist\ (q\ k\ um)\ (q\ (k + 1)\ um) - dist\ (q\ k\ w)\ (q\ (k + 1)\ w) + 10 * deltaG(TYPE('a)) + 4 * QC\ k$
using ‹$deltaG(TYPE('a)) < delta$› **by** *auto*
then have $0 \leq dist\ (q\ (k + 1)\ um)\ (q\ (k + 1)\ w) + 5 * delta + 2 * QC\ k - dist\ (q\ k\ um)\ (q\ (k + 1)\ um) - dist\ (q\ k\ w)\ (q\ (k + 1)\ w)$
using ‹$deltaG(TYPE('a)) < delta$› **by** *auto*
also have ... $= dist\ (q\ (k + 1)\ um)\ (q\ (k + 1)\ w) + 5 * delta + 2 * QC\ k - 2\hat{}(k+1) * dm$
by (*simp only*: ‹$dist\ (q\ k\ w)\ (q\ (k+1)\ w) = 2\hat{}k * dm$› ‹$dist\ (q\ k\ um)\ (q\ (k+1)\ um) = 2\hat{}k * dm$›, *auto*)
finally have $*$: $2\hat{}(k+1) * dm - 5 * delta - 2 * QC\ k \leq dist\ (q\ (k+1)\ um)\ (q\ (k+1)\ w)$
using ‹$deltaG(TYPE('a)) < delta$› **by** *auto*
have $L - 4 * delta + 7 * QC\ (k+1) \leq 2 * dm - 5 * delta - 2 * QC\ k$
unfolding *QC-def L-def* **using** ‹$delta > 0$› *Laux I* ‹$C \geq 0$› **by** *auto*
also have ... $\leq 2\hat{}(k+1) * dm - 5 * delta - 2 * QC\ k$
using *aux* **by** (*auto simp add*: *algebra-simps*)
finally show $L - 4 * delta + 7 * QC\ (Suc\ k) \leq dist\ (q\ (Suc\ k)\ um)\ (q\ (Suc\ k)\ w)$
using $*$ **by** *auto*
qed
then show *?thesis*
by *simp*
qed
qed
qed

This is the end of the main induction over k. To conclude, choose k large enough so that the second alternative in this induction is impossible. It follows that the first alternative holds, i.e., the desired inequality is true.

have $dm > 0$ **using** *I* ‹$delta > 0$› ‹$C \geq 0$› *Laux* **by** *auto*
have $\exists k.\ 2\hat{}k > dist\ (f\ um)\ (p\ um)/dm + 1$
by (*simp add*: *real-arch-pow*)
then obtain k **where** $2\hat{}k > dist\ (f\ um)\ (p\ um)/dm + 1$
by *blast*
then have $dist\ (f\ um)\ (p\ um) < (2\hat{}k - 1) * dm$
using ‹$dm > 0$› **by** (*auto simp add*: *divide-simps algebra-simps*)
also have ... $\leq (2\hat{}(Suc\ k) - 1) * dm$
by (*intro mono-intros*, *auto*)
finally have $\neg((2\ \hat{}\ (k + 1) - 1) * dm \leq dist\ (f\ um)\ (p\ um))$
by *simp*
then show $Gromov\text{-}product\text{-}at\ (f\ z)\ (f\ um)\ (f\ uM) \leq lambda^2 * (D + 3/2 * L + delta + 11/2 * C) - 2 * delta + Kmult * (1 - exp\ (- K * (uM - um)))$
using *Ind-k*[*of k*] **by** *auto*

end of the case where $D + 4 * C \leq dm$ and $dM \leq dm$.

next

case *3*

This is the exact copy of the previous case, except that the roles of the points before and after z are exchanged. In a perfect world, one would use a lemma subsuming both cases, but in practice copy-paste seems to work better here as there are two many details to be changed regarding the direction of inequalities.

then have I: $D + 4 * C \leq dM\ dm \leq dM$ **by** *auto*
define V **where** $V = (\lambda k::nat.\ (\bigcup g\in H.\ cball\ g\ ((2\hat{}k - 1) * dM)))$
define QC **where** $QC = (\lambda k::nat.\ if\ k = 0\ then\ 0\ else\ 8 * delta)$
have $QC\ k \geq 0$ **for** k **unfolding** *QC-def* **using** ‹$delta > 0$› **by** *auto*
have Q: *quasiconvex* $(0 + 8 * deltaG(TYPE('a)))$ $(V\ k)$ **for** k
unfolding *V-def* **apply** (*rule quasiconvex-thickening*) **using** *geodesic-segmentI*[*OF H*]
by (*auto simp add*: *quasiconvex-of-geodesic*)
have *quasiconvex* $(QC\ k)$ $(V\ k)$ **for** k
apply (*cases* $k = 0$)
apply (*simp add*: *V-def QC-def quasiconvex-of-geodesic geodesic-segmentI*[*OF H*])
apply (*rule quasiconvex-mono*[*OF - Q*[*of k*]]) **using** ‹$deltaG(TYPE('a)) < delta$› *QC-def* **by** *auto*
define q::$nat \Rightarrow real \Rightarrow 'a$ **where** $q = (\lambda k\ x.\ geodesic\text{-}segment\text{-}param\ \{p\ x{-}{-}f\ x\}\ (p\ x)\ ((2\hat{}k - 1) * dM))$

have *Ind-k*: $(Gromov\text{-}product\text{-}at\ (f\ z)\ (f\ um)\ (f\ uM) \leq lambda\hat{}2 * (D + 3/2 * L + delta + 11/2 * C) - 2 * delta + Kmult * (1 - exp(- K * (uM - um))))$
$\vee (\exists x \in \{yM..uM\}.\ (\forall y \in \{x..uM\}.\ dist\ (f\ y)\ (p\ y) \geq (2\hat{}(k+1)-1) * dM) \wedge dist\ (q\ k\ uM)\ (q\ k\ x) \geq L - 4 * delta + 7 * QC\ k)$ **for** k
proof (*induction k*)
case *0*
have $*$: $\exists x\in \{yM..uM\}.\ (\forall y \in \{x..uM\}.\ dist\ (f\ y)\ (p\ y) \geq (2\hat{}(0+1)-1) * dM) \wedge dist\ (q\ 0\ uM)\ (q\ 0\ x) \geq L - 4 * delta + 7 * QC\ 0$
proof (*rule bexI*[*of - yM*], *auto simp add*: *V-def q-def QC-def*)
show $yM \leq uM$ **using** ‹$yM \in \{z..uM\}$› **by** *auto*
show $L -4 * delta \leq dist\ (p\ uM)\ (p\ yM)$
using $yM(2)$ **apply** *auto* **using** *metric-space-class.zero-le-dist*[*of pi-z p uM*] **by** *linarith*
show $\bigwedge y.\ y \leq uM \Longrightarrow yM \leq y \Longrightarrow dM \leq dist\ (f\ y)\ (p\ y)$
using *dM-def closestM proj-setD*(*2*)[*OF p*] **by** *auto*
qed
then show *?case*
by *blast*
next
case *Suck*: (*Suc k*)
show *?case*
proof (*cases* $Gromov\text{-}product\text{-}at\ (f\ z)\ (f\ um)\ (f\ uM) \leq lambda^2 * (D + 3/2 * L + delta + 11/2 * C) - 2 * delta + Kmult * (1 - exp\ (- K * (uM - um))))$)

case *True*
then show *?thesis* **by** *simp*
next
case *False*
then obtain x **where** x: $x \in \{yM..uM\}$ $dist\ (q\ k\ uM)\ (q\ k\ x) \geq L - 4 * delta + 7 * QC\ k$
$\bigwedge w.\ w \in \{x..uM\} \Longrightarrow dist\ (f\ w)\ (p\ w) \geq (2\hat{\ }(k+1)-1) * dM$
using *Suck.IH* **by** *auto*
have *aux*: $(2\ \hat{}\ k - 1) * dM \leq (2*2\hat{\ }k-1) * dM$ $0 \leq 2 * 2\ \hat{}\ k - (1::real)$ $dM \leq dM * 2\ \hat{}\ k$
apply (*auto simp add*: *algebra-simps*)
apply (*metis power.simps*(*2*) *two-realpow-ge-one*)
using ‹$0 \leq dM$› *less-eq-real-def* **by** *fastforce*
have $L + C = (L/D) * (D + (D/L) * C)$
using ‹$L > 0$› ‹$D > 0$› **by** (*simp add*: *algebra-simps divide-simps*)
also have ... $\leq (L/D) * (D + 4 * C)$
apply (*intro mono-intros*)
using ‹$L > 0$› ‹$D > 0$› ‹$C \geq 0$› ‹$D \leq 4 * L$› **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... $\leq (L/D) * dM$
apply (*intro mono-intros*) **using** *I* ‹$L > 0$› ‹$D > 0$› **by** *auto*
finally have $L + C \leq (L/D) * dM$
by *simp*
moreover have $2 * delta \leq (2 * delta)/D * dM$
using *I* ‹$C \geq 0$› ‹$delta > 0$› ‹$D > 0$› **by** (*auto simp add*: *algebra-simps divide-simps*)
ultimately have *aux2*: $L + C + 2 * delta \leq ((L + 2 * delta)/D) * dM$
by (*auto simp add*: *algebra-simps divide-simps*)
have *aux3*: $(1-alpha) * D + alpha * 2\hat{\ }k * dM \leq dM * 2\hat{\ }k - C/2 - QC\ k$
proof (*cases* $k = 0$)
case *True*
show *?thesis*
using *I* ‹$C \geq 0$› **unfolding** *True QC-def alpha-def* **by** *auto*
next
case *False*
have $C/2 + QC\ k + (1-alpha) * D \leq 2 * (1-alpha) * dM$
using *I* ‹$C \geq 0$› **unfolding** *QC-def alpha-def* **using** *False Laux* **by** *auto*
also have ... $\leq 2\hat{\ }k * (1-alpha) * dM$
apply (*intro mono-intros*) **using** *False alphaaux I* ‹$D > 0$› ‹$C \geq 0$› **by** *auto*
finally show *?thesis*
by (*simp add*: *algebra-simps*)
qed

have $\exists w \in \{x..uM\}.\ (dist\ (q\ k\ uM)\ (q\ k\ w) \in \{(9 * delta + 4 * QC\ k) - 4 * delta - 2 * QC\ k\ ..\ 9 * delta + 4 * QC\ k\})$

$\wedge$ ($\forall$ *v* $\in$ {*w*..*uM*}. *dist* (*q k uM*) (*q k v*) $\leq$ *9* $*$ *delta* + *4* $*$ *QC k*)
proof (*rule quasi-convex-projection-small-gaps'*[**where** *?f* = *f* **and** *?G* = *V k*])
show *continuous-on* {*x*..*uM*} *f*
apply (*rule continuous-on-subset*[*OF* ‹*continuous-on* {*a*..*b*} *f*›])
using ‹*uM* $\in$ {*z*..*b*}› ‹*z* $\in$ {*a*..*b*}› ‹*yM* $\in$ {*z*..*uM*}› ‹*x* $\in$ {*yM*..*uM*}› **by** *auto*
show *x* $\leq$ *uM* **using** ‹*x* $\in$ {*yM*..*uM*}› **by** *auto*
show *quasiconvex* (*QC k*) (*V k*) **by** *fact*
show *deltaG TYPE*(*'a*) < *delta* **by** *fact*
show *9* $*$ *delta* + *4* $*$ *QC k* $\in$ {*4* $*$ *delta* + *2* $*$ *QC k*..*dist* (*q k x*) (*q k uM*)}
using *x*(*2*) ‹*delta* > *0*› ‹*QC k* $\geq$ *0*› *Laux* **by** (*auto simp add*: *metric-space-class.dist-commute*)
show *q k w* $\in$ *proj-set* (*f w*) (*V k*) **if** *w* $\in$ {*x*..*uM*} **for** *w*
unfolding *V-def q-def* **apply** (*rule proj-set-thickening*)
using *aux p x*(*3*)[*OF that*] **by** (*auto simp add*: *metric-space-class.dist-commute*)
qed
then obtain *w* **where** *w*: *w* $\in$ {*x*..*uM*}
dist (*q k uM*) (*q k w*) $\in$ {(*9* $*$ *delta* + *4* $*$ *QC k*) − *4* $*$ *delta* − *2* $*$ *QC k* .. *9* $*$ *delta* + *4* $*$ *QC k*}
$\bigwedge$*v*. *v* $\in$ {*w*..*uM*} $\Longrightarrow$ *dist* (*q k uM*) (*q k v*) $\leq$ *9* $*$ *delta* + *4* $*$ *QC k*
by *auto*
show *?thesis*
proof (*cases* $\exists$ *v* $\in$ {*w*..*uM*}. *dist* (*f v*) (*p v*) $\leq$ (*2*^(*k*+*2*)−*1*) $*$ *dM*)
case *True*
then obtain *v* **where** *v*: *v* $\in$ {*w*..*uM*} *dist* (*f v*) (*p v*) $\leq$ (*2*^(*k*+*2*)−*1*) $*$ *dM*
by *auto*
have *aux4*: *dM* $*$ *2* ^ *k* $\leq$ *infdist* (*f r*) (*V k*) **if** *r* $\in$ {*x*..*v*} **for** *r*
proof −
have $*$: *q k r* $\in$ *proj-set* (*f r*) (*V k*)
unfolding *q-def V-def* **apply** (*rule proj-set-thickening*)
using *aux p*[*of r*] *x*(*3*)[*of r*] *that* ‹*v* $\in$ {*w*..*uM*}› ‹*w* $\in$ {*x*..*uM*}› **by** (*auto simp add*: *metric-space-class.dist-commute*)
have *infdist* (*f r*) (*V k*) = *dist* (*geodesic-segment-param* {*p r*−−*f r*} (*p r*) (*dist* (*p r*) (*f r*))) (*geodesic-segment-param* {*p r*−−*f r*} (*p r*) ((*2* ^ *k* − *1*) $*$ *dM*))
using *proj-setD*(*2*)[*OF* $*$] **unfolding** *q-def* **by** *auto*
also have ... = *abs*(*dist* (*p r*) (*f r*) − (*2* ^ *k* − *1*) $*$ *dM*)
apply (*rule geodesic-segment-param*(*7*)[**where** *?y* = *f r*])
using *x*(*3*)[*of r*] ‹*r* $\in$ {*x*..*v*}› ‹*v* $\in$ {*w*..*uM*}› ‹*w* $\in$ {*x*..*uM*}› *aux* **by** (*auto simp add*: *metric-space-class.dist-commute*)
also have ... = *dist* (*f r*) (*p r*) − (*2* ^ *k* − *1*) $*$ *dM*
using *x*(*3*)[*of r*] ‹*r* $\in$ {*x*..*v*}› ‹*v* $\in$ {*w*..*uM*}› ‹*w* $\in$ {*x*..*uM*}› *aux* **by** (*auto simp add*: *metric-space-class.dist-commute*)
finally have *dist* (*f r*) (*p r*) = *infdist* (*f r*) (*V k*) + (*2* ^ *k* − *1*) $*$ *dM* **by** *simp*

moreover have $(2\widehat{\ }(k+1) - 1) * dM \leq dist\ (f\ r)\ (p\ r)$
apply $(rule\ x(3))$ **using** ‹$r \in \{x..v\}$› ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› **by** *auto*
ultimately have $(2\widehat{\ }(k+1) - 1) * dM \leq infdist\ (f\ r)\ (V\ k) + (2\ \widehat{\ }\ k - 1) * dM$
by *simp*
then show *?thesis* **by** (*auto simp add*: *algebra-simps*)
qed

have $infdist\ (f\ v)\ H \leq (2\widehat{\ }(k+2)-1) * dM$
using v *proj-setD*$(2)[OF\ p[of\ v]]$ **by** *auto*
have $dist\ closestm\ v \leq lambda * (infdist\ (f\ closestm)\ H + (L + C + 2 * delta) + infdist\ (f\ v)\ H)$
apply (*rule D*)
using ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› ‹$x \in \{yM..uM\}$› ‹$yM \in \{z..uM\}$› ‹$uM \in \{z..b\}$› ‹$z \in \{a..b\}$› ‹$closestm \in \{um..ym\}$› ‹$ym \in \{um..z\}$› ‹$um \in \{a..z\}$› **by** *auto*
also have ... $\leq lambda * (dm + 1 * (L + C + 2 * delta) + (2\widehat{\ }(k+2)-1) * dM)$
apply (*intro mono-intros* ‹$infdist\ (f\ v)\ H \leq (2\widehat{\ }(k+2)-1) * dM$›)
using *dm-def* ‹$lambda \geq 1$› ‹$L > 0$› ‹$C \geq 0$› ‹$delta > 0$› **by** (*auto simp add*: *metric-space-class.dist-commute*)
also have ... $\leq lambda * (dM + 2\widehat{\ }k * (((L + 2 * delta)/D) * dM) + (2\widehat{\ }(k+2)-1) * dM)$
apply (*intro mono-intros*) **using** I ‹$lambda \geq 1$› ‹$C \geq 0$› ‹$delta > 0$› ‹$L > 0$› *aux2* **by** *auto*
also have ... $= lambda * 2\widehat{\ }k * (4 + (L + 2 * delta)/D) * dM$
by (*simp add*: *algebra-simps*)
finally have $*$: $dist\ closestm\ v\ /\ (lambda * (4 + (L + 2 * delta)/D)) \leq 2\widehat{\ }k * dM$
using ‹$lambda \geq 1$› ‹$L > 0$› ‹$D > 0$› ‹$delta > 0$› **by** (*simp add*: *divide-simps*, *simp add*: *algebra-simps*)

have $exp(-\ (alpha * (2\widehat{\ }k * dM) * ln\ 2\ /\ (5 * delta))) \leq exp(-(alpha * (dist\ closestm\ v\ /\ (lambda * (4 + (L + 2 * delta)/D))) * ln\ 2\ /\ (5 * delta)))$
apply (*intro mono-intros* $*$) **using** *alphaaux* ‹$delta > 0$› **by** *auto*
also have ... $= exp(-K * dist\ closestm\ v)$
unfolding *K-def* **by** (*simp add*: *divide-simps*)
also have ... $= exp(-K * (v - closestm))$
unfolding *dist-real-def* **using** ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› ‹$x \in \{yM..uM\}$› ‹$yM \in \{z..uM\}$› ‹$ym \in \{um..z\}$› ‹$closestm \in \{um..ym\}$› ‹$K > 0$› **by** *auto*
finally have $exp(-\ (alpha * (2\widehat{\ }k * dM) * ln\ 2\ /\ (5 * delta))) \leq exp(-K * (v - closestm))$
by *simp*
then have $K * (v - x) * exp(-\ (alpha * (2\widehat{\ }k * dM) * ln\ 2\ /\ (5 * delta))) \leq K * (v - x) * exp(-K * (v - closestm))$
apply (*rule mult-left-mono*)
using ‹$delta > 0$› ‹$lambda \geq 1$› ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› ‹K

> 0› **by** *auto*
also have ... $= ((1 + K * (v - x)) - 1) * exp(- K * (v - closestm))$
by (*auto simp add*: *algebra-simps*)
also have ... $\leq (exp\ (K * (v - x)) - 1) * exp(-K * (v - closestm))$
by (*intro mono-intros*, *auto*)
also have ... $= exp(-K * (x - closestm)) - exp(-K * (v - closestm))$
by (*simp add*: *algebra-simps mult-exp-exp*)
also have ... $\leq exp(-K * (x - closestm)) - exp(-K * (uM - um))$
using ‹$K > 0$› ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› ‹$x \in \{yM..uM\}$› ‹$yM \in \{z..uM\}$› ‹$ym \in \{um..z\}$› ‹$closestm \in \{um..ym\}$› **by** *auto*
finally have B: $(v - x) * exp(- alpha * 2\hat{}k * dM * ln\ 2\ /\ (5 * delta)) \leq$
$(exp(-K * (x - closestm)) - exp(-K * (uM - um)))/K$
using ‹$K > 0$› **by** (*auto simp add*: *divide-simps algebra-simps*)

The projections of $f(v)$ and $f(x)$ on the cylinder V_k are well separated, by construction. This implies that v and x themselves are well separated.

have $L - 4 * delta + 7 * QC\ k \leq dist\ (q\ k\ uM)\ (q\ k\ x)$
using x **by** *simp*
also have ... $\leq dist\ (q\ k\ uM)\ (q\ k\ v) + dist\ (q\ k\ v)\ (q\ k\ x)$
by (*intro mono-intros*)
also have ... $\leq (9 * delta + 4 * QC\ k) + dist\ (q\ k\ v)\ (q\ k\ x)$
using $w(3)[of\ v]$ ‹$v \in \{w..uM\}$› **by** *auto*
finally have $L - 13 * delta + 3 * QC\ k \leq dist\ (q\ k\ x)\ (q\ k\ v)$
by (*simp add*: *metric-space-class.dist-commute*)
also have ... $\leq 3 * QC\ k + max\ (5 * deltaG(TYPE('a)))\ ((4 * exp(1/2 * ln\ 2)) * lambda * (v - x) * exp(-(dM * 2\hat{}k - C/2 - QC\ k) * ln\ 2\ /\ (5 * delta)))$
proof (*cases* $k = 0$)
case *True*
have $dist\ (q\ k\ x)\ (q\ k\ v) \leq max\ (5 * deltaG(TYPE('a)))\ ((4 * exp(1/2 * ln\ 2)) * lambda * (v - x) * exp(-(dM * 2\hat{}k - C/2) * ln\ 2\ /\ (5 * delta)))$
proof (*rule geodesic-projection-exp-contracting*[**where** $?G = V\ k$ **and** $?f = f$])
show *geodesic-segment* $(V\ k)$ **unfolding** *V-def True* **using** *geodesic-segmentI*$[OF\ H]$ **by** *auto*
show $x \leq v$ **using** ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› **by** *auto*
show $q\ k\ v \in proj\text{-}set\ (f\ v)\ (V\ k)$
unfolding *q-def V-def* **apply** (*rule proj-set-thickening*)
using *aux* $p[of\ v]$ $x(3)[of\ v]$ ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› **by** (*auto simp add*: *metric-space-class.dist-commute*)
show $q\ k\ x \in proj\text{-}set\ (f\ x)\ (V\ k)$
unfolding *q-def V-def* **apply** (*rule proj-set-thickening*)
using *aux* $p[of\ x]$ $x(3)[of\ x]$ ‹$w \in \{x..uM\}$› **by** (*auto simp add*: *metric-space-class.dist-commute*)
show $15/2 * delta + C/2 \leq dM * 2\hat{}k$
using I ‹$delta > 0$› ‹$C \geq 0$› *Laux* **unfolding** *QC-def True* **by** *auto*
show $deltaG\ TYPE('a) < delta$ **by** *fact*

show $\bigwedge t.\ t \in \{x..v\} \Longrightarrow dM * 2\ \hat{}\ k \leq infdist\ (f\ t)\ (V\ k)$
using *aux4* **by** *auto*
show $0 \leq C\ 0 \leq lambda$ **using** ‹$C \geq 0$› ‹$lambda \geq 1$› **by** *auto*
show $dist\ (f\ x1)\ (f\ x2) \leq lambda * dist\ x1\ x2 + C$ **if** $x1 \in \{x..v\}$ $x2 \in \{x..v\}$ **for** $x1\ x2$
using *quasi-isometry-onD*(1)[*OF assms*(2)] *that* ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› ‹$x \in \{yM..uM\}$› ‹$yM \in \{z..uM\}$› ‹$uM \in \{z..b\}$› ‹$z \in \{a..b\}$› **by** *auto*
qed
then show *?thesis* **unfolding** *QC-def True* **by** *auto*
next
case *False*
have $dist\ (q\ k\ x)\ (q\ k\ v) \leq 2 * QC\ k + 8 * delta + max\ (5 * deltaG(TYPE('a)))\ ((4 * exp(1/2 * ln\ 2)) * lambda * (v - x) * exp(-(dM * 2\hat{}k - QC\ k - C/2) * ln\ 2\ /\ (5 * delta)))$
proof (*rule quasiconvex-projection-exp-contracting*[**where** $?G = V\ k$ **and** $?f = f$])
show *quasiconvex* $(QC\ k)\ (V\ k)$ **by** *fact*
show $x \leq v$ **using** ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› **by** *auto*
show $q\ k\ v \in proj\text{-}set\ (f\ v)\ (V\ k)$
unfolding *q-def V-def* **apply** (*rule proj-set-thickening*)
using *aux p*[*of v*] *x*(3)[*of v*] ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› **by** (*auto simp add*: *metric-space-class.dist-commute*)
show $q\ k\ x \in proj\text{-}set\ (f\ x)\ (V\ k)$
unfolding *q-def V-def* **apply** (*rule proj-set-thickening*)
using *aux p*[*of x*] *x*(3)[*of x*] ‹$w \in \{x..uM\}$› **by** (*auto simp add*: *metric-space-class.dist-commute*)
show $15/2 * delta + QC\ k + C/2 \leq dM * 2\hat{}k$
apply (*rule order-trans*[*of - dM*])
using *I* ‹$delta > 0$› ‹$C \geq 0$› *Laux* **unfolding** *QC-def* **by** *auto*
show $deltaG\ TYPE('a) < delta$ **by** *fact*
show $\bigwedge t.\ t \in \{x..v\} \Longrightarrow dM * 2\ \hat{}\ k \leq infdist\ (f\ t)\ (V\ k)$
using *aux4* **by** *auto*
show $0 \leq C\ 0 \leq lambda$ **using** ‹$C \geq 0$› ‹$lambda \geq 1$› **by** *auto*
show $dist\ (f\ x1)\ (f\ x2) \leq lambda * dist\ x1\ x2 + C$ **if** $x1 \in \{x..v\}$ $x2 \in \{x..v\}$ **for** $x1\ x2$
using *quasi-isometry-onD*(1)[*OF assms*(2)] *that* ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› ‹$x \in \{yM..uM\}$› ‹$yM \in \{z..uM\}$› ‹$uM \in \{z..b\}$› ‹$z \in \{a..b\}$› **by** *auto*
qed
then show *?thesis* **unfolding** *QC-def* **using** *False* **by** (*auto simp add*: *algebra-simps*)
qed
finally have $L - 13 * delta \leq max\ (5 * deltaG(TYPE('a)))\ ((4 * exp(1/2 * ln\ 2)) * lambda * (v - x) * exp(-(dM * 2\hat{}k - C/2 - QC\ k) * ln\ 2\ /\ (5 * delta)))$
by *auto*
then have $L - 13 * delta \leq (4 * exp(1/2 * ln\ 2)) * lambda * (v - x) * exp(-(dM * 2\hat{}k - C/2 - QC\ k) * ln\ 2\ /\ (5 * delta))$

using ‹$delta > deltaG(TYPE('a))$› *Laux* **by** *auto*
also have ... $\leq$ $(4 * exp(1/2 * ln\ 2)) * lambda * (v - x) * exp(-((1-alpha) * D + alpha * 2\hat{}k * dM) * ln\ 2 / (5 * delta))$
apply (*intro mono-intros*) **using** *aux3* ‹$delta > 0$› ‹$lambda \geq 1$› ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› **by** *auto*
also have ... $= (4 * exp(1/2 * ln\ 2)) * lambda * (v - x) * (exp(-(1-alpha) * D * ln\ 2 / (5 * delta)) * exp(-alpha * 2\hat{}k * dM * ln\ 2 / (5 * delta)))$
unfolding *mult-exp-exp* **by** (*auto simp add*: *algebra-simps divide-simps*)
finally have *A*: $L - 13 * delta \leq (4 * exp(1/2 * ln\ 2)) * lambda * exp(-(1-alpha) * D * ln\ 2 / (5 * delta)) * ((v - x) * exp(-alpha * 2\hat{}k * dM * ln\ 2 / (5 * delta)))$
by (*simp add*: *algebra-simps*)

also have ... $\leq (4 * exp(1/2 * ln\ 2)) * lambda * exp\ 0 * ((v - x) * exp\ 0)$
apply (*intro mono-intros*) **using** ‹$delta > 0$› ‹$lambda \geq 1$› ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› *alphaaux* ‹$D > 0$› ‹$C \geq 0$› *I*
by (*auto simp add*: *divide-simps mult-nonpos-nonneg*)
also have ... $= (4 * exp(1/2 * ln\ 2)) * lambda * (v - x)$
by *simp*
also have ... $\leq 20 * lambda * (v - x)$
apply (*intro mono-intros*, *approximation 10*)
using ‹$delta > 0$› ‹$lambda \geq 1$› ‹$v \in \{w..uM\}$› ‹$w \in \{x..uM\}$› **by** *auto*
finally have $v - x \geq (1/4) * delta / lambda$
using ‹$lambda \geq 1$› *L-def* ‹$delta > 0$› **by** (*simp add*: *divide-simps algebra-simps*)
then have $x - closestm + (1/4) * delta / lambda \leq v - closestm$
by *simp*
also have ... $\leq uM - um$
using ‹$closestm \in \{um..ym\}$› ‹$v \in \{w..uM\}$› **by** *auto*
also have ... $\leq Suc\ n * (1/4) * delta / lambda$ **by** *fact*
finally have $x - closestm \leq n * (1/4) * delta / lambda$
unfolding *Suc-eq-plus1* **by** (*auto simp add*: *algebra-simps add-divide-distrib*)

have $L + 4 * delta = ((L + 4 * delta)/(L - 13 * delta)) * (L - 13 * delta)$
using *Laux* ‹$delta > 0$› **by** (*simp add*: *algebra-simps divide-simps*)
also have ... $\leq ((L + 4 * delta)/(L - 13 * delta)) * ((4 * exp(1/2 * ln\ 2)) * lambda * exp\ (-\ (1 - alpha) * D * ln\ 2 / (5 * delta)) * ((v - x) * exp\ (-\ alpha * 2\ \hat{}\ k * dM * ln\ 2 / (5 * delta))))$
apply (*rule mult-left-mono*) **using** *A Laux* ‹$delta > 0$› **by** (*auto simp add*: *divide-simps*)
also have ... $\leq ((L + 4 * delta)/(L - 13 * delta)) * ((4 * exp(1/2 * ln\ 2)) * lambda * exp\ (-\ (1 - alpha) * D * ln\ 2 / (5 * delta)) * ((exp(-K * (x - closestm)) - exp(-K * (uM - um)))/K))$
apply (*intro mono-intros B*) **using** *Laux* ‹$delta > 0$› ‹$lambda \geq 1$› **by** (*auto simp add*: *divide-simps*)

finally have *C*: *L* + *4* ∗ *delta* ≤ *Kmult* ∗ (*exp*(−*K* ∗ (*x* − *closestm*)) − *exp*(−*K* ∗ (*uM* − *um*)))
unfolding *Kmult-def* **by** *argo*

have *Gromov-product-at* (*f z*) (*f um*) (*f uM*) ≤ *Gromov-product-at* (*f z*) (*f closestm*) (*f x*) + (*L* + *4* ∗ *delta*)
apply (*rule Rec*) **using** ‹*closestm* ∈ {*um..ym*}› ‹*x* ∈ {*yM..uM*}› ‹*yM* ∈ {*z..uM*}› **by** *auto*
also have ... ≤ (*lambda*^*2* ∗ (*D* + *3/2* ∗ *L* + *delta* + *11/2* ∗ *C*) − *2* ∗ *delta* + *Kmult* ∗ (*1* − *exp*(− *K* ∗ (*x* − *closestm*)))) + (*Kmult* ∗ (*exp*(−*K* ∗ (*x* − *closestm*)) − *exp*(−*K* ∗ (*uM*−*um*))))
apply (*intro mono-intros C Suc.IH*)
using ‹*x* ∈ {*yM..uM*}› ‹*yM* ∈ {*z..uM*}› ‹*um* ∈ {*a..z*}› ‹*closestm* ∈ {*um..ym*}› ‹*ym* ∈ {*um..z*}› ‹*uM* ∈ {*z..b*}› ‹*x* − *closestm* ≤ *n* ∗ (*1/4*) ∗ *delta* / *lambda*› **by** *auto*
also have ... = (*lambda*^*2* ∗ (*D* + *3/2* ∗ *L* + *delta* + *11/2* ∗ *C*) − *2* ∗ *delta* + *Kmult* ∗ (*1* − *exp*(− *K* ∗ (*uM* − *um*))))
unfolding *K-def* **by** (*simp add*: *algebra-simps*)
finally show *?thesis* **by** *auto*
next
case *False*
have ∃ *w*∈{*yM..uM*}. (∀ *r*∈{*w..uM*}. (*2* ^ (*Suc k* + *1*) − *1*) ∗ *dM* ≤ *dist* (*f r*) (*p r*)) ∧ *L* − *4* ∗ *delta* + *7* ∗ *QC* (*Suc k*) ≤ *dist* (*q* (*Suc k*) *uM*) (*q* (*Suc k*) *w*)
proof (*rule bexI*[*of* - *w*], *auto*)
show *w* ≤ *uM yM* ≤ *w* **using** ‹*w* ∈ {*x..uM*}› ‹*x* ∈ {*yM..uM*}› **by** *auto*
show (*4* ∗ *2* ^ *k* − *1*) ∗ *dM* ≤ *dist* (*f x*) (*p x*) **if** *x* ≤ *uM w* ≤ *x* **for** *x*
using *False* ‹*dM* ≥ *0*› *that* **by** *force*

have *dist* (*q k uM*) (*q* (*k*+*1*) *uM*) = *2*^*k* ∗ *dM*
unfolding *q-def* **apply** (*subst geodesic-segment-param*(*7*)[**where** *?y* = *f uM*])
using *x*(*3*)[*of uM*] ‹*x* ∈ {*yM..uM*}› *aux* **by** (*auto simp add*: *metric-space-class.dist-commute*, *simp add*: *algebra-simps*)
have *dist* (*q k w*) (*q* (*k*+*1*) *w*) = *2*^*k* ∗ *dM*
unfolding *q-def* **apply** (*subst geodesic-segment-param*(*7*)[**where** *?y* = *f w*])
using *x*(*3*)[*of w*] ‹*w* ∈ {*x..uM*}› ‹*x* ∈ {*yM..uM*}› *aux* **by** (*auto simp add*: *metric-space-class.dist-commute*, *simp add*: *algebra-simps*)
have *i*: *q k uM* ∈ *proj-set* (*q* (*k*+*1*) *uM*) (*V k*)
unfolding *q-def V-def* **apply** (*rule proj-set-thickening'*[*of* - *f uM*])
using *p x*(*3*)[*of uM*] ‹*x* ∈ {*yM..uM*}› *aux* **by** (*auto simp add*: *algebra-simps metric-space-class.dist-commute*)
have *j*: *q k w* ∈ *proj-set* (*q* (*k*+*1*) *w*) (*V k*)
unfolding *q-def V-def* **apply** (*rule proj-set-thickening'*[*of* - *f w*])
using *p x*(*3*)[*of w*] ‹*x* ∈ {*yM..uM*}› ‹*w* ∈ {*x..uM*}› *aux* **by** (*auto simp add*: *algebra-simps metric-space-class.dist-commute*)

have $5 * delta + 2 * QC\ k \leq dist\ (q\ k\ uM)\ (q\ k\ w)$ **using** $w(2)$ **by** *simp*
also have ... $\leq max\ (5 * deltaG(TYPE('a)) + 2 * QC\ k)$
$(dist\ (q\ (k + 1)\ uM)\ (q\ (k + 1)\ w) - dist\ (q\ k\ uM)\ (q\ (k + 1)\ uM) - dist\ (q\ k\ w)\ (q\ (k + 1)\ w) + 10 * deltaG(TYPE('a)) + 4 * QC\ k)$
by (*rule proj-along-quasiconvex-contraction*[*OF* ‹*quasiconvex* (*QC* *k*) (*V* *k*)› *i j*])
finally have $5 * delta + 2 * QC\ k \leq dist\ (q\ (k + 1)\ uM)\ (q\ (k + 1)\ w) - dist\ (q\ k\ uM)\ (q\ (k + 1)\ uM) - dist\ (q\ k\ w)\ (q\ (k + 1)\ w) + 10 * deltaG(TYPE('a)) + 4 * QC\ k$
using ‹$deltaG(TYPE('a)) < delta$› **by** *auto*
then have $0 \leq dist\ (q\ (k + 1)\ uM)\ (q\ (k + 1)\ w) + 5 * delta + 2 * QC\ k - dist\ (q\ k\ uM)\ (q\ (k + 1)\ uM) - dist\ (q\ k\ w)\ (q\ (k + 1)\ w)$
using ‹$deltaG(TYPE('a)) < delta$› **by** *auto*
also have ... $= dist\ (q\ (k + 1)\ uM)\ (q\ (k + 1)\ w) + 5 * delta + 2 * QC\ k - 2\hat{\ }(k+1) * dM$
by (*simp only*: ‹$dist\ (q\ k\ w)\ (q\ (k+1)\ w) = 2\hat{\ }k * dM$› ‹$dist\ (q\ k\ uM)\ (q\ (k+1)\ uM) = 2\hat{\ }k * dM$›, *auto*)
finally have $*$: $2\hat{\ }(k+1) * dM - 5 * delta - 2 * QC\ k \leq dist\ (q\ (k+1)\ uM)\ (q\ (k+1)\ w)$
using ‹$deltaG(TYPE('a)) < delta$› **by** *auto*
have $L - 4 * delta + 7 * QC\ (k+1) \leq 2 * dM - 5 * delta - 2 * QC\ k$
unfolding *QC-def L-def* **using** ‹$delta > 0$› *Laux I* ‹$C \geq 0$› **by** *auto*
also have ... $\leq 2\hat{\ }(k+1) * dM - 5 * delta - 2 * QC\ k$
using *aux* **by** (*auto simp add*: *algebra-simps*)
finally show $L - 4 * delta + 7 * QC\ (Suc\ k) \leq dist\ (q\ (Suc\ k)\ uM)\ (q\ (Suc\ k)\ w)$
using $*$ **by** *auto*
qed
then show *?thesis*
by *simp*
qed
qed
qed
have $dM > 0$ **using** *I* ‹$delta > 0$› ‹$C \geq 0$› *Laux* **by** *auto*
have $\exists k.\ 2\hat{\ }k > dist\ (f\ uM)\ (p\ uM)/dM + 1$
by (*simp add*: *real-arch-pow*)
then obtain *k* **where** $2\hat{\ }k > dist\ (f\ uM)\ (p\ uM)/dM + 1$
by *blast*
then have $dist\ (f\ uM)\ (p\ uM) < (2\hat{\ }k - 1) * dM$
using ‹$dM > 0$› **by** (*auto simp add*: *divide-simps algebra-simps*)
also have ... $\leq (2\hat{\ }(Suc\ k) - 1) * dM$
by (*intro mono-intros*, *auto*)
finally have $\neg((2\ \hat{\ }(k + 1) - 1) * dM \leq dist\ (f\ uM)\ (p\ uM))$
by *simp*
then show $Gromov\text{-}product\text{-}at\ (f\ z)\ (f\ um)\ (f\ uM) \leq lambda^2 * (D + 3/2 * L + delta + 11/2 * C) - 2 * delta + Kmult * (1 - exp\ (- K * (uM - um)))$

        using *Ind-k*[*of k*] **by** *auto*
      qed
    qed
  qed

The main induction is over. To conclude, one should apply its result to the original geodesic segment joining the points $f(a)$ and $f(b)$.

obtain *n::nat* **where** $(b - a)/((1/4) * delta / lambda) \le n$
  using *real-arch-simple* **by** *blast*
then have $b - a \le n * (1/4) * delta / lambda$
  using ‹*delta* > *0*› ‹*lambda* ≥ *1*› **by** (*auto simp add*: *divide-simps*)
have *infdist* (*f z*) *G* ≤ *Gromov-product-at* (*f z*) (*f a*) (*f b*) + *2* ∗ *deltaG*(*TYPE*(′*a*))
  apply (*intro mono-intros*) **using** *assms* **by** *auto*
also have ... ≤ (*lambda^2* ∗ (*D* + *3/2* ∗ *L* + *delta* + *11/2* ∗ *C*) − *2* ∗ *delta* + *Kmult* ∗ (*1* − *exp*(−*K* ∗ (*b* − *a*)))) + *2* ∗ *delta*
  apply (*intro mono-intros Main*[*OF* - - ‹*b* − *a* ≤ *n* ∗ (*1/4*) ∗ *delta* / *lambda*›]) **using** *assms* **by** *auto*
also have ... = *lambda^2* ∗ (*D* + *3/2* ∗ *L* + *delta* + *11/2* ∗ *C*) + *Kmult* ∗ (*1* − *exp*(−*K* ∗ (*b* − *a*)))
  by *simp*
also have ... ≤ *lambda^2* ∗ (*D* + *3/2* ∗ *L* + *delta* + *11/2* ∗ *C*) + *Kmult* ∗ (*1* − *0*)
  apply (*intro mono-intros*) **using** ‹*Kmult* > *0*› **by** *auto*
also have ... = *lambda^2* ∗ (*11/2* ∗ *C* + (*3200*∗*exp*(−*459/50*∗*ln 2*)/*ln 2* + *83*) ∗ *delta*)
  unfolding *Kmult-def K-def L-def alpha-def D-def* **using** ‹*delta* > *0*› ‹*lambda* ≥ *1*› **by** (*simp add*: *algebra-simps divide-simps power2-eq-square mult-exp-exp*)
also have ... ≤ *lambda^2* ∗ (*11/2* ∗ *C* + *91* ∗ *delta*)
  apply (*intro mono-intros*, *simp add*: *divide-simps*, *approximation 14*)
  using ‹*delta* > *0*› **by** *auto*
finally show *?thesis* **by** (*simp add*: *algebra-simps*)
qed

Still assuming that our quasi-isometry is Lipschitz, we will improve slightly on the previous result, first going down to the hyperbolicity constant of the space, and also showing that, conversely, the geodesic is contained in a neighborhood of the quasi-geodesic. The argument for this last point goes as follows. Consider a point x on the geodesic. Define two sets to be the D-thickenings of $[a, x]$ and $[x, b]$ respectively, where D is such that any point on the quasi-geodesic is within distance D of the geodesic (as given by the previous theorem). The union of these two sets covers the quasi-geodesic, and they are both closed and nonempty. By connectedness, there is a point z in their intersection, D-close both to a point x^- before x and to a point x^+ after x. Then x belongs to a geodesic between x^- and x^+, which is contained in a 4δ-neighborhood of geodesics from x^+ to z and from x^- to z by hyperbolicity. It follows that x is at distance at most $D + 4\delta$ of z, concluding the proof.

lemma (**in** *Gromov-hyperbolic-space-geodesic*) *Morse-Gromov-theorem-aux2*:
fixes $f::real \Rightarrow 'a$
assumes *continuous-on* $\{a..b\}$ f
lambda C*−quasi-isometry-on* $\{a..b\}$ f
geodesic-segment-between G $(f\ a)$ $(f\ b)$
shows *hausdorff-distance* $(f`\{a..b\})$ $G \leq lambda\hat{}2 * (11/2 * C + 92 * deltaG(TYPE('a)))$
proof (*cases* $a \leq b$)
case *True*
have $lambda \geq 1$ $C \geq 0$ **using** *quasi-isometry-onD*[*OF assms*(*2*)] **by** *auto*
have ∗: *infdist* $(f\ z)$ $G \leq lambda\hat{}2 * (11/2 * C + 91 * delta)$ **if** $z \in \{a..b\}$ $delta > deltaG(TYPE('a))$ **for** z *delta*
by (*rule Morse-Gromov-theorem-aux1*[*OF assms*(*1*) *assms*(*2*) *True assms*(*3*) *that*])
define D **where** $D = lambda\hat{}2 * (11/2 * C + 91 * deltaG(TYPE('a)))$
have $D \geq 0$ **unfolding** *D-def* **using** ‹$C \geq 0$› **by** *auto*
have *I*: *infdist* $(f\ z)$ $G \leq D$ **if** $z \in \{a..b\}$ **for** z
proof −
have $($*infdist* $(f\ z)$ $G/ lambda\hat{}2 - 11/2 * C)/91 \leq delta$ **if** $delta > deltaG(TYPE('a))$ **for** *delta*
using ∗[*OF* ‹$z \in \{a..b\}$› *that*] ‹$lambda \geq 1$› **by** (*auto simp add*: *divide-simps algebra-simps*)
then have $($*infdist* $(f\ z)$ $G/ lambda\hat{}2 - 11/2 * C)/91 \leq deltaG(TYPE('a))$
using *dense-ge* **by** *blast*
then show *?thesis* **unfolding** *D-def* **using** ‹$lambda \geq 1$› **by** (*auto simp add*: *divide-simps algebra-simps*)
qed
show *?thesis*
proof (*rule hausdorff-distanceI*)
show $0 \leq lambda^2 * (11/2 * C + 92 * deltaG\ TYPE('a))$ **using** ‹$C \geq 0$› **by** *auto*
fix x **assume** $x \in f`\{a..b\}$
then obtain z **where** z: $x = f\ z$ $z \in \{a..b\}$ **by** *blast*
show *infdist* x $G \leq lambda^2 * (11/2 * C + 92 * deltaG\ TYPE('a))$
unfolding *z*(*1*) **by** (*rule order-trans*[*OF I*[*OF* ‹$z \in \{a..b\}$›]], *auto simp add*: *algebra-simps D-def*)
next
fix x **assume** $x \in G$
have *infdist* x $(f`\{a..b\}) \leq D + 1 * deltaG\ TYPE('a)$
proof −
define p **where** $p =$ *geodesic-segment-param* G $(f\ a)$
then have p: $p\ 0 = f\ a$ $p\ (dist\ (f\ a)\ (f\ b)) = f\ b$
unfolding *p-def* **using** *assms*(*3*) **by** *auto*
obtain t **where** t: $x = p\ t$ $t \in \{0..dist\ (f\ a)\ (f\ b)\}$
unfolding *p-def* **using** ‹$x \in G$› ‹*geodesic-segment-between* G $(f\ a)$ $(f\ b)$› **by** (*metis geodesic-segment-param*(*5*) *imageE*)
define *Km* **where** $Km = (\bigcup z \in p`\{0..t\}.\ cball\ z\ D)$
define *KM* **where** $KM = (\bigcup z \in p`\{t..dist\ (f\ a)\ (f\ b)\}.\ cball\ z\ D)$
have $f`\{a..b\} \subseteq Km \cup KM$
proof

fix *x* **assume** *x*: $x \in f`\{a..b\}$
have $\exists z \in G.$ *infdist x G* = *dist x z*
apply (*rule infdist-proper-attained*)
using *geodesic-segment-topology*[*OF geodesic-segmentI*[*OF assms*(*3*)]] **by** *auto*
then obtain *z* **where** *z*: $z \in G$ *infdist x G* = *dist x z*
by *auto*
obtain *tz* **where** *tz*: *z* = *p tz tz* $\in$ {*0..dist* (*f a*) (*f b*)}
unfolding *p-def* **using** ‹$z \in G$› ‹*geodesic-segment-between G* (*f a*) (*f b*)› **by** (*metis geodesic-segment-param*(*5*) *imageE*)
have *infdist x G* $\leq$ *D*
using *I* ‹$x \in f`\{a..b\}$› **by** *auto*
then have *dist z x* $\leq$ *D*
using *z*(*2*) **by** (*simp add*: *metric-space-class.dist-commute*)
then show $x \in$ *Km* $\cup$ *KM*
unfolding *Km-def KM-def* **using** *tz* **by** *force*
qed
then have $*$: $f`\{a..b\}$ = (*Km* $\cap$ $f`\{a..b\}$) $\cup$ (*KM* $\cap$ $f`\{a..b\}$) **by** *auto*
have (*Km* $\cap$ $f`\{a..b\}$) $\cap$ (*KM* $\cap$ $f`\{a..b\}$) $\neq$ {}
proof (*rule connected-as-closed-union*[*OF - $*$*])
have *closed* (*f* ` {*a..b*})
apply (*intro compact-imp-closed compact-continuous-image*) **using** *assms*(*1*) **by** *auto*
have *closed Km*
unfolding *Km-def* **apply** (*intro compact-has-closed-thickening compact-continuous-image*)
apply (*rule continuous-on-subset*[*of* {*0..dist* (*f a*) (*f b*)} *p*])
unfolding *p-def* **using** *assms*(*3*) ‹$t \in$ {*0..dist* (*f a*) (*f b*)}› **by** (*auto simp add*: *isometry-on-continuous*)
then show *closed* (*Km* $\cap$ $f`\{a..b\}$)
by (*rule topological-space-class.closed-Int*) *fact*

have *closed KM*
unfolding *KM-def* **apply** (*intro compact-has-closed-thickening compact-continuous-image*)
apply (*rule continuous-on-subset*[*of* {*0..dist* (*f a*) (*f b*)} *p*])
unfolding *p-def* **using** *assms*(*3*) ‹$t \in$ {*0..dist* (*f a*) (*f b*)}› **by** (*auto simp add*: *isometry-on-continuous*)
then show *closed* (*KM* $\cap$ $f`\{a..b\}$)
by (*rule topological-space-class.closed-Int*) *fact*

show *connected* ($f`\{a..b\}$)
apply (*rule connected-continuous-image*) **using** *assms*(*1*) **by** *auto*
have *f a* $\in$ *Km* $\cap$ $f`\{a..b\}$ **using** *True* **apply** *auto*
unfolding *Km-def* **apply** *auto* **apply** (*rule bexI*[*of - 0*])
unfolding *p* **using** ‹$D \geq 0$› *t*(*2*) **by** *auto*
then show *Km* $\cap$ $f`\{a..b\}$ $\neq$ {} **by** *auto*
have *f b* $\in$ *KM* $\cap$ $f`\{a..b\}$ **apply** *auto*
unfolding *KM-def* **apply** *auto* **apply** (*rule bexI*[*of - dist* (*f a*) (*f b*)])

```
        unfolding p using ‹D ≥ 0› t(2) True by auto
      then show KM ∩ f`{a..b} ≠ {} by auto
    qed
    then obtain y where y: y ∈ f`{a..b} y ∈ Km y ∈ KM by auto
    obtain tm where tm: tm ∈ {0..t} dist (p tm) y ≤ D
      using y(2) unfolding Km-def by auto
    obtain tM where tM: tM ∈ {t..dist (f a) (f b)} dist (p tM) y ≤ D
      using y(3) unfolding KM-def by auto
    define H where H = p`{tm..tM}
    have ∗: geodesic-segment-between H (p tm) (p tM)
      unfolding H-def p-def apply (rule geodesic-segmentI2)
     using assms(3) ‹tm ∈ {0..t}› ‹tM ∈ {t..dist (f a) (f b)}› isometry-on-subset
      using assms(3) geodesic-segment-param(4) by (auto) fastforce
    have x ∈ H
       unfolding t(1) H-def using ‹tm ∈ {0..t}› ‹tM ∈ {t..dist (f a) (f b)}› by
auto
    have infdist x (f ` {a..b}) ≤ dist x y
      by (rule infdist-le[OF y(1)])
    also have ... ≤ max (dist (p tm) y) (dist (p tM) y) + deltaG(TYPE('a))
      by (rule dist-le-max-dist-triangle[OF ∗ ‹x ∈ H›])
    finally show ?thesis
      using tm(2) tM(2) by auto
  qed
  also have ... ≤ D + lambda^2 ∗ deltaG TYPE('a)
    apply (intro mono-intros) using ‹lambda ≥ 1› by auto
   finally show infdist x (f ` {a..b}) ≤ lambda^2 ∗ (11/2 ∗ C + 92 ∗ deltaG
TYPE('a))
    unfolding D-def by (simp add: algebra-simps)
qed
next
  case False
  then have f`{a..b} = {}
    by auto
  then have hausdorff-distance (f ` {a..b}) G = 0
    unfolding hausdorff-distance-def by auto
  then show ?thesis
    using quasi-isometry-onD(4)[OF assms(2)] by auto
qed
```

The full statement of the Morse-Gromov Theorem, asserting that a quasi-geodesic is within controlled distance of a geodesic with the same endpoints. It is given in the formulation of Shchur [Shc13], with optimal control in terms of the parameters of the quasi-isometry. This statement follows readily from the previous one and from the fact that quasi-geodesics can be approximated by Lipschitz ones.

```
theorem (in Gromov-hyperbolic-space-geodesic) Morse-Gromov-theorem:
  fixes f::real ⇒ 'a
  assumes lambda C−quasi-isometry-on {a..b} f
          geodesic-segment-between G (f a) (f b)
```

shows *hausdorff-distance* (*f*'{*a..b*}) *G* ≤ *92* ∗ *lambda*^*2* ∗ (*C* + *deltaG*(*TYPE*('*a*)))
proof −
  have *C*: *C* ≥ *0* *lambda* ≥ *1* **using** *quasi-isometry-onD*[*OF assms*(*1*)] **by** *auto*
  consider *dist* (*f a*) (*f b*) ≥ *2* ∗ *C* ∧ *a* ≤ *b* | *dist* (*f a*) (*f b*) ≤ *2* ∗ *C* ∧ *a* ≤ *b* | *b* < *a*
    by *linarith*
  then show *?thesis*
  proof (*cases*)
    case *1*
    have ∃ *d*. *continuous-on* {*a..b*} *d* ∧ *d a* = *f a* ∧ *d b* = *f b*
                ∧ (∀ *x*∈{*a..b*}. *dist* (*f x*) (*d x*) ≤ *4* ∗ *C*)
                ∧ *lambda* (*4* ∗ *C*)−*quasi-isometry-on* {*a..b*} *d*
                ∧ (*2* ∗ *lambda*)−*lipschitz-on* {*a..b*} *d*
                ∧ *hausdorff-distance* (*f*'{*a..b*}) (*d*'{*a..b*}) ≤ *2* ∗ *C*
      apply (*rule quasi-geodesic-made-lipschitz*[*OF assms*(*1*)]) **using** *1* **by** *auto*
    then obtain *d* **where** *d*: *d a* = *f a* *d b* = *f b*
                        ⋀*x*. *x* ∈ {*a..b*} ⟹ *dist* (*f x*) (*d x*) ≤ *4* ∗ *C*
                        lambda (*4* ∗ *C*)−*quasi-isometry-on* {*a..b*} *d*
                        (*2* ∗ *lambda*)−*lipschitz-on* {*a..b*} *d*
                        hausdorff-distance (*f*'{*a..b*}) (*d*'{*a..b*}) ≤ *2* ∗ *C*
      by *auto*
    have *a*: *hausdorff-distance* (*d*'{*a..b*}) *G* ≤ *lambda*^*2* ∗ ((*11*/*2*) ∗ (*4* ∗ *C*) + *92* ∗ *deltaG*(*TYPE*('*a*)))
      apply (*rule Morse-Gromov-theorem-aux2*) **using** *d assms lipschitz-on-continuous-on* **by** *auto*

    have *hausdorff-distance* (*f*'{*a..b*}) *G* ≤
          hausdorff-distance (*f*'{*a..b*}) (*d*'{*a..b*}) + *hausdorff-distance* (*d*'{*a..b*}) *G*
      apply (*rule hausdorff-distance-triangle*)
      using *1* **apply** *simp*
      by (*rule quasi-isometry-on-bounded*[*OF d*(*4*)], *auto*)
    also have ... ≤ *lambda*^*2* ∗ ((*11*/*2*) ∗ (*4* ∗ *C*) + *92* ∗ *deltaG*(*TYPE*('*a*))) + *1* ∗ *2* ∗ *C*
      using *a d* **by** *auto*
    also have ... ≤ *lambda*^*2* ∗ ((*11*/*2*) ∗ (*4* ∗ *C*) + *92* ∗ *deltaG*(*TYPE*('*a*))) + *lambda*^*2* ∗ *2* ∗ *C*
      apply (*intro mono-intros*) **using** ‹*lambda* ≥ *1*› ‹*C* ≥ *0*› **by** *auto*
    also have ... = *lambda*^*2* ∗ (*24* ∗ *C* + *92* ∗ *deltaG*(*TYPE*('*a*)))
      by (*simp add*: *algebra-simps divide-simps*)
    also have ... ≤ *lambda*^*2* ∗ (*92* ∗ *C* + *92* ∗ *deltaG*(*TYPE*('*a*)))
      apply (*intro mono-intros*) **using** ‹*lambda* ≥ *1*› ‹*C* ≥ *0*› **by** *auto*
    finally show *?thesis* **by** (*auto simp add*: *algebra-simps*)
  next
    case *2*
    have (*1*/*lambda*) ∗ *dist a b* − *C* ≤ *dist* (*f a*) (*f b*)
      apply (*rule quasi-isometry-onD*[*OF assms*(*1*)]) **using** *2* **by** *auto*
    also have ... ≤ *2* ∗ *C* **using** *2* **by** *auto*
    finally have *dist a b* ≤ *3* ∗ *lambda* ∗ *C*
      using *C* **by** (*auto simp add*: *algebra-simps divide-simps*)

then have ∗: *b* − *a* ≤ *3* ∗ *lambda* ∗ *C* **using** *2* **unfolding** *dist-real-def* **by** *auto*
show *?thesis*
proof (*rule hausdorff-distanceI2*)
show *0* ≤ *92* ∗ *lambda*^{2} ∗ (*C* + *deltaG TYPE*(*'a*)) **using** *C* **by** *auto*
fix *x* **assume** *x* ∈ *f*‘{*a*..*b*}
then obtain *t* **where** *t*: *x* = *f t t* ∈ {*a*..*b*} **by** *auto*
have *dist x* (*f a*) ≤ *lambda* ∗ *dist t a* + *C*
unfolding *t*(*1*) **using** *quasi-isometry-onD*(*1*)[*OF assms*(*1*) *t*(*2*)] *2* **by** *auto*
also have ... ≤ *lambda* ∗ (*b* − *a*) + *1* ∗ *1* ∗ *C* + *0* ∗ *0* ∗ *deltaG*(*TYPE*(*'a*)) **using** *t*(*2*) *2 C* **unfolding** *dist-real-def* **by** *auto*
also have ... ≤ *lambda* ∗ (*3* ∗ *lambda* ∗ *C*) + *lambda*^*2* ∗ (*92*−*3*) ∗ *C* + *lambda*^*2* ∗ *92* ∗ *deltaG*(*TYPE*(*'a*))
apply (*intro mono-intros* ∗) **using** *C* **by** *auto*
finally have ∗: *dist x* (*f a*) ≤ *92* ∗ *lambda*^{2} ∗ (*C* + *deltaG TYPE*(*'a*))
by (*simp add*: *algebra-simps power2-eq-square*)
show ∃ *y*∈*G*. *dist x y* ≤ *92* ∗ *lambda*^{2} ∗ (*C* + *deltaG TYPE*(*'a*))
apply (*rule bexI*[*of - f a*]) **using** ∗ *2 assms*(*2*) **by** *auto*
next
fix *x* **assume** *x* ∈ *G*
then have *dist x* (*f a*) ≤ *dist* (*f a*) (*f b*)
by (*meson assms geodesic-segment-dist-le geodesic-segment-endpoints*(*1*) *local.some-geodesic-is-geodesic-segment*(*1*))
also have ... ≤ *1* ∗ *2* ∗ *C* + *lambda*^*2* ∗ *0* ∗ *deltaG*(*TYPE*(*'a*))
using *2* **by** *auto*
also have ... ≤ *lambda*^*2* ∗ *92* ∗ *C* + *lambda*^*2* ∗ *92* ∗ *deltaG*(*TYPE*(*'a*))
apply (*intro mono-intros*) **using** *C* **by** *auto*
finally have ∗: *dist x* (*f a*) ≤ *92* ∗ *lambda*^{2} ∗ (*C* + *deltaG TYPE*(*'a*))
by (*simp add*: *algebra-simps*)
show ∃ *y*∈*f*‘{*a*..*b*}. *dist x y* ≤ *92* ∗ *lambda*^{2} ∗ (*C* + *deltaG TYPE*(*'a*))
apply (*rule bexI*[*of - f a*]) **using** ∗ *2* **by** *auto*
qed
next
case *3*
then have *hausdorff-distance* (*f* ‘ {*a*..*b*}) *G* = *0*
unfolding *hausdorff-distance-def* **by** *auto*
then show *?thesis*
using *C* **by** *auto*
qed
qed

This theorem implies the same statement for two quasi-geodesics sharing their endpoints.

theorem (**in** *Gromov-hyperbolic-space-geodesic*) *Morse-Gromov-theorem2*:
fixes *c d*::*real* ⇒ *'a*
assumes *lambda C*−*quasi-isometry-on* {*A*..*B*} *c*
lambda C−*quasi-isometry-on* {*A*..*B*} *d*
c A = *d A c B* = *d B*
shows *hausdorff-distance* (*c*‘{*A*..*B*}) (*d*‘{*A*..*B*}) ≤ *184* ∗ *lambda*^*2* ∗ (*C* +

```
deltaG(TYPE('a)))
proof (cases A ≤ B)
  case False
  then have hausdorff-distance (c`{A..B}) (d`{A..B}) = 0 by auto
   then show ?thesis using quasi-isometry-onD[OF assms(1)] delta-nonneg by
auto
next
  case True
   have hausdorff-distance (c`{A..B}) {c A−−c B} ≤ 92 ∗ lambda^2 ∗ (C +
deltaG(TYPE('a)))
    by (rule Morse-Gromov-theorem[OF assms(1)], auto)
  moreover have hausdorff-distance {c A−−c B} (d`{A..B}) ≤ 92 ∗ lambda^2 ∗
(C + deltaG(TYPE('a)))
    unfolding ‹c A = d A› ‹c B = d B› apply (subst hausdorff-distance-sym)
    by (rule Morse-Gromov-theorem[OF assms(2)], auto)
  moreover have hausdorff-distance (c`{A..B}) (d`{A..B}) ≤ hausdorff-distance
(c`{A..B}) {c A−−c B} + hausdorff-distance {c A−−c B} (d`{A..B})
    apply (rule hausdorff-distance-triangle)
    using True compact-imp-bounded[OF some-geodesic-compact] by auto
  ultimately show ?thesis by auto
qed
```

We deduce from the Morse lemma that hyperbolicity is invariant under quasi-isometry.

First, we note that the image of a geodesic segment under a quasi-isometry is close to a geodesic segment in Hausdorff distance, as it is a quasi-geodesic.

```
lemma geodesic-quasi-isometric-image:
  fixes f::'a::metric-space ⇒ 'b::Gromov-hyperbolic-space-geodesic
  assumes lambda C−quasi-isometry-on UNIV f
          geodesic-segment-between G x y
 shows hausdorff-distance (f`G) {f x−−f y} ≤ 92 ∗ lambda^2 ∗ (C + deltaG(TYPE('b)))
proof −
  define c where c = f o (geodesic-segment-param G x)
  have ∗: (1 ∗ lambda) (0 ∗ lambda + C)−quasi-isometry-on {0..dist x y} c
   unfolding c-def by (rule quasi-isometry-on-compose[where Y = UNIV], auto
intro!: isometry-quasi-isometry-on simp add: assms)
  have hausdorff-distance (c`{0..dist x y}) {c 0−−c (dist x y)} ≤ 92 ∗ lambda^2
∗ (C + deltaG(TYPE('b)))
    apply (rule Morse-Gromov-theorem) using ∗ by auto
  moreover have c`{0..dist x y} = f`G
    unfolding c-def image-comp[symmetric] using assms(2) by auto
  moreover have c 0 = f x c (dist x y) = f y
    unfolding c-def using assms(2) by auto
  ultimately show ?thesis by auto
qed
```

We deduce that hyperbolicity is invariant under quasi-isometry. The proof goes as follows: we want to see that a geodesic triangle is delta-thin, i.e., a point on a side Gxy is close to the union of the two other sides Gxz and Gyz. Pull everything back by the quasi-isometry: we obtain three quasi-geodesic, each of which is close to the corresponding geodesic segment by the Morse lemma. As the geodesic triangle is thin, it follows that the quasi-geodesic triangle is also thin, i.e., a point on $f^{-1}Gxy$ is close to $f^{-1}Gxz \cup f^{-1}Gyz$ (for some explicit, albeit large, constant). Then push everything forward by f: as it is a quasi-isometry, it will again distort distances by a bounded amount.

lemma *Gromov-hyperbolic-invariant-under-quasi-isometry-explicit*:
  fixes *f*::*'a*::*geodesic-space* ⇒ *'b*::*Gromov-hyperbolic-space-geodesic*
  assumes *lambda C−quasi-isometry f*
  shows *Gromov-hyperbolic-subset* (*752* ∗ *lambda^3* ∗ (*C* + *deltaG*(*TYPE*(*'b*)))) (*UNIV*::(*'a set*))
proof −
  have *C*: *lambda* ≥ *1 C* ≥ *0*
    using *quasi-isometry-onD*[*OF assms*] **by** *auto*

The Morse lemma gives a control bounded by K below. Following the proof, we deduce a bound on the thinness of triangles by an ugly constant L. We bound it by a more tractable (albeit still ugly) constant M.

  define *K* **where** *K* = *92* ∗ *lambda^2* ∗ (*C* + *deltaG*(*TYPE*(*'b*)))
  have *HD*: *hausdorff-distance* (*f'G*) {*f a*−−*f b*} ≤ *K* **if** *geodesic-segment-between G a b* **for** *G a b*
    unfolding *K-def* **by** (*rule geodesic-quasi-isometric-image*[*OF assms that*])
  define *L* **where** *L* = *lambda* ∗ (*4* ∗ *1* ∗ *deltaG*(*TYPE*(*'b*)) + *1* ∗ *1* ∗ *C* + *2* ∗ *K*)
  define *M* **where** *M* = *188* ∗ *lambda^3* ∗ (*C* + *deltaG*(*TYPE*(*'b*)))

  have *L* ≤ *lambda* ∗ (*4* ∗ *lambda^2* ∗ *deltaG*(*TYPE*(*'b*)) + *4* ∗ *lambda^2* ∗ *C* + *2* ∗ *K*)
    unfolding *L-def* **apply** (*intro mono-intros*) **using** *C* **by** *auto*
  also have ... = *M*
    unfolding *M-def K-def* **by** (*auto simp add*: *algebra-simps power2-eq-square power3-eq-cube*)
  finally have *L* ≤ *M* **by** *simp*

After these preliminaries, we start the real argument per se, showing that triangles are thin in the type b.

  have *Thin*: *infdist w* (*Gxz* ∪ *Gyz*) ≤ *M* **if**
   H: *geodesic-segment-between Gxy x y geodesic-segment-between Gxz x z geodesic-segment-between Gyz y z w* ∈ *Gxy*
    for *w x y z*::*'a* **and** *Gxy Gyz Gxz*
  proof −
    obtain *w2* **where** *w2*: *w2* ∈ {*f x*−−*f y*} *infdist* (*f w*) {*f x*−−*f y*} = *dist* (*f w*) *w2*
      using *infdist-proper-attained*[*OF proper-of-compact*, *of* {*f x*−−*f y*} *f w*] **by** *auto*

have *dist* (*f w*) *w2* = *infdist* (*f w*) {*f x*−− *f y*}
using *w2* **by** *simp*
also have ... ≤ *hausdorff-distance* (*f'Gxy*) {*f x*−− *f y*}
using *geodesic-segment-topology*(*4*)[*OF geodesic-segmentI*] *H*
by (*auto intro*!: *quasi-isometry-on-bounded*[*OF quasi-isometry-on-subset*[*OF assms*]] *infdist-le-hausdorff-distance*)
also have ... ≤ *K* **using** *HD*[*OF H*(*1*)] **by** *simp*
finally have ∗: *dist* (*f w*) *w2* ≤ *K* **by** *simp*

have *infdist w2* (*f'Gxz* ∪ *f'Gyz*) ≤ *infdist w2* ({*f x*−−*f z*} ∪ {*f y*−−*f z*})
+ *hausdorff-distance* ({*f x*−−*f z*} ∪ {*f y*−−*f z*}) (*f'Gxz* ∪ *f'Gyz*)
apply (*rule hausdorff-distance-infdist-triangle*)
using *geodesic-segment-topology*(*4*)[*OF geodesic-segmentI*] *H*
by (*auto intro*!: *quasi-isometry-on-bounded*[*OF quasi-isometry-on-subset*[*OF assms*]])
also have ... ≤ *4* ∗ *deltaG*(*TYPE*(*'b*)) + *hausdorff-distance* ({*f x*−−*f z*} ∪ {*f y*−−*f z*}) (*f'Gxz* ∪ *f'Gyz*)
apply (*simp*, *rule thin-triangles*[*of* {*f x*−−*f z*} *f z f x* {*f y*−−*f z*} *f y* {*f x*−−*f y*} *w2*])
using *w2* **apply** *auto*
using *geodesic-segment-commute some-geodesic-is-geodesic-segment*(*1*) **by** *blast*+
also have ... ≤ *4* ∗ *deltaG*(*TYPE*(*'b*)) + *max* (*hausdorff-distance* {*f x*−−*f z*} (*f'Gxz*)) (*hausdorff-distance* {*f y*−−*f z*} (*f'Gyz*))
apply (*intro mono-intros*) **using** *H* **by** *auto*
also have ... ≤ *4* ∗ *deltaG*(*TYPE*(*'b*)) + *K*
using *HD*[*OF H*(*2*)] *HD*[*OF H*(*3*)] **by** (*auto simp add*: *hausdorff-distance-sym*)
finally have ∗∗: *infdist w2* (*f'Gxz* ∪ *f'Gyz*) ≤ *4* ∗ *deltaG*(*TYPE*(*'b*)) + *K* **by** *simp*

have *infdist* (*f w*) (*f'Gxz* ∪ *f'Gyz*) ≤ *infdist w2* (*f'Gxz* ∪ *f'Gyz*) + *dist* (*f w*) *w2*
by (*rule infdist-triangle*)
then have *A*: *infdist* (*f w*) (*f'*(*Gxz* ∪ *Gyz*)) ≤ *4* ∗ *deltaG*(*TYPE*(*'b*)) + *2* ∗ *K*
using ∗ ∗∗ **by** (*auto simp add*: *image-Un*)

have *infdist w* (*Gxz* ∪ *Gyz*) ≤ *L* + *epsilon* **if** *epsilon* > *0* **for** *epsilon*
proof −
have ∗: *epsilon*/*lambda* > *0* **using** *that C* **by** *auto*
have ∃ *z* ∈ *f'*(*Gxz* ∪ *Gyz*). *dist* (*f w*) *z* < *4* ∗ *deltaG*(*TYPE*(*'b*)) + *2* ∗ *K* + *epsilon*/*lambda*
apply (*rule infdist-almost-attained*)
using *A* ∗ *H*(*2*) **by** *auto*
then obtain *z* **where** *z*: *z* ∈ *Gxz* ∪ *Gyz dist* (*f w*) (*f z*) < *4* ∗ *deltaG*(*TYPE*(*'b*)) + *2* ∗ *K* + *epsilon*/*lambda*
by *auto*

have *infdist w* (*Gxz* ∪ *Gyz*) ≤ *dist w z*
by (*auto intro*!: *infdist-le z*(*1*))

also have ... $\leq$ *lambda* $*$ *dist* (*f w*) (*f z*) + *C* $*$ *lambda*
using *quasi-isometry-onD*[*OF assms*] **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... $\leq$ *lambda* $*$ (*4* $*$ *deltaG*(*TYPE*('*b*)) + *2* $*$ *K* + *epsilon*/*lambda*) + *C* $*$ *lambda*
apply (*intro mono-intros*) **using** *z*(*2*) *C* **by** *auto*
also have ... = *L* + *epsilon*
unfolding *K-def L-def* **using** *C* **by** (*auto simp add*: *algebra-simps*)
finally show *?thesis* **by** *simp*
qed
then have *infdist w* (*Gxz* $\cup$ *Gyz*) $\leq$ *L*
using *field-le-epsilon* **by** *blast*
then show *?thesis*
using ‹*L* $\leq$ *M*› **by** *auto*
qed
then have *Gromov-hyperbolic-subset* (*4* $*$ *M*) (*UNIV*::'*a set*)
using *thin-triangles-implies-hyperbolic*[*OF Thin*] **by** *auto*
then show *?thesis* **unfolding** *M-def* **by** (*auto simp add*: *algebra-simps*)
qed

Most often, the precise value of the constant in the previous theorem is irrelevant, it is used in the following form.

theorem *Gromov-hyperbolic-invariant-under-quasi-isometry*:
assumes *quasi-isometric* (*UNIV*::('*a*::*geodesic-space*) *set*) (*UNIV*::('*b*::*Gromov-hyperbolic-space-geodesic*) *set*)
shows $\exists$ *delta*. *Gromov-hyperbolic-subset delta* (*UNIV*::'*a set*)
proof −
obtain *C lambda f* **where** *f*: *lambda C*−*quasi-isometry-between* (*UNIV*::'*a set*) (*UNIV*::'*b set*) *f*
using *assms* **unfolding** *quasi-isometric-def* **by** *auto*
show *?thesis*
using *Gromov-hyperbolic-invariant-under-quasi-isometry-explicit*[*OF quasi-isometry-betweenD*(*1*)[*OF f*]] **by** *blast*
qed

A central feature of hyperbolic spaces is that a path from x to y can not deviate too much from a geodesic from x to y unless it is extremely long (exponentially long in terms of the distance from x to y). This is useful both to ensure that short paths (for instance quasi-geodesics) stay close to geodesics, see the Morse lemme below, and to ensure that paths that avoid a given large ball of radius R have to be exponentially long in terms of R (this is extremely useful for random walks). This proposition is the first non-trivial result on hyperbolic spaces in [BH99] (Proposition III.H.1.6). We follow their proof.

The proof is geometric, and uses the existence of geodesics and the fact that geodesic triangles are thin. In fact, the result still holds if the space is not geodesic, as it can be deduced by embedding the hyperbolic space in a geodesic hyperbolic space and using the result there.

proposition (**in** *Gromov-hyperbolic-space-geodesic*) *lipschitz-path-close-to-geodesic*:
fixes *c*::*real* ⇒ *'a*
assumes *M*−*lipschitz-on* {*A*..*B*} *c*
geodesic-segment-between G (*c A*) (*c B*)
x ∈ *G*
shows *infdist x* (*c*‘{*A*..*B*}) ≤ (*4*/*ln 2*) ∗ *deltaG*(*TYPE*(*'a*)) ∗ *max 0* (*ln* (*B*−*A*)) + *M*
proof −
have *M* ≥ *0* **by** (*rule lipschitz-on-nonneg*[*OF assms*(*1*)])
have *Main*: *a* ∈ {*A*..*B*} ⟹ *b* ∈ {*A*..*B*} ⟹ *a* ≤ *b* ⟹ *b*−*a* ≤ *2*^(*n*+*1*) ⟹ *geodesic-segment-between H* (*c a*) (*c b*)
⟹ *y* ∈ *H* ⟹ *infdist y* (*c*‘{*A*..*B*}) ≤ *4* ∗ *deltaG*(*TYPE*(*'a*)) ∗ *n* + *M* **for** *a b H y n*
proof (*induction n arbitrary*: *a b H y*)
case *0*
have *infdist y* (*c* ‘ {*A*..*B*}) ≤ *dist y* (*c b*)
apply (*rule infdist-le*) **using** ‹*b* ∈ {*A*..*B*}› **by** *auto*
moreover have *infdist y* (*c* ‘ {*A*..*B*}) ≤ *dist y* (*c a*)
apply (*rule infdist-le*) **using** ‹*a* ∈ {*A*..*B*}› **by** *auto*
ultimately have *2* ∗ *infdist y* (*c* ‘ {*A*..*B*}) ≤ *dist* (*c a*) *y* + *dist y* (*c b*)
by (*auto simp add*: *metric-space-class.dist-commute*)
also have ... = *dist* (*c a*) (*c b*)
by (*rule geodesic-segment-dist*[*OF* ‹*geodesic-segment-between H* (*c a*) (*c b*)› ‹*y* ∈ *H*›])
also have ... ≤ *M* ∗ *abs*(*b* − *a*)
using *lipschitz-onD*(*1*)[*OF assms*(*1*) ‹*a* ∈ {*A*..*B*}› ‹*b* ∈ {*A*..*B*}›] **unfolding** *dist-real-def*
by (*simp add*: *abs-minus-commute*)
also have ... ≤ *M* ∗ *2*
using ‹*a* ≤ *b*› ‹*b* − *a* ≤ *2*^(*0* + *1*)› ‹*M* ≥ *0*› *mult-left-mono* **by** *auto*
finally show *?case* **by** *simp*
next
case (*Suc n*)
define *m* **where** *m* = (*a* + *b*)/*2*
have *m* ∈ {*A*..*B*} **using** ‹*a* ∈ {*A*..*B*}› ‹*b* ∈ {*A*..*B*}› **unfolding** *m-def* **by** *auto*
define *Ha* **where** *Ha* = {*c m*−−*c a*}
define *Hb* **where** *Hb* = {*c m*−−*c b*}
have *I*: *geodesic-segment-between Ha* (*c m*) (*c a*) *geodesic-segment-between Hb* (*c m*) (*c b*)
unfolding *Ha-def Hb-def* **by** *auto*
then have *Ha* ≠ {} *Hb* ≠ {} *compact Ha compact Hb*
by (*auto intro*: *geodesic-segment-topology*)

have ∗: *infdist y* (*Ha* ∪ *Hb*) ≤ *4* ∗ *deltaG*(*TYPE*(*'a*))
by (*rule thin-triangles*[*OF I* ‹*geodesic-segment-between H* (*c a*) (*c b*)› ‹*y* ∈ *H*›])
then have *infdist y Ha* ≤ *4* ∗ *deltaG*(*TYPE*(*'a*)) ∨ *infdist y Hb* ≤ *4* ∗ *deltaG*(*TYPE*(*'a*))
unfolding *infdist-union-min*[*OF* ‹*Ha* ≠ {}› ‹*Hb* ≠ {}›] **by** *auto*

then show *?case*
proof
assume H: *infdist y Ha* $\leq$ *4* $*$ *deltaG TYPE('a)*
obtain z **where** z: $z \in Ha$ *infdist y Ha* $=$ *dist y z*
using *infdist-proper-attained*[*OF proper-of-compact*[*OF* ‹*compact Ha*›] ‹*Ha* $\neq$ {}›] **by** *auto*
have *Iz*: *infdist z* (*c*'{*A..B*}) $\leq$ *4* $*$ *deltaG*(*TYPE('a)*) $*$ *n* $+$ *M*
proof (*rule Suc.IH*[*OF* ‹*a* $\in$ {*A..B*}› ‹*m* $\in$ {*A..B*}›, *of Ha*])
show *a* $\leq$ *m* **unfolding** *m-def* **using** ‹*a* $\leq$ *b*› **by** *auto*
show *m* $-$ *a* $\leq$ *2*^(*n+1*) **using** ‹*b* $-$ *a* $\leq$ *2*^(*Suc n* $+$ *1*)› ‹*a* $\leq$ *b*› **unfolding** *m-def* **by** *auto*
show *geodesic-segment-between Ha* (*c a*) (*c m*) **by** (*simp add*: *I*(*1*) *geodesic-segment-commute*)
show $z \in Ha$ **using** z **by** *auto*
qed
have *infdist y* (*c*'{*A..B*}) $\leq$ *dist y z* $+$ *infdist z* (*c*'{*A..B*})
by (*metis add.commute infdist-triangle*)
also have ... $\leq$ *4* $*$ *deltaG TYPE('a)* $+$ (*4* $*$ *deltaG*(*TYPE('a)*) $*$ *n* $+$ *M*)
using *H z Iz* **by** (*auto intro*: *add-mono*)
finally show *infdist y* (*c* ' {*A..B*}) $\leq$ *4* $*$ *deltaG TYPE('a)* $*$ *real* (*Suc n*) $+$ *M*
by (*auto simp add*: *algebra-simps*)
next
assume H: *infdist y Hb* $\leq$ *4* $*$ *deltaG TYPE('a)*
obtain z **where** z: $z \in Hb$ *infdist y Hb* $=$ *dist y z*
using *infdist-proper-attained*[*OF proper-of-compact*[*OF* ‹*compact Hb*›] ‹*Hb* $\neq$ {}›] **by** *auto*
have *Iz*: *infdist z* (*c*'{*A..B*}) $\leq$ *4* $*$ *deltaG*(*TYPE('a)*) $*$ *n* $+$ *M*
proof (*rule Suc.IH*[*OF* ‹*m* $\in$ {*A..B*}› ‹*b* $\in$ {*A..B*}›, *of Hb*])
show *m* $\leq$ *b* **unfolding** *m-def* **using** ‹*a* $\leq$ *b*› **by** *auto*
show *b* $-$ *m* $\leq$ *2*^(*n+1*) **using** ‹*b* $-$ *a* $\leq$ *2*^(*Suc n* $+$ *1*)› ‹*a* $\leq$ *b*›
unfolding *m-def* **by** (*auto simp add*: *divide-simps*)
show *geodesic-segment-between Hb* (*c m*) (*c b*) **by** (*simp add*: *I*(*2*))
show $z \in Hb$ **using** z **by** *auto*
qed
have *infdist y* (*c*'{*A..B*}) $\leq$ *dist y z* $+$ *infdist z* (*c*'{*A..B*})
by (*metis add.commute infdist-triangle*)
also have ... $\leq$ *4* $*$ *deltaG TYPE('a)* $+$ (*4* $*$ *deltaG*(*TYPE('a)*) $*$ *n* $+$ *M*)
using *H z Iz* **by** (*auto intro*: *add-mono*)
finally show *infdist y* (*c* ' {*A..B*}) $\leq$ *4* $*$ *deltaG TYPE('a)* $*$ *real* (*Suc n*) $+$ *M*
by (*auto simp add*: *algebra-simps*)
qed
qed
consider *B*$-$*A* $<$*0* | *B*$-$*A* $\geq$ *0* $\wedge$ *B*$-$*A* $\leq$ *2* | *B*$-$*A* $>$ *2* **by** *linarith*
then show *?thesis*
proof (*cases*)
case *1*
then have *c*'{*A..B*} $=$ {} **by** *auto*

then show *?thesis* **unfolding** *infdist-def* **using** ‹$M \geq 0$› **by** *auto*
next
case *2*
have *infdist x* $(c`\{A..B\}) \leq 4 * deltaG(TYPE('a)) * real\ 0 + M$
apply (*rule Main*[*OF - - - -* ‹*geodesic-segment-between G* (*c A*) (*c B*)› ‹$x \in G$›])
using *2* **by** *auto*
also have ... $\leq (4/ln\ 2) * deltaG(TYPE('a)) * max\ 0\ (ln\ (B-A)) + M$
using *delta-nonneg* **by** *auto*
finally show *?thesis* **by** *auto*
next
case *3*
define *n*::*nat* **where** $n = nat(floor\ (log\ 2\ (B-A)))$
have $log\ 2\ (B-A) > 0$ **using** *3* **by** *auto*
then have *n*: $n \leq log\ 2\ (B-A)$ $log\ 2\ (B-A) < n+1$
unfolding *n-def* **by** (*auto simp add*: *floor-less-cancel*)
then have ∗: $B-A \leq 2\hat{}(n+1)$
by (*meson le-log-of-power linear not-less one-less-numeral-iff semiring-norm*(*76*))
have $n \leq ln\ (B-A) * (1/ln\ 2)$ **using** *n* **unfolding** *log-def* **by** *auto*
then have $n \leq (1/ln\ 2) * max\ 0\ (ln\ (B-A))$
using *3* **by** (*auto simp add*: *algebra-simps divide-simps*)
have *infdist x* $(c`\{A..B\}) \leq 4 * deltaG(TYPE('a)) * n + M$
apply (*rule Main*[*OF - - - -* ‹*geodesic-segment-between G* (*c A*) (*c B*)› ‹$x \in G$›])
using ∗ *3* **by** *auto*
also have ... $\leq 4 * deltaG(TYPE('a)) * ((1/ln\ 2) * max\ 0\ (ln\ (B-A))) + M$
apply (*intro mono-intros*) **using** ‹$n \leq (1/ln\ 2) * max\ 0\ (ln\ (B-A))$› *delta-nonneg* **by** *auto*
finally show *?thesis* **by** *auto*
qed
qed

By rescaling coordinates at the origin, one obtains a variation around the previous statement.

proposition (**in** *Gromov-hyperbolic-space-geodesic*) *lipschitz-path-close-to-geodesic'*:
fixes *c*::*real* ⇒ *'a*
assumes $M-$*lipschitz-on* $\{A..B\}$ *c*
geodesic-segment-between G (*c A*) (*c B*)
$x \in G$
$a > 0$
shows *infdist x* $(c`\{A..B\}) \leq (4/ln\ 2) * deltaG(TYPE('a)) * max\ 0\ (ln\ (a * (B-A))) + M/a$
proof −
define *d* **where** $d = c\ o\ (\lambda t.\ (1/a) * t)$
have ∗: $(M * ((1/a)* 1))-$*lipschitz-on* $\{a * A..a * B\}$ *d*
unfolding *d-def* **apply** (*rule lipschitz-on-compose*, *intro lipschitz-intros*) **using** *assms* **by** *auto*
have $d`\{a * A..a * B\} = c`\{A..B\}$
unfolding *d-def image-comp*[*symmetric*]

apply (*rule arg-cong*[**where** *?f* = *image c*]) **using** ‹*a* > *0*› **by** *auto*
then have *infdist x* (*c*‘{*A..B*}) = *infdist x* (*d*‘{*a* ∗ *A..a* ∗ *B*}) **by** *auto*
also have ... ≤ (*4*/*ln 2*) ∗ *deltaG*(*TYPE*(′*a*)) ∗ *max 0* (*ln* ((*a* ∗ *B*)− (*a* ∗ *A*))) + *M*/*a*
apply (*rule lipschitz-path-close-to-geodesic*[*OF* - - ‹*x* ∈ *G*›])
using ∗ *assms* **unfolding** *d-def* **by** *auto*
finally show *?thesis* **by** (*auto simp add*: *algebra-simps*)
qed

We can now give another proof of the Morse-Gromov Theorem, as described in [BH99]. It is more direct than the one we have given above, but it gives a worse dependence in terms of the quasi-isometry constants. In particular, when $C = \delta = 0$, it does not recover the fact that a quasi-geodesic has to coincide with a geodesic.

theorem (**in** *Gromov-hyperbolic-space-geodesic*) *Morse-Gromov-theorem-BH-proof*:
fixes *c*::*real* ⇒ ′*a*
assumes *lambda C*−*quasi-isometry-on* {*A..B*} *c*
shows *hausdorff-distance* (*c*‘{*A..B*}) {*c A*−−*c B*} ≤ *72* ∗ *lambda*ˆ*2* ∗ (*C* + *lambda* + *deltaG*(*TYPE*(′*a*))ˆ*2*)
proof −
have *C*: *C* ≥ *0 lambda* ≥ *1* **using** *quasi-isometry-onD*[*OF assms*] **by** *auto*
consider *B*−*A* < *0* | *B*−*A* ≥ *0* ∧ *dist* (*c A*) (*c B*) ≤ *2* ∗ *C* | *B*−*A* ≥ *0* ∧ *dist* (*c A*) (*c B*) > *2* ∗ *C* **by** *linarith*
then show *?thesis*
proof (*cases*)
case *1*
then have *c*‘{*A..B*} = {} **by** *auto*
then show *?thesis* **unfolding** *hausdorff-distance-def* **using** *delta-nonneg C* **by** *auto*
next
case *2*
have (*1*/*lambda*) ∗ *dist A B* − *C* ≤ *dist* (*c A*) (*c B*)
apply (*rule quasi-isometry-onD*[*OF assms*]) **using** *2* **by** *auto*
also have ... ≤ *2* ∗ *C* **using** *2* **by** *auto*
finally have *dist A B* ≤ *3* ∗ *lambda* ∗ *C*
using *C* **by** (*auto simp add*: *algebra-simps divide-simps*)
then have ∗: *B* − *A* ≤ *3* ∗ *lambda* ∗ *C* **using** *2* **unfolding** *dist-real-def* **by** *auto*
show *?thesis*
proof (*rule hausdorff-distanceI2*)
show *0* ≤ *72* ∗ *lambda*ˆ*2* ∗ (*C* + *lambda* + *deltaG*(*TYPE*(′*a*))ˆ*2*) **using** *C* **by** *auto*
fix *x* **assume** *x* ∈ *c*‘{*A..B*}
then obtain *t* **where** *t*: *x* = *c t t* ∈ {*A..B*} **by** *auto*
have *dist x* (*c A*) ≤ *lambda* ∗ *dist t A* + *C*
unfolding *t*(*1*) **using** *quasi-isometry-onD*(*1*)[*OF assms t*(*2*), *of A*] *2* **by** *auto*
also have ... ≤ *lambda* ∗ (*B*−*A*) + *C* **using** *t*(*2*) *2 C* **unfolding** *dist-real-def* **by** *auto*

also have ... ≤ *3 ∗ lambda ∗ lambda ∗ C + 1 ∗ 1 ∗ C* **using** *∗ C* **by** *auto*
also have ... ≤ *3 ∗ lambda ∗ lambda ∗ C + lambda ∗ lambda ∗ C*
apply (*intro mono-intros*) **using** *C* **by** *auto*
also have ... = *4 ∗ lambda ∗ lambda ∗ (C + 0 + 0^2)*
by *auto*
also have ... ≤ *72 ∗ lambda ∗ lambda ∗ (C + lambda + deltaG(TYPE('a))^2)*
apply (*intro mono-intros*) **using** *C delta-nonneg* **by** *auto*
finally have ∗: *dist x (c A) ≤ 72 ∗ lambda^2 ∗ (C + lambda + deltaG(TYPE('a))^2)*
unfolding *power2-eq-square* **by** *simp*
show *∃ y∈{c A−−c B}. dist x y ≤ 72 ∗ lambda^2 ∗ (C + lambda + deltaG(TYPE('a))^2)*
apply (*rule bexI[of - c A]*) **using** ∗ **by** *auto*
next
fix *x* **assume** *x ∈ {c A−− c B}*
then have *dist x (c A) ≤ dist (c A) (c B)*
by (*meson geodesic-segment-dist-le geodesic-segment-endpoints(1) local.some-geodesic-is-geodesic-segment*
also have ... ≤ *2 ∗ C*
using *2* **by** *auto*
also have ... ≤ *2 ∗ 1 ∗ 1 ∗ (C + lambda + 0)* **using** *2 C* **unfolding** *dist-real-def* **by** *auto*
also have ... ≤ *72 ∗ lambda ∗ lambda ∗ (C + lambda + deltaG(TYPE('a)) ∗ deltaG(TYPE('a)))*
apply (*intro mono-intros*) **using** *C delta-nonneg* **by** *auto*
finally have ∗: *dist x (c A) ≤ 72 ∗ lambda ∗ lambda ∗ (C + lambda + deltaG(TYPE('a)) ∗ deltaG(TYPE('a)))*
by *simp*
show *∃ y∈c'{A..B}. dist x y ≤ 72 ∗ lambda^2 ∗ (C + lambda + deltaG(TYPE('a))^2)*
apply (*rule bexI[of - c A]*) **unfolding** *power2-eq-square* **using** *∗ 2* **by** *auto*
qed
next
case *3*
then obtain *d* **where** *d*: *continuous-on {A..B} d d A = c A d B = c B*
⋀x. x ∈ {A..B} ⟹ dist (c x) (d x) ≤ 4 ∗C
lambda (4 ∗ C)−quasi-isometry-on {A..B} d
(2 ∗ lambda)−lipschitz-on {A..B} d
hausdorff-distance (c'{A..B}) (d'{A..B}) ≤ 2 ∗ C
using *quasi-geodesic-made-lipschitz[OF assms] C(1)* **by** *fastforce*

have *A ∈ {A..B} B ∈ {A..B}* **using** *3* **by** *auto*

We show that the distance of any point in the geodesic from $c(A)$ to $c(B)$ is a bounded distance away from the quasi-geodesic d, by considering a point x where the distance D is maximal and arguing around this point.

Consider the point x_m on the geodesic $[c(A), c(B)]$ at distance $2D$ from x, and the closest point y_m on the image of d. Then the distance between x_m and y_m is at most D. Hence a point on $[x_m, y_m]$ is at distance at least $2D - D = D$ of x. In the same way, define x_M and y_M on the other side of x. Then the excursion from x_m to y_m, then to y_M along d, then to x_M, has length at most $D + (\lambda \cdot 6D + C) + D$ and is always at distance at least D from x. It follows from the previous lemma that $D \leq \log(length)$, which implies a bound on D.

This argument has to be amended if x is at distance $< 2D$ from $c(A)$ or $c(B)$. In this case, simply use $x_m = y_m = c(A)$ or $x_M = y_M = c(B)$, then everything goes through.

have $\exists x \in \{c\ A{-}{-}c\ B\}.\ \forall y \in \{c\ A{-}{-}c\ B\}.$ *infdist y* $(d`\{A..B\}) \leq$ *infdist x* $(d`\{A..B\})$
by (*rule continuous-attains-sup*, *auto intro*: *continuous-intros*)
then obtain *x* **where** *x*: $x \in \{c\ A{-}{-}c\ B\}$ $\bigwedge y.\ y \in \{c\ A{-}{-}c\ B\} \Longrightarrow$ *infdist y* $(d`\{A..B\}) \leq$ *infdist x* $(d`\{A..B\})$
by *auto*
define *D* **where** $D =$ *infdist x* $(d`\{A..B\})$
have $D \geq 0$ **unfolding** *D-def* **by** (*rule infdist-nonneg*)
have *D-bound*: $D \leq 24 * lambda + 12 * C + 24 * deltaG(TYPE('a))\hat{}2$
proof (*cases* $D \leq 1$)
case *True*
have $1 * 1 + 1 * 0 + 0 * 0 \leq 24 * lambda + 12 * C + 24 * deltaG(TYPE('a))\hat{}2$
apply (*intro mono-intros*) **using** *C delta-nonneg* **by** *auto*
then show *?thesis* **using** *True* **by** *auto*
next
case *False*
then have $D \geq 1$ **by** *auto*
have *ln2mult*: $2 * ln\ t = ln\ (t * t)$ **if** $t > 0$ **for** *t::real* **by** (*simp add*: *that ln-mult*)
have *infdist* $(c\ A)$ $(d`\{A..B\}) = 0$ **using** ‹$d\ A = c\ A$› **by** (*metis* ‹$A \in \{A..B\}$› *image-eqI infdist-zero*)
then have $x \neq c\ A$ **using** ‹$D \geq 1$› *D-def* **by** *auto*

define *tx* **where** $tx = dist\ (c\ A)\ x$
then have $tx \in \{0..dist\ (c\ A)\ (c\ B)\}$
using ‹$x \in \{c\ A{-}{-}c\ B\}$›
by (*meson atLeastAtMost-iff geodesic-segment-dist-le some-geodesic-is-geodesic-segment*(*1*) *metric-space-class.zero-le-dist some-geodesic-endpoints*(*1*))
have $tx > 0$ **using** ‹$x \neq c\ A$› *tx-def* **by** *auto*
have *x-param*: $x =$ *geodesic-segment-param* $\{c\ A{-}{-}c\ B\}$ $(c\ A)$ *tx*
using ‹$x \in \{c\ A{-}{-}c\ B\}$› *geodesic-segment-param*[*OF some-geodesic-is-geodesic-segment*(*1*)] *tx-def* **by** *auto*

define *tm* **where** $tm = max\ (tx - 2 * D)\ 0$
have $tm \in \{0..dist\ (c\ A)\ (c\ B)\}$ **unfolding** *tm-def* **using** ‹$tx \in \{0..dist\ (c\ A)\ (c\ B)\}$› ‹$D \geq 0$› **by** *auto*
define *xm* **where** $xm =$ *geodesic-segment-param* $\{c\ A{-}{-}c\ B\}$ $(c\ A)$ *tm*
have $xm \in \{c\ A{-}{-}c\ B\}$ **using** ‹$tm \in \{0..dist\ (c\ A)\ (c\ B)\}$›
by (*metis geodesic-segment-param*(*3*) *local.some-geodesic-is-geodesic-segment*(*1*) *xm-def*)
have $dist\ xm\ x = abs((max\ (tx - 2 * D)\ 0) - tx)$
unfolding *xm-def tm-def x-param* **apply** (*rule geodesic-segment-param*[*of* -

- *c B*], *auto*)
using ‹*tx* ∈ {*0..dist* (*c A*) (*c B*)}› ‹*D* ≥ *0*› **by** *auto*
also have ... ≤ *2* ∗ *D* **by** (*simp add*: ‹*0* ≤ *D*› *tx-def*)
finally have *dist xm x* ≤ *2* ∗ *D* **by** *auto*
have ∃ *ym*∈*d*‘{*A..B*}. *infdist xm* (*d*‘{*A..B*}) = *dist xm ym*
apply (*rule infdist-proper-attained*) **using** *3 d*(*1*) *proper-of-compact compact-continuous-image* **by** *auto*
then obtain *ym* **where** *ym*: *ym* ∈ *d*‘{*A..B*} *dist xm ym* = *infdist xm* (*d*‘{*A..B*})
by *metis*
then obtain *um* **where** *um*: *um* ∈ {*A..B*} *ym* = *d um* **by** *auto*
have *dist xm ym* ≤ *D*
unfolding *D-def* **using** *x ym* **by** (*simp add*: ‹*xm* ∈ {*c A*−−*c B*}›)
have *D1*: *dist x z* ≥ *D* **if** *z* ∈ {*xm*−−*ym*} **for** *z*
proof (*cases tx* − *2* ∗ *D* < *0*)
case *True*
then have *tm* = *0* **unfolding** *tm-def* **by** *auto*
then have *xm* = *c A* **unfolding** *xm-def*
by (*meson geodesic-segment-param*(*1*) *local.some-geodesic-is-geodesic-segment*(*1*))
then have *infdist xm* (*d*‘{*A..B*}) = *0*
using ‹*d A* = *c A*› ‹*A* ∈ {*A..B*}› **by** (*metis image-eqI infdist-zero*)
then have *ym* = *xm* **using** *ym*(*2*) **by** *auto*
then have *z* = *xm* **using** ‹*z* ∈ {*xm*−−*ym*}› *geodesic-segment-between-x-x*(*3*)
by (*metis empty-iff insert-iff some-geodesic-is-geodesic-segment*(*1*))
then have *z* ∈ *d*‘{*A..B*} **using** ‹*ym* = *xm*› *ym*(*1*) **by** *blast*
then show *dist x z* ≥ *D* **unfolding** *D-def* **by** (*simp add*: *infdist-le*)
next
case *False*
then have ∗: *tm* = *tx* − *2* ∗ *D* **unfolding** *tm-def* **by** *auto*
have *dist xm x* = *abs*((*tx* − *2* ∗ *D*) − *tx*)
unfolding *xm-def x-param* ∗ **apply** (*rule geodesic-segment-param*[*of - - c B*], *auto*)
using *False* ‹*tx* ∈ {*0..dist* (*c A*) (*c B*)}› ‹*D* ≥ *0*› **by** *auto*
then have *2* ∗ *D* = *dist xm x* **using** ‹*D* ≥ *0*› **by** *auto*
also have ... ≤ *dist xm z* + *dist x z* **using** *metric-space-class.dist-triangle2* **by** *auto*
also have ... ≤ *dist xm ym* + *dist x z*
using ‹*z* ∈ {*xm*−−*ym*}› **by** (*auto*, *meson geodesic-segment-dist-le some-geodesic-is-geodesic-segment*(*1*) *some-geodesic-endpoints*(*1*))
also have ... ≤ *D* + *dist x z*
using ‹*dist xm ym* ≤ *D*› **by** *simp*
finally show *dist x z* ≥ *D* **by** *auto*
qed

define *tM* **where** *tM* = *min* (*tx* + *2* ∗ *D*) (*dist* (*c A*) (*c B*))
have *tM* ∈ {*0..dist* (*c A*) (*c B*)} **unfolding** *tM-def* **using** ‹*tx* ∈ {*0..dist* (*c A*) (*c B*)}› ‹*D* ≥ *0*› **by** *auto*
have *tm* ≤ *tM*
unfolding *tM-def* **using** ‹*D* ≥ *0*› ‹*tm* ∈ {*0..dist* (*c A*) (*c B*)}› ‹*tx* ≡ *dist*

(*c A*) *x*› *tm-def* **by** *auto*
define *xM* **where** *xM* = *geodesic-segment-param* {*c A*−−*c B*} (*c A*) *tM*
have *xM* ∈ {*c A*−−*c B*} **using** ‹*tM* ∈ {*0..dist* (*c A*) (*c B*)}›
by (*metis geodesic-segment-param*(*3*) *local.some-geodesic-is-geodesic-segment*(*1*) *xM-def*)
have *dist xM x* = *abs*((*min* (*tx* + *2* ∗ *D*) (*dist* (*c A*) (*c B*))) − *tx*)
unfolding *xM-def tM-def x-param* **apply** (*rule geodesic-segment-param*[*of* - - *c B*], *auto*)
using ‹*tx* ∈ {*0..dist* (*c A*) (*c B*)}› ‹*D* ≥ *0*› **by** *auto*
also have ... ≤ *2* ∗ *D* **using** ‹*0* ≤ *D*› ‹*tx* ∈ {*0..dist* (*c A*) (*c B*)}› **by** *auto*
finally have *dist xM x* ≤ *2* ∗ *D* **by** *auto*
have ∃ *yM*∈*d*‘{*A..B*}. *infdist xM* (*d*‘{*A..B*}) = *dist xM yM*
apply (*rule infdist-proper-attained*) **using** *3 d*(*1*) *proper-of-compact compact-continuous-image* **by** *auto*
then obtain *yM* **where** *yM*: *yM* ∈ *d*‘{*A..B*} *dist xM yM* = *infdist xM* (*d*‘{*A..B*})
by *metis*
then obtain *uM* **where** *uM*: *uM* ∈ {*A..B*} *yM* = *d uM* **by** *auto*
have *dist xM yM* ≤ *D*
unfolding *D-def* **using** *x yM* **by** (*simp add*: ‹*xM* ∈ {*c A*−−*c B*}›)
have *D3*: *dist x z* ≥ *D* **if** *z* ∈ {*xM*−−*yM*} **for** *z*
proof (*cases tx* + *2* ∗ *D* > *dist* (*c A*) (*c B*))
case *True*
then have *tM* = *dist* (*c A*) (*c B*) **unfolding** *tM-def* **by** *auto*
then have *xM* = *c B* **unfolding** *xM-def*
by (*meson geodesic-segment-param*(*2*) *local.some-geodesic-is-geodesic-segment*(*1*))
then have *infdist xM* (*d*‘{*A..B*}) = *0*
using ‹*d B* = *c B*› ‹*B* ∈ {*A..B*}› **by** (*metis image-eqI infdist-zero*)
then have *yM* = *xM* **using** *yM*(*2*) **by** *auto*
then have *z* = *xM* **using** ‹*z* ∈ {*xM*−−*yM*}› *geodesic-segment-between-x-x*(*3*)
by (*metis empty-iff insert-iff some-geodesic-is-geodesic-segment*(*1*))
then have *z* ∈ *d*‘{*A..B*} **using** ‹*yM* = *xM*› *yM*(*1*) **by** *blast*
then show *dist x z* ≥ *D* **unfolding** *D-def* **by** (*simp add*: *infdist-le*)
next
case *False*
then have ∗: *tM* = *tx* + *2* ∗ *D* **unfolding** *tM-def* **by** *auto*
have *dist xM x* = *abs*((*tx* + *2* ∗ *D*) − *tx*)
unfolding *xM-def x-param* ∗ **apply** (*rule geodesic-segment-param*[*of* - - *c B*], *auto*)
using *False* ‹*tx* ∈ {*0..dist* (*c A*) (*c B*)}› ‹*D* ≥ *0*› **by** *auto*
then have *2* ∗ *D* = *dist xM x* **using** ‹*D* ≥ *0*› **by** *auto*
also have ... ≤ *dist xM z* + *dist x z* **using** *metric-space-class.dist-triangle2* **by** *auto*
also have ... ≤ *dist xM yM* + *dist x z*
using ‹*z* ∈ {*xM*−−*yM*}› **by** (*auto*, *meson geodesic-segment-dist-le local.some-geodesic-is-geodesic-segment*(*1*) *some-geodesic-endpoints*(*1*))
also have ... ≤ *D* + *dist x z*
using ‹*dist xM yM* ≤ *D*› **by** *simp*
finally show *dist x z* ≥ *D* **by** *auto*

qed

define *excursion*:: *real*⇒′*a* **where** *excursion* = (λ*t*.
if t ∈ {*0..dist xm ym*} *then* (*geodesic-segment-param* {*xm*−−*ym*} *xm t*)
else if t ∈ {*dist xm ym..dist xm ym* + *abs*(*uM* − *um*)} *then d* (*um* + *sgn*(*uM*−*um*) ∗ (*t* − *dist xm ym*))
else geodesic-segment-param {*yM*−−*xM*} *yM* (*t* − *dist xm ym* − *abs* (*uM* −*um*)))
define *L* **where** *L* = *dist xm ym* + *abs*(*uM* − *um*) + *dist yM xM*
have *E1*: *excursion t* = *geodesic-segment-param* {*xm*−−*ym*} *xm t* **if** *t* ∈ {*0..dist xm ym*} **for** *t*
unfolding *excursion-def* **using** *that* **by** *auto*
have *E2*: *excursion t* = *d* (*um* + *sgn*(*uM*−*um*) ∗ (*t* − *dist xm ym*)) **if** *t* ∈ {*dist xm ym..dist xm ym* + *abs*(*uM* − *um*)} **for** *t*
unfolding *excursion-def* **using** *that* **by** (*auto simp add*: ‹*ym* = *d um*›)
have *E3*: *excursion t* = *geodesic-segment-param* {*yM*−−*xM*} *yM* (*t* − *dist xm ym* − *abs* (*uM* −*um*))
if *t* ∈ {*dist xm ym* + |*uM* − *um*|..*dist xm ym* + |*uM* − *um*| + *dist yM xM*} **for** *t*
unfolding *excursion-def* **using** *that* ‹*yM* = *d uM*› ‹*ym* = *d um*› **by** (*auto simp add*: *sgn-mult-abs*)
have *E0*: *excursion 0* = *xm*
unfolding *excursion-def* **by** *auto*
have *EL*: *excursion L* = *xM*
unfolding *excursion-def L-def* **apply** (*auto simp add*: *uM*(*2*) *um*(*2*) *sgn-mult-abs*)
by (*metis* (*mono-tags*, *opaque-lifting*) *add.left-neutral add-increasing2 add-le-same-cancel1 dist-real-def*
Gromov-product-e-x-x Gromov-product-nonneg metric-space-class.dist-le-zero-iff)
have [*simp*]: *L* ≥ *0* **unfolding** *L-def* **by** *auto*
have *L* > *0*
proof (*rule ccontr*)
assume ¬(*L* > *0*)
then have *L* = *0* **using** ‹*L* ≥ *0*› **by** *simp*
then have *xm* = *xM* **using** *E0 EL* **by** *auto*
then have *tM* = *tm* **unfolding** *xm-def xM-def*
using ‹*tM* ∈ {*0..dist* (*c A*) (*c B*)}› ‹*tm* ∈ {*0..dist* (*c A*) (*c B*)}› *local.geodesic-segment-param-in-geodesic-spaces*(*6*) **by** *fastforce*
also have ... < *tx* **unfolding** *tm-def* **using** ‹*tx* > *0*› ‹*D* ≥ *1*› **by** *auto*
also have ... ≤ *tM* **unfolding** *tM-def* **using** ‹*D* ≥ *0*› ‹*tx* ∈ {*0..dist* (*c A*) (*c B*)}› **by** *auto*
finally show *False* **by** *simp*
qed

have (*1*/*lambda*) ∗ *dist um uM* − (*4* ∗ *C*) ≤ *dist* (*d um*) (*d uM*)
by (*rule quasi-isometry-onD*(*2*)[*OF* ‹*lambda* (*4* ∗ *C*)−*quasi-isometry-on* {*A..B*} *d*› ‹*um* ∈ {*A..B*}› ‹*uM* ∈ {*A..B*}›])
also have ... ≤ *dist ym xm* + *dist xm x* + *dist x xM* + *dist xM yM*
unfolding *um*(*2*)[*symmetric*] *uM*(*2*)[*symmetric*] **by** (*rule dist-triangle5*)

also have ... $\leq D + (2*D) + (2*D) + D$
using ‹*dist xm ym* $\leq$ *D*› ‹*dist xm x* $\leq$ *2*D*› ‹*dist xM x* $\leq$ *2*D*› ‹*dist xM yM* $\leq$ *D*›
by (*auto simp add*: *metric-space-class.dist-commute intro*: *add-mono*)
finally have (*1/lambda*) * *dist um uM* $\leq$ *6*D* + *4*C* **by** *auto*
then have *dist um uM* $\leq$ *6*D*lambda* + *4*C*lambda*
using *C* **by** (*auto simp add*: *divide-simps algebra-simps*)
then have *L* $\leq$ *D* + (*6*D*lambda* + *4*C*lambda*) + *D*
unfolding *L-def dist-real-def* **using** ‹*dist xm ym* $\leq$ *D*› ‹*dist xM yM* $\leq$ *D*›
by (*auto simp add*: *metric-space-class.dist-commute intro*: *add-mono*)
also have ... $\leq$ *8* * *D* * *lambda* + *4*C*lambda*
using *C* ‹*D* $\geq$ *0*› **by** (*auto intro*: *mono-intros*)
finally have *L-bound*: *L* $\leq$ *lambda* * (*8* * *D* + *4* * *C*)
by (*auto simp add*: *algebra-simps*)

have *1* * (*1* * *1* + *0*) $\leq$ *lambda* * (*8* * *D* + *4* * *C*)
using *C* ‹*D* $\geq$ *1*› **by** (*intro mono-intros*, *auto*)

consider *um* < *uM* | *um* = *uM* | *um* > *uM* **by** *linarith*
then have ((λ*t*. *um* + *sgn* (*uM* − *um*) * (*t* − *dist xm ym*)) ‘ {*dist xm ym*..*dist xm ym* + |*uM* − *um*|}) $\subseteq$ {*min um uM*..*max um uM*}
by (*cases*, *auto*)
also have ... $\subseteq$ {*A*..*B*} **using** ‹*um* $\in$ {*A*..*B*}› ‹*uM* $\in$ {*A*..*B*}› **by** *auto*
finally have *middle*: ((λ*t*. *um* + *sgn* (*uM* − *um*) * (*t* − *dist xm ym*)) ‘ {*dist xm ym*..*dist xm ym* + |*uM* − *um*|}) $\subseteq$ {*A*..*B*}
by *simp*

have (*2* * *lambda*)−*lipschitz-on* {*0*..*L*} *excursion*
proof (*unfold L-def*, *rule lipschitz-on-closed-Union*[*of* {{*0*..*dist xm ym*}, {*dist xm ym*..*dist xm ym* + *abs*(*uM* − *um*)}, {*dist xm ym* + *abs*(*uM* − *um*)..*dist xm ym* + *abs*(*uM* − *um*) + *dist yM xM*}} - λ *i*. *i*], *auto*)
show *lambda* $\geq$ *0* **using** *C* **by** *auto*

have *: *1*−*lipschitz-on* {*0*..*dist xm ym*} (*geodesic-segment-param* {*xm*−−*ym*} *xm*)
by (*rule isometry-on-lipschitz*, *simp*)
have **: *1*−*lipschitz-on* {*0*..*dist xm ym*} *excursion*
using *lipschitz-on-transform*[*OF* * *E1*] **by** *simp*
show (*2* * *lambda*)−*lipschitz-on* {*0*..*dist xm ym*} *excursion*
apply (*rule lipschitz-on-mono*[*OF* **]) **using** *C* **by** *auto*

have *: (*1**(*1*+*0*))−*lipschitz-on* {*dist xm ym* + |*uM* − *um*|..*dist xm ym* + |*uM* − *um*| + *dist yM xM*}
((*geodesic-segment-param* {*yM*−−*xM*} *yM*) *o* (λ*t*. *t* − (*dist xm ym* + *abs* (*uM* −*um*))))
by (*intro lipschitz-intros*, *rule isometry-on-lipschitz*, *auto*)
have **: (*1**(*1*+*0*))−*lipschitz-on* {*dist xm ym* + |*uM* − *um*|..*dist xm ym* + |*uM* − *um*| + *dist yM xM*} *excursion*
apply (*rule lipschitz-on-transform*[*OF* *]) **using** *E3* **unfolding** *comp-def*

by (*auto simp add*: *algebra-simps*)
show $(2 * lambda)-lipschitz\text{-}on$ $\{dist\ xm\ ym + |uM - um|..dist\ xm\ ym + |uM - um| + dist\ yM\ xM\}$ *excursion*
apply (*rule lipschitz-on-mono*[*OF* $**$]) **using** *C* **by** *auto*

have $**$: $((2 * lambda) * (0 + abs(sgn\ (uM - um)) * (1 + 0)))-lipschitz\text{-}on$ $\{dist\ xm\ ym..dist\ xm\ ym + abs(uM - um)\}$ $(d\ o\ (\lambda t.\ um + sgn(uM-um) * (t - dist\ xm\ ym)))$
apply (*intro lipschitz-intros*, *rule lipschitz-on-subset*[*OF* - *middle*])
using ‹$(2 * lambda)-lipschitz\text{-}on$ $\{A..B\}$ *d*› **by** *simp*
have $***$: $(2 * lambda)-lipschitz\text{-}on$ $\{dist\ xm\ ym..dist\ xm\ ym + abs(uM - um)\}$ $(d\ o\ (\lambda t.\ um + sgn(uM-um) * (t - dist\ xm\ ym)))$
apply (*rule lipschitz-on-mono*[*OF* $**$]) **using** *C* **by** *auto*
show $(2 * lambda)-lipschitz\text{-}on$ $\{dist\ xm\ ym..dist\ xm\ ym + abs(uM - um)\}$ *excursion*
apply (*rule lipschitz-on-transform*[*OF* $***$]) **using** *E2* **by** *auto*
qed

have $*$: $dist\ x\ z \geq D$ **if** z: $z \in excursion`\{0..L\}$ **for** z
proof $-$
obtain *tz* **where** *tz*: $z = excursion\ tz\ tz \in \{0..dist\ xm\ ym + abs(uM - um) + dist\ yM\ xM\}$
using *z L-def* **by** *auto*
consider $tz \in \{0..dist\ xm\ ym\}$ | $tz \in \{dist\ xm\ ym<..dist\ xm\ ym + abs(uM - um)\}$ | $tz \in \{dist\ xm\ ym + abs(uM - um)<..dist\ xm\ ym + abs(uM - um) + dist\ yM\ xM\}$
using *tz* **by** *force*
then show *?thesis*
proof (*cases*)
case *1*
then have $z \in \{xm--ym\}$ **unfolding** *tz*(*1*) *excursion-def* **by** *auto*
then show *?thesis* **using** *D1* **by** *auto*
next
case *3*
then have $z \in \{yM--xM\}$ **unfolding** *tz*(*1*) *excursion-def* **using** *tz*(*2*) **by** *auto*
then show *?thesis* **using** *D3* **by** (*simp add*: *some-geodesic-commute*)
next
case *2*
then have $z \in d`\{A..B\}$ **unfolding** *tz*(*1*) *excursion-def* **using** *middle* **by** *force*
then show *?thesis* **unfolding** *D-def* **by** (*simp add*: *infdist-le*)
qed
qed

Now comes the main point: the excursion is always at distance at least D of x, but this distance is also bounded by the log of its length, i.e., essentially $\log D$. To have an efficient estimate, we use a rescaled version, to get rid of one term on the right hand side.

have $1 * 1 * 1 * (1 + 0/1) \leq 512 * lambda * lambda * (1+C/D)$
apply (*intro mono-intros*) **using** ‹$lambda \geq 1$› ‹$D \geq 1$› ‹$C \geq 0$› **by** *auto*
then have $ln\ (512 * lambda * lambda * (1+C/D)) \geq 0$
apply (*subst ln-ge-zero-iff*) **by** *auto*
define a **where** $a = 64 * lambda/D$
have $a > 0$ **unfolding** *a-def* **using** ‹$D \geq 1$› ‹$lambda \geq 1$› **by** *auto*

have $D \leq infdist\ x\ (excursion`\{0..L\})$
unfolding *infdist-def* **apply** *auto* **apply** (*rule cInf-greatest*) **using** $*$ **by** *auto*
also have ... $\leq (4/ln\ 2) * deltaG(TYPE('a)) * max\ 0\ (ln\ (a * (L-0))) + (2 * lambda) / a$
proof (*rule lipschitz-path-close-to-geodesic'*[*of - - - - geodesic-subsegment* $\{c\ A--c\ B\}$ $(c\ A)$ *tm tM*])
show $(2 * lambda)-$*lipschitz-on* $\{0..L\}$ *excursion* **by** *fact*
have $*$: *geodesic-subsegment* $\{c\ A--c\ B\}$ $(c\ A)$ *tm tM* = *geodesic-segment-param* $\{c\ A--c\ B\}$ $(c\ A)$ ‘ $\{tm..tM\}$
apply (*rule geodesic-subsegment*(1)[*of - - c B*])
using ‹$tm \in \{0..dist\ (c\ A)\ (c\ B)\}$› ‹$tM \in \{0..dist\ (c\ A)\ (c\ B)\}$› ‹$tm \leq tM$› **by** *auto*
show $x \in$ *geodesic-subsegment* $\{c\ A--c\ B\}$ $(c\ A)$ *tm tM*
unfolding $*$ **unfolding** *x-param tm-def tM-def* **using** ‹$tx \in \{0..dist\ (c\ A)\ (c\ B)\}$› ‹$0 \leq D$› **by** *simp*
show *geodesic-segment-between* (*geodesic-subsegment* $\{c\ A--c\ B\}$ $(c\ A)$ *tm tM*) (*excursion 0*) (*excursion L*)
unfolding *E0 EL xm-def xM-def* **apply** (*rule geodesic-subsegment*[*of - - c B*])
using ‹$tm \in \{0..dist\ (c\ A)\ (c\ B)\}$› ‹$tM \in \{0..dist\ (c\ A)\ (c\ B)\}$› ‹$tm \leq tM$› **by** *auto*
qed (*fact*)
also have ... $= (4/ln\ 2) * deltaG(TYPE('a)) * max\ 0\ (ln\ (a *L)) + D/32$
unfolding *a-def* **using** ‹$D \geq 1$› ‹$lambda \geq 1$› **by** (*simp add*: *algebra-simps*)
finally have $(31 * ln\ 2\ /\ 128) * D \leq deltaG(TYPE('a)) * max\ 0\ (ln\ (a * L))$
by (*auto simp add*: *algebra-simps divide-simps*)
also have ... $\leq deltaG(TYPE('a)) * max\ 0\ (ln\ ((64 * lambda/D) * (lambda * (8 * D + 4 * C))))$
unfolding *a-def* **apply** (*intro mono-intros*)
using *L-bound* ‹$L > 0$› ‹$lambda \geq 1$› ‹$D \geq 1$› **by** *auto*
also have ... $\leq deltaG(TYPE('a)) * max\ 0\ (ln\ ((64 * lambda/D) * (lambda * (8 * D + 8 * C))))$
apply (*intro mono-intros*)
using *L-bound* ‹$L > 0$› ‹$lambda \geq 1$› ‹$D \geq 1$› ‹$C \geq 0$› **by** *auto*
also have ... $= deltaG(TYPE('a)) * max\ 0\ (ln\ (512 * lambda * lambda * (1+C/D)))$
using ‹$D \geq 1$› **by** (*auto simp add*: *algebra-simps*)
also have ... $= deltaG(TYPE('a)) * ln\ (512 * lambda * lambda * (1+C/D))$
using ‹$ln\ (512 * lambda * lambda * (1+C/D)) \geq 0$› **by** *auto*
also have ... $\leq deltaG(TYPE('a)) * ln\ (512 * lambda * lambda * (1+C/1))$

apply (*intro mono-intros*) **using** ‹*lambda* ≥ *1*› ‹*C* ≥ *0*› ‹*D* ≥ *1*›
by (*auto simp add*: *divide-simps mult-ge1-mono*(*1*))

We have obtained a bound on D, of the form $D \leq M\delta \ln(M\lambda^2(1+C))$. This is a nice bound, but we tweak it a little bit to obtain something more manageable, without the logarithm.

also have ... = *deltaG*(*TYPE*(*'a*)) ∗ (*ln 512* + *2* ∗ *ln lambda* + *ln* (*1*+*C*))
apply (*subst ln2mult*) **using** ‹*C* ≥ *0*› ‹*lambda* ≥ *1*› **apply** *simp*
using ‹*C* ≥ *0*› ‹*lambda* ≥ *1*› **by** (*simp add*:*ln-mult*)
also have ... = (*deltaG*(*TYPE*(*'a*)) ∗ *1*) ∗ *ln 512* + *2* ∗ (*deltaG*(*TYPE*(*'a*)) ∗ *ln lambda*) + (*deltaG*(*TYPE*(*'a*)) ∗ *ln* (*1*+*C*))
by (*auto simp add*: *algebra-simps*)

For each term, of the form $\delta \ln c$, we bound it by $(\delta^2 + (\ln c)^2)/2$, and then bound $(\ln c)^2$ by $2c-2$. In fact, to get coefficients of the same order of magnitude on δ^2 and λ, we tweak a little bit the inequality for the last two terms, using rather $uv \leq (u^2/2 + 2v^2)/2$. We also bound ln(32) by a good approximation 16/3.

also have ... ≤ (*deltaG*(*TYPE*(*'a*))^*2*/*2* + *1*^*2*/*2*) ∗ (*25*/*4*)
+ *2* ∗ ((*1*/*2*) ∗ *deltaG*(*TYPE*(*'a*))^*2*/*2* + *2* ∗ (*ln lambda*)^*2* / *2*) + ((*1*/*2*) ∗ *deltaG*(*TYPE*(*'a*))^*2*/*2* + *2* ∗ (*ln* (*1*+*C*))^*2* / *2*)
by (*intro mono-intros*, *auto*, *approximation 10*)
also have ... = (*31*/*8*) ∗ *deltaG*(*TYPE*(*'a*))^*2* + *25*/*8* + *2* ∗ (*ln lambda*)^*2* + (*ln* (*1*+*C*))^*2*
by (*auto simp add*: *algebra-simps*)
also have ... ≤ *4* ∗ *deltaG*(*TYPE*(*'a*))^*2* + *4* + *2* ∗ (*2* ∗ *lambda* − *2*) + (*2* ∗ (*1*+*C*) − *2*)
apply (*intro mono-intros*) **using** ‹*C* ≥ *0*› ‹*lambda* ≥ *1*› **by** *auto*
also have ... ≤ *4* ∗ *deltaG*(*TYPE*(*'a*))^*2* + *4* ∗ *lambda* + *2* ∗ *C*
by *auto*
finally have *D* ≤ (*128* / (*31* ∗ *ln 2*)) ∗ (*4* ∗ *deltaG*(*TYPE*(*'a*))^*2* + *4* ∗ *lambda* + *2* ∗ *C*)
by (*auto simp add*: *divide-simps algebra-simps*)
also have ... ≤ *6* ∗ (*4* ∗ *deltaG*(*TYPE*(*'a*))^*2* + *4* ∗ *lambda* + *2* ∗ *C*)
apply (*intro mono-intros*, *approximation 10*) **using** ‹*lambda* ≥ *1*› ‹*C* ≥ *0*› **by** *auto*
also have ... ≤ *24* ∗ *deltaG*(*TYPE*(*'a*))^*2* + *24* ∗ *lambda* + *12* ∗ *C*
using ‹*lambda* ≥ *1*› ‹*C* ≥ *0*› **by** *auto*
finally show *?thesis* **by** *simp*
qed
define *D0* **where** *D0* = *24* ∗ *lambda* + *12* ∗ *C* + *24* ∗ *deltaG*(*TYPE*(*'a*))^*2*
have *first-step*: *infdist y* (*d*‘{*A*..*B*}) ≤ *D0* **if** *y* ∈ {*c A*−−*c B*} **for** *y*
using *x*(*2*)[*OF that*] *D-bound* **unfolding** *D0-def D-def* **by** *auto*
have *1* ∗ *1* + *4* ∗ *0* + *24* ∗ *0* ≤ *D0*
unfolding *D0-def* **apply** (*intro mono-intros*) **using** *C delta-nonneg* **by** *auto*
then have *D0* > *0* **by** *simp*

This is the end of the first step, i.e., showing that $[c(A), c(B)]$ is included in the neighborhood of size $D0$ of the quasi-geodesic.

Now, we start the second step: we show that the quasi-geodesic is included in the neighborhood of size $D1$ of the geodesic, where $D1 \geq D0$ is the constant defined below. The argument goes as follows. Assume that a point y on the quasi-geodesic is at distance $> D0$ of the geodesic. Consider the last point y_m before y which is at distance $D0$ of the geodesic, and the first point y_M after y likewise. On (y_m, y_M), one is always at distance $> D0$ of the geodesic. However, by the first step, the geodesic is covered by the balls of radius $D0$ centered at points on the quasi-geodesic – and only the points before y_m or after y_M can be used. Let K_m be the points on the geodesics that are at distance $\leq D0$ of a point on the quasi-geodesic before y_m, and likewise define K_M. These are two closed subsets of the geodesic. By connectedness, they have to intersect. This implies that some points before y_m and after y_M are at distance at most $2D0$. Since we are dealing with a quasi-geodesic, this gives a bound on the distance between y_m and y_M, and therefore a bound between y and the geodesic, as desired.

define D1 **where** D1 = lambda * lambda * (72 * lambda + 44 * C + 72 * deltaG(TYPE('a))^2)
have 1 * 1 * (24 * lambda + 12 * C + 24 * deltaG(TYPE('a))^2)
≤ lambda * lambda * (72 * lambda + 44 * C + 72 * deltaG(TYPE('a))^2)
apply (intro mono-intros) **using** C **by** auto
then have D0 ≤ D1 **unfolding** D0-def D1-def **by** auto
have second-step: infdist y {c A−−c B} ≤ D1 **if** y ∈ d‘{A..B} **for** y
proof (cases infdist y {c A−−c B} ≤ D0)
case True
then show ?thesis **using** ‹D0 ≤ D1› **by** auto
next
case False
obtain ty **where** ty ∈ {A..B} y = d ty **using** ‹y ∈ d‘{A..B}› **by** auto

define tm **where** tm = Sup ((λt. infdist (d t) {c A−−c B})−‘{..D0} ∩ {A..ty})
have tm: tm ∈ (λt. infdist (d t) {c A−−c B})−‘{..D0} ∩ {A..ty}
unfolding tm-def **proof** (rule closed-contains-Sup)
show closed ((λt. infdist (d t) {c A−−c B})−‘{..D0} ∩ {A..ty})
apply (rule closed-vimage-Int, auto, intro continuous-intros)
apply (rule continuous-on-subset[OF d(1)]) **using** ‹ty ∈ {A..B}› **by** auto
have A ∈ (λt. infdist (d t) {c A−−c B})−‘{..D0} ∩ {A..ty}
using ‹D0 > 0› ‹ty ∈ {A..B}› **by** (auto simp add: ‹d A = c A›)
then show (λt. infdist (d t) {c A−−c B})−‘{..D0} ∩ {A..ty} ≠ {} **by** auto
show bdd-above ((λt. infdist (d t) {c A−−c B}) −‘ {..D0} ∩ {A..ty}) **by** auto
qed
have ∗: infdist (d t) {c A−−c B} > D0 **if** t ∈ {tm<..ty} **for** t
proof (rule ccontr)
assume ¬(infdist (d t) {c A−−c B} > D0)
then have ∗: t ∈ (λt. infdist (d t) {c A−−c B})−‘{..D0} ∩ {A..ty}
using that tm **by** auto

have $t \leq tm$ **unfolding** *tm-def* **apply** (*rule cSup-upper*) **using** $*$ **by** *auto*
then show *False* **using** *that* **by** *auto*
qed

define *tM* **where** $tM = Inf\ ((\lambda t.\ infdist\ (d\ t)\ \{c\ A--c\ B\})-`\{..D0\} \cap \{ty..B\})$
have *tM*: $tM \in (\lambda t.\ infdist\ (d\ t)\ \{c\ A--c\ B\})-`\{..D0\} \cap \{ty..B\}$
unfolding *tM-def* **proof** (*rule closed-contains-Inf*)
show *closed* $((\lambda t.\ infdist\ (d\ t)\ \{c\ A--c\ B\})-`\{..D0\} \cap \{ty..B\})$
apply (*rule closed-vimage-Int*, *auto*, *intro continuous-intros*)
apply (*rule continuous-on-subset*[*OF d*(*1*)]) **using** ‹$ty \in \{A..B\}$› **by** *auto*
have $B \in (\lambda t.\ infdist\ (d\ t)\ \{c\ A--c\ B\})-`\{..D0\} \cap \{ty..B\}$
using ‹$D0 > 0$› ‹$ty \in \{A..B\}$› **by** (*auto simp add*: ‹$d\ B = c\ B$›)
then show $(\lambda t.\ infdist\ (d\ t)\ \{c\ A--c\ B\})-`\{..D0\} \cap \{ty..B\} \neq \{\}$ **by** *auto*
show *bdd-below* $((\lambda t.\ infdist\ (d\ t)\ \{c\ A--c\ B\})\ -`\ \{..D0\} \cap \{ty..B\})$ **by** *auto*
qed
have $infdist\ (d\ t)\ \{c\ A--c\ B\} > D0$ **if** $t \in \{ty..<tM\}$ **for** t
proof (*rule ccontr*)
assume $\neg(infdist\ (d\ t)\ \{c\ A--c\ B\} > D0)$
then have $*$: $t \in (\lambda t.\ infdist\ (d\ t)\ \{c\ A--c\ B\})-`\{..D0\} \cap \{ty..B\}$
using *that tM* **by** *auto*
have $t \geq tM$ **unfolding** *tM-def* **apply** (*rule cInf-lower*) **using** $*$ **by** *auto*
then show *False* **using** *that* **by** *auto*
qed
then have *lower-tm-tM*: $infdist\ (d\ t)\ \{c\ A--c\ B\} > D0$ **if** $t \in \{tm<..<tM\}$ **for** t
using $*$ *that* **by** (*cases* $t \geq ty$, *auto*)

define *Km* **where** $Km = (\bigcup z \in d`\{A..tm\}.\ cball\ z\ D0)$
define *KM* **where** $KM = (\bigcup z \in d`\{tM..B\}.\ cball\ z\ D0)$
have $\{c\ A--c\ B\} \subseteq Km \cup KM$
proof
fix x **assume** $x \in \{c\ A--c\ B\}$
have $\exists z \in d`\{A..B\}.\ infdist\ x\ (d`\{A..B\}) = dist\ x\ z$
apply (*rule infdist-proper-attained*[*OF proper-of-compact*], *rule compact-continuous-image*[*OF* ‹*continuous-on* $\{A..B\}$ *d*›])
using *that* **by** *auto*
then obtain *tx* **where** $tx \in \{A..B\}$ $infdist\ x\ (d`\{A..B\}) = dist\ x\ (d\ tx)$ **by** *blast*
then have $dist\ x\ (d\ tx) \leq D0$
using *first-step*[*OF* ‹$x \in \{c\ A--c\ B\}$›] **by** *auto*
then have $x \in cball\ (d\ tx)\ D0$ **by** (*auto simp add*: *metric-space-class.dist-commute*)
consider $tx \in \{A..tm\}$ | $tx \in \{tm<..<tM\}$ | $tx \in \{tM..B\}$
using ‹$tx \in \{A..B\}$› **by** *fastforce*
then show $x \in Km \cup KM$
proof (*cases*)
case *1*
then have $x \in Km$ **unfolding** *Km-def* **using** ‹$x \in cball\ (d\ tx)\ D0$› **by**

auto
then show *?thesis* **by** *simp*
next
case *3*
then have $x \in KM$ **unfolding** *KM-def* **using** ‹$x \in cball\ (d\ tx)\ D0$› **by** *auto*
then show *?thesis* **by** *simp*
next
case *2*
have $infdist\ (d\ tx)\ \{c\ A{-}{-}c\ B\} \leq dist\ (d\ tx)\ x$ **using** ‹$x \in \{c\ A{-}{-}c\ B\}$› **by** (*rule infdist-le*)
also have ... $\leq D0$ **using** ‹$x \in cball\ (d\ tx)\ D0$› **by** *auto*
finally have *False* **using** *lower-tm-tM*[*OF 2*] **by** *simp*
then show *?thesis* **by** *simp*
qed
qed
then have ∗: $\{c\ A{-}{-}c\ B\} = (Km \cap \{c\ A{-}{-}c\ B\}) \cup (KM \cap \{c\ A{-}{-}c\ B\})$ **by** *auto*
have $(Km \cap \{c\ A{-}{-}c\ B\}) \cap (KM \cap \{c\ A{-}{-}c\ B\}) \neq \{\}$
proof (*rule connected-as-closed-union*[*OF - ∗*])
have *closed Km*
unfolding *Km-def* **apply** (*rule compact-has-closed-thickening*)
apply (*rule compact-continuous-image*)
apply (*rule continuous-on-subset*[*OF* ‹*continuous-on* $\{A..B\}$ *d*›])
using *tm* ‹$ty \in \{A..B\}$› **by** *auto*
then show *closed* $(Km \cap \{c\ A{-}{-}c\ B\})$ **by** (*rule topological-space-class.closed-Int*, *auto*)

have *closed KM*
unfolding *KM-def* **apply** (*rule compact-has-closed-thickening*)
apply (*rule compact-continuous-image*)
apply (*rule continuous-on-subset*[*OF* ‹*continuous-on* $\{A..B\}$ *d*›])
using *tM* ‹$ty \in \{A..B\}$› **by** *auto*
then show *closed* $(KM \cap \{c\ A{-}{-}c\ B\})$ **by** (*rule topological-space-class.closed-Int*, *auto*)

show *connected* $\{c\ A{-}{-}c\ B\}$ **by** *simp*
have $c\ A \in Km \cap \{c\ A{-}{-}c\ B\}$ **apply** *auto*
unfolding *Km-def* **using** *tm* ‹$d\ A = c\ A$› ‹$D0 > 0$› **by** (*auto*) (*rule bexI*[*of - A*], *auto*)
then show $Km \cap \{c\ A{-}{-}c\ B\} \neq \{\}$ **by** *auto*
have $c\ B \in KM \cap \{c\ A{-}{-}c\ B\}$ **apply** *auto*
unfolding *KM-def* **using** *tM* ‹$d\ B = c\ B$› ‹$D0 > 0$› **by** (*auto*) (*rule bexI*[*of - B*], *auto*)
then show $KM \cap \{c\ A{-}{-}c\ B\} \neq \{\}$ **by** *auto*
qed
then obtain *w* **where** $w \in \{c\ A{-}{-}c\ B\}$ $w \in Km$ $w \in KM$ **by** *auto*
then obtain *twm twM* **where** *tw*: $twm \in \{A..tm\}$ $w \in cball\ (d\ twm)\ D0$ $twM \in \{tM..B\}$ $w \in cball\ (d\ twM)\ D0$

unfolding *Km-def KM-def* **by** *auto*
have $(1/lambda) * dist\ twm\ twM - (4*C) \leq dist\ (d\ twm)\ (d\ twM)$
apply (*rule quasi-isometry-onD*(*2*)[*OF d*(*5*)]) **using** *tw tm tM* **by** *auto*
also have ... $\leq dist\ (d\ twm)\ w + dist\ w\ (d\ twM)$
by (*rule metric-space-class.dist-triangle*)
also have ... $\leq 2 * D0$ **using** *tw* **by** (*auto simp add*: *metric-space-class.dist-commute*)
finally have $dist\ twm\ twM \leq lambda * (4*C + 2*D0)$
using *C* **by** (*auto simp add*: *divide-simps algebra-simps*)
then have $*$: $dist\ twm\ ty \leq lambda * (4*C + 2*D0)$
using *tw tm tM dist-real-def* **by** *auto*

have $dist\ (d\ ty)\ w \leq dist\ (d\ ty)\ (d\ twm) + dist\ (d\ twm)\ w$
by (*rule metric-space-class.dist-triangle*)
also have ... $\leq (lambda * dist\ ty\ twm + (4*C)) + D0$
apply (*intro add-mono*, *rule quasi-isometry-onD*(*1*)[*OF d*(*5*)]) **using** *tw tm tM* **by** *auto*
also have ... $\leq (lambda * (lambda * (4*C + 2*D0))) + (4*C) + D0$
apply (*intro mono-intros*) **using** C $*$ **by** (*auto simp add*: *metric-space-class.dist-commute*)
also have ... $= lambda * lambda * (4*C + 2*D0) + 1 * 1 * (4 * C) + 1 * 1 * D0$
by *simp*
also have ... $\leq lambda * lambda * (4*C + 2*D0) + lambda * lambda * (4 * C) + lambda * lambda * D0$
apply (*intro mono-intros*) **using** C $*$ ‹$D0 > 0$› **by** *auto*
also have ... $= lambda * lambda * (8 * C + 3 * D0)$
by (*auto simp add*: *algebra-simps*)
also have ... $= lambda * lambda * (44 * C + 72 * lambda + 72 * deltaG(TYPE('a))\hat{\ }2)$
unfolding *D0-def* **by** *auto*
finally have $dist\ y\ w \leq D1$ **unfolding** *D1-def* ‹$y = d\ ty$› **by** (*auto simp add*: *algebra-simps*)
then show $infdist\ y\ \{c\ A{-}{-}c\ B\} \leq D1$ **using** *infdist-le*[*OF* ‹$w \in \{c\ A{-}{-}c\ B\}$›, *of y*] **by** *auto*
qed

This concludes the second step.

Putting the two steps together, we deduce that the Hausdorff distance between the geodesic and the quasi-geodesic is bounded by $D1$. A bound between the geodesic and the original (untamed) quasi-geodesic follows.

have *a*: *hausdorff-distance* $(d`\{A..B\})\ \{c\ A{-}{-}c\ B\} \leq D1$
proof (*rule hausdorff-distanceI*)
show $D1 \geq 0$ **unfolding** *D1-def* **using** *C delta-nonneg* **by** *auto*
fix x **assume** $x \in d\ `\ \{A..B\}$
then show $infdist\ x\ \{c\ A{-}{-}c\ B\} \leq D1$ **using** *second-step* **by** *auto*
next
fix x **assume** $x \in \{c\ A{-}{-}c\ B\}$
then show $infdist\ x\ (d`\{A..B\}) \leq D1$ **using** *first-step* ‹$D0 \leq D1$› **by** *force*
qed

have *hausdorff-distance* $(c`\{A..B\})$ $\{c\ A{-}{-}c\ B\} \leq$
hausdorff-distance $(c`\{A..B\})$ $(d`\{A..B\})$ + *hausdorff-distance* $(d`\{A..B\})$ $\{c\ A{-}{-}c\ B\}$
apply (*rule hausdorff-distance-triangle*)
using ‹$A \in \{A..B\}$› **apply** *blast*
by (*rule quasi-isometry-on-bounded*[*OF d*(*5*)], *auto*)
also have ... $\leq D1 + 2{*}C$ **using** *a d* **by** *auto*
also have ... = $lambda * lambda * (72 * lambda + 44 * C + 72 * deltaG(TYPE('a))\hat{~}2) + 1 * 1 * (2 * C)$
unfolding *D1-def* **by** *auto*
also have ... $\leq lambda * lambda * (72 * lambda + 44 * C + 72 * deltaG(TYPE('a))\hat{~}2) + lambda * lambda * (28 * C)$
apply (*intro mono-intros*) **using** *C delta-nonneg* **by** *auto*
also have ... = $72 * lambda\hat{~}2 * (lambda + C + deltaG(TYPE('a))\hat{~}2)$
by (*auto simp add*: *algebra-simps power2-eq-square*)
finally show *?thesis* **by** (*auto simp add*: *algebra-simps*)
qed
qed

end

12 The Bonk Schramm extension

theory *Bonk-Schramm-Extension*
imports *Morse-Gromov-Theorem*
begin

We want to show that any metric space is isometrically embedded in a metric space which is geodesic (i.e., there is an embedded geodesic between any two points) and complete. There are many such constructions, but a very interesting one has been given by Bonk and Schramm in [BS00], together with an additional property of the completion: if the space is delta-hyperbolic (in the sense of Gromov), then its completion also is, with the same constant delta. It follows in particular that a 0-hyperbolic space embeds in a 0-hyperbolic geodesic space, i.e., a metric tree (there is an easier direct construction in this case).

Another embedding of a metric space in a geodesic one is constructed by Mineyev [Min05], it is more canonical in a sense (isometries of the original space extend to the new space), but it is not clear if it preserves hyperbolicity.

The argument of Bonk and Schramm goes as follows: - first, if one wants to add the middle of a pair of points a and b in a space E, there is a nice formula for the distance on a new space $E \cup \{*\}$ (where $*$ will by construction be a middle of a and b). - by transfinite induction on all the pair of points in the space, one adds all the missing middles - then one completes the space - then one adds all the middles - then one goes on like that, transfinitely many times - at some point, the process stops for cardinality reasons

The resulting space is complete and has middles for all pairs of points. It is then standard that it is geodesic (this is proved in `Geodesic_Spaces.thy`).

Implementing this construction in Isabelle is interesting and nontrivial, as transfinite induction is not that easy, especially when intermingled with metric completion (i.e., taking the quotient space of all Cauchy sequences). In particular, taking sequences of metric completions would mean changing types at each step, along a transfinite number of steps. It does not seem possible to do it naively in this way.

We avoid taking quotients in the middle of the argument, as this is too messy. Instead, we define a pseudo-distance (i.e., a function satisyfing the triangular inequality, but such that $d(x, y)$ can vanish even if x and y are different) on an increasing set, which should contain middles and limits of Cauchy sequences (identified with their defining Cauchy sequence). Thus, we consider a datatype containing points in the original space and closed under two operations: taking a pair of points in the datatype (we think of the resulting pair as the middle of the pair) and taking a sequence with values in the datatype (we think of the resulting sequence as the limit of the sequence if it is Cauchy, for a distance yet to be defined, and as something we discard if the sequence is not Cauchy).

Defining such an object is apparently not trivial. However, it is well defined, for cardinality reasons, as this process will end after the continuum cardinality iterations (as a sequence taking value in the continuum cardinality is in fact contained in a strictly smaller ordinal, which means that all sequences in the construction will appear at a step strictly before the continuum cardinality). The datatype construction in Isabelle/HOL contains these cardinality considerations as an automatic process, and is thus able to construct the datatype directly, without the need for any additional proof!

Then, we define a wellorder on the datatype, such that every middle and every sequence appear after each of its ancestors. This construction of a wellorder should work for any datatype, but we provide a naive proof in our use case.

Then, we define, inductively on z, a pseudodistance on the pair of points in $\{x : x \leq z\}$. In the induction, one should add one point at a time. If it is a middle, one uses the Bonk-Schramm recipe. If it is a sequence, then either the sequence is Cauchy and one uses the limit of the distances to the points in the sequence, or it is not Cauchy and one discards the new point by setting $d(a, a) = 1$. (This means that, in the Bonk-Schramm recipe, we only use the points with $d(x, x) = 0$, and show the triangular inequality there).

In the end, we obtain a space with a pseudodistance. The desired space is obtained by quotienting out the space $\{x : d(x, x) = 0\}$ by the equivalence relation given by $d(x, y) = 0$. The triangular inequality for the pseudodistance shows that it descends to a genuine distance on the quotient. This is the desired geodesic complete extension of the original space.

12.1 Unfolded Bonk Schramm extension

The unfolded Bonk Schramm extension, as explained at the beginning of this file, is a type made of the initial type, adding all possible middles and all possible limits of Cauchy sequences, without any quotienting process

datatype *'a Bonk-Schramm-extension-unfolded* =
basepoint 'a
| *middle 'a Bonk-Schramm-extension-unfolded 'a Bonk-Schramm-extension-unfolded*
| *would-be-Cauchy nat* ⇒ *'a Bonk-Schramm-extension-unfolded*

context *metric-space*
begin

The construction of the distance will be done by transfinite induction, with respect to a well-order for which the basepoints form an initial segment, and for which middles of would-be Cauchy sequences are larger than the elements they are made of. We will first prove the existence of such a well-order.

The idea is first to construct a function `map_aux` to another type, with a well-order `wo_aux`, such that the image of `middle a b` is larger than the images of `a` and `b` (take for instance the successor of the maximum of the two), and likewise for a Cauchy sequence. A definition by induction works if the target cardinal is large enough.

Then, pullback the well-order `wo_aux` by the map `map_aux`: this gives a relation that satisfies all the required properties, except that two different elements can be equal for the order. Extending it essentially arbitrarily to distinguish between all elements (this is done in Lemma `Well_order_pullback`) gives the desired well-order

definition *Bonk-Schramm-extension-unfolded-wo* **where**
Bonk-Schramm-extension-unfolded-wo = (*SOME* (r::*'a Bonk-Schramm-extension-unfolded rel*).
well-order-on UNIV r
∧ (∀ x ∈ *range basepoint*. ∀ y ∈ − *range basepoint*. (x, y) ∈ r)
∧ (∀ a b. (a, *middle a b*) ∈ r)
∧ (∀ a b. (b, *middle a b*) ∈ r)
∧ (∀ u n. (u n, *would-be-Cauchy u*) ∈ r))

We prove the existence of the well order

definition *wo-aux* **where**
wo-aux = (*SOME* (r:: (*nat* + *'a Bonk-Schramm-extension-unfolded set*) *rel*).
Card-order r ∧ ¬*finite*(*Field r*) ∧ *regularCard r* ∧ |*UNIV*::*'a Bonk-Schramm-extension-unfolded set*| <*o* r)

lemma *wo-aux-exists*:
Card-order wo-aux $\wedge$ $\neg$*finite* (*Field wo-aux*) $\wedge$ *regularCard wo-aux* $\wedge$ |*UNIV*::*'a Bonk-Schramm-extension-unfolded set*| <*o wo-aux*
proof −
have $*$: $\forall$ *r* $\in$ {|*UNIV*::*'a Bonk-Schramm-extension-unfolded set*|}. *Card-order r* **by** *auto*
have $**$: $\exists$(*r*::(*nat* + *'a Bonk-Schramm-extension-unfolded set*) *rel*).
Card-order r $\wedge$ $\neg$*finite*(*Field r*) $\wedge$ *regularCard r* $\wedge$ (|*UNIV*::*'a Bonk-Schramm-extension-unfolded set*| <*o r*)
by (*metis card-of-card-order-on Field-card-of singletonI infinite-regularCard-exists*[*OF* $*$])
then show *?thesis* **unfolding** *wo-aux-def* **using** *someI-ex*[*OF* $**$] **by** *auto*
qed

interpretation *wo-aux*: *wo-rel wo-aux*
using *wo-aux-exists Card-order-wo-rel* **by** *auto*

primrec *map-aux*::*'a Bonk-Schramm-extension-unfolded* $\Rightarrow$ *nat* + *'a Bonk-Schramm-extension-unfolded set* **where**
map-aux (*basepoint x*) = *wo-aux.zero*
| *map-aux* (*middle a b*) = *wo-aux.suc* ({*map-aux a*} $\cup$ {*map-aux b*})
| *map-aux* (*would-be-Cauchy u*) = *wo-aux.suc* ((*map-aux o u*)'*UNIV*)

lemma *map-aux-AboveS-not-empty*:
assumes *map-aux'S* $\subseteq$ *Field wo-aux*
shows *wo-aux.AboveS* (*map-aux'S*) $\neq$ {}
apply (*rule AboveS-not-empty-in-regularCard'*[*of S*])
using *wo-aux-exists assms* **apply** *auto*
using *card-of-UNIV ordLeq-ordLess-trans* **by** *blast*

lemma *map-aux-in-Field*:
map-aux x $\in$ *Field wo-aux*
proof (*induction*)
case (*basepoint x*)
have *wo-aux.zero* $\in$ *Field wo-aux*
using *Card-order-infinite-not-under wo-aux-exists under-empty wo-aux.zero-in-Field* **by** *fastforce*
then show *?case* **by** *auto*
next
case *mid*: (*middle a b*)
have ({*map-aux a*} $\cup$ {*map-aux b*}) $\subseteq$ *Field wo-aux* **using** *mid.IH* **by** *auto*
then have *wo-aux.AboveS* ({*map-aux a*} $\cup$ {*map-aux b*}) $\neq$ {}
using *map-aux-AboveS-not-empty*[*of* {*a*} $\cup$ {*b*}] **by** *auto*
then show *?case*
by (*simp add*: *AboveS-Field wo-aux.suc-def*)
next
case *cauchy*: (*would-be-Cauchy u*)
have (*map-aux o u*)'*UNIV* $\subseteq$ *Field wo-aux* **using** *cauchy.IH* **by** *auto*
then have *wo-aux.AboveS* ((*map-aux o u*)'*UNIV*) $\neq$ {}

using *map-aux-AboveS-not-empty*[*of u*‘(*UNIV*)] **by** (*simp add*: *image-image*)
then show *?case*
by (*simp add*: *AboveS-Field wo-aux.suc-def*)
qed

lemma *middle-rel-a*:
(*map-aux a*, *map-aux* (*middle a b*)) $\in$ *wo-aux* $-$ *Id*
proof $-$
have $*$: ({*map-aux a*} $\cup$ {*map-aux b*}) $\subseteq$ *Field wo-aux* **using** *map-aux-in-Field*
by *auto*
then have *wo-aux.AboveS* ({*map-aux a*} $\cup$ {*map-aux b*}) $\neq$ {}
using *map-aux-AboveS-not-empty*[*of* {*a*} $\cup$ {*b*}] **by** *auto*
then show *?thesis*
using $*$ **by** (*simp add*: *wo-aux.suc-greater Id-def*)
qed

lemma *middle-rel-b*:
(*map-aux b*, *map-aux* (*middle a b*)) $\in$ *wo-aux* $-$ *Id*
proof $-$
have $*$: ({*map-aux a*} $\cup$ {*map-aux b*}) $\subseteq$ *Field wo-aux* **using** *map-aux-in-Field*
by *auto*
then have *wo-aux.AboveS* ({*map-aux a*} $\cup$ {*map-aux b*}) $\neq$ {}
using *map-aux-AboveS-not-empty*[*of* {*a*} $\cup$ {*b*}] **by** *auto*
then show *?thesis*
using $*$ **by** (*simp add*: *wo-aux.suc-greater Id-def*)
qed

lemma *cauchy-rel*:
(*map-aux* (*u n*), *map-aux* (*would-be-Cauchy u*)) $\in$ *wo-aux* $-$ *Id*
proof $-$
have $*$: (*map-aux o u*)‘*UNIV* $\subseteq$ *Field wo-aux* **using** *map-aux-in-Field* **by** *auto*
then have *wo-aux.AboveS* ((*map-aux o u*)‘*UNIV*) $\neq$ {}
using *map-aux-AboveS-not-empty*[*of u*‘(*UNIV*)] **by** (*simp add*: *image-image*)
then show *?thesis*
using $*$ **by** (*simp add*: *wo-aux.suc-greater Id-def*)
qed

From the above properties of `wo_aux`, it follows using `Well_order_pullback` that an order satisfying all the properties we want of `Bonk_Schramm_extension_unfolded_wo` exists. Hence, we get the following lemma.

lemma *Bonk-Schramm-extension-unfolded-wo-props*:
well-order-on UNIV Bonk-Schramm-extension-unfolded-wo
$\forall$ *x* $\in$ *range basepoint*. $\forall$ *y* $\in -$ *range basepoint*. (*x*, *y*) $\in$ *Bonk-Schramm-extension-unfolded-wo*
$\forall$ *a b*. (*a*, *middle a b*) $\in$ *Bonk-Schramm-extension-unfolded-wo*
$\forall$ *a b*. (*b*, *middle a b*) $\in$ *Bonk-Schramm-extension-unfolded-wo*
$\forall$ *u n*. (*u n*, *would-be-Cauchy u*) $\in$ *Bonk-Schramm-extension-unfolded-wo*
proof $-$
obtain *r*::′*a Bonk-Schramm-extension-unfolded rel* **where** *r*:
Well-order r

Field r = UNIV
$\bigwedge$*x y.* (*map-aux x*, *map-aux y*) ∈ *wo-aux* − *Id* ⟹ (*x*, *y*) ∈ *r*
using *Well-order-pullback*[*of wo-aux map-aux*] **by** (*metis wo-aux.WELL*)

have (*x*, *y*) ∈ *r* **if** *x* ∈ *range basepoint y* ∈ − *range basepoint* **for** *x y*
apply (*rule r*(*3*)) **using** *that*
apply (*cases y*)
apply (*auto cong del*: *image-cong-simp*)
apply (*metis insert-is-Un map-aux.simps*(*2*) *map-aux-in-Field wo-aux.zero-smallest*)
apply (*metis Diff-iff insert-is-Un wo-aux.leq-zero-imp map-aux.simps*(*2*) *middle-rel-a pair-in-Id-conv*)
apply (*metis map-aux.simps*(*3*) *map-aux-in-Field wo-aux.zero-smallest*)
apply (*metis Diff-iff cauchy-rel wo-aux.leq-zero-imp map-aux.simps*(*3*) *pair-in-Id-conv*)
done
moreover have (*a*, *middle a b*) ∈ *r* **for** *a b*
apply (*rule r*(*3*)) **using** *middle-rel-a* **by** *auto*
moreover have (*b*, *middle a b*) ∈ *r* **for** *a b*
apply (*rule r*(*3*)) **using** *middle-rel-b* **by** *auto*
moreover have (*u n*, *would-be-Cauchy u*) ∈ *r* **for** *u n*
apply (*rule r*(*3*)) **using** *cauchy-rel* **by** *auto*
moreover have *well-order-on UNIV r*
using *r*(*1*) *r*(*2*) **by** *auto*
ultimately have ∗: ∃ (*r*::′*a Bonk-Schramm-extension-unfolded rel*).
well-order-on UNIV r
∧ (∀ *x* ∈ *range basepoint*. ∀ *y* ∈ − *range basepoint*. (*x*, *y*) ∈ *r*)
∧ (∀ *a b*. (*a*, *middle a b*) ∈ *r*)
∧ (∀ *a b*. (*b*, *middle a b*) ∈ *r*)
∧ (∀ *u n*. (*u n*, *would-be-Cauchy u*) ∈ *r*)
by *blast*

show
well-order-on UNIV Bonk-Schramm-extension-unfolded-wo
∀ *x* ∈ *range basepoint*. ∀ *y* ∈ − *range basepoint*. (*x*, *y*) ∈ *Bonk-Schramm-extension-unfolded-wo*
∀ *a b*. (*a*, *middle a b*) ∈ *Bonk-Schramm-extension-unfolded-wo*
∀ *a b*. (*b*, *middle a b*) ∈ *Bonk-Schramm-extension-unfolded-wo*
∀ *u n*. (*u n*, *would-be-Cauchy u*) ∈ *Bonk-Schramm-extension-unfolded-wo*
unfolding *Bonk-Schramm-extension-unfolded-wo-def* **using** *someI-ex*[*OF* ∗] **by** *auto*
qed

interpretation *wo*: *wo-rel Bonk-Schramm-extension-unfolded-wo*
using *well-order-on-Well-order wo-rel-def wfrec-def Bonk-Schramm-extension-unfolded-wo-props*(*1*)
by *blast*

We reformulate in the interpretation `wo` the main properties of `Bonk_Schramm_extension_unfolded_w` that we established in Lemma `Bonk_Schramm_extension_unfolded_wo_props`

lemma *Bonk-Schramm-extension-unfolded-wo-props*′:
a ∈ *wo.underS* (*middle a b*)
b ∈ *wo.underS* (*middle a b*)

u n ∈ *wo.underS* (*would-be-Cauchy u*)
proof −
  have (*a*, *middle a b*) ∈ *Bonk-Schramm-extension-unfolded-wo*
    using *Bonk-Schramm-extension-unfolded-wo-props*(*3*) **by** *auto*
  then show *a* ∈ *wo.underS* (*middle a b*)
    by (*metis Diff-iff middle-rel-a pair-in-Id-conv underS-I*)
  have (*b*, *middle a b*) ∈ *Bonk-Schramm-extension-unfolded-wo*
    using *Bonk-Schramm-extension-unfolded-wo-props*(*4*) **by** *auto*
  then show *b* ∈ *wo.underS* (*middle a b*)
    by (*metis Diff-iff middle-rel-b pair-in-Id-conv underS-I*)
  have (*u n*, *would-be-Cauchy u*) ∈ *Bonk-Schramm-extension-unfolded-wo*
    using *Bonk-Schramm-extension-unfolded-wo-props*(*5*) **by** *auto*
  then show *u n* ∈ *wo.underS* (*would-be-Cauchy u*)
    by (*metis Diff-iff cauchy-rel pair-in-Id-conv underS-I*)
qed

We want to define by transfinite induction a distance on `'a Bonk_Schramm_extension_unfolded`, adding one point at a time (i.e., if the distance is defined on E, then one wants to define it on $E \cup \{x\}$, if x is a middle or a potential Cauchy sequence, by prescribing the distance from x to all the points in E.

Technically, we define a family of distances, indexed by x, on $\{y : y \leq x\}^2$. As all functions should be defined everywhere, this will be a family of functions on $X \times X$, indexed by points in X. They will have a compatibility condition, making it possible to define a global distance by gluing them together.

Technically, transfinite induction is implemented in Isabelle/HOL by an updating rule: a function that associates, to a family of distances indexed by x, a new family of distances indexed by x. The result of the transfinite induction is obtained by starting from an arbitrary object, and then applying the updating rule infinitely many times. The characteristic property of the result of this transfinite induction is that it is a fixed point of the updating rule, as it should.

Below, this is implemented as follows:

- `extend_distance` is the updating rule.
- Its fixed point `extend_distance_fp` is by definition `wo.worec extend_distance` (it only makes sense if the udpating rule satisfies a compatibility condition `wo.adm_wo extend_distance` saying that the update of a family, at x, only depends on the value of the family strictly below x.
- Finally, the global distance `extended_distance` is taken as the value of the fixed point above, on xyy' (i.e., using the distance indexed by x) for any $x \geq \max(y, y')$. For definiteness, we use $\max(y, y')$, but it does not matter as everything is compatible.

fun *extend-distance*::($'a$ *Bonk-Schramm-extension-unfolded* $\Rightarrow$ ($'a$ *Bonk-Schramm-extension-unfolded* $\Rightarrow$ $'a$ *Bonk-Schramm-extension-unfolded* $\Rightarrow$ *real*))
$\Rightarrow$ ($'a$ *Bonk-Schramm-extension-unfolded* $\Rightarrow$ ($'a$ *Bonk-Schramm-extension-unfolded* $\Rightarrow$ $'a$ *Bonk-Schramm-extension-unfolded* $\Rightarrow$ *real*))
where
extend-distance f (*basepoint x*) = ($\lambda y\ z$. *if* $y \in$ *range basepoint* $\wedge$ $z \in$ *range basepoint then*
dist (*SOME* y'. y = *basepoint* y') (*SOME* z'. z = *basepoint* z') *else 1*)
| *extend-distance f* (*middle a b*) = ($\lambda y\ z$.
if ($y \in$ *wo.underS* (*middle a b*)) $\wedge$ ($z \in$ *wo.underS* (*middle a b*)) *then f* (*wo.max2 y z*) *y z*
else if ($y \in$ *wo.underS* (*middle a b*)) $\wedge$ (z = *middle a b*) *then* (*f* (*wo.max2 a b*) *a b*)/*2* + (*SUP* $w \in \{z \in$ *wo.underS* (*middle a b*). *f z z z* = *0*$\}$. *f* (*wo.max2 y w*) *y w* $-$ *max* (*f* (*wo.max2 a w*) *a w*) (*f* (*wo.max2 b w*) *b w*))
else if (y = *middle a b*) $\wedge$ ($z \in$ *wo.underS* (*middle a b*)) *then* (*f* (*wo.max2 a b*) *a b*)/*2* + (*SUP* $w \in \{z \in$ *wo.underS* (*middle a b*). *f z z z* = *0*$\}$. *f* (*wo.max2 z w*) *z w* $-$ *max* (*f* (*wo.max2 a w*) *a w*) (*f* (*wo.max2 b w*) *b w*))
else if (y = *middle a b*) $\wedge$ (z = *middle a b*) $\wedge$ (*f a a a* = *0*) $\wedge$ (*f b b b* = *0*) *then 0*
else 1)
| *extend-distance f* (*would-be-Cauchy u*) = ($\lambda y\ z$.
if ($y \in$ *wo.underS* (*would-be-Cauchy u*)) $\wedge$ ($z \in$ *wo.underS* (*would-be-Cauchy u*)) *then f* (*wo.max2 y z*) *y z*
else if ($\neg$($\forall$ *eps* > (*0*::*real*). $\exists N$. $\forall n \geq N$. $\forall m \geq N$. *f* (*wo.max2* (*u n*) (*u m*)) (*u n*) (*u m*) < *eps*)) *then 1*
else if ($y \in$ *wo.underS* (*would-be-Cauchy u*)) $\wedge$ (z = *would-be-Cauchy u*) *then lim* (λn. *f* (*wo.max2* (*u n*) *y*) (*u n*) *y*)
else if (y = *would-be-Cauchy u*) $\wedge$ ($z \in$ *wo.underS* (*would-be-Cauchy u*)) *then lim* (λn. *f* (*wo.max2* (*u n*) *z*) (*u n*) *z*)
else if (y = *would-be-Cauchy u*) $\wedge$ (z = *would-be-Cauchy u*) $\wedge$ ($\forall n$. *f* (*u n*) (*u n*) (*u n*) = *0*) *then 0*
else 1)

definition *extend-distance-fp* = *wo.worec extend-distance*

definition *extended-distance x y* = *extend-distance-fp* (*wo.max2 x y*) *x y*

definition *extended-distance-set* = $\{z$. *extended-distance z z* = *0*$\}$

lemma *wo-adm-extend-distance*:
wo.adm-wo extend-distance
unfolding *wo.adm-wo-def* **proof** (*auto*)
fix *f g*::$'a$ *Bonk-Schramm-extension-unfolded* $\Rightarrow$ $'a$ *Bonk-Schramm-extension-unfolded* $\Rightarrow$ $'a$ *Bonk-Schramm-extension-unfolded* $\Rightarrow$ *real*
fix *x*::$'a$ *Bonk-Schramm-extension-unfolded*
assume $\forall y \in$ *wo.underS x*. *f y* = *g y*
then have $*$: *f y* = *g y* **if** $y \in$ *wo.underS x* **for** *y* **using** *that* **by** *auto*
show *extend-distance f x* = *extend-distance g x*
apply (*cases x*)

apply (*insert Bonk-Schramm-extension-unfolded-wo-props′* ∗)
apply *auto*

apply (*rule ext*)+
apply (*rule if-cong*, *simp*, *simp*)+ **apply** (*rule SUP-cong*, *fastforce*, *blast*)
apply (*rule if-cong*, *simp*, *simp*)+ **apply** (*rule SUP-cong*, *fastforce*, *blast*)
apply (*rule if-cong*, *simp*, *simp*)+ **apply** *simp*

apply (*rule ext*)+
apply (*rule if-cong*, *simp*, *simp*)+
apply *simp*
done
qed

lemma *extend-distance-fp*:
extend-distance-fp = *extend-distance* (*extend-distance-fp*)
using *wo.worec-fixpoint*[*OF wo-adm-extend-distance*] **unfolding** *extend-distance-fp-def*.

lemma *extended-distance-symmetric*:
extended-distance x y = *extended-distance y x*
proof −
have ∗: *extend-distance* (*extend-distance-fp*) *x x y* = *extend-distance* (*extend-distance-fp*) *x y x* **if** *y* ∈ *wo.underS x* **for** *x y*
apply (*cases x*)
apply (*simp add*: *that dist-commute*)+
by *blast*
have ∗∗: *extended-distance x y* = *extended-distance y x* **if** *y* ∈ *wo.underS x* **for** *x y*
unfolding *extended-distance-def* **using** *that* ∗[*OF that*] *extend-distance-fp* **by** *simp*
consider *y* ∈ *wo.underS x*|*x* ∈ *wo.underS y*|*x* = *y*
by (*metis UNIV-I Bonk-Schramm-extension-unfolded-wo-props*(*1*) *that*(*1*) *underS-I well-order-on-Well-order wo.TOTALS*)
then show *?thesis*
apply (*cases*) **using** ∗∗ **by** *auto*
qed

lemma *extended-distance-basepoint*:
extended-distance (*basepoint x*) (*basepoint y*) = *dist x y*
proof −
consider *wo.max2* (*basepoint x*) (*basepoint y*) = *basepoint x* | *wo.max2* (*basepoint x*) (*basepoint y*) = *basepoint y*
by (*meson wo.max2-def*)
then show *?thesis*
apply *cases*
unfolding *extended-distance-def* **by** (*subst extend-distance-fp*, *simp*)+
qed

lemma *extended-distance-set-basepoint*:
  basepoint x ∈ *extended-distance-set*
unfolding *extended-distance-set-def* **using** *extended-distance-basepoint* **by** *auto*

lemma *extended-distance-set-middle*:
  assumes *a* ∈ *extended-distance-set b* ∈ *extended-distance-set*
  shows *middle a b* ∈ *extended-distance-set*
using *assms* **unfolding** *extended-distance-set-def extended-distance-def* **apply** *auto*
by (*metis* (*no-types*, *lifting*) *extend-distance-fp extend-distance.simps*(*2*) *underS-E*)

lemma *extended-distance-set-middle′*:
  assumes *middle a b* ∈ *extended-distance-set*
  shows *a* ∈ *extended-distance-set* ∩ *wo.underS* (*middle a b*)
    b ∈ *extended-distance-set* ∩ *wo.underS* (*middle a b*)
proof −
  have *extend-distance* (*extend-distance-fp*) (*middle a b*) (*middle a b*) (*middle a b*) = *0*
    apply (*subst extend-distance-fp*[*symmetric*])
    using *assms* **unfolding** *extended-distance-set-def extended-distance-def* **by** *simp*
  then have *a* ∈ *extended-distance-set b* ∈ *extended-distance-set*
    unfolding *extended-distance-set-def extended-distance-def* **apply** *auto*
    by (*metis zero-neq-one*)+
  moreover have *a* ∈ *wo.underS* (*middle a b*) *b* ∈ *wo.underS* (*middle a b*)
    by (*auto simp add*: *Bonk-Schramm-extension-unfolded-wo-props′*)
  ultimately show *a* ∈ *extended-distance-set* ∩ *wo.underS* (*middle a b*)
        b ∈ *extended-distance-set* ∩ *wo.underS* (*middle a b*)
    by *auto*
qed

lemma *extended-distance-middle-formula*:
  assumes *x* ∈ *wo.underS* (*middle a b*)
  shows *extended-distance x* (*middle a b*) = (*extended-distance a b*)/*2*
    + (*SUP w*∈*wo.underS* (*middle a b*) ∩ *extended-distance-set*.
      extended-distance x w − *max* (*extended-distance a w*) (*extended-distance b w*))
unfolding *extended-distance-set-def extended-distance-def*
apply (*subst extend-distance-fp*)
apply (*simp add*: *assms*)
apply (*rule SUP-cong*)
apply (*auto simp add*: *wo.max2-def*)
done

lemma *extended-distance-set-Cauchy*:
  assumes (*would-be-Cauchy u*) ∈ *extended-distance-set*
  shows *u n* ∈ *extended-distance-set* ∩ *wo.underS* (*would-be-Cauchy u*)
    $\forall$ *eps* > (*0*::*real*). $\exists$ *N*. $\forall$ *n* ≥ *N*. $\forall$ *m* ≥ *N*. *extended-distance* (*u n*) (*u m*) < *eps*
proof −
  have ∗: *extend-distance* (*extend-distance-fp*) (*would-be-Cauchy u*) (*would-be-Cauchy*

u) (*would-be-Cauchy u*) = *0*
apply (*subst extend-distance-fp*[*symmetric*])
using *assms* **unfolding** *extended-distance-set-def extended-distance-def* **by** *simp*
then have $u\ n \in$ *extended-distance-set*
unfolding *extended-distance-set-def extended-distance-def* **apply** *auto*
by (*metis* (*no-types*, *opaque-lifting*) *underS-notIn zero-neq-one*)
moreover have $u\ n \in$ *wo.underS* (*would-be-Cauchy u*)
by (*auto simp add*: *Bonk-Schramm-extension-unfolded-wo-props′*)
ultimately show $u\ n \in$ *extended-distance-set* $\cap$ *wo.underS* (*would-be-Cauchy u*)
by *auto*
show $\forall\, eps > (0{::}real).\ \exists\, N.\ \forall\, n \geq N.\ \forall\, m \geq N.$ *extended-distance* ($u\ n$) ($u\ m$) $<$ *eps*
using $*$ **unfolding** *extended-distance-set-def extended-distance-def* **apply** *auto*
by (*metis* (*no-types*, *opaque-lifting*) *zero-neq-one*)
qed

lemma *extended-distance-triang-ineq*:
assumes $x \in$ *extended-distance-set*
$y \in$ *extended-distance-set*
$z \in$ *extended-distance-set*
shows *extended-distance* $x\ z \leq$ *extended-distance* $x\ y$ + *extended-distance* $y\ z$
proof −

have *ineq-rec*: $\forall\, x\ y\ z.\ x \in$ *wo.under* $t \cap$ *extended-distance-set* $\longrightarrow y \in$ *wo.under* $t \cap$ *extended-distance-set* $\longrightarrow z \in$ *wo.under* $t \cap$ *extended-distance-set*
$\longrightarrow$ *extended-distance* $x\ z \leq$ *extended-distance* $x\ y$ + *extended-distance* $y\ z$ **for** t
proof (*rule wo.well-order-induct*[*of - t*])
fix t
assume *IH-orig*: $\forall\, t2.\ t2 \neq t \wedge (t2,\ t) \in$ *Bonk-Schramm-extension-unfolded-wo* $\longrightarrow$
($\forall\, x\ y\ z.\ x \in$ *wo.under* $t2 \cap$ *extended-distance-set* $\longrightarrow$
$y \in$ *wo.under* $t2 \cap$ *extended-distance-set* $\longrightarrow$
$z \in$ *wo.under* $t2 \cap$ *extended-distance-set* $\longrightarrow$
extended-distance $x\ z \leq$ *extended-distance* $x\ y$ + *extended-distance* $y\ z$)

then have *IH*: *extended-distance* $x\ z \leq$ *extended-distance* $x\ y$ + *extended-distance* $y\ z$
if $x \in$ *wo.underS* $t \cap$ *extended-distance-set*
$y \in$ *wo.underS* $t \cap$ *extended-distance-set*
$z \in$ *wo.underS* $t \cap$ *extended-distance-set*
for $x\ y\ z$
proof −
define *t2* **where** $t2 =$ *wo.max2* (*wo.max2* $x\ y$) z
have $t2 \in$ *wo.underS* t **using** *that t2-def* **by** *auto*
have $x \in$ *wo.under* $t2$ $y \in$ *wo.under* $t2$ $z \in$ *wo.under* $t2$ **unfolding** *t2-def*
by (*metis UNIV-I Bonk-Schramm-extension-unfolded-wo-props*(*1*) *mem-Collect-eq under-def well-order-on-Well-order wo.TOTALS wo.max2-iff*)+

then show *?thesis* **using** *that IH-orig* ‹*t2* ∈ *wo.underS t*› *underS-E* **by** *fastforce*
qed

have *pos*: *extended-distance x y* ≥ *0* **if** *x* ∈ *wo.underS t* ∩ *extended-distance-set* *y* ∈ *wo.underS t* ∩ *extended-distance-set* **for** *x y*
proof −
have *0* = *extended-distance x x* **using** *that*(*1*) *extended-distance-set-def* **by** *auto*
also have ... ≤ *extended-distance x y* + *extended-distance y x*
using *IH that* **by** *auto*
also have ... = *2* ∗ *extended-distance x y*
using *extended-distance-symmetric* **by** *auto*
finally show *?thesis* **by** *auto*
qed

consider *t* ∉ *extended-distance-set* | *t* ∈ *extended-distance-set* **by** *auto*
then show ∀ *x y z*. *x* ∈ *wo.under t* ∩ *extended-distance-set* ⟶
y ∈ *wo.under t* ∩ *extended-distance-set* ⟶
z ∈ *wo.under t* ∩ *extended-distance-set* ⟶
extended-distance x z ≤ *extended-distance x y* + *extended-distance y z*
proof (*cases*)
case *1*
then have *wo.under t* ∩ *extended-distance-set* = *wo.underS t* ∩ *extended-distance-set*
apply *auto*
apply (*metis mem-Collect-eq underS-I under-def*)
by (*simp add*: *underS-E under-def*)
then show *?thesis* **using** *IH* **by** *auto*
next
case *2*

have *main-ineq*: *extended-distance x z* ≤ *extended-distance x t* + *extended-distance t z*
∧ *extended-distance x t* ≤ *extended-distance x z* + *extended-distance z t*
if *x* ∈ *wo.underS t* ∩ *extended-distance-set*
z ∈ *wo.underS t* ∩ *extended-distance-set*
for *x z*
proof (*cases t*)

case *A*: (*basepoint t'*)
then have *x* ∈ *range basepoint* **using** *Bonk-Schramm-extension-unfolded-wo-props*(*2*)
by (*metis that*(*1*) *Compl-iff Int-iff range-eqI wo.max2-def wo.max2-underS'*(*2*))
then obtain *x'* **where** *x*: *x* = *basepoint x'* **by** *auto*
have *z* ∈ *range basepoint* **using** *Bonk-Schramm-extension-unfolded-wo-props*(*2*) *A*
by (*metis that*(*2*) *Compl-iff Int-iff range-eqI wo.max2-def wo.max2-underS'*(*2*))
then obtain *z'* **where** *z*: *z* = *basepoint z'* **by** *auto*

show *extended-distance x z* ≤ *extended-distance x t* + *extended-distance t z*
  ∧ *extended-distance x t* ≤ *extended-distance x z* + *extended-distance z t*
 unfolding *x z A extended-distance-basepoint* **by** (*simp add*: *dist-triangle*)
next

case *M*: (*middle a b*)
then have *ab*: *a* ∈ *extended-distance-set* ∩ *wo.underS* (*middle a b*)
  b ∈ *extended-distance-set* ∩ *wo.underS* (*middle a b*)
 using *2 extended-distance-set-middle′*[*of a b*] **by** *auto*
have *dxt*: *extended-distance x t* = (*extended-distance a b*)/*2*
 + (*SUP w*∈*wo.underS* (*middle a b*) ∩ *extended-distance-set*.
  extended-distance x w − *max* (*extended-distance a w*) (*extended-distance b w*))
 using *that*(*1*) **unfolding** *M* **using** *extended-distance-middle-formula* **by** *auto*
have *dzt*: *extended-distance z t* = (*extended-distance a b*)/*2*
 + (*SUP w*∈*wo.underS* (*middle a b*) ∩ *extended-distance-set*.
  extended-distance z w − *max* (*extended-distance a w*) (*extended-distance b w*))
 using *that*(*2*) **unfolding** *M* **using** *extended-distance-middle-formula* **by** *auto*

have *bdd*: *bdd-above* ((λ*w*. *extended-distance x w* − *max* (*extended-distance a w*) (*extended-distance b w*))‘ (*wo.underS* (*middle a b*) ∩ *extended-distance-set*))
 if *x* ∈ *wo.underS t* ∩ *extended-distance-set* **for** *x*
proof (*rule bdd-aboveI2*)
 fix *w* **assume** *w*: *w* ∈ *wo.underS* (*middle a b*) ∩ *extended-distance-set*
 have *extended-distance x w* ≤ *extended-distance x a* + *extended-distance a w*
  apply (*rule IH*) **using** *ab w M that*(*1*) **by** *auto*
  also have ... ≤ *extended-distance x a* + *max* (*extended-distance a w*) (*extended-distance b w*)
  by *auto*
  finally show *extended-distance x w* − *max* (*extended-distance a w*) (*extended-distance b w*)
   ≤ *extended-distance x a*
  by *auto*
qed

have (λ*w*. *extended-distance x z* + *extended-distance z w* − *max* (*extended-distance a w*) (*extended-distance b w*)) ‘ (*underS Bonk-Schramm-extension-unfolded-wo* (*middle a b*) ∩ *extended-distance-set*)
  = (λ*s*. *s* + *extended-distance x z*)‘ (λ*w*. *extended-distance z w* − *max* (*extended-distance a w*) (*extended-distance b w*)) ‘ (*underS Bonk-Schramm-extension-unfolded-wo* (*middle a b*) ∩ *extended-distance-set*)
 by *auto*
 moreover have *bdd-above* ((λ*s*. *s* + *extended-distance x z*)‘ (λ*w*. *extended-distance z w* − *max* (*extended-distance a w*) (*extended-distance b w*)) ‘ (*underS Bonk-Schramm-extension-unfolded-wo* (*middle a b*) ∩ *extended-distance-set*))

apply (*rule bdd-above-image-mono*) **using** *bdd that* **by** (*auto simp add*: *mono-def*)
ultimately have *bdd-3*: *bdd-above* ((λ*w*. *extended-distance x z* + *extended-distance z w* − *max* (*extended-distance a w*) (*extended-distance b w*)) ‘ (*underS Bonk-Schramm-extension-unfolded-wo* (*middle a b*) ∩ *extended-distance-set*))
by *simp*

have ∗∗: *max* (*extended-distance a a*) (*extended-distance b a*) = *extended-distance b a*
apply (*rule max-absorb2*) **using** *pos ab extended-distance-set-def M* **by** *auto*
then have −*extended-distance a b* / *2* + *extended-distance x a*
= (*extended-distance a b*)/*2* + *extended-distance x a* − *max* (*extended-distance a a*) (*extended-distance b a*)
unfolding *extended-distance-symmetric*[*of a b*] **by** *auto*
also have ... ≤ *extended-distance x t*
unfolding *dxt* **apply** (*simp*, *rule cSUP-upper*, *simp*) **using** *bdd that M ab* **by** *auto*
finally have *D1*: −*extended-distance a b* / *2* + *extended-distance x a* ≤ *extended-distance x t*
by *simp*

have ∗∗: *max* (*extended-distance a b*) (*extended-distance b b*) = *extended-distance a b*
apply (*rule max-absorb1*) **using** *pos ab extended-distance-set-def M* **by** *auto*
then have −*extended-distance a b* / *2* + *extended-distance x b*
= (*extended-distance a b*)/*2* + *extended-distance x b* − *max* (*extended-distance a b*) (*extended-distance b b*)
unfolding *extended-distance-symmetric*[*of a b*] **by** *auto*
also have ... ≤ *extended-distance x t*
unfolding *dxt* **apply** (*simp*, *rule cSUP-upper*, *simp*) **using** *bdd that ab* **by** *auto*
finally have −*extended-distance a b* / *2* + *extended-distance x b* ≤ *extended-distance x t*
by *simp*
then have *D2*: −*extended-distance a b* / *2* + *max* (*extended-distance x a*) (*extended-distance x b*) ≤ *extended-distance x t*
using *D1* **by** *auto*

have *extended-distance x z* = (−*extended-distance a b* / *2* + *max* (*extended-distance x a*) (*extended-distance x b*)) +
(*extended-distance a b* / *2* + *extended-distance x z* − *max* (*extended-distance x a*) (*extended-distance x b*))
by *auto*
also have ... ≤ *extended-distance x t* +
(*extended-distance a b* / *2* + *extended-distance z x* − *max* (*extended-distance a x*) (*extended-distance b x*))
using *D2 extended-distance-symmetric* **by** *auto*

also have ... $\leq$ *extended-distance x t* + *extended-distance z t*
unfolding *dzt* **apply** (*simp*, *rule cSUP-upper*) **using** *bdd that M ab* **by** *auto*
finally have *I*: *extended-distance x z* $\leq$ *extended-distance x t* + *extended-distance z t*
using *extended-distance-symmetric* **by** *auto*

have *T*: *underS Bonk-Schramm-extension-unfolded-wo* (*middle a b*) $\cap$ *extended-distance-set* $\neq$ {}
mono ((+) (*extended-distance x z*))
bij ((+) (*extended-distance x z*))
using *ab*(*1*) **apply** *blast*
by (*simp add*: *monoI*, *rule bij-betw-byWitness*[*of* - λ*s. s* − (*extended-distance x z*)], *auto*)
have *extended-distance x t* $\leq$ (*extended-distance a b*)/*2*
+ (*SUP w*$\in$*wo.underS* (*middle a b*) $\cap$ *extended-distance-set*.
extended-distance x z + *extended-distance z w* − *max* (*extended-distance a w*) (*extended-distance b w*))
unfolding *dxt* **apply** (*simp*, *rule cSUP-subset-mono*)
using *M that IH bdd-3* **by** (*auto*)
also have ... = *extended-distance x z* + *extended-distance z t*
unfolding *dzt* **apply** *simp*
using *mono-cSup-bij*[*of* (λ*w. extended-distance z w* − *max* (*extended-distance a w*) (*extended-distance b w*))‘(*wo.underS* (*middle a b*) $\cap$ *extended-distance-set*) λ*s. extended-distance x z* + *s*, *OF* - - *T*(*2*) *T*(*3*)]
by (*auto simp add*: *bdd* [*OF that*(*2*)] *ab*(*1*) *T*(*1*) *add-diff-eq image-comp*)
finally have *extended-distance x t* $\leq$ *extended-distance x z* + *extended-distance z t* **by** *simp*
then show *extended-distance x z* $\leq$ *extended-distance x t* + *extended-distance t z*
$\wedge$ *extended-distance x t* $\leq$ *extended-distance x z* + *extended-distance z t*
using *I extended-distance-symmetric* **by** *auto*
next

case *C*: (*would-be-Cauchy u*)
then have *un*: *u n* $\in$ *extended-distance-set* $\cap$ *wo.underS* (*would-be-Cauchy u*) **for** *n*
using *extended-distance-set-Cauchy 2* **by** *auto*
have *lim*: (λ*n. extended-distance y* (*u n*)) $\longrightarrow$ (*extended-distance y* (*would-be-Cauchy u*))
if *y*: *y* $\in$ *extended-distance-set* $\cap$ *wo.underS* (*would-be-Cauchy u*) **for** *y*
proof −
have *extend-distance extend-distance-fp* (*wo.max2* (*would-be-Cauchy u*) (*would-be-Cauchy u*)) (*would-be-Cauchy u*) (*would-be-Cauchy u*) = *0*
using *2* **unfolding** *C extended-distance-set-def extended-distance-def*
using *extend-distance-fp* **by** *auto*
then have *cauch*: $\exists N. \forall n \geq N. \forall m \geq N.$ *extend-distance-fp* (*wo.max2* (*u n*) (*u m*)) (*u n*) (*u m*) < *e* **if** *e* > *0* **for** *e*

apply *auto* **using** ‹$e > 0$› **by** (*metis* (*no-types*, *opaque-lifting*) *zero-neq-one*)
have $\exists N. \forall n \geq N. \forall m \geq N.$ *abs*(*extended-distance y* (*u n*) $-$ *extended-distance y* (*u m*)) $<$ *e* **if** $e > 0$ **for** *e*
proof $-$
obtain *N* **where** $*$: *extend-distance-fp* (*wo.max2* (*u n*) (*u m*)) (*u n*) (*u m*) $<$ *e* **if** $n \geq N$ $m \geq N$ **for** *m n*
using *cauch* **by** (*meson* ‹$0 < e$›)
{
fix *m n* **assume** $m \geq N$ $n \geq N$
then have *e*: *extended-distance* (*u n*) (*u m*) $<$ *e* **using** $*$ **unfolding** *extended-distance-def* **by** *auto*
have *extended-distance y* (*u n*) $\leq$ *extended-distance y* (*u m*) $+$ *extended-distance* (*u m*) (*u n*)
using *IH y un C* **by** *blast*
then have *1*: *extended-distance y* (*u n*) $-$ *extended-distance y* (*u m*) $<$ *e*
using *e extended-distance-symmetric* **by** *auto*
have *extended-distance y* (*u m*) $\leq$ *extended-distance y* (*u n*) $+$ *extended-distance* (*u n*) (*u m*)
using *IH y un C* **by** *blast*
then have *extended-distance y* (*u m*) $-$ *extended-distance y* (*u n*) $<$ *e*
using *e extended-distance-symmetric* **by** *auto*
then have *abs*(*extended-distance y* (*u n*) $-$ *extended-distance y* (*u m*)) $<$ *e*
using *1* **by** *auto*
}
then show *?thesis* **by** *auto*
qed
then have *convergent* ($\lambda n.$ *extended-distance y* (*u n*))
by (*simp add*: *Cauchy-iff real-Cauchy-convergent*)
then have *lim*: ($\lambda n.$ *extended-distance y* (*u n*)) $\longrightarrow$ *lim* ($\lambda n.$ *extended-distance y* (*u n*))
using *convergent-LIMSEQ-iff* **by** *auto*
have $*$: *wo.max2 y* (*would-be-Cauchy u*) $=$ *would-be-Cauchy u* $y \neq$ *would-be-Cauchy u* **using** *y* **by** *auto*
have *extended-distance y* (*would-be-Cauchy u*) $=$ *lim* ($\lambda n.$ *extended-distance* (*u n*) *y*)
unfolding *extended-distance-def* **apply** (*subst extend-distance-fp*) **unfolding** $*$
using $*(2)$ *y cauch* **by** *auto*
then show ($\lambda n.$ *extended-distance y* (*u n*)) $\longrightarrow$ *extended-distance y* (*would-be-Cauchy u*)
using *lim extended-distance-symmetric* **by** *auto*
qed
have *extended-distance x z* $\leq$ *extended-distance x* (*u n*) $+$ *extended-distance* (*u n*) *z* **for** *n*
using *IH un that C* **by** *auto*
moreover have ($\lambda n.$ *extended-distance x* (*u n*) $+$ *extended-distance* (*u n*)

$z) \longrightarrow$ *extended-distance* $x\ t$ + *extended-distance* $t\ z$
apply (*auto intro*!: *tendsto-add*)
using *lim that extended-distance-symmetric* **unfolding** C **by** *auto*
ultimately have I: *extended-distance* $x\ z \leq$ *extended-distance* $x\ t$ + *extended-distance* $t\ z$
using *LIMSEQ-le-const* **by** *blast*

have *extended-distance* $x\ (u\ n) \leq$ *extended-distance* $x\ z$ + *extended-distance* $z\ (u\ n)$ **for** n
using *IH un that* C **by** *auto*
moreover have (λn. *extended-distance* $x\ (u\ n)) \longrightarrow$ *extended-distance* $x\ t$
using *lim that extended-distance-symmetric* **unfolding** C **by** *auto*
moreover have (λn. *extended-distance* $x\ z$ + *extended-distance* $z\ (u\ n)) \longrightarrow$ *extended-distance* $x\ z$ + *extended-distance* $z\ t$
apply (*auto intro*!: *tendsto-add*)
using *lim that extended-distance-symmetric* **unfolding** C **by** *auto*
ultimately have *extended-distance* $x\ t \leq$ *extended-distance* $x\ z$ + *extended-distance* $z\ t$
using *LIMSEQ-le* **by** *blast*
then show *extended-distance* $x\ z \leq$ *extended-distance* $x\ t$ + *extended-distance* $t\ z$
$\wedge$ *extended-distance* $x\ t \leq$ *extended-distance* $x\ z$ + *extended-distance* $z\ t$
using I **by** *auto*
qed

{
fix $x\ y\ z$ **assume** H: $x \in$ *wo.under* $t\ \cap$ *extended-distance-set*
$y \in$ *wo.under* $t\ \cap$ *extended-distance-set*
$z \in$ *wo.under* $t\ \cap$ *extended-distance-set*
have t: *extended-distance* $t\ t = 0$ *extended-distance* $t\ t \geq 0$ **using** *2 extended-distance-set-def* **by** *auto*
have $*$: (($x \in$ *wo.underS* $t\ \cap$ *extended-distance-set*) $\vee$ ($x = t$))
$\wedge$ (($y \in$ *wo.underS* $t\ \cap$ *extended-distance-set*) $\vee$ ($y = t$))
$\wedge$ (($z \in$ *wo.underS* $t\ \cap$ *extended-distance-set*) $\vee$ ($z = t$))
using H **by** (*simp add*: *underS-def under-def*)
have *extended-distance* $x\ z \leq$ *extended-distance* $x\ y$ + *extended-distance* $y\ z$
using $*$ **apply** *auto*
using t *main-ineq extended-distance-symmetric IH pos* **apply** *blast*
using t *main-ineq extended-distance-symmetric IH pos* **apply** *blast*
using t *main-ineq extended-distance-symmetric IH pos* **apply** *blast*
using t *main-ineq extended-distance-symmetric IH pos* **apply** *blast*
using t *main-ineq extended-distance-symmetric IH pos* **apply** (*metis* $*$ *Int-commute add.commute underS-notIn*)
using t *main-ineq extended-distance-symmetric IH pos* **apply** (*metis* (*mono-tags*, *lifting*) $*$ *extended-distance-set-def mem-Collect-eq underS-notIn*)
using t **by** *auto*

}
then show *?thesis* by *auto*
qed
qed

define *t* where $t = wo.max2\ (wo.max2\ x\ y)\ z$
have $x \in wo.under\ t$ $y \in wo.under\ t$ $z \in wo.under\ t$
unfolding *t-def*
by (*metis UNIV-I Bonk-Schramm-extension-unfolded-wo-props*(*1*) *mem-Collect-eq under-def well-order-on-Well-order wo.max2-equals1 wo.max2-iff wo.max2-xx*)+
then show *?thesis* using *assms ineq-rec* by *auto*
qed

We can now show the two main properties of the construction: the middle is indeed a middle from the metric point of view (in `extended_distance_middle`), and Cauchy sequences have a limit (the corresponding `would_be_Cauchy` point).

lemma *extended-distance-pos*:
assumes $a \in$ *extended-distance-set*
$b \in$ *extended-distance-set*
shows *extended-distance a b* ≥ 0
using *assms extended-distance-set-def extended-distance-triang-ineq*[*of a b a*]
unfolding *extended-distance-symmetric*[*of b a*] **by** *auto*

lemma *extended-distance-middle*:
assumes $a \in$ *extended-distance-set*
$b \in$ *extended-distance-set*
shows *extended-distance a* (*middle a b*) = *extended-distance a b* / *2*
extended-distance b (*middle a b*) = *extended-distance a b* / *2*
proof −
have *0* = *extended-distance a b* − *max* (*extended-distance a b*) (*extended-distance b b*)
using *extended-distance-pos*[*OF assms*] *assms*(*2*) *extended-distance-set-def* **by** *auto*
also have ... ≤ (*SUP w∈wo.underS* (*middle a b*) ∩ *extended-distance-set*.
extended-distance a w − *max* (*extended-distance a w*) (*extended-distance b w*))
apply (*rule cSUP-upper*)
apply (*simp add*: *assms*(*2*) *Bonk-Schramm-extension-unfolded-wo-props'*(*2*))
by (*rule bdd-aboveI2*[*of - - 0*], *auto*)
ultimately have *0* ≤ (*SUP w∈wo.underS* (*middle a b*) ∩ *extended-distance-set*.
extended-distance a w − *max* (*extended-distance a w*) (*extended-distance b w*))
by *auto*
moreover have (*SUP w∈wo.underS* (*middle a b*) ∩ *extended-distance-set*.
extended-distance a w − *max* (*extended-distance a w*) (*extended-distance b w*)) ≤ *0*
apply (*rule cSUP-least*)
using *assms*(*1*) *Bonk-Schramm-extension-unfolded-wo-props'*(*1*) **by** (*fastforce*,

auto)
moreover have *extended-distance a* (*middle a b*) = (*extended-distance a b*)/*2*
+ (*SUP w*∈*wo.underS* (*middle a b*) ∩ *extended-distance-set*.
extended-distance a w − *max* (*extended-distance a w*) (*extended-distance b w*))
by (*rule extended-distance-middle-formula*, *simp add*: *Bonk-Schramm-extension-unfolded-wo-props*′(*1*))
ultimately show *extended-distance a* (*middle a b*) = (*extended-distance a b*)/*2*
by *auto*

have *0* = *extended-distance b a* − *max* (*extended-distance a a*) (*extended-distance b a*)
using *extended-distance-pos*[*OF assms*] *assms*(*1*) *extended-distance-set-def extended-distance-symmetric* **by** *auto*
also have ... ≤ (*SUP w*∈*wo.underS* (*middle a b*) ∩ *extended-distance-set*.
extended-distance b w − *max* (*extended-distance a w*) (*extended-distance b w*))
apply (*rule cSUP-upper*)
apply (*simp add*: *assms*(*1*) *Bonk-Schramm-extension-unfolded-wo-props*′(*1*))
by (*rule bdd-aboveI2*[*of - - 0*], *auto*)
ultimately have *0* ≤ (*SUP w*∈*wo.underS* (*middle a b*) ∩ *extended-distance-set*.
extended-distance b w − *max* (*extended-distance a w*) (*extended-distance b w*))
by *auto*
moreover have (*SUP w*∈*wo.underS* (*middle a b*) ∩ *extended-distance-set*.
extended-distance b w − *max* (*extended-distance a w*) (*extended-distance b w*)) ≤ *0*
apply (*rule cSUP-least*)
using *assms*(*1*) *Bonk-Schramm-extension-unfolded-wo-props*′(*1*) **by** (*fastforce*, *auto*)
moreover have *extended-distance b* (*middle a b*) = (*extended-distance a b*)/*2*
+ (*SUP w*∈*wo.underS* (*middle a b*) ∩ *extended-distance-set*.
extended-distance b w − *max* (*extended-distance a w*) (*extended-distance b w*))
by (*rule extended-distance-middle-formula*, *simp add*: *Bonk-Schramm-extension-unfolded-wo-props*′(*2*))
ultimately show *extended-distance b* (*middle a b*) = (*extended-distance a b*)/*2*
by *auto*
qed

lemma *extended-distance-Cauchy*:
assumes ⋀(*n*::*nat*). *u n* ∈ *extended-distance-set*
and ∀ *eps* > (*0*::*real*). ∃ *N*. ∀ *n* ≥ *N*. ∀ *m* ≥ *N*. *extended-distance* (*u n*) (*u m*) < *eps*
shows *would-be-Cauchy u* ∈ *extended-distance-set*
(λ*n*. *extended-distance* (*u n*) (*would-be-Cauchy u*)) ⟶ *0*
proof −
show *2*: *would-be-Cauchy u* ∈ *extended-distance-set*
unfolding *extended-distance-set-def extended-distance-def* **apply** (*simp*, *subst extend-distance-fp*)
using *assms* **unfolding** *extended-distance-set-def extended-distance-def* **by** *simp*

have *lim*: (λn. *extended-distance y* (*u n*)) $\longrightarrow$ (*extended-distance y* (*would-be-Cauchy u*))
if *y*: $y \in$ *extended-distance-set* $\cap$ *wo.underS* (*would-be-Cauchy u*) **for** *y*
proof $-$
have $\exists N. \forall n \geq N. \forall m \geq N.$ *abs*(*extended-distance y* (*u n*) $-$ *extended-distance y* (*u m*)) $<$ *e* **if** $e > 0$ **for** *e*
proof $-$
obtain *N* **where** $*$: *extended-distance* (*u n*) (*u m*) $<$ *e* **if** $n \geq N$ $m \geq N$ **for** *m n*
using *assms*(*2*) *that* ‹$e > 0$› **by** *meson*
{
fix *m n* **assume** $m \geq N$ $n \geq N$
then have *e*: *extended-distance* (*u n*) (*u m*) $<$ *e* **using** $*$ **by** *auto*
have *extended-distance y* (*u n*) $\leq$ *extended-distance y* (*u m*) $+$ *extended-distance* (*u m*) (*u n*)
using *extended-distance-triang-ineq y assms*(*1*) **by** *blast*
then have *1*: *extended-distance y* (*u n*) $-$ *extended-distance y* (*u m*) $<$ *e*
using *e extended-distance-symmetric* **by** *auto*
have *extended-distance y* (*u m*) $\leq$ *extended-distance y* (*u n*) $+$ *extended-distance* (*u n*) (*u m*)
using *extended-distance-triang-ineq y assms*(*1*) **by** *blast*
then have *extended-distance y* (*u m*) $-$ *extended-distance y* (*u n*) $<$ *e*
using *e extended-distance-symmetric* **by** *auto*
then have *abs*(*extended-distance y* (*u n*) $-$ *extended-distance y* (*u m*)) $<$ *e*
using *1* **by** *auto*
}
then show *?thesis* **by** *auto*
qed
then have *convergent* (λn. *extended-distance y* (*u n*))
by (*simp add*: *Cauchy-iff real-Cauchy-convergent*)
then have *lim*: (λn. *extended-distance y* (*u n*)) $\longrightarrow$ *lim* (λn. *extended-distance y* (*u n*))
using *convergent-LIMSEQ-iff* **by** *auto*
have $*$: *wo.max2 y* (*would-be-Cauchy u*) = *would-be-Cauchy u* *y* $\neq$ *would-be-Cauchy u* **using** *y* **by** *auto*
have *extended-distance y* (*would-be-Cauchy u*) = *lim* (λn. *extended-distance* (*u n*) *y*)
unfolding *extended-distance-def* **apply** (*subst extend-distance-fp*) **unfolding** $*$
using $*$(*2*) *y assms*(*2*) *extended-distance-def* **by** *auto*
then show (λn. *extended-distance y* (*u n*)) $\longrightarrow$ *extended-distance y* (*would-be-Cauchy u*)
using *lim extended-distance-symmetric* **by** *auto*
qed

have $\exists N. \forall n \geq N.$ *abs*(*extended-distance* (*u n*) (*would-be-Cauchy u*)) $<$ *e* **if** $e > 0$ **for** *e*
proof $-$

obtain *N* **where** ∗: *extended-distance* (*u n*) (*u m*) $<$ *e*/*2* **if** $n \geq N$ $m \geq N$ **for** *m n*
using *assms*(*2*) *that* ‹*e* $>$ *0*› **by** (*meson half-gt-zero*)
have *abs*(*extended-distance* (*u n*) (*would-be-Cauchy u*)) $\leq$ *e*/*2* **if** $n \geq N$ **for** *n*
proof −
have *eventually* (λ*m*. *extended-distance* (*u n*) (*u m*) $\leq$ *e*/*2*) *sequentially*
apply (*rule eventually-sequentiallyI*[*of N*]) **using** ∗[*OF* ‹$n \geq N$›] *less-imp-le* **by** *auto*
moreover have (λ*m*. *extended-distance* (*u n*) (*u m*)) ⟶ *extended-distance* (*u n*) (*would-be-Cauchy u*)
apply (*rule lim*) **using** *2 extended-distance-set-Cauchy* **by** *auto*
ultimately have *extended-distance* (*u n*) (*would-be-Cauchy u*) $\leq$ *e*/*2*
by (*meson* ∗ *LIMSEQ-le-const2 less-imp-le that*)
then show *?thesis* **using** *extended-distance-pos*[*OF assms*(*1*)[*of n*] *2*] **by** *auto*
qed
then show *?thesis* **using** ‹*e* $>$ *0*› **by** *force*
qed
then show (λ*n*. *extended-distance* (*u n*) (*would-be-Cauchy u*)) ⟶ *0*
using *LIMSEQ-iff* **by** *force*
qed

end

12.2 The Bonk Schramm extension

quotient-type (**overloaded**) ′*a Bonk-Schramm-extension* =
(′*a*::*metric-space*) *Bonk-Schramm-extension-unfolded*
/ *partial*: λ*x y*. (*x* ∈ *extended-distance-set* ∧ *y* ∈ *extended-distance-set* ∧ *extended-distance x y* = *0*)
unfolding *part-equivp-def* **proof**(*auto intro*!: *ext simp*: *extended-distance-set-def*)
show ∃*x*. *extended-distance x x* = *0*
using *extended-distance-set-basepoint extended-distance-set-def* **by** *auto*
next
fix *x y z*::′*a Bonk-Schramm-extension-unfolded*
assume *H*: *extended-distance x x* = *0* *extended-distance y y* = *0* *extended-distance z z* = *0*
extended-distance x y = *0* *extended-distance x z* = *0*
have *extended-distance y z* $\leq$ *extended-distance y x* + *extended-distance x z*
apply (*rule extended-distance-triang-ineq*)
using *H* **unfolding** *extended-distance-set-def* **by** *auto*
also have ... $\leq$ *0*
by (*auto simp add*: *extended-distance-symmetric H*)
finally show *extended-distance y z* = *0*
using *extended-distance-pos*[*of y z*] *H* **unfolding** *extended-distance-set-def* **by** *auto*
next
fix *x y z*::′*a Bonk-Schramm-extension-unfolded*
assume *H*: *extended-distance x x* = *0* *extended-distance y y* = *0* *extended-distance z z* = *0*

extended-distance x y = *0 extended-distance y z* = *0*
have *extended-distance x z* ≤ *extended-distance x y* + *extended-distance y z*
apply (*rule extended-distance-triang-ineq*)
using *H* **unfolding** *extended-distance-set-def* **by** *auto*
also have ... ≤ *0*
by (*auto simp add*: *extended-distance-symmetric H*)
finally show *extended-distance x z* = *0*
using *extended-distance-pos*[*of x z*] *H* **unfolding** *extended-distance-set-def* **by** *auto*
qed (*metis*)

instantiation *Bonk-Schramm-extension* :: (*metric-space*) *metric-space*
begin

lift-definition *dist-Bonk-Schramm-extension*::(*'a*::*metric-space*) *Bonk-Schramm-extension* ⇒ *'a Bonk-Schramm-extension* ⇒ *real*
is *λx y. extended-distance x y*
proof −
fix *x y z t*::*'a Bonk-Schramm-extension-unfolded*
assume *H*: *x* ∈ *extended-distance-set* ∧ *y* ∈ *extended-distance-set* ∧ *extended-distance x y* = *0*
z ∈ *extended-distance-set* ∧ *t* ∈ *extended-distance-set* ∧ *extended-distance z t* = *0*
have *extended-distance x z* ≤ *extended-distance x y* + *extended-distance y t* + *extended-distance t z*
using *extended-distance-triang-ineq*[*of x y z*] *extended-distance-triang-ineq*[*of y t z*] *H*
by *auto*
also have ... = *extended-distance y t*
using *H* **by** (*auto simp add*: *extended-distance-symmetric*)
finally have ∗: *extended-distance x z* ≤ *extended-distance y t* **by** *simp*
have *extended-distance y t* ≤ *extended-distance y x* + *extended-distance x z* + *extended-distance z t*
using *extended-distance-triang-ineq*[*of y x t*] *extended-distance-triang-ineq*[*of x z t*] *H*
by *auto*
also have ... = *extended-distance x z*
using *H* **by** (*auto simp add*: *extended-distance-symmetric*)
finally show *extended-distance x z* = *extended-distance y t* **using** ∗ **by** *simp*
qed

To define a metric space in the current library of Isabelle/HOL, one should also introduce a uniformity structure and a topology, as follows (they are prescribed by the distance):

definition *uniformity-Bonk-Schramm-extension*::((*'a Bonk-Schramm-extension*) × (*'a Bonk-Schramm-extension*)) *filter*
where *uniformity-Bonk-Schramm-extension* = (*INF e*∈{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open-Bonk-Schramm-extension* :: *'a Bonk-Schramm-extension set* ⇒ *bool*
where *open-Bonk-Schramm-extension U* = (∀ *x*∈*U*. *eventually* (λ(*x'*, *y*). *x'* = *x* ⟶ *y* ∈ *U*) *uniformity*)

instance proof
fix *x y*::*'a Bonk-Schramm-extension*
have *C*: *rep-Bonk-Schramm-extension x* ∈ *extended-distance-set*
rep-Bonk-Schramm-extension y ∈ *extended-distance-set*
using *Quotient3-Bonk-Schramm-extension Quotient3-rep-reflp* **by** *fastforce*+
show (*dist x y* = *0*) = (*x* = *y*)
apply (*subst Quotient3-rel-rep*[*OF Quotient3-Bonk-Schramm-extension*, *symmetric*])
unfolding *dist-Bonk-Schramm-extension-def* **using** *C* **by** *auto*
next
fix *x y z*::*'a Bonk-Schramm-extension*
have *C*: *rep-Bonk-Schramm-extension x* ∈ *extended-distance-set*
rep-Bonk-Schramm-extension y ∈ *extended-distance-set*
rep-Bonk-Schramm-extension z ∈ *extended-distance-set*
using *Quotient3-Bonk-Schramm-extension Quotient3-rep-reflp* **by** *fastforce*+
show *dist x y* ≤ *dist x z* + *dist y z*
unfolding *dist-Bonk-Schramm-extension-def* **apply** *auto*
by (*metis C extended-distance-symmetric extended-distance-triang-ineq*)
qed (*auto simp add*: *uniformity-Bonk-Schramm-extension-def open-Bonk-Schramm-extension-def*)
end

instance *Bonk-Schramm-extension* :: (*metric-space*) *complete-space*
proof
fix *X*::*nat* ⇒ *'a Bonk-Schramm-extension* **assume** *Cauchy X*
have ∗: ⋀*n*. *rep-Bonk-Schramm-extension* (*X n*) ∈ *extended-distance-set*
using *Quotient3-Bonk-Schramm-extension Quotient3-rep-reflp* **by** *fastforce*
have ∗∗: *extended-distance* (*rep-Bonk-Schramm-extension* (*X n*)) (*rep-Bonk-Schramm-extension* (*X m*)) = *dist* (*X n*) (*X m*) **for** *m n*
unfolding *dist-Bonk-Schramm-extension-def* **by** *auto*
define *y* **where** *y* = *would-be-Cauchy* (λ*n*. *rep-Bonk-Schramm-extension* (*X n*))
have *y* ∈ *extended-distance-set*
unfolding *y-def* **apply** (*rule extended-distance-Cauchy*)
using ∗ ‹*Cauchy X*› **unfolding** *Cauchy-def* ∗∗[*symmetric*] **by** *auto*
define *x* **where** *x* = *abs-Bonk-Schramm-extension y*
have *dist* (*X n*) *x* = *extended-distance* (*rep-Bonk-Schramm-extension* (*X n*)) *y* **for** *n*
unfolding *x-def* **apply** (*subst Quotient3-abs-rep*[*OF Quotient3-Bonk-Schramm-extension*, *symmetric*])
apply (*rule dist-Bonk-Schramm-extension.abs-eq*) **using** ∗ ‹*y* ∈ *extended-distance-set*›
by (*auto simp add*: *extended-distance-set-def*)
moreover have (λ*n*. *extended-distance* (*rep-Bonk-Schramm-extension* (*X n*)) *y*) ⟶ *0*
unfolding *y-def* **apply** (*rule extended-distance-Cauchy*)

```
    using * ‹Cauchy X› unfolding Cauchy-def **[symmetric] by auto
  ultimately have *: (λn. dist (X n) x) ⟶ 0 by simp
  have X ⟶ x
    apply (rule tendstoI) using * by (auto simp add: order-tendsto-iff)
  then show convergent X unfolding convergent-def by auto
qed

instance Bonk-Schramm-extension :: (metric-space) geodesic-space
proof (rule complete-with-middles-imp-geodesic)
  fix x y::'a Bonk-Schramm-extension
  have H: rep-Bonk-Schramm-extension x ∈ extended-distance-set
          rep-Bonk-Schramm-extension y ∈ extended-distance-set
    using Quotient3-Bonk-Schramm-extension Quotient3-rep-reflp by fastforce+
  define M where M = middle (rep-Bonk-Schramm-extension x) (rep-Bonk-Schramm-extension
y)
  then have M: M ∈ extended-distance-set
    using extended-distance-set-middle[OF H] by simp
  define m where m = abs-Bonk-Schramm-extension M

  have dist x m = extended-distance (rep-Bonk-Schramm-extension x) M
    apply (subst Quotient3-abs-rep[OF Quotient3-Bonk-Schramm-extension, sym-
metric]) unfolding m-def
    apply (rule dist-Bonk-Schramm-extension.abs-eq)
    using H M extended-distance-set-def by auto
  also have ... = extended-distance (rep-Bonk-Schramm-extension x) (rep-Bonk-Schramm-extension
y) / 2
    unfolding M-def by (rule extended-distance-middle[OF H])
  also have ... = dist x y / 2
    unfolding dist-Bonk-Schramm-extension-def by auto
  finally have *: dist x m = dist x y / 2 by simp

  have dist m y = extended-distance M (rep-Bonk-Schramm-extension y)
    apply (subst Quotient3-abs-rep[OF Quotient3-Bonk-Schramm-extension, of y,
symmetric]) unfolding m-def
    apply (rule dist-Bonk-Schramm-extension.abs-eq)
    using H M extended-distance-set-def by auto
  also have ... = extended-distance (rep-Bonk-Schramm-extension x) (rep-Bonk-Schramm-extension
y) / 2
    unfolding M-def using extended-distance-middle(2)[OF H] by (simp add:
extended-distance-symmetric)
  also have ... = dist x y / 2
    unfolding dist-Bonk-Schramm-extension-def by auto
  finally have dist m y = dist x y / 2 by simp
  then show ∃m. dist x m = dist x y / 2 ∧ dist m y = dist x y / 2
    using * by auto
qed

definition to-Bonk-Schramm-extension::'a::metric-space ⇒ 'a Bonk-Schramm-extension
  where to-Bonk-Schramm-extension x = abs-Bonk-Schramm-extension (basepoint
```

x)

lemma *to-Bonk-Schramm-extension-isometry*:
isometry-on UNIV to-Bonk-Schramm-extension
proof (*rule isometry-onI*)
fix *x y*::*'a*
show *dist* (*to-Bonk-Schramm-extension x*) (*to-Bonk-Schramm-extension y*) = *dist x y*
unfolding *to-Bonk-Schramm-extension-def* **apply** (*subst dist-Bonk-Schramm-extension.abs-eq*)
unfolding *extended-distance-set-def* **by** (*auto simp add*: *extended-distance-basepoint*)
qed

13 Bonk-Schramm extension of hyperbolic spaces

13.1 The Bonk-Schramm extension preserves hyperbolicity

A central feature of the Bonk-Schramm extension is that it preserves hyperbolicity, with the same hyperbolicity constant δ, as we prove now.

lemma (**in** *Gromov-hyperbolic-space*) *Bonk-Schramm-extension-unfolded-hyperbolic*:
fixes *x y z t*::(*'a*::*metric-space*) *Bonk-Schramm-extension-unfolded*
assumes $x \in$ *extended-distance-set*
$y \in$ *extended-distance-set*
$z \in$ *extended-distance-set*
$t \in$ *extended-distance-set*
shows *extended-distance x y* + *extended-distance z t* $\leq$ *max* (*extended-distance x z* + *extended-distance y t*) (*extended-distance x t* + *extended-distance y z*) + *2* $*$ *deltaG*(*TYPE*(*'a*))
proof −
interpret *wo*: *wo-rel Bonk-Schramm-extension-unfolded-wo*
using *well-order-on-Well-order wo-rel-def wfrec-def metric-space-class.Bonk-Schramm-extension-unfolded-wo-*
by *blast*

have *ineq-rec*: $\forall$ *x y z t*. $x \in$ *wo.under a* $\cap$ *extended-distance-set* $\longrightarrow$ $y \in$ *wo.under a* $\cap$ *extended-distance-set* $\longrightarrow$ $z \in$ *wo.under a* $\cap$ *extended-distance-set* $\longrightarrow$ $t \in$ *wo.under a* $\cap$ *extended-distance-set*
$\longrightarrow$ *extended-distance x y* + *extended-distance z t* $\leq$ *max* (*extended-distance x z* + *extended-distance y t*) (*extended-distance x t* + *extended-distance y z*) + *2* $*$ *deltaG*(*TYPE*(*'a*))
for *a*::*'a Bonk-Schramm-extension-unfolded*
proof (*rule wo.well-order-induct*[*of - a*])
fix *a*::*'a Bonk-Schramm-extension-unfolded*
assume *IH-orig*: $\forall$ *b*. $b \neq a \wedge (b, a) \in$ *Bonk-Schramm-extension-unfolded-wo* $\longrightarrow$
($\forall$ *x y z t*. $x \in$ *wo.under b* $\cap$ *extended-distance-set* $\longrightarrow$
$y \in$ *wo.under b* $\cap$ *extended-distance-set* $\longrightarrow$
$z \in$ *wo.under b* $\cap$ *extended-distance-set* $\longrightarrow$
$t \in$ *wo.under b* $\cap$ *extended-distance-set* $\longrightarrow$

extended-distance x y + extended-distance z t $\leq$ *max* (*extended-distance x z + extended-distance y t*) (*extended-distance x t + extended-distance y z*) + *2* $*$ *deltaG*(*TYPE*(*'a*)))

then have *IH*: *extended-distance x y + extended-distance z t* $\leq$ *max* (*extended-distance x z + extended-distance y t*) (*extended-distance x t + extended-distance y z*) + *2* $*$ *deltaG*(*TYPE*(*'a*))
if *x* $\in$ *wo.underS a* $\cap$ *extended-distance-set*
y $\in$ *wo.underS a* $\cap$ *extended-distance-set*
z $\in$ *wo.underS a* $\cap$ *extended-distance-set*
t $\in$ *wo.underS a* $\cap$ *extended-distance-set*
for *x y z t*
proof −
define *b* **where** *b = wo.max2* (*wo.max2 x y*) (*wo.max2 z t*)
have *b* $\in$ *wo.underS a* **using** *that b-def* **by** *auto*
have *x* $\in$ *wo.under b y* $\in$ *wo.under b z* $\in$ *wo.under b t* $\in$ *wo.under b* **unfolding** *b-def*
apply (*auto simp add*: *under-def*)
by (*metis UNIV-I metric-space-class.Bonk-Schramm-extension-unfolded-wo-props*(*1*) *mem-Collect-eq under-def well-order-on-Well-order wo.TOTALS wo.max2-iff*)+
then show *?thesis* **using** *that IH-orig* ‹*b* $\in$ *wo.underS a*› *underS-E* **by** *fastforce*
qed

consider *a* $\notin$ *extended-distance-set* | *a* $\in$ *extended-distance-set* **by** *auto*
then show $\forall$ *x y z t. x* $\in$ *wo.under a* $\cap$ *extended-distance-set* $\longrightarrow$
y $\in$ *wo.under a* $\cap$ *extended-distance-set* $\longrightarrow$
z $\in$ *wo.under a* $\cap$ *extended-distance-set* $\longrightarrow$
t $\in$ *wo.under a* $\cap$ *extended-distance-set* $\longrightarrow$
extended-distance x y + extended-distance z t $\leq$ *max* (*extended-distance x z + extended-distance y t*) (*extended-distance x t + extended-distance y z*) + *2* $*$ *deltaG*(*TYPE*(*'a*))
proof (*cases*)

case *1*
then have *wo.under a* $\cap$ *extended-distance-set = wo.underS a* $\cap$ *extended-distance-set*
apply *auto*
apply (*metis mem-Collect-eq underS-I under-def*)
by (*simp add*: *underS-E under-def*)
then show *?thesis* **using** *IH* **by** *auto*
next

case *2*
then have *a*: *extended-distance a a = 0* **unfolding** *metric-space-class.extended-distance-set-def* **by** *auto*
have *main-ineq*: *extended-distance a y + extended-distance z t* $\leq$ *max* (*extended-distance a z + extended-distance y t*) (*extended-distance a t + extended-distance y z*) + *2* $*$ *deltaG*(*TYPE*(*'a*))
if *yzt*: *y* $\in$ *wo.underS a* $\cap$ *extended-distance-set*

$z \in$ *wo.underS a* $\cap$ *extended-distance-set*
$t \in$ *wo.underS a* $\cap$ *extended-distance-set*
for *y z t*
proof (*cases a*)

case *A*: (*basepoint a'*)
then have $y \in$ *range basepoint* **using** *metric-space-class.Bonk-Schramm-extension-unfolded-wo-props*(*2*)
by (*metis yzt*(*1*) *Compl-iff Int-iff range-eqI wo.max2-def wo.max2-underS'*(*2*))
then obtain *y'* **where** *y*: *y* = *basepoint y'* **by** *auto*
have $z \in$ *range basepoint* **using** *metric-space-class.Bonk-Schramm-extension-unfolded-wo-props*(*2*)
A
by (*metis yzt*(*2*) *Compl-iff Int-iff range-eqI wo.max2-def wo.max2-underS'*(*2*))
then obtain *z'* **where** *z*: *z* = *basepoint z'* **by** *auto*
have $t \in$ *range basepoint* **using** *metric-space-class.Bonk-Schramm-extension-unfolded-wo-props*(*2*)
A
by (*metis yzt*(*3*) *Compl-iff Int-iff range-eqI wo.max2-def wo.max2-underS'*(*2*))
then obtain *t'* **where** *t*: *t* = *basepoint t'* **by** *auto*
show *?thesis*
unfolding *y z t A metric-space-class.extended-distance-basepoint*
using *hyperb-quad-ineq UNIV-I* **unfolding** *Gromov-hyperbolic-subset-def*
by *auto*
next

case *C*: (*would-be-Cauchy u*)
then have *u*: *would-be-Cauchy u* $\in$ *extended-distance-set*
u n $\in$ *extended-distance-set* $\cap$ *wo.underS* (*would-be-Cauchy u*) **for**
n
using *metric-space-class.extended-distance-set-Cauchy 2* **by** *auto*
have *lim*: (λn. *extended-distance y* (*u n*)) $\longrightarrow$ (*extended-distance y*
(*would-be-Cauchy u*))
if *y*: $y \in$ *extended-distance-set* **for** *y*
proof −
have *a*: *abs*(*extended-distance y* (*u n*) − *extended-distance y* (*would-be-Cauchy*
u)) $\leq$ *extended-distance* (*u n*) (*would-be-Cauchy u*) **for** *n*
using *u*(*2*)[*of n*] *2 y metric-space-class.extended-distance-triang-ineq*
unfolding *C*
apply (*subst abs-le-iff*) **apply** (*auto simp add*: *algebra-simps*)
by (*metis metric-space-class.extended-distance-symmetric*)
have *b*: (λn. *extended-distance* (*u n*) (*would-be-Cauchy u*)) $\longrightarrow$ *0*
unfolding *C* **apply** (*rule metric-space-class.extended-distance-Cauchy*(*2*))
using *metric-space-class.extended-distance-set-Cauchy*[*of u*] *C 2* **by** *auto*
have (λn. *abs*(*extended-distance y* (*u n*) − *extended-distance y* (*would-be-Cauchy*
u))) $\longrightarrow$ *0*
apply (*rule tendsto-sandwich*[*of* λ-. *0*, *OF* - - - *b*]) **using** *a* **by** *auto*
then show (λn. *extended-distance y* (*u n*)) $\longrightarrow$ *extended-distance y*
(*would-be-Cauchy u*)
using *Lim-null tendsto-rabs-zero-cancel* **by** *blast*
qed
have *max* (*extended-distance* (*u n*) *z* + *extended-distance y t*) (*extended-distance*

$(u\ n)\ t + extended\text{-}distance\ y\ z) + 2 * deltaG(TYPE('a)) - extended\text{-}distance\ (u\ n)\ y - extended\text{-}distance\ z\ t \geq 0$ **for** n
using *IH[of u n y z t] u yzt C* **by** *auto*
moreover have $(\lambda n.\ max\ (extended\text{-}distance\ (u\ n)\ z + extended\text{-}distance\ y\ t)\ (extended\text{-}distance\ (u\ n)\ t + extended\text{-}distance\ y\ z) + 2 * deltaG(TYPE('a)) - extended\text{-}distance\ (u\ n)\ y - extended\text{-}distance\ z\ t)$
$\longrightarrow max\ (extended\text{-}distance\ (would\text{-}be\text{-}Cauchy\ u)\ z + extended\text{-}distance\ y\ t)\ (extended\text{-}distance\ (would\text{-}be\text{-}Cauchy\ u)\ t + extended\text{-}distance\ y\ z) + 2 * deltaG(TYPE('a)) - extended\text{-}distance\ (would\text{-}be\text{-}Cauchy\ u)\ y - extended\text{-}distance\ z\ t$
apply (*auto intro*!: *tendsto-intros*)
using *lim that u* **by** (*auto simp add*: *metric-space-class.extended-distance-symmetric*)
ultimately have *I*: $max\ (extended\text{-}distance\ (would\text{-}be\text{-}Cauchy\ u)\ z + extended\text{-}distance\ y\ t)\ (extended\text{-}distance\ (would\text{-}be\text{-}Cauchy\ u)\ t + extended\text{-}distance\ y\ z) + 2 * deltaG(TYPE('a)) - extended\text{-}distance\ (would\text{-}be\text{-}Cauchy\ u)\ y - extended\text{-}distance\ z\ t \geq 0$
using *LIMSEQ-le-const* **by** *blast*
then show *?thesis* **unfolding** *C* **by** *auto*
next

case *M*: (*middle c d*)
then have *cd*: $c \in extended\text{-}distance\text{-}set \cap wo.underS\ (middle\ c\ d)$
$d \in extended\text{-}distance\text{-}set \cap wo.underS\ (middle\ c\ d)$
using *2 metric-space-class.extended-distance-set-middle'[of c d]* **by** *auto*

have *bdd*: $bdd\text{-}above\ ((\lambda w.\ extended\text{-}distance\ s\ w - max\ (extended\text{-}distance\ c\ w)\ (extended\text{-}distance\ d\ w))\text{`}\ (wo.underS\ (middle\ c\ d) \cap extended\text{-}distance\text{-}set))$
if $s \in extended\text{-}distance\text{-}set$ **for** s
proof (*rule bdd-aboveI2*)
fix w **assume** w: $w \in wo.underS\ (middle\ c\ d) \cap extended\text{-}distance\text{-}set$
have $extended\text{-}distance\ s\ w \leq extended\text{-}distance\ s\ c + extended\text{-}distance\ c\ w$
using *w that metric-space-class.extended-distance-triang-ineq cd* **by** *auto*
also have $... \leq extended\text{-}distance\ s\ c + max\ (extended\text{-}distance\ c\ w)\ (extended\text{-}distance\ d\ w)$
by *auto*
finally show $extended\text{-}distance\ s\ w - max\ (extended\text{-}distance\ c\ w)\ (extended\text{-}distance\ d\ w)$
$\leq extended\text{-}distance\ s\ c$
by *auto*
qed

have *I*: $extended\text{-}distance\ y\ w - max\ (extended\text{-}distance\ c\ w)\ (extended\text{-}distance\ d\ w)$
$\leq max\ (extended\text{-}distance\ y\ z + extended\text{-}distance\ t\ (middle\ c\ d))\ (extended\text{-}distance\ y\ t + extended\text{-}distance\ z\ (middle\ c\ d)) + 2 * deltaG(TYPE('a)) - (extended\text{-}distance\ c\ d)/2 - extended\text{-}distance\ z\ t$
if w: $w \in wo.underS\ (middle\ c\ d) \cap extended\text{-}distance\text{-}set$ **for** w
proof –

have *J*: (*extended-distance c d*)/*2* + *extended-distance s w* − *max* (*extended-distance c w*) (*extended-distance d w*) ≤ *extended-distance s* (*middle c d*)
if *s* ∈ *wo.underS a* ∩ *extended-distance-set* **for** *s*
proof −
have (*extended-distance c d*)/*2* + *extended-distance s w* − *max* (*extended-distance c w*) (*extended-distance d w*)
≤ (*extended-distance c d*)/*2*
+ (*SUP w*∈*wo.underS* (*middle c d*) ∩ *extended-distance-set*. *extended-distance s w* − *max* (*extended-distance c w*) (*extended-distance d w*))
apply *auto* **apply** (*rule cSUP-upper*) **using** *w bdd that* **by** *auto*
also have ... = *extended-distance s* (*middle c d*)
apply (*rule metric-space-class.extended-distance-middle-formula*[*symmetric*]) **using** *that M* **by** *auto*
finally show *?thesis* **by** *simp*
qed
have (*extended-distance c d*)/*2* + *extended-distance y w* − *max* (*extended-distance c w*) (*extended-distance d w*) + *extended-distance z t*
≤ (*extended-distance c d*)/*2* + *max* (*extended-distance y z* + *extended-distance t w*) (*extended-distance y t* + *extended-distance z w*) + *2* ∗ *deltaG*(*TYPE*(*'a*)) − *max* (*extended-distance c w*) (*extended-distance d w*)
using *IH*[*of y w z t*] *w yzt M* **by** (*auto simp add*: *metric-space-class.extended-distance-symmetric*)
also have ... = *max* (*extended-distance y z* + (*extended-distance c d*)/*2* + *extended-distance t w* − *max* (*extended-distance c w*) (*extended-distance d w*))
(*extended-distance y t* + (*extended-distance c d*)/*2* + *extended-distance z w* − *max* (*extended-distance c w*) (*extended-distance d w*))
+ *2* ∗ *deltaG*(*TYPE*(*'a*))
by *auto*
also have ... ≤ *max* (*extended-distance y z* + *extended-distance t* (*middle c d*)) (*extended-distance y t* + *extended-distance z* (*middle c d*)) + *2* ∗ *deltaG*(*TYPE*(*'a*))
using *J*[*OF yzt*(*3*)] *J*[*OF yzt*(*2*)] **by** *auto*
finally show *?thesis* **by** *simp*
qed
have ∗: (*SUP w*∈*wo.underS* (*middle c d*) ∩ *extended-distance-set*. *extended-distance y w* − *max* (*extended-distance c w*) (*extended-distance d w*)) ≤
max (*extended-distance y z* + *extended-distance t* (*middle c d*)) (*extended-distance y t* + *extended-distance z* (*middle c d*)) + *2* ∗ *deltaG*(*TYPE*(*'a*))
− (*extended-distance c d*)/*2* − *extended-distance z t*
apply (*rule cSUP-least*) **using** *yzt*(*1*) *M I* **by** *auto*
have *extended-distance y* (*middle c d*) + *extended-distance z t*
= (*extended-distance c d*)/*2* + (*SUP w*∈*wo.underS* (*middle c d*) ∩ *extended-distance-set*. *extended-distance y w* − *max* (*extended-distance c w*) (*extended-distance d w*)) + *extended-distance z t*
apply *simp* **apply** (*rule metric-space-class.extended-distance-middle-formula*) **using** *yzt*(*1*) *M* **by** *auto*
also have ... ≤ *max* (*extended-distance y z* + *extended-distance t* (*middle c d*)) (*extended-distance y t* + *extended-distance z* (*middle c d*)) + *2* ∗ *deltaG*(*TYPE*(*'a*))
using ∗ **by** *simp*

finally show *extended-distance a y* + *extended-distance z t*
$\leq$ *max* (*extended-distance a z* + *extended-distance y t*) (*extended-distance a t* + *extended-distance y z*) + *2* $*$ *deltaG*(*TYPE*('*a*))
unfolding *M* **by** (*auto simp add*: *metric-space-class.extended-distance-symmetric*)
qed

show *?thesis*
proof (*auto*)
fix *x y z t* **assume** *H*: *x* $\in$ *wo.under a x* $\in$ *extended-distance-set*
y $\in$ *wo.under a y* $\in$ *extended-distance-set*
z $\in$ *wo.under a z* $\in$ *extended-distance-set*
t $\in$ *wo.under a t* $\in$ *extended-distance-set*
have $*$: ((*x* $\in$ *wo.underS a* $\cap$ *extended-distance-set*) $\vee$ (*x* = *a*))
$\wedge$ ((*y* $\in$ *wo.underS a* $\cap$ *extended-distance-set*) $\vee$ (*y* = *a*))
$\wedge$ ((*z* $\in$ *wo.underS a* $\cap$ *extended-distance-set*) $\vee$ (*z* = *a*))
$\wedge$ ((*t* $\in$ *wo.underS a* $\cap$ *extended-distance-set*) $\vee$ (*t* = *a*))
using *H* **by** (*simp add*: *underS-def under-def*)
have *d*: *2* $*$ *deltaG*(*TYPE*('*a*)) $\geq$ *0* **by** *auto*
show *extended-distance x y* + *extended-distance z t* $\leq$ *max* (*extended-distance x z* + *extended-distance y t*) (*extended-distance x t* + *extended-distance y z*) + *2* $*$ *deltaG*(*TYPE*('*a*))
using $*$ **apply** (*auto simp add*: *metric-space-class.extended-distance-symmetric a*)
using *IH*[*of x y z t*] **apply** (*simp add*: *metric-space-class.extended-distance-symmetric*)
using *main-ineq*[*of z x y*] **apply** (*simp add*: *metric-space-class.extended-distance-symmetric*)
using *main-ineq*[*of t x y*] **apply** (*simp add*: *metric-space-class.extended-distance-symmetric*)
using *2 metric-space-class.extended-distance-triang-ineq*[*of x a y*] *H* **apply** (*simp add*: *metric-space-class.extended-distance-symmetric*) **using** *d* **apply** *linarith*
using *main-ineq*[*of x z t*] **apply** (*simp add*: *metric-space-class.extended-distance-symmetric*)
using *d* **apply** *linarith*
using *d* **apply** *linarith*
using *main-ineq*[*of y z t*] **apply** (*simp add*: *metric-space-class.extended-distance-symmetric*)
using *d* **apply** *linarith*
using *d* **apply** *linarith*
using *2 metric-space-class.extended-distance-triang-ineq*[*of t a z*] *H* **apply** (*simp add*: *metric-space-class.extended-distance-symmetric*) **using** *d* **apply** *linarith*
done
qed
qed
qed
define *b* **where** *b* = *wo.max2* (*wo.max2 x y*) (*wo.max2 z t*)
have *x* $\in$ *wo.under b y* $\in$ *wo.under b z* $\in$ *wo.under b t* $\in$ *wo.under b* **unfolding** *b-def*
apply (*auto simp add*: *under-def*)
by (*metis UNIV-I metric-space-class.Bonk-Schramm-extension-unfolded-wo-props*(*1*) *mem-Collect-eq under-def well-order-on-Well-order wo.TOTALS wo.max2-iff*)+

then show *?thesis* **using** *ineq-rec*[*of b*] *assms* **by** *auto*
qed

lemma (**in** *Gromov-hyperbolic-space*) *Bonk-Schramm-extension-hyperbolic*:
Gromov-hyperbolic-subset (*deltaG*(*TYPE*('*a*))) (*UNIV*::('*a Bonk-Schramm-extension*) *set*)
apply (*rule Gromov-hyperbolic-subsetI*, *simp*, *transfer fixing*: *deltaG*)
using *metric-space-class.extended-distance-set-def Bonk-Schramm-extension-unfolded-hyperbolic*
by *auto*

instantiation *Bonk-Schramm-extension* :: (*Gromov-hyperbolic-space*) *Gromov-hyperbolic-space-geodesic*
begin
definition *deltaG-Bonk-Schramm-extension*::('*a Bonk-Schramm-extension*) *itself* ⇒ *real* **where**
deltaG-Bonk-Schramm-extension - = deltaG(*TYPE*('*a*))

instance apply *standard*
unfolding *deltaG-Bonk-Schramm-extension-def* **using** *Bonk-Schramm-extension-hyperbolic*
by *auto*
end

Finally, it follows that the Bonk Schramm extension of a 0-hyperbolic space (in which it embeds isometrically) is a metric tree or, equivalently, a geodesic 0-hyperbolic space (the equivalence is proved at the end of `Geodesic_Spaces.thy`).

instance *Bonk-Schramm-extension* :: (*Gromov-hyperbolic-space-0*) *Gromov-hyperbolic-space-0-geodesic*
by (*standard*, *simp add*: *deltaG-Bonk-Schramm-extension-def delta0*)

It then follows that it is also a metric tree, from what we have already proved. We write explicitly for definiteness.

instance *Bonk-Schramm-extension* :: (*Gromov-hyperbolic-space-0*) *metric-tree*
by *standard*

13.2 Applications

We deduce that we can extend results on Gromov-hyperbolic spaces without the geodesicity assumption, even if it is used in the proofs. These results are given for illustrative purpose mainly, as one works most often in geodesic spaces anyway.

The following results have already been proved in hyperbolic geodesic spaces. The same results follow in a general hyperbolic space, as everything is invariant under isometries and can thus be pulled from the corresponding result in the Bonk Schramm extension. The straightforward proofs only express this invariance under isometries of all the properties under consideration.

proposition (**in** *Gromov-hyperbolic-space*) *lipschitz-path-close-to-geodesic'*:
fixes *c*::*real* ⇒ '*a*
assumes *lipschitz-on M* {*A*..*B*} *c*
geodesic-segment-between G (*c A*) (*c B*)

$x \in G$
shows *infdist x* (c‘{$A..B$}) $\leq$ ($4/ln\ 2$) $*$ *deltaG*(*TYPE*('*a*)) $*$ *max 0* (*ln* ($B-A$))
$+$ *M*
proof $-$
interpret *BS*: *Gromov-hyperbolic-space-geodesic dist*::('*a Bonk-Schramm-extension*
$\Rightarrow$ '*a Bonk-Schramm-extension* $\Rightarrow$ *real*) *uniformity open* (λ-. *deltaG*(*TYPE*('*a*)))
apply *standard* **using** *Bonk-Schramm-extension-hyperbolic* **by** *auto*

have *infdist x* (c‘{$A..B$}) = *infdist* (*to-Bonk-Schramm-extension x*) ((*to-Bonk-Schramm-extension*
o c)‘{$A..B$})
unfolding *image-comp*[*symmetric*] **apply** (*rule isometry-preserves-infdist*[*symmetric*,
of UNIV])
using *to-Bonk-Schramm-extension-isometry* **by** *auto*
also have ... $\leq$ ($4/ln\ 2$) $*$ *deltaG*(*TYPE*(('*a*))) $*$ *max 0* (*ln* ($B-A$)) + ($1*M$)
apply (*rule BS.lipschitz-path-close-to-geodesic*[*of - - - - to-Bonk-Schramm-extension*‘*G*])
apply (*rule lipschitz-on-compose*)
using *assms* **apply** *simp*
apply (*meson UNIV-I isometry-on-lipschitz lipschitz-on-def to-Bonk-Schramm-extension-isometry*)
unfolding *comp-def* **apply** (*rule isometry-preserves-geodesic-segment-between*[*of*
UNIV])
using *assms to-Bonk-Schramm-extension-isometry* **by** *auto*
finally show *?thesis* **by** *auto*
qed

theorem (**in** *Gromov-hyperbolic-space*) *Morse-Gromov-theorem'*:
fixes *f*::*real* $\Rightarrow$ '*a*
assumes *lambda C*$-$*quasi-isometry-on* {*a..b*} *f*
geodesic-segment-between G (*f a*) (*f b*)
shows *hausdorff-distance* (*f*‘{*a..b*}) *G* $\leq$ *92* $*$ *lambda*^{2} $*$ (*C* + *deltaG*(*TYPE*('*a*)))
proof $-$
interpret *BS*: *Gromov-hyperbolic-space-geodesic dist*::('*a Bonk-Schramm-extension*
$\Rightarrow$ '*a Bonk-Schramm-extension* $\Rightarrow$ *real*) *uniformity open* (λ-. *deltaG*(*TYPE*('*a*)))
apply *standard* **using** *Bonk-Schramm-extension-hyperbolic* **by** *auto*
have *hausdorff-distance* (*f*‘{*a..b*}) (*G*) = *hausdorff-distance* ((*to-Bonk-Schramm-extension*
o f)‘{*a..b*}) ((*to-Bonk-Schramm-extension*)‘*G*)
unfolding *image-comp*[*symmetric*] **apply** (*rule isometry-preserves-hausdorff-distance*[*symmetric*,
of UNIV])
using *to-Bonk-Schramm-extension-isometry* **by** *auto*
also have ... $\leq$ *92* $*$ (*lambda*$*$*1*)^*2* $*$ ((*C*$*$*1*+*0*) + *deltaG*(*TYPE*('*a*)))
apply (*intro BS.Morse-Gromov-theorem quasi-isometry-on-compose*[**where** *Y*
= *UNIV*])
using *assms isometry-quasi-isometry-on to-Bonk-Schramm-extension-isometry*
apply *auto*
using *isometry-preserves-geodesic-segment-between* **by** *blast*
finally show *?thesis* **by** *simp*
qed

theorem (**in** *Gromov-hyperbolic-space*) *Morse-Gromov-theorem2'*:
fixes *c d*::*real* $\Rightarrow$ '*a*

assumes *lambda C−quasi-isometry-on* {*A..B*} *c*
lambda C−quasi-isometry-on {*A..B*} *d*
c A = *d A c B* = *d B*
shows *hausdorff-distance* (*c*‘{*A..B*}) (*d*‘{*A..B*}) ≤ *184* ∗ *lambda*^*2* ∗ (*C* + *deltaG*(*TYPE*(′*a*)))
proof −
interpret *BS*: *Gromov-hyperbolic-space-geodesic dist*::(′*a Bonk-Schramm-extension* ⇒ ′*a Bonk-Schramm-extension* ⇒ *real*) *uniformity open* (λ-. *deltaG*(*TYPE*(′*a*)))
apply *standard* **using** *Bonk-Schramm-extension-hyperbolic* **by** *auto*
have *hausdorff-distance* (*c*‘{*A..B*}) (*d*‘{*A..B*}) = *hausdorff-distance* ((*to-Bonk-Schramm-extension o c*)‘{*A..B*}) ((*to-Bonk-Schramm-extension o d*)‘{*A..B*})
unfolding *image-comp*[*symmetric*] **apply** (*rule isometry-preserves-hausdorff-distance*[*symmetric*, *of UNIV*])
using *to-Bonk-Schramm-extension-isometry* **by** *auto*
also have ... ≤ *184* ∗ (*lambda*∗*1*)^*2* ∗ ((*C*∗*1*+*0*) + *deltaG*(*TYPE*(′*a*)))
apply (*intro BS.Morse-Gromov-theorem2 quasi-isometry-on-compose*[**where** *Y* = *UNIV*])
using *assms isometry-quasi-isometry-on to-Bonk-Schramm-extension-isometry* **by** *auto*
finally show *?thesis* **by** *simp*
qed

lemma *Gromov-hyperbolic-invariant-under-quasi-isometry-explicit*′:
fixes *f*::′*a*::*geodesic-space* ⇒ ′*b*::*Gromov-hyperbolic-space*
assumes *lambda C−quasi-isometry f*
shows *Gromov-hyperbolic-subset* (*752* ∗ *lambda*^*3* ∗ (*C* + *deltaG*(*TYPE*(′*b*)))) (*UNIV*::(′*a set*))
proof −
interpret *BS*: *Gromov-hyperbolic-space-geodesic dist*::(′*b Bonk-Schramm-extension* ⇒ ′*b Bonk-Schramm-extension* ⇒ *real*) *uniformity open* (λ-. *deltaG*(*TYPE*(′*b*)))
apply *standard* **using** *Bonk-Schramm-extension-hyperbolic* **by** *auto*
have *A*: (*lambda* ∗ *1*) (*C* ∗ *1* + *0*)−*quasi-isometry-on UNIV* (*to-Bonk-Schramm-extension o f*)
by (*rule quasi-isometry-on-compose*[*OF assms*, *of* - - *UNIV*])
(*auto simp add*: *isometry-quasi-isometry-on*[*OF to-Bonk-Schramm-extension-isometry*])
have ∗: *deltaG*(*TYPE*(′*b*)) = *deltaG*(*TYPE*(′*b Bonk-Schramm-extension*))
by (*simp add*: *deltaG-Bonk-Schramm-extension-def*)
show *?thesis*
unfolding ∗
apply (*rule Gromov-hyperbolic-invariant-under-quasi-isometry-explicit*[*of* - - *to-Bonk-Schramm-extension o f*])
using *A* **by** *auto*
qed

theorem *Gromov-hyperbolic-invariant-under-quasi-isometry*′:
assumes *quasi-isometric* (*UNIV*::(′*a*::*geodesic-space*) *set*) (*UNIV*::(′*b*::*Gromov-hyperbolic-space*) *set*)
shows ∃ *delta*. *Gromov-hyperbolic-subset delta* (*UNIV*::′*a set*)
proof −

obtain *C lambda f* **where** *f*: *lambda C−quasi-isometry-between* (*UNIV*::*'a set*) (*UNIV*::*'b set*) *f*
using *assms* **unfolding** *quasi-isometric-def* **by** *auto*
show *?thesis* **using** *Gromov-hyperbolic-invariant-under-quasi-isometry-explicit'*[*OF quasi-isometry-betweenD*(*1*)[*OF f*]] **by** *blast*
qed

end

theory *Gromov-Boundary*
imports *Gromov-Hyperbolicity Eexp-Eln*
begin

14 Constructing a distance from a quasi-distance

Below, we will construct a distance on the Gromov completion of a hyperbolic space. The geometrical object that arises naturally is almost a distance, but it does not satisfy the triangular inequality. There is a general process to turn such a quasi-distance into a genuine distance, as follows: define the new distance $\tilde{d}(x,y)$ to be the infimum of $d(x,u_1)+d(u_1,u_2)+\cdots+d(u_{n-1},x)$ over all sequences of points (of any length) connecting x to y. It is clear that it satisfies the triangular inequality, is symmetric, and $\tilde{d}(x,y) \leq d(x,y)$. What is not clear, however, is if $\tilde{d}(x,y)$ can be zero if $x \neq y$, or more generally how one can bound $\tilde{d}$ from below. The main point of this contruction is that, if d satisfies the inequality $d(x,z) \leq \sqrt{2}\max(d(x,y),d(y,z))$, then one has $\tilde{d}(x,y) \geq d(x,y)/2$ (and in particular $\tilde{d}$ defines the same topology, the same set of Lipschitz functions, and so on, as d).

This statement can be found in [Bourbaki, topologie generale, chapitre 10] or in [Ghys-de la Harpe] for instance. We follow their proof.

definition *turn-into-distance*::(*'a* ⇒ *'a* ⇒ *real*) ⇒ (*'a* ⇒ *'a* ⇒ *real*)
where *turn-into-distance f x y* = *Inf* {(∑ *i* ∈ {*0*..<*n*}. *f* (*u i*) (*u* (*Suc i*)))| *u* (*n*::*nat*). *u 0* = *x* ∧ *u n* = *y*}

locale *Turn-into-distance* =
fixes *f*::*'a* ⇒ *'a* ⇒ *real*
assumes *nonneg*: *f x y* ≥ *0*
and *sym*: *f x y* = *f y x*
and *self-zero*: *f x x* = *0*
and *weak-triangle*: *f x z* ≤ *sqrt 2* ∗ *max* (*f x y*) (*f y z*)
begin

The two lemmas below are useful when dealing with Inf results, as they always require the set under consideration to be non-empty and bounded from below.

lemma *bdd-below* [*simp*]:

bdd-below $\{(\sum\ i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i)))|\ u\ (n::nat).\ u\ 0 = x \wedge u\ n = y\}$
apply (*rule bdd-belowI*[*of - 0*]) **using** *nonneg* **by** (*auto simp add*: *sum-nonneg*)

lemma *nonempty*:
$\{\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))\ |u\ n.\ u\ 0 = x \wedge u\ n = y\} \neq \{\}$
proof −
define *u*::*nat* ⇒ ′*a* **where** $u = (\lambda n.\ if\ n = 0\ then\ x\ else\ y)$
define *n*::*nat* **where** $n = 1$
have $u\ 0 = x \wedge u\ n = y$ **unfolding** *u-def n-def* **by** *auto*
then have $(\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) \in \{\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))\ |u\ n.\ u\ 0 = x \wedge u\ n = y\}$
by *auto*
then show *?thesis* **by** *auto*
qed

We can now prove that `turn_into_distance f` satisfies all the properties of a distance. First, it is nonnegative.

lemma *TID-nonneg*:
turn-into-distance f x y ≥ 0
unfolding *turn-into-distance-def* **apply** (*rule cInf-greatest*[*OF nonempty*])
using *nonneg* **by** (*auto simp add*: *sum-nonneg*)

For the symmetry, we use the symmetry of f, and go backwards along a chain of points, replacing a sequence from x to y with a sequence from y to x.

lemma *TID-sym*:
turn-into-distance f x y = *turn-into-distance f y x*
proof −
have *turn-into-distance f x y* $\leq$ *Inf* $\{(\sum\ i \in \{0..<n\}.\ f\ (u\ i)\ (u\ (Suc\ i)))|\ u\ (n::nat).\ u\ 0 = y \wedge u\ n = x\}$ **for** *x y*
proof (*rule cInf-greatest*[*OF nonempty*], *auto*)
fix *u*::*nat* ⇒ ′*a* **and** *n* **assume** *U*: $y = u\ 0\ x = u\ n$
define *v*::*nat* ⇒′*a* **where** $v = (\lambda i.\ u\ (n-i))$
have *V*: $v\ 0 = x\ v\ n = y$ **unfolding** *v-def* **using** *U* **by** *auto*

have $(\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) = (\sum i = 0..<n.\ (\lambda i.\ f\ (u\ i)\ (u\ (Suc\ i)))\ (n-1-i))$
apply (*rule sum.reindex-bij-betw*[*symmetric*])
by (*rule bij-betw-byWitness*[*of -* $\lambda i.\ n-1-i$], *auto*)
also have ... $= (\sum\ i = 0..<n.\ f\ (v\ (Suc\ i))\ (v\ i))$
apply (*rule sum.cong*) **unfolding** *v-def* **by** (*auto simp add*: *Suc-diff-Suc*)
also have ... $= (\sum\ i = 0..<n.\ f\ (v\ i)\ (v\ (Suc\ i)))$
using *sym* **by** *auto*
finally have $(\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) = (\sum\ i = 0..<n.\ f\ (v\ i)\ (v\ (Suc\ i)))$
by *simp*

moreover have *turn-into-distance f x y* $\leq (\sum\ i = 0..<n.\ f\ (v\ i)\ (v\ (Suc\ i)))$
unfolding *turn-into-distance-def* **apply** (*rule cInf-lower*)

using *V* **by** *auto*
finally show *turn-into-distance f* $(u\ n)$ $(u\ 0) \leq (\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i)))$
using *U* **by** *auto*
qed
then have $*$: *turn-into-distance f x y* $\leq$ *turn-into-distance f y x* **for** *x y*
unfolding *turn-into-distance-def* **by** *auto*
show *?thesis* **using** $*[of\ x\ y]$ $*[of\ y\ x]$ **by** *simp*
qed

There is a trivial upper bound by f, using the single chain x, y.

lemma *upper*:
turn-into-distance f x y $\leq f\ x\ y$
unfolding *turn-into-distance-def* **proof** (*rule cInf-lower*, *auto*)
define $u::nat \Rightarrow 'a$ **where** $u = (\lambda n.\ if\ n = 0\ then\ x\ else\ y)$
define $n::nat$ **where** $n = 1$
have $u\ 0 = x \wedge u\ n = y \wedge f\ x\ y = (\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i)))$ **unfolding** *u-def n-def* **by** *auto*
then show $\exists\, u\ n.\ f\ x\ y = (\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) \wedge u\ 0 = x \wedge u\ n = y$
by *auto*
qed

The new distance vanishes on a pair of equal points, as this is already the case for f.

lemma *TID-self-zero*:
turn-into-distance f x x $= 0$
using *upper*[*of x x*] *TID-nonneg*[*of x x*] *self-zero*[*of x*] **by** *auto*

For the triangular inequality, we concatenate a sequence from x to y almost realizing the infimum, and a sequence from y to z almost realizing the infimum, to obtain a sequence from x to z along which the sums of f is almost bounded by `turn_into_distance f x y + turn_into_distance f y z`.

lemma *triangle*:
turn-into-distance f x z $\leq$ *turn-into-distance f x y* $+$ *turn-into-distance f y z*
proof $-$
have *turn-into-distance f x z* $\leq$ *turn-into-distance f x y* $+$ *turn-into-distance f y z* $+ e$ **if** $e > 0$ **for** e
proof $-$
have $Inf\ \{(\sum\ i \in \{0..<n\}.\ f\ (u\ i)\ (u\ (Suc\ i)))|\ u\ (n::nat).\ u\ 0 = x \wedge u\ n = y\}$ $<$ *turn-into-distance f x y* $+ e/2$
unfolding *turn-into-distance-def* **using** $\langle e > 0\rangle$ **by** *auto*
then have $\exists\, a \in \{(\sum\ i \in \{0..<n\}.\ f\ (u\ i)\ (u\ (Suc\ i)))|\ u\ (n::nat).\ u\ 0 = x \wedge u\ n = y\}.\ a <$ *turn-into-distance f x y* $+ e/2$
by (*rule cInf-lessD*[*OF nonempty*])
then obtain *u n* **where** *U*: $u\ 0 = x$ $u\ n = y$ $(\sum\ i \in \{0..<n\}.\ f\ (u\ i)\ (u\ (Suc\ i)))$ $<$ *turn-into-distance f x y* $+ e/2$
by *auto*

have *Inf* $\{(\sum\ i \in \{0..<m\}.\ f\ (v\ i)\ (v\ (Suc\ i)))|\ v\ (m::nat).\ v\ 0 = y \wedge v\ m = z\} <$ *turn-into-distance f y z* $+\ e/2$
unfolding *turn-into-distance-def* **using** ‹$e > 0$› **by** *auto*
then have $\exists\, a \in \{(\sum\ i \in \{0..<m\}.\ f\ (v\ i)\ (v\ (Suc\ i)))|\ v\ (m::nat).\ v\ 0 = y \wedge v\ m = z\}.\ a <$ *turn-into-distance f y z* $+\ e/2$
by (*rule cInf-lessD*[*OF nonempty*])
then obtain *v m* **where** *V*: $v\ 0 = y\ v\ m = z\ (\sum\ i \in \{0..<m\}.\ f\ (v\ i)\ (v\ (Suc\ i))) <$ *turn-into-distance f y z* $+\ e/2$
by *auto*

define *w* **where** $w = (\lambda i.\ \mathit{if}\ i < n\ \mathit{then}\ u\ i\ \mathit{else}\ v\ (i{-}n))$
have $*$: $w\ 0 = x\ w\ (n{+}m) = z$
unfolding *w-def* **using** *U V* **by** *auto*
have *turn-into-distance f x z* $\leq (\sum i = 0..<n{+}m.\ f\ (w\ i)\ (w\ (Suc\ i)))$
unfolding *turn-into-distance-def* **apply** (*rule cInf-lower*) **using** $*$ **by** *auto*
also have ... $= (\sum i = 0..<n.\ f\ (w\ i)\ (w\ (Suc\ i))) + (\sum i = n..<n{+}m.\ f\ (w\ i)\ (w\ (Suc\ i)))$
by (*simp add*: *sum.atLeastLessThan-concat*)
also have ... $= (\sum i = 0..<n.\ f\ (w\ i)\ (w\ (Suc\ i))) + (\sum i = 0..<m.\ f\ (w\ (i{+}n))\ (w\ (Suc\ (i{+}n))))$
by (*auto intro*!: *sum.reindex-bij-betw*[*symmetric*] *bij-betw-byWitness*[*of* - $\lambda i.\ i{-}n$])
also have ... $= (\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) + (\sum i = 0..<m.\ f\ (v\ i)\ (v\ (Suc\ i)))$
unfolding *w-def* **apply** (*auto intro*!: *sum.cong*)
using *U*(*2*) *V*(*1*) *Suc-lessI* **by** *fastforce*
also have ... $<$ *turn-into-distance f x y* $+\ e/2\ +$ *turn-into-distance f y z* $+\ e/2$
using *U*(*3*) *V*(*3*) **by** *auto*
finally show *?thesis* **by** *auto*
qed
then show *?thesis*
using *field-le-epsilon* **by** *blast*
qed

Now comes the only nontrivial statement of the construction, the fact that the new distance is bounded from below by $f/2$.

Here is the mathematical proof. We show by induction that all chains from x to y satisfy this bound. Assume this is done for all chains of length $< n$, we do it for a chain of length n. Write $S = \sum f(u_i, u_{i+1})$ for the sum along the chain. Introduce p the last index where the sum is $\leq S/2$. Then the sum from 0 to p is $\leq S/2$, and the sum from $p+1$ to n is also $\leq S/2$ (by maximality of p). The induction assumption gives that $f(x, u_p)$ is bounded by twice the sum from 0 to p, which is at most S. Same thing for $f(u_{p+1}, y)$. With the weird triangle inequality applied two times, we get $f(x, y) \leq 2\max(f(x, u_p), f(u_p, u_{p+1}), f(u_{p+1}, y)) \leq 2S$, as claimed.

The formalization presents no difficulty.

lemma *lower*:
$f\ x\ y \leq 2 * turn\text{-}into\text{-}distance\ f\ x\ y$
proof −
have *I*: $f\ (u\ 0)\ (u\ n) \leq (\sum\ i \in \{0..<n\}.\ f\ (u\ i)\ (u\ (Suc\ i))) * 2$ **for** *n u*
proof (*induction n arbitrary*: *u rule*: *less-induct*)
case (*less n*)
show $f\ (u\ 0)\ (u\ n) \leq (\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) * 2$
proof (*cases* $n = 0$)
case *True*
then have $f\ (u\ 0)\ (u\ n) = 0$ **using** *self-zero* **by** *auto*
then show *?thesis* **using** *True* **by** *auto*
next
case *False*
then have $n > 0$ **by** *auto*
define *S* **where** $S = (\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i)))$
have $S \geq 0$ **unfolding** *S-def* **using** *nonneg* **by** (*auto simp add*: *sum-nonneg*)
have $\exists p.\ p < n \wedge (\sum i = 0..<p.\ f\ (u\ i)\ (u\ (Suc\ i))) \leq S/2 \wedge (\sum i = Suc\ p..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) \leq S/2$
proof (*cases* $S = 0$)
case *True*
have $(\sum i = Suc\ 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) = (\sum i = 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) - f(u\ 0)\ (u\ (Suc\ 0))$
using *sum.atLeast-Suc-lessThan*[*OF* ‹$n > 0$›, *of* $\lambda i.\ f\ (u\ i)\ (u\ (Suc\ i))$] **by** *simp*
also have ... $\leq S/2$ **using** *True S-def nonneg* **by** *auto*
finally have $0 < n \wedge (\sum i = 0..<0.\ f\ (u\ i)\ (u\ (Suc\ i))) \leq S/2 \wedge (\sum i = Suc\ 0..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) \leq S/2$
using ‹$n > 0$› ‹$S = 0$› **by** *auto*
then show *?thesis* **by** *auto*
next
case *False*
then have $S > 0$ **using** ‹$S \geq 0$› **by** *simp*
define *A* **where** $A = \{q.\ q \leq n \wedge (\sum i = 0..<q.\ f\ (u\ i)\ (u\ (Suc\ i))) \leq S/2\}$
have $0 \in A$ **unfolding** *A-def* **using** ‹$S > 0$› ‹$n > 0$› **by** *auto*
have $n \notin A$ **unfolding** *A-def* **using** ‹$S > 0$› **unfolding** *S-def* **by** *auto*
define *p* **where** $p = Max\ A$
have $p \in A$ **unfolding** *p-def* **apply** (*rule Max-in*) **using** ‹$0 \in A$› **unfolding** *A-def* **by** *auto*
then have *L*: $p \leq n\ (\sum i = 0..<p.\ f\ (u\ i)\ (u\ (Suc\ i))) \leq S/2$ **unfolding** *A-def* **by** *auto*
then have $p < n$ **using** ‹$n \notin A$› ‹$p \in A$› *le-neq-trans* **by** *blast*
have $Suc\ p \notin A$ **unfolding** *p-def*
by (*metis* (*no-types*, *lifting*) *A-def Max-ge Suc-n-not-le-n infinite-nat-iff-unbounded mem-Collect-eq not-le p-def*)
then have $*$: $(\sum i = 0..<Suc\ p.\ f\ (u\ i)\ (u\ (Suc\ i))) > S/2$
unfolding *A-def* **using** ‹$p < n$› **by** *auto*
have $(\sum\ i = Suc\ p..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) = S - (\sum i = 0..<Suc\ p.\ f\ (u\ i)\ (u\ (Suc\ i)))$

unfolding *S-def* **using** ‹$p < n$› **by** (*metis* (*full-types*) *Suc-le-eq sum-diff-nat-ivl zero-le*)
also have ... $\leq S/2$ **using** $*$ **by** *auto*
finally have $p < n \wedge (\sum i = 0..<p.\ f\ (u\ i)\ (u\ (Suc\ i))) \leq S/2 \wedge (\sum i = Suc\ p..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) \leq S/2$
using ‹$p < n$› $L(2)$ **by** *auto*
then show *?thesis* **by** *auto*
qed
then obtain p **where** P: $p < n\ (\sum i = 0..<p.\ f\ (u\ i)\ (u\ (Suc\ i))) \leq S/2\ (\sum i = Suc\ p..<n.\ f\ (u\ i)\ (u\ (Suc\ i))) \leq S/2$
by *auto*
have $f\ (u\ 0)\ (u\ p) \leq (\sum i = 0..<p.\ f\ (u\ i)\ (u\ (Suc\ i))) * 2$
apply (*rule less.IH*) **using** ‹$p < n$› **by** *auto*
then have A: $f\ (u\ 0)\ (u\ p) \leq S$ **using** $P(2)$ **by** *auto*
have B: $f\ (u\ p)\ (u\ (Suc\ p)) \leq S$
apply (*rule sum-nonneg-leq-bound*[*of* $\{0..<n\}$ $\lambda i.\ f\ (u\ i)\ (u\ (Suc\ i))$])
using *nonneg S-def* ‹$p < n$› **by** *auto*
have $f\ (u\ (0 + Suc\ p))\ (u\ ((n-Suc\ p) + Suc\ p)) \leq (\sum i = 0..<n-Suc\ p.\ f\ (u\ (i + Suc\ p))\ (u\ (Suc\ i + Suc\ p))) * 2$
apply (*rule less.IH*) **using** ‹$n > 0$› **by** *auto*
also have ... $= 2 * (\sum i = Suc\ p..<n.\ f\ (u\ i)\ (u\ (Suc\ i)))$
by (*auto intro*!: *sum.reindex-bij-betw bij-betw-byWitness*[*of* - $\lambda i.\ i - Suc\ p$])
also have ... $\leq S$ **using** $P(3)$ **by** *simp*
finally have C: $f\ (u\ (Suc\ p))\ (u\ n) \leq S$
using ‹$p < n$› **by** *auto*

have $f\ (u\ 0)\ (u\ n) \leq sqrt\ 2 * max\ (f\ (u\ 0)\ (u\ p))\ (f\ (u\ p)\ (u\ n))$
using *weak-triangle* **by** *simp*
also have ... $\leq sqrt\ 2 * max\ (f\ (u\ 0)\ (u\ p))\ (sqrt\ 2 * max\ (f\ (u\ p)\ (u\ (Suc\ p)))\ (f\ (u\ (Suc\ p))\ (u\ n)))$
using *weak-triangle* **by** *simp* (*meson max.cobounded2 order-trans*)
also have ... $\leq sqrt\ 2 * max\ S\ (sqrt\ 2 * max\ S\ S)$
using $A\ B\ C$ **by** *auto* (*simp add*: *le-max-iff-disj*)
also have ... $\leq sqrt\ 2 * max\ (sqrt\ 2 * S)\ (sqrt\ 2 * max\ S\ S)$
apply (*intro mult-left-mono max.mono*) **using** ‹$S \geq 0$› *less-eq-real-def* **by** *auto*
also have ... $= 2 * S$
by *auto*
finally show *?thesis*
unfolding *S-def* **by** *simp*
qed
qed
have $f\ x\ y/2 \leq turn\text{-}into\text{-}distance\ f\ x\ y$
unfolding *turn-into-distance-def* **by** (*rule cInf-greatest*[*OF nonempty*], *auto simp add*: *I*)
then show *?thesis* **by** *simp*
qed

end

15 The Gromov completion of a hyperbolic space

15.1 The Gromov boundary as a set

A sequence in a Gromov hyperbolic space converges to a point in the boundary if the Gromov product $(u_n, u_m)_e$ tends to infinity when $m, n \to_i nfty$. The point at infinity is defined as the equivalence class of such sequences, for the relation $u \sim v$ iff $(u_n, v_n)_e \to \infty$ (or, equivalently, $(u_n, v_m)_e \to \infty$ when $m, n \to \infty$, or one could also change basepoints). Hence, the Gromov boundary is naturally defined as a quotient type. There is a difficulty: it can be empty in general, hence defining it as a type is not always possible. One could introduce a new typeclass of Gromov hyperbolic spaces for which the boundary is not empty (unboundedness is not enough, think of infinitely many segments $[0, n]$ all joined at 0), and then only define the boundary of such spaces. However, this is tedious. Rather, we work with the Gromov completion (containing the space and its boundary), this is always not empty. The price to pay is that, in the definition of the completion, we have to distinguish between sequences converging to the boundary and sequences converging inside the space. This is more natural to proceed in this way as the interesting features of the boundary come from the fact that its sits at infinity of the initial space, so their relations (and the topology of $X \cup \partial X$) are central.

definition *Gromov-converging-at-boundary*::(*nat* ⇒ ('*a*::*Gromov-hyperbolic-space*)) ⇒ *bool*
  where *Gromov-converging-at-boundary u* = (∀ *a*. ∀ (*M*::*real*). ∃ *N*. ∀ *n* ≥ *N*. ∀ *m* ≥ *N*. *Gromov-product-at a* (*u m*) (*u n*) ≥ *M*)

lemma *Gromov-converging-at-boundaryI*:
  assumes ⋀*M*. ∃ *N*. ∀ *n* ≥ *N*. ∀ *m* ≥ *N*. *Gromov-product-at a* (*u m*) (*u n*) ≥ *M*
  shows *Gromov-converging-at-boundary u*
unfolding *Gromov-converging-at-boundary-def* **proof** (*auto*)
  fix *b*::'*a* **and** *M*::*real*
  obtain *N* **where** ∗: ⋀*m n*. *n* ≥ *N* ⟹ *m* ≥ *N* ⟹ *Gromov-product-at a* (*u m*) (*u n*) ≥ *M* + *dist a b*
    using *assms*[*of M* + *dist a b*] **by** *auto*
  have *Gromov-product-at b* (*u m*) (*u n*) ≥ *M* **if** *m* ≥ *N n* ≥ *N* **for** *m n*
      using ∗[*OF that*] *Gromov-product-at-diff1*[*of a u m u n b*] **by** (*smt Gromov-product-commute*)
  then show ∃ *N*. ∀ *n* ≥ *N*. ∀ *m* ≥ *N*. *M* ≤ *Gromov-product-at b* (*u m*) (*u n*) **by** *auto*
qed

lemma *Gromov-converging-at-boundary-imp-unbounded*:
  assumes *Gromov-converging-at-boundary u*
  shows (λ*n*. *dist a* (*u n*)) ⟶ ∞
proof −
  have ∃ *N*. ∀ *n* ≥ *N*. *dist a* (*u n*) ≥ *M* **for** *M*::*real*

using *assms* **unfolding** *Gromov-converging-at-boundary-def Gromov-product-e-x-x*[*symmetric*]
by *meson*
then show *?thesis*
unfolding *tendsto-PInfty eventually-sequentially* **by** (*meson dual-order.strict-trans1 gt-ex less-ereal.simps*(*1*))
qed

lemma *Gromov-converging-at-boundary-imp-not-constant*:
$\neg$(*Gromov-converging-at-boundary* ($\lambda n.$ *x*))
using *Gromov-converging-at-boundary-imp-unbounded*[*of* ($\lambda n.$ *x*) *x*] *Lim-bounded-PInfty*
by *auto*

lemma *Gromov-converging-at-boundary-imp-not-constant'*:
assumes *Gromov-converging-at-boundary u*
shows $\neg(\forall\, m\ n.\ u\ m = u\ n)$
using *Gromov-converging-at-boundary-imp-not-constant*
by (*metis* (*no-types*) *Gromov-converging-at-boundary-def assms order-refl*)

We introduce a partial equivalence relation, defined over the sequences that converge to infinity, and the constant sequences. Quotienting the space of admissible sequences by this equivalence relation will give rise to the Gromov completion.

definition *Gromov-completion-rel*::(*nat* $\Rightarrow$ *'a*::*Gromov-hyperbolic-space*) $\Rightarrow$ (*nat* $\Rightarrow$ *'a*) $\Rightarrow$ *bool*
where *Gromov-completion-rel u v* =
(((*Gromov-converging-at-boundary u* $\wedge$ *Gromov-converging-at-boundary v*
$\wedge$ ($\forall$ *a.* ($\lambda n.$ *Gromov-product-at a* (*u n*) (*v n*)) $\longlonglongrightarrow \infty$)))
$\vee$ $(\forall\, n\ m.\ u\ n = v\ m \wedge u\ n = u\ m \wedge v\ n = v\ m))$

We need some basic lemmas to work separately with sequences tending to the boundary and with constant sequences, as follows.

lemma *Gromov-completion-rel-const* [*simp*]:
Gromov-completion-rel ($\lambda n.$ *x*) ($\lambda n.$ *x*)
unfolding *Gromov-completion-rel-def* **by** *auto*

lemma *Gromov-completion-rel-to-const*:
assumes *Gromov-completion-rel u* ($\lambda n.$ *x*)
shows $u\ n = x$
using *assms* **unfolding** *Gromov-completion-rel-def* **using** *Gromov-converging-at-boundary-imp-not-constant*[*o*
x] **by** *auto*

lemma *Gromov-completion-rel-to-const'*:
assumes *Gromov-completion-rel* ($\lambda n.$ *x*) *u*
shows $u\ n = x$
using *assms* **unfolding** *Gromov-completion-rel-def* **using** *Gromov-converging-at-boundary-imp-not-constant*[*o*
x] **by** *auto*

lemma *Gromov-product-tendsto-PInf-a-b*:
assumes ($\lambda n.$ *Gromov-product-at a* (*u n*) (*v n*)) $\longlonglongrightarrow \infty$

shows ($\lambda n.$ *Gromov-product-at b* (*u n*) (*v n*)) $\longrightarrow \infty$
proof (*rule tendsto-sandwich*[*of* $\lambda n.$ *ereal*(*Gromov-product-at a* (*u n*) (*v n*)) + ($-$ *dist a b*) - - $\lambda n.$ ∞])
have *ereal*(*Gromov-product-at b* (*u n*) (*v n*)) $\geq$ *ereal*(*Gromov-product-at a* (*u n*) (*v n*)) + ($-$ *dist a b*) **for** *n*
using *Gromov-product-at-diff1* [*of a u n v n b*] **by** *auto*
then show $\forall_F$ *n in sequentially. ereal* (*Gromov-product-at a* (*u n*) (*v n*)) + *ereal* ($-$ *dist a b*) $\leq$ *ereal* (*Gromov-product-at b* (*u n*) (*v n*))
by *auto*
have ($\lambda n.$ *ereal*(*Gromov-product-at a* (*u n*) (*v n*)) + ($-$ *dist a b*)) $\longrightarrow \infty$ + ($-$ *dist a b*)
apply (*intro tendsto-intros*) **using** *assms* **by** *auto*
then show ($\lambda n.$ *ereal* (*Gromov-product-at a* (*u n*) (*v n*)) + *ereal* ($-$ *dist a b*)) $\longrightarrow \infty$ **by** *simp*
qed (*auto*)

lemma *Gromov-converging-at-boundary-rel*:
assumes *Gromov-converging-at-boundary u*
shows *Gromov-completion-rel u u*
unfolding *Gromov-completion-rel-def* **using** *Gromov-converging-at-boundary-imp-unbounded*[*OF assms*] *assms* **by** *auto*

We can now prove that we indeed have an equivalence relation.

lemma *part-equivp-Gromov-completion-rel*:
part-equivp Gromov-completion-rel
proof (*rule part-equivpI*)
show $\exists$ *x*::(*nat* $\Rightarrow$ *'a*). *Gromov-completion-rel x x*
apply (*rule exI*[*of* - $\lambda n.$ (*SOME a*::*'a. True*)]) **unfolding** *Gromov-completion-rel-def* **by** (*auto simp add*: *convergent-const*)

show *symp Gromov-completion-rel*
unfolding *symp-def Gromov-completion-rel-def* **by** (*auto simp add*: *Gromov-product-commute*) *metis+*

show *transp* (*Gromov-completion-rel*::(*nat* $\Rightarrow$ *'a*) $\Rightarrow$ (*nat* $\Rightarrow$ *'a*) $\Rightarrow$ *bool*)
unfolding *transp-def* **proof** (*intro allI impI*)
fix *u v w*::*nat*$\Rightarrow$*'a*
assume *UV*: *Gromov-completion-rel u v*
and *VW*: *Gromov-completion-rel v w*
show *Gromov-completion-rel u w*
proof (*cases* $\forall$ *n m. v n* = *v m*)
case *True*
define *a* **where** *a* = *v 0*
have $*$: *v* = ($\lambda n.$ *a*) **unfolding** *a-def* **using** *True* **by** *auto*
then have *u n* = *v 0 w n* = *v 0* **for** *n*
using *Gromov-completion-rel-to-const' Gromov-completion-rel-to-const UV VW* **unfolding** $*$ **by** *auto force*
then show *?thesis*
using *UV VW* **unfolding** *Gromov-completion-rel-def* **by** *auto*

next
case *False*
have ($\lambda n.$ *Gromov-product-at a* (*u n*) (*w n*)) $\longrightarrow \infty$ **for** *a*
proof (*rule tendsto-sandwich*[*of* $\lambda n.$ *min* (*ereal* (*Gromov-product-at a* (*u n*) (*v n*))) (*ereal* (*Gromov-product-at a* (*v n*) (*w n*))) + ($-$ *deltaG*(*TYPE*('*a*))) - - $\lambda n.$ ∞])
have *min* (*Gromov-product-at a* (*u n*) (*v n*)) (*Gromov-product-at a* (*v n*) (*w n*)) $-$ *deltaG*(*TYPE*('*a*)) $\leq$ *Gromov-product-at a* (*u n*) (*w n*) **for** *n*
by (*rule hyperb-ineq*)
then have *min* (*ereal* (*Gromov-product-at a* (*u n*) (*v n*))) (*ereal* (*Gromov-product-at a* (*v n*) (*w n*))) + *ereal* ($-$ *deltaG TYPE*('*a*)) $\leq$ *ereal* (*Gromov-product-at a* (*u n*) (*w n*)) **for** *n*
by (*auto simp del*: *ereal-min simp add*: *ereal-min*[*symmetric*])
then show $\forall_F$ *n in sequentially. min* (*ereal* (*Gromov-product-at a* (*u n*) (*v n*))) (*ereal* (*Gromov-product-at a* (*v n*) (*w n*)))
+ *ereal* ($-$ *deltaG TYPE*('*a*)) $\leq$ *ereal* (*Gromov-product-at a* (*u n*) (*w n*))
unfolding *eventually-sequentially* **by** *auto*

have ($\lambda n.$ *min* (*ereal* (*Gromov-product-at a* (*u n*) (*v n*))) (*ereal* (*Gromov-product-at a* (*v n*) (*w n*))) + ($-$ *deltaG*(*TYPE*('*a*)))) $\longrightarrow$ *min* ∞ ∞ + ($-$ *deltaG*(*TYPE*('*a*)))
apply (*intro tendsto-intros*) **using** *UV VW False* **unfolding** *Gromov-completion-rel-def* **by** *auto*
then show ($\lambda n.$ *min* (*ereal* (*Gromov-product-at a* (*u n*) (*v n*))) (*ereal* (*Gromov-product-at a* (*v n*) (*w n*))) + ($-$ *deltaG*(*TYPE*('*a*)))) $\longrightarrow \infty$ **by** *auto*
qed (*auto*)
then show *?thesis*
using *False UV VW* **unfolding** *Gromov-completion-rel-def* **by** *auto*
qed
qed
qed

We can now define the Gromov completion of a Gromov hyperbolic space, considering either sequences converging to a point on the boundary, or sequences converging inside the space, and quotienting by the natural equivalence relation.

quotient-type (**overloaded**) '*a Gromov-completion* =
nat $\Rightarrow$ ('*a*::*Gromov-hyperbolic-space*)
/ *partial*: *Gromov-completion-rel*
by (*rule part-equivp-Gromov-completion-rel*)

The Gromov completion contains is made of a copy of the original space, and new points forming the Gromov boundary.

definition *to-Gromov-completion*::('*a*::*Gromov-hyperbolic-space*) $\Rightarrow$ '*a Gromov-completion*
where *to-Gromov-completion x* = *abs-Gromov-completion* ($\lambda n.$ *x*)

definition *from-Gromov-completion*::('*a*::*Gromov-hyperbolic-space*) *Gromov-completion* $\Rightarrow$ '*a*
where *from-Gromov-completion* = *inv to-Gromov-completion*

definition *Gromov-boundary*::(*'a*::*Gromov-hyperbolic-space*) *Gromov-completion set*
  where *Gromov-boundary* = *UNIV* − *range to-Gromov-completion*

lemma *to-Gromov-completion-inj*:
  inj to-Gromov-completion
proof (*rule injI*)
  fix *x y*::*'a* **assume** *H*: *to-Gromov-completion x* = *to-Gromov-completion y*
  have *Gromov-completion-rel* (*λn. x*) (*λn. y*)
    apply (*subst Quotient3-rel*[*OF Quotient3-Gromov-completion*, *symmetric*])
    using *H* **unfolding** *to-Gromov-completion-def* **by** *auto*
  then show *x* = *y*
    using *Gromov-completion-rel-to-const* **by** *auto*
qed

lemma *from-to-Gromov-completion* [*simp*]:
  from-Gromov-completion (*to-Gromov-completion x*) = *x*
unfolding *from-Gromov-completion-def* **by** (*simp add*: *to-Gromov-completion-inj*)

lemma *to-from-Gromov-completion*:
  assumes *x* ∉ *Gromov-boundary*
  shows *to-Gromov-completion* (*from-Gromov-completion x*) = *x*
using *assms to-Gromov-completion-inj* **unfolding** *Gromov-boundary-def from-Gromov-completion-def*
by (*simp add*: *f-inv-into-f*)

lemma *not-in-Gromov-boundary*:
  assumes *x* ∉ *Gromov-boundary*
  shows ∃ *a*. *x* = *to-Gromov-completion a*
using *assms* **unfolding** *Gromov-boundary-def* **by** *auto*

lemma *not-in-Gromov-boundary'* [*simp*]:
  to-Gromov-completion x ∉ *Gromov-boundary*
unfolding *Gromov-boundary-def* **by** *auto*

lemma *abs-Gromov-completion-in-Gromov-boundary* [*simp*]:
  assumes *Gromov-converging-at-boundary u*
  shows *abs-Gromov-completion u* ∈ *Gromov-boundary*
using *Gromov-completion-rel-to-const Gromov-converging-at-boundary-imp-not-constant'*
  Gromov-converging-at-boundary-rel[*OF assms*]
  Quotient3-rel[*OF Quotient3-Gromov-completion*] *assms not-in-Gromov-boundary*
to-Gromov-completion-def
  by *fastforce*

lemma *rep-Gromov-completion-to-Gromov-completion* [*simp*]:
  rep-Gromov-completion (*to-Gromov-completion y*) = (*λn. y*)
proof −
  have *Gromov-completion-rel* (*λn. y*) (*rep-Gromov-completion* (*abs-Gromov-completion*
(*λn. y*)))
    by (*metis Gromov-completion-rel-const Quotient3-Gromov-completion rep-abs-rsp*)

then show *?thesis*
unfolding *to-Gromov-completion-def* **using** *Gromov-completion-rel-to-const'*
by *blast*
qed

To distinguish the case of points inside the space or in the boundary, we introduce the following case distinction.

lemma *Gromov-completion-cases* [*case-names to-Gromov-completion boundary*, *cases type*: *Gromov-completion*]:
$(\bigwedge x.\ z = \textit{to-Gromov-completion}\ x \Longrightarrow P) \Longrightarrow (z \in \textit{Gromov-boundary} \Longrightarrow P) \Longrightarrow P$
apply (*cases* $z \in$ *Gromov-boundary*) **using** *not-in-Gromov-boundary* **by** *auto*

15.2 Extending the original distance and the original Gromov product to the completion

In this subsection, we extend the Gromov product to the boundary, by taking limits along sequences tending to the point in the boundary. This does not converge, but it does up to δ, so for definiteness we use a lim inf over all sequences tending to the boundary point – one interest of this definition is that the extended Gromov product still satisfies the hyperbolicity inequality. One difficulty is that this extended Gromov product can take infinite values (it does so exactly on the pair (x, x) where x is in the boundary), so we should define this product in extended nonnegative reals.

We also extend the original distance, by $+\infty$ on the boundary. This is not a really interesting function, but it will be instrumental below. Again, this extended Gromov distance (not to be mistaken for the genuine distance we will construct later on on the completion) takes values in extended nonnegative reals.

Since the extended Gromov product and the extension of the original distance both take values in $[0, +\infty]$, it may seem natural to define them in ennreal. This is the choice that was made in a previous implementation, but it turns out that one keeps computing with these numbers, writing down inequalities and subtractions. ennreal is ill suited for this kind of computations, as it only works well with additions. Hence, the implementation was switched to ereal, where proofs are indeed much smoother.

To define the extended Gromov product, one takes a limit of the Gromov product along any sequence, as it does not depend up to δ on the chosen sequence. However, if one wants to keep the exact inequality that defines hyperbolicity, but at all points, then using an infimum is the best choice.

definition *extended-Gromov-product-at*::(*'a*::*Gromov-hyperbolic-space*) $\Rightarrow$ *'a Gromov-completion* $\Rightarrow$ *'a Gromov-completion* $\Rightarrow$ *ereal*
where *extended-Gromov-product-at e x y* = *Inf* {*liminf* ($\lambda n.$ *ereal*(*Gromov-product-at e* (*u n*) (*v n*))) |*u v*. *abs-Gromov-completion u* = *x* $\wedge$ *abs-Gromov-completion v* = *y* $\wedge$ *Gromov-completion-rel u u* $\wedge$ *Gromov-completion-rel v v*}

definition *extended-Gromov-distance*::(*'a*::*Gromov-hyperbolic-space*) *Gromov-completion* $\Rightarrow$ *'a Gromov-completion* $\Rightarrow$ *ereal*
where *extended-Gromov-distance x y* =
(*if x* $\in$ *Gromov-boundary* $\vee$ *y* $\in$ *Gromov-boundary then* ∞
else ereal (*dist* (*inv to-Gromov-completion x*) (*inv to-Gromov-completion y*)))

The extended distance and the extended Gromov product are invariant under exchange of the points, readily from the definition.

lemma *extended-Gromov-distance-commute*:
extended-Gromov-distance x y = *extended-Gromov-distance y x*
unfolding *extended-Gromov-distance-def* **by** (*simp add*: *dist-commute*)

lemma *extended-Gromov-product-nonneg* [*mono-intros*, *simp*]:
0 $\leq$ *extended-Gromov-product-at e x y*
unfolding *extended-Gromov-product-at-def* **by** (*rule Inf-greatest*, *auto intro*: *Liminf-bounded always-eventually*)

lemma *extended-Gromov-distance-nonneg* [*mono-intros*, *simp*]:
0 $\leq$ *extended-Gromov-distance x y*
unfolding *extended-Gromov-distance-def* **by** *auto*

lemma *extended-Gromov-product-at-commute*:
extended-Gromov-product-at e x y = *extended-Gromov-product-at e y x*
unfolding *extended-Gromov-product-at-def*
proof (*rule arg-cong*[*of - - Inf*])
have {*liminf* (λ*n*. *ereal* (*Gromov-product-at e* (*u n*) (*v n*))) |*u v*.
abs-Gromov-completion u = *x* $\wedge$ *abs-Gromov-completion v* = *y* $\wedge$ *Gromov-completion-rel u u* $\wedge$ *Gromov-completion-rel v v*} =
{*liminf* (λ*n*. *ereal* (*Gromov-product-at e* (*v n*) (*u n*))) |*u v*.
abs-Gromov-completion v = *y* $\wedge$ *abs-Gromov-completion u* = *x* $\wedge$ *Gromov-completion-rel v v* $\wedge$ *Gromov-completion-rel u u*}
by (*auto simp add*: *Gromov-product-commute*)
then show {*liminf* (λ*n*. *ereal* (*Gromov-product-at e* (*u n*) (*v n*))) |*u v*.
abs-Gromov-completion u = *x* $\wedge$ *abs-Gromov-completion v* = *y* $\wedge$ *Gromov-completion-rel u u* $\wedge$ *Gromov-completion-rel v v*} =
{*liminf* (λ*n*. *ereal* (*Gromov-product-at e* (*u n*) (*v n*))) |*u v*.
abs-Gromov-completion u = *y* $\wedge$ *abs-Gromov-completion v* = *x* $\wedge$ *Gromov-completion-rel u u* $\wedge$ *Gromov-completion-rel v v*}
by *auto*
qed

Inside the space, the extended distance and the extended Gromov product coincide with the original ones.

lemma *extended-Gromov-distance-inside* [*simp*]:
extended-Gromov-distance (*to-Gromov-completion x*) (*to-Gromov-completion y*) = *dist x y*

unfolding *extended-Gromov-distance-def Gromov-boundary-def* **by** (*auto simp add*: *to-Gromov-completion-inj*)

lemma *extended-Gromov-product-inside* [*simp*] :
extended-Gromov-product-at e (*to-Gromov-completion x*) (*to-Gromov-completion y*) = *Gromov-product-at e x y*
proof −
have *A*: $u = (\lambda n.\ z)$ **if** *H*: *abs-Gromov-completion u* = *abs-Gromov-completion* ($\lambda n.\ z$) *Gromov-completion-rel u u* **for** *u* **and** $z::'a$
proof −
have *Gromov-completion-rel u* ($\lambda n.\ z$)
apply (*subst Quotient3-rel*[*OF Quotient3-Gromov-completion*, *symmetric*])
using *H uniformity-dist-class-def* **by** *auto*
then show *?thesis* **using** *Gromov-completion-rel-to-const* **by** *auto*
qed
then have $*$: {*u*. *abs-Gromov-completion u* = *to-Gromov-completion z* ∧ *Gromov-completion-rel u u*} = {($\lambda n.\ z$)} **for** $z::'a$
unfolding *to-Gromov-completion-def* **by** *auto*
have $**$: {*F u v* |*u v*. *abs-Gromov-completion u* = *to-Gromov-completion x* ∧ *abs-Gromov-completion v* = *to-Gromov-completion y* ∧ *Gromov-completion-rel u u* ∧ *Gromov-completion-rel v v*}
= {*F* ($\lambda n.\ x$) ($\lambda n.\ y$)} **for** $F::(nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a) \Rightarrow ereal$
using $*$[*of x*] $*$[*of y*] **unfolding** *extended-Gromov-product-at-def* **by** (*auto*, *smt mem-Collect-eq singletonD*)

have *extended-Gromov-product-at e* (*to-Gromov-completion x*) (*to-Gromov-completion y*) = *Inf* {*liminf* ($\lambda n.$ *ereal*(*Gromov-product-at e* (($\lambda n.\ x$) *n*) (($\lambda n.\ y$) *n*)))}
unfolding *extended-Gromov-product-at-def* $**$ **by** *simp*
also have ... = *ereal*(*Gromov-product-at e x y*)
by (*auto simp add*: *Liminf-const*)
finally show *extended-Gromov-product-at e* (*to-Gromov-completion x*) (*to-Gromov-completion y*) = *Gromov-product-at e x y*
by *simp*
qed

A point in the boundary is at infinite extended distance of everyone, including itself: the extended distance is obtained by taking the supremum along all sequences tending to this point, so even for one single point one can take two sequences tending to it at different speeds, which results in an infinite extended distance.

lemma *extended-Gromov-distance-PInf-boundary* [*simp*]:
assumes $x \in$ *Gromov-boundary*
shows *extended-Gromov-distance x y* = ∞ *extended-Gromov-distance y x* = ∞
unfolding *extended-Gromov-distance-def* **using** *assms* **by** *auto*

By construction, the extended distance still satisfies the triangle inequality.

lemma *extended-Gromov-distance-triangle* [*mono-intros*]:
extended-Gromov-distance x z ≤ *extended-Gromov-distance x y* + *extended-Gromov-distance y z*

proof (*cases* $x \in$ *Gromov-boundary* $\vee$ $y \in$ *Gromov-boundary* $\vee$ $z \in$ *Gromov-boundary*)
case *True*
then have $*$: *extended-Gromov-distance* x y + *extended-Gromov-distance* y z = ∞ **by** *auto*
show *?thesis* **by** (*simp add*: $*$)
next
case *False*
then obtain a b c **where** *abc*: x = *to-Gromov-completion* a y = *to-Gromov-completion* b z = *to-Gromov-completion* c
unfolding *Gromov-boundary-def* **by** *auto*
show *?thesis*
unfolding *abc* **using** *dist-triangle*[*of* a c b] *ennreal-leI* **by** *fastforce*
qed

The extended Gromov product can be bounded by the extended distance, just like inside the space.

lemma *extended-Gromov-product-le-dist* [*mono-intros*]:
extended-Gromov-product-at e x y $\leq$ *extended-Gromov-distance* (*to-Gromov-completion* e) x
proof (*cases* x)
case *boundary*
then show *?thesis* **by** *simp*
next
case (*to-Gromov-completion* a)
define v **where** v = *rep-Gromov-completion* y
have $*$: *abs-Gromov-completion* ($\lambda n.$ a) = x $\wedge$ *abs-Gromov-completion* v = y $\wedge$ *Gromov-completion-rel* ($\lambda n.$ a) ($\lambda n.$ a) $\wedge$ *Gromov-completion-rel* v v
unfolding *v-def to-Gromov-completion to-Gromov-completion-def*
by (*auto simp add*: *Quotient3-rep-reflp*[*OF Quotient3-Gromov-completion*] *Quotient3-abs-rep*[*OF Quotient3-Gromov-completion*])
have *extended-Gromov-product-at* e x y $\leq$ *liminf* ($\lambda n.$ *ereal*(*Gromov-product-at* e a (v n)))
unfolding *extended-Gromov-product-at-def* **apply** (*rule Inf-lower*) **using** $*$ **by** *force*
also have ... $\leq$ *liminf* ($\lambda n.$ *ereal*(*dist* e a))
using *Gromov-product-le-dist*(*1*)[*of* e a] **by** (*auto intro*!: *Liminf-mono*)
also have ... = *ereal*(*dist* e a)
by (*simp add*: *Liminf-const*)
also have ... = *extended-Gromov-distance* (*to-Gromov-completion* e) x
unfolding *to-Gromov-completion* **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *extended-Gromov-product-le-dist'* [*mono-intros*]:
extended-Gromov-product-at e x y $\leq$ *extended-Gromov-distance* (*to-Gromov-completion* e) y
using *extended-Gromov-product-le-dist*[*of* e y x] **by** (*simp add*: *extended-Gromov-product-at-commute*)

The Gromov product inside the space varies by at most the distance when one varies one of the points. We will need the same statement for the extended Gromov product. The proof is done using this inequality inside the space, and passing to the limit.

lemma *extended-Gromov-product-at-diff3* [*mono-intros*]:
extended-Gromov-product-at e x y $\leq$ *extended-Gromov-product-at e x z* + *extended-Gromov-distance y z*
proof (*cases* (*extended-Gromov-distance y z* = ∞) $\vee$ (*extended-Gromov-product-at e x z* = ∞))
case *False*
then have *y* $\notin$ *Gromov-boundary z* $\notin$ *Gromov-boundary*
using *extended-Gromov-distance-PInf-boundary* **by** *auto*
then obtain *b c* **where** *b*: *y* = *to-Gromov-completion b* **and** *c*: *z* = *to-Gromov-completion c*
unfolding *Gromov-boundary-def* **by** *auto*
have *extended-Gromov-distance y z* = *ereal*(*dist b c*)
unfolding *b c* **by** *auto*
have *extended-Gromov-product-at e x y* $\leq$ (*extended-Gromov-product-at e x z* + *extended-Gromov-distance y z*) + *h* **if** *h*>*0* **for** *h*
proof −
have $\exists$ *t*$\in${*liminf* (λ*n*. *ereal*(*Gromov-product-at e* (*u n*) (*w n*))) |*u w*. *abs-Gromov-completion u* = *x*
$\wedge$ *abs-Gromov-completion w* = *z* $\wedge$ *Gromov-completion-rel u u* $\wedge$ *Gromov-completion-rel w w*}.
t < *extended-Gromov-product-at e x z* + *h*
apply (*subst Inf-less-iff*[*symmetric*]) **using** *False* ‹*h* > *0*› *extended-Gromov-product-nonneg*[*of e x z*] **unfolding** *extended-Gromov-product-at-def*[*symmetric*]
by (*metis add.right-neutral ereal-add-left-cancel-less order-refl*)
then obtain *u w* **where** *H*: *abs-Gromov-completion u* = *x abs-Gromov-completion w* = *z*
Gromov-completion-rel u u Gromov-completion-rel w w
liminf (λ*n*. *ereal*(*Gromov-product-at e* (*u n*) (*w n*))) < *extended-Gromov-product-at e x z* + *h*
by *auto*
then have *w*: *w n* = *c* **for** *n*
using *c Gromov-completion-rel-to-const Quotient3-Gromov-completion Quotient3-rel to-Gromov-completion-def* **by** *fastforce*
define *v* **where** *v*: *v* = (λ*n*::*nat*. *b*)
have *abs-Gromov-completion v* = *y Gromov-completion-rel v v*
unfolding *v* **by** (*auto simp add*: *b to-Gromov-completion-def*)

have *Gromov-product-at e* (*u n*) (*v n*) $\leq$ *Gromov-product-at e* (*u n*) (*w n*) + *dist b c* **for** *n*
unfolding *v w* **using** *Gromov-product-at-diff3*[*of e u n b c*] **by** *auto*
then have $*$: *ereal*(*Gromov-product-at e* (*u n*) (*v n*)) $\leq$ *ereal*(*Gromov-product-at e* (*u n*) (*w n*)) + *extended-Gromov-distance y z* **for** *n*
unfolding ‹*extended-Gromov-distance y z* = *ereal*(*dist b c*)› **by** *fastforce*
have *extended-Gromov-product-at e x y* $\leq$ *liminf*(λ*n*. *ereal*(*Gromov-product-at e* (*u n*) (*v n*)))
unfolding *extended-Gromov-product-at-def* **by** (*rule Inf-lower*, *auto*, *rule*

exI[*of* - *u*], *rule exI*[*of* - *v*], *auto*, *fact*+)
also have ... $\leq$ *liminf*(λ*n*. *ereal*(*Gromov-product-at e* (*u n*) (*w n*)) + *extended-Gromov-distance y z*)
apply (*rule Liminf-mono*) **using** $*$ **unfolding** *eventually-sequentially* **by** *auto*
also have ... = *liminf*(λ*n*. *ereal*(*Gromov-product-at e* (*u n*) (*w n*))) + *extended-Gromov-distance y z*
apply (*rule Liminf-add-ereal-right*) **using** *False* **by** *auto*
also have ... $\leq$ *extended-Gromov-product-at e x z* + *h* + *extended-Gromov-distance y z*
using *less-imp-le*[*OF H*(*5*)] **by** (*auto intro*: *mono-intros*)
finally show *?thesis*
by (*simp add*: *algebra-simps*)
qed
then show *?thesis*
using *ereal-le-epsilon* **by** *blast*
next
case *True*
then show *?thesis* **by** *auto*
qed

lemma *extended-Gromov-product-at-diff2* [*mono-intros*]:
extended-Gromov-product-at e x y $\leq$ *extended-Gromov-product-at e z y* + *extended-Gromov-distance x z*
using *extended-Gromov-product-at-diff3*[*of e y x z*] **by** (*simp add*: *extended-Gromov-product-at-commute*)

lemma *extended-Gromov-product-at-diff1* [*mono-intros*]:
extended-Gromov-product-at e x y $\leq$ *extended-Gromov-product-at f x y* + *dist e f*
proof (*cases extended-Gromov-product-at f x y* = ∞)
case *False*
have *extended-Gromov-product-at e x y* $\leq$ (*extended-Gromov-product-at f x y* + *dist e f*) + *h* **if** *h* > *0* **for** *h*
proof −
have $\exists$ *t*$\in${*liminf* (λ*n*. *ereal*(*Gromov-product-at f* (*u n*) (*v n*))) |*u v*. *abs-Gromov-completion u* = *x*
$\wedge$ *abs-Gromov-completion v* = *y* $\wedge$ *Gromov-completion-rel u u* $\wedge$ *Gromov-completion-rel v v*}.
t < *extended-Gromov-product-at f x y* + *h*
apply (*subst Inf-less-iff*[*symmetric*]) **using** *False* ‹*h* > *0*› *extended-Gromov-product-nonneg*[*of f x y*] **unfolding** *extended-Gromov-product-at-def*[*symmetric*]
by (*metis add.right-neutral ereal-add-left-cancel-less order-refl*)
then obtain *u v* **where** *H*: *abs-Gromov-completion u* = *x abs-Gromov-completion v* = *y*
Gromov-completion-rel u u Gromov-completion-rel v v
liminf (λ*n*. *ereal*(*Gromov-product-at f* (*u n*) (*v n*))) < *extended-Gromov-product-at f x y* + *h*
by *auto*

have *Gromov-product-at e* (*u n*) (*v n*) $\leq$ *Gromov-product-at f* (*u n*) (*v n*) + *dist e f* **for** *n*

using *Gromov-product-at-diff1* [*of e u n v n f*] **by** *auto*
then have ∗: *ereal*(*Gromov-product-at e* (*u n*) (*v n*)) ≤ *ereal*(*Gromov-product-at f* (*u n*) (*v n*)) + *dist e f* **for** *n*
by *fastforce*
have *extended-Gromov-product-at e x y* ≤ *liminf*(λ*n*. *ereal*(*Gromov-product-at e* (*u n*) (*v n*)))
unfolding *extended-Gromov-product-at-def* **by** (*rule Inf-lower*, *auto*, *rule exI*[*of - u*], *rule exI*[*of - v*], *auto*, *fact*+)
also have ... ≤ *liminf*(λ*n*. *ereal*(*Gromov-product-at f* (*u n*) (*v n*)) + *dist e f*)
apply (*rule Liminf-mono*) **using** ∗ **unfolding** *eventually-sequentially* **by** *auto*
also have ... = *liminf*(λ*n*. *ereal*(*Gromov-product-at f* (*u n*) (*v n*))) + *dist e f*
apply (*rule Liminf-add-ereal-right*) **using** *False* **by** *auto*
also have ... ≤ *extended-Gromov-product-at f x y* + *h* + *dist e f*
using *less-imp-le*[*OF H*(*5*)] **by** (*auto intro*: *mono-intros*)
finally show *?thesis*
by (*simp add*: *algebra-simps*)
qed
then show *?thesis*
using *ereal-le-epsilon* **by** *blast*
next
case *True*
then show *?thesis* **by** *auto*
qed

A point in the Gromov boundary is represented by a sequence tending to infinity and converging in the Gromov boundary, essentially by definition.

lemma *Gromov-boundary-abs-converging*:
assumes *x* ∈ *Gromov-boundary abs-Gromov-completion u* = *x Gromov-completion-rel u u*
shows *Gromov-converging-at-boundary u*
proof −
have *Gromov-converging-at-boundary u* ∨ (∀ *m n*. *u n* = *u m*)
using *assms* **unfolding** *Gromov-completion-rel-def* **by** *auto*
moreover have ¬(∀ *m n*. *u n* = *u m*)
proof (*rule ccontr*, *simp*)
assume ∗: ∀ *m n*. *u n* = *u m*
define *z* **where** *z* = *u 0*
then have *z*: *u* = (λ*n*. *z*)
using ∗ **by** *auto*
then have *x* = *to-Gromov-completion z*
using *assms* **unfolding** *z to-Gromov-completion-def* **by** *auto*
then show *False* **using** ‹*x* ∈ *Gromov-boundary*› **unfolding** *Gromov-boundary-def* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *Gromov-boundary-rep-converging*:
assumes *x* ∈ *Gromov-boundary*

shows *Gromov-converging-at-boundary* (*rep-Gromov-completion x*)
apply (*rule Gromov-boundary-abs-converging*[*OF assms*])
using *Quotient3-Gromov-completion Quotient3-abs-rep Quotient3-rep-reflp* **by** *fastforce+*

We can characterize the points for which the Gromov product is infinite: they have to be the same point, at infinity. This is essentially equivalent to the definition of the Gromov completion, but there is some boilerplate to get the proof working.

lemma *Gromov-boundary-extended-product-PInf* [*simp*]:
extended-Gromov-product-at e x y = ∞ ⟷ (*x* ∈ *Gromov-boundary* ∧ *y* = *x*)
proof
fix *x y*::′*a Gromov-completion* **assume** *x* ∈ *Gromov-boundary* ∧ *y* = *x*
then have *H*: *y* = *x x* ∈ *Gromov-boundary* **by** *auto*
have ∗: *liminf* (λ*n*. *ereal* (*Gromov-product-at e* (*u n*) (*v n*))) = ∞ **if**
abs-Gromov-completion u = *x abs-Gromov-completion v* = *y*
Gromov-completion-rel u u Gromov-completion-rel v v **for** *u v*
proof −
have *Gromov-converging-at-boundary u Gromov-converging-at-boundary v*
using *Gromov-boundary-abs-converging that H* **by** *auto*
have *Gromov-completion-rel u v* **using** *that* ‹*y* = *x*›
using *Quotient3-rel*[*OF Quotient3-Gromov-completion*] **by** *fastforce*
then have (λ*n*. *Gromov-product-at e* (*u n*) (*v n*)) ⟶ ∞
unfolding *Gromov-completion-rel-def* **using** *Gromov-converging-at-boundary-imp-not-constant*′[*OF* ‹*Gromov-converging-at-boundary u*›] **by** *auto*
then show *?thesis*
by (*simp add*: *tendsto-iff-Liminf-eq-Limsup*)
qed
then show *extended-Gromov-product-at e x y* = ∞
unfolding *extended-Gromov-product-at-def* **by** (*auto intro*: *Inf-eqI*)
next
fix *x y*::′*a Gromov-completion* **assume** *H*: *extended-Gromov-product-at e x y* = ∞
then have *extended-Gromov-distance* (*to-Gromov-completion e*) *x* = ∞
using *extended-Gromov-product-le-dist*[*of e x y*] *neq-top-trans* **by** *auto*
then have *x* ∈ *Gromov-boundary*
by (*metis ereal.distinct*(*1*) *extended-Gromov-distance-def infinity-ereal-def not-in-Gromov-boundary*′)
have *extended-Gromov-distance* (*to-Gromov-completion e*) *y* = ∞
using *extended-Gromov-product-le-dist*[*of e y x*] *neq-top-trans H* **by** (*auto simp add*: *extended-Gromov-product-at-commute*)
then have *y* ∈ *Gromov-boundary*
by (*metis ereal.distinct*(*1*) *extended-Gromov-distance-def infinity-ereal-def not-in-Gromov-boundary*′)
define *u* **where** *u* = *rep-Gromov-completion x*
define *v* **where** *v* = *rep-Gromov-completion y*
have *A*: *Gromov-converging-at-boundary u Gromov-converging-at-boundary v*
unfolding *u-def v-def* **using** ‹*x* ∈ *Gromov-boundary*› ‹*y* ∈ *Gromov-boundary*›
by (*auto simp add*: *Gromov-boundary-rep-converging*)

have *abs-Gromov-completion u* = *x* ∧ *abs-Gromov-completion v* = *y* ∧ *Gro-*

mov-completion-rel u u $\wedge$ *Gromov-completion-rel v v*
unfolding *u-def v-def*
using *Quotient3-abs-rep*[*OF Quotient3-Gromov-completion*] *Quotient3-rep-reflp*[*OF Quotient3-Gromov-completion*] **by** *auto*
then have *extended-Gromov-product-at e x y* $\leq$ *liminf* ($\lambda n.$ *ereal*(*Gromov-product-at e* (*u n*) (*v n*)))
unfolding *extended-Gromov-product-at-def* **by** (*auto intro*!: *Inf-lower*)
then have ($\lambda n.$ *ereal*(*Gromov-product-at e* (*u n*) (*v n*))) $\longrightarrow \infty$
unfolding *H* **by** (*simp add*: *liminf-PInfty*)
then have ($\lambda n.$ *ereal*(*Gromov-product-at a* (*u n*) (*v n*))) $\longrightarrow \infty$ **for** *a*
using *Gromov-product-tendsto-PInf-a-b* **by** *auto*

then have *Gromov-completion-rel u v*
unfolding *Gromov-completion-rel-def* **using** *A* **by** *auto*
then have *abs-Gromov-completion u* = *abs-Gromov-completion v*
using *Quotient3-rel-abs*[*OF Quotient3-Gromov-completion*] **by** *auto*
then have $x = y$
unfolding *u-def v-def Quotient3-abs-rep*[*OF Quotient3-Gromov-completion*] **by** *auto*
then show $x \in$ *Gromov-boundary* $\wedge$ $y = x$
using ‹$x \in$ *Gromov-boundary*› **by** *auto*
qed

As for points inside the space, we deduce that the extended Gromov product between x and x is just the extended distance to the basepoint.

lemma *extended-Gromov-product-e-x-x* [*simp*]:
extended-Gromov-product-at e x x = *extended-Gromov-distance* (*to-Gromov-completion e*) *x*
proof (*cases x*)
case *boundary*
then show *?thesis* **using** *Gromov-boundary-extended-product-PInf* **by** *auto*
next
case (*to-Gromov-completion a*)
then show *?thesis* **by** *auto*
qed

The inequality in terms of Gromov products characterizing hyperbolicity extends in the same form to the Gromov completion, by taking limits of this inequality in the space.

lemma *extended-hyperb-ineq* [*mono-intros*]:
extended-Gromov-product-at (*e*::*'a*::*Gromov-hyperbolic-space*) *x z* $\geq$
min (*extended-Gromov-product-at e x y*) (*extended-Gromov-product-at e y z*) − *deltaG*(*TYPE*(*'a*))
proof −
have *min* (*extended-Gromov-product-at e x y*) (*extended-Gromov-product-at e y z*) − *deltaG*(*TYPE*(*'a*)) $\leq$
Inf {*liminf* ($\lambda n.$ *ereal* (*Gromov-product-at e* (*u n*) (*v n*))) |*u v*.
abs-Gromov-completion u = *x* $\wedge$ *abs-Gromov-completion v* = *z* $\wedge$ *Gromov-completion-rel u u* $\wedge$ *Gromov-completion-rel v v*}

proof (*rule cInf-greatest*, *auto*)
define *u* **where** *u* = *rep-Gromov-completion x*
define *w* **where** *w* = *rep-Gromov-completion z*
have *abs-Gromov-completion u* = *x* ∧ *abs-Gromov-completion w* = *z* ∧ *Gromov-completion-rel u u* ∧ *Gromov-completion-rel w w*
unfolding *u-def w-def*
using *Quotient3-abs-rep*[*OF Quotient3-Gromov-completion*] *Quotient3-rep-reflp*[*OF Quotient3-Gromov-completion*] **by** *auto*
then show ∃ *t u*. *Gromov-completion-rel u u* ∧
(∃ *v*. *abs-Gromov-completion v* = *z* ∧ *abs-Gromov-completion u* = *x* ∧ *t* = *liminf* (*λn*. *ereal* (*Gromov-product-at e* (*u n*) (*v n*))) ∧ *Gromov-completion-rel v v*)
by *auto*
next
fix *u w* **assume** *H*: *x* = *abs-Gromov-completion u z* = *abs-Gromov-completion w*
Gromov-completion-rel u u Gromov-completion-rel w w
define *v* **where** *v* = *rep-Gromov-completion y*
have *Y*: *y* = *abs-Gromov-completion v Gromov-completion-rel v v*
unfolding *v-def*
by (*auto simp add*: *Quotient3-abs-rep*[*OF Quotient3-Gromov-completion*] *Quotient3-rep-reflp*[*OF Quotient3-Gromov-completion*])

have ∗: *min* (*ereal*(*Gromov-product-at e* (*u n*) (*v n*))) (*ereal*(*Gromov-product-at e* (*v n*) (*w n*))) ≤ *ereal*(*Gromov-product-at e* (*u n*) (*w n*)) + *deltaG*(*TYPE*(*'a*)) **for** *n*
by (*subst ereal-min*[*symmetric*], *subst plus-ereal.simps*(*1*), *intro mono-intros*)

have *extended-Gromov-product-at e* (*abs-Gromov-completion u*) *y* ≤ *liminf* (*λn*. *ereal*(*Gromov-product-at e* (*u n*) (*v n*)))
unfolding *extended-Gromov-product-at-def* **using** *Y H* **by** (*auto intro*!: *Inf-lower*)
moreover have *extended-Gromov-product-at e y* (*abs-Gromov-completion w*) ≤ *liminf* (*λn*. *ereal*(*Gromov-product-at e* (*v n*) (*w n*)))
unfolding *extended-Gromov-product-at-def* **using** *Y H* **by** (*auto intro*!: *Inf-lower*)
ultimately have *min* (*extended-Gromov-product-at e* (*abs-Gromov-completion u*) *y*) (*extended-Gromov-product-at e y* (*abs-Gromov-completion w*))
≤ *min* (*liminf* (*λn*. *ereal*(*Gromov-product-at e* (*u n*) (*v n*)))) (*liminf* (*λn*. *ereal*(*Gromov-product-at e* (*v n*) (*w n*))))
by (*intro mono-intros*, *auto*)
also have ... = *liminf* (*λn*. *min* (*ereal*(*Gromov-product-at e* (*u n*) (*v n*))) (*ereal*(*Gromov-product-at e* (*v n*) (*w n*))))
by (*rule Liminf-min-eq-min-Liminf*[*symmetric*])
also have ... ≤ *liminf* (*λn*. *ereal*(*Gromov-product-at e* (*u n*) (*w n*)) + *deltaG*(*TYPE*(*'a*)))
using ∗ **by** (*auto intro*!: *Liminf-mono*)
also have ... = *liminf* (*λn*. *ereal*(*Gromov-product-at e* (*u n*) (*w n*))) + *deltaG*(*TYPE*(*'a*))
by (*intro Liminf-add-ereal-right*, *auto*)
finally show *min* (*extended-Gromov-product-at e* (*abs-Gromov-completion u*)

y) (*extended-Gromov-product-at e y* (*abs-Gromov-completion w*))
$\leq$ *liminf* ($\lambda n.$ *ereal* (*Gromov-product-at e* (*u n*) (*w n*))) + *ereal* (*deltaG TYPE*(*'a*))
by *simp*
qed
then show *?thesis* **unfolding** *extended-Gromov-product-at-def* **by** *auto*
qed

lemma *extended-hyperb-ineq'* [*mono-intros*]:
extended-Gromov-product-at (*e*::*'a*::*Gromov-hyperbolic-space*) *x z* + *deltaG*(*TYPE*(*'a*)) $\geq$
min (*extended-Gromov-product-at e x y*) (*extended-Gromov-product-at e y z*)
using *extended-hyperb-ineq*[*of e x y z*] **unfolding** *ereal-minus-le-iff* **by** (*simp add*: *add.commute*)

lemma *zero-le-ereal* [*mono-intros*]:
assumes *0* $\leq$ *z*
shows *0* $\leq$ *ereal z*
using *assms* **by** *auto*

lemma *extended-hyperb-ineq-4-points'* [*mono-intros*]:
Min {*extended-Gromov-product-at* (*e*::*'a*::*Gromov-hyperbolic-space*) *x y*, *extended-Gromov-product-at e y z*, *extended-Gromov-product-at e z t*} $\leq$ *extended-Gromov-product-at e x t* + *2* $*$ *deltaG*(*TYPE*(*'a*))
proof −
have *min* (*extended-Gromov-product-at e x y* + *0*) (*min* (*extended-Gromov-product-at e y z*) (*extended-Gromov-product-at e z t*))
$\leq$ *min* (*extended-Gromov-product-at e x y* + *deltaG*(*TYPE*(*'a*))) (*extended-Gromov-product-at e y t* + *deltaG*(*TYPE*(*'a*)))
by (*intro mono-intros*)
also have ... = *min* (*extended-Gromov-product-at e x y*) (*extended-Gromov-product-at e y t*) + *deltaG*(*TYPE*(*'a*))
by (*simp add*: *add-mono-thms-linordered-semiring*(*3*) *dual-order.antisym min-def*)
also have ... $\leq$ (*extended-Gromov-product-at e x t* + *deltaG*(*TYPE*(*'a*))) + *deltaG*(*TYPE*(*'a*))
by (*intro mono-intros*)
finally show *?thesis* **apply** (*auto simp add*: *algebra-simps*)
by (*metis* (*no-types*, *opaque-lifting*) *add.commute add.left-commute mult-2-right plus-ereal.simps*(*1*))
qed

lemma *extended-hyperb-ineq-4-points* [*mono-intros*]:
Min {*extended-Gromov-product-at* (*e*::*'a*::*Gromov-hyperbolic-space*) *x y*, *extended-Gromov-product-at e y z*, *extended-Gromov-product-at e z t*} − *2* $*$ *deltaG*(*TYPE*(*'a*)) $\leq$ *extended-Gromov-product-at e x t*
using *extended-hyperb-ineq-4-points'*[*of e x y z*] **unfolding** *ereal-minus-le-iff* **by** (*simp add*: *add.commute*)

15.3 Construction of the distance on the Gromov completion

We want now to define the natural topology of the Gromov completion. Most textbooks first define a topology on ∂X, or sometimes on $X \cup \partial X$, and then much later a distance on ∂X (but they never do the tedious verification that the distance defines the same topology as the topology defined before). I have not seen one textbook defining a distance on $X \cup \partial X$. It turns out that one can in fact define a distance on $X \cup \partial X$, whose restriction to ∂X is the usual distance on the Gromov boundary, and define the topology of $X \cup \partial X$ using it. For formalization purposes, this is very convenient as topologies defined with distances are automatically nice and tractable (no need to check separation axioms, for instance). The price to pay is that, once we have defined the distance, we have to check that it defines the right notion of convergence one expects.

What we would like to take for the distance is $d(x, y) = e^{-(x,y)_o}$, where o is some fixed basepoint in the space. However, this does not behave like a distance at small scales (but it is essentially the right thing at large scales), and it does not really satisfy the triangle inequality. However, $e^{-\epsilon(x,y)_o}$ almost satisfies the triangle inequality if ϵ is small enough, i.e., it is equivalent to a function satisfying the triangle inequality. This gives a genuine distance on the boundary, but not inside the space as it does not vanish on pairs (x, x).

A third try would be to take $d(x, y) = \min(\tilde{d}(x, y), e^{-\epsilon(x,y)_o})$ where $\tilde{d}$ is the natural extension of d to the Gromov completion (it is infinite if x or y belongs to the boundary). However, we can not prove that it is equivalent to a distance.

Finally, it works with $d(x, y) \asymp \min(\tilde{d}(x, y)^{1/2}, e^{-\epsilon(x,y)_o}$. This is what we will prove below. To construct the distance, we use the results proved in the locale `Turn_into_distance`. For this, we need to check that our quasi-distance satisfies a weird version of the triangular inequality.

All this construction depends on a basepoint, that we fix arbitrarily once and for all.

definition *epsilonG*::(*'a*::*Gromov-hyperbolic-space*) *itself* ⇒ *real*
  where *epsilonG - = ln 2 / (2+2∗deltaG(TYPE('a)))*

definition *basepoint*::*'a*
  where *basepoint = (SOME a. True)*

lemma *constant-in-extended-predist-pos* [*simp*, *mono-intros*]:
  epsilonG(TYPE('a::Gromov-hyperbolic-space)) > 0
  epsilonG(TYPE('a::Gromov-hyperbolic-space)) ≥ 0
  ennreal (epsilonG(TYPE('a))) ∗ top = top
proof −
  have ∗: *2+2∗deltaG(TYPE('a)) ≥ 2 + 2 ∗ 0*
    by (*intro mono-intros*, *auto*)
  show ∗∗: *epsilonG(TYPE('a)) > 0*

unfolding *epsilonG-def* **apply** (*auto simp add*: *divide-simps*) **using** ∗ **by** *auto*
then show *ennreal* (*epsilonG*(*TYPE*(′*a*))) ∗ *top* = *top*
using *ennreal-mult-top* **by** *auto*
show *epsilonG*(*TYPE*(′*a*::*Gromov-hyperbolic-space*)) ≥ *0*
using ∗∗ **by** *simp*
qed

definition *extended-predist*::(′*a*::*Gromov-hyperbolic-space*) *Gromov-completion* ⇒ ′*a* *Gromov-completion* ⇒ *real*
where *extended-predist x y* = *real-of-ereal* (*min* (*esqrt* (*extended-Gromov-distance x y*))
(*eexp* (− *epsilonG*(*TYPE*(′*a*)) ∗ *extended-Gromov-product-at basepoint x y*)))

lemma *extended-predist-ereal*:
ereal (*extended-predist x* (*y*::(′*a*::*Gromov-hyperbolic-space*) *Gromov-completion*)) = *min* (*esqrt* (*extended-Gromov-distance x y*))
(*eexp* (− *epsilonG*(*TYPE*(′*a*)) ∗ *extended-Gromov-product-at basepoint x y*))
proof −
have *eexp* (− *epsilonG*(*TYPE*(′*a*)) ∗ *extended-Gromov-product-at basepoint x y*) ≤ *eexp* (*0*)
by (*intro mono-intros*, *simp add*: *ereal-mult-le-0-iff*)
then have *A*: *min* (*esqrt* (*extended-Gromov-distance x y*)) (*eexp* (− *epsilonG*(*TYPE*(′*a*)) ∗ *extended-Gromov-product-at basepoint x y*)) ≤ *1*
unfolding *min-def* **using** *order-trans* **by** *fastforce*
show *?thesis*
unfolding *extended-predist-def* **apply** (*rule ereal-real*′) **using** *A* **by** *auto*
qed

lemma *extended-predist-nonneg* [*simp*, *mono-intros*]:
extended-predist x y ≥ *0*
unfolding *extended-predist-def min-def* **by** (*auto intro*: *real-of-ereal-pos*)

lemma *extended-predist-commute*:
extended-predist x y = *extended-predist y x*
unfolding *extended-predist-def* **by** (*simp add*: *extended-Gromov-distance-commute extended-Gromov-product-at-commute*)

lemma *extended-predist-self0* [*simp*]:
extended-predist x y = *0* ⟷ *x* = *y*
proof (*auto*)
show *extended-predist y y* = *0*
proof (*cases y*)
case *boundary*
then have ∗: *extended-Gromov-product-at basepoint y y* = ∞
using *Gromov-boundary-extended-product-PInf* **by** *auto*
show *?thesis* **unfolding** *extended-predist-def* ∗ **apply** (*auto simp add*: *min-def*)
using *constant-in-extended-predist-pos*(*1*)[**where** *?*′*a* = ′*a*] *boundary* **by** *auto*

next
case (*to-Gromov-completion a*)
then show *?thesis* **unfolding** *extended-predist-def* **by** (*auto simp add*: *min-def*)
qed
assume *extended-predist x y* = *0*
then have *esqrt* (*extended-Gromov-distance x y*) = *0* ∨ *eexp* (− *epsilonG*(*TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint x y*) = *0*
by (*metis extended-predist-ereal min-def zero-ereal-def*)
then show *x* = *y*
proof
assume *esqrt* (*extended-Gromov-distance x y*) = *0*
then have ∗: *extended-Gromov-distance x y* = *0*
using *extended-Gromov-distance-nonneg* **by** (*metis ereal-zero-mult esqrt-square*)
then have ¬(*x* ∈ *Gromov-boundary*) ¬(*y* ∈ *Gromov-boundary*) **by** *auto*
then obtain *a b* **where** *ab*: *x* = *to-Gromov-completion a y* = *to-Gromov-completion b*
unfolding *Gromov-boundary-def* **by** *auto*
have *a* = *b* **using** ∗ **unfolding** *ab* **by** *auto*
then show *x* = *y* **using** *ab* **by** *auto*
next
assume *eexp* (− *epsilonG*(*TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint x y*) = *0*
then have *extended-Gromov-product-at basepoint x y* = ∞
by *auto*
then show *x* = *y*
using *Gromov-boundary-extended-product-PInf*[*of basepoint x y*] **by** *auto*
qed
qed

lemma *extended-predist-le1* [*simp*, *mono-intros*]:
extended-predist x y ≤ *1*
proof −
have *eexp* (− *epsilonG*(*TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint x y*) ≤ *eexp* (*0*)
by (*intro mono-intros*, *simp add*: *ereal-mult-le-0-iff*)
then have *min* (*esqrt* (*extended-Gromov-distance x y*)) (*eexp* (− *epsilonG*(*TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint x y*)) ≤ *1*
unfolding *min-def* **using** *order-trans* **by** *fastforce*
then show *?thesis*
unfolding *extended-predist-def* **by** (*simp add*: *real-of-ereal-le-1*)
qed

lemma *extended-predist-weak-triangle*:
extended-predist x z ≤ *sqrt 2* ∗ *max* (*extended-predist x y*) (*extended-predist y z*)
proof −
have *Z*: *esqrt 2* = *eexp* (*ereal*(*ln 2*/*2*))
by (*subst esqrt-eq-iff-square*, *auto simp add*: *exp-add*[*symmetric*])

have *A*: *eexp*(*ereal*(*epsilonG TYPE*(*'a*)) ∗ *1*) ≤ *esqrt 2*

unfolding *Z epsilonG-def* **apply** *auto*
apply (*auto simp add*: *algebra-simps divide-simps intro*!: *mono-intros*)
using *delta-nonneg*[**where** *?'a* = *'a*] **by** *auto*

We have to show an inequality $d(x, z) \leq \sqrt{2} \max(d(x, y), d(y, z))$. Each of $d(x, y)$ and $d(y, z)$ is either the extended distance, or the exponential of minus the Gromov product, depending on which is smaller. We split according to the four cases.

have (*esqrt* (*extended-Gromov-distance x y*) $\leq$ *eexp* ($-$ *epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint x y*)
$\vee$ *esqrt* (*extended-Gromov-distance x y*) $\geq$ *eexp* ($-$ *epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint x y*))
$\wedge$
((*esqrt* (*extended-Gromov-distance y z*) $\leq$ *eexp* ($-$ *epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint y z*)
$\vee$ *esqrt* (*extended-Gromov-distance y z*) $\geq$ *eexp* ($-$ *epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint y z*)))
by *auto*
then have *ereal*(*extended-predist x z*) $\leq$ *ereal* (*sqrt 2*) $*$ *max* (*ereal*(*extended-predist x y*)) (*ereal* (*extended-predist y z*))
proof (*auto*)

First, consider the case where the minimum is the extended distance for both cases. Then $ed(x, z) \leq ed(x, y) + ed(y, z) \leq 2 \max(ed(x, y), ed(y, z))$. Therefore, $ed(x, z)^{1/2} \leq \sqrt{2} \max(ed(x, y)^{1/2}, ed(y, z)^{1/2})$. As predist is defined using the square root of ed, this readily gives the result.

assume *H*: *esqrt* (*extended-Gromov-distance x y*) $\leq$ *eexp* (*ereal* ($-$ *epsilonG TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint x y*)
esqrt (*extended-Gromov-distance y z*) $\leq$ *eexp* (*ereal* ($-$ *epsilonG TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint y z*)
have *extended-Gromov-distance x z* $\leq$ *extended-Gromov-distance x y* $+$ *extended-Gromov-distance y z*
by (*rule extended-Gromov-distance-triangle*)
also have ... $\leq$ *2* $*$ *max* (*extended-Gromov-distance x y*) (*extended-Gromov-distance y z*)
by (*simp add*: *add-mono add-mono-thms-linordered-semiring*(*1*) *mult-2-ereal*)
finally have *esqrt* (*extended-Gromov-distance x z*) $\leq$ *esqrt* (*2* $*$ *max* (*extended-Gromov-distance x y*) (*extended-Gromov-distance y z*))
by (*intro mono-intros*)
also have ... = *esqrt 2* $*$ *max* (*esqrt* (*extended-Gromov-distance x y*)) (*esqrt* (*extended-Gromov-distance y z*))
by (*auto simp add*: *esqrt-mult max-of-mono*[*OF esqrt-mono*])
finally show *?thesis* **unfolding** *extended-predist-ereal min-def* **using** *H* **by** *auto*

next

Next, consider the case where the minimum comes from the Gromov product for both cases. Then, the conclusion will come for the hyperbolicity inequality (which is valid in the Gromov completion as well). There is an additive loss of δ in this inequality, which is converted to a multiplicative loss after taking the exponential to get the distance. Since, in the formula for the distance, the Gromov product is multiplied by a constant ϵ by design, the loss we get in the end is $\exp(\delta\epsilon)$. The precise value of ϵ we have taken is designed so that this is at most $\sqrt{2}$, giving the desired conclusion.

assume *H*: *eexp* (*ereal* (− *epsilonG TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint x y*) ≤ *esqrt* (*extended-Gromov-distance x y*)
eexp (*ereal* (− *epsilonG TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint y z*) ≤ *esqrt* (*extended-Gromov-distance y z*)

First, check that ϵ and δ satisfy the required inequality $\exp(\epsilon\delta) \leq \sqrt{2}$ (but in the extended reals as this is what we will use.

have *B*: *eexp* (*epsilonG*(*TYPE*(*'a*)) ∗ *deltaG*(*TYPE*(*'a*))) ≤ *esqrt 2*
unfolding *epsilonG-def* ‹*esqrt 2* = *eexp* (*ereal*(*ln 2*/*2*))›
apply (*auto simp add*: *algebra-simps divide-simps intro*!: *mono-intros*)
using *delta-nonneg*[**where** *?'a* = *'a*] **by** *auto*

We start the computation. First, use the hyperbolicity inequality.

have *eexp* (− *epsilonG TYPE*(*'a*) ∗ *extended-Gromov-product-at basepoint x z*)
≤ *eexp* (− *epsilonG TYPE*(*'a*) ∗ ((*min* (*extended-Gromov-product-at basepoint x y*) (*extended-Gromov-product-at basepoint y z*) − *deltaG*(*TYPE*(*'a*)))))
apply (*subst uminus-ereal.simps*(*1*)[*symmetric*], *subst ereal-mult-minus-left*)+
by (*intro mono-intros*)

Use distributivity to isolate the term $\epsilon\delta$. This requires some care as multiplication is not distributive in general in ereal.

also have ... = *eexp* (− *epsilonG TYPE*(*'a*) ∗ *min* (*extended-Gromov-product-at basepoint x y*) (*extended-Gromov-product-at basepoint y z*)
+ *epsilonG TYPE*(*'a*) ∗ *deltaG TYPE*(*'a*))
apply (*rule cong*[*of eexp*], *auto*)
apply (*subst times-ereal.simps*(*1*)[*symmetric*])
apply (*subst ereal-distrib-minus-left*, *auto*)
apply (*subst uminus-ereal.simps*(*1*)[*symmetric*])+
apply (*subst ereal-minus*(*6*))
by *simp*

Use multiplicativity of exponential to extract the multiplicative error factor.

also have ... = *eexp*(− *epsilonG TYPE*(*'a*) ∗ (*min* (*extended-Gromov-product-at basepoint x y*) (*extended-Gromov-product-at basepoint y z*)))
∗ *eexp*(*epsilonG*(*TYPE*(*'a*))∗ *deltaG*(*TYPE*(*'a*)))
by (*rule eexp-add-mult*, *auto*)

Extract the min outside of the exponential, using that all functions are monotonic.

also have ... = *eexp*(*epsilonG*(*TYPE*(*'a*))∗ *deltaG*(*TYPE*(*'a*)))

$*$ (*max* (*eexp*(− *epsilonG TYPE*(′*a*) $*$ *extended-Gromov-product-at basepoint x y*))
(*eexp*(− *epsilonG TYPE*(′*a*) $*$ *extended-Gromov-product-at basepoint y z*)))
apply (*subst max-of-antimono*[*of* λ (*t*::*ereal*). −*epsilonG TYPE*(′*a*) $*$ *t*, *symmetric*])
apply (*metis antimonoI constant-in-extended-predist-pos*(*2*) *enn2ereal-ennreal enn2ereal-nonneg ereal-minus-le-minus ereal-mult-left-mono ereal-mult-minus-left uminus-ereal.simps*(*1*))
apply (*subst max-of-mono*[*OF eexp-mono*])
apply (*simp add*: *mult.commute*)
done

We recognize the distance of x to y and the distance from y to z on the right.

also have ... = *eexp*(*epsilonG*(*TYPE*(′*a*)) $*$ *deltaG*(*TYPE*(′*a*))) $*$ (*max* (*ereal* (*extended-predist x y*)) (*extended-predist y z*))
unfolding *extended-predist-ereal min-def* **using** *H* **by** *auto*
also have ... ≤ *esqrt 2* $*$ *max* (*ereal*(*extended-predist x y*)) (*ereal*(*extended-predist y z*))
apply (*intro mono-intros B*) **using** *extended-predist-nonneg*[*of x y*] **by** (*simp add*: *max-def*)
finally show *?thesis* **unfolding** *extended-predist-ereal min-def* **by** *auto*

next

Next consider the case where $d(x, y)$ comes from the exponential of minus the Gromov product, but $d(y, z)$ comes from their extended distance. Then $d(y, z) \leq 1$ (as $d(y, z)$ is smaller then the exponential of minus the Gromov distance, which is at most 1), and this is all we use: the Gromov product between x and y or x and z differ by at most the distance from y to z, i.e., 1. Then the result follows directly as $\exp(\epsilon) \leq \sqrt{2}$, by the choice of ϵ.

assume *H*: *eexp* (− *epsilonG TYPE*(′*a*) $*$ *extended-Gromov-product-at basepoint x y*) ≤ *esqrt* (*extended-Gromov-distance x y*)
esqrt (*extended-Gromov-distance y z*) ≤ *eexp* (− *epsilonG TYPE*(′*a*) $*$ *extended-Gromov-product-at basepoint y z*)
then have *esqrt*(*extended-Gromov-distance y z*) ≤ *1*
by (*auto intro*!: *order-trans*[*OF H*(*2*)] *simp add*: *ereal-mult-le-0-iff*)
then have *extended-Gromov-distance y z* ≤ *1*
by (*metis eq-iff esqrt-mono2 esqrt-simps*(*2*) *esqrt-square extended-Gromov-distance-nonneg le-cases zero-less-one-ereal*)

have *ereal*(*extended-predist x z*) ≤ *eexp*(− *epsilonG TYPE*(′*a*) $*$ *extended-Gromov-product-at basepoint x z*)
unfolding *extended-predist-ereal min-def* **by** *auto*
also have ... ≤ *eexp*(− *epsilonG TYPE*(′*a*) $*$ *extended-Gromov-product-at basepoint x y*
\+ *epsilonG TYPE*(′*a*) $*$ *extended-Gromov-distance y z*)

apply (*intro mono-intros*) **apply** (*subst uminus-ereal.simps*(*1*)[*symmetric*])+
apply (*subst ereal-mult-minus-left*)+
apply (*intro mono-intros*)
using *extended-Gromov-product-at-diff3*[*of basepoint x y z*]
by (*meson constant-in-extended-predist-pos*(*2*) *ereal-le-distrib ereal-mult-left-mono order-trans zero-le-ereal*)
also have ... $\leq$ *eexp*($-$*epsilonG TYPE*($'a$) $*$ *extended-Gromov-product-at basepoint x y* + *ereal*(*epsilonG TYPE*($'a$)) $*$ *1*)
by (*intro mono-intros*, *fact*)
also have ... = *eexp*($-$*epsilonG TYPE*($'a$) $*$ *extended-Gromov-product-at basepoint x y*) $*$ *eexp*(*ereal*(*epsilonG TYPE*($'a$)) $*$ *1*)
by (*rule eexp-add-mult*, *auto*)
also have ... $\leq$ *eexp*($-$*epsilonG TYPE*($'a$) $*$ *extended-Gromov-product-at basepoint x y*) $*$ *esqrt 2*
by (*intro mono-intros A*)
also have ... = *esqrt 2* $*$ *ereal*(*extended-predist x y*)
unfolding *extended-predist-ereal min-def* **using** *H* **by** (*auto simp add*: *mult.commute*)
also have ... $\leq$ *esqrt 2* $*$ *max* (*ereal*(*extended-predist x y*)) (*ereal*(*extended-predist y z*))
unfolding *max-def* **by** (*auto intro*!: *mono-intros*)
finally show *?thesis* **by** *auto*

next

The last case is the symmetric of the previous one, and is proved similarly.

assume *H*: *eexp* ($-$ *epsilonG TYPE*($'a$) $*$ *extended-Gromov-product-at basepoint y z*) $\leq$ *esqrt* (*extended-Gromov-distance y z*)
esqrt (*extended-Gromov-distance x y*) $\leq$ *eexp* ($-$ *epsilonG TYPE*($'a$) $*$ *extended-Gromov-product-at basepoint x y*)
then have *esqrt*(*extended-Gromov-distance x y*) $\leq$ *1*
by (*auto intro*!: *order-trans*[*OF H*(*2*)] *simp add*: *ereal-mult-le-0-iff*)
then have *extended-Gromov-distance x y* $\leq$ *1*
by (*metis eq-iff esqrt-mono2 esqrt-simps*(*2*) *esqrt-square extended-Gromov-distance-nonneg le-cases zero-less-one-ereal*)

have *ereal*(*extended-predist x z*) $\leq$ *eexp*($-$ *epsilonG TYPE*($'a$) $*$ *extended-Gromov-product-at basepoint x z*)
unfolding *extended-predist-ereal min-def* **by** *auto*
also have ... $\leq$ *eexp*($-$ *epsilonG TYPE*($'a$) $*$ *extended-Gromov-product-at basepoint y z*
+ *epsilonG TYPE*($'a$) $*$ *extended-Gromov-distance x y*)
apply (*intro mono-intros*) **apply** (*subst uminus-ereal.simps*(*1*)[*symmetric*])+
apply (*subst ereal-mult-minus-left*)+
apply (*intro mono-intros*)
using *extended-Gromov-product-at-diff3*[*of basepoint z y x*]
apply (*simp add*: *extended-Gromov-product-at-commute extended-Gromov-distance-commute*)
by (*meson constant-in-extended-predist-pos*(*2*) *ereal-le-distrib ereal-mult-left-mono order-trans zero-le-ereal*)

also have ... $\leq$ *eexp*($-$*epsilonG TYPE*($'a$) $*$ *extended-Gromov-product-at basepoint y z* + *ereal*(*epsilonG TYPE*($'a$)) $*$ *1*)
by (*intro mono-intros*, *fact*)
also have ... = *eexp*($-$*epsilonG TYPE*($'a$) $*$ *extended-Gromov-product-at basepoint y z*) $*$ *eexp*(*ereal*(*epsilonG TYPE*($'a$)) $*$ *1*)
by (*rule eexp-add-mult*, *auto*)
also have ... $\leq$ *eexp*($-$*epsilonG TYPE*($'a$) $*$ *extended-Gromov-product-at basepoint y z*) $*$ *esqrt 2*
by (*intro mono-intros A*)
also have ... = *esqrt 2* $*$ *ereal*(*extended-predist y z*)
unfolding *extended-predist-ereal min-def* **using** *H* **by** (*auto simp add*: *mult.commute*)
also have ... $\leq$ *esqrt 2* $*$ *max* (*ereal*(*extended-predist x y*)) (*ereal*(*extended-predist y z*))
unfolding *max-def* **by** (*auto intro*!: *mono-intros*)
finally show *?thesis* **by** *auto*
qed
then show *extended-predist x z* $\leq$ *sqrt 2* $*$ *max* (*extended-predist x y*) (*extended-predist y z*)
unfolding *ereal-sqrt2*[*symmetric*] *max-of-mono*[*OF ereal-mono*] *times-ereal.simps*(*1*) **by** *auto*
qed

instantiation *Gromov-completion* :: (*Gromov-hyperbolic-space*) *metric-space*
begin

definition *dist-Gromov-completion*::($'a$::*Gromov-hyperbolic-space*) *Gromov-completion* $\Rightarrow$ $'a$ *Gromov-completion* $\Rightarrow$ *real*
where *dist-Gromov-completion* = *turn-into-distance extended-predist*

To define a metric space in the current library of Isabelle/HOL, one should also introduce a uniformity structure and a topology, as follows (they are prescribed by the distance):

definition *uniformity-Gromov-completion*::(($'a$ *Gromov-completion*) $\times$ ($'a$ *Gromov-completion*)) *filter*
where *uniformity-Gromov-completion* = (*INF* $e\in\{0 <..\}$. *principal* $\{(x, y).$ *dist* $x\ y < e\}$)

definition *open-Gromov-completion* :: $'a$ *Gromov-completion set* $\Rightarrow$ *bool*
where *open-Gromov-completion U* = ($\forall x\in U$. *eventually* ($\lambda(x', y)$. $x' = x \longrightarrow y \in U$) *uniformity*)

instance proof
interpret *Turn-into-distance extended-predist*
by (*standard*, *auto intro*: *extended-predist-weak-triangle extended-predist-commute*)
fix *x y z*::$'a$ *Gromov-completion*
show (*dist x y* = *0*) = ($x = y$)
using *TID-nonneg*[*of x y*] *lower*[*of x y*] *TID-self-zero upper*[*of x y*] *extended-predist-self0*[*of x y*] **unfolding** *dist-Gromov-completion-def*

by (*auto*, *linarith*)
show *dist x y* $\leq$ *dist x z* + *dist y z*
unfolding *dist-Gromov-completion-def* **using** *triangle* **by** (*simp add*: *TID-sym*)
qed (*auto simp add*: *uniformity-Gromov-completion-def open-Gromov-completion-def*)
end

The only relevant property of the distance on the Gromov completion is that it is comparable to the minimum of (the square root of) the extended distance, and the exponential of minus the Gromov product. The precise formula we use to define it is just an implementation detail, in a sense. We summarize these properties in the next theorem. From this point on, we will only use this, and never come back to the definition based on `extended_predist` and `turn_into_distance`.

theorem *Gromov-completion-dist-comparison* [*mono-intros*]:
fixes *x y*::(*'a*::*Gromov-hyperbolic-space*) *Gromov-completion*
shows *ereal*(*dist x y*) $\leq$ *esqrt*(*extended-Gromov-distance x y*)
ereal(*dist x y*) $\leq$ *eexp* (− *epsilonG*(*TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint x y*)
min (*esqrt*(*extended-Gromov-distance x y*)) (*eexp* (− *epsilonG*(*TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint x y*)) $\leq$ *2* ∗ *ereal*(*dist x y*)
proof −
interpret *Turn-into-distance extended-predist*
by (*standard*, *auto intro*: *extended-predist-weak-triangle extended-predist-commute*)
have *ereal*(*dist x y*) $\leq$ *ereal*(*extended-predist x y*)
unfolding *dist-Gromov-completion-def* **by** (*auto intro*!: *upper mono-intros*)
then show *ereal*(*dist x y*) $\leq$ *esqrt*(*extended-Gromov-distance x y*)
ereal(*dist x y*) $\leq$ *eexp* (− *epsilonG*(*TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint x y*)
unfolding *extended-predist-ereal* **by** *auto*
have *ereal*(*extended-predist x y*) $\leq$ *ereal*(*2* ∗ *dist x y*)
unfolding *dist-Gromov-completion-def* **by** (*auto intro*!: *lower mono-intros*)
also have ... = *2* ∗ *ereal* (*dist x y*)
by *simp*
finally show *min* (*esqrt*(*extended-Gromov-distance x y*)) (*eexp* (− *epsilonG*(*TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint x y*)) $\leq$ *2* ∗ *ereal*(*dist x y*)
unfolding *extended-predist-ereal* **by** *auto*
qed

lemma *Gromov-completion-dist-le-1* [*simp*, *mono-intros*]:
fixes *x y*::(*'a*::*Gromov-hyperbolic-space*) *Gromov-completion*
shows *dist x y* $\leq$ *1*
proof −
have *ereal*(*dist x y*) $\leq$ *eexp* (− *epsilonG*(*TYPE*(*'a*)) ∗ *extended-Gromov-product-at basepoint x y*)
using *Gromov-completion-dist-comparison*(*2*)[*of x y*] **by** *simp*
also have ... $\leq$ *eexp*(−*0*)
by (*intro mono-intros*) (*simp add*: *ereal-mult-le-0-iff*)
finally show *?thesis*

by *auto*
qed

To avoid computations with exponentials, the following lemma is very convenient. It asserts that if x is close enough to infinity, and y is close enough to x, then the Gromov product between x and y is large.

lemma *large-Gromov-product-approx*:
assumes $(M::ereal) < \infty$
shows $\exists\, e\, D.\ e > 0 \wedge D < \infty \wedge (\forall\, x\, y.\ dist\ x\ y \leq e \longrightarrow extended\text{-}Gromov\text{-}distance\ x\ (to\text{-}Gromov\text{-}completion\ basepoint) \geq D \longrightarrow extended\text{-}Gromov\text{-}product\text{-}at\ basepoint\ x\ y \geq M)$
proof −
obtain $M0::real$ **where** $M \leq ereal\ M0$ **using** *assms* **by** (*cases M*, *auto*)
define $e::real$ **where** $e = exp(-epsilonG(TYPE('a)) * M0)/2$
define $D::ereal$ **where** $D = ereal\ M0 + 4$
have $e > 0$
unfolding *e-def* **by** *auto*
moreover have $D < \infty$
unfolding *D-def* **by** *auto*
moreover have $extended\text{-}Gromov\text{-}product\text{-}at\ basepoint\ x\ y \geq M0$
if $dist\ x\ y \leq e\ extended\text{-}Gromov\text{-}distance\ x\ (to\text{-}Gromov\text{-}completion\ basepoint) \geq D$ **for** $x\ y::'a\ Gromov\text{-}completion$
proof (*cases* $esqrt(extended\text{-}Gromov\text{-}distance\ x\ y) \leq eexp\ (-\ epsilonG(TYPE('a)) * extended\text{-}Gromov\text{-}product\text{-}at\ basepoint\ x\ y)$)
case *False*
then have $eexp\ (-\ epsilonG(TYPE('a)) * extended\text{-}Gromov\text{-}product\text{-}at\ basepoint\ x\ y) \leq 2 * ereal(dist\ x\ y)$
using *Gromov-completion-dist-comparison*(*3*)[*of x y*] **unfolding** *min-def* **by** *auto*
also have ... $\leq exp(-epsilonG(TYPE('a)) * M0)$
using ‹$dist\ x\ y \leq e$› **unfolding** *e-def* **by** (*auto simp add*: *numeral-mult-ennreal*)
finally have $ereal\ M0 \leq extended\text{-}Gromov\text{-}product\text{-}at\ basepoint\ x\ y$
unfolding *eexp-ereal*[*symmetric*] **apply** (*simp only*: *eexp-le-eexp-iff-le*)
unfolding *times-ereal.simps*(*1*)[*symmetric*] *uminus-ereal.simps*(*1*)[*symmetric*] *ereal-mult-minus-left ereal-minus-le-minus*
using *ereal-mult-le-mult-iff*[*of ereal* (*epsilonG TYPE*('*a*))] **apply** *auto*
by (*metis* ‹$\bigwedge r\ p.\ ereal\ (r * p) = ereal\ r * ereal\ p$›)
then show $M0 \leq extended\text{-}Gromov\text{-}product\text{-}at\ basepoint\ x\ y$
by *auto*
next
case *True*
then have $esqrt(extended\text{-}Gromov\text{-}distance\ x\ y) \leq 2 * ereal(dist\ x\ y)$
using *Gromov-completion-dist-comparison*(*3*)[*of x y*] **unfolding** *min-def* **by** *auto*
also have ... $\leq esqrt\ 4$
by *simp*
finally have $*$: $extended\text{-}Gromov\text{-}distance\ x\ y \leq 4$
unfolding *esqrt-le* **using** *antisym* **by** *fastforce*
have $ereal\ M0{+}4 \leq D$

unfolding *D-def* **by** *auto*
also have ... $\leq$ *extended-Gromov-product-at basepoint x x*
using *that* **by** (*auto simp add*: *extended-Gromov-distance-commute*)
also have ... $\leq$ *extended-Gromov-product-at basepoint x y* + *extended-Gromov-distance x y*
by (*rule extended-Gromov-product-at-diff3*[*of basepoint x x y*])
also have ... $\leq$ *extended-Gromov-product-at basepoint x y* + *4*
by (*intro mono-intros* $*$)
finally show *M0* $\leq$ *extended-Gromov-product-at basepoint x y*
by (*metis* (*no-types*, *lifting*) *PInfty-neq-ereal*(*1*) *add.commute add-nonneg-nonneg ereal-add-strict-mono ereal-le-distrib mult-2-ereal not-le numeral-Bit0 numeral-eq-ereal one-add-one zero-less-one-ereal*)
qed
ultimately show *?thesis* **using** *order-trans*[*OF* ‹*M* $\leq$ *ereal M0*›] **by** *force*
qed

On the other hand, far away from infinity, it is equivalent to control the extended Gromov distance or the new distance on the space.

lemma *inside-Gromov-distance-approx*:
assumes $C < (\infty::ereal)$
shows $\exists$ *e* > *0*. $\forall$ *x y*. *extended-Gromov-distance* (*to-Gromov-completion basepoint*) *x* $\leq$ *C* $\longrightarrow$ *dist x y* $\leq$ *e*
$\longrightarrow$ *esqrt*(*extended-Gromov-distance x y*) $\leq$ *2* $*$ *ereal*(*dist x y*)
proof −
obtain *C0* **where** *C* $\leq$ *ereal C0* **using** *assms* **by** (*cases C*, *auto*)
define *e0* **where** *e0* = *exp*(−*epsilonG*(*TYPE*(*'a*)) $*$ *C0*)
have *e0* > *0*
unfolding *e0-def* **using** *assms* **by** *auto*
define *e* **where** *e* = *e0*/*4*
have *e* > *0*
unfolding *e-def* **using** ‹*e0* > *0*› **by** *auto*
moreover have *esqrt*(*extended-Gromov-distance x y*) $\leq$ *2* $*$ *ereal*(*dist x y*)
if *extended-Gromov-distance* (*to-Gromov-completion basepoint*) *x* $\leq$ *C0 dist x y* $\leq$ *e* **for** *x y*::*'a Gromov-completion*
proof −
have *R*: *min a b* $\leq$ *c* $\Longrightarrow$ *a* $\leq$ *c* $\vee$ *b* $\leq$ *c* **for** *a b c*::*ereal* **unfolding** *min-def*
by *presburger*
have *2* $*$ *ereal* (*dist x y*) $\leq$ *2* $*$ *ereal e*
using *that* **by** (*intro mono-intros*, *auto*)
also have ... = *ereal*(*e0*/*2*)
unfolding *e-def* **by** *auto*
also have ... < *ereal e0*
apply (*intro mono-intros*) **using** ‹*e0* > *0*› **by** *auto*
also have ... $\leq$ *eexp*(−*epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-distance* (*to-Gromov-completion basepoint*) *x*)
unfolding *e0-def eexp-ereal*[*symmetric*] *ereal-mult-minus-left mult-minus-left uminus-ereal.simps*(*1*)[*symmetric*] *times-ereal.simps*(*1*)[*symmetric*]
by (*intro mono-intros that*)
also have ... $\leq$ *eexp*(−*epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at base-*

point x y)
unfolding *ereal-mult-minus-left mult-minus-left uminus-ereal.simps*(*1*)[*symmetric*] *times-ereal.simps*(*1*)[*symmetric*]
by (*intro mono-intros*)
finally show *?thesis*
using *R*[*OF Gromov-completion-dist-comparison*(*3*)[*of x y*]] **by** *auto*
qed
ultimately show *?thesis* **using** *order-trans*[*OF - ‹C ≤ ereal C0›*] **by** *auto*
qed

15.4 Characterizing convergence in the Gromov boundary

The convergence of sequences in the Gromov boundary can be characterized, essentially by definition: sequences tend to a point at infinity iff the Gromov product with this point tends to infinity, while sequences tend to a point inside iff the extended distance tends to 0. In both cases, it is just a matter of unfolding the definition of the distance, and see which one of the two terms (exponential of minus the Gromov product, or extended distance) realizes the minimum. We have constructed the distance essentially so that this property is satisfied.

We could also have defined first the topology, satisfying these conditions, but then we would have had to check that it coincides with the topology that the distance defines, so it seems more economical to proceed in this way.

lemma *Gromov-completion-boundary-limit*:
assumes $x \in$ *Gromov-boundary*
shows ($u \longrightarrow x$) $F \longleftrightarrow$ ((λn. *extended-Gromov-product-at basepoint* (u n) x) $\longrightarrow \infty$) F
proof
assume $*$: ((λn. *extended-Gromov-product-at basepoint* (u n) x) $\longrightarrow \infty$) F
have ((λn. *ereal*(*dist* (u n) x)) $\longrightarrow$ *0*) F
proof (*rule tendsto-sandwich*[*of* λ-. *0* - - (λn. *eexp* ($-$*epsilonG*(*TYPE*('a)) $*$ *extended-Gromov-product-at basepoint* (u n) x))])
have ((λn. *eexp* ($-$ *epsilonG*(*TYPE*('a)) $*$ *extended-Gromov-product-at basepoint* (u n) x)) $\longrightarrow$ *eexp* ($-$ *epsilonG*(*TYPE*('a)) $*$ (∞::*ereal*))) F
apply (*intro tendsto-intros* $*$) **by** *auto*
then show ((λn. *eexp* ($-$*epsilonG*(*TYPE*('a)) $*$ *extended-Gromov-product-at basepoint* (u n) x)) $\longrightarrow$ *0*) F
using *constant-in-extended-predist-pos*(*1*)[**where** *?'a* = *'a*] **by** *auto*
qed (*auto simp add*: *Gromov-completion-dist-comparison*)
then have ((λn. *real-of-ereal*(*ereal*(*dist* (u n) x))) $\longrightarrow$ *0*) F
by (*simp add*: *zero-ereal-def*)
then show ($u \longrightarrow x$) F
by (*subst tendsto-dist-iff*, *auto*)
next
assume $*$: ($u \longrightarrow x$) F
have *A*: *1* / *ereal* ($-$ *epsilonG TYPE*('a)) $*$ (*ereal* ($-$ *epsilonG TYPE*('a))) =

1
apply *auto* **using** *constant-in-extended-predist-pos*(*1*)[**where** *?'a* = *'a*] **by** *auto*
have *a*: *esqrt*(*extended-Gromov-distance* (*u n*) *x*) = ∞ **for** *n*
unfolding *extended-Gromov-distance-PInf-boundary*(*2*)[*OF assms*, *of u n*] **by** *auto*
have *min* (*esqrt*(*extended-Gromov-distance* (*u n*) *x*)) (*eexp* ($-$ *epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint* (*u n*) *x*))
= *eexp* ($-$ *epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint* (*u n*) *x*) **for** *n*
unfolding *a min-def* **using** *neq-top-trans* **by** *force*
moreover have ((λ*n*. *min* (*esqrt*(*extended-Gromov-distance* (*u n*) *x*)) (*eexp* ($-$ *epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint* (*u n*) *x*))) $\longrightarrow$ *0*) *F*
proof (*rule tendsto-sandwich*[*of* λ-. *0* - - λ*n*. *2* $*$ *ereal*(*dist* (*u n*) *x*)])
have ((λ*n*. *2* $*$ *ereal* (*dist* (*u n*) *x*)) $\longrightarrow$ *2* $*$ *ereal 0*) *F*
apply (*intro tendsto-intros*) **using** $*$ *tendsto-dist-iff* **by** *auto*
then show ((λ*n*. *2* $*$ *ereal* (*dist* (*u n*) *x*)) $\longrightarrow$ *0*) *F* **by** (*simp add*: *zero-ereal-def*)
show $\forall_F$ *n in F*. *0* $\leq$ *min* (*esqrt* (*extended-Gromov-distance* (*u n*) *x*)) (*eexp* (*ereal* ($-$ *epsilonG TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint* (*u n*) *x*))
by (*rule always-eventually*, *auto*)
show $\forall_F$ *n in F*.
min (*esqrt* (*extended-Gromov-distance* (*u n*) *x*)) (*eexp* (*ereal* ($-$ *epsilonG TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint* (*u n*) *x*)) $\leq$ *2* $*$ *ereal* (*dist* (*u n*) *x*)
apply (*rule always-eventually*) **using** *Gromov-completion-dist-comparison*(*3*) **by** *auto*
qed (*auto*)
ultimately have ((λ*n*. *eexp* ($-$ *epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint* (*u n*) *x*)) $\longrightarrow$ *0*) *F*
by *auto*
then have ((λ*n*. $-$ *epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint* (*u n*) *x*) $\longrightarrow$ $-\infty$) *F*
unfolding *eexp-special-values*(*3*)[*symmetric*] *eexp-tendsto'* **by** *auto*
then have ((λ*n*. *1*/*ereal*($-$*epsilonG*(*TYPE*(*'a*))) $*$ ($-$ *epsilonG*(*TYPE*(*'a*)) $*$ *extended-Gromov-product-at basepoint* (*u n*) *x*)) $\longrightarrow$ *1*/*ereal*($-$*epsilonG*(*TYPE*(*'a*))) $*$ ($-\infty$)) *F*
by (*intro tendsto-intros*, *auto*)
moreover have *1*/*ereal*($-$*epsilonG*(*TYPE*(*'a*))) $*$ ($-\infty$) = ∞
apply *auto* **using** *constant-in-extended-predist-pos*(*1*)[**where** *?'a* = *'a*] **by** *auto*
ultimately show ((λ*n*. *extended-Gromov-product-at basepoint* (*u n*) *x*) $\longrightarrow$ ∞) *F*
unfolding *ab-semigroup-mult-class.mult-ac*(*1*)[*symmetric*] *A* **by** *auto*
qed

lemma *extended-Gromov-product-tendsto-PInf-a-b*:
assumes ((λ*n*. *extended-Gromov-product-at a* (*u n*) (*v n*)) $\longrightarrow$ ∞) *F*
shows ((λ*n*. *extended-Gromov-product-at b* (*u n*) (*v n*)) $\longrightarrow$ ∞) *F*
proof (*rule tendsto-sandwich*[*of* λ*n*. *extended-Gromov-product-at a* (*u n*) (*v n*) $-$ *dist a b* - - λ-. ∞])

have *extended-Gromov-product-at a* $(u\ n)$ $(v\ n)$ $-$ *ereal* (*dist a b*) $\leq$ *extended-Gromov-product-at b* $(u\ n)$ $(v\ n)$ **for** *n*
using *extended-Gromov-product-at-diff1*[*of a u n v n b*] **by** (*simp add*: *add.commute ereal-minus-le-iff*)
then show $\forall_F$ *n in F*. *extended-Gromov-product-at a* $(u\ n)$ $(v\ n)$ $-$ *ereal* (*dist a b*) $\leq$ *extended-Gromov-product-at b* $(u\ n)$ $(v\ n)$
by *auto*
have ((λn. *extended-Gromov-product-at a* $(u\ n)$ $(v\ n)$ $-$ *ereal* (*dist a b*)) $\longrightarrow$ ∞ $-$ *ereal* (*dist a b*)) *F*
by (*intro tendsto-intros assms*) *auto*
then show ((λn. *extended-Gromov-product-at a* $(u\ n)$ $(v\ n)$ $-$ *ereal* (*dist a b*)) $\longrightarrow$ ∞) *F*
by *auto*
qed (*auto*)

lemma *Gromov-completion-inside-limit*:
assumes $x \notin$ *Gromov-boundary*
shows ($u \longrightarrow x$) $F \longleftrightarrow$ ((λn. *extended-Gromov-distance* $(u\ n)$ x) $\longrightarrow$ *0*) *F*
proof
assume $*$: ((λn. *extended-Gromov-distance* $(u\ n)$ x) $\longrightarrow$ *0*) *F*
have ((λn. *ereal*(*dist* $(u\ n)$ x)) $\longrightarrow$ *ereal 0*) *F*
proof (*rule tendsto-sandwich*[*of* λ-. *0* - - λn. *esqrt* (*extended-Gromov-distance* $(u\ n)$ x)])
have ((λn. *esqrt* (*extended-Gromov-distance* $(u\ n)$ x)) $\longrightarrow$ *esqrt 0*) *F*
by (*intro tendsto-intros* $*$)
then show ((λn. *esqrt* (*extended-Gromov-distance* $(u\ n)$ x)) $\longrightarrow$ *ereal 0*) *F*
by (*simp add*: *zero-ereal-def*)
qed (*auto simp add*: *Gromov-completion-dist-comparison zero-ereal-def*)
then have ((λn. *real-of-ereal*(*ereal*(*dist* $(u\ n)$ x))) $\longrightarrow$ *0*) *F*
by (*intro lim-real-of-ereal*)
then show ($u \longrightarrow x$) *F*
by (*subst tendsto-dist-iff*, *auto*)
next
assume $*$: ($u \longrightarrow x$) *F*
have $x \in$ *range to-Gromov-completion* **using** *assms* **unfolding** *Gromov-boundary-def* **by** *auto*
have ((λn. *esqrt*(*extended-Gromov-distance* $(u\ n)$ x)) $\longrightarrow$ *0*) *F*
proof (*rule tendsto-sandwich*[*of* λ-. *0* - - λn. *2* $*$ *ereal*(*dist* $(u\ n)$ x)])
have *A*: *extended-Gromov-distance* (*to-Gromov-completion basepoint*) $x < \infty$
by (*simp add*: *assms extended-Gromov-distance-def*)
obtain *e* **where** *e*: $e > 0$ $\bigwedge y$. *dist x y* $\leq e \Longrightarrow$ *esqrt*(*extended-Gromov-distance x y*) $\leq$ *2* $*$ *ereal* (*dist x y*)
using *inside-Gromov-distance-approx*[*OF A*] **by** *auto*
have *B*: *eventually* (λn. *dist x* $(u\ n)$ $< e$) *F*
using *order-tendstoD*(*2*)[*OF iffD1*[*OF tendsto-dist-iff* $*$] ‹$e > 0$›] **by** (*simp add*: *dist-commute*)
then have *eventually* (λn. *esqrt*(*extended-Gromov-distance x* $(u\ n)$) $\leq$ *2* $*$ *ereal* (*dist x* $(u\ n)$)) *F*
using *eventually-mono*[*OF* - *e*(*2*)] *less-imp-le* **by** (*metis* (*mono-tags*, *lifting*))

then show *eventually* ($\lambda n.$ *esqrt*(*extended-Gromov-distance* (*u n*) *x*) $\leq$ *2* $*$ *ereal* (*dist* (*u n*) *x*)) *F*
by (*simp add*: *dist-commute extended-Gromov-distance-commute*)
have (($\lambda n.$ *2* $*$ *ereal*(*dist* (*u n*) *x*)) $\longrightarrow$ *2* $*$ *ereal 0*) *F*
apply (*intro tendsto-intros*) **using** *tendsto-dist-iff* $*$ **by** *auto*
then show (($\lambda n.$ *2* $*$ *ereal*(*dist* (*u n*) *x*)) $\longrightarrow$ *0*) *F*
by (*simp add*: *zero-ereal-def*)
qed (*auto*)
then have (($\lambda n.$ *esqrt*(*extended-Gromov-distance* (*u n*) *x*) $*$ *esqrt*(*extended-Gromov-distance* (*u n*) *x*)) $\longrightarrow$ *0* $*$ *0*) *F*
by (*intro tendsto-intros*, *auto*)
then show (($\lambda n.$ *extended-Gromov-distance* (*u n*) *x*) $\longrightarrow$ *0*) *F*
by *auto*
qed

lemma *to-Gromov-completion-lim* [*simp*, *tendsto-intros*]:
(($\lambda n.$ *to-Gromov-completion* (*u n*)) $\longrightarrow$ *to-Gromov-completion a*) *F* $\longleftrightarrow$ (*u* $\longrightarrow$ *a*) *F*
proof (*subst Gromov-completion-inside-limit*, *auto*)
assume (($\lambda n.$ *ereal* (*dist* (*u n*) *a*)) $\longrightarrow$ *0*) *F*
then have (($\lambda n.$ *real-of-ereal*(*ereal* (*dist* (*u n*) *a*))) $\longrightarrow$ *0*) *F*
unfolding *zero-ereal-def* **by** (*rule lim-real-of-ereal*)
then show (*u* $\longrightarrow$ *a*) *F*
by (*subst tendsto-dist-iff*, *auto*)
next
assume (*u* $\longrightarrow$ *a*) *F*
then have (($\lambda n.$ *dist* (*u n*) *a*) $\longrightarrow$ *0*) *F*
using *tendsto-dist-iff* **by** *auto*
then show (($\lambda n.$ *ereal* (*dist* (*u n*) *a*)) $\longrightarrow$ *0*) *F*
unfolding *zero-ereal-def* **by** (*intro tendsto-intros*)
qed

Now, we can also come back to our original definition of the completion, where points on the boundary correspond to equivalence classes of sequences whose mutual Gromov product tends to infinity. We show that this is compatible with our topology: the sequences that are in the equivalence class of a point on the boundary are exactly the sequences that converge to this point. This is also a direct consequence of the definitions, although the proof requires some unfolding (and playing with the hyperbolicity inequality several times).

First, we show that a sequence in the equivalence class of x converges to x.

lemma *Gromov-completion-converge-to-boundary-aux*:
assumes $x \in$ *Gromov-boundary abs-Gromov-completion* $v = x$ *Gromov-completion-rel v v*
shows ($\lambda n.$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (*v n*)) *x*) $\longrightarrow \infty$
proof –

have A: *eventually* ($\lambda n.$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (v n)) $x \geq$ *ereal* M) *sequentially* **for** M
proof −
have *Gromov-converging-at-boundary v*
using *Gromov-boundary-abs-converging assms* **by** *blast*
then obtain N **where** N: $\bigwedge m\ n.\ m \geq N \Longrightarrow n \geq N \Longrightarrow$ *Gromov-product-at basepoint* (v m) (v n) $\geq M$ + *deltaG*(*TYPE*('*a*))
unfolding *Gromov-converging-at-boundary-def* **by** *metis*
have *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (v n)) $x \geq$ *ereal* M **if** $n \geq N$ **for** n
unfolding *extended-Gromov-product-at-def* **proof** (*rule Inf-greatest*, *auto*)
fix *wv wx* **assume** H: *abs-Gromov-completion wv* = *to-Gromov-completion* (v n)
x = *abs-Gromov-completion wx*
Gromov-completion-rel wv wv Gromov-completion-rel wx wx
then have *wv*: *wv* p = v n **for** p
using *Gromov-completion-rel-to-const Quotient3-Gromov-completion Quotient3-rel to-Gromov-completion-def* **by** *fastforce*
have *Gromov-completion-rel v wx*
using *assms H Quotient3-rel*[*OF Quotient3-Gromov-completion*] **by** *auto*
then have $*$: ($\lambda p.$ *Gromov-product-at basepoint* (v p) (*wx* p)) $\longrightarrow \infty$
unfolding *Gromov-completion-rel-def* **using** *Gromov-converging-at-boundary-imp-not-constant'* ‹*Gromov-converging-at-boundary v*› **by** *auto*
have *eventually* ($\lambda p.$ *ereal*(*Gromov-product-at basepoint* (v p) (*wx* p)) $>$ M + *deltaG*(*TYPE*('*a*))) *sequentially*
using *order-tendstoD*[*OF* $*$, *of ereal* (M + *deltaG TYPE*('*a*))] **by** *auto*
then obtain P **where** P: $\bigwedge p.\ p \geq P \Longrightarrow$ *ereal*(*Gromov-product-at basepoint* (v p) (*wx* p)) $>$ M + *deltaG*(*TYPE*('*a*))
unfolding *eventually-sequentially* **by** *auto*
have $*$: *ereal* (*Gromov-product-at basepoint* (v n) (*wx* p)) $\geq$ *ereal* M **if** $p \geq$ *max* P N **for** p
proof (*intro mono-intros*)
have $M \leq$ *min* (M + *deltaG*(*TYPE*('*a*))) (M + *deltaG*(*TYPE*('*a*))) − *deltaG*(*TYPE*('*a*))
by *auto*
also have ... $\leq$ *min* (*Gromov-product-at basepoint* (v n) (v p)) (*Gromov-product-at basepoint* (v p) (*wx* p)) − *deltaG*(*TYPE*('*a*))
apply (*intro mono-intros*)
using N[*OF* ‹$n \geq N$›, *of* p] ‹$p \geq$ *max* P N› P[*of* p] ‹$p \geq$ *max* P N› **by** *auto*
also have ... $\leq$ *Gromov-product-at basepoint* (v n) (*wx* p)
by (*rule hyperb-ineq*)
finally show $M \leq$ *Gromov-product-at basepoint* (v n) (*wx* p)
by *simp*
qed
then have *eventually* ($\lambda p.$ *ereal* (*Gromov-product-at basepoint* (v n) (*wx* p)) $\geq$ *ereal* M) *sequentially*
unfolding *eventually-sequentially* **by** *metis*
then show *ereal* $M \leq$ *liminf* ($\lambda p.$ *ereal* (*Gromov-product-at basepoint* (*wv* p)

$(wx\ p)))$
unfolding *wv* **by** (*simp add*: *Liminf-bounded*)
qed
then show *?thesis* **unfolding** *eventually-sequentially* **by** *auto*
qed
have *B*: *eventually* ($\lambda n.$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* $(v\ n))\ x > M)$ *sequentially* **if** $M < \infty$ **for** M
proof −
obtain N **where** *ereal* $N > M$ **using** ‹$M < \infty$› *ereal-dense2* **by** *blast*
then have $a \geq$ *ereal* $N \Longrightarrow a > M$ **for** a **by** *auto*
then show *?thesis* **using** $A[of\ N]$ *eventually-elim2* **by** *force*
qed
then show *?thesis*
by (*rule order-tendstoI*, *auto*)
qed

Then, we prove the converse and therefore the equivalence.

lemma *Gromov-completion-converge-to-boundary*:
assumes $x \in$ *Gromov-boundary*
shows (($\lambda n.$ *to-Gromov-completion* $(u\ n)) \longrightarrow x) \longleftrightarrow$ (*Gromov-completion-rel* $u\ u \wedge$ *abs-Gromov-completion* $u = x$)
proof
assume *Gromov-completion-rel* $u\ u \wedge$ *abs-Gromov-completion* $u = x$
then show (($\lambda n.$ *to-Gromov-completion*$(u\ n)) \longrightarrow x)$
using *Gromov-completion-converge-to-boundary-aux*[*OF assms*, *of u*] **unfolding** *Gromov-completion-boundary-limit*[*OF assms*] **by** *auto*
next
assume *H*: ($\lambda n.$ *to-Gromov-completion* $(u\ n)) \longrightarrow x$
have *Lu*: ($\lambda n.$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* $(u\ n))\ x) \longrightarrow \infty$
using *iffD1*[*OF Gromov-completion-boundary-limit*[*OF assms*] *H*] **by** *simp*
have *A*: $\exists N.\ \forall n \geq N.\ \forall\ m \geq N.$ *Gromov-product-at basepoint* $(u\ m)\ (u\ n) \geq M$ **for** M
proof −
have *eventually* ($\lambda n.$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* $(u\ n))\ x > M +$ *deltaG*(*TYPE*('*a*))) *sequentially*
by (*rule order-tendstoD*[*OF Lu*], *auto*)
then obtain N **where** N: $\bigwedge n.\ n \geq N \Longrightarrow$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* $(u\ n))\ x > M +$ *deltaG*(*TYPE*('*a*))
unfolding *eventually-sequentially* **by** *auto*
have *Gromov-product-at basepoint* $(u\ m)\ (u\ n) \geq M$ **if** $n \geq N\ m \geq N$ **for** $m\ n$
proof −
have *ereal* $M \leq$ *min* (*ereal* ($M +$ *deltaG*(*TYPE*('*a*)))) (*ereal* ($M +$ *deltaG*(*TYPE*('*a*)))) − *ereal*(*deltaG*(*TYPE*('*a*)))
by *simp*
also have ... $\leq$ *min* (*extended-Gromov-product-at basepoint* (*to-Gromov-completion* $(u\ m))\ x)$ (*extended-Gromov-product-at basepoint* x (*to-Gromov-completion* $(u\ n)))$ − *deltaG*(*TYPE*('*a*))
apply (*intro mono-intros*) **using** $N[OF$ ‹$n \geq N$›$]$ $N[OF$ ‹$m \geq N$›$]$

by (*auto simp add*: *extended-Gromov-product-at-commute*)
also have ... $\leq$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (*u m*)) (*to-Gromov-completion* (*u n*))
by (*rule extended-hyperb-ineq*)
finally show *?thesis* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed
have $\exists N. \forall n \geq N. \forall\ m \geq N.$ *Gromov-product-at a* (*u m*) (*u n*) $\geq M$ **for** *M a*
proof −
obtain *N* **where** *N*: $\bigwedge m\ n.\ m \geq N \Longrightarrow n \geq N \Longrightarrow$ *Gromov-product-at basepoint* (*u m*) (*u n*) $\geq M +$ *dist a basepoint*
using *A*[*of M* + *dist a basepoint*] **by** *auto*
have *Gromov-product-at a* (*u m*) (*u n*) $\geq M$ **if** $m \geq N$ $n \geq N$ **for** *m n*
using *N*[*OF that*] *Gromov-product-at-diff1*[*of a u m u n basepoint*] **by** *auto*
then show *?thesis* **by** *auto*
qed
then have *Gromov-converging-at-boundary u*
unfolding *Gromov-converging-at-boundary-def* **by** *auto*
then have *Gromov-completion-rel u u* **using** *Gromov-converging-at-boundary-rel* **by** *auto*

define *v* **where** *v* = *rep-Gromov-completion x*
then have *Gromov-converging-at-boundary v*
using *Gromov-boundary-rep-converging*[*OF assms*] **by** *auto*
have *v*: *abs-Gromov-completion v* = *x* *Gromov-completion-rel v v*
using *Quotient3-abs-rep*[*OF Quotient3-Gromov-completion*] *Quotient3-rep-reflp*[*OF Quotient3-Gromov-completion*]
unfolding *v-def* **by** *auto*
then have *Lv*: ($\lambda n.$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (*v n*)) *x*) $\longrightarrow \infty$
using *Gromov-completion-converge-to-boundary-aux*[*OF assms*] **by** *auto*

have $*$: ($\lambda n.$ *min* (*extended-Gromov-product-at basepoint* (*to-Gromov-completion* (*u n*)) *x*) (*extended-Gromov-product-at basepoint x* (*to-Gromov-completion* (*v n*))) −
ereal (*deltaG TYPE*('*a*))) $\longrightarrow$ *min* ∞ ∞ − *ereal* (*deltaG TYPE*('*a*))
apply (*intro tendsto-intros*) **using** *Lu Lv* **by** (*auto simp add*: *extended-Gromov-product-at-commute*)
have ($\lambda n.$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (*u n*)) (*to-Gromov-completion* (*v n*))) $\longrightarrow \infty$
apply (*rule tendsto-sandwich*[*of* $\lambda n.$ *min* (*extended-Gromov-product-at basepoint* (*to-Gromov-completion* (*u n*)) *x*)
(*extended-Gromov-product-at basepoint x* (*to-Gromov-completion* (*v n*))) − *deltaG*(*TYPE*('*a*)) - - λ-. ∞])
using *extended-hyperb-ineq not-eventuallyD* **apply** *blast* **using** $*$ **by** *auto*
then have ($\lambda n.$ *Gromov-product-at basepoint* (*u n*) (*v n*)) $\longrightarrow \infty$
by *auto*
then have ($\lambda n.$ *Gromov-product-at a* (*u n*) (*v n*)) $\longrightarrow \infty$ **for** *a*
using *Gromov-product-tendsto-PInf-a-b* **by** *auto*

then have *Gromov-completion-rel u v*
unfolding *Gromov-completion-rel-def*
using ‹*Gromov-converging-at-boundary u*› ‹*Gromov-converging-at-boundary v*›
by *auto*
then have *abs-Gromov-completion u* = *abs-Gromov-completion v*
using *Quotient3-rel*[*OF Quotient3-Gromov-completion*] *v*(*2*) ‹*Gromov-completion-rel u u*› **by** *auto*
then have *abs-Gromov-completion u* = *x*
using *v*(*1*) **by** *auto*
then show *Gromov-completion-rel u u* ∧ *abs-Gromov-completion u* = *x*
using ‹*Gromov-completion-rel u u*› **by** *auto*
qed

In particular, it follows that a sequence which is `Gromov_converging_at_boundary` is indeed converging to a point on the boundary, the equivalence class of this sequence.

lemma *Gromov-converging-at-boundary-converges*:
assumes *Gromov-converging-at-boundary u*
shows $\exists\, x \in$ *Gromov-boundary*. ($\lambda n.$ *to-Gromov-completion* (*u n*)) $\longrightarrow x$
apply (*rule bexI*[*of - abs-Gromov-completion u*])
apply (*subst Gromov-completion-converge-to-boundary*)
using *assms* **by** (*auto simp add*: *Gromov-converging-at-boundary-rel*)

lemma *Gromov-converging-at-boundary-converges'*:
assumes *Gromov-converging-at-boundary u*
shows *convergent* ($\lambda n.$ *to-Gromov-completion* (*u n*))
unfolding *convergent-def* **using** *Gromov-converging-at-boundary-converges*[*OF assms*]
by *auto*

lemma *lim-imp-Gromov-converging-at-boundary*:
fixes *u*::*nat* ⇒ *'a*::*Gromov-hyperbolic-space*
assumes ($\lambda n.$ *to-Gromov-completion* (*u n*)) $\longrightarrow$ *x x* ∈ *Gromov-boundary*
shows *Gromov-converging-at-boundary u*
using *Gromov-boundary-abs-converging Gromov-completion-converge-to-boundary assms* **by** *blast*

If two sequences tend to the same point at infinity, then their Gromov product tends to infinity.

lemma *same-limit-imp-Gromov-product-tendsto-infinity*:
assumes *z* ∈ *Gromov-boundary*
($\lambda n.$ *to-Gromov-completion* (*u n*)) $\longrightarrow z$
($\lambda n.$ *to-Gromov-completion* (*v n*)) $\longrightarrow z$
shows $\exists N. \forall n \geq N. \forall m \geq N.$ *Gromov-product-at a* (*u n*) (*v m*) $\geq C$
proof −
have *Gromov-completion-rel u u Gromov-completion-rel v v abs-Gromov-completion u* = *abs-Gromov-completion v*
using *iffD1*[*OF Gromov-completion-converge-to-boundary*[*OF assms*(*1*)]] *assms*
by *auto*
then have ∗: *Gromov-completion-rel u v*

using *Quotient3-Gromov-completion Quotient3-rel* **by** *fastforce*
have $**$: *Gromov-converging-at-boundary u*
using *assms lim-imp-Gromov-converging-at-boundary* **by** *blast*
then obtain M **where** M: $\bigwedge m\ n.\ m \geq M \Longrightarrow n \geq M \Longrightarrow$ *Gromov-product-at a* $(u\ m)\ (u\ n) \geq C + deltaG(TYPE('a))$
unfolding *Gromov-converging-at-boundary-def* **by** *blast*

have $(\lambda n.$ *Gromov-product-at a* $(u\ n)\ (v\ n)) \longrightarrow \infty$
using $*$ *Gromov-converging-at-boundary-imp-not-constant'*[*OF* $**$] **unfolding** *Gromov-completion-rel-def* **by** *auto*
then have *eventually* $(\lambda n.$ *Gromov-product-at a* $(u\ n)\ (v\ n) \geq C + deltaG(TYPE('a)))$ *sequentially*
by (*meson Lim-PInfty ereal-less-eq*(*3*) *eventually-sequentiallyI*)
then obtain N **where** N: $\bigwedge n.\ n \geq N \Longrightarrow$ *Gromov-product-at a* $(u\ n)\ (v\ n) \geq C + deltaG(TYPE('a))$
unfolding *eventually-sequentially* **by** *auto*
have *Gromov-product-at a* $(u\ n)\ (v\ m) \geq C$ **if** $n \geq max\ M\ N\ m \geq max\ M\ N$ **for** $m\ n$
proof −
have $C + deltaG(TYPE('a)) \leq min$ (*Gromov-product-at a* $(u\ n)\ (u\ m)$) (*Gromov-product-at a* $(u\ m)\ (v\ m)$)
using *M N that* **by** *auto*
also have ... $\leq$ *Gromov-product-at a* $(u\ n)\ (v\ m) + deltaG(TYPE('a))$
by (*intro mono-intros*)
finally show *?thesis* **by** *simp*
qed
then show *?thesis*
by *blast*
qed

An admissible sequence converges in the Gromov boundary, to the point it defines. This follows from the definition of the topology in the two cases, inner and boundary.

lemma *abs-Gromov-completion-limit*:
assumes *Gromov-completion-rel u u*
shows $(\lambda n.$ *to-Gromov-completion* $(u\ n)) \longrightarrow$ *abs-Gromov-completion u*
proof (*cases abs-Gromov-completion u*)
case (*to-Gromov-completion x*)
then show *?thesis*
using *Gromov-completion-rel-to-const Quotient3-Gromov-completion Quotient3-rel assms to-Gromov-completion-def* **by** *fastforce*
next
case *boundary*
show *?thesis*
unfolding *Gromov-completion-converge-to-boundary*[*OF boundary*]
using *assms Gromov-boundary-rep-converging Gromov-converging-at-boundary-rel Quotient3-Gromov-completion Quotient3-abs-rep boundary* **by** *fastforce*
qed

In particular, a point in the Gromov boundary is the limit of its representative sequence in the space.

lemma *rep-Gromov-completion-limit*:
$(\lambda n.\ \textit{to-Gromov-completion}\ (\textit{rep-Gromov-completion}\ x\ n)) \longrightarrow x$
using *abs-Gromov-completion-limit*[*of rep-Gromov-completion x*] *Quotient3-Gromov-completion Quotient3-abs-rep Quotient3-rep-reflp* **by** *fastforce*

15.5 Continuity properties of the extended Gromov product and distance

We have defined our extended Gromov product in terms of sequences satisfying the equivalence relation. However, we would like to avoid this definition as much as possible, and express things in terms of the topology of the space. Hence, we reformulate this definition in topological terms, first when one of the two points is inside and the other one is on the boundary, then for all cases, and then we come back to the case where one point is inside, removing the assumption that the other one is on the boundary.

lemma *extended-Gromov-product-inside-boundary-aux*:
assumes $y \in$ *Gromov-boundary*
shows *extended-Gromov-product-at e* (*to-Gromov-completion x*) $y =$ *Inf* {*liminf* (λn. *ereal*(*Gromov-product-at e x* (v n))) |v. (λn. *to-Gromov-completion* (v n)) $\longrightarrow y$}
proof −
have *A*: *abs-Gromov-completion* $v =$ *to-Gromov-completion* $x \wedge$ *Gromov-completion-rel* v $v \longleftrightarrow (v = (\lambda n.\ x))$ **for** v
apply (*auto simp add*: *to-Gromov-completion-def*)
by (*metis* (*mono-tags*) *Gromov-completion-rel-def Quotient3-Gromov-completion abs-Gromov-completion-in-Gromov-boundary not-in-Gromov-boundary′ rep-Gromov-completion-to-Gromov-cor rep-abs-rsp to-Gromov-completion-def*)
have $*$: {F u v |u v. *abs-Gromov-completion* u = *to-Gromov-completion* x $\wedge$ *abs-Gromov-completion* $v = y \wedge$ *Gromov-completion-rel* u u $\wedge$ *Gromov-completion-rel* v v}
= {F (λn. x) v |v. (λn. *to-Gromov-completion* (v n)) $\longrightarrow y$} **for** F::(*nat* $\Rightarrow$ *'a*) $\Rightarrow$ (*nat* $\Rightarrow$ *'a*) $\Rightarrow$ *ereal*
unfolding *Gromov-completion-converge-to-boundary*[*OF* ‹$y \in$ *Gromov-boundary*›]
using *A* **by** *force*
show *?thesis*
unfolding *extended-Gromov-product-at-def* $*$ **by** *simp*
qed

lemma *extended-Gromov-product-boundary-inside-aux*:
assumes $y \in$ *Gromov-boundary*
shows *extended-Gromov-product-at e y* (*to-Gromov-completion x*) = *Inf* {*liminf* (λn. *ereal*(*Gromov-product-at e* (v n) x)) |v. (λn. *to-Gromov-completion* (v n)) $\longrightarrow y$}
using *extended-Gromov-product-inside-boundary-aux*[*OF assms*] **by** (*simp add*: *extended-Gromov-product-at-commute Gromov-product-commute*)

lemma *extended-Gromov-product-at-topological*:
extended-Gromov-product-at e x y = *Inf* {*liminf* (λ*n*. *ereal*(*Gromov-product-at e* (*u n*) (*v n*))) |*u v*. (λ*n*. *to-Gromov-completion* (*u n*)) ⟶ *x* ∧ (λ*n*. *to-Gromov-completion* (*v n*)) ⟶ *y*}
proof (*cases x*)
case *boundary*
show *?thesis*
proof (*cases y*)
case *boundary*
then show *?thesis*
unfolding *extended-Gromov-product-at-def Gromov-completion-converge-to-boundary*[*OF* ‹*x* ∈ *Gromov-boundary*›] *Gromov-completion-converge-to-boundary*[*OF* ‹*y* ∈ *Gromov-boundary*›]
by *meson*
next
case (*to-Gromov-completion yi*)
have *A*: *liminf* (λ*n*. *ereal* (*Gromov-product-at e* (*u n*) (*v n*))) = *liminf* (λ*n*. *ereal* (*Gromov-product-at e* (*u n*) *yi*)) **if** *v* ⟶ *yi* **for** *u v*
proof −
define *h* **where** *h* = (λ*n*. *Gromov-product-at e* (*u n*) (*v n*) − *Gromov-product-at e* (*u n*) *yi*)
have *h*: *h* ⟶ *0*
apply (*rule tendsto-rabs-zero-cancel*, *rule tendsto-sandwich*[*of* λ*n*. *0* - - λ*n*. *dist* (*v n*) *yi*])
unfolding *h-def* **using** *Gromov-product-at-diff3*[*of e* - - *yi*] *that* **apply** *auto*
using *tendsto-dist-iff* **by** *blast*
have ∗: *ereal* (*Gromov-product-at e* (*u n*) (*v n*)) = *h n* + *ereal* (*Gromov-product-at e* (*u n*) *yi*) **for** *n*
unfolding *h-def* **by** *auto*
have *liminf* (λ*n*. *ereal* (*Gromov-product-at e* (*u n*) (*v n*))) = *0* + *liminf* (λ*n*. *ereal* (*Gromov-product-at e* (*u n*) *yi*))
unfolding ∗ **apply** (*rule ereal-liminf-lim-add*) **using** *h* **by** (*auto simp add*: *zero-ereal-def*)
then show *?thesis* **by** *simp*
qed
show *?thesis*
unfolding *to-Gromov-completion extended-Gromov-product-boundary-inside-aux*[*OF* ‹*x* ∈ *Gromov-boundary*›] **apply** (*rule cong*[*of Inf Inf*], *auto*)
using *A* **by** *fast*+
qed
next
case (*to-Gromov-completion xi*)
show *?thesis*
proof (*cases y*)
case *boundary*
have *A*: *liminf* (λ*n*. *ereal* (*Gromov-product-at e* (*u n*) (*v n*))) = *liminf* (λ*n*. *ereal* (*Gromov-product-at e xi* (*v n*))) **if** *u* ⟶ *xi* **for** *u v*
proof −

define *h* **where** $h = (\lambda n.$ *Gromov-product-at e* $(u\ n)\ (v\ n) -$ *Gromov-product-at e xi* $(v\ n))$
have *h*: $h \longrightarrow 0$
apply (*rule tendsto-rabs-zero-cancel*, *rule tendsto-sandwich*[*of* $\lambda n.$ *0* - - $\lambda n.$ *dist* $(u\ n)$ *xi*])
unfolding *h-def* **using** *Gromov-product-at-diff2*[*of e* - - *xi*] *that* **apply** *auto*
using *tendsto-dist-iff* **by** *blast*
have $*$: *ereal* (*Gromov-product-at e* $(u\ n)\ (v\ n)) = h\ n +$ *ereal* (*Gromov-product-at e xi* $(v\ n))$ **for** *n*
unfolding *h-def* **by** *auto*
have *liminf* ($\lambda n.$ *ereal* (*Gromov-product-at e* $(u\ n)\ (v\ n))) = 0 +$ *liminf* ($\lambda n.$ *ereal* (*Gromov-product-at e xi* $(v\ n)))$
unfolding $*$ **apply** (*rule ereal-liminf-lim-add*) **using** *h* **by** (*auto simp add*: *zero-ereal-def*)
then show *?thesis* **by** *simp*
qed
show *?thesis*
unfolding *to-Gromov-completion extended-Gromov-product-inside-boundary-aux*[*OF* ‹$y \in$ *Gromov-boundary*›] **apply** (*rule cong*[*of Inf Inf*], *auto*)
using *A* **by** *fast+*
next
case (*to-Gromov-completion yi*)
have *B*: *liminf* ($\lambda n.$ *Gromov-product-at e* $(u\ n)\ (v\ n)) =$ *Gromov-product-at e xi yi* **if** $u \longrightarrow xi$ $v \longrightarrow yi$ **for** *u v*
proof $-$
have ($\lambda n.$ *Gromov-product-at e* $(u\ n)\ (v\ n)) \longrightarrow$ *Gromov-product-at e xi yi*
apply (*rule Gromov-product-at-continuous*) **using** *that* **by** *auto*
then show *liminf* ($\lambda n.$ *Gromov-product-at e* $(u\ n)\ (v\ n)) =$ *Gromov-product-at e xi yi*
by (*simp add*: *lim-imp-Liminf*)
qed
have $*$: {*liminf* ($\lambda n.$ *ereal* (*Gromov-product-at e* $(u\ n)\ (v\ n)))$ |*u v*. $u \longrightarrow xi \wedge v \longrightarrow yi$} = {*ereal* (*Gromov-product-at e xi yi*)}
using *B* **apply** *auto* **by** (*rule exI*[*of* - $\lambda n.$ *xi*], *rule exI*[*of* - $\lambda n.$ *yi*], *auto*)
show *?thesis*
unfolding ‹$x =$ *to-Gromov-completion xi*› ‹$y =$ *to-Gromov-completion yi*› **by** (*auto simp add*: $*$)
qed
qed

lemma *extended-Gromov-product-inside-boundary*:
extended-Gromov-product-at e (*to-Gromov-completion x*) $y =$ *Inf* {*liminf* ($\lambda n.$ *ereal*(*Gromov-product-at e x* $(v\ n)))$ |*v*. ($\lambda n.$ *to-Gromov-completion* $(v\ n)) \longrightarrow y$}
proof $-$
have *A*: *liminf* ($\lambda n.$ *ereal* (*Gromov-product-at e* $(u\ n)\ (v\ n))) =$ *liminf* ($\lambda n.$ *ereal* (*Gromov-product-at e x* $(v\ n)))$ **if** $u \longrightarrow x$ **for** *u v*
proof $-$

define *h* **where** $h = (\lambda n.\ \textit{Gromov-product-at e (u n) (v n)} - \textit{Gromov-product-at e x (v n)})$
have *h*: $h \longrightarrow 0$
apply (*rule tendsto-rabs-zero-cancel*, *rule tendsto-sandwich*[*of* $\lambda n.$ *0* - - $\lambda n.$ *dist* (*u n*) *x*])
unfolding *h-def* **using** *Gromov-product-at-diff2*[*of e* - - *x*] *that* **apply** *auto*
using *tendsto-dist-iff* **by** *blast*
have $*$: *ereal* (*Gromov-product-at e* (*u n*) (*v n*)) = *h n* + *ereal* (*Gromov-product-at e x* (*v n*)) **for** *n*
unfolding *h-def* **by** *auto*
have *liminf* ($\lambda n.$ *ereal* (*Gromov-product-at e* (*u n*) (*v n*))) = *0* + *liminf* ($\lambda n.$ *ereal* (*Gromov-product-at e x* (*v n*)))
unfolding $*$ **apply** (*rule ereal-liminf-lim-add*) **using** *h* **by** (*auto simp add*: *zero-ereal-def*)
then show *?thesis* **by** *simp*
qed
show *?thesis*
unfolding *extended-Gromov-product-at-topological* **apply** (*rule cong*[*of Inf Inf*], *auto*)
using *A* **by** *fast+*
qed

lemma *extended-Gromov-product-boundary-inside*:
extended-Gromov-product-at e y (*to-Gromov-completion x*) = *Inf* {*liminf* ($\lambda n.$ *ereal*(*Gromov-product-at e* (*v n*) *x*)) |*v*. ($\lambda n.$ *to-Gromov-completion* (*v n*)) $\longrightarrow$ *y*}
using *extended-Gromov-product-inside-boundary* **by** (*simp add*: *extended-Gromov-product-at-commute Gromov-product-commute*)

Now, we compare the extended Gromov product to a sequence of Gromov products for converging sequences. As the extended Gromov product is defined as an Inf of limings, it is clearly smaller than the liminf. More interestingly, it is also of the order of magnitude of the limsup, for whatever sequence one uses. In other words, it is canonically defined, up to 2δ.

lemma *extended-Gromov-product-le-liminf*:
assumes ($\lambda n.$ *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*
($\lambda n.$ *to-Gromov-completion* (*v n*)) $\longrightarrow$ *eta*
shows *liminf* ($\lambda n.$ *Gromov-product-at e* (*u n*) (*v n*)) $\geq$ *extended-Gromov-product-at e xi eta*
unfolding *extended-Gromov-product-at-topological* **using** *assms* **by** (*auto intro*!: *Inf-lower*)

lemma *limsup-le-extended-Gromov-product-inside*:
assumes ($\lambda n.$ *to-Gromov-completion* (*v n*)) $\longrightarrow$ (*eta*::(*'a*::*Gromov-hyperbolic-space*) *Gromov-completion*)
shows *limsup* ($\lambda n.$ *Gromov-product-at e x* (*v n*)) $\leq$ *extended-Gromov-product-at e* (*to-Gromov-completion x*) *eta* + *deltaG*(*TYPE*(*'a*))
proof (*cases eta*)
case *boundary*

have *A*: *limsup* ($\lambda n.$ *Gromov-product-at e x* (*v n*)) $\leq$ *liminf* ($\lambda n.$ *Gromov-product-at e x* (v' *n*)) + *deltaG*(*TYPE*('*a*))
if *H*: ($\lambda n.$ *to-Gromov-completion* (v' *n*)) $\longrightarrow$ *eta* **for** v'
proof −
have *ereal a* $\leq$ *liminf* ($\lambda n.$ *Gromov-product-at e x* (v' *n*)) + *deltaG*(*TYPE*('*a*)) **if** *L*: *ereal a* < *limsup* ($\lambda n.$ *Gromov-product-at e x* (*v n*)) **for** *a*
proof −
obtain *Nv* **where** *Nv*: $\bigwedge m\ n.\ m \geq Nv \Longrightarrow n \geq Nv \Longrightarrow$ *Gromov-product-at e* (*v m*) (v' *n*) $\geq a$
using *same-limit-imp-Gromov-product-tendsto-infinity*[*OF* ‹*eta* ∈ *Gromov-boundary*› *assms H*] **by** *blast*
obtain *N* **where** *N*: *ereal a* < *Gromov-product-at e x* (*v N*) $N \geq Nv$
using *limsup-obtain*[*OF L*] **by** *blast*
have ∗: *a* − *deltaG*(*TYPE*('*a*)) $\leq$ *Gromov-product-at e x* (v' *n*) **if** $n \geq Nv$ **for** *n*
proof −
have $a \leq$ *min* (*Gromov-product-at e x* (*v N*)) (*Gromov-product-at e* (*v N*) (v' *n*))
apply *auto* **using** *N*(*1*) *Nv*[*OF* ‹$N \geq Nv$› ‹$n \geq Nv$›] **by** *auto*
also have ... $\leq$ *Gromov-product-at e x* (v' *n*) + *deltaG*(*TYPE*('*a*))
by (*intro mono-intros*)
finally show *?thesis* **by** *auto*
qed
have *a* − *deltaG*(*TYPE*('*a*)) $\leq$ *liminf* ($\lambda n.$ *Gromov-product-at e x* (v' *n*))
apply (*rule Liminf-bounded*) **unfolding** *eventually-sequentially* **using** ∗ **by** *fastforce*
then show *?thesis*
unfolding *ereal-minus*(*1*)[*symmetric*] **by** (*subst ereal-minus-le*[*symmetric*], *auto*)
qed
then show *?thesis*
using *ereal-dense2 not-less* **by** *blast*
qed
have *limsup* ($\lambda n.$ *Gromov-product-at e x* (*v n*)) − *deltaG*(*TYPE*('*a*)) $\leq$ *extended-Gromov-product-at e* (*to-Gromov-completion x*) *eta*
unfolding *extended-Gromov-product-inside-boundary* **by** (*rule Inf-greatest*, *auto simp add*: *A*)
then show *?thesis* **by** *auto*
next
case (*to-Gromov-completion y*)
then have *v* $\longrightarrow$ *y* **using** *assms* **by** *auto*
have *L*: ($\lambda n.$ *Gromov-product-at e x* (*v n*)) $\longrightarrow$ *ereal*(*Gromov-product-at e x y*)
using *Gromov-product-at-continuous*[*OF* - - ‹*v* $\longrightarrow$ *y*›, *of* $\lambda n.$ *e e* $\lambda n.$ *x x*] **by** *auto*
show *?thesis*
unfolding *to-Gromov-completion* **using** *lim-imp-Limsup*[*OF* - *L*] **by** *auto*
qed

lemma *limsup-le-extended-Gromov-product-inside'*:
assumes ($\lambda n.$ *to-Gromov-completion* (v n)) $\longrightarrow$ (*eta*::('*a*::*Gromov-hyperbolic-space*) *Gromov-completion*)
shows *limsup* ($\lambda n.$ *Gromov-product-at* e (v n) x) $\leq$ *extended-Gromov-product-at* e *eta* (*to-Gromov-completion* x) + *deltaG*(*TYPE*('*a*))
using *limsup-le-extended-Gromov-product-inside*[*OF assms*] **by** (*simp add*: *Gromov-product-commute extended-Gromov-product-at-commute*)

lemma *limsup-le-extended-Gromov-product*:
assumes ($\lambda n.$ *to-Gromov-completion* (u n)) $\longrightarrow$ (*xi*::('*a*::*Gromov-hyperbolic-space*) *Gromov-completion*)
($\lambda n.$ *to-Gromov-completion* (v n)) $\longrightarrow$ *eta*
shows *limsup* ($\lambda n.$ *Gromov-product-at* e (u n) (v n)) $\leq$ *extended-Gromov-product-at* e *xi eta* + *2* $*$ *deltaG*(*TYPE*('*a*))
proof −
consider *xi* $\in$ *Gromov-boundary* $\wedge$ *eta* $\in$ *Gromov-boundary* | *xi* $\notin$ *Gromov-boundary* | *eta* $\notin$ *Gromov-boundary*
by *blast*
then show *?thesis*
proof (*cases*)
case *1*
then have *B*: *xi* $\in$ *Gromov-boundary eta* $\in$ *Gromov-boundary* **by** *auto*
have *A*: *limsup* ($\lambda n.$ *Gromov-product-at* e (u n) (v n)) $\leq$ *liminf* ($\lambda n.$ *Gromov-product-at* e (u' n) (v' n)) + *2* $*$ *deltaG*(*TYPE*('*a*))
if *H*: ($\lambda n.$ *to-Gromov-completion* (u' n)) $\longrightarrow$ *xi* ($\lambda n.$ *to-Gromov-completion* (v' n)) $\longrightarrow$ *eta* **for** u' v'
proof −
have *ereal* a $\leq$ *liminf* ($\lambda n.$ *Gromov-product-at* e (u' n) (v' n)) + *2* $*$ *deltaG*(*TYPE*('*a*)) **if** *L*: *ereal* a < *limsup* ($\lambda n.$ *Gromov-product-at* e (u n) (v n)) **for** a
proof −
obtain *Nu* **where** *Nu*: $\bigwedge m\ n.\ m \geq Nu \Longrightarrow n \geq Nu \Longrightarrow$ *Gromov-product-at* e (u' m) (u n) $\geq a$
using *same-limit-imp-Gromov-product-tendsto-infinity*[*OF* ‹*xi* $\in$ *Gromov-boundary*› *H*(*1*) *assms*(*1*)] **by** *blast*
obtain *Nv* **where** *Nv*: $\bigwedge m\ n.\ m \geq Nv \Longrightarrow n \geq Nv \Longrightarrow$ *Gromov-product-at* e (v m) (v' n) $\geq a$
using *same-limit-imp-Gromov-product-tendsto-infinity*[*OF* ‹*eta* $\in$ *Gromov-boundary*› *assms*(*2*) *H*(*2*)] **by** *blast*
obtain N **where** N: *ereal* a < *Gromov-product-at* e (u N) (v N) $N \geq$ *max Nu Nv*
using *limsup-obtain*[*OF L*] **by** *blast*
then have $N \geq Nu$ $N \geq Nv$ **by** *auto*
have $*$: a − *2* $*$ *deltaG*(*TYPE*('*a*)) $\leq$ *Gromov-product-at* e (u' n) (v' n) **if** $n \geq$ *max Nu Nv* **for** n
proof −
have n: $n \geq Nu$ $n \geq Nv$ **using** *that* **by** *auto*
have $a \leq$ *Min* {*Gromov-product-at* e (u' n) (u N), *Gromov-product-at* e (u N) (v N), *Gromov-product-at* e (v N) (v' n)}

apply *auto* **using** *N(1) Nu*[*OF n(1)* ‹*N* ≥ *Nu*›] *Nv*[*OF* ‹*N* ≥ *Nv*› *n(2)*]
by *auto*
also have ... ≤ *Gromov-product-at e* (*u′ n*) (*v′ n*) + *2* ∗ *deltaG*(*TYPE*(*′a*))
by (*intro mono-intros*)
finally show *?thesis* **by** *auto*
qed
have *a* − *2* ∗ *deltaG*(*TYPE*(*′a*)) ≤ *liminf* (*λn. Gromov-product-at e* (*u′ n*) (*v′ n*))
apply (*rule Liminf-bounded*) **unfolding** *eventually-sequentially* **using** ∗
by *fastforce*
then show *?thesis*
unfolding *ereal-minus*(*1*)[*symmetric*] **by** (*subst ereal-minus-le*[*symmetric*], *auto*)
qed
then show *?thesis*
using *ereal-dense2 not-less* **by** *blast*
qed
have *limsup* (*λn. Gromov-product-at e* (*u n*) (*v n*)) − *2* ∗ *deltaG*(*TYPE*(*′a*)) ≤ *extended-Gromov-product-at e xi eta*
unfolding *extended-Gromov-product-at-topological* **by** (*rule Inf-greatest*, *auto simp add*: *A*)
then show *?thesis* **by** *auto*
next
case *2*
then obtain *x* **where** *x*: *xi* = *to-Gromov-completion x* **by** (*cases xi*, *auto*)
have *A*: *limsup* (*λn. ereal* (*Gromov-product-at e* (*u n*) (*v n*))) = *limsup* (*λn. ereal* (*Gromov-product-at e x* (*v n*)))
proof −
define *h* **where** *h* = (*λn. Gromov-product-at e* (*u n*) (*v n*) − *Gromov-product-at e x* (*v n*))
have *h*: *h* ⟶ *0*
apply (*rule tendsto-rabs-zero-cancel*, *rule tendsto-sandwich*[*of λn. 0 - - λn. dist* (*u n*) *x*])
unfolding *h-def* **using** *Gromov-product-at-diff2*[*of e - - x*] *assms*(*1*) **unfolding** *x* **apply** *auto*
using *tendsto-dist-iff* **by** *blast*
have ∗: *ereal* (*Gromov-product-at e* (*u n*) (*v n*)) = *h n* + *ereal* (*Gromov-product-at e x* (*v n*)) **for** *n*
unfolding *h-def* **by** *auto*
have *limsup* (*λn. ereal* (*Gromov-product-at e* (*u n*) (*v n*))) = *0* + *limsup* (*λn. ereal* (*Gromov-product-at e x* (*v n*)))
unfolding ∗ **apply** (*rule ereal-limsup-lim-add*) **using** *h* **by** (*auto simp add*: *zero-ereal-def*)
then show *?thesis* **by** *simp*
qed
have ∗: *ereal* (*deltaG TYPE*(*′a*)) ≤ *ereal* (*2* ∗ *deltaG TYPE*(*′a*))
by *auto*
show *?thesis*
unfolding *A x* **using** *limsup-le-extended-Gromov-product-inside*[*OF assms*(*2*),

of e x] ∗
by (*meson add-left-mono order.trans*)
next
case *3*
then obtain *y* **where** *y*: *eta* = *to-Gromov-completion y* **by** (*cases eta*, *auto*)
have *A*: *limsup* (*λn. ereal* (*Gromov-product-at e* (*u n*) (*v n*))) = *limsup* (*λn. ereal* (*Gromov-product-at e* (*u n*) *y*))
proof −
define *h* **where** *h* = (*λn. Gromov-product-at e* (*u n*) (*v n*) − *Gromov-product-at e* (*u n*) *y*)
have *h*: *h* ⟶ *0*
apply (*rule tendsto-rabs-zero-cancel*, *rule tendsto-sandwich*[*of λn. 0 - - λn. dist* (*v n*) *y*])
unfolding *h-def* **using** *Gromov-product-at-diff3*[*of e - - y*] *assms*(*2*) **unfolding** *y* **apply** *auto*
using *tendsto-dist-iff* **by** *blast*
have ∗: *ereal* (*Gromov-product-at e* (*u n*) (*v n*)) = *h n* + *ereal* (*Gromov-product-at e* (*u n*) *y*) **for** *n*
unfolding *h-def* **by** *auto*
have *limsup* (*λn. ereal* (*Gromov-product-at e* (*u n*) (*v n*))) = *0* + *limsup* (*λn. ereal* (*Gromov-product-at e* (*u n*) *y*))
unfolding ∗ **apply** (*rule ereal-limsup-lim-add*) **using** *h* **by** (*auto simp add*: *zero-ereal-def*)
then show *?thesis* **by** *simp*
qed
have ∗: *ereal* (*deltaG TYPE*(*'a*)) ≤ *ereal* (*2* ∗ *deltaG TYPE*(*'a*))
by *auto*
show *?thesis*
unfolding *A y* **using** *limsup-le-extended-Gromov-product-inside'*[*OF assms*(*1*), *of e y*] ∗
by (*meson add-left-mono order.trans*)
qed
qed

One can then extend to the boundary the fact that $(y,z)_x + (x,z)_y = d(x,y)$, up to a constant δ, by taking this identity inside and passing to the limit.

lemma *extended-Gromov-product-add-le*:
extended-Gromov-product-at x xi (*to-Gromov-completion y*) + *extended-Gromov-product-at y xi* (*to-Gromov-completion x*) ≤ *dist x y*
proof −
obtain *u* **where** *u*: (*λn. to-Gromov-completion* (*u n*)) ⟶ *xi*
using *rep-Gromov-completion-limit* **by** *blast*
have *liminf* (*λn. ereal* (*Gromov-product-at a b* (*u n*))) ≥ *0* **for** *a b*
by (*rule Liminf-bounded*[*OF always-eventually*], *auto*)
then have ∗: *liminf* (*λn. ereal* (*Gromov-product-at a b* (*u n*))) ≠ −∞ **for** *a b*
by *auto*
have *extended-Gromov-product-at x xi* (*to-Gromov-completion y*) + *extended-Gromov-product-at y xi* (*to-Gromov-completion x*)
≤ *liminf* (*λn. ereal* (*Gromov-product-at x y* (*u n*))) + *liminf* (*λn. Gro-*

mov-product-at y x (*u n*))
  apply (*intro mono-intros*)
  using *extended-Gromov-product-le-liminf* [*OF u*, *of* λn. *y to-Gromov-completion y x*]
    extended-Gromov-product-le-liminf [*OF u*, *of* λn. *x to-Gromov-completion x y*] **by** (*auto simp add*: *Gromov-product-commute*)
 also have ... $\leq$ *liminf* (λn. *ereal* (*Gromov-product-at x y* (*u n*)) + *Gromov-product-at y x* (*u n*))
  by (*rule ereal-liminf-add-mono*, *auto simp add*: $*$)
 also have ... = *dist x y*
  apply (*simp add*: *Gromov-product-add*)
  by (*metis lim-imp-Liminf sequentially-bot tendsto-const*)
 finally show *?thesis* **by** *auto*
qed

lemma *extended-Gromov-product-add-ge*:
 extended-Gromov-product-at (*x*::*'a*::*Gromov-hyperbolic-space*) *xi* (*to-Gromov-completion y*) + *extended-Gromov-product-at y xi* (*to-Gromov-completion x*) $\geq$ *dist x y* $-$ *deltaG*(*TYPE*(*'a*))
proof $-$
 have *A*: *dist x y* $-$ *extended-Gromov-product-at y* (*to-Gromov-completion x*) *xi* $-$ *deltaG*(*TYPE*(*'a*)) $\leq$ *liminf* (λn. *ereal* (*Gromov-product-at x y* (*u n*)))
  if (λn. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi* **for** *u*
 proof $-$
   have *dist x y* = *liminf* (λn. *ereal* (*Gromov-product-at x y* (*u n*)) + *Gromov-product-at y x* (*u n*))
    apply (*simp add*: *Gromov-product-add*)
    by (*metis lim-imp-Liminf sequentially-bot tendsto-const*)
  also have ... $\leq$ *liminf* (λn. *ereal* (*Gromov-product-at x y* (*u n*))) + *limsup* (λn. *Gromov-product-at y x* (*u n*))
    by (*rule ereal-liminf-limsup-add*)
  also have ... $\leq$ *liminf* (λn. *ereal* (*Gromov-product-at x y* (*u n*))) + (*extended-Gromov-product-at y* (*to-Gromov-completion x*) *xi* + *deltaG*(*TYPE*(*'a*)))
    by (*intro mono-intros limsup-le-extended-Gromov-product-inside*[*OF that*])
  finally show *?thesis* **by** (*auto simp add*: *algebra-simps*)
 qed
 have *dist x y* $-$ *extended-Gromov-product-at y* (*to-Gromov-completion x*) *xi* $-$ *deltaG*(*TYPE*(*'a*)) $\leq$ *extended-Gromov-product-at x* (*to-Gromov-completion y*) *xi*
  unfolding *extended-Gromov-product-inside-boundary*[*of x*] **apply** (*rule Inf-greatest*) **using** *A* **by** *auto*
 then show *?thesis*
  apply (*auto simp add*: *algebra-simps extended-Gromov-product-at-commute*)
  unfolding *ereal-minus*(*1*)[*symmetric*] **by** (*subst ereal-minus-le*, *auto simp add*: *algebra-simps*)
qed

If one perturbs a sequence inside the space by a bounded distance, one does not change the limit on the boundary.

lemma *Gromov-converging-at-boundary-bounded-perturbation*:

```
  assumes (λn. to-Gromov-completion (u n)) ⟶ x
          x ∈ Gromov-boundary
          ⋀n. dist (u n) (v n) ≤ C
  shows (λn. to-Gromov-completion (v n)) ⟶ x
proof –
  have (λn. extended-Gromov-product-at basepoint (to-Gromov-completion (v n))
x) ⟶ ∞
  proof (rule tendsto-sandwich[of λn. extended-Gromov-product-at basepoint (to-Gromov-completion
(u n)) x − C - - λn. ∞])
    show ∀F n in sequentially. extended-Gromov-product-at basepoint (to-Gromov-completion
(u n)) x − ereal C ≤ extended-Gromov-product-at basepoint (to-Gromov-completion
(v n)) x
    proof (rule always-eventually, auto)
      fix n::nat
      have extended-Gromov-product-at basepoint (to-Gromov-completion (u n)) x
≤ extended-Gromov-product-at basepoint (to-Gromov-completion (v n)) x
                       + extended-Gromov-distance (to-Gromov-completion (u n))
(to-Gromov-completion (v n))
        by (intro mono-intros)
      also have ... ≤ extended-Gromov-product-at basepoint (to-Gromov-completion
(v n)) x + C
        using assms(3)[of n] by (intro mono-intros, auto)
      finally show extended-Gromov-product-at basepoint (to-Gromov-completion
(u n)) x ≤ extended-Gromov-product-at basepoint (to-Gromov-completion (v n)) x
+ ereal C
        by auto
    qed
    have (λn. extended-Gromov-product-at basepoint (to-Gromov-completion (u n))
x − ereal C) ⟶ ∞ − ereal C
      apply (intro tendsto-intros)
       unfolding Gromov-completion-boundary-limit[OF ‹x ∈ Gromov-boundary›,
symmetric] using assms(1) by auto
    then show (λn. extended-Gromov-product-at basepoint (to-Gromov-completion
(u n)) x − ereal C) ⟶ ∞
      by auto
  qed (auto)
  then show ?thesis
    unfolding Gromov-completion-boundary-limit[OF ‹x ∈ Gromov-boundary›] by
simp
qed
```

We prove that the extended Gromov distance is a continuous function of one variable, by separating the different cases at infinity and inside the space. Note that it is not a continuous function of both variables: if u_n is inside the space but tends to a point x in the boundary, then the extended Gromov distance between u_n and u_n is 0, but for the limit it is ∞.

```
lemma extended-Gromov-distance-continuous:
  continuous-on UNIV (λy. extended-Gromov-distance x y)
proof (cases x)
```

First, if x is in the boundary, then all distances to x are infinite, and the statement is trivial.

case *boundary*
then have $*$: *extended-Gromov-distance* x y = ∞ **for** y
by *auto*
show *?thesis*
unfolding $*$ **using** *continuous-on-topological* **by** *blast*
next

Next, consider the case where x is inside the space. We split according to whether y is inside the space or at infinity.

case (*to-Gromov-completion* a)
have ($\lambda n.$ *extended-Gromov-distance* x (u n)) $\longrightarrow$ *extended-Gromov-distance* x y **if** u $\longrightarrow$ y **for** u y
proof (*cases* y)

If y is at infinity, then we know that the Gromov product of u_n and y tends to infinity. Therefore, the extended distance from u_n to any fixed point also tends to infinity (as the Gromov product is bounded from below by the extended distance).

case *boundary*
have $*$: ($\lambda n.$ *extended-Gromov-product-at* a (u n) y) $\longrightarrow$ ∞
by (*rule extended-Gromov-product-tendsto-PInf-a-b*[*OF iffD1* [*OF Gromov-completion-boundary-limit*, *OF boundary* ‹u $\longrightarrow$ y›]])
have ($\lambda n.$ *extended-Gromov-distance* x (u n)) $\longrightarrow$ ∞
apply (*rule tendsto-sandwich*[*of* $\lambda n.$ *extended-Gromov-product-at* a (u n) y - - λ-. ∞])
unfolding *to-Gromov-completion* **using** *extended-Gromov-product-le-dist*[*of* a u - y] $*$ **by** *auto*
then show *?thesis* **using** *boundary* **by** *auto*
next

If y is inside the space, then we use the triangular inequality for the extended Gromov distance to conclure.

case (*to-Gromov-completion* b)
then have F: $y \notin$ *Gromov-boundary* **by** *auto*
have $*$: ($\lambda n.$ *extended-Gromov-distance* (u n) y) $\longrightarrow$ *0*
by (*rule iffD1* [*OF Gromov-completion-inside-limit*[*OF F*] ‹u $\longrightarrow$ y›])
show ($\lambda n.$ *extended-Gromov-distance* x (u n)) $\longrightarrow$ *extended-Gromov-distance* x y
proof (*rule tendsto-sandwich*[*of* $\lambda n.$ *extended-Gromov-distance* x y $-$ *extended-Gromov-distance* (u n) y - - $\lambda n.$ *extended-Gromov-distance* x y + *extended-Gromov-distance* (u n) y])
have *extended-Gromov-distance* x y $-$ *extended-Gromov-distance* (u n) y $\leq$ *extended-Gromov-distance* x (u n) **for** n
using *extended-Gromov-distance-triangle*[*of* y x u n]

by (*auto simp add*: *extended-Gromov-distance-commute F ennreal-minus-le-iff extended-Gromov-distance-def*)
then show $\forall_F$ *n in sequentially. extended-Gromov-distance x y* $-$ *extended-Gromov-distance* (*u n*) *y* $\leq$ *extended-Gromov-distance x* (*u n*)
by *auto*
have *extended-Gromov-distance x* (*u n*) $\leq$ *extended-Gromov-distance x y* + *extended-Gromov-distance* (*u n*) *y* **for** *n*
using *extended-Gromov-distance-triangle*[*of x u n y*] **by** (*auto simp add*: *extended-Gromov-distance-commute*)
then show $\forall_F$ *n in sequentially. extended-Gromov-distance x* (*u n*) $\leq$ *extended-Gromov-distance x y* + *extended-Gromov-distance* (*u n*) *y*
by *auto*
have (λ*n. extended-Gromov-distance x y* $-$ *extended-Gromov-distance* (*u n*) *y*) $\longrightarrow$ *extended-Gromov-distance x y* $-$ *0*
by (*intro tendsto-intros* $*$, *auto*)
then show (λ*n. extended-Gromov-distance x y* $-$ *extended-Gromov-distance* (*u n*) *y*) $\longrightarrow$ *extended-Gromov-distance x y*
by *simp*
have (λ*n. extended-Gromov-distance x y* + *extended-Gromov-distance* (*u n*) *y*) $\longrightarrow$ *extended-Gromov-distance x y* + *0*
by (*intro tendsto-intros* $*$, *auto*)
then show (λ*n. extended-Gromov-distance x y* + *extended-Gromov-distance* (*u n*) *y*) $\longrightarrow$ *extended-Gromov-distance x y*
by *simp*
qed
qed
then show *?thesis*
unfolding *continuous-on-sequentially comp-def* **by** *auto*
qed

lemma *extended-Gromov-distance-continuous'*:
continuous-on UNIV (λ*x. extended-Gromov-distance x y*)
using *extended-Gromov-distance-continuous*[*of y*] *extended-Gromov-distance-commute*[*of - y*] **by** *auto*

15.6 Topology of the Gromov boundary

We deduce the basic fact that the original space is open in the Gromov completion from the continuity of the extended distance.

lemma *to-Gromov-completion-range-open*:
open (*range to-Gromov-completion*)
proof $-$
have $*$: *range to-Gromov-completion* = (λ*x. extended-Gromov-distance* (*to-Gromov-completion basepoint*) *x*)$-$‘{..<∞}
using *Gromov-boundary-def extended-Gromov-distance-PInf-boundary*(*2*) **by** *fastforce*
show *?thesis*
unfolding $*$ **using** *extended-Gromov-distance-continuous open-lessThan open-vimage* **by** *blast*

qed

lemma *Gromov-boundary-closed*:
closed Gromov-boundary
unfolding *Gromov-boundary-def* **using** *to-Gromov-completion-range-open* **by** *auto*

The original space is also dense in its Gromov completion, as all points at infinity are by definition limits of some sequence in the space.

lemma *to-Gromov-completion-range-dense* [*simp*]:
closure (*range to-Gromov-completion*) = *UNIV*
apply (*auto simp add*: *closure-sequential*) **using** *rep-Gromov-completion-limit* **by** *force*

lemma *to-Gromov-completion-homeomorphism*:
homeomorphism-on UNIV to-Gromov-completion
by (*rule homeomorphism-on-sequentially*, *auto*)

lemma *to-Gromov-completion-continuous*:
continuous-on UNIV to-Gromov-completion
by (*rule homeomorphism-on-continuous*[*OF to-Gromov-completion-homeomorphism*])

lemma *from-Gromov-completion-continuous*:
homeomorphism-on (*range to-Gromov-completion*) *from-Gromov-completion*
continuous-on (*range to-Gromov-completion*) *from-Gromov-completion*
$\bigwedge x$::($'a$::*Gromov-hyperbolic-space*) *Gromov-completion*. $x \in$ *range to-Gromov-completion*
$\Longrightarrow$ *continuous* (*at* x) *from-Gromov-completion*
proof −
show ∗: *homeomorphism-on* (*range to-Gromov-completion*) *from-Gromov-completion*
using *homeomorphism-on-inverse*[*OF to-Gromov-completion-homeomorphism*]
unfolding *from-Gromov-completion-def*[*symmetric*] **by** *simp*
show *continuous-on* (*range to-Gromov-completion*) *from-Gromov-completion*
by (*simp add*: ∗ *homeomorphism-on-continuous*)
then show *continuous* (*at* x) *from-Gromov-completion* **if** $x \in$ *range to-Gromov-completion*
for x::$'a$ *Gromov-completion*
using *continuous-on-eq-continuous-at that to-Gromov-completion-range-open*
by *auto*
qed

The Gromov boundary is always complete. Indeed, consider a Cauchy sequence u_n in the boundary, and approximate well enough u_n by a point v_n inside. Then the sequence v_n is Gromov converging at infinity (the respective Gromov products tend to infinity essentially by definition), and its limit point is the limit of the original sequence u.

proposition *Gromov-boundary-complete*:
complete Gromov-boundary
proof (*rule completeI*)
fix u::*nat* ⇒ $'a$ *Gromov-completion* **assume** ∀ n. u n ∈ *Gromov-boundary Cauchy* u

then have u: $\bigwedge n.\ u\ n \in$ *Gromov-boundary* **by** *auto*
have $*$: $\exists x \in$ *range to-Gromov-completion. dist* $(u\ n)\ x < 1/real(n+1)$ **for** n
by (*rule closure-approachableD, auto simp add*: *to-Gromov-completion-range-dense*)
have $\exists v.\ \forall n.$ *dist* (*to-Gromov-completion* $(v\ n)$) $(u\ n) < 1/real(n+1)$
using *of-nat-less-top* **apply** (*intro choice*) **using** $*$ **by** (*auto simp add*: *dist-commute*)
then obtain v **where** v: $\bigwedge n.$ *dist* (*to-Gromov-completion* $(v\ n)$) $(u\ n) < 1/real(n+1)$
by *blast*
have ($\lambda n.$ *dist* (*to-Gromov-completion* $(v\ n)$) $(u\ n)$) $\longrightarrow 0$
apply (*rule tendsto-sandwich*[*of* λ-. 0 - - $\lambda n.\ 1/real(n+1)$])
using v *LIMSEQ-ignore-initial-segment*[*OF lim-1-over-n*, *of 1*] **unfolding** *eventually-sequentially*
by (*auto simp add*: *less-imp-le*)

have *Gromov-converging-at-boundary* v
proof (*rule Gromov-converging-at-boundaryI*[*of basepoint*])
fix M::*real*
obtain *D1 e1* **where** *D1*: $e1 > 0$ $D1 < \infty$ $\bigwedge x\ y$::'*a Gromov-completion. dist* $x\ y \le e1 \Longrightarrow$ *extended-Gromov-distance* x (*to-Gromov-completion basepoint*) $\ge D1 \Longrightarrow$ *extended-Gromov-product-at basepoint* $x\ y \ge$ *ereal* M
using *large-Gromov-product-approx*[*of ereal M*] **by** *auto*
obtain *D2 e2* **where** *D2*: $e2 > 0$ $D2 < \infty$ $\bigwedge x\ y$::'*a Gromov-completion. dist* $x\ y \le e2 \Longrightarrow$ *extended-Gromov-distance* x (*to-Gromov-completion basepoint*) $\ge D2 \Longrightarrow$ *extended-Gromov-product-at basepoint* $x\ y \ge D1$
using *large-Gromov-product-approx*[*OF* ‹$D1 < \infty$›] **by** *auto*
define e **where** $e = (min\ e1\ e2)/3$
have $e > 0$ **unfolding** *e-def* **using** ‹$e1 > 0$› ‹$e2 > 0$› **by** *auto*
then obtain *N1* **where** *N1*: $\bigwedge n\ m.\ n \ge N1 \Longrightarrow m \ge N1 \Longrightarrow$ *dist* $(u\ n)\ (u\ m) < e$
using ‹*Cauchy u*› **unfolding** *Cauchy-def* **by** *blast*
have *eventually* ($\lambda n.$ *dist* (*to-Gromov-completion* $(v\ n)$) $(u\ n) < e$) *sequentially*
by (*rule order-tendstoD*[*OF* ‹($\lambda n.$ *dist* (*to-Gromov-completion* $(v\ n)$) $(u\ n)$) $\longrightarrow 0$›], *fact*)
then obtain *N2* **where** *N2*: $\bigwedge n.\ n \ge N2 \Longrightarrow$ *dist* (*to-Gromov-completion* $(v\ n)$) $(u\ n) < e$
unfolding *eventually-sequentially* **by** *auto*
have *ereal* $M \le$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* $(v\ m)$) (*to-Gromov-completion* $(v\ n)$)
if $n \ge max\ N1\ N2$ $m \ge max\ N1\ N2$ **for** $m\ n$
proof (*rule D1*(*3*))
have *dist* (*to-Gromov-completion* $(v\ m)$) (*to-Gromov-completion* $(v\ n)$)
$\le$ *dist* (*to-Gromov-completion* $(v\ m)$) $(u\ m)$ + *dist* $(u\ m)\ (u\ n)$ + *dist* $(u\ n)$ (*to-Gromov-completion* $(v\ n)$)
by (*intro mono-intros*)
also have ... $\le e + e + e$
apply (*intro mono-intros*)
using *N1*[*of m n*] *N2*[*of n*] *N2*[*of m*] *that* **by** (*auto simp add*: *dist-commute*)
also have ... $\le e1$ **unfolding** *e-def* **by** *auto*
finally show *dist* (*to-Gromov-completion* $(v\ m)$) (*to-Gromov-completion* $(v\ n)$) $\le e1$ **by** *simp*

have *e* ≤ *e2* **unfolding** *e-def* **using** ‹*e2* > *0*› **by** *auto*
have *D1* ≤ *extended-Gromov-product-at basepoint* (*u m*) (*to-Gromov-completion* (*v m*))
apply (*rule D2*(*3*)) **using** *N2*[*of m*] *that* ‹*e* ≤ *e2*› *u*[*of m*] **by** (*auto simp add*: *dist-commute*)
also have ... ≤ *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*to-Gromov-completion* (*v m*))
using *extended-Gromov-product-le-dist*[*of basepoint to-Gromov-completion* (*v m*) *u m*]
by (*simp add*: *extended-Gromov-product-at-commute*)
finally show *D1* ≤ *extended-Gromov-distance* (*to-Gromov-completion* (*v m*)) (*to-Gromov-completion basepoint*)
by (*simp add*: *extended-Gromov-distance-commute*)
qed
then have *M* ≤ *Gromov-product-at basepoint* (*v m*) (*v n*) **if** *n* ≥ *max N1 N2* *m* ≥ *max N1 N2* **for** *m n*
using *that* **by** *auto*
then show ∃ *N*. ∀ *n* ≥ *N*. ∀ *m* ≥ *N*. *M* ≤ *Gromov-product-at basepoint* (*v m*) (*v n*)
by *blast*
qed
then obtain *l* **where** *l*: *l* ∈ *Gromov-boundary* (λ*n*. *to-Gromov-completion* (*v n*)) ⟶ *l*
using *Gromov-converging-at-boundary-converges* **by** *blast*
have (λ*n*. *dist* (*u n*) *l*) ⟶ *0*+*0*
proof (*rule tendsto-sandwich*[*of* λ-. *0* - - λ*n*. *dist* (*u n*) (*to-Gromov-completion* (*v n*)) + *dist* (*to-Gromov-completion* (*v n*)) *l*])
show (λ*n*. *dist* (*u n*) (*to-Gromov-completion* (*v n*)) + *dist* (*to-Gromov-completion* (*v n*)) *l*) ⟶ *0* + *0*
apply (*intro tendsto-intros*)
using *iffD1*[*OF tendsto-dist-iff l*(*2*)] ‹(λ*n*. *dist* (*to-Gromov-completion* (*v n*)) (*u n*)) ⟶ *0*›
by (*auto simp add*: *dist-commute*)
qed (*auto simp add*: *dist-triangle*)
then have *u* ⟶ *l*
using *iffD2*[*OF tendsto-dist-iff*] **by** *auto*
then show ∃ *l*∈*Gromov-boundary*. *u* ⟶ *l*
using *l*(*1*) **by** *auto*
qed

When the initial space is complete, then the whole Gromov completion is also complete: for Cauchy sequences tending to the Gromov boundary, then the convergence is proved as in the completeness of the boundary above. For Cauchy sequences that remain bounded, the convergence is reduced to the convergence inside the original space, which holds by assumption.

proposition *Gromov-completion-complete*:
assumes *complete* (*UNIV*::′*a*::*Gromov-hyperbolic-space set*)
shows *complete* (*UNIV*::′*a Gromov-completion set*)

proof (*rule completeI*, *auto*)
fix *u0*::*nat* ⇒ *'a Gromov-completion* **assume** *Cauchy u0*
show ∃ *l*. *u0* $\longrightarrow$ *l*
proof (*cases limsup* (λ*n*. *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u0 n*)) = ∞)
case *True*
then obtain *r* **where** *r*: *strict-mono r* (λ*n*. *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u0* (*r n*))) $\longrightarrow$ ∞
using *limsup-subseq-lim*[*of* (λ*n*. *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u0 n*))] **unfolding** *comp-def*
by *auto*
define *u* **where** *u* = *u0 o r*
then have (λ*n*. *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u n*)) $\longrightarrow$ ∞
unfolding *comp-def* **using** *r*(*2*) **by** *simp*
have *Cauchy u*
using ‹*Cauchy u0*› *r*(*1*) *u-def* **by** (*simp add*: *Cauchy-subseq-Cauchy*)

have ∗: ∃ *x* ∈ *range to-Gromov-completion*. *dist* (*u n*) *x* < *1*/*real*(*n*+*1*) **for** *n*
by (*rule closure-approachableD*, *auto*)
have ∃ *v*. ∀ *n*. *dist* (*to-Gromov-completion* (*v n*)) (*u n*) < *1*/*real*(*n*+*1*)
using *of-nat-less-top* **apply** (*intro choice*) **using** ∗ **by** (*auto simp add*: *dist-commute*)
then obtain *v* **where** *v*: $\bigwedge$*n*. *dist* (*to-Gromov-completion* (*v n*)) (*u n*) < *1*/*real*(*n*+*1*)
by *blast*
have (λ*n*. *dist* (*to-Gromov-completion* (*v n*)) (*u n*)) $\longrightarrow$ *0*
apply (*rule tendsto-sandwich*[*of* λ-. *0* - - λ*n*. *1*/*real*(*n*+*1*)])
using *v LIMSEQ-ignore-initial-segment*[*OF lim-1-over-n*, *of 1*] **unfolding** *eventually-sequentially*
by (*auto simp add*: *less-imp-le*)

have *Gromov-converging-at-boundary v*
proof (*rule Gromov-converging-at-boundaryI*[*of basepoint*])
fix *M*::*real*
obtain *D1 e1* **where** *D1*: *e1* > *0 D1* < ∞ $\bigwedge$*x y*::*'a Gromov-completion*. *dist x y* ≤ *e1* ⟹ *extended-Gromov-distance x* (*to-Gromov-completion basepoint*) ≥ *D1* ⟹ *extended-Gromov-product-at basepoint x y* ≥ *ereal M*
using *large-Gromov-product-approx*[*of ereal M*] **by** *auto*
obtain *D2 e2* **where** *D2*: *e2* > *0 D2* < ∞ $\bigwedge$*x y*::*'a Gromov-completion*. *dist x y* ≤ *e2* ⟹ *extended-Gromov-distance x* (*to-Gromov-completion basepoint*) ≥ *D2* ⟹ *extended-Gromov-product-at basepoint x y* ≥ *D1*
using *large-Gromov-product-approx*[*OF* ‹*D1* < ∞›] **by** *auto*
define *e* **where** *e* = (*min e1 e2*)/*3*
have *e* > *0* **unfolding** *e-def* **using** ‹*e1* > *0*› ‹*e2* > *0*› **by** *auto*
then obtain *N1* **where** *N1*: $\bigwedge$*n m*. *n* ≥ *N1* ⟹ *m* ≥ *N1* ⟹ *dist* (*u n*) (*u m*) < *e*
using ‹*Cauchy u*› **unfolding** *Cauchy-def* **by** *blast*
have *eventually* (λ*n*. *dist* (*to-Gromov-completion* (*v n*)) (*u n*) < *e*) *sequentially*

by (*rule order-tendstoD*[*OF* ‹(λ*n*. *dist* (*to-Gromov-completion* (*v n*)) (*u n*)) ⟶ *0*›], *fact*)
then obtain *N2* **where** *N2*: ⋀*n*. *n* ≥ *N2* ⟹ *dist* (*to-Gromov-completion* (*v n*)) (*u n*) < *e*
unfolding *eventually-sequentially* **by** *auto*
have *eventually* (λ*n*. *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u n*) > *D2*) *sequentially*
by (*rule order-tendstoD*[*OF* ‹(λ*n*. *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u n*)) ⟶ ∞›], *fact*)
then obtain *N3* **where** *N3*: ⋀*n*. *n* ≥ *N3* ⟹ *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u n*) > *D2*
unfolding *eventually-sequentially* **by** *auto*
define *N* **where** *N* = *N1*+*N2*+*N3*
have *N*: *N* ≥ *N1* *N* ≥ *N2* *N* ≥ *N3* **unfolding** *N-def* **by** *auto*
have *ereal M* ≤ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (*v m*)) (*to-Gromov-completion* (*v n*))
if *n* ≥ *N* *m* ≥ *N* **for** *m n*
proof (*rule D1*(*3*))
have *dist* (*to-Gromov-completion* (*v m*)) (*to-Gromov-completion* (*v n*))
≤ *dist* (*to-Gromov-completion* (*v m*)) (*u m*) + *dist* (*u m*) (*u n*) + *dist* (*u n*) (*to-Gromov-completion* (*v n*))
by (*intro mono-intros*)
also have ... ≤ *e* + *e* + *e*
apply (*intro mono-intros*)
using *N1*[*of m n*] *N2*[*of n*] *N2*[*of m*] *that N* **by** (*auto simp add*: *dist-commute*)
also have ... ≤ *e1* **unfolding** *e-def* **by** *auto*
finally show *dist* (*to-Gromov-completion* (*v m*)) (*to-Gromov-completion* (*v n*)) ≤ *e1* **by** *simp*

have *e* ≤ *e2* **unfolding** *e-def* **using** ‹*e2* > *0*› **by** *auto*
have *D1* ≤ *extended-Gromov-product-at basepoint* (*u m*) (*to-Gromov-completion* (*v m*))
apply (*rule D2*(*3*)) **using** *N2*[*of m*] *N3*[*of m*] *that N* ‹*e* ≤ *e2*›
by (*auto simp add*: *dist-commute extended-Gromov-distance-commute*)
also have ... ≤ *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*to-Gromov-completion* (*v m*))
using *extended-Gromov-product-le-dist*[*of basepoint to-Gromov-completion* (*v m*) *u m*]
by (*simp add*: *extended-Gromov-product-at-commute*)
finally show *D1* ≤ *extended-Gromov-distance* (*to-Gromov-completion* (*v m*)) (*to-Gromov-completion basepoint*)
by (*simp add*: *extended-Gromov-distance-commute*)
qed
then have *M* ≤ *Gromov-product-at basepoint* (*v m*) (*v n*) **if** *n* ≥ *N* *m* ≥ *N* **for** *m n*
using *that* **by** *auto*
then show ∃ *N*. ∀ *n* ≥ *N*. ∀ *m* ≥ *N*. *M* ≤ *Gromov-product-at basepoint* (*v m*) (*v n*)

by *blast*
qed
then obtain *l* **where** *l*: *l* ∈ *Gromov-boundary* (*λn. to-Gromov-completion* (*v n*)) ⟶ *l*
using *Gromov-converging-at-boundary-converges* **by** *blast*
have (*λn. dist* (*u n*) *l*) ⟶ *0+0*
proof (*rule tendsto-sandwich*[*of λ-. 0 - - λn. dist* (*u n*) (*to-Gromov-completion* (*v n*)) + *dist* (*to-Gromov-completion* (*v n*)) *l*])
show (*λn. dist* (*u n*) (*to-Gromov-completion* (*v n*)) + *dist* (*to-Gromov-completion* (*v n*)) *l*) ⟶ *0* + *0*
apply (*intro tendsto-intros*)
using *iffD1*[*OF tendsto-dist-iff l*(*2*)] ‹(*λn. dist* (*to-Gromov-completion* (*v n*)) (*u n*)) ⟶ *0*›
by (*auto simp add*: *dist-commute*)
qed (*auto simp add*: *dist-triangle*)
then have *u* ⟶ *l*
using *iffD2*[*OF tendsto-dist-iff*] **by** *auto*
then have *u0* ⟶ *l*
unfolding *u-def* **using** *r*(*1*) ‹*Cauchy u0*› *Cauchy-converges-subseq* **by** *auto*
then show ∃ *l. u0* ⟶ *l*
by *auto*
next
case *False*
define *C* **where** *C* = *limsup* (*λn. extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u0 n*)) + *1*
have *C* < ∞ **unfolding** *C-def* **using** *False less-top* **by** *fastforce*
have ∗: *limsup* (*λn. extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u0 n*)) ≥ *0*
by (*intro le-Limsup always-eventually*, *auto*)
have *limsup* (*λn. extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u0 n*)) < *C*
unfolding *C-def* **using** *False* ∗ *ereal-add-left-cancel-less* **by** *force*
then have *eventually* (*λn. extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u0 n*) < *C*) *sequentially*
using *Limsup-lessD* **by** *blast*
then obtain *N* **where** *N*: ⋀*n. n* ≥ *N* ⟹ *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u0 n*) < *C*
unfolding *eventually-sequentially* **by** *auto*
define *r* **where** *r* = (*λn. n* + *N*)
have *r*: *strict-mono r* **unfolding** *r-def strict-mono-def* **by** *auto*
define *u* **where** *u* = (*u0 o r*)
have *Cauchy u*
using ‹*Cauchy u0*› *r*(*1*) *u-def* **by** (*simp add*: *Cauchy-subseq-Cauchy*)
have *u*: *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*u n*) ≤ *C* **for** *n*
unfolding *u-def comp-def r-def* **using** *N* **by** (*auto simp add*: *less-imp-le*)
define *v* **where** *v* = (*λn. from-Gromov-completion* (*u n*))
have *uv*: *u n* = *to-Gromov-completion* (*v n*) **for** *n*
unfolding *v-def* **apply** (*rule to-from-Gromov-completion*[*symmetric*]) **using**

u[*of n*] ‹$C < \infty$› **by** *auto*
  have *Cauchy v*
  proof (*rule metric-CauchyI*)
  obtain *a*::*real* **where** *a*: $a > 0$ $\bigwedge x\ y$::'*a Gromov-completion. extended-Gromov-distance* (*to-Gromov-completion basepoint*) $x \leq C \Longrightarrow$ *dist x y* $\leq a$
    $\Longrightarrow$ *esqrt*(*extended-Gromov-distance x y*) $\leq 2 * $ *ereal*(*dist x y*)
   using *inside-Gromov-distance-approx*[*OF* ‹$C < \infty$›] **by** *auto*
  fix *e*::*real* **assume** $e > 0$
  define *e2* **where** *e2* = *min* (*sqrt* (*e*/*2*) /*2*) *a*
  have $e2 > 0$ **unfolding** *e2-def* **using** ‹$e > 0$› ‹$a > 0$› **by** *auto*
  then obtain *N* **where** *N*: $\bigwedge m\ n.\ m \geq N \Longrightarrow n \geq N \Longrightarrow$ *dist* (*u m*) (*u n*) < *e2*
   using ‹*Cauchy u*› **unfolding** *Cauchy-def* **by** *blast*
  have *dist* (*v m*) (*v n*) < *e* **if** $n \geq N$ $m \geq N$ **for** *m n*
  proof −
   have *ereal*(*sqrt*(*dist* (*v m*) (*v n*))) = *esqrt*(*extended-Gromov-distance* (*u m*) (*u n*))
    unfolding *uv* **by** (*auto simp add*: *esqrt-ereal-ereal-sqrt*)
   also have ... $\leq 2 *$ *ereal*(*dist* (*u m*) (*u n*))
     apply (*rule a*(*2*)) **using** *u*[*of m*] *N*[*OF* ‹$m \geq N$› ‹$n \geq N$›] **unfolding** *e2-def* **by** *auto*
   also have ... = *ereal*(*2* ∗ *dist* (*u m*) (*u n*))
    by *simp*
   also have ... $\leq$ *ereal*(*2* ∗ *e2*)
    apply (*intro mono-intros*) **using** *N*[*OF* ‹$m \geq N$› ‹$n \geq N$›] *less-imp-le* **by** *auto*
   finally have *sqrt*(*dist* (*v m*) (*v n*)) $\leq 2 *$ *e2*
    using ‹$e2 > 0$› **by** *auto*
   also have ... $\leq$ *sqrt* (*e*/*2*)
    unfolding *e2-def* **by** *auto*
   finally have *dist* (*v m*) (*v n*) $\leq e/2$
    by *auto*
   then show *?thesis*
    using ‹$e > 0$› **by** *auto*
  qed
  then show $\exists M.\ \forall m \geq M.\ \forall n \geq M.$ *dist* (*v m*) (*v n*) < *e* **by** *auto*
 qed
 then obtain *l* **where** $v \longrightarrow l$
  using ‹*complete* (*UNIV*::'*a set*)› *complete-def* **by** *blast*
 then have $u \longrightarrow$ (*to-Gromov-completion l*)
  unfolding *uv* **by** *auto*
 then have $u0 \longrightarrow$ (*to-Gromov-completion l*)
  unfolding *u-def* **using** *r*(*1*) ‹*Cauchy u0*› *Cauchy-converges-subseq* **by** *auto*
 then show $\exists l.\ u0 \longrightarrow l$
  by *auto*
 qed
qed

instance *Gromov-completion*::({*Gromov-hyperbolic-space*, *complete-space*}) *com-*

plete-space
apply *standard*
using *Gromov-completion-complete complete-def convergent-def complete-UNIV*
by *auto*

When the original space is proper, i.e., closed balls are compact, and geodesic, then the Gromov completion (and therefore the Gromov boundary) are compact. The idea to extract a convergent subsequence of a sequence u_n in the boundary is to take the point v_n at distance T along a geodesic tending to the point u_n on the boundary, where T is fixed and large. The points v_n live in a bounded subset of the space, hence they have a convergent subsequence $v_{j(n)}$. It follows that $u_{j(n)}$ is almost converging, up to an error that tends to 0 when T tends to infinity. By a diagonal argument, we obtain a convergent subsequence of u_n.

As we have already proved that the space is complete, there is a shortcut to the above argument, avoiding subsequences and diagonal argument altogether. Indeed, in a complete space it suffices to show that for any $\epsilon > 0$ it is covered by finitely many balls of radius ϵ to get the compactness. This is what we do in the following proof, although the argument is precisely modelled on the first proof we have explained.

theorem *Gromov-completion-compact*:
assumes *proper* (*UNIV*::*'a*::*Gromov-hyperbolic-space-geodesic set*)
shows *compact* (*UNIV*::*'a Gromov-completion set*)
proof −
have $\exists$ *k. finite k* $\wedge$ (*UNIV*::*'a Gromov-completion set*) $\subseteq$ ($\bigcup$*x*$\in$*k. ball x e*) **if** *e* > *0* **for** *e*
proof −
define *D*::*real* **where** *D* = *max 0* (−*ln*(*e*/*4*)/*epsilonG*(*TYPE*(*'a*)))
have *D* $\geq$ *0* **unfolding** *D-def* **by** *auto*
have *exp*(−*epsilonG*(*TYPE*(*'a*)) ∗ *D*) $\leq$ *exp*(*ln* (*e* / *4*))
unfolding *D-def* **apply** (*intro mono-intros*) **unfolding** *max-def*
apply *auto*
using *constant-in-extended-predist-pos*(*1*)[**where** *?'a* = *'a*] **by** (*auto simp add*: *divide-simps*)
then have *exp*(−*epsilonG*(*TYPE*(*'a*)) ∗ *D*) $\leq$ *e*/*4* **using** ‹*e* > *0*› **by** *auto*
define *e0*::*real* **where** *e0* = *e* ∗ *e* / *16*
have *e0* > *0* **using** ‹*e* > *0*› **unfolding** *e0-def* **by** *auto*
obtain *k*::*'a set* **where** *k*: *finite k cball basepoint D* $\subseteq$ ($\bigcup$*x*$\in$*k. ball x e0*)
using *compact-eq-totally-bounded*[*of cball* (*basepoint*::*'a*) *D*] *assms* ‹*e0* > *0*›
unfolding *proper-def* **by** *auto*
have *A*: $\exists$ *y* $\in$ *k. dist* (*to-Gromov-completion y*) (*to-Gromov-completion x*) $\leq$ *e*/*4* **if** *dist basepoint x* $\leq$ *D* **for** *x*::*'a*
proof −
obtain *z* **where** *z*: *z* $\in$ *k dist z x* < *e0* **using** ‹*dist basepoint x* $\leq$ *D*› *k*(*2*) **by** *auto*
have *ereal*(*dist* (*to-Gromov-completion z*) (*to-Gromov-completion x*)) $\leq$ *esqrt*(*extended-Gromov-distance* (*to-Gromov-completion z*) (*to-Gromov-completion*

x))
by (*intro mono-intros*)
also have ... = $ereal(sqrt\ (dist\ z\ x))$
by *auto*
finally have *dist* (*to-Gromov-completion z*) (*to-Gromov-completion x*) $\leq$ *sqrt* (*dist z x*)
by *auto*
also have ... $\leq$ *sqrt e0*
using *z*(*2*) **by** *auto*
also have ... $\leq e/4$
unfolding *e0-def* **using** ‹$e > 0$› **by** (*auto simp add*: *less-imp-le real-sqrt-divide*)
finally have *dist* (*to-Gromov-completion z*) (*to-Gromov-completion x*) $\leq e/4$
by *auto*
then show *?thesis*
using ‹$z \in k$› **by** *auto*
qed
have *B*: $\exists y \in k$. *dist* (*to-Gromov-completion y*) (*to-Gromov-completion x*) $\leq$ $e/2$ **for** *x*
proof (*cases dist basepoint x* $\leq$ *D*)
case *True*
have $e/4 \leq e/2$ **using** ‹$e > 0$› **by** *auto*
then show *?thesis* **using** *A*[*OF True*] **by** *force*
next
case *False*
define *x2* **where** *x2* = *geodesic-segment-param* {*basepoint*−−*x*} *basepoint D*
have ∗: *Gromov-product-at basepoint x x2* = *D*
unfolding *x2-def* **apply** (*rule Gromov-product-geodesic-segment*) **using** *False* ‹$D \geq 0$› **by** *auto*
have *ereal*(*dist* (*to-Gromov-completion x*) (*to-Gromov-completion x2*))
$\leq$ *eexp* (− *epsilonG*(*TYPE*('*a*)) ∗ *extended-Gromov-product-at basepoint* (*to-Gromov-completion x*) (*to-Gromov-completion x2*))
by (*intro mono-intros*)
also have ... = *eexp* (− *epsilonG*(*TYPE*('*a*)) ∗ *ereal D*)
using ∗ **by** *auto*
also have ... = *ereal*(*exp*(−*epsilonG*(*TYPE*('*a*)) ∗ *D*))
by *auto*
also have ... $\leq$ *ereal*($e/4$)
by (*intro mono-intros*, *fact*)
finally have *dist* (*to-Gromov-completion x*) (*to-Gromov-completion x2*) $\leq$ $e/4$
using ‹$e > 0$› **by** *auto*
have *dist basepoint x2* $\leq$ *D*
unfolding *x2-def* **using** *False* ‹$0 \leq D$› **by** *auto*
then obtain *y* **where** $y \in k$ *dist* (*to-Gromov-completion y*) (*to-Gromov-completion x2*) $\leq e/4$
using *A* **by** *auto*
have *dist* (*to-Gromov-completion y*) (*to-Gromov-completion x*)
$\leq$ *dist* (*to-Gromov-completion y*) (*to-Gromov-completion x2*) + *dist* (*to-Gromov-completion x*) (*to-Gromov-completion x2*)

by (*intro mono-intros*)
also have ... $\leq$ *e/4* + *e/4*
by (*intro mono-intros*, *fact*, *fact*)
also have ... = *e/2* **by** *simp*
finally show *?thesis* **using** ‹*y* ∈ *k*› **by** *auto*
qed
have *C*: ∃ *y* ∈ *k*. *dist* (*to-Gromov-completion y*) *x* < *e* **for** *x*
proof −
obtain *x1* **where** *x1*: *dist x x1* < *e/2 x1* ∈ *range to-Gromov-completion*
using *to-Gromov-completion-range-dense* ‹*e* > *0*›
by (*metis* (*no-types*, *opaque-lifting*) *UNIV-I closure-approachableD divide-pos-pos zero-less-numeral*)
then obtain *z* **where** *z*: *x1* = *to-Gromov-completion z* **by** *auto*
then obtain *y* **where** *y*: *y* ∈ *k dist* (*to-Gromov-completion y*) (*to-Gromov-completion z*) $\leq$ *e/2*
using *B* **by** *auto*
have *dist* (*to-Gromov-completion y*) *x* $\leq$
dist (*to-Gromov-completion y*) (*to-Gromov-completion z*) + *dist x x1*
unfolding *z* **by** (*intro mono-intros*)
also have ... < *e/2* + *e/2*
using *x1*(*1*) *y*(*2*) **by** *auto*
also have ... = *e*
by *auto*
finally show *?thesis* **using** ‹*y* ∈ *k*› **by** *auto*
qed
show *?thesis*
apply (*rule exI*[*of* - *to-Gromov-completion‘k*])
using *C* ‹*finite k*› **by** *auto*
qed
then show *?thesis*
unfolding *compact-eq-totally-bounded*
using *Gromov-completion-complete*[*OF complete-of-proper*[*OF assms*]] **by** *auto*
qed

If the inner space is second countable, so is its completion, as the former is dense in the latter.

instance *Gromov-completion*::({*Gromov-hyperbolic-space*, *second-countable-topology*}) *second-countable-topology*
proof
obtain *A*::*'a set* **where** *countable A closure A* = *UNIV*
using *second-countable-metric-dense-subset* **by** *auto*
define *Ab* **where** *Ab* = *to-Gromov-completion‘A*
have *range to-Gromov-completion* ⊆ *closure Ab*
unfolding *Ab-def*
by (*metis* ‹*closure A* = *UNIV*› *closed-closure closure-subset image-closure-subset to-Gromov-completion-continuous*)
then have *closure Ab* = *UNIV*
by (*metis closed-closure closure-minimal dual-order.antisym to-Gromov-completion-range-dense top-greatest*)

moreover have *countable Ab* **unfolding** *Ab-def* **using** ‹*countable A*› **by** *auto*
ultimately have $\exists$ *Ab*::′*a Gromov-completion set. countable Ab* $\wedge$ *closure Ab* = *UNIV*
by *auto*
then show $\exists$ *B*::′*a Gromov-completion set set. countable B* $\wedge$ *open* = *generate-topology B*
using *second-countable-iff-dense-countable-subset topological-basis-imp-subbasis*
by *auto*
qed

The same follows readily for the Polish space property.

instance *metric-completion*::({*Gromov-hyperbolic-space*, *polish-space*}) *polish-space*
by *standard*

15.7 The Gromov completion of the real line.

We show in the paragraph that the Gromov completion of the real line is obtained by adding one point at $+\infty$ and one point at $-\infty$. In other words, it coincides with ereal.

To show this, we have to understand which sequences of reals are Gromov-converging to the boundary. We show in the next lemma that they are exactly the sequences that converge to $-\infty$ or to $+\infty$.

lemma *real-Gromov-converging-to-boundary*:
fixes *u*::*nat* $\Rightarrow$ *real*
shows *Gromov-converging-at-boundary u* $\longleftrightarrow$ ((*u* $\longrightarrow$ ∞) $\vee$ (*u* $\longrightarrow$ $-\infty$))
proof −
have ∗: *Gromov-product-at 0 m n* $\geq$ *min m n* **for** *m n*::*real*
unfolding *Gromov-product-at-def dist-real-def* **by** *auto*
have *A*: *Gromov-converging-at-boundary u* **if** *u* $\longrightarrow$ ∞ **for** *u*::*nat* $\Rightarrow$ *real*
proof (*rule Gromov-converging-at-boundaryI*[*of 0*])
fix *M*::*real*
have *eventually* (λ*n. ereal* (*u n*) > *M*) *sequentially*
by (*rule order-tendstoD*(*1*)[*OF* ‹*u* $\longrightarrow$ ∞›, *of ereal M*], *auto*)
then obtain *N* **where** $\bigwedge$*n. n* $\geq$ *N* $\Longrightarrow$ *ereal* (*u n*) > *M*
unfolding *eventually-sequentially* **by** *auto*
then have *A*: *u n* $\geq$ *M* **if** *n* $\geq$ *N* **for** *n*
by (*simp add*: *less-imp-le that*)
have *M* $\leq$ *Gromov-product-at 0* (*u m*) (*u n*) **if** *n* $\geq$ *N m* $\geq$ *N* **for** *m n*
using *A*[*OF* ‹*m* $\geq$ *N*›] *A*[*OF* ‹*n* $\geq$ *N*›] ∗[*of u m u n*] **by** *auto*
then show $\exists$ *N*. $\forall$ *n* $\geq$ *N*. $\forall$ *m* $\geq$ *N*. *M* $\leq$ *Gromov-product-at 0* (*u m*) (*u n*)
by *auto*
qed
have ∗: *Gromov-product-at 0 m n* $\geq$ − *max m n* **for** *m n*::*real*
unfolding *Gromov-product-at-def dist-real-def* **by** *auto*
have *B*: *Gromov-converging-at-boundary u* **if** *u* $\longrightarrow$ $-\infty$ **for** *u*::*nat* $\Rightarrow$ *real*
proof (*rule Gromov-converging-at-boundaryI*[*of 0*])
fix *M*::*real*
have *eventually* (λ*n. ereal* (*u n*) < − *M*) *sequentially*

by (*rule order-tendstoD*(*2*)[*OF* ‹$u \longrightarrow -\infty$›, *of ereal* ($-M$)], *auto*)
then obtain N **where** $\bigwedge n.\ n \geq N \Longrightarrow$ *ereal* ($u\ n$) $< -\ M$
unfolding *eventually-sequentially* **by** *auto*
then have A: $u\ n \leq -\ M$ **if** $n \geq N$ **for** n
by (*simp add*: *less-imp-le that*)
have $M \leq$ *Gromov-product-at 0* ($u\ m$) ($u\ n$) **if** $n \geq N\ m \geq N$ **for** $m\ n$
using A[*OF* ‹$m \geq N$›] A[*OF* ‹$n \geq N$›] $*$[*of* $u\ m\ u\ n$] **by** *auto*
then show $\exists N.\ \forall n \geq N.\ \forall m \geq N.\ M \leq$ *Gromov-product-at 0* ($u\ m$) ($u\ n$)
by *auto*
qed
have L: ($u \longrightarrow \infty$) $\vee$ ($u \longrightarrow -\ \infty$) **if** *Gromov-converging-at-boundary u* **for** u::*nat* $\Rightarrow$ *real*
proof −
have ($\lambda n.$ *abs*($u\ n$)) $\longrightarrow \infty$
using *Gromov-converging-at-boundary-imp-unbounded*[*OF that*, *of 0*] **unfolding** *dist-real-def* **by** *auto*

obtain r **where** r: *strict-mono* r ($\lambda n.$ *ereal* (u ($r\ n$))) $\longrightarrow$ *liminf* ($\lambda n.$ *ereal*($u\ n$))
using *liminf-subseq-lim*[*of* $\lambda n.$ *ereal*($u\ n$)] **unfolding** *comp-def* **by** *auto*
have ($\lambda n.$ *abs*(*ereal* (u ($r\ n$)))) $\longrightarrow$ *abs*(*liminf* ($\lambda n.$ *ereal*($u\ n$)))
apply (*intro tendsto-intros*) **using** r(*2*) **by** *auto*
moreover have ($\lambda n.$ *abs*(*ereal* (u ($r\ n$)))) $\longrightarrow \infty$
using ‹($\lambda n.$ *abs*($u\ n$)) $\longrightarrow \infty$› **apply** *auto*
using *filterlim-compose filterlim-subseq*[*OF* r(*1*)] **by** *blast*
ultimately have A: *abs*(*liminf* ($\lambda n.$ *ereal*($u\ n$))) $= \infty$
using *LIMSEQ-unique* **by** *auto*

obtain r **where** r: *strict-mono* r ($\lambda n.$ *ereal* (u ($r\ n$))) $\longrightarrow$ *limsup* ($\lambda n.$ *ereal*($u\ n$))
using *limsup-subseq-lim*[*of* $\lambda n.$ *ereal*($u\ n$)] **unfolding** *comp-def* **by** *auto*
have ($\lambda n.$ *abs*(*ereal* (u ($r\ n$)))) $\longrightarrow$ *abs*(*limsup* ($\lambda n.$ *ereal*($u\ n$)))
apply (*intro tendsto-intros*) **using** r(*2*) **by** *auto*
moreover have ($\lambda n.$ *abs*(*ereal* (u ($r\ n$)))) $\longrightarrow \infty$
using ‹($\lambda n.$ *abs*($u\ n$)) $\longrightarrow \infty$› **apply** *auto*
using *filterlim-compose filterlim-subseq*[*OF* r(*1*)] **by** *blast*
ultimately have B: *abs*(*limsup* ($\lambda n.$ *ereal*($u\ n$))) $= \infty$
using *LIMSEQ-unique* **by** *auto*

have $\neg$(*liminf* $u = -\ \infty \wedge$ *limsup* $u = \infty$)
proof (*rule ccontr*, *auto*)
assume *liminf* $u = -\infty$ *limsup* $u = \infty$
have $\exists N.\ \forall n \geq N.\ \forall m \geq N.$ *Gromov-product-at 0* ($u\ m$) ($u\ n$) ≥ 1
using *that* **unfolding** *Gromov-converging-at-boundary-def* **by** *blast*
then obtain N **where** N: $\bigwedge m\ n.\ m \geq N \Longrightarrow n \geq N \Longrightarrow$ *Gromov-product-at 0* ($u\ m$) ($u\ n$) ≥ 1
by *auto*
have $\exists n \geq N.$ *ereal*($u\ n$) $>$ *ereal 0*
apply (*rule limsup-obtain*) **unfolding** ‹*limsup* $u = \infty$› **by** *auto*

then obtain n **where** n: $n \geq N$ u $n > 0$ **by** *auto*

have $\exists\, n \geq N.$ *ereal*$(u\ n) <$ *ereal* 0
apply (*rule liminf-obtain*) **unfolding** ‹*liminf* $u = -\infty$› **by** *auto*
then obtain m **where** m: $m \geq N$ u $m < 0$ **by** *auto*

have *Gromov-product-at* 0 $(u\ m)$ $(u\ n) = 0$
unfolding *Gromov-product-at-def dist-real-def* **using** m n **by** *auto*
then show *False* **using** $N[OF\ m(1)\ n(1)]$ **by** *auto*
qed
then have *liminf* $u = \infty \vee$ *limsup* $u = -\infty$
using A B **by** *auto*
then show *?thesis*
by (*simp add*: *Liminf-PInfty Limsup-MInfty*)
qed
show *?thesis* **using** L A B **by** *auto*
qed

There is one single point at infinity in the Gromov completion of reals, i.e., two sequences tending to infinity are equivalent.

lemma *real-Gromov-completion-rel-PInf*:
fixes u v::*nat* $\Rightarrow$ *real*
assumes $u \longrightarrow \infty$ $v \longrightarrow \infty$
shows *Gromov-completion-rel* u v
proof $-$
have $*$: *Gromov-product-at* 0 m $n \geq$ *min* m n **for** m n::*real*
unfolding *Gromov-product-at-def dist-real-def* **by** *auto*
have $**$: *Gromov-product-at* a m $n \geq$ *min* m $n - $ *abs* a **for** m n a::*real*
using *Gromov-product-at-diff1*[*of* 0 m n a] $*$[*of* m n] **by** *auto*
have ($\lambda n.$ *Gromov-product-at* a $(u\ n)$ $(v\ n)$) $\longrightarrow \infty$ **for** a
proof (*rule tendsto-sandwich*[*of* $\lambda n.$ *min* $(u\ n)$ $(v\ n) -$ *abs* a - - $\lambda n.\ \infty$])
have *ereal* (*min* $(u\ n)$ $(v\ n) - |a|) \leq$ *ereal* (*Gromov-product-at* a $(u\ n)$ $(v\ n)$) **for** n
using $**$[*of* u n v n a] **by** *auto*
then show $\forall_F$ n *in sequentially*. *ereal* (*min* $(u\ n)$ $(v\ n) - |a|) \leq$ *ereal* (*Gromov-product-at* a $(u\ n)$ $(v\ n)$)
by *auto*
have ($\lambda x.$ *min* (*ereal*$(u\ x)$) (*ereal* $(v\ x)$) $-$ *ereal* $|a|$) $\longrightarrow$ *min* ∞ ∞ $-$ *ereal* $|a|$
apply (*intro tendsto-intros*) **using** *assms* **by** *auto*
then show ($\lambda x.$ *ereal* (*min* $(u\ x)$ $(v\ x) - |a|$)) $\longrightarrow \infty$
apply *auto* **unfolding** *ereal-minus*(1)[*symmetric*] **by** *auto*
qed (*auto*)
moreover have *Gromov-converging-at-boundary* u *Gromov-converging-at-boundary* v
using *real-Gromov-converging-to-boundary assms* **by** *auto*
ultimately show *?thesis* **unfolding** *Gromov-completion-rel-def* **by** *auto*
qed

There is one single point at minus infinity in the Gromov completion of reals, i.e., two sequences tending to minus infinity are equivalent.

lemma *real-Gromov-completion-rel-MInf*:
  fixes *u v*::*nat* ⇒ *real*
  assumes $u \longrightarrow -\infty$ $v \longrightarrow -\infty$
  shows *Gromov-completion-rel u v*
proof −
  have ∗: *Gromov-product-at 0 m n* ≥ − *max m n* **for** *m n*::*real*
    unfolding *Gromov-product-at-def dist-real-def* **by** *auto*
  have ∗∗: *Gromov-product-at a m n* ≥ − *max m n* − *abs a* **for** *m n a*::*real*
    using *Gromov-product-at-diff1* [*of 0 m n a*] ∗[*of m n*] **by** *auto*
  have (λ*n*. *Gromov-product-at a* (*u n*) (*v n*)) ⟶ ∞ **for** *a*
  proof (*rule tendsto-sandwich*[*of* λ*n*. *min* (−*u n*) (−*v n*) − *abs a* - - λ*n*. ∞])
    have *ereal* (*min* (−*u n*) (−*v n*) − |*a*|) ≤ *ereal* (*Gromov-product-at a* (*u n*) (*v n*)) **for** *n*
      using ∗∗[*of u n v n a*] **by** *auto*
     then show ∀$_F$ *n in sequentially*. *ereal* (*min* (−*u n*) (−*v n*) − |*a*|) ≤ *ereal* (*Gromov-product-at a* (*u n*) (*v n*))
      by *auto*
    have (λ*x*. *min* (−*ereal*(*u x*)) (−*ereal* (*v x*)) − *ereal* |*a*|) ⟶ *min* (−(−∞)) (−(−∞)) − *ereal* |*a*|
      apply (*intro tendsto-intros*) **using** *assms* **by** *auto*
    then show (λ*x*. *ereal* (*min* (−*u x*) (−*v x*) − |*a*|)) ⟶ ∞
      apply *auto* **unfolding** *ereal-minus*(*1*)[*symmetric*] **by** *auto*
  qed (*auto*)
  moreover have *Gromov-converging-at-boundary u Gromov-converging-at-boundary v*
    using *real-Gromov-converging-to-boundary assms* **by** *auto*
  ultimately show *?thesis* **unfolding** *Gromov-completion-rel-def* **by** *auto*
qed

It follows from the two lemmas above that the Gromov completion of reals is obtained by adding one single point at infinity and one single point at minus infinity. Hence, it is in bijection with the extended reals.

function *to-real-Gromov-completion*::*ereal* ⇒ *real Gromov-completion*
  where *to-real-Gromov-completion* (*ereal r*) = *to-Gromov-completion r*
  | *to-real-Gromov-completion* (∞) = *abs-Gromov-completion* (λ*n*. *n*)
  | *to-real-Gromov-completion* (−∞) = *abs-Gromov-completion* (λ*n*. −*n*)
by (*auto intro*: *ereal-cases*)
termination by *standard* (*rule wf-empty*)

To prove the bijectivity, we prove by hand injectivity and surjectivity using the above lemmas.

lemma *bij-to-real-Gromov-completion*:
  bij to-real-Gromov-completion
proof −
  have [*simp*]: *Gromov-completion-rel* (λ*n*. *n*) (λ*n*. *n*)
    by (*intro real-Gromov-completion-rel-PInf tendsto-intros*)

have [*simp*]: *Gromov-completion-rel* ($\lambda n.$ $-$*real* n) ($\lambda n.$ $-$*real* n)
by (*intro real-Gromov-completion-rel-MInf tendsto-intros*)

have $\exists x.$ *to-real-Gromov-completion* $x = y$ **for** y
proof (*cases* y)
case (*to-Gromov-completion* x)
then have $y =$ *to-real-Gromov-completion* x **by** *auto*
then show *?thesis* **by** *blast*
next
case *boundary*
define u **where** u: $u =$ *rep-Gromov-completion* y
have y: *abs-Gromov-completion* $u = y$ *Gromov-completion-rel* u u
unfolding u **using** *Quotient3-abs-rep*[*OF Quotient3-Gromov-completion*] *Quotient3-rep-reflp*[*OF Quotient3-Gromov-completion*] **by** *auto*
have *Gromov-converging-at-boundary* u
using u *boundary* **by** (*simp add*: *Gromov-boundary-rep-converging*)
then have ($u \longrightarrow \infty$) $\vee$ ($u \longrightarrow -\infty$) **using** *real-Gromov-converging-to-boundary* **by** *auto*
then show *?thesis*
proof
assume $u \longrightarrow \infty$
have *abs-Gromov-completion* ($\lambda n.$ n) $=$ *abs-Gromov-completion* u
apply (*rule Quotient3-rel-abs*[*OF Quotient3-Gromov-completion*])
by (*intro real-Gromov-completion-rel-PInf*[*OF* - ‹$u \longrightarrow \infty$›] *tendsto-intros*)
then have *to-real-Gromov-completion* $\infty = y$
unfolding y **by** *auto*
then show *?thesis* **by** *blast*
next
assume $u \longrightarrow -\infty$
have *abs-Gromov-completion* ($\lambda n.$ $-$*real* n) $=$ *abs-Gromov-completion* u
apply (*rule Quotient3-rel-abs*[*OF Quotient3-Gromov-completion*])
by (*intro real-Gromov-completion-rel-MInf*[*OF* - ‹$u \longrightarrow -\infty$›] *tendsto-intros*)
then have *to-real-Gromov-completion* ($-\infty$) $= y$
unfolding y **by** *auto*
then show *?thesis* **by** *blast*
qed
qed
then have *surj to-real-Gromov-completion*
unfolding *surj-def* **by** *metis*

have *to-real-Gromov-completion* $\infty \in$ *Gromov-boundary*
to-real-Gromov-completion ($-\infty$) $\in$ *Gromov-boundary*
by (*auto intro*!: *abs-Gromov-completion-in-Gromov-boundary tendsto-intros simp add*: *real-Gromov-converging-to-boundary*)
moreover have *to-real-Gromov-completion* $\infty \neq$ *to-real-Gromov-completion* ($-\infty$)
proof $-$
have *Gromov-product-at* *0* (*real* n) ($-$*real* n) $=$ *0* **for** n::*nat*
unfolding *Gromov-product-at-def dist-real-def* **by** *auto*

then have $*$: $(\lambda n.\ \mathit{ereal}(\mathit{Gromov\text{-}product\text{-}at}\ 0\ (\mathit{real}\ n)\ (-\mathit{real}\ n))) \longrightarrow \mathit{ereal}$ 0 **by** *auto*
have $\neg((\lambda n.\ \mathit{Gromov\text{-}product\text{-}at}\ 0\ (\mathit{real}\ n)\ (-\mathit{real}\ n)) \longrightarrow \infty)$
using *LIMSEQ-unique*[*OF* $*$] **by** *fastforce*
then have $\neg(\mathit{Gromov\text{-}completion\text{-}rel}\ (\lambda n.\ n)\ (\lambda n.\ -n))$
unfolding *Gromov-completion-rel-def* **by** *auto* (*metis nat.simps*(*3*) *of-nat-0 of-nat-eq-0-iff*)
then show *?thesis*
using *Quotient3-rel*[*OF Quotient3-Gromov-completion*, *of* $\lambda n.\ n\ \lambda n.\ -\mathit{real}\ n$] **by** *auto*
qed
ultimately have $x = y$ **if** *to-real-Gromov-completion* $x =$ *to-real-Gromov-completion* y **for** $x\ y$
using *that injD*[*OF to-Gromov-completion-inj*] **apply** (*cases x y rule*: *ereal2-cases*)
by (*auto*) (*metis not-in-Gromov-boundary'*)+
then have *inj to-real-Gromov-completion*
unfolding *inj-def* **by** *auto*
then show *bij to-real-Gromov-completion*
using ‹*surj to-real-Gromov-completion*› **by** (*simp add*: *bijI*)
qed

Next, we prove that we have a homeomorphism. By compactness of ereals, it suffices to show that the inclusion map is continuous everywhere. It would be a pain to distinguish all the time if points are at infinity or not, we rather use a criterion saying that it suffices to prove sequential continuity for sequences taking values in a dense subset of the space, here we take the reals. Hence, it suffices to show that if a sequence of reals v_n converges to a limit a in the extended reals, then the image of v_n in the Gromov completion (which is an inner point) converges to the point corresponding to a. We treat separately the cases $a \in \mathbb{R}$, $a = \infty$ and $a = -\infty$. In the first case, everything is trivial. In the other cases, we have characterized in general sequences inside the space that converge to a boundary point, as sequences in the equivalence class defining this boundary point. Since we have described explicitly these equivalence classes in the case of the Gromov completion of the reals (they are respectively the sequences tending to ∞ and to $-\infty$), the result follows readily without any additional computation.

proposition *homeo-to-real-Gromov-completion*:
homeomorphism-on UNIV to-real-Gromov-completion
proof (*rule homeomorphism-on-compact*)
show *inj to-real-Gromov-completion*
using *bij-to-real-Gromov-completion* **by** (*simp add*: *bij-betw-def*)
show *compact* (*UNIV*::*ereal set*)
by (*simp add*: *compact-UNIV*)
show *continuous-on UNIV to-real-Gromov-completion*
proof (*rule continuous-on-extension-sequentially*[*of* - $\{-\infty<..<\infty\}$], *auto*)
fix u::*nat* $\Rightarrow$ *ereal* **and** b::*ereal* **assume** u: $\forall n.\ u\ n \neq -\infty \wedge u\ n \neq \infty\ u \longrightarrow b$

define *v* **where** $v = (\lambda n.\ \textit{real-of-ereal}\ (u\ n))$
have *uv*: $u\ n = \textit{ereal}\ (v\ n)$ **for** *n*
using *u* **unfolding** *v-def* **by** (*simp add*: *ereal-infinity-cases ereal-real*)
show $(\lambda n.\ \textit{to-real-Gromov-completion}\ (u\ n)) \longrightarrow \textit{to-real-Gromov-completion}\ b$
proof (*cases b*)
case (*real r*)
then show *?thesis* **using** ‹$u \longrightarrow b$› **unfolding** *uv* **by** *auto*
next
case *PInf*
then have $*$: $(\lambda n.\ \textit{ereal}\ (v\ n)) \longrightarrow \infty$ **using** ‹$u \longrightarrow b$› **unfolding** *uv* **by** *auto*
have *A*: *Gromov-completion-rel real v Gromov-completion-rel real real Gromov-completion-rel v v*
by (*auto intro*!: *real-Gromov-completion-rel-PInf* $*$ *tendsto-intros*)
then have *B*: *abs-Gromov-completion v* = *abs-Gromov-completion real*
using *Quotient3-rel-abs*[*OF Quotient3-Gromov-completion*] **by** *force*
then show *?thesis* **using** ‹$u \longrightarrow b$› *PInf*
unfolding *uv* **apply** *auto*
apply (*subst Gromov-completion-converge-to-boundary*)
using *id-nat-ereal-tendsto-PInf real-Gromov-converging-to-boundary A B* **by** *auto*
next
case *MInf*
then have $*$: $(\lambda n.\ \textit{ereal}\ (v\ n)) \longrightarrow -\infty$ **using** ‹$u \longrightarrow b$› **unfolding** *uv* **by** *auto*
have *A*: *Gromov-completion-rel* $(\lambda n.\ -\textit{real}\ n)$ *v Gromov-completion-rel* $(\lambda n.\ -\textit{real}\ n)$ $(\lambda n.\ -\textit{real}\ n)$ *Gromov-completion-rel v v*
by (*auto intro*!: *real-Gromov-completion-rel-MInf* $*$ *tendsto-intros*)
then have *B*: *abs-Gromov-completion v* = *abs-Gromov-completion* $(\lambda n.\ -\textit{real}\ n)$
using *Quotient3-rel-abs*[*OF Quotient3-Gromov-completion*] **by** *force*
then show *?thesis* **using** ‹$u \longrightarrow b$› *MInf*
unfolding *uv* **apply** *auto*
apply (*subst Gromov-completion-converge-to-boundary*)
using *id-nat-ereal-tendsto-PInf real-Gromov-converging-to-boundary A B*
by (*auto simp add*: *ereal-minus-real-tendsto-MInf*)
qed
qed
qed

end

theory *Boundary-Extension*
imports *Morse-Gromov-Theorem Gromov-Boundary*
begin

16 Extension of quasi-isometries to the boundary

In this section, we show that a quasi-isometry between geodesic Gromov hyperbolic spaces extends to a homeomorphism between their boundaries.

Applying a quasi-isometry on a geodesic triangle essentially sends it to a geodesic triangle, in hyperbolic spaces. It follows that, up to an additive constant, the Gromov product, which is the distance to the center of the triangle, is multiplied by a constant between λ^{-1} and λ when one applies a quasi-isometry. This argument is given in the next lemma. This implies that two points are close in the Gromov completion if and only if their images are also close in the Gromov completion of the image. Essentially, this lemma implies that a quasi-isometry has a continuous extension to the Gromov boundary, which is a homeomorphism.

lemma *Gromov-product-at-quasi-isometry*:
fixes *f*::*'a*::*Gromov-hyperbolic-space-geodesic* ⇒ *'b*::*Gromov-hyperbolic-space-geodesic*
assumes *lambda C−quasi-isometry f*
shows *Gromov-product-at (f x) (f y) (f z) ≥ Gromov-product-at x y z / lambda − 187 ∗ lambda^2 ∗ (C + deltaG(TYPE('a)) + deltaG(TYPE('b)))*
Gromov-product-at (f x) (f y) (f z) ≤ lambda ∗ Gromov-product-at x y z + 187 ∗ lambda^2 ∗ (C + deltaG(TYPE('a)) + deltaG(TYPE('b)))
proof −
have *lambda ≥ 1 C ≥ 0* **using** *quasi-isometry-onD*[*OF assms*(*1*)] **by** *auto*
define *D* **where** *D = 92 ∗ lambda^2 ∗ (C + deltaG(TYPE('b)))*
have *Dxy*: *hausdorff-distance (f'{x−−y}) {f x−−f y} ≤ D*
unfolding *D-def* **apply** (*rule geodesic-quasi-isometric-image*[*OF assms*(*1*)]) **by** *auto*
have *Dyz*: *hausdorff-distance (f'{y−−z}) {f y−−f z} ≤ D*
unfolding *D-def* **apply** (*rule geodesic-quasi-isometric-image*[*OF assms*(*1*)]) **by** *auto*
have *Dxz*: *hausdorff-distance (f'{x−−z}) {f x−−f z} ≤ D*
unfolding *D-def* **apply** (*rule geodesic-quasi-isometric-image*[*OF assms*(*1*)]) **by** *auto*

define *E* **where** *E = (lambda ∗ (4 ∗ deltaG(TYPE('a))) + C) + D*
have *E ≥ 0* **unfolding** *E-def D-def* **using** ‹*lambda ≥ 1*› ‹*C ≥ 0*› **by** *auto*
obtain *w* **where** *w*: *infdist w {x−−y} ≤ 4 ∗ deltaG(TYPE('a))*
infdist w {x−−z} ≤ 4 ∗ deltaG(TYPE('a))
infdist w {y−−z} ≤ 4 ∗ deltaG(TYPE('a))
dist w x = Gromov-product-at x y z
using *slim-triangle*[*of {x−−y} x y {x−−z} z {y−−z}*] **by** *auto*
have *infdist (f w) {f x−−f y} ≤ infdist (f w) (f'{x−−y}) + hausdorff-distance (f'{x−−y}) {f x−−f y}*
by (*intro mono-intros quasi-isometry-on-bounded*[*OF quasi-isometry-on-subset*[*OF assms*(*1*)], *of {x−−y}*], *auto*)
also have ... *≤ (lambda ∗ infdist w {x−−y} + C) + D*
apply (*intro mono-intros*) **using** *quasi-isometry-on-infdist*[*OF assms*(*1*)] *Dxy* **by** *auto*

also have ... $\leq$ (*lambda* $*$ (*4* $*$ *deltaG*(*TYPE*('*a*))) + *C*) + *D*
apply (*intro mono-intros*) **using** *w* ‹*lambda* $\geq$ *1*› **by** *auto*
finally have *Exy*: *infdist* (*f w*) {*f x*−−*f y*} $\leq$ *E* **unfolding** *E-def* **by** *auto*

have *infdist* (*f w*) {*f y*−−*f z*} $\leq$ *infdist* (*f w*) (*f*'{*y*−−*z*}) + *hausdorff-distance* (*f*'{*y*−−*z*}) {*f y*−−*f z*}
by (*intro mono-intros quasi-isometry-on-bounded*[*OF quasi-isometry-on-subset*[*OF assms*(*1*)], *of* {*y*−−*z*}], *auto*)
also have ... $\leq$ (*lambda* $*$ *infdist w* {*y*−−*z*} + *C*) + *D*
apply (*intro mono-intros*) **using** *quasi-isometry-on-infdist*[*OF assms*(*1*)] *Dyz* **by** *auto*
also have ... $\leq$ (*lambda* $*$ (*4* $*$ *deltaG*(*TYPE*('*a*))) + *C*) + *D*
apply (*intro mono-intros*) **using** *w* ‹*lambda* $\geq$ *1*› **by** *auto*
finally have *Eyz*: *infdist* (*f w*) {*f y*−−*f z*} $\leq$ *E* **unfolding** *E-def* **by** *auto*

have *infdist* (*f w*) {*f x*−−*f z*} $\leq$ *infdist* (*f w*) (*f*'{*x*−−*z*}) + *hausdorff-distance* (*f*'{*x*−−*z*}) {*f x*−−*f z*}
by (*intro mono-intros quasi-isometry-on-bounded*[*OF quasi-isometry-on-subset*[*OF assms*(*1*)], *of* {*x*−−*z*}], *auto*)
also have ... $\leq$ (*lambda* $*$ *infdist w* {*x*−−*z*} + *C*) + *D*
apply (*intro mono-intros*) **using** *quasi-isometry-on-infdist*[*OF assms*(*1*)] *Dxz* **by** *auto*
also have ... $\leq$ (*lambda* $*$ (*4* $*$ *deltaG*(*TYPE*('*a*))) + *C*) + *D*
apply (*intro mono-intros*) **using** *w* ‹*lambda* $\geq$ *1*› **by** *auto*
finally have *Exz*: *infdist* (*f w*) {*f x*−−*f z*} $\leq$ *E* **unfolding** *E-def* **by** *auto*

have *2* $*$ ((*1*/*lambda* $*$ *dist w x* − *C*)) $\leq$ *2* $*$ *dist* (*f w*) (*f x*)
using *quasi-isometry-onD*(*2*)[*OF assms*(*1*), *of w x*] **by** *auto*
also have ... = (*dist* (*f w*) (*f x*) + *dist* (*f w*) (*f y*)) + (*dist* (*f w*) (*f x*) + *dist* (*f w*) (*f z*)) − (*dist* (*f w*) (*f y*) + *dist* (*f w*) (*f z*))
by *auto*
also have ... $\leq$ (*dist* (*f x*) (*f y*) + *2* $*$ *infdist* (*f w*) {*f x*−−*f y*}) + (*dist* (*f x*) (*f z*) + *2* $*$ *infdist* (*f w*) {*f x*−−*f z*}) − *dist* (*f y*) (*f z*)
by (*intro geodesic-segment-distance mono-intros*, *auto*)
also have ... $\leq$ *2* $*$ *Gromov-product-at* (*f x*) (*f y*) (*f z*) + *4* $*$ *E*
unfolding *Gromov-product-at-def* **using** *Exy Exz* **by** (*auto simp add*: *algebra-simps divide-simps*)
finally have $*$: *Gromov-product-at x y z* / *lambda* − *C* − *2* $*$ *E* $\leq$ *Gromov-product-at* (*f x*) (*f y*) (*f z*)
unfolding *w*(*4*) **by** *simp*

have *2* $*$ *Gromov-product-at* (*f x*) (*f y*) (*f z*) − *2* $*$ *E* $\leq$ *2* $*$ *Gromov-product-at* (*f x*) (*f y*) (*f z*) − *2* $*$ *infdist* (*f w*) {*f y*−−*f z*}
using *Eyz* **by** *auto*
also have ... = *dist* (*f x*) (*f y*) + *dist* (*f x*) (*f z*) − (*dist* (*f y*) (*f z*) + *2* $*$ *infdist* (*f w*) {*f y*−−*f z*})
unfolding *Gromov-product-at-def* **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... $\leq$ (*dist* (*f w*) (*f x*) + *dist* (*f w*) (*f y*)) + (*dist* (*f w*) (*f x*) + *dist* (*f w*) (*f z*)) − (*dist* (*f w*) (*f y*) + *dist* (*f w*) (*f z*))

by (*intro geodesic-segment-distance mono-intros*, *auto*)
also have ... $= 2 * dist\ (f\ w)\ (f\ x)$
by *auto*
also have ... $\leq 2 * (lambda * dist\ w\ x + C)$
using *quasi-isometry-onD*(*1*)[*OF assms*(*1*), *of w x*] by *auto*
finally have *Gromov-product-at* $(f\ x)\ (f\ y)\ (f\ z) \leq lambda * dist\ w\ x + C + E$
by *auto*
then have $**$: *Gromov-product-at* $(f\ x)\ (f\ y)\ (f\ z) \leq lambda *$ *Gromov-product-at* $x\ y\ z + C + 2 * E$
unfolding $w(4)$ using ‹$E \geq 0$› by *auto*

have $C + 2 * E = 3 * 1 * C + 8 * lambda * deltaG(TYPE('a)) + 184 * lambda\hat{}2 * C + 184 * lambda\hat{}2 * deltaG(TYPE('b))$
unfolding *E-def D-def* by (*auto simp add*: *algebra-simps*)
also have ... $\leq 3 * lambda\hat{}2 * C + 187 * lambda\hat{}2 * deltaG(TYPE('a)) + 184 * lambda\hat{}2 * C + 187 * lambda\hat{}2 * deltaG(TYPE('b))$
apply (*intro mono-intros*) using ‹$lambda \geq 1$› ‹$C \geq 0$› by *auto*
finally have I: $C + 2 * E \leq 187 * lambda\hat{}2 * (C + deltaG(TYPE('a)) + deltaG(TYPE('b)))$
by (*auto simp add*: *algebra-simps*)

show *Gromov-product-at* $(f\ x)\ (f\ y)\ (f\ z) \geq$ *Gromov-product-at* $x\ y\ z\ /\ lambda - 187 * lambda\hat{}2 * (C + deltaG(TYPE('a)) + deltaG(TYPE('b)))$
using $*$ I by *auto*
show *Gromov-product-at* $(f\ x)\ (f\ y)\ (f\ z) \leq lambda *$ *Gromov-product-at* $x\ y\ z + 187 * lambda\hat{}2 * (C + deltaG(TYPE('a)) + deltaG(TYPE('b)))$
using $**$ I by *auto*
qed

lemma *Gromov-converging-at-infinity-quasi-isometry*:
fixes f::$'a$::*Gromov-hyperbolic-space-geodesic* $\Rightarrow$ $'b$::*Gromov-hyperbolic-space-geodesic*
assumes *lambda C*−*quasi-isometry f*
shows *Gromov-converging-at-boundary* $(\lambda n.\ f\ (u\ n)) \longleftrightarrow$ *Gromov-converging-at-boundary* u
proof
assume *Gromov-converging-at-boundary u*
show *Gromov-converging-at-boundary* $(\lambda n.\ f\ (u\ n))$
proof (*rule Gromov-converging-at-boundaryI*[*of f* (*basepoint*)])
have $lambda \geq 1$ $C \geq 0$ using *quasi-isometry-onD*[*OF assms*(*1*)] by *auto*
define D where $D = 187 * lambda\hat{}2 * (C + deltaG(TYPE('a)) + deltaG(TYPE('b)))$
fix M::*real*
obtain $M2$::*real* where $M2$: $M = M2/lambda - D$
using ‹$lambda \geq 1$› by (*auto simp add*: *algebra-simps divide-simps*)
obtain N where N: $\bigwedge m\ n.\ m \geq N \Longrightarrow n \geq N \Longrightarrow$ *Gromov-product-at basepoint* $(u\ m)\ (u\ n) \geq M2$
using ‹*Gromov-converging-at-boundary u*› unfolding *Gromov-converging-at-boundary-def* by *blast*
have *Gromov-product-at* $(f\ basepoint)\ (f\ (u\ m))\ (f\ (u\ n)) \geq M$ if $m \geq N$ $n \geq N$ for $m\ n$

proof −
have $M \leq$ *Gromov-product-at basepoint* $(u\ m)\ (u\ n)/lambda - D$
unfolding *M2* **using** $N[OF\ that]$ ‹*lambda* $\geq$ *1*› **by** (*auto simp add*: *divide-simps*)
also have ... $\leq$ *Gromov-product-at* (*f basepoint*) $(f\ (u\ m))\ (f\ (u\ n))$
unfolding *D-def* **by** (*rule Gromov-product-at-quasi-isometry*[*OF assms*(*1*)])
finally show *?thesis* **by** *simp*
qed
then show $\exists N.\ \forall n \geq N.\ \forall m \geq N.\ M \leq$ *Gromov-product-at* (*f basepoint*) $(f\ (u\ m))\ (f\ (u\ n))$
unfolding *comp-def* **by** *auto*
qed
next
assume *Gromov-converging-at-boundary* $(\lambda n.\ f\ (u\ n))$
show *Gromov-converging-at-boundary u*
proof (*rule Gromov-converging-at-boundaryI*[*of basepoint*])
have $lambda \geq 1\ C \geq 0$ **using** *quasi-isometry-onD*[*OF assms*(*1*)] **by** *auto*
define *D* **where** $D = 187 * lambda\hat{}2 * (C + deltaG(TYPE('a)) + deltaG(TYPE('b)))$
fix *M*::*real*
define *M2* **where** $M2 = lambda * M + D$
have *M2*: $M = (M2 - D)/lambda$ **unfolding** *M2-def* **using** ‹*lambda* $\geq$ *1*› **by** (*auto simp add*: *algebra-simps divide-simps*)
obtain *N* **where** *N*: $\bigwedge m\ n.\ m \geq N \Longrightarrow n \geq N \Longrightarrow$ *Gromov-product-at* (*f basepoint*) $(f\ (u\ m))\ (f\ (u\ n)) \geq M2$
using ‹*Gromov-converging-at-boundary* $(\lambda n.\ f\ (u\ n))$› **unfolding** *Gromov-converging-at-boundary-def* **by** *blast*
have *Gromov-product-at basepoint* $(u\ m)\ (u\ n) \geq M$ **if** $m \geq N\ n \geq N$ **for** $m\ n$
proof −
have $M2 \leq$ *Gromov-product-at* (*f basepoint*) $(f\ (u\ m))\ (f\ (u\ n))$
using $N[OF\ that]$ **by** *auto*
also have ... $\leq lambda *$ *Gromov-product-at basepoint* $(u\ m)\ (u\ n) + D$
unfolding *D-def* **by** (*rule Gromov-product-at-quasi-isometry*[*OF assms*(*1*)])
finally show $M \leq$ *Gromov-product-at basepoint* $(u\ m)\ (u\ n)$
unfolding *M2* **using** ‹*lambda* $\geq$ *1*› **by** (*auto simp add*: *algebra-simps divide-simps*)
qed
then show $\exists N.\ \forall n \geq N.\ \forall m \geq N.$ *Gromov-product-at basepoint* $(u\ m)\ (u\ n) \geq M$
by *auto*
qed
qed

We define the extension to the completion of a function $f : X \to Y$ where X and Y are geodesic Gromov-hyperbolic spaces, as a function from $X \cup \partial X$ to $Y \cup \partial Y$, as follows. If x is in the space, we just use $f(x)$ (with the suitable coercions for the definition). Otherwise, we wish to define $f(x)$ as the limit of $f(u_n)$ for all sequences tending to x. For the definition, we use one such sequence chosen arbitrarily (this is the role of `rep_Gromov_completion x` below, it is indeed a sequence in the space tending to x), and we use the limit of $f(u_n)$ (if it exists, otherwise the framework will choose some point for us but it will make no sense whatsoever).

For quasi-isometries, we have indeed that $f(u_n)$ converges if u_n converges to a boundary point, by `Gromov_converging_at_infinity_quasi_isometry`, so this definition is meaningful. Moreover, continuity of the extension follows readily from this (modulo a suitable criterion for continuity based on sequences convergence, established in `continuous_at_extension_sequentially'`).

definition *Gromov-extension*::('*a*::*Gromov-hyperbolic-space* ⇒ '*b*::*Gromov-hyperbolic-space*) ⇒ ('*a Gromov-completion* ⇒ '*b Gromov-completion*)
 where *Gromov-extension f x* = (*if x* ∈ *Gromov-boundary then lim* (*to-Gromov-completion o f o* (*rep-Gromov-completion x*))
 else to-Gromov-completion (*f* (*from-Gromov-completion x*)))

lemma *Gromov-extension-inside-space* [*simp*]:
 Gromov-extension f (*to-Gromov-completion x*) = *to-Gromov-completion* (*f x*)
unfolding *Gromov-extension-def* **by** *auto*

lemma *Gromov-extension-id* [*simp*]:
 Gromov-extension (*id*::'*a*::*Gromov-hyperbolic-space* ⇒ '*a*) = *id*
 Gromov-extension (λ*x*::'*a*. *x*) = (λ*x*. *x*)
proof −
 have *Gromov-extension id x* = *id x* **for** *x*::'*a Gromov-completion*
 unfolding *Gromov-extension-def comp-def*
 using *limI rep-Gromov-completion-limit* **by** (*auto simp add*: *to-from-Gromov-completion*)
 then show *Gromov-extension* (*id*::'*a* ⇒ '*a*) = *id*
 by *auto*
 then show *Gromov-extension* (λ*x*::'*a*. *x*) = (λ*x*. *x*)
 unfolding *id-def* **by** *auto*
qed

The Gromov extension of a quasi-isometric map sends the boundary to the boundary.

lemma *Gromov-extension-quasi-isometry-boundary-to-boundary*:
 fixes *f*::'*a*::*Gromov-hyperbolic-space-geodesic* ⇒ '*b*::*Gromov-hyperbolic-space-geodesic*
 assumes *lambda C−quasi-isometry f*
 x ∈ *Gromov-boundary*
 shows (*Gromov-extension f*) *x* ∈ *Gromov-boundary*
proof −
 have ∗: *Gromov-converging-at-boundary* (λ*n*. *f* (*rep-Gromov-completion x n*))
 by (*simp add*: *Gromov-converging-at-infinity-quasi-isometry*[*OF assms*(*1*)] *Gromov-boundary-rep-converging assms*(*2*))
 show *?thesis*
 unfolding *Gromov-extension-def* **using** *assms*(*2*) **unfolding** *comp-def* **apply** *auto*
 by (*metis Gromov-converging-at-boundary-converges* ∗ *limI*)
qed

If the original function is continuous somewhere inside the space, then its Gromov extension is continuous at the corresponding point inside the completion. This is clear as the original space is open in the Gromov completion, but the proof requires to go back and forth between one space and the other.

lemma *Gromov-extension-continuous-inside*:
  fixes f::$'a$::*Gromov-hyperbolic-space* $\Rightarrow$ $'b$::*Gromov-hyperbolic-space*
  assumes *continuous* (*at x within S*) *f*
  shows *continuous* (*at* (*to-Gromov-completion x*) *within* (*to-Gromov-completion‘S*)) (*Gromov-extension f*)
proof −
  have ∗: *continuous* (*at* (*to-Gromov-completion x*) *within* (*to-Gromov-completion‘S*)) (*to-Gromov-completion o f o from-Gromov-completion*)
    apply (*intro continuous-within-compose*, *auto*)
    using *from-Gromov-completion-continuous*(*3*) *continuous-at-imp-continuous-within*
apply *blast*
    using *assms* **apply** (*simp add*: *continuous-within-topological*)
    using *continuous-at-imp-continuous-within continuous-on-eq-continuous-within to-Gromov-completion-continuous* **by** *blast*
  have (*to-Gromov-completion o f o from-Gromov-completion*) *y* = *Gromov-extension f y*
    if *y* ∈ *range to-Gromov-completion* **for** *y*
    unfolding *comp-def* **using** *that* **by** *auto*
  moreover have *eventually* (λ*y*. *y* ∈ *range to-Gromov-completion*) (*at* (*to-Gromov-completion x*) *within* (*to-Gromov-completion‘S*))
    using *to-Gromov-completion-range-open eventually-at-topological* **by** *blast*
  ultimately have ∗∗: *eventually* (λ*y*. (*to-Gromov-completion o f o from-Gromov-completion*) *y* = *Gromov-extension f y*)
                    (*at* (*to-Gromov-completion x*) *within* (*to-Gromov-completion‘S*))
    by (*rule eventually-mono*[*rotated*])
  show *?thesis*
    by (*rule continuous-within-cong*[*OF* ∗ ∗∗], *auto*)
qed

The extension to the boundary of a quasi-isometry is continuous. This is a nontrivial statement, but it follows readily from the fact we have already proved that sequences converging at the boundary are mapped to sequences converging to the boundary. The proof is expressed using a convenient continuity criterion for which we only need to control what happens for sequences inside the space.

proposition *Gromov-extension-continuous*:
  fixes f::$'a$::*Gromov-hyperbolic-space-geodesic* $\Rightarrow$ $'b$::*Gromov-hyperbolic-space-geodesic*
  assumes *lambda C*−*quasi-isometry f*
          x ∈ *Gromov-boundary*
  shows *continuous* (*at x*) (*Gromov-extension f*)
proof −
  have *continuous* (*at x within* (*range to-Gromov-completion* ∪ *Gromov-boundary*)) (*Gromov-extension f*)
  proof (*rule continuous-at-extension-sequentially'*[*OF* ‹*x* ∈ *Gromov-boundary*›])

fix *b*::*'a Gromov-completion* **assume** *b* ∈ *Gromov-boundary*
show ∃ *u*. (∀ *n*. *u n* ∈ *range to-Gromov-completion*) ∧ *u* ⟶ *b* ∧ (λ*n*. *Gromov-extension f* (*u n*)) ⟶ *Gromov-extension f b*
apply (*rule exI*[*of - to-Gromov-completion o* (*rep-Gromov-completion b*)], *auto simp add*: *comp-def*)
unfolding *Gromov-completion-converge-to-boundary*[*OF* ‹*b* ∈ *Gromov-boundary*›]
using *Quotient3-abs-rep*[*OF Quotient3-Gromov-completion*] *Quotient3-rep-reflp*[*OF Quotient3-Gromov-completion*] **apply** *auto*[*1*]
unfolding *Gromov-extension-def* **using** ‹*b* ∈ *Gromov-boundary*› **unfolding** *comp-def*
by (*auto simp add*: *convergent-LIMSEQ-iff*[*symmetric*] *Gromov-boundary-rep-converging Gromov-converging-at-infinity-quasi-isometry*[*OF assms*(*1*)]
intro!: *Gromov-converging-at-boundary-converges'*)
next
fix *u* **and** *b*::*'a Gromov-completion*
assume *u*: ∀ *n*. *u n* ∈ *range to-Gromov-completion b* ∈ *Gromov-boundary u* ⟶ *b*
define *v* **where** *v* = (λ*n*. *from-Gromov-completion* (*u n*))
have *v*: *u n* = *to-Gromov-completion* (*v n*) **for** *n*
using *u*(*1*) **unfolding** *v-def* **by** (*simp add*: *f-inv-into-f from-Gromov-completion-def*)

show *convergent* (λ*n*. *Gromov-extension f* (*u n*))
using *u* **unfolding** *v*
apply (*auto intro*!: *Gromov-converging-at-boundary-converges' simp add*: *Gromov-converging-at-infinity-quasi-isometry*[*OF assms*(*1*)])
using *Gromov-boundary-abs-converging Gromov-completion-converge-to-boundary* **by** *blast*
qed
then show *?thesis* **by** (*simp add*: *Gromov-boundary-def*)
qed

Combining the two previous statements on continuity inside the space and continuity at the boundary, we deduce that a continuous quasi-isometry extends to a continuous map everywhere.

proposition *Gromov-extension-continuous-everywhere*:
fixes *f*::*'a*::*Gromov-hyperbolic-space-geodesic* ⇒ *'b*::*Gromov-hyperbolic-space-geodesic*
assumes *lambda C*−*quasi-isometry f*
continuous-on UNIV f
shows *continuous-on UNIV* (*Gromov-extension f*)
using *Gromov-extension-continuous-inside Gromov-extension-continuous*[*OF assms*(*1*)]
by (*metis UNIV-I assms*(*2*) *continuous-on-eq-continuous-within continuous-within-open not-in-Gromov-boundary rangeI to-Gromov-completion-range-open*)

The extension to the boundary is functorial on the category of quasi-isometries, i.e., the composition of extensions is the extension of the composition. This is clear inside the space, and it follows from the continuity at boundary points.

lemma *Gromov-extension-composition*:

fixes f::'a::*Gromov-hyperbolic-space-geodesic* $\Rightarrow$ 'b::*Gromov-hyperbolic-space-geodesic*
and g::'b::*Gromov-hyperbolic-space-geodesic* $\Rightarrow$ 'c::*Gromov-hyperbolic-space-geodesic*
assumes *lambda C−quasi-isometry f*
mu D−quasi-isometry g
shows *Gromov-extension (g o f) = Gromov-extension g o Gromov-extension f*
proof −
have *In*: *Gromov-extension (g o f) x = (Gromov-extension g o Gromov-extension f) x* **if** *H*: *x* ∈ *range to-Gromov-completion* **for** *x*
proof −
obtain *y* **where** ∗: *x = to-Gromov-completion y*
using *H* **by** *auto*
show *?thesis*
unfolding ∗ *comp-def* **by** *auto*
qed
moreover have *Gromov-extension (g o f) x = (Gromov-extension g o Gromov-extension f) x* **if** *H*: *x* ∈ *Gromov-boundary* **for** *x*
proof −
obtain *u* **where** *u*: $\bigwedge$*n. u n* ∈ *range to-Gromov-completion u* ⟶ *x*
using *closure-sequential to-Gromov-completion-range-dense* **by** *blast*
have (λ*n. Gromov-extension (g o f) (u n))* ⟶ *Gromov-extension (g o f) x*
using *continuous-within-tendsto-compose*[*OF Gromov-extension-continuous*[*OF quasi-isometry-on-compose*[*OF assms(1) assms(2), simplified*] *H*] - *u(2)*] **by** *simp*
then have *A*: (λ*n. (Gromov-extension g) ((Gromov-extension f) (u n)))* ⟶ *Gromov-extension (g o f) x*
unfolding *In*[*OF u(1)*] **unfolding** *comp-def* **by** *auto*

have ∗: (λ*n. (Gromov-extension f) (u n))* ⟶ *(Gromov-extension f) x*
using *continuous-within-tendsto-compose*[*OF Gromov-extension-continuous*[*OF assms(1) H*] - *u(2)*] **by** *simp*
have (λ*n. (Gromov-extension g) ((Gromov-extension f) (u n)))* ⟶ *Gromov-extension g ((Gromov-extension f) x)*
using *continuous-within-tendsto-compose*[*OF Gromov-extension-continuous*[*OF assms(2)*] - ∗]
H Gromov-extension-quasi-isometry-boundary-to-boundary assms(1) **by** *auto*
then show *?thesis* **using** *LIMSEQ-unique A comp-def* **by** *auto*
qed
ultimately have *Gromov-extension (g o f) x = (Gromov-extension g o Gromov-extension f) x* **for** *x*
using *not-in-Gromov-boundary* **by** *force*
then show *?thesis* **by** *auto*
qed

Now, we turn to the same kind of statement, but for homeomorphisms. We claim that if a quasi-isometry f is a homeomorphism on a subset X of the space, then its extension is a homeomorphism on X union the boundary of the space. For the proof, we have to show that a sequence u_n tends to a point x if and only if $f(u_n)$ tends to $f(x)$. We separate the cases x in the boundary, and x inside the space. For x in the boundary, we use a homeomorphism criterion expressed solely in terms of sequences converging to the boundary, for which we already know everything. For x in the space, the proof is straightforward, but tedious. We argue that eventually u_n is in the space for the direct implication, or $f(u_n)$ is in the space for the second implication, and then we use that f is a homeomorphism inside the space to conclude.

lemma *Gromov-extension-homeomorphism*:
fixes *f*::*'a*::*Gromov-hyperbolic-space-geodesic* ⇒ *'b*::*Gromov-hyperbolic-space-geodesic*
assumes *lambda C−quasi-isometry f*
homeomorphism-on X f
shows *homeomorphism-on* (*to-Gromov-completion'X* ∪ *Gromov-boundary*) (*Gromov-extension f*)
proof (*rule homeomorphism-on-sequentially*)
fix *x u* **assume** *H0*: *x* ∈ *to-Gromov-completion* ' *X* ∪ *Gromov-boundary*
∀ *n*::*nat*. *u n* ∈ *to-Gromov-completion* ' *X* ∪ *Gromov-boundary*
then consider *x* ∈ *Gromov-boundary* | *x* ∈ *to-Gromov-completion'X* **by** *auto*
then show *u* ⟶ *x* = (*λn*. *Gromov-extension f* (*u n*)) ⟶ *Gromov-extension f x*
proof (*cases*)

First, consider the case where the limit point x is in the boundary. We use a good criterion expressing everything in terms of sequences inside the space.

case *1*
show *?thesis*
proof (*rule homeomorphism-on-extension-sequentially-precise*[*of range to-Gromov-completion Gromov-boundary*])
show *x* ∈ *Gromov-boundary* **by** *fact*
fix *n*::*nat* **show** *u n* ∈ *range to-Gromov-completion* ∪ *Gromov-boundary*
unfolding *Gromov-boundary-def* **by** *auto*
next
fix *u* **and** *b*::*'a Gromov-completion*
assume *u*: ∀ *n*. *u n* ∈ *range to-Gromov-completion* *b* ∈ *Gromov-boundary* *u* ⟶ *b*
define *v* **where** *v* = (*λn*. *from-Gromov-completion* (*u n*))
have *v*: *u n* = *to-Gromov-completion* (*v n*) **for** *n*
using *u*(*1*) **unfolding** *v-def* **by** (*simp add*: *f-inv-into-f from-Gromov-completion-def*)
show *convergent* (*λn*. *Gromov-extension f* (*u n*))
using *u* **unfolding** *v* **apply** *auto*
apply (*rule Gromov-converging-at-boundary-converges'*)
by (*auto simp add*: *Gromov-converging-at-infinity-quasi-isometry*[*OF assms*(*1*)] *lim-imp-Gromov-converging-at-boundary*)
next
fix *u c*
assume *u*: ∀ *n*. *u n* ∈ *range to-Gromov-completion* *c* ∈ *Gromov-extension f* ' *Gromov-boundary* (*λn*. *Gromov-extension f* (*u n*)) ⟶ *c*
then have *c* ∈ *Gromov-boundary* **using** *Gromov-extension-quasi-isometry-boundary-to-boundary*[*OF assms*(*1*)] **by** *auto*
define *v* **where** *v* = (*λn*. *from-Gromov-completion* (*u n*))
have *v*: *u n* = *to-Gromov-completion* (*v n*) **for** *n*

using *u*(*1*) **unfolding** *v-def* **by** (*simp add*: *f-inv-into-f from-Gromov-completion-def*)
have *Gromov-converging-at-boundary* (*λn. f* (*v n*))
apply (*rule lim-imp-Gromov-converging-at-boundary*[*OF - ‹c ∈ Gromov-boundary›*])
using *u*(*3*) **unfolding** *v* **by** *auto*
then show *convergent u*
using *u* **unfolding** *v*
by (*auto intro*!: *Gromov-converging-at-boundary-converges′ simp add*: *Gromov-converging-at-infinity-quasi-isometry*[*OF assms*(*1*), *symmetric*])
next
fix *b*::′*a Gromov-completion* **assume** *b* ∈ *Gromov-boundary*
show ∃ *u*. (∀ *n*. *u n* ∈ *range to-Gromov-completion*) ∧ *u* ⟶ *b* ∧ (*λn*. *Gromov-extension f* (*u n*)) ⟶ *Gromov-extension f b*
apply (*rule exI*[*of - to-Gromov-completion o* (*rep-Gromov-completion b*)], *auto simp add*: *comp-def*)
unfolding *Gromov-completion-converge-to-boundary*[*OF ‹b ∈ Gromov-boundary›*]
using *Quotient3-abs-rep*[*OF Quotient3-Gromov-completion*] *Quotient3-rep-reflp*[*OF Quotient3-Gromov-completion*] **apply** *auto*[*1*]
unfolding *Gromov-extension-def* **using** *‹b ∈ Gromov-boundary›* **unfolding** *comp-def*
by (*auto simp add*: *convergent-LIMSEQ-iff*[*symmetric*] *Gromov-boundary-rep-converging Gromov-converging-at-infinity-quasi-isometry*[*OF assms*(*1*)]
intro!: *Gromov-converging-at-boundary-converges′*)
qed
next

Next, consider the case where x is inside the space. Then we show everything by going back and forth between the original space and its copy in the completion, and arguing that f is a homeomorphism on the original space.

case *2*
then have *fx*: *Gromov-extension f x* ∈ *range to-Gromov-completion*
using *Gromov-extension-inside-space* **by** *blast*
have *x*: *x* ∈ *range to-Gromov-completion*
using *2* **by** *blast*
show *?thesis*
proof
assume *H*: (*λn. Gromov-extension f* (*u n*)) ⟶ *Gromov-extension f x*
then have *fu-in*: *eventually* (*λn. Gromov-extension f* (*u n*) ∈ *range to-Gromov-completion*) *sequentially*
using *fx to-Gromov-completion-range-open H topological-tendstoD* **by** *fastforce*
have *u-in*: *eventually* (*λn. u n* ∈ *range to-Gromov-completion*) *sequentially*
using *Gromov-extension-quasi-isometry-boundary-to-boundary*[*OF assms*(*1*)] *eventually-mono*[*OF fu-in*]
by (*metis DiffE DiffI Gromov-boundary-def iso-tuple-UNIV-I*)

have *B*: *from-Gromov-completion* (*Gromov-extension f y*) = *f* (*from-Gromov-completion y*) **if** *Gromov-extension f y* ∈ *range to-Gromov-completion* **for** *y*
by (*metis Gromov-extension-quasi-isometry-boundary-to-boundary Gromov-extension-def assms*(*1*) *from-to-Gromov-completion not-in-Gromov-boundary′*

rangeE that)
have ($\lambda n.$ *from-Gromov-completion* (*Gromov-extension f* (*u n*))) $\longrightarrow$ *from-Gromov-completion* (*Gromov-extension f x*)
by (*rule continuous-on-tendsto-compose*[*OF from-Gromov-completion-continuous*(*2*) *H fx fu-in*])
then have *C*: ($\lambda n.$ *f* (*from-Gromov-completion* (*u n*))) $\longrightarrow$ *f* (*from-Gromov-completion x*)
unfolding *B*[*OF fx*, *symmetric*]
by (*force intro*: *Lim-transform-eventually eventually-mono*[*OF fu-in B*])
have ($\lambda n.$ *from-Gromov-completion* (*u n*)) $\longrightarrow$ *from-Gromov-completion x*
apply (*rule iffD2*[*OF homeomorphism-on-compose*[*OF assms*(*2*)] *C*])
using *2* **apply** *auto*
by (*metis* (*no-types*, *lifting*) *eventually-mono*[*OF u-in*] *H0*(*2*) *Un-iff f-inv-into-f from-to-Gromov-completion inv-into-into not-in-Gromov-boundary′*)
then have *L*: ($\lambda n.$ *to-Gromov-completion* (*from-Gromov-completion* (*u n*))) $\longrightarrow$ *to-Gromov-completion* (*from-Gromov-completion x*)
using *continuous-on-tendsto-compose*[*OF to-Gromov-completion-continuous*] **by** *auto*

have $**$: *to-Gromov-completion* (*from-Gromov-completion y*) = *y* **if** $y \in$ *range to-Gromov-completion* **for** *y*::′*a Gromov-completion*
using *Gromov-extension-quasi-isometry-boundary-to-boundary assms*(*1*) *that to-from-Gromov-completion* **by** *fastforce*
then have *eventually* ($\lambda n.$ *to-Gromov-completion* (*from-Gromov-completion* (*u n*)) = *u n*) *sequentially*
using *u-in eventually-mono* **by** *force*
then have *u* $\longrightarrow$ *to-Gromov-completion* (*from-Gromov-completion x*)
by (*rule Lim-transform-eventually*[*OF L*])
then show *u* $\longrightarrow$ *x*
using $**$ **by** (*simp add*: *x*)
next
assume *u* $\longrightarrow$ *x*
then have *u-in*: *eventually* ($\lambda n.$ *u n* $\in$ *range to-Gromov-completion*) *sequentially*
using *x to-Gromov-completion-range-open topological-tendstoD* **by** *fastforce*
define *y* **where** *y* = *from-Gromov-completion x*
have $y \in X$ **unfolding** *y-def* **using** *2* **by** *auto*
then have $*$: *continuous* (*at y within X*) *f*
using *homeomorphism-on-continuous*[*OF assms*(*2*)] *continuous-on-eq-continuous-within* **by** *blast*
have $**$: *continuous* (*at x within to-Gromov-completion‘X*) (*Gromov-extension f*)
using *Gromov-extension-continuous-inside*[*OF* $*$] *y-def 2* **by** *auto*

show ($\lambda n.$ *Gromov-extension f* (*u n*)) $\longrightarrow$ *Gromov-extension f x*
apply (*rule continuous-within-tendsto-compose*[*OF* $**$ - ‹*u* $\longrightarrow$ *x*›])
using *u-in H0*(*2*) **by** (*metis* (*mono-tags*, *lifting*) *UnE eventually-mono f-inv-into-f not-in-Gromov-boundary′*)
qed

qed
qed

In particular, it follows that the extension to the boundary of a quasi-isometry is always a homeomorphism, regardless of the continuity properties of the original map.

proposition *Gromov-extension-boundary-homeomorphism*:
fixes f::$'a$::*Gromov-hyperbolic-space-geodesic* $\Rightarrow$ $'b$::*Gromov-hyperbolic-space-geodesic*
assumes *lambda C−quasi-isometry f*
shows *homeomorphism-on Gromov-boundary (Gromov-extension f)*
using *Gromov-extension-homeomorphism*[*OF assms, of* {}] **by** *auto*

When the quasi-isometric embedding is a quasi-isometric isomorphism, i.e., it is onto up to a bounded distance C, then its Gromov extension is onto on the boundary. Indeed, a point in the image boundary is a limit of a sequence inside the space. Perturbing by a bounded distance (which does not change the asymptotic behavior), it is the limit of a sequence inside the image of f. Then the preimage under f of this sequence does converge, and its limit is sent by the extension on the original point, proving the surjectivity.

lemma *Gromov-extension-onto*:
fixes f::$'a$::*Gromov-hyperbolic-space-geodesic* $\Rightarrow$ $'b$::*Gromov-hyperbolic-space-geodesic*
assumes *lambda C−quasi-isometry-between UNIV UNIV f*
y ∈ *Gromov-boundary*
shows ∃ *x* ∈ *Gromov-boundary. Gromov-extension f x* = *y*
proof −
define *u* **where** *u* = *rep-Gromov-completion y*
have ∗: (λ*n. to-Gromov-completion* (*u n*)) ⟶ *y*
unfolding *u-def* **using** *rep-Gromov-completion-limit* **by** *fastforce*
have ∃ *v.* ∀ *n. dist* (*f* (*v n*)) (*u n*) ≤ *C*
apply (*intro choice*) **using** *quasi-isometry-betweenD*(*3*)[*OF assms*(*1*)] **by** *auto*
then obtain *v* **where** *v*: ⋀*n. dist* (*f* (*v n*)) (*u n*) ≤ *C* **by** *auto*
have ∗: (λ*n. to-Gromov-completion* (*f* (*v n*))) ⟶ *y*
apply (*rule Gromov-converging-at-boundary-bounded-perturbation*[*OF* ∗ ‹*y* ∈ *Gromov-boundary*›])
using *v* **by** (*simp add*: *dist-commute*)
then have *Gromov-converging-at-boundary* (λ*n. f* (*v n*))
using *assms*(*2*) *lim-imp-Gromov-converging-at-boundary* **by** *force*
then have *Gromov-converging-at-boundary v*
using *Gromov-converging-at-infinity-quasi-isometry*[*OF quasi-isometry-betweenD*(*1*)[*OF assms*(*1*)]] **by** *auto*
then obtain *x* **where** *x* ∈ *Gromov-boundary* (λ*n. to-Gromov-completion* (*v n*)) ⟶ *x*
using *Gromov-converging-at-boundary-converges* **by** *blast*
then have (λ*n.* (*Gromov-extension f*) (*to-Gromov-completion* (*v n*))) ⟶ *Gromov-extension f x*
using *isCont-tendsto-compose*[*OF Gromov-extension-continuous*[*OF quasi-isometry-betweenD*(*1*)[*OF assms*(*1*)] ‹*x* ∈ *Gromov-boundary*›]] **by** *fastforce*
then have *y* = *Gromov-extension f x*

using $*$ *LIMSEQ-unique* **by** *auto*
then show *?thesis* **using** ‹$x \in$ *Gromov-boundary*› **by** *auto*
qed

lemma *Gromov-extension-onto'*:
fixes *f*::*'a*::*Gromov-hyperbolic-space-geodesic* $\Rightarrow$ *'b*::*Gromov-hyperbolic-space-geodesic*
assumes *lambda C−quasi-isometry-between UNIV UNIV f*
shows (*Gromov-extension f*)‘*Gromov-boundary* = *Gromov-boundary*
using *Gromov-extension-onto*[*OF assms*] *Gromov-extension-quasi-isometry-boundary-to-boundary*[*OF quasi-isometry-betweenD*(*1*)[*OF assms*]] **by** *auto*

Finally, we obtain that a quasi-isometry between two Gromov hyperbolic spaces induces a homeomorphism of their boundaries.

theorem *Gromov-boundaries-homeomorphic*:
fixes *f*::*'a*::*Gromov-hyperbolic-space-geodesic* $\Rightarrow$ *'b*::*Gromov-hyperbolic-space-geodesic*
assumes *lambda C−quasi-isometry-between UNIV UNIV f*
shows (*Gromov-boundary*::*'a Gromov-completion set*) *homeomorphic* (*Gromov-boundary*::*'b Gromov-completion set*)
using *Gromov-extension-boundary-homeomorphism*[*OF quasi-isometry-betweenD*(*1*)[*OF assms*]] *Gromov-extension-onto'*[*OF assms*]
unfolding *homeomorphic-def homeomorphism-on-def* **by** *auto*

17 Extensions of isometries to the boundary

The results of the previous section can be improved for isometries, as there is no need for geodesicity any more. We follow the same proofs as in the previous section

An isometry preserves the Gromov product.

lemma *Gromov-product-isometry*:
assumes *isometry-on UNIV f*
shows *Gromov-product-at* (*f x*) (*f y*) (*f z*) = *Gromov-product-at x y z*
unfolding *Gromov-product-at-def* **by** (*simp add*: *isometry-onD*[*OF assms*])

An isometry preserves convergence at infinity.

lemma *Gromov-converging-at-infinity-isometry*:
fixes *f*::*'a*::*Gromov-hyperbolic-space* $\Rightarrow$ *'b*::*Gromov-hyperbolic-space*
assumes *isometry-on UNIV f*
shows *Gromov-converging-at-boundary* ($\lambda n.$ *f* (*u n*)) $\longleftrightarrow$ *Gromov-converging-at-boundary u*
proof
assume $*$: *Gromov-converging-at-boundary u*
show *Gromov-converging-at-boundary* ($\lambda n.$ *f* (*u n*))
apply (*rule Gromov-converging-at-boundaryI*[*of f* (*basepoint*)])
using $*$ **unfolding** *Gromov-converging-at-boundary-def Gromov-product-isometry*[*OF assms*] **by** *auto*
next

assume ∗: *Gromov-converging-at-boundary* ($\lambda n.$ *f* (*u n*))
have ∗∗: $\exists N. \forall n \geq N. \forall m \geq N. M \leq$ *Gromov-product-at* (*f basepoint*) (*f* (*u m*)) (*f* (*u n*)) **for** *M*
using ∗ **unfolding** *Gromov-converging-at-boundary-def* **by** *auto*
show *Gromov-converging-at-boundary u*
apply (*rule Gromov-converging-at-boundaryI*[*of basepoint*])
using ∗∗ **unfolding** *Gromov-converging-at-boundary-def Gromov-product-isometry*[*OF assms*] **by** *auto*
qed

The Gromov extension of an isometry sends the boundary to the boundary.

lemma *Gromov-extension-isometry-boundary-to-boundary*:
fixes *f*::*'a*::*Gromov-hyperbolic-space* ⇒ *'b*::*Gromov-hyperbolic-space*
assumes *isometry-on UNIV f*
$x \in$ *Gromov-boundary*
shows (*Gromov-extension f*) $x \in$ *Gromov-boundary*
proof −
have ∗: *Gromov-converging-at-boundary* ($\lambda n.$ *f* (*rep-Gromov-completion x n*))
by (*simp add*: *Gromov-converging-at-infinity-isometry*[*OF assms*(*1*)] *Gromov-boundary-rep-converging assms*(*2*))
show *?thesis*
unfolding *Gromov-extension-def* **using** *assms*(*2*) **unfolding** *comp-def* **apply** *auto*
by (*metis Gromov-converging-at-boundary-converges* ∗ *limI*)
qed

The Gromov extension of an isometry is a homeomorphism. (We copy the proof for quasi-isometries, with some simplifications.)

lemma *Gromov-extension-isometry-homeomorphism*:
fixes *f*::*'a*::*Gromov-hyperbolic-space* ⇒ *'b*::*Gromov-hyperbolic-space*
assumes *isometry-on UNIV f*
shows *homeomorphism-on UNIV* (*Gromov-extension f*)
proof (*rule homeomorphism-on-sequentially*)
fix *x u*
show $u \longrightarrow x = (\lambda n.$ *Gromov-extension f* (*u n*)) $\longrightarrow$ *Gromov-extension f x*
proof (*cases x*)

First, consider the case where the limit point x is in the boundary. We use a good criterion expressing everything in terms of sequences inside the space.

case *boundary*
show *?thesis*
proof (*rule homeomorphism-on-extension-sequentially-precise*[*of range to-Gromov-completion Gromov-boundary*])
show $x \in$ *Gromov-boundary* **by** *fact*
fix *n*::*nat* **show** *u n* $\in$ *range to-Gromov-completion* $\cup$ *Gromov-boundary*
unfolding *Gromov-boundary-def* **by** *auto*
next

fix *u* **and** *b*::′*a Gromov-completion*
assume *u*: ∀ *n*. *u n* ∈ *range to-Gromov-completion b* ∈ *Gromov-boundary u* ⟶ *b*
define *v* **where** *v* = (λ*n*. *from-Gromov-completion* (*u n*))
have *v*: *u n* = *to-Gromov-completion* (*v n*) **for** *n*
using *u*(*1*) **unfolding** *v-def* **by** (*simp add*: *f-inv-into-f from-Gromov-completion-def*)
show *convergent* (λ*n*. *Gromov-extension f* (*u n*))
using *u* **unfolding** *v* **apply** *auto*
apply (*rule Gromov-converging-at-boundary-converges*′)
by (*auto simp add*: *Gromov-converging-at-infinity-isometry*[*OF assms*(*1*)] *lim-imp-Gromov-converging-at-boundary*)
next
fix *u c*
assume *u*: ∀ *n*. *u n* ∈ *range to-Gromov-completion c* ∈ *Gromov-extension f* ‘ *Gromov-boundary* (λ*n*. *Gromov-extension f* (*u n*)) ⟶ *c*
then have *c* ∈ *Gromov-boundary* **using** *Gromov-extension-isometry-boundary-to-boundary*[*OF assms*(*1*)] **by** *auto*
define *v* **where** *v* = (λ*n*. *from-Gromov-completion* (*u n*))
have *v*: *u n* = *to-Gromov-completion* (*v n*) **for** *n*
using *u*(*1*) **unfolding** *v-def* **by** (*simp add*: *f-inv-into-f from-Gromov-completion-def*)
have *Gromov-converging-at-boundary* (λ*n*. *f* (*v n*))
apply (*rule lim-imp-Gromov-converging-at-boundary*[*OF* - ‹*c* ∈ *Gromov-boundary*›])
using *u*(*3*) **unfolding** *v* **by** *auto*
then show *convergent u*
using *u* **unfolding** *v*
by (*auto intro*!: *Gromov-converging-at-boundary-converges*′ *simp add*: *Gromov-converging-at-infinity-isometry*[*OF assms*(*1*), *symmetric*])
next
fix *b*::′*a Gromov-completion* **assume** *b* ∈ *Gromov-boundary*
show ∃ *u*. (∀ *n*. *u n* ∈ *range to-Gromov-completion*) ∧ *u* ⟶ *b* ∧ (λ*n*. *Gromov-extension f* (*u n*)) ⟶ *Gromov-extension f b*
apply (*rule exI*[*of* - *to-Gromov-completion o* (*rep-Gromov-completion b*)], *auto simp add*: *comp-def*)
unfolding *Gromov-completion-converge-to-boundary*[*OF* ‹*b* ∈ *Gromov-boundary*›]
using *Quotient3-abs-rep*[*OF Quotient3-Gromov-completion*] *Quotient3-rep-reflp*[*OF Quotient3-Gromov-completion*] **apply** *auto*[*1*]
unfolding *Gromov-extension-def* **using** ‹*b* ∈ *Gromov-boundary*› **unfolding** *comp-def*
by (*auto simp add*: *convergent-LIMSEQ-iff*[*symmetric*] *Gromov-boundary-rep-converging Gromov-converging-at-infinity-isometry*[*OF assms*(*1*)]
intro!: *Gromov-converging-at-boundary-converges*′)
qed
next

Next, consider the case where x is inside the space. Then we show everything by going back and forth between the original space and its copy in the completion, and arguing that f is a homeomorphism on the original space.

case (*to-Gromov-completion xin*)
then have *fx*: *Gromov-extension f x* ∈ *range to-Gromov-completion*

using *Gromov-extension-inside-space* **by** *blast*
have *x*: $x \in$ *range to-Gromov-completion*
using *to-Gromov-completion* **by** *blast*
show *?thesis*
proof
assume *H*: (λn. *Gromov-extension f* (*u n*)) $\longrightarrow$ *Gromov-extension f x*
then have *fu-in*: *eventually* (λn. *Gromov-extension f* (*u n*) $\in$ *range to-Gromov-completion*) *sequentially*
using *fx to-Gromov-completion-range-open H topological-tendstoD* **by** *fast-force*
have *u-in*: *eventually* (λn. *u n* $\in$ *range to-Gromov-completion*) *sequentially*
using *Gromov-extension-isometry-boundary-to-boundary*[*OF assms*(*1*)] *eventually-mono*[*OF fu-in*]
by (*metis DiffE DiffI Gromov-boundary-def iso-tuple-UNIV-I*)

have *B*: *from-Gromov-completion* (*Gromov-extension f y*) = *f* (*from-Gromov-completion y*) **if** *Gromov-extension f y* $\in$ *range to-Gromov-completion* **for** *y*
by (*metis Gromov-extension-isometry-boundary-to-boundary Gromov-extension-def assms*(*1*) *from-to-Gromov-completion not-in-Gromov-boundary′ rangeE that*)
have (λn. *from-Gromov-completion* (*Gromov-extension f* (*u n*))) $\longrightarrow$ *from-Gromov-completion* (*Gromov-extension f x*)
by (*rule continuous-on-tendsto-compose*[*OF from-Gromov-completion-continuous*(*2*) *H fx fu-in*])
then have *C*: (λn. *f* (*from-Gromov-completion* (*u n*))) $\longrightarrow$ *f* (*from-Gromov-completion x*)
unfolding *B*[*OF fx*, *symmetric*]
by (*force intro*: *Lim-transform-eventually eventually-mono*[*OF fu-in B*])
have (λn. *from-Gromov-completion* (*u n*)) $\longrightarrow$ *from-Gromov-completion x*
apply (*rule iffD2*[*OF homeomorphism-on-compose*[*OF isometry-on-homeomorphism*(*2*)[*OF assms*]] *C*])
using *to-Gromov-completion* **by** *auto*
then have *L*: (λn. *to-Gromov-completion* (*from-Gromov-completion* (*u n*))) $\longrightarrow$ *to-Gromov-completion* (*from-Gromov-completion x*)
using *continuous-on-tendsto-compose*[*OF to-Gromov-completion-continuous*] **by** *auto*

have $**$: *to-Gromov-completion* (*from-Gromov-completion y*) = *y* **if** *y* $\in$ *range to-Gromov-completion* **for** *y*::′*a Gromov-completion*
using *Gromov-extension-isometry-boundary-to-boundary assms*(*1*) *that to-from-Gromov-completion* **by** *fastforce*
then have *eventually* (λn. *to-Gromov-completion* (*from-Gromov-completion* (*u n*)) = *u n*) *sequentially*
using *u-in eventually-mono* **by** *force*
then have *u* $\longrightarrow$ *to-Gromov-completion* (*from-Gromov-completion x*)
by (*rule Lim-transform-eventually*[*OF L*])
then show *u* $\longrightarrow$ *x*
using $**$ **by** (*simp add*: *x*)
next
assume *u* $\longrightarrow$ *x*

then have *u-in*: *eventually* ($\lambda n.$ *u n* $\in$ *range to-Gromov-completion*) *sequentially*
using *x to-Gromov-completion-range-open topological-tendstoD* **by** *fastforce*
define *y* **where** *y* = *from-Gromov-completion x*
then have $*$: *continuous* (*at y*) *f*
using *homeomorphism-on-continuous*[*OF isometry-on-homeomorphism*(*2*)[*OF assms*]] *continuous-on-eq-continuous-within* **by** *blast*
have $**$: *continuous* (*at x within to-Gromov-completion‘UNIV*) (*Gromov-extension f*)
using *Gromov-extension-continuous-inside*[*OF* $*$] *y-def to-Gromov-completion* **by** *auto*

show ($\lambda n.$ *Gromov-extension f* (*u n*)) $\longrightarrow$ *Gromov-extension f x*
apply (*rule continuous-within-tendsto-compose*[*OF* $**$ - ‹*u* $\longrightarrow$ *x*›])
using *u-in* **by** *auto*
qed
qed
qed

The composition of the Gromov extension of two isometries is the Gromov extension of the composition.

lemma *Gromov-extension-isometry-on-composition*:
assumes *isometry-on UNIV f*
isometry-on UNIV g
shows *Gromov-extension* (*g o f*) = *Gromov-extension g o Gromov-extension f*
proof −
have *In*: *Gromov-extension* (*g o f*) *x* = (*Gromov-extension g o Gromov-extension f*) *x* **if** *H*: *x* $\in$ *range to-Gromov-completion* **for** *x*
proof −
obtain *y* **where** $*$: *x* = *to-Gromov-completion y*
using *H* **by** *auto*
show *?thesis*
unfolding $*$ *comp-def* **by** *auto*
qed
moreover have *Gromov-extension* (*g o f*) *x* = (*Gromov-extension g o Gromov-extension f*) *x* **if** *H*: *x* $\in$ *Gromov-boundary* **for** *x*
proof −
obtain *u* **where** *u*: $\bigwedge n.$ *u n* $\in$ *range to-Gromov-completion u* $\longrightarrow$ *x*
using *closure-sequential to-Gromov-completion-range-dense* **by** *blast*
have ($\lambda n.$ *Gromov-extension* (*g o f*) (*u n*)) $\longrightarrow$ *Gromov-extension* (*g o f*) *x*
apply (*rule continuous-within-tendsto-compose*[*OF* - - *u*(*2*), *of UNIV*])
using *homeomorphism-on-continuous*[*OF Gromov-extension-isometry-homeomorphism*[*OF isometry-on-compose*[*OF assms*(*1*) *isometry-on-subset*[*OF assms*(*2*)]]]] **unfolding** *comp-def*
by (*auto simp add*: *continuous-on-eq-continuous-within*)
then have *A*: ($\lambda n.$ (*Gromov-extension g*) ((*Gromov-extension f*) (*u n*))) $\longrightarrow$ *Gromov-extension* (*g o f*) *x*
unfolding *In*[*OF u*(*1*)] **unfolding** *comp-def* **by** *auto*

have ∗: $(\lambda n.\ (Gromov\text{-}extension\ f)\ (u\ n)) \longrightarrow (Gromov\text{-}extension\ f)\ x$
apply (*rule continuous-within-tendsto-compose*[*OF - - u*(*2*), *of UNIV*])
using *homeomorphism-on-continuous*[*OF Gromov-extension-isometry-homeomorphism*[*OF assms*(*1*)]] **unfolding** *comp-def*
by (*auto simp add*: *continuous-on-eq-continuous-within*)
have $(\lambda n.\ (Gromov\text{-}extension\ g)\ ((Gromov\text{-}extension\ f)\ (u\ n))) \longrightarrow Gromov\text{-}extension\ g\ ((Gromov\text{-}extension\ f)\ x)$
apply (*rule continuous-within-tendsto-compose*[*OF - - ∗*, *of UNIV*])
using *homeomorphism-on-continuous*[*OF Gromov-extension-isometry-homeomorphism*[*OF assms*(*2*)]] **unfolding** *comp-def*
by (*auto simp add*: *continuous-on-eq-continuous-within*)
then show *?thesis* **using** *LIMSEQ-unique A comp-def* **by** *auto*
qed
ultimately have *Gromov-extension* (*g o f*) *x* = (*Gromov-extension g o Gromov-extension f*) *x* **for** *x*
using *not-in-Gromov-boundary* **by** *force*
then show *?thesis* **by** *auto*
qed

We specialize the previous results to bijective isometries, as this is the setting where they will be used most of the time.

lemma *Gromov-extension-isometry*:
assumes *isometry f*
shows *homeomorphism-on UNIV* (*Gromov-extension f*)
continuous-on UNIV (*Gromov-extension f*)
continuous (*at x*) (*Gromov-extension f*)
using *Gromov-extension-isometry-homeomorphism*[*OF isometryD*(*1*)[*OF assms*]] *homeomorphism-on-continuous* **apply** *auto*
using ‹*homeomorphism-on UNIV* (*Gromov-extension f*)› *continuous-on-eq-continuous-within homeomorphism-on-continuous* **by** *blast*

lemma *Gromov-extension-isometry-composition*:
assumes *isometry f*
isometry g
shows *Gromov-extension* (*g o f*) = *Gromov-extension g o Gromov-extension f*
using *Gromov-extension-isometry-on-composition*[*OF isometryD*(*1*)[*OF assms*(*1*)] *isometryD*(*1*)[*OF assms*(*2*)]] **by** *simp*

lemma *Gromov-extension-isometry-iterates*:
fixes *f*::′*a* ⇒ (′*a*::*Gromov-hyperbolic-space*)
assumes *isometry f*
shows *Gromov-extension* (*f*^^*n*) = (*Gromov-extension f*)^^*n*
apply (*induction n*) **using** *Gromov-extension-isometry-composition*[*OF isometry-iterates*[*OF assms*] *assms*] **unfolding** *comp-def* **by** *auto*

lemma *Gromov-extension-isometry-inv*:
assumes *isometry f*
shows *inv* (*Gromov-extension f*) = *Gromov-extension* (*inv f*)
bij (*Gromov-extension f*)

proof −
 have ∗: (*inv f*) *o f* = *id*
  using *isometry-inverse*(*2*)[*OF assms*] **by** (*simp add*: *bij-is-inj*)
 have *Gromov-extension* ((*inv f*) *o f*) = *Gromov-extension* (*inv f*) *o Gromov-extension f*
  by (*rule Gromov-extension-isometry-composition*[*OF assms isometry-inverse*(*1*)[*OF assms*]])
 then have *A*: *Gromov-extension* (*inv f*) *o Gromov-extension f* = *id*
  unfolding ∗ **by** *auto*
 have ∗: *f o* (*inv f*) = *id*
  using *isometry-inverse*(*2*)[*OF assms*] **by** (*meson bij-is-surj surj-iff*)
 have *Gromov-extension* (*f o* (*inv f*)) = *Gromov-extension f o Gromov-extension* (*inv f*)
  by (*rule Gromov-extension-isometry-composition*[*OF isometry-inverse*(*1*)[*OF assms*] *assms*])
 then have *B*: *Gromov-extension f o Gromov-extension* (*inv f*) = *id*
  unfolding ∗ **by** *auto*
 show *bij* (*Gromov-extension f*)
  using *A B* **unfolding** *bij-def* **apply** *auto*
  by (*metis inj-on-id inj-on-imageI2*, *metis B comp-apply id-def rangeI*)
 show *inv* (*Gromov-extension f*) = *Gromov-extension* (*inv f*)
  using *B* ‹*bij* (*Gromov-extension f*)› *bij-is-inj inv-o-cancel left-right-inverse-eq* **by** *blast*
qed

We will especially use fixed points on the boundary. We note that if a point is fixed by (the Gromov extension of) a map, then it is fixed by (the Gromov extension of) its inverse.

lemma *Gromov-extension-inv-fixed-point*:
 assumes *isometry* (*f*::′*a*::*Gromov-hyperbolic-space* ⇒ ′*a*) *Gromov-extension f xi* = *xi*
 shows *Gromov-extension* (*inv f*) *xi* = *xi*
by (*metis Gromov-extension-isometry-inv*(*1*) *Gromov-extension-isometry-inv*(*2*) *assms*(*1*) *assms*(*2*) *bij-betw-def inv-f-f*)

The extended Gromov product is invariant under isometries. This follows readily from the definition, but still the proof is not fully automatic, unfortunately.

lemma *Gromov-extension-preserves-extended-Gromov-product*:
 assumes *isometry f*
 shows *extended-Gromov-product-at* (*f x*) (*Gromov-extension f xi*) (*Gromov-extension f eta*) = *extended-Gromov-product-at x xi eta*
proof −
 have {*liminf* (λ*n*. *ereal* (*Gromov-product-at* (*f x*) (*u n*) (*v n*))) |*u v*.
   (λ*n*. *to-Gromov-completion* (*u n*)) ⟶ *Gromov-extension f xi* ∧ (λ*n*. *to-Gromov-completion* (*v n*)) ⟶ *Gromov-extension f eta*} =
  {*liminf* (λ*n*. *ereal* (*Gromov-product-at x* (*u n*) (*v n*))) |*u v*.
   (λ*n*. *to-Gromov-completion* (*u n*)) ⟶ *xi* ∧ (λ*n*. *to-Gromov-completion* (*v n*)) ⟶ *eta*}

proof (*auto*)
fix *u v* **assume** *H*: (*λn. to-Gromov-completion* (*u n*)) ⟶ *Gromov-extension f xi*
(*λn. to-Gromov-completion* (*v n*)) ⟶ *Gromov-extension f eta*
define *u′* **where** *u′* = (*λn.* (*inv f*) (*u n*))
define *v′* **where** *v′* = (*λn.* (*inv f*) (*v n*))
have (*λn. to-Gromov-completion* (*u′ n*)) ⟶ *Gromov-extension* (*inv f*) (*Gromov-extension f xi*)
unfolding *u′-def Gromov-extension-inside-space*[*symmetric*]
apply (*rule iffD1*[*OF homeomorphism-on-compose*[*OF Gromov-extension-isometry-homeomorphism*[*OF isometryD*(*1*)[*OF isometry-inverse*(*1*)[*OF assms*]]]]])
using *H*(*1*) **by** *auto*
moreover have *Gromov-extension* (*inv f*) (*Gromov-extension f xi*) = *xi*
using *Gromov-extension-isometry-composition*[*OF assms isometry-inverse*(*1*)[*OF assms*], *symmetric*] **unfolding** *comp-def*
using *bij-is-inj*[*OF isometry-inverse*(*2*)[*OF assms*]]
by (*simp add*: ‹*Gromov-extension* (*inv f*) ∘ *Gromov-extension f* = *Gromov-extension* (*inv f* ∘ *f*)› *pointfree-idE*)
ultimately have *U*: (*λn. to-Gromov-completion* (*u′ n*)) ⟶ *xi* **by** *simp*
have (*λn. to-Gromov-completion* (*v′ n*)) ⟶ *Gromov-extension* (*inv f*) (*Gromov-extension f eta*)
unfolding *v′-def Gromov-extension-inside-space*[*symmetric*]
apply (*rule iffD1*[*OF homeomorphism-on-compose*[*OF Gromov-extension-isometry-homeomorphism*[*OF isometryD*(*1*)[*OF isometry-inverse*(*1*)[*OF assms*]]]]])
using *H*(*2*) **by** *auto*
moreover have *Gromov-extension* (*inv f*) (*Gromov-extension f eta*) = *eta*
using *Gromov-extension-isometry-composition*[*OF assms isometry-inverse*(*1*)[*OF assms*], *symmetric*] **unfolding** *comp-def*
using *bij-is-inj*[*OF isometry-inverse*(*2*)[*OF assms*]]
by (*simp add*: ‹*Gromov-extension* (*inv f*) ∘ *Gromov-extension f* = *Gromov-extension* (*inv f* ∘ *f*)› *pointfree-idE*)
ultimately have *V*: (*λn. to-Gromov-completion* (*v′ n*)) ⟶ *eta* **by** *simp*
have *uv*: *u n* = *f* (*u′ n*) *v n* = *f* (*v′ n*) **for** *n*
unfolding *u′-def v′-def* **by** (*auto simp add*: *assms isometryD*(*3*) *surj-f-inv-f*)
have *Gromov-product-at* (*f x*) (*u n*) (*v n*) = *Gromov-product-at x* (*u′ n*) (*v′ n*) **for** *n*
unfolding *uv* **using** *assms* **by** (*simp add*: *Gromov-product-isometry isometry-def*)
then have *liminf* (*λn. ereal* (*Gromov-product-at* (*f x*) (*u n*) (*v n*))) = *liminf* (*λn. ereal* (*Gromov-product-at x* (*u′ n*) (*v′ n*)))
by *auto*
then show ∃ *u′ v′*.
liminf (*λn. ereal* (*Gromov-product-at* (*f x*) (*u n*) (*v n*))) = *liminf* (*λn. ereal* (*Gromov-product-at x* (*u′ n*) (*v′ n*))) ∧
(*λn. to-Gromov-completion* (*u′ n*)) ⟶ *xi* ∧ (*λn. to-Gromov-completion* (*v′ n*)) ⟶ *eta*
using *U V* **by** *blast*
next
fix *u v* **assume** *H*: (*λn. to-Gromov-completion* (*u n*)) ⟶ *xi*

$(\lambda n.\ \textit{to-Gromov-completion}\ (v\ n)) \longrightarrow \textit{eta}$
define u' **where** $u' = (\lambda n.\ f\ (u\ n))$
define v' **where** $v' = (\lambda n.\ f\ (v\ n))$
have U: $(\lambda n.\ \textit{to-Gromov-completion}\ (u'\ n)) \longrightarrow \textit{Gromov-extension f xi}$
unfolding *u'-def Gromov-extension-inside-space*[*symmetric*]
apply (*rule iffD1*[*OF homeomorphism-on-compose*[*OF Gromov-extension-isometry-homeomorphism*[*OF isometryD*(*1*)[*OF assms*]]]])
using *H*(*1*) **by** *auto*
have V: $(\lambda n.\ \textit{to-Gromov-completion}\ (v'\ n)) \longrightarrow \textit{Gromov-extension f eta}$
unfolding *v'-def Gromov-extension-inside-space*[*symmetric*]
apply (*rule iffD1*[*OF homeomorphism-on-compose*[*OF Gromov-extension-isometry-homeomorphism*[*OF isometryD*(*1*)[*OF assms*]]]])
using *H*(*2*) **by** *auto*
have *Gromov-product-at* (*f x*) (*u' n*) (*v' n*) = *Gromov-product-at x* (*u n*) (*v n*) **for** *n*
unfolding *u'-def v'-def* **using** *assms* **by** (*simp add*: *Gromov-product-isometry isometry-def*)
then have *liminf* (λn. *ereal* (*Gromov-product-at x* (*u n*) (*v n*))) = *liminf* (λn. *ereal* (*Gromov-product-at* (*f x*) (*u' n*) (*v' n*)))
by *auto*
then show $\exists\, u'\ v'$.
liminf (λn. *ereal* (*Gromov-product-at x* (*u n*) (*v n*))) = *liminf* (λn. *ereal* (*Gromov-product-at* (*f x*) (*u' n*) (*v' n*))) $\wedge$
$(\lambda n.\ \textit{to-Gromov-completion}\ (u'\ n)) \longrightarrow \textit{Gromov-extension f xi}\ \wedge$ $(\lambda n.\ \textit{to-Gromov-completion}\ (v'\ n)) \longrightarrow \textit{Gromov-extension f eta}$
using *U V* **by** *auto*
qed
then show *?thesis*
unfolding *extended-Gromov-product-at-topological* **by** *auto*
qed

end

18 Busemann functions

theory *Busemann-Function*
imports *Boundary-Extension Ergodic-Theory.Fekete*
begin

The Busemann function $B_\xi(x, y)$ measures the difference $d(\xi, x) - d(\xi, y)$, where ξ is a point at infinity and x and y are inside a Gromov hyperbolic space. This is not well defined in this way, as we are subtracting two infinities, but one can make sense of this difference by considering the behavior along a sequence tending to ξ. The limit may depend on the sequence, but as usual in Gromov hyperbolic spaces it only depends on the sequence up to a uniform constant. Thus, we may define the Busemann function using for instance the supremum of the limsup over all possible sequences – other choices would give rise to equivalent definitions, up to some multiple of δ.

definition *Busemann-function-at*::(*'a*::*Gromov-hyperbolic-space*) *Gromov-completion* $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *real*
where *Busemann-function-at xi x y* = *real-of-ereal* (
Sup {*limsup* (λ*n*. *ereal*(*dist x* (*u n*) $-$ *dist y* (*u n*))) |*u*. (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*})

Since limsups are only defined for complete orders currently, the definition goes through ereals, and we go back to reals afterwards. However, there is no real difficulty here, as eveything is bounded above and below (by $d(x, y)$ and $-d(x, y)$ respectively.

lemma *Busemann-function-ereal*:
ereal(*Busemann-function-at xi x y*) = *Sup* {*limsup* (λ*n*. *ereal*(*dist x* (*u n*) $-$ *dist y* (*u n*))) |*u*. (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*}
proof $-$
have *A*: *Sup* {*limsup* (λ*n*. *ereal*(*dist x* (*u n*) $-$ *dist y* (*u n*))) |*u*. (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*} $\leq$ *dist x y*
by (*rule Sup-least*, *auto intro*!: *Limsup-bounded always-eventually mono-intros simp add*: *algebra-simps*)
have *B*: *Sup* {*limsup* (λ*n*. *ereal*(*dist x* (*u n*) $-$ *dist y* (*u n*))) |*u*. (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*} $\geq$ $-$*dist x y*
proof $-$
obtain *u* **where** $*$: (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*
using *rep-Gromov-completion-limit*[*of xi*] **by** *blast*
have *ereal*($-$*dist x y*) $\leq$ *limsup* (λ*n*. *ereal*(*dist x* (*u n*) $-$ *dist y* (*u n*)))
by (*rule le-Limsup*, *auto intro*!: *always-eventually mono-intros simp add*: *algebra-simps*)
also have ... $\leq$ *Sup* {*limsup* (λ*n*. *ereal*(*dist x* (*u n*) $-$ *dist y* (*u n*))) |*u*. (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*}
apply (*rule Sup-upper*) **using** $*$ **by** *auto*
finally show *?thesis* **by** *simp*
qed
show *?thesis*
unfolding *Busemann-function-at-def* **apply** (*rule ereal-real'*) **using** *A B* **by** *auto*
qed

If ξ is not at infinity, then the Busemann function is simply the difference of the distances.

lemma *Busemann-function-inner*:
Busemann-function-at (*to-Gromov-completion z*) *x y* = *dist x z* $-$ *dist y z*
proof $-$
have *L*: *limsup* (λ*n*. *ereal*(*dist x* (*u n*) $-$ *dist y* (*u n*))) = *dist x z* $-$ *dist y z* **if** *u* $\longrightarrow$ *z* **for** *u*
by (*rule lim-imp-Limsup*, *simp*, *intro tendsto-intros that*)
have *Sup* {*limsup* (λ*n*. *ereal*(*dist x* (*u n*) $-$ *dist y* (*u n*))) |*u*. *u* $\longrightarrow$ *z*}
= *dist x z* $-$ *dist y z*
proof $-$
obtain *u* **where** *u*: *u* $\longrightarrow$ *z*
by *auto*

show *?thesis*
apply (*rule order.antisym*)
apply (*subst Sup-le-iff*) **using** *L* **apply** *auto*[*1*]
apply (*subst L*[*OF u*, *symmetric*]) **apply** (*rule Sup-upper*) **using** *u* **by** *auto*
qed
then have *ereal* (*Busemann-function-at* (*to-Gromov-completion z*) *x y*) = *dist x z* − *dist y z*
unfolding *Busemann-function-ereal* **by** *auto*
then show *?thesis* **by** *auto*
qed

The Busemann function measured at the same points vanishes.

lemma *Busemann-function-xx* [*simp*]:
Busemann-function-at xi x x = *0*
proof −
have ∗: {*limsup* (*λn. ereal*(*dist x* (*u n*) − *dist x* (*u n*))) |*u.* (*λn. to-Gromov-completion* (*u n*)) ⟶ *xi*} = {*0*}
by (*auto simp add*: *zero-ereal-def*[*symmetric*] *intro*!: *lim-imp-Limsup rep-Gromov-completion-limit*[*of xi*])
have *ereal* (*Busemann-function-at xi x x*) = *ereal 0*
unfolding *Busemann-function-ereal* ∗ **by** *auto*
then show *?thesis*
by *auto*
qed

Perturbing the points gives rise to a variation of the Busemann function bounded by the size of the variations. This is obvious for inner Busemann functions, and everything passes readily to the limit.

lemma *Busemann-function-mono* [*mono-intros*]:
Busemann-function-at xi x y ≤ *Busemann-function-at xi x′ y′* + *dist x x′* + *dist y y′*
proof −
have *A*: *limsup* (*λn. ereal* (*dist x* (*u n*) − *dist y* (*u n*)))
≤ *ereal*(*Busemann-function-at xi x′ y′*) + *ereal* (*dist x x′* + *dist y y′*)
if (*λn. to-Gromov-completion* (*u n*)) ⟶ *xi* **for** *u*
proof −
have ∗: *dist x z* + *dist y′ z* ≤ *dist x x′* + (*dist y y′* + (*dist x′ z* + *dist y z*)) **for** *z*
using *add-mono*[*OF dist-triangle*[*of x z x′*] *dist-triangle*[*of y′ z y*]] *dist-commute*[*of y y′*] **by** *auto*
have *limsup* (*λn. ereal* (*dist x* (*u n*) − *dist y* (*u n*))) + (− *ereal* (*dist x x′* + *dist y y′*))
= *limsup* (*λn. ereal* (*dist x* (*u n*) − *dist y* (*u n*)) + (− *ereal* (*dist x x′* + *dist y y′*)))
by (*rule Limsup-add-ereal-right*[*symmetric*], *auto*)
also have ... ≤ *limsup* (*λn. ereal* (*dist x′* (*u n*) − *dist y′* (*u n*)))
by (*auto intro*!: *Limsup-mono always-eventually simp*: *algebra-simps* ∗)
also have ... ≤ *Sup* {*limsup* (*λn. ereal* (*dist x′* (*u n*) − *dist y′* (*u n*))) |*u.* (*λn. to-Gromov-completion* (*u n*)) ⟶ *xi*}

apply (*rule Sup-upper*) **using** *that* **by** *auto*
finally have *limsup* ($\lambda n.$ *ereal* (*dist x* (*u n*) $-$ *dist y* (*u n*))) + ($-$ *ereal* (*dist x x′* + *dist y y′*))
$\leq$ *ereal*(*Busemann-function-at xi x′ y′*)
unfolding *Busemann-function-ereal* **by** *auto*
then show *?thesis*
unfolding *minus-ereal-def*[*symmetric*] **by** (*subst ereal-minus-le*[*symmetric*], *auto*)
qed
have *ereal* (*Busemann-function-at xi x y*) $\leq$ *ereal*(*Busemann-function-at xi x′ y′*) + *dist x x′* + *dist y y′*
unfolding *Busemann-function-ereal*[*of xi x y*] **using** *A* **by** (*auto intro*!: *Sup-least simp*: *algebra-simps*)
then show *?thesis* **by** *simp*
qed

In particular, it follows that the Busemann function $B_\xi(x, y)$ is bounded in absolute value by $d(x, y)$.

lemma *Busemann-function-le-dist* [*mono-intros*]:
abs(*Busemann-function-at xi x y*) $\leq$ *dist x y*
using *Busemann-function-mono*[*of xi x y y y*] *Busemann-function-mono*[*of xi x x x y*] **by** *auto*

lemma *Busemann-function-Lipschitz* [*mono-intros*]:
abs(*Busemann-function-at xi x y* $-$ *Busemann-function-at xi x′ y′*) $\leq$ *dist x x′* + *dist y y′*
using *Busemann-function-mono*[*of xi x y x′ y′*] *Busemann-function-mono*[*of xi x′ y′ x y*] **by** (*simp add*: *dist-commute*)

By the very definition of the Busemann function, the difference of distance functions is bounded above by the Busemann function when one converges to ξ.

lemma *Busemann-function-limsup*:
assumes ($\lambda n.$ *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*
shows *limsup* ($\lambda n.$ *dist x* (*u n*) $-$ *dist y* (*u n*)) $\leq$ *Busemann-function-at xi x y*
unfolding *Busemann-function-ereal* **apply** (*rule Sup-upper*) **using** *assms* **by** *auto*

There is also a corresponding bound below, but with the loss of a constant. This follows from the hyperbolicity of the space and a simple computation.

lemma *Busemann-function-liminf*:
assumes ($\lambda n.$ *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*
shows *Busemann-function-at xi x y* $\leq$ *liminf* ($\lambda n.$ *dist* (*x*::*′a*::*Gromov-hyperbolic-space*) (*u n*) $-$ *dist y* (*u n*)) + *2* $*$ *deltaG*(*TYPE*(*′a*))
proof (*cases xi*)
case (*to-Gromov-completion z*)
have $*$: *liminf* ($\lambda n.$ *dist x* (*u n*) $-$ *dist y* (*u n*)) = *dist x z* $-$ *dist y z*
apply (*rule lim-imp-Liminf*, *simp*, *intro tendsto-intros*)
using *assms* **unfolding** *to-Gromov-completion* **by** *auto*

show *?thesis*
unfolding *to-Gromov-completion plus-ereal.simps*(*1*)[*symmetric*] *Busemann-function-inner* ∗ **by** *auto*
next
case *boundary*
have *I*: *limsup* (*λn. ereal*(*dist x* (*v n*) − *dist y* (*v n*))) ≤ *liminf* (*λn. ereal*(*dist x* (*u n*) − *dist y* (*u n*))) + *2* ∗ *deltaG*(*TYPE*(*'a*))
if *v*: (*λn. to-Gromov-completion* (*v n*)) ⟶ *xi* **for** *v*
proof −
obtain *N* **where** *N*: ⋀*m n. m* ≥ *N* ⟹ *n* ≥ *N* ⟹ *Gromov-product-at x* (*u m*) (*v n*) ≥ *dist x y*
using *same-limit-imp-Gromov-product-tendsto-infinity*[*OF boundary assms v*] **by** *blast*
have *A*: *dist x* (*v n*) − *dist y* (*v n*) − *2* ∗ *deltaG*(*TYPE*(*'a*)) ≤ *dist x* (*u m*) − *dist y* (*u m*) **if** *m* ≥ *N n* ≥ *N* **for** *m n*
proof −
have *Gromov-product-at x* (*v n*) *y* ≤ *dist x y*
by (*intro mono-intros*)
then have *min* (*Gromov-product-at x* (*u m*) (*v n*)) (*Gromov-product-at x* (*v n*) *y*) = *Gromov-product-at x* (*v n*) *y*
using *N*[*OF* ‹*m* ≥ *N*› ‹*n* ≥ *N*›] **by** *linarith*
moreover have *Gromov-product-at x* (*u m*) *y* ≥ *min* (*Gromov-product-at x* (*u m*) (*v n*)) (*Gromov-product-at x* (*v n*) *y*) − *deltaG*(*TYPE*(*'a*))
by (*intro mono-intros*)
ultimately have *Gromov-product-at x* (*u m*) *y* ≥ *Gromov-product-at x* (*v n*) *y* − *deltaG*(*TYPE*(*'a*))
by *auto*
then show *?thesis*
unfolding *Gromov-product-at-def* **by** (*auto simp add*: *algebra-simps divide-simps dist-commute*)
qed
have *B*: *dist x* (*v n*) − *dist y* (*v n*) − *2* ∗ *deltaG*(*TYPE*(*'a*)) ≤ *liminf* (*λm. dist x* (*u m*) − *dist y* (*u m*)) **if** *n* ≥ *N* **for** *n*
apply (*rule Liminf-bounded*) **using** *A*[*OF* - *that*] **unfolding** *eventually-sequentially* **by** *auto*
have *C*: *dist x* (*v n*) − *dist y* (*v n*) ≤ *liminf* (*λm. dist x* (*u m*) − *dist y* (*u m*)) + *2* ∗ *deltaG*(*TYPE*(*'a*)) **if** *n* ≥ *N* **for** *n*
using *B*[*OF that*] **by** (*subst ereal-minus-le*[*symmetric*], *auto*)
show *?thesis*
apply (*rule Limsup-bounded*) **unfolding** *eventually-sequentially* **apply** (*rule exI*[*of* - *N*]) **using** *C* **by** *auto*
qed
show *?thesis*
unfolding *Busemann-function-ereal* **apply** (*rule Sup-least*) **using** *I* **by** *auto*
qed

To avoid formulating things in terms of liminf and limsup on ereal, the following formulation of the two previous lemmas is useful.

lemma *Busemann-function-inside-approx*:

assumes *e* > (*0*::*real*) (*λn. to-Gromov-completion* (*t n*::*'a*::*Gromov-hyperbolic-space*)) ⟶ *xi*
shows *eventually* (*λn. Busemann-function-at* (*to-Gromov-completion* (*t n*)) *x y* ≤ *Busemann-function-at xi x y* + *e*
∧ *Busemann-function-at* (*to-Gromov-completion* (*t n*)) *x y* ≥ *Busemann-function-at xi x y* − *2* ∗ *deltaG*(*TYPE*(*'a*)) − *e*) *sequentially*
proof −
have *A*: *eventually* (*λn. Busemann-function-at* (*to-Gromov-completion* (*t n*)) *x y* < *Busemann-function-at xi x y* + *ereal e*) *sequentially*
apply (*rule Limsup-lessD*)
unfolding *Busemann-function-inner* **using** *le-less-trans*[*OF Busemann-function-limsup*[*OF assms*(*2*)]] ‹*e* > *0*› **by** *auto*
have *B*: *eventually* (*λn. Busemann-function-at* (*to-Gromov-completion* (*t n*)) *x y* > *Busemann-function-at xi x y* −*2* ∗ *deltaG*(*TYPE*(*'a*)) − *ereal e*) *sequentially*
apply (*rule less-LiminfD*)
unfolding *Busemann-function-inner* **using** *less-le-trans*[*OF* - *Busemann-function-liminf*[*OF assms*(*2*)], *of ereal*(*Busemann-function-at xi x y*) − *ereal e x y*] ‹*e* > *0*› **apply** *auto*
apply (*unfold ereal-minus*(*1*)[*symmetric*], *subst ereal-minus-less-iff*, *simp*)+
unfolding *ereal-minus*(*1*)[*symmetric*] **by** (*simp only*: *ereal-minus-less-iff*, *auto simp add*: *algebra-simps*)
show *?thesis*
by (*rule eventually-mono*[*OF eventually-conj*[*OF A B*]], *auto*)
qed

The Busemann function is essentially a morphism, i.e., it should satisfy $B_\xi(x,z) = B_\xi(x,y) + B_\xi(y,z)$, as it is defined as a difference of distances. This is not exactly the case as there is a choice in the definition, but it is the case up to a uniform constant, as we show in the next few lemmas. One says that it is a *quasi-morphism*.

lemma *Busemann-function-triangle* [*mono-intros*]:
Busemann-function-at xi x z ≤ *Busemann-function-at xi x y* + *Busemann-function-at xi y z*
proof −
have *limsup* (*λn. dist x* (*u n*) − *dist z* (*u n*)) ≤ *Busemann-function-at xi x y* + *Busemann-function-at xi y z*
if (*λn. to-Gromov-completion* (*u n*)) ⟶ *xi* **for** *u*
proof −
have *limsup* (*λn. dist x* (*u n*) − *dist z* (*u n*)) = *limsup* (*λn. ereal* (*dist x* (*u n*) − *dist y* (*u n*)) + (*dist y* (*u n*) − *dist z* (*u n*)))
by *auto*
also have ... ≤ *limsup* (*λn. dist x* (*u n*) − *dist y* (*u n*)) + *limsup* (*λn. dist y* (*u n*) − *dist z* (*u n*))
by (*rule ereal-limsup-add-mono*)
also have ... ≤ *ereal*(*Busemann-function-at xi x y*) + *Busemann-function-at xi y z*
unfolding *Busemann-function-ereal* **using** *that* **by** (*auto intro*!: *add-mono Sup-upper*)
finally show *?thesis* **by** *auto*
qed

then have *ereal* (*Busemann-function-at xi x z*) $\leq$ *Busemann-function-at xi x y* + *Busemann-function-at xi y z*
unfolding *Busemann-function-ereal*[*of xi x z*] **by** (*auto intro*!: *Sup-least*)
then show *?thesis*
by *auto*
qed

lemma *Busemann-function-xy-yx* [*mono-intros*]:
Busemann-function-at xi x y + *Busemann-function-at xi y* (*x*::*'a*::*Gromov-hyperbolic-space*) $\leq$ *2* $*$ *deltaG*(*TYPE*(*'a*))
proof −
have $*$: − *liminf* (λ*n*. *ereal* (*dist y* (*u n*) − *dist x* (*u n*))) $\leq$ *ereal* (*2* $*$ *deltaG TYPE*(*'a*) − *Busemann-function-at xi y x*)
if (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi* **for** *u*
using *Busemann-function-liminf*[*of - xi y x*, *OF that*] *ereal-minus-le-minus-plus* **unfolding** *ereal-minus*(*1*)[*symmetric*]
by *fastforce*

have *ereal* (*Busemann-function-at xi x y*) = *Sup* {*limsup* (λ*n*. *ereal*(*dist x* (*u n*) − *dist y* (*u n*))) |*u*. (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*}
unfolding *Busemann-function-ereal* **by** *auto*
also have ... = *Sup* {*limsup* (λ*n*. − *ereal*(*dist y* (*u n*) − *dist x* (*u n*))) |*u*. (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*}
by *auto*
also have ... = *Sup* {− *liminf* (λ*n*. *ereal*(*dist y* (*u n*) − *dist x* (*u n*))) |*u*. (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*}
unfolding *ereal-Limsup-uminus* **by** *auto*
also have ... $\leq$ *2* $*$ *deltaG*(*TYPE*(*'a*)) − *ereal*(*Busemann-function-at xi y x*)
by (*auto intro*!: *Sup-least* $*$)
finally show *?thesis*
by *simp*
qed

theorem *Busemann-function-quasi-morphism* [*mono-intros*]:
|*Busemann-function-at xi x y* + *Busemann-function-at xi y z* − *Busemann-function-at xi x* (*z*::*'a*::*Gromov-hyperbolic-space*)| $\leq$ *2* $*$ *deltaG*(*TYPE*(*'a*))
using *Busemann-function-triangle*[*of xi x z y*] *Busemann-function-triangle*[*of xi x y z*] *Busemann-function-xy-yx*[*of xi y z*] **by** *auto*

The extended Gromov product can be bounded from below by the Busemann function.

lemma *Busemann-function-le-Gromov-product*:
− *Busemann-function-at xi y x*/*2* $\leq$ *extended-Gromov-product-at x xi* (*to-Gromov-completion y*)
proof −
have *A*: −*ereal*(*Busemann-function-at xi y x*/*2*) $\leq$ *liminf* (λ*n*. *Gromov-product-at x* (*u n*) *y*)
if (λ*n*. *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi* **for** *u*
proof −

have ∗: *limsup* (*λn.* − *ereal* (*Gromov-product-at x* (*u n*) *y*) ∗ *2*) ≤ *limsup* (*λn. ereal* (*dist y* (*u n*) − *dist x* (*u n*)))
by (*auto intro*!: *Limsup-mono always-eventually simp*: *algebra-simps Gromov-product-at-def divide-simps dist-commute*)
also have ... ≤ *ereal*(*Busemann-function-at xi y x*)
unfolding *Busemann-function-ereal* **using** *that* **by** (*auto intro*!: *Sup-upper*)
finally have −*ereal*(*Busemann-function-at xi y x*) ≤ *liminf* (*λn. Gromov-product-at x* (*u n*) *y*) ∗ *ereal 2*
apply (*subst ereal-uminus-le-reorder*, *subst ereal-mult-minus-left*[*symmetric*], *subst ereal-Limsup-uminus*[*symmetric*])
by (*subst limsup-ereal-mult-right*[*symmetric*], *auto*)
moreover have −*ereal*(*z*/*2*) ≤ *t* **if** −*ereal z* ≤ *t* ∗ *ereal 2* **for** *z t*
proof −
have ∗: −*ereal*(*z*/*2*) = −*ereal z* / *ereal 2*
unfolding *ereal-divide* **by** *auto*
have *0* < *ereal 2*
by *auto*
then show *?thesis* **unfolding** ∗ **using** *that*
by (*metis* (*no-types*) *PInfty-neq-ereal*(*2*) *ereal-divide-le-posI ereal-uminus-eq-iff mult.commute that*)
qed
ultimately show *?thesis* **by** *auto*
qed
show *?thesis*
unfolding *extended-Gromov-product-at-def* **proof** (*rule Inf-greatest*, *auto*)
fix *u v* **assume** *uv*: *xi* = *abs-Gromov-completion u abs-Gromov-completion v* = *to-Gromov-completion y Gromov-completion-rel u u Gromov-completion-rel v v*
then have *L*: (*λn. to-Gromov-completion* (*u n*)) ⟶ *xi*
using *abs-Gromov-completion-limit* **by** *auto*
have ∗: *v n* = *y* **for** *n*
using *uv* **by** (*metis* (*mono-tags*, *opaque-lifting*) *Gromov-completion-rel-def Quotient3-Gromov-completion Quotient3-rep-abs abs-Gromov-completion-in-Gromov-boundary not-in-Gromov-boundary′ rep-Gromov-completion-to-Gromov-completion*)
show *ereal* (− (*Busemann-function-at* (*abs-Gromov-completion u*) *y x* / *2*)) ≤ *liminf* (*λn. ereal* (*Gromov-product-at x* (*u n*) (*v n*)))
unfolding *uv*(*1*)[*symmetric*] ∗ **using** *A*[*OF L*] **by** *simp*
qed
qed

It follows that, if the Busemann function tends to minus infinity, i.e., the distance to ξ becomes smaller and smaller in a suitable sense, then the sequence is converging to ξ. This is only an implication: one can have sequences tending to ξ for which the Busemann function does not tend to $-\infty$. This is in fact a stronger notion of convergence, sometimes called radial convergence.

proposition *Busemann-function-minus-infinity-imp-convergent*:
assumes ((*λn. Busemann-function-at xi* (*u n*) *x*) ⟶ −∞) *F*
shows ((*λn. to-Gromov-completion* (*u n*)) ⟶ *xi*) *F*
proof (*cases trivial-limit F*)

case *True*
then show *?thesis* **by** *auto*
next
case *False*
have *xi* ∈ *Gromov-boundary*
proof (*cases xi*)
case (*to-Gromov-completion z*)
then have *ereal*(*Busemann-function-at xi* (*u n*) *x*) ≥ − *dist x z* **for** *n*
unfolding *to-Gromov-completion Busemann-function-inner* **by** *auto*
then have −∞ ≥ −*dist x z*
using *tendsto-lowerbound*[*OF assms always-eventually False*] **by** *metis*
then have *False*
by *auto*
then show *?thesis* **by** *auto*
qed
have ((λ*n*. − *ereal* (*Busemann-function-at xi* (*u n*) *x*) / *2*) ⟶ (− (−∞)/*2*)) *F*
apply (*intro tendsto-intros*) **using** *assms* **by** *auto*
then have ∗: ((λ*n*. − *ereal* (*Busemann-function-at xi* (*u n*) *x*) / *2*) ⟶ ∞) *F*
by *auto*
have ∗∗: ((λ*n*. *extended-Gromov-product-at x xi* (*to-Gromov-completion* (*u n*))) ⟶ ∞) *F*
apply (*rule tendsto-sandwich*[*of* λ*n*. − *ereal* (*Busemann-function-at xi* (*u n*) *x*) / *2* - - λ*n*. ∞, *OF always-eventually always-eventually*])
using *Busemann-function-le-Gromov-product*[*of xi* - *x*] ∗ **by** *auto*
show *?thesis*
using *extended-Gromov-product-tendsto-PInf-a-b*[*OF* ∗∗, *of basepoint*]
by (*auto simp add*: *Gromov-completion-boundary-limit*[*OF* ‹*xi* ∈ *Gromov-boundary*›] *extended-Gromov-product-at-commute*)
qed

Busemann functions are invariant under isometries. This is trivial as everything is defined in terms of the distance, but the definition in terms of supremum and limsups makes the proof tedious.

lemma *Busemann-function-isometry*:
assumes *isometry f*
shows *Busemann-function-at* (*Gromov-extension f xi*) (*f x*) (*f y*) = *Busemann-function-at xi x y*
proof −
have {*limsup* (λ*n*. *ereal*(*dist x* (*u n*) − *dist y* (*u n*))) |*u*. (λ*n*. *to-Gromov-completion* (*u n*)) ⟶ *xi*}
= {*limsup* (λ*n*. *ereal*(*dist* (*f x*) (*v n*) − *dist* (*f y*) (*v n*))) |*v*. (λ*n*. *to-Gromov-completion* (*v n*)) ⟶ *Gromov-extension f xi*}
proof (*auto*)
fix *u* **assume** *u*: (λ*n*. *to-Gromov-completion* (*u n*)) ⟶ *xi*
define *v* **where** *v* = *f o u*
have (λ*n*. *to-Gromov-completion* (*v n*)) ⟶ *Gromov-extension f xi*
unfolding *v-def comp-def Gromov-extension-inside-space*[*symmetric*] **using** *u* *Gromov-extension-isometry*(*2*)[*OF* ‹*isometry f*›]

by (*metis continuous-on filterlim-compose iso-tuple-UNIV-I tendsto-at-iff-tendsto-nhds*)
moreover have *limsup* ($\lambda n.$ *ereal* (*dist x* (*u n*) $-$ *dist y* (*u n*))) $=$ *limsup* ($\lambda n.$ *ereal* (*dist* (*f x*) (*v n*) $-$ *dist* (*f y*) (*v n*)))
unfolding *v-def comp-def isometryD*(*2*)[*OF* ‹*isometry f*›] **by** *simp*
ultimately show $\exists$ *v. limsup* ($\lambda n.$ *ereal* (*dist x* (*u n*) $-$ *dist y* (*u n*))) $=$ *limsup* ($\lambda n.$ *ereal* (*dist* (*f x*) (*v n*) $-$ *dist* (*f y*) (*v n*))) $\wedge$
($\lambda n.$ *to-Gromov-completion* (*v n*)) $\longrightarrow$ *Gromov-extension f xi*
by *blast*
next
fix *v* **assume** *v*: ($\lambda n.$ *to-Gromov-completion* (*v n*)) $\longrightarrow$ *Gromov-extension f xi*
define *u* **where** *u* $=$ (*inv f*) *o v*
have *isometry* (*inv f*)
using *isometry-inverse*(*1*)[*OF* ‹*isometry f*›] **by** *simp*
have $*$: *inv f* (*f z*) $=$ *z* **for** *z*
using *isometry-inverse*(*2*)[*OF* ‹*isometry f*›] **by** (*simp add*: *bij-betw-def*)
have $**$: (*Gromov-extension* (*inv f*) (*Gromov-extension f xi*)) $=$ *xi*
using *Gromov-extension-isometry-composition*[*OF* ‹*isometry f*› ‹*isometry* (*inv f*)›]
unfolding *comp-def* **using** *isometry-inverse*(*2*)[*OF* ‹*isometry f*›] **by** (*auto simp*: $*$, *metis*)
have ($\lambda n.$ *to-Gromov-completion* (*u n*)) $\longrightarrow$ *Gromov-extension* (*inv f*) (*Gromov-extension f xi*)
unfolding *u-def comp-def Gromov-extension-inside-space*[*symmetric*] **using** *v Gromov-extension-isometry*(*2*)[*OF* ‹*isometry* (*inv f*)›]
by (*metis continuous-on filterlim-compose iso-tuple-UNIV-I tendsto-at-iff-tendsto-nhds*)
then have ($\lambda n.$ *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*
using $**$ **by** *auto*
moreover have *limsup* ($\lambda n.$ *ereal* (*dist* ((*inv f*) (*f x*)) (*u n*) $-$ *dist* ((*inv f*) (*f y*)) (*u n*))) $=$ *limsup* ($\lambda n.$ *ereal* (*dist* (*f x*) (*v n*) $-$ *dist* (*f y*) (*v n*)))
unfolding *u-def comp-def isometryD*(*2*)[*OF* ‹*isometry* (*inv f*)›] **by** *simp*
ultimately show $\exists$ *u. limsup* ($\lambda n.$ *ereal* (*dist* (*f x*) (*v n*) $-$ *dist* (*f y*) (*v n*))) $=$ *limsup* ($\lambda n.$ *ereal* (*dist x* (*u n*) $-$ *dist y* (*u n*))) $\wedge$ ($\lambda n.$ *to-Gromov-completion* (*u n*)) $\longrightarrow$ *xi*
by (*simp add*: $*$, *force*)
qed
then have *ereal* (*Busemann-function-at xi x y*) $=$ *ereal* (*Busemann-function-at* (*Gromov-extension f xi*) (*f x*) (*f y*))
unfolding *Busemann-function-ereal* **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *dist-le-max-Busemann-functions* [*mono-intros*]:
assumes *xi* $\neq$ *eta*
shows *dist x* (*y*::*'a*::*Gromov-hyperbolic-space*) $\leq$ *2* $*$ *real-of-ereal* (*extended-Gromov-product-at y xi eta*)
$+$ *max* (*Busemann-function-at xi x y*) (*Busemann-function-at eta x y*) $+$ *2* $*$ *deltaG*(*TYPE*(*'a*))
proof $-$

have *A*: *ereal*(*dist x y* − *2* ∗ *deltaG*(*TYPE*(*'a*)) − *max* (*Busemann-function-at xi x y*) (*Busemann-function-at eta x y*)) / *ereal 2* ≤
liminf (λ*n*. *ereal*(*Gromov-product-at y* (*u n*) (*v n*)))
if *uv*: *abs-Gromov-completion u* = *xi abs-Gromov-completion v* = *eta Gromov-completion-rel u u Gromov-completion-rel v v* **for** *u v*
proof −
have *C*: (λ*n*. *to-Gromov-completion* (*u n*)) ⟶ *xi* (λ*n*. *to-Gromov-completion* (*v n*)) ⟶ *eta*
using *uv abs-Gromov-completion-limit* **by** *auto*
have *ereal*(*dist x y*) ≤ *ereal*(*2* ∗ *Gromov-product-at y* (*u n*) (*v n*)) + *2* ∗ *deltaG*(*TYPE*(*'a*)) + *max* (*ereal*(*dist x* (*u n*) − *dist y* (*u n*))) (*ereal*(*dist x* (*v n*) − *dist y* (*v n*))) **for** *n*
proof −
have *min* (*Gromov-product-at y* (*u n*) *x*) (*Gromov-product-at y x* (*v n*)) ≤ *Gromov-product-at y* (*u n*) (*v n*) + *deltaG*(*TYPE*(*'a*))
by (*intro mono-intros*)
then consider *Gromov-product-at y* (*u n*) *x* ≤ *Gromov-product-at y* (*u n*) (*v n*) + *deltaG*(*TYPE*(*'a*))|*Gromov-product-at y x* (*v n*) ≤ *Gromov-product-at y* (*u n*) (*v n*) + *deltaG*(*TYPE*(*'a*))
by *linarith*
then have *dist x y* ≤ *2* ∗ *Gromov-product-at y* (*u n*) (*v n*) + *2* ∗ *deltaG*(*TYPE*(*'a*)) + *max* (*dist x* (*u n*) − *dist y* (*u n*)) (*dist x* (*v n*) − *dist y* (*v n*))
unfolding *Gromov-product-at-def*[*of - x*] *Gromov-product-at-def*[*of - - x*] **apply** (*cases*)
by (*auto simp add*: *algebra-simps divide-simps dist-commute*)
then show *?thesis*
unfolding *ereal-max*[*symmetric*] *plus-ereal.simps*(*1*) **by** *auto*
qed
then have *ereal* (*dist x y*) ≤ *liminf* (λ*n*. *ereal*(*2* ∗ *Gromov-product-at y* (*u n*) (*v n*)) + *2* ∗ *deltaG*(*TYPE*(*'a*)) + *max* (*ereal*(*dist x* (*u n*) − *dist y* (*u n*))) (*ereal*(*dist x* (*v n*) − *dist y* (*v n*))))
by (*intro Liminf-bounded always-eventually*, *auto*)
also have ... ≤ *liminf* (λ*n*. *ereal*(*2* ∗ *Gromov-product-at y* (*u n*) (*v n*)) + *2* ∗ *deltaG*(*TYPE*(*'a*))) + *limsup* (λ*n*. *max* (*ereal*(*dist x* (*u n*) − *dist y* (*u n*))) (*ereal*(*dist x* (*v n*) − *dist y* (*v n*))))
by (*rule ereal-liminf-limsup-add*)
also have ... = *liminf* (λ*n*. *ereal*(*2* ∗ *Gromov-product-at y* (*u n*) (*v n*))) + *2* ∗ *deltaG*(*TYPE*(*'a*)) + *max* (*limsup* (λ*n*. *ereal*(*dist x* (*u n*) − *dist y* (*u n*)))) (*limsup* (λ*n*. *ereal*(*dist x* (*v n*) − *dist y* (*v n*))))
apply (*subst Liminf-add-ereal-right*) **by** (*auto simp add*: *Limsup-max-eq-max-Limsup*)
also have ... ≤ *liminf* (λ*n*. *ereal*(*2* ∗ *Gromov-product-at y* (*u n*) (*v n*))) + *2* ∗ *deltaG*(*TYPE*(*'a*)) + *max* (*ereal*(*Busemann-function-at xi x y*)) (*Busemann-function-at eta x y*)
unfolding *Busemann-function-ereal* **apply** (*intro mono-intros Sup-upper*) **using** *C* **by** *auto*
finally have *ereal*(*dist x y*) − *ereal*(*2* ∗ *deltaG*(*TYPE*(*'a*)) + *max* (*Busemann-function-at xi x y*) (*Busemann-function-at eta x y*)) ≤
liminf (λ*n*. *ereal*(*2* ∗ *Gromov-product-at y* (*u n*) (*v n*)))
unfolding *ereal-max*[*symmetric*] *add.assoc plus-ereal.simps*(*1*) **by** (*subst*

ereal-minus-le, *auto*)
then have *ereal*(*dist x y* − *2* ∗ *deltaG*(*TYPE*(′*a*)) − *max* (*Busemann-function-at xi x y*) (*Busemann-function-at eta x y*)) ≤
liminf (λ*n*. *ereal*(*2* ∗ *Gromov-product-at y* (*u n*) (*v n*)))
unfolding *ereal-minus*(*1*) **by** (*auto simp add*: *algebra-simps*)
also have ... = *ereal 2* ∗ *liminf* (λ*n*. *ereal*(*Gromov-product-at y* (*u n*) (*v n*)))
unfolding *times-ereal.simps*(*1*)[*symmetric*] **by** (*subst Liminf-ereal-mult-left*, *auto*)
finally show *?thesis*
by (*subst ereal-divide-le-pos*, *auto*)
qed
have *ereal*(*dist x y* − *2* ∗ *deltaG*(*TYPE*(′*a*)) − *max* (*Busemann-function-at xi x y*) (*Busemann-function-at eta x y*)) / *ereal 2* ≤
extended-Gromov-product-at y xi eta
unfolding *extended-Gromov-product-at-def* **apply** (*rule Inf-greatest*) **using** *A* **by** *auto*
also have ... = *ereal*(*real-of-ereal*(*extended-Gromov-product-at y xi eta*))
using *assms* **by** *simp*
finally show *?thesis*
by *simp*
qed

lemma *dist-minus-Busemann-max-ineq*:
dist (*x*::′*a*::*Gromov-hyperbolic-space*) *z* − *Busemann-function-at xi z x* ≤ *max* (*dist x y* − *Busemann-function-at xi y x*) (*dist y z* − *Busemann-function-at xi z y* − *2* ∗ *Busemann-function-at xi y x*) + *8* ∗ *deltaG*(*TYPE*(′*a*))
proof −
have *I*: *dist x z* − *Busemann-function-at* (*to-Gromov-completion t*) *z x* ≤ *max* (*dist x y* − *Busemann-function-at* (*to-Gromov-completion t*) *y x*)
(*dist y z* − *Busemann-function-at* (*to-Gromov-completion t*) *z y* − *2* ∗ *Busemann-function-at* (*to-Gromov-completion t*) *y x*)
\+ *2* ∗ *deltaG*(*TYPE*(′*a*)) **for** *t*
proof −
have *2* ∗ *dist x t* + − *max* (*dist x y* − *Busemann-function-at* (*to-Gromov-completion t*) *y x*) (*dist y z* − *Busemann-function-at* (*to-Gromov-completion t*) *z y* − *2* ∗ *Busemann-function-at* (*to-Gromov-completion t*) *y x*)
= *min* (*2* ∗ *dist x t* − (*dist x y* − *Busemann-function-at* (*to-Gromov-completion t*) *y x*)) (*2* ∗ *dist x t* − (*dist y z* − *Busemann-function-at* (*to-Gromov-completion t*) *z y* − *2* ∗ *Busemann-function-at* (*to-Gromov-completion t*) *y x*))
unfolding *minus-max-eq-min min-add-distrib-right* **by** *auto*
also have ... = *min* (*2* ∗ *Gromov-product-at t x y*) (*2* ∗ *Gromov-product-at t y z*)
apply (*rule cong*[*of min - min -*], *rule cong* [*of min min*])
unfolding *Gromov-product-at-def Busemann-function-inner* **by** (*auto simp add*: *algebra-simps dist-commute divide-simps*)
also have ... = *2* ∗ (*min* (*Gromov-product-at t x y*) (*Gromov-product-at t y z*))
by *auto*
also have ... ≤ *2* ∗ (*Gromov-product-at t x z* + *deltaG*(*TYPE*(′*a*)))
by (*intro mono-intros*, *auto*)

also have ... = *2 ∗ dist x t* − (*dist x z* − *Busemann-function-at* (*to-Gromov-completion t*) *z x*) + *2 ∗ deltaG*(*TYPE*(*'a*))
unfolding *Gromov-product-at-def Busemann-function-inner* **by** (*auto simp add*: *algebra-simps dist-commute divide-simps*)
finally show *?thesis*
by *auto*
qed
have *dist x z* − *Busemann-function-at xi z x* ≤ *max* (*dist x y* − *Busemann-function-at xi y x*) (*dist y z* − *Busemann-function-at xi z y* − *2 ∗ Busemann-function-at xi y x*) + *8 ∗ deltaG*(*TYPE*(*'a*)) + *d*
if *d* > *0* **for** *d*
proof −
define *e* **where** *e* = *d*/*4*
have *e* > *0* **unfolding** *e-def* **using** *that* **by** *auto*
obtain *t* **where** *t*: (*λn. to-Gromov-completion* (*t n*)) ⟶ *xi*
using *rep-Gromov-completion-limit* **by** *auto*
have *A*: *eventually* (*λn. Busemann-function-at xi y x* ≤ *Busemann-function-at* (*to-Gromov-completion* (*t n*)) *y x* + *2 ∗ deltaG*(*TYPE*(*'a*)) + *e*) *sequentially*
by (*rule eventually-mono*[*OF Busemann-function-inside-approx*[*OF* ‹*e* > *0*› *t*, *of y x*]], *auto*)
have *B*: *eventually* (*λn. Busemann-function-at xi z y* ≤ *Busemann-function-at* (*to-Gromov-completion* (*t n*)) *z y* + *2 ∗ deltaG*(*TYPE*(*'a*)) + *e*) *sequentially*
by (*rule eventually-mono*[*OF Busemann-function-inside-approx*[*OF* ‹*e* > *0*› *t*, *of z y*]], *auto*)
have *C*: *eventually* (*λn. Busemann-function-at xi z x* ≥ *Busemann-function-at* (*to-Gromov-completion* (*t n*)) *z x* − *e*) *sequentially*
by (*rule eventually-mono*[*OF Busemann-function-inside-approx*[*OF* ‹*e* > *0*› *t*, *of z x*]], *auto*)
obtain *n* **where** *H*: *Busemann-function-at xi y x* ≤ *Busemann-function-at* (*to-Gromov-completion* (*t n*)) *y x* + *2 ∗ deltaG*(*TYPE*(*'a*)) + *e*
Busemann-function-at xi z y ≤ *Busemann-function-at* (*to-Gromov-completion* (*t n*)) *z y* + *2 ∗ deltaG*(*TYPE*(*'a*)) + *e*
Busemann-function-at xi z x ≥ *Busemann-function-at* (*to-Gromov-completion* (*t n*)) *z x* − *e*
using *eventually-conj*[*OF A eventually-conj*[*OF B C*]] *eventually-sequentially* **by** *auto*
have *dist x z* − *Busemann-function-at xi z x* − *e* ≤ *dist x z* − *Busemann-function-at* (*to-Gromov-completion* (*t n*)) *z x*
using *H* **by** *auto*
also have ... ≤ *max* (*dist x y* − *Busemann-function-at* (*to-Gromov-completion* (*t n*)) *y x*)
(*dist y z* − *Busemann-function-at* (*to-Gromov-completion* (*t n*)) *z y* − *2 ∗ Busemann-function-at* (*to-Gromov-completion* (*t n*)) *y x*)
+ *2 ∗ deltaG*(*TYPE*(*'a*))
using *I* **by** *auto*
also have ... ≤ *max* (*dist x y* − (*Busemann-function-at xi y x* − *2 ∗ deltaG*(*TYPE*(*'a*)) − *e*))
(*dist y z* − (*Busemann-function-at xi z y* − *2 ∗ deltaG*(*TYPE*(*'a*)) − *e*) − *2 ∗* (*Busemann-function-at xi y x* − *2 ∗ deltaG*(*TYPE*(*'a*)) − *e*))

$+ 2 * deltaG(TYPE('a))$
apply (*intro mono-intros*) **using** *H* **by** *auto*
also have ... $\le$ *max* (*dist x y* $-$ *Busemann-function-at xi y x* + *6* $*$ *deltaG*(*TYPE*('*a*)) + *3* $*$ *e*)
(*dist y z* $-$ *Busemann-function-at xi z y* $-$ *2* $*$ *Busemann-function-at xi y x* + *6* $*$ *deltaG*(*TYPE*('*a*)) + *3* $*$ *e*)
+ *2* $*$ *deltaG*(*TYPE*('*a*))
apply (*intro add-mono max.mono*) **using** ‹*e* > *0*› **by** *auto*
also have ... = *max* (*dist x y* $-$ *Busemann-function-at xi y x*) (*dist y z* $-$ *Busemann-function-at xi z y* $-$ *2* $*$ *Busemann-function-at xi y x*) + *8* $*$ *deltaG*(*TYPE*('*a*)) + *3* $*$ *e*
by *auto*
finally show *?thesis* **unfolding** *e-def* **by** *auto*
qed
then show *?thesis* **by** (*rule field-le-epsilon*)
qed

end

19 Classification of isometries on a Gromov hyperbolic space

theory *Isometries-Classification*
imports *Gromov-Boundary Busemann-Function*
begin

Isometries of Gromov hyperbolic spaces are of three types:

- Elliptic ones, for which orbits are bounded.
- Parabolic ones, which are not elliptic and have exactly one fixed point at infinity.
- Loxodromic ones, which are not elliptic and have exactly two fixed points at infinity.

In this file, we show that all isometries are indeed of this form, and give further properties for each type.

For the definition, we use another characterization in terms of stable translation length: for isometries which are not elliptic, then they are parabolic if the stable translation length is 0, loxodromic if it is positive. This gives a very efficient definition, and it is clear from this definition that the three categories of isometries are disjoint. All the work is then to go from this general definition to the dynamical properties in terms of fixed points on the boundary.

19.1 The translation length

The translation length is the minimal translation distance of an isometry. The stable translation length is the limit of the translation length of f^n divided by n.

definition *translation-length*::(('*a*::*metric-space*) ⇒ '*a*) ⇒ *real*
  where *translation-length f* = *Inf* {*dist x* (*f x*)|*x*. *True*}

lemma *translation-length-nonneg* [*simp*, *mono-intros*]:
  translation-length f ≥ *0*
unfolding *translation-length-def* **by** (*rule cInf-greatest*, *auto*)

lemma *translation-length-le* [*mono-intros*]:
  translation-length f ≤ *dist x* (*f x*)
unfolding *translation-length-def* **apply** (*rule cInf-lower*) **by** (*auto intro*: *bdd-belowI*[*of* - *0*])

definition *stable-translation-length*::(('*a*::*metric-space*) ⇒ '*a*) ⇒ *real*
  where *stable-translation-length f* = *Inf* {*translation-length* (*f*^^*n*)/*n* |*n*. *n* > *0*}

lemma *stable-translation-length-nonneg* [*simp*]:
  stable-translation-length f ≥ *0*
unfolding *stable-translation-length-def* **by** (*rule cInf-greatest*, *auto*)

lemma *stable-translation-length-le-translation-length* [*mono-intros*]:
  n ∗ *stable-translation-length f* ≤ *translation-length* (*f*^^*n*)
proof −
  have ∗: *stable-translation-length f* ≤ *translation-length* (*f*^^*n*)/*n* **if** *n* > *0* **for** *n*
    unfolding *stable-translation-length-def* **apply** (*rule cInf-lower*) **using** *that* **by** (*auto intro*: *bdd-belowI*[*of* - *0*])
  show *?thesis*
    apply (*cases n* = *0*) **using** ∗ **by** (*auto simp add*: *divide-simps algebra-simps*)
qed

lemma *semicontraction-iterates*:
  fixes *f*::('*a*::*metric-space*) ⇒ '*a*
  assumes *1*−*lipschitz-on UNIV f*
  shows *1*−*lipschitz-on UNIV* (*f*^^*n*)
by (*induction n*, *auto intro*!: *lipschitz-onI lipschitz-on-compose2*[*of 1 UNIV* - *1 f*, *simplified*] *lipschitz-on-subset*[*OF assms*])

If f is a semicontraction, then its stable translation length is the limit of $d(x, f^n x)/n$ for any n. While it is obvious that the liminf of this quantity is at least the stable translation length (which is defined as an inf over all points and all times), the opposite inequality is more interesting. One may find a point y and a time k for which $d(y, f^k y)/k$ is very close to the stable translation length. By subadditivity of the sequence $n \mapsto f(y, f^n y)$ and Fekete's Lemma, it follows that, for any large n, then $d(y, f^n y)/n$ is also very close to the stable translation length. Since this is equal to $d(x, f^n x)/n$ up to $\pm 2d(x,y)/n$, the result follows.

proposition *stable-translation-length-as-pointwise-limit*:
assumes *1−lipschitz-on UNIV f*
shows (*λn. dist x* ((*f^^n*) *x*)/*n*) ⟶ *stable-translation-length f*
proof −
have ∗: *subadditive* (*λn. dist y* ((*f^^n*) *y*)) **for** *y*
proof (*rule subadditiveI*)
fix *m n*::*nat*
have *dist y* ((*f* ^^ (*m* + *n*)) *y*) ≤ *dist y* ((*f^^m*) *y*) + *dist* ((*f^^m*) *y*) ((*f^^*(*m*+*n*)) *y*)
by (*rule dist-triangle*)
also have ... = *dist y* ((*f^^m*) *y*) + *dist* ((*f^^m*) *y*) ((*f^^m*) ((*f^^n*) *y*))
by (*auto simp add*: *funpow-add*)
also have ... ≤ *dist y* ((*f^^m*) *y*) + *dist y* ((*f^^n*) *y*)
using *semicontraction-iterates*[*OF assms*, *of m*] **unfolding** *lipschitz-on-def* **by** *auto*
finally show *dist y* ((*f* ^^ (*m* + *n*)) *y*) ≤ *dist y* ((*f* ^^ *m*) *y*) + *dist y* ((*f* ^^ *n*) *y*)
by *simp*
qed
have *Ly*: (*λn. dist y* ((*f^^n*) *y*) / *n*) ⟶ *Inf* {*dist y* ((*f^^n*) *y*) / *n* |*n. n* > *0*} **for** *y*
by (*auto intro*!: *bdd-belowI*[*of* - *0*] *subadditive-converges-bounded'*[*OF subadditive-imp-eventually-subadditive*[*OF* ∗]])
have *eventually* (*λn. dist x* ((*f^^n*) *x*)/*n* < *l*) *sequentially* **if** *stable-translation-length f* < *l* **for** *l*
proof −
obtain *m* **where** *m*: *stable-translation-length f* < *m m* < *l*
using ‹*stable-translation-length f* < *l*› *dense* **by** *auto*
have ∃ *t* ∈ {*translation-length* (*f^^n*)/*n* |*n. n* > *0*}. *t* < *m*
apply (*subst cInf-less-iff*[*symmetric*])
using *m* **unfolding** *stable-translation-length-def* **by** (*auto intro*!: *bdd-belowI*[*of* - *0*])
then obtain *k* **where** *k*: *k* > *0 translation-length* (*f^^k*)/*k* < *m*
by *auto*
have *translation-length* (*f^^k*) < *k* ∗ *m*
using *k* **by** (*simp add*: *divide-simps algebra-simps*)
then have ∃ *t* ∈ {*dist y* ((*f^^k*) *y*) |*y. True*}. *t* < *k* ∗ *m*
apply (*subst cInf-less-iff*[*symmetric*])
unfolding *translation-length-def* **by** (*auto intro*!: *bdd-belowI*[*of* - *0*])
then obtain *y* **where** *y*: *dist y* ((*f^^k*) *y*) < *k* ∗ *m*
by *auto*
have *A*: *eventually* (*λn. dist y* ((*f^^n*) *y*)/*n* < *m*) *sequentially*
apply (*auto intro*!: *order-tendstoD*[*OF Ly*] *iffD2*[*OF cInf-less-iff*] *bdd-belowI*[*of* - *0*] *exI*[*of* - *dist y* ((*f^^k*) *y*)/*k*])
using *y k* **by** (*auto simp add*: *algebra-simps divide-simps*)
have *B*: *eventually* (*λn. dist x y* ∗ (*1*/*n*) < (*l*−*m*)/*2*) *sequentially*
apply (*intro order-tendstoD*[*of* - *dist x y* ∗ *0*] *tendsto-intros*)

using ‹*m* < *l*› **by** *simp*
have *C*: *dist x* ((*f*^^*n*) *x*)/*n* < *l* **if** *n* > *0* *dist y* ((*f*^^*n*) *y*)/*n* < *m* *dist x y* ∗ (*1*/*n*) < (*l*−*m*)/*2* **for** *n*
proof −
have *dist x* ((*f*^^*n*) *x*) ≤ *dist x y* + *dist y* ((*f*^^*n*) *y*) + *dist* ((*f*^^*n*) *y*) ((*f*^^*n*) *x*)
by (*intro mono-intros*)
also have ... ≤ *dist x y* + *dist y* ((*f*^^*n*) *y*) + *dist y x*
using *semicontraction-iterates*[*OF assms*, *of n*] **unfolding** *lipschitz-on-def* **by** *auto*
also have ... = *2* ∗ *dist x y* + *dist y* ((*f*^^*n*) *y*)
by (*simp add*: *dist-commute*)
also have ... < *2* ∗ *real n* ∗ (*l*−*m*)/*2* + *n* ∗ *m*
apply (*intro mono-intros*) **using** *that* **by** (*auto simp add*: *algebra-simps divide-simps*)
also have ... = *n* ∗ *l*
by (*simp add*: *algebra-simps divide-simps*)
finally show *?thesis*
using *that* **by** (*simp add*: *algebra-simps divide-simps*)
qed
show *eventually* (λ*n*. *dist x* ((*f*^^*n*) *x*)/*n* < *l*) *sequentially*
by (*rule eventually-mono*[*OF eventually-conj*[*OF eventually-conj*[*OF A B*] *eventually-gt-at-top*[*of 0*]] *C*], *auto*)
qed
moreover have *eventually* (λ*n*. *dist x* ((*f*^^*n*) *x*)/*n* > *l*) *sequentially* **if** *stable-translation-length f* > *l* **for** *l*
proof −
have ∗: *dist x* ((*f*^^*n*) *x*)/*n* > *l* **if** *n* > *0* **for** *n*
proof −
have *n* ∗ *l* < *n* ∗ *stable-translation-length f*
using ‹*stable-translation-length f* > *l*› ‹*n* > *0*› **by** *auto*
also have ... ≤ *translation-length* (*f*^^*n*)
by (*intro mono-intros*)
also have ... ≤ *dist x* ((*f*^^*n*) *x*)
by (*intro mono-intros*)
finally show *?thesis*
using ‹*n* > *0*› **by** (*auto simp add*: *algebra-simps divide-simps*)
qed
then show *?thesis*
by (*rule eventually-mono*[*rotated*], *auto*)
qed
ultimately show *?thesis*
by (*rule order-tendstoI*[*rotated*])
qed

It follows from the previous proposition that the stable translation length is also the limit of the renormalized translation length of f^n.

proposition *stable-translation-length-as-limit*:
assumes *1*−*lipschitz-on UNIV f*

shows ($\lambda n.$ *translation-length* (f^^n) / n) $\longlongrightarrow$ *stable-translation-length f*
proof –
obtain x::$'a$ **where** *True* **by** *auto*
show *?thesis*
proof (*rule tendsto-sandwich*[*of* $\lambda n.$ *stable-translation-length f* - - $\lambda n.$ *dist x* ((f^^n) x)/n])
have *stable-translation-length f* $\leq$ *translation-length* (f ^^ n) / *real n* **if** $n > 0$ **for** n
using *stable-translation-length-le-translation-length*[*of n f*] *that* **by** (*simp add*: *divide-simps algebra-simps*)
then show *eventually* ($\lambda n.$ *stable-translation-length f* $\leq$ *translation-length* (f ^^ n) / *real n*) *sequentially*
by (*rule eventually-mono*[*rotated*], *auto*)
have *translation-length* (f ^^ n) / *real n* $\leq$ *dist x* ((f ^^ n) x) / *real n* **for** n
using *translation-length-le*[*of f*^^*n x*] **by** (*auto simp add*: *divide-simps*)
then show *eventually* ($\lambda n.$ *translation-length* (f ^^ n) / *real n* $\leq$ *dist x* ((f ^^ n) x) / *real n*) *sequentially*
by *auto*
qed (*auto simp add*: *stable-translation-length-as-pointwise-limit*[*OF assms*])
qed

lemma *stable-translation-length-inv*:
assumes *isometry f*
shows *stable-translation-length* (*inv f*) = *stable-translation-length f*
proof –
have *: *dist basepoint* (((*inv f*)^^n) *basepoint*) = *dist basepoint* ((f^^n) *basepoint*) **for** n
proof –
have *basepoint* = (f^^n) (((*inv f*)^^n) *basepoint*)
by (*metis assms comp-apply fn-o-inv-fn-is-id isometry-inverse*(*2*))
then have *dist basepoint* ((f^^n) *basepoint*) = *dist* ((f^^n) (((*inv f*)^^n) *basepoint*)) ((f^^n) *basepoint*)
by *auto*
also have ... = *dist* (((*inv f*)^^n) *basepoint*) *basepoint*
unfolding *isometryD*(*2*)[*OF isometry-iterates*[*OF assms*]] **by** *simp*
finally show *?thesis* **by** (*simp add*: *dist-commute*)
qed

have ($\lambda n.$ *dist basepoint* ((f^^n) *basepoint*)/n) $\longlongrightarrow$ *stable-translation-length f*
using *stable-translation-length-as-pointwise-limit*[*OF isometryD*(*4*)[*OF assms*]] **by** *simp*
moreover have ($\lambda n.$ *dist basepoint* ((f^^n) *basepoint*)/n) $\longlongrightarrow$ *stable-translation-length* (*inv f*)
unfolding *[*symmetric*]
using *stable-translation-length-as-pointwise-limit*[*OF isometryD*(*4*)[*OF isometry-inverse*(*1*)[*OF assms*]]] **by** *simp*
ultimately show *?thesis*
using *LIMSEQ-unique* **by** *auto*
qed

19.2 The strength of an isometry at a fixed point at infinity

The additive strength of an isometry at a fixed point at infinity is the asymptotic average every point is moved towards the fixed point at each step. It is measured using the Busemann function.

definition *additive-strength*::(*'a*::*Gromov-hyperbolic-space* ⇒ *'a*) ⇒ (*'a Gromov-completion*) ⇒ *real*
  where *additive-strength f xi* = *lim* (λ*n*. (*Busemann-function-at xi* ((*f*^^*n*) *basepoint*) *basepoint*)/*n*)

For additivity reasons, as the Busemann function is a quasi-morphism, the additive strength measures the deplacement even at finite times. It is also uniform in terms of the basepoint. This shows that an isometry sends horoballs centered at a fixed point to horoballs, up to a uniformly bounded error depending only on δ.

lemma *Busemann-function-eq-additive-strength*:
  assumes *isometry f Gromov-extension f xi* = *xi*
  shows |*Busemann-function-at xi* ((*f*^^*n*) *x*) (*x*::*'a*::*Gromov-hyperbolic-space*) − *real n* ∗ *additive-strength f xi*| ≤ *2* ∗ *deltaG*(*TYPE*(*'a*))
proof −
  define *u* **where** *u* = (λ*y n*. *Busemann-function-at xi* ((*f*^^*n*) *y*) *y*)
  have ∗: *abs*(*u y* (*m*+*n*) − *u y m* − *u y n*) ≤ *2* ∗ *deltaG*(*TYPE*(*'a*)) **for** *n m y*
  proof −
    have *P*: *Gromov-extension* (*f*^^*m*) *xi* = *xi*
    unfolding *Gromov-extension-isometry-iterates*[*OF assms*(*1*)] **apply** (*induction m*) **using** *assms* **by** *auto*
    have ∗: *u y n* = *Busemann-function-at xi* ((*f*^^*m*) ((*f*^^*n*) *y*)) ((*f*^^*m*) *y*)
    apply (*subst P*[*symmetric*]) **unfolding** *Busemann-function-isometry*[*OF isometry-iterates*[*OF* ‹*isometry f*›]] *u-def* **by** *auto*
    show *?thesis*
      unfolding ∗ **unfolding** *u-def* **using** *Busemann-function-quasi-morphism*[*of xi* (*f*^^(*m*+*n*)) *y* (*f*^^*m*) *y y*]
      unfolding *funpow-add comp-def* **by** *auto*
  qed
  define *l* **where** *l* = (λ*y*. *lim* (λ*n*. *u y n*/*n*))
  have *A*: *abs*(*u y k* − *k* ∗ *l y*) ≤ *2* ∗ *deltaG*(*TYPE*(*'a*)) **for** *y k*
    unfolding *l-def* **using** *almost-additive-converges*(*2*) ∗ **by** *auto*
  then have ∗: *abs*(*Busemann-function-at xi* ((*f*^^*k*) *y*) *y* − *k* ∗ *l y*) ≤ *2* ∗ *deltaG*(*TYPE*(*'a*)) **for** *y k*
    unfolding *u-def* **by** *auto*
  have *l basepoint* = *additive-strength f xi*
    unfolding *l-def u-def additive-strength-def* **by** *auto*

  have *abs*(*k* ∗ *l basepoint* − *k* ∗ *l x*) ≤ *4* ∗ *deltaG*(*TYPE*(*'a*)) + *2* ∗ *dist basepoint x* **for** *k*::*nat*
  proof −
    have *abs*(*k* ∗ *l basepoint* − *k* ∗ *l x*) = *abs*((*Busemann-function-at xi* ((*f*^^*k*) *x*) *x* − *k* ∗ *l x*) − (*Busemann-function-at xi* ((*f*^^*k*) *basepoint*) *basepoint* − *k* ∗ *l*

basepoint)

+ (*Busemann-function-at xi* ((*f*^^*k*) *basepoint*) *basepoint* − *Busemann-function-at xi* ((*f*^^*k*) *x*) *x*))

by *auto*

also have ... ≤ *abs* (*Busemann-function-at xi* ((*f*^^*k*) *x*) *x* − *k* ∗ *l x*) + *abs* (*Busemann-function-at xi* ((*f*^^*k*) *basepoint*) *basepoint* − *k* ∗ *l basepoint*)

+ *abs* (*Busemann-function-at xi* ((*f*^^*k*) *basepoint*) *basepoint* − *Busemann-function-at xi* ((*f*^^*k*) *x*) *x*)

by *auto*

also have ... ≤ *2* ∗ *deltaG*(*TYPE*(′*a*)) + *2* ∗ *deltaG*(*TYPE*(′*a*)) + (*dist* ((*f*^^*k*) *basepoint*) ((*f*^^*k*) *x*) + *dist basepoint x*)

by (*intro mono-intros* ∗)

also have ... = *4* ∗ *deltaG*(*TYPE*(′*a*)) + *2* ∗ *dist basepoint x*

unfolding *isometryD*[*OF isometry-iterates*[*OF assms*(*1*)]] **by** *auto*

finally show *?thesis* **by** *auto*

qed

moreover have *u* = *v* **if** *H*: ⋀*k*::*nat*. *abs*(*k* ∗ *u* − *k* ∗ *v*) ≤ *C* **for** *u v C*::*real*

proof −

have (λ*n*. *abs*(*u* − *v*)) ⟶ *0*

proof (*rule tendsto-sandwich*[*of* λ*n*. *0* - - λ*n*::*nat*. *C*/*n*], *auto*)

have (λ*n*. *C*∗(*1*/*n*)) ⟶ *C* ∗ *0* **by** (*intro tendsto-intros*)

then show (λ*n*. *C*/*n*) ⟶ *0* **by** *auto*

have |*u* − *v*| ≤ *C* / *real n* **if** *n* ≥ *1* **for** *n*

using *H*[*of n*] *that* **apply** (*simp add*: *divide-simps algebra-simps*)

by (*metis H abs-mult abs-of-nat right-diff-distrib*′)

then show ∀$_F$ *n in sequentially*. |*u* − *v*| ≤ *C* / *real n*

unfolding *eventually-sequentially* **by** *auto*

qed

then show *?thesis*

by (*metis LIMSEQ-const-iff abs-0-eq eq-iff-diff-eq-0*)

qed

ultimately have *l basepoint* = *l x* **by** *auto*

show *?thesis*

using *A*[*of x n*] **unfolding** *u-def* ‹*l basepoint* = *l x*›[*symmetric*] ‹*l basepoint* = *additive-strength f xi*› **by** *auto*

qed

lemma *additive-strength-as-limit* [*tendsto-intros*]:

assumes *isometry f Gromov-extension f xi* = *xi*

shows (λ*n*. *Busemann-function-at xi* ((*f*^^*n*) *x*) *x*/*n*) ⟶ *additive-strength f xi*

proof −

have (λ*n*. *abs*(*Busemann-function-at xi* ((*f*^^*n*) *x*) *x*/*n* − *additive-strength f xi*)) ⟶ *0*

apply (*rule tendsto-sandwich*[*of* λ*n*. *0* - - λ*n*. (*2* ∗ *deltaG*(*TYPE*(′*a*))) ∗ (*1*/*real n*)], *auto*)

unfolding *eventually-sequentially* **apply** (*rule exI*[*of* - *1*])

using *Busemann-function-eq-additive-strength*[*OF assms*] **apply** (*simp add*: *divide-simps algebra-simps*)

using *tendsto-mult*[*OF - lim-1-over-n*] **by** *auto*
then show *?thesis*
using *LIM-zero-iff tendsto-rabs-zero-cancel* **by** *blast*
qed

The additive strength measures the amount of displacement towards a fixed point at infinity. Therefore, the distance from x to $f^n x$ is at least n times the additive strength, but one might think that it might be larger, if there is displacement along the horospheres. It turns out that this is not the case: the displacement along the horospheres is at most logarithmic (this is a classical property of parabolic isometries in hyperbolic spaces), and in fact it is bounded for loxodromic elements. We prove here that the growth is at most logarithmic in all cases, using a small computation based on the hyperbolicity inequality, expressed in Lemma `dist_minus_Busemann_max_ineq` above. This lemma will be used below to show that the translation length is the absolute value of the additive strength.

lemma *dist-le-additive-strength*:
assumes *isometry* $(f::'a::Gromov\text{-}hyperbolic\text{-}space \Rightarrow 'a)$ *Gromov-extension f xi = xi additive-strength f xi* $\geq$ *0 n* $\geq$ *1*
shows *dist x* $((f \frown\frown n)\ x) \leq$ *dist x* $(f\ x)$ + *real n* $*$ *additive-strength f xi* + *ceiling* $(log\ 2\ n) * 16 *$ *deltaG*$(TYPE('a))$
proof −
have *A*: $\bigwedge n.\ n \in \{1..2\hat{}k\} \Longrightarrow$ *dist x* $((f \frown\frown n)\ x) -$ *real n* $*$ *additive-strength f xi* $\leq$ *dist x* $(f\ x) + k * 16 *$ *deltaG*$(TYPE('a))$ **for** *k*
proof (*induction k*)
case *0*
fix *n*::*nat* **assume** $n \in \{1..2\hat{}0\}$
then have $n = 1$ **by** *auto*
then show *dist x* $((f \frown\frown n)\ x) -$ *real n* $*$ *additive-strength f xi* $\leq$ *dist x* $(f\ x)$ + *real 0* $* 16 *$ *deltaG*$(TYPE('a))$
using *assms*(*3*) **by** *auto*
next
case (*Suc k*)
fix *N*::*nat* **assume** $N \in \{1..2\hat{}(Suc\ k)\}$
then consider $N \in \{1..2\hat{}k\} \mid N \in \{2\hat{}k<..2\hat{}(Suc\ k)\}$ **using** *not-le* **by** *auto*
then show *dist x* $((f \frown\frown N)\ x) -$ *real N* $*$ *additive-strength f xi* $\leq$ *dist x* $(f\ x)$ + *real* $(Suc\ k) * 16 *$ *deltaG TYPE*$('a)$
proof (*cases*)
case *1*
show *?thesis* **by** (*rule order-trans*[*OF Suc.IH*[*OF 1*]], *auto simp add*: *algebra-simps*)
next
case *2*
define *m*::*nat* **where** $m = N - 2\hat{}k$
define *n*::*nat* **where** $n = 2\hat{}k$
have *nm*: $N = n+m\ m \in \{1..2\hat{}k\}\ n \in \{1..2\hat{}k\}$**unfolding** *m-def n-def* **using** *2* **by** *auto*
have $*$: $(f \frown\frown (n+m))\ x = (f \frown\frown n)\ ((f \frown\frown m)\ x)$

unfolding *funpow-add comp-def* **by** *auto*
have ∗∗: $(f^{\frown\frown}(n+m))\ x = (f^{\frown\frown}m)\ ((f^{\frown\frown}n)\ x)$
apply (*subst add.commute*) **unfolding** *funpow-add comp-def* **by** *auto*

have *dist* $x\ ((f^{\frown\frown}N)\ x) - N *$ *additive-strength f xi* $- 2 *$ *deltaG*(*TYPE*('*a*)) $\leq$ *dist* $x\ ((f^{\frown\frown}(n+m))\ x) -$ *Busemann-function-at xi* $((f^{\frown\frown}(n+m))\ x)\ x$
unfolding *nm*(*1*) **using** *Busemann-function-eq-additive-strength*[*OF assms*(*1*) *assms*(*2*), *of n*+*m x*] **by** *auto*
also have ... $\leq$ *max* (*dist* $x\ ((f^{\frown\frown}n)\ x) -$ *Busemann-function-at xi* $((f^{\frown\frown}n)\ x)\ x)$ (*dist* $((f^{\frown\frown}n)\ x)\ ((f^{\frown\frown}(n+m))\ x) -$ *Busemann-function-at xi* $((f^{\frown\frown}(n+m))\ x)\ ((f^{\frown\frown}n)\ x) - 2 *$ *Busemann-function-at xi* $((f^{\frown\frown}n)\ x)\ x) + 8 *$ *deltaG*(*TYPE*('*a*))
using *dist-minus-Busemann-max-ineq* **by** *auto*
also have ... $\leq$ *max* (*dist* $x\ ((f^{\frown\frown}n)\ x) - (n *$ *additive-strength f xi* $- 2 *$ *deltaG*(*TYPE*('*a*)))) (*dist* $((f^{\frown\frown}n)\ x)\ ((f^{\frown\frown}(n+m))\ x) - (m *$ *additive-strength f xi* $- 2 *$ *deltaG*(*TYPE*('*a*))) $- 2 * (n *$ *additive-strength f xi* $- 2 *$ *deltaG*(*TYPE*('*a*)))) $+ 8 *$ *deltaG*(*TYPE*('*a*))
unfolding ∗∗ **apply** (*intro mono-intros*)
using *Busemann-function-eq-additive-strength*[*OF assms*(*1*) *assms*(*2*), *of n x*] *Busemann-function-eq-additive-strength*[*OF assms*(*1*) *assms*(*2*), *of m* $(f^{\frown\frown}n)\ x$] **by** *auto*
also have ... $\leq$ *max* (*dist* $x\ ((f^{\frown\frown}n)\ x) - n *$ *additive-strength f xi* $+ 6 *$ *deltaG*(*TYPE*('*a*))) (*dist* $x\ ((f^{\frown\frown}m)\ x) - m *$ *additive-strength f xi* $+ 6 *$ *deltaG*(*TYPE*('*a*))) $+ 8 *$ *deltaG*(*TYPE*('*a*))
unfolding ∗ *isometryD*(*2*)[*OF isometry-iterates*[*OF assms*(*1*)], *of n*] **using** *assms*(*3*) **by** (*intro mono-intros*, *auto*)
also have ... = *max* (*dist* $x\ ((f^{\frown\frown}n)\ x) - n *$ *additive-strength f xi*) (*dist* $x\ ((f^{\frown\frown}m)\ x) - m *$ *additive-strength f xi*) $+ 14 *$ *deltaG*(*TYPE*('*a*))
unfolding *max-add-distrib-left*[*symmetric*] **by** *auto*
also have ... $\leq$ *dist* $x\ (f\ x) + k * 16 *$ *deltaG*(*TYPE*('*a*)) $+ 14 *$ *deltaG*(*TYPE*('*a*))
using *nm* **by** (*auto intro*!: *Suc.IH*)
finally show *?thesis* **by** (*auto simp add*: *algebra-simps*)
qed
qed
define *k*::*nat* **where** $k =$ *nat*(*ceiling* (*log* 2 *n*))
have $n \leq 2^k$ **unfolding** *k-def*
by (*meson less-log2-of-power not-le real-nat-ceiling-ge*)
then have *dist* $x\ ((f^{\frown\frown}n)\ x) -$ *real* $n *$ *additive-strength f xi* $\leq$ *dist* $x\ (f\ x) + k * 16 *$ *deltaG*(*TYPE*('*a*))
using *A*[*of n k*] ‹$n \geq 1$› **by** *auto*
moreover have *real* (*nat* ⌈*log* 2 (*real n*)⌉) = *real-of-int* ⌈*log* 2 (*real n*)⌉
by (*metis Transcendental.log-one* ‹$n \leq 2\ \hat{}\ k$› *assms*(*4*) *ceiling-zero int-ops*(*2*) *k-def le-antisym nat-eq-iff2 of-int-1 of-int-of-nat-eq order-refl power-0*)
ultimately show *?thesis* **unfolding** *k-def* **by** (*auto simp add*: *algebra-simps*)
qed

The strength of the inverse of a map is the opposite of the strength of the map.

lemma *additive-strength-inv*:
assumes *isometry* (*f*::'*a*::*Gromov-hyperbolic-space* ⇒ '*a*) *Gromov-extension f xi*

= *xi*
shows *additive-strength* (*inv f*) *xi* = − *additive-strength f xi*
proof −
have ∗: (*inv f* ^^ *n*) ((*f* ^^ *n*) *x*) = *x* **for** *n x*
by (*metis assms*(*1*) *comp-apply inv-fn-o-fn-is-id isometry-inverse*(*2*))
have *A*: *abs*(*real n* ∗ *additive-strength f xi* + *real n* ∗ *additive-strength* (*inv f*) *xi*) ≤ *6* ∗ *deltaG* (*TYPE*(′*a*)) **for** *n*::*nat* **and** *x*::′*a*
using *Busemann-function-quasi-morphism*[*of xi x* (*f*^^*n*) *x x*] *Busemann-function-eq-additive-strength*[*OF assms*, *of n x*] *Busemann-function-eq-additive-strength*[*OF isometry-inverse*(*1*)[*OF assms*(*1*)]
Gromov-extension-inv-fixed-point[*OF assms*], *of n* (*f*^^*n*) *x*] **unfolding** ∗ **by** *auto*
have *B*: *abs*(*additive-strength f xi* + *additive-strength* (*inv f*) *xi*) ≤ *6* ∗ *deltaG*(*TYPE*(′*a*)) ∗ (*1*/*n*) **if** *n* ≥ *1* **for** *n*::*nat*
using *that A*[*of n*] **apply** (*simp add*: *divide-simps algebra-simps*) **unfolding** *distrib-left*[*symmetric*] **by** *auto*
have (λ*n*. *abs*(*additive-strength f xi* + *additive-strength* (*inv f*) *xi*)) ⟶ *6* ∗ *deltaG*(*TYPE*(′*a*)) ∗ *0*
apply (*rule tendsto-sandwich*[*of* λ*n*. *0* - - λ*n*. *6* ∗ *deltaG*(*TYPE*(′*a*)) ∗ (*1*/*real n*)], *simp*)
unfolding *eventually-sequentially* **apply** (*rule exI*[*of* - *1*]) **using** *B* **apply** *simp*
by (*simp*, *intro tendsto-intros*)
then show *?thesis*
using *LIMSEQ-unique mult-zero-right tendsto-const* **by** *fastforce*
qed

We will now prove that the stable translation length of an isometry is given by the absolute value of its strength at any fixed point. We start with the case where the strength is nonnegative, and then reduce to this case by considering the map or its inverse.

lemma *stable-translation-length-eq-additive-strength-aux*:
assumes *isometry* (*f*::′*a*::*Gromov-hyperbolic-space* ⇒ ′*a*) *Gromov-extension f xi* = *xi additive-strength f xi* ≥ *0*
shows *stable-translation-length f* = *additive-strength f xi*
proof −
have (λ*n*. *dist x* ((*f*^^*n*) *x*)/*n*) ⟶ *additive-strength f xi* **for** *x*
proof (*rule tendsto-sandwich*[*of* λ*n*. (*n* ∗ *additive-strength f xi* − *2* ∗ *deltaG*(*TYPE*(′*a*)))/*real n* - - λ*n*. (*dist x* (*f x*) + *n* ∗ *additive-strength f xi* + *ceiling* (*log 2 n*) ∗ *16* ∗ *deltaG*(*TYPE*(′*a*)))/ *n*])
have *n* ∗ *additive-strength f xi* − *2* ∗ *deltaG TYPE*(′*a*) ≤ *dist x* ((*f* ^^ *n*) *x*) **for** *n*
using *Busemann-function-eq-additive-strength*[*OF assms*(*1*) *assms*(*2*), *of n x*] *Busemann-function-le-dist*[*of xi* (*f*^^*n*) *x x*]
by (*simp add*: *dist-commute*)
then have (*n* ∗ *additive-strength f xi* − *2* ∗ *deltaG TYPE*(′*a*)) / *n* ≤ *dist x* ((*f* ^^ *n*) *x*) / *n* **if** *n* ≥ *1* **for** *n*
using *that* **by** (*simp add*: *divide-simps*)
then show $\forall_F$ *n in sequentially*. (*real n* ∗ *additive-strength f xi* − *2* ∗ *deltaG TYPE*(′*a*)) / *real n* ≤ *dist x* ((*f* ^^ *n*) *x*) / *real n*

unfolding *eventually-sequentially* **by** *auto*

have *B*: $(\lambda n.\ additive\text{-}strength\ f\ xi - (2 * deltaG(TYPE('a))) * (1/n)) \longrightarrow additive\text{-}strength\ f\ xi - (2 * deltaG(TYPE('a))) * 0$
by (*intro tendsto-intros*)
show $(\lambda n.\ (real\ n * additive\text{-}strength\ f\ xi - 2 * deltaG\ TYPE('a))\ /\ real\ n) \longrightarrow additive\text{-}strength\ f\ xi$
proof (*rule Lim-transform-eventually*)
show *eventually* $(\lambda n.\ additive\text{-}strength\ f\ xi - (2 * deltaG(TYPE('a))) * (1/n) = (real\ n * additive\text{-}strength\ f\ xi - 2 * deltaG\ TYPE('a))\ /\ real\ n)$ *sequentially*
unfolding *eventually-sequentially* **apply** (*rule exI*[*of - 1*]) **by** (*simp add*: *divide-simps*)
qed (*use B* **in** *auto*)

have $dist\ x\ ((f\widehat{\ }\widehat{\ }n)\ x) \leq dist\ x\ (f\ x) + n * additive\text{-}strength\ f\ xi + ceiling\ (log\ 2\ n) * 16 * deltaG(TYPE('a))$ **if** $n \geq 1$ **for** *n*
using *dist-le-additive-strength*[*OF assms that*] **by** *simp*
then have $(dist\ x\ ((f\widehat{\ }\widehat{\ }n)\ x))/n \leq (dist\ x\ (f\ x) + n * additive\text{-}strength\ f\ xi + ceiling\ (log\ 2\ n) * 16 * deltaG(TYPE('a)))/n$ **if** $n \geq 1$ **for** *n*
using *that* **by** (*simp add*: *divide-simps*)
then show $\forall_F\ n\ in\ sequentially.\ dist\ x\ ((f\ \widehat{\ }\widehat{\ }\ n)\ x)\ /\ real\ n \leq (dist\ x\ (f\ x) + real\ n * additive\text{-}strength\ f\ xi + real\text{-}of\text{-}int\ (\lceil log\ 2\ (real\ n)\rceil * 16) * deltaG\ TYPE('a))\ /\ real\ n$
unfolding *eventually-sequentially* **by** *auto*

have *B*: $(\lambda n.\ dist\ x\ (f\ x) * (1/n) + additive\text{-}strength\ f\ xi + 16 * deltaG\ TYPE('a) * (\lceil log\ 2\ n\rceil\ /\ n)) \longrightarrow dist\ x\ (f\ x) * 0 + additive\text{-}strength\ f\ xi + 16 * deltaG\ TYPE('a) * 0$
by (*intro tendsto-intros*)
show $(\lambda n.\ (dist\ x\ (f\ x) + n * additive\text{-}strength\ f\ xi + real\text{-}of\text{-}int\ (\lceil log\ 2\ n\rceil * 16) * deltaG\ TYPE('a))\ /\ real\ n) \longrightarrow additive\text{-}strength\ f\ xi$
proof (*rule Lim-transform-eventually*)
show *eventually* $(\lambda n.\ dist\ x\ (f\ x) * (1/n) + additive\text{-}strength\ f\ xi + 16 * deltaG\ TYPE('a) * (\lceil log\ 2\ n\rceil\ /\ n) = (dist\ x\ (f\ x) + real\ n * additive\text{-}strength\ f\ xi + real\text{-}of\text{-}int\ (\lceil log\ 2\ (real\ n)\rceil * 16) * deltaG\ TYPE('a))\ /\ real\ n)$ *sequentially*
unfolding *eventually-sequentially* **apply** (*rule exI*[*of - 1*]) **by** (*simp add*: *algebra-simps divide-simps*)
qed (*use B* **in** *auto*)
qed
then show *?thesis*
using *LIMSEQ-unique stable-translation-length-as-pointwise-limit*[*OF isometryD(4)*[*OF assms(1)*]] **by** *blast*
qed

lemma *stable-translation-length-eq-additive-strength*:
assumes *isometry* $(f::'a::Gromov\text{-}hyperbolic\text{-}space \Rightarrow 'a)$ *Gromov-extension f xi = xi*
shows *stable-translation-length f* = *abs*(*additive-strength f xi*)
proof (*cases additive-strength f xi* ≥ 0)

case *True*
then show *?thesis* **using** *stable-translation-length-eq-additive-strength-aux*[*OF assms*] **by** *auto*
next
case *False*
then have ∗: *abs*(*additive-strength f xi*) = *additive-strength* (*inv f*) *xi*
unfolding *additive-strength-inv*[*OF assms*] **by** *auto*
show *?thesis*
unfolding ∗ *stable-translation-length-inv*[*OF assms*(*1*), *symmetric*]
using *stable-translation-length-eq-additive-strength-aux*[*OF isometry-inverse*(*1*)[*OF assms*(*1*)] *Gromov-extension-inv-fixed-point*[*OF assms*]] ∗ **by** *auto*
qed

19.3 Elliptic isometries

Elliptic isometries are the simplest ones: they have bounded orbits.

definition *elliptic-isometry*::(*'a*::*Gromov-hyperbolic-space* ⇒ *'a*) ⇒ *bool*
where *elliptic-isometry f* = (*isometry f* ∧ (∀ *x*. *bounded* {(*f*^^*n*) *x*|*n*. *True*}))

lemma *elliptic-isometryD*:
assumes *elliptic-isometry f*
shows *bounded* {(*f*^^*n*) *x* |*n*. *True*}
isometry f
using *assms* **unfolding** *elliptic-isometry-def* **by** *auto*

lemma *elliptic-isometryI* [*intro*]:
assumes *bounded* {(*f*^^*n*) *x* |*n*. *True*}
isometry f
shows *elliptic-isometry f*
proof −
have *bounded* {(*f*^^*n*) *y* |*n*. *True*} **for** *y*
proof −
obtain *c e* **where** *c*: ⋀*n*. *dist c* ((*f*^^*n*) *x*) ≤ *e*
using *assms*(*1*) **unfolding** *bounded-def* **by** *auto*
have *dist c* ((*f*^^*n*) *y*) ≤ *e* + *dist x y* **for** *n*
proof −
have *dist c* ((*f*^^*n*) *y*) ≤ *dist c* ((*f*^^*n*) *x*) + *dist* ((*f*^^*n*) *x*) ((*f*^^*n*) *y*)
by (*intro mono-intros*)
also have ... ≤ *e* + *dist x y*
using *c*[*of n*] *isometry-iterates*[*OF assms*(*2*), *of n*] **by** (*intro mono-intros*, *auto simp add*: *isometryD*)
finally show *?thesis* **by** *simp*
qed
then show *?thesis*
unfolding *bounded-def* **by** *auto*
qed
then show *?thesis* **unfolding** *elliptic-isometry-def* **using** *assms* **by** *auto*
qed

The inverse of an elliptic isometry is an elliptic isometry.

lemma *elliptic-isometry-inv*:
  assumes *elliptic-isometry f*
  shows *elliptic-isometry* (*inv f*)
proof −
  obtain $c\ e$ **where** A: $\bigwedge n.\ dist\ c\ ((f \widehat{\ } \widehat{\ } n)\ basepoint) \leq e$
   using *elliptic-isometryD*(*1*)[*OF assms*, *of basepoint*] **unfolding** *bounded-def* **by** *auto*
  have $c = (f \widehat{\ } \widehat{\ } n)\ (((inv\ f) \widehat{\ } \widehat{\ } n)\ c)$ **for** n
    using *fn-o-inv-fn-is-id*[*OF isometry-inverse*(*2*)[*OF elliptic-isometryD*(*2*)[*OF assms*]], *of n*]
    unfolding *comp-def* **by** *metis*
  then have $dist\ ((f \widehat{\ } \widehat{\ } n)\ (((inv\ f) \widehat{\ } \widehat{\ } n)\ c))\ ((f \widehat{\ } \widehat{\ } n)\ basepoint) \leq e$ **for** n
    using A **by** *auto*
  then have B: $dist\ basepoint\ (((inv\ f) \widehat{\ } \widehat{\ } n)\ c) \leq e$ **for** n
    unfolding *isometryD*(*2*)[*OF isometry-iterates*[*OF elliptic-isometryD*(*2*)[*OF assms*]]] **by** (*auto simp add*: *dist-commute*)
  show *?thesis*
    apply (*rule elliptic-isometryI*[*of - c*])
    using *isometry-inverse*(*1*)[*OF elliptic-isometryD*(*2*)[*OF assms*]] B **unfolding** *bounded-def* **by** *auto*
qed

The inverse of a bijective map is an elliptic isometry if and only if the original map is.

lemma *elliptic-isometry-inv-iff*:
  assumes *bij f*
  shows *elliptic-isometry* (*inv f*) $\longleftrightarrow$ *elliptic-isometry f*
using *elliptic-isometry-inv*[*of f*] *elliptic-isometry-inv*[*of inv f*] *inv-inv-eq*[*OF assms*] **by** *auto*

The identity is an elliptic isometry.

lemma *elliptic-isometry-id*:
  elliptic-isometry id
by (*intro elliptic-isometryI isometryI*, *auto*)

The translation length of an elliptic isometry is 0.

lemma *elliptic-isometry-stable-translation-length*:
  assumes *elliptic-isometry f*
  shows *stable-translation-length f* = *0*
proof −
  obtain $x::'a$ **where** *True* **by** *auto*
  have *bounded* $\{(f \widehat{\ } \widehat{\ } n)\ x | n.\ True\}$
    using *elliptic-isometryD*[*OF assms*] **by** *auto*
  then obtain $c\ C$ **where** cC: $\bigwedge n.\ dist\ c\ ((f \widehat{\ } \widehat{\ } n)\ x) \leq C$
    unfolding *bounded-def* **by** *auto*
  have $(\lambda n.\ dist\ x\ ((f \widehat{\ } \widehat{\ } n)\ x)/n) \longrightarrow 0$
  proof (*rule tendsto-sandwich*[*of* λ-. *0* - *sequentially* $\lambda n.\ 2 * C\ /\ n$])

have $(\lambda n.\ 2 * C * (1\ /\ real\ n)) \longrightarrow 2 * C * 0$ **by** (*intro tendsto-intros*)
then show $(\lambda n.\ 2 * C\ /\ real\ n) \longrightarrow 0$ **by** *auto*
have *dist x* ((*f* ^^ *n*) *x*) / *real n* $\leq$ 2 $*$ *C* / *real n* **for** *n*
using *cC*[*of 0*] *cC*[*of n*] *dist-triangle*[*of x* (*f*^^*n*) *x c*] **by** (*auto simp add*: *algebra-simps divide-simps dist-commute*)
then show *eventually* (λn. *dist x* ((*f* ^^ *n*) *x*) / *real n* $\leq$ 2 $*$ *C* / *real n*) *sequentially*
by *auto*
qed (*auto*)
moreover have (λn. *dist x* ((*f*^^*n*) *x*)/*n*) $\longrightarrow$ *stable-translation-length f*
by (*rule stable-translation-length-as-pointwise-limit*[*OF isometry-on-lipschitz*[*OF isometryD*(*1*)[*OF elliptic-isometryD*(*2*)[*OF assms*]]]])
ultimately show *?thesis*
using *LIMSEQ-unique* **by** *auto*
qed

If an isometry has a fixed point, then it is elliptic.

lemma *isometry-with-fixed-point-is-elliptic*:
assumes *isometry f f x* = *x*
shows *elliptic-isometry f*
proof −
have $*$: (*f*^^*n*) *x* = *x* **for** *n*
apply (*induction n*) **using** *assms*(*2*) **by** *auto*
show *?thesis*
apply (*rule elliptic-isometryI*[*of - x*, *OF - assms*(*1*)]) **unfolding** $*$ **by** *auto*
qed

19.4 Parabolic and loxodromic isometries

An isometry is parabolic if it is not elliptic and if its translation length vanishes.

definition *parabolic-isometry*::('*a*::*Gromov-hyperbolic-space* ⇒ '*a*) ⇒ *bool*
where *parabolic-isometry f* = (*isometry f* ∧ ¬*elliptic-isometry f* ∧ *stable-translation-length f* = *0*)

An isometry is loxodromic if it is not elliptic and if its translation length is nonzero.

definition *loxodromic-isometry*::('*a*::*Gromov-hyperbolic-space* ⇒ '*a*) ⇒ *bool*
where *loxodromic-isometry f* = (*isometry f* ∧ ¬*elliptic-isometry f* ∧ *stable-translation-length f* ≠ *0*)

The main features of such isometries are expressed in terms of their fixed points at infinity. We define them now, but proving that the definitions make sense will take some work.

definition *neutral-fixed-point*::('*a*::*Gromov-hyperbolic-space* ⇒ '*a*) ⇒ '*a Gromov-completion*
where *neutral-fixed-point f* = (*SOME xi*. *xi* ∈ *Gromov-boundary* ∧ *Gromov-extension f xi* = *xi* ∧ *additive-strength f xi* = *0*)

definition *attracting-fixed-point*::(*'a*::*Gromov-hyperbolic-space* ⇒ *'a*) ⇒ *'a Gromov-completion*
  where *attracting-fixed-point f* = (*SOME xi. xi* ∈ *Gromov-boundary* ∧ *Gromov-extension f xi* = *xi* ∧ *additive-strength f xi* < *0*)

definition *repelling-fixed-point*::(*'a*::*Gromov-hyperbolic-space* ⇒ *'a*) ⇒ *'a Gromov-completion*
  where *repelling-fixed-point f* = (*SOME xi. xi* ∈ *Gromov-boundary* ∧ *Gromov-extension f xi* = *xi* ∧ *additive-strength f xi* > *0*)

lemma *parabolic-isometryD*:
  assumes *parabolic-isometry f*
  shows *isometry f*
    ¬*bounded* {(*f*^^*n*) *x*|*n*. *True*}
    stable-translation-length f = *0*
    ¬*elliptic-isometry f*
using *assms* **unfolding** *parabolic-isometry-def* **by** *auto*

lemma *parabolic-isometryI*:
  assumes *isometry f*
    ¬*bounded* {(*f*^^*n*) *x*|*n*. *True*}
    stable-translation-length f = *0*
  shows *parabolic-isometry f*
using *assms* **unfolding** *parabolic-isometry-def elliptic-isometry-def* **by** *auto*

lemma *loxodromic-isometryD*:
  assumes *loxodromic-isometry f*
  shows *isometry f*
    ¬*bounded* {(*f*^^*n*) *x*|*n*. *True*}
    stable-translation-length f > *0*
    ¬*elliptic-isometry f*
using *assms* **unfolding** *loxodromic-isometry-def*
by (*auto*, *meson dual-order.antisym not-le stable-translation-length-nonneg*)

To have a loxodromic isometry, it suffices to know that the stable translation length is nonzero, as elliptic isometries have zero translation length.

lemma *loxodromic-isometryI*:
  assumes *isometry f*
    stable-translation-length f ≠ *0*
  shows *loxodromic-isometry f*
using *assms elliptic-isometry-stable-translation-length* **unfolding** *loxodromic-isometry-def*
by *auto*

Any isometry is elliptic, or parabolic, or loxodromic, and these possibilities are mutually exclusive.

lemma *elliptic-or-parabolic-or-loxodromic*:
  assumes *isometry f*
  shows *elliptic-isometry f* ∨ *parabolic-isometry f* ∨ *loxodromic-isometry f*

using *assms* **unfolding** *parabolic-isometry-def loxodromic-isometry-def* **by** *auto*

lemma *elliptic-imp-not-parabolic-loxodromic*:
  assumes *elliptic-isometry f*
  shows ¬*parabolic-isometry f*
    ¬*loxodromic-isometry f*
using *assms* **unfolding** *parabolic-isometry-def loxodromic-isometry-def* **by** *auto*

lemma *parabolic-imp-not-elliptic-loxodromic*:
  assumes *parabolic-isometry f*
  shows ¬*elliptic-isometry f*
    ¬*loxodromic-isometry f*
using *assms* **unfolding** *parabolic-isometry-def loxodromic-isometry-def* **by** *auto*

lemma *loxodromic-imp-not-elliptic-parabolic*:
  assumes *loxodromic-isometry f*
  shows ¬*elliptic-isometry f*
    ¬*parabolic-isometry f*
using *assms* **unfolding** *parabolic-isometry-def loxodromic-isometry-def* **by** *auto*

The inverse of a parabolic isometry is parabolic.

lemma *parabolic-isometry-inv*:
  assumes *parabolic-isometry f*
  shows *parabolic-isometry* (*inv f*)
unfolding *parabolic-isometry-def* **using** *isometry-inverse*[*of f*] *elliptic-isometry-inv-iff* [*of f*]
parabolic-isometryD[*OF assms*] *stable-translation-length-inv*[*of f*] **by** *auto*

The inverse of a loxodromic isometry is loxodromic.

lemma *loxodromic-isometry-inv*:
  assumes *loxodromic-isometry f*
  shows *loxodromic-isometry* (*inv f*)
unfolding *loxodromic-isometry-def* **using** *isometry-inverse*[*of f*] *elliptic-isometry-inv-iff* [*of f*]
loxodromic-isometryD[*OF assms*] *stable-translation-length-inv*[*of f*] **by** *auto*

We will now prove that an isometry which is not elliptic has a fixed point at infinity. This is very easy if the space is proper (ensuring that the Gromov completion is compact), but in fact this holds in general. One constructs it by considering a sequence r_n such that $f^{r_n} 0$ tends to infinity, and additionally $d(f^l 0, 0) < d(f^{r_n} 0, 0)$ for $l < r_n$: this implies the convergence at infinity of $f^{r_n} 0$, by an argument based on a Gromov product computation – and the limit is a fixed point. Moreover, it has nonpositive additive strength, essentially by construction.

lemma *high-scores*:
  fixes *u*::*nat* ⇒ *real* **and** *i*::*nat* **and** *C*::*real*
  assumes ¬(*bdd-above* (*range u*))
  shows ∃ *n*. (∀ *l* ≤ *n*. *u l* ≤ *u n*) ∧ *u n* ≥ *C* ∧ *n* ≥ *i*

```
proof -
  define M where M = max C (Max {u l|l. l < i})
  define n where n = Inf {m. u m > M}
  have ¬(range u ⊆ {..M})
    using assms by (meson bdd-above-Iic bdd-above-mono)
  then have {m. u m > M} ≠ {}
    using assms by (simp add: image-subset-iff not-less)
  then have n ∈ {m. u m > M} unfolding n-def using Inf-nat-def1 by metis
  then have u n > M by simp
  have n ≥ i
  proof (rule ccontr)
    assume ¬ i ≤ n
    then have *: n < i by simp
    have u n ≤ Max {u l|l. l < i} apply (rule Max-ge) using * by auto
    then show False using ‹u n > M› M-def by auto
  qed
  moreover have u l ≤ u n if l ≤ n for l
  proof (cases l = n)
    case True
    then show ?thesis by simp
  next
    case False
    then have l < n using ‹l ≤ n› by auto
    then have l ∉ {m. u m > M}
      unfolding n-def by (meson bdd-below-def cInf-lower not-le zero-le)
    then show ?thesis using ‹u n > M› by auto
  qed
  ultimately show ?thesis
    using ‹u n > M› M-def less-eq-real-def by auto
qed

lemma isometry-not-elliptic-has-attracting-fixed-point:
  assumes isometry f
        ¬(elliptic-isometry f)
  shows ∃ xi ∈ Gromov-boundary. Gromov-extension f xi = xi ∧ additive-strength
f xi ≤ 0
proof -
  define u where u = (λn. dist basepoint ((f^^n) basepoint))
  have NB: ¬(bdd-above (range u))
  proof
    assume bdd-above (range u)
    then obtain C where *: ⋀n. u n ≤ C unfolding bdd-above-def by auto
    have bounded {(f^^n) basepoint|n. True}
      unfolding bounded-def apply (rule exI[of - basepoint], rule exI[of - C])
      using * unfolding u-def by auto
    then show False
      using elliptic-isometryI assms by auto
  qed
  have ∃ r. ∀ n. ((∀ l ≤ r n. u l ≤ u (r n)) ∧ u (r n) ≥ 2 * n) ∧ r (Suc n) ≥ r n
```

+ 1
  apply (*rule dependent-nat-choice*) **using** *high-scores*[*OF NB*] **by** (*auto*) *blast*
 then obtain *r*::*nat* ⇒ *nat* **where** *r*: $\bigwedge n\ l.\ l \leq r\ n \Longrightarrow u\ l \leq u\ (r\ n)$
$\bigwedge n.\ u\ (r\ n) \geq 2 * n$ $\bigwedge n.\ r\ (Suc\ n) \geq r\ n + 1$
  by *auto*
 then have *strict-mono r*
  by (*metis Suc-eq-plus1 Suc-le-lessD strict-monoI-Suc*)
 then have $r\ n \geq n$ **for** *n*
  by (*simp add*: *seq-suble*)

 have *A*: *dist* ((*f*^^(*r p*)) *basepoint*) ((*f*^^(*r n*)) *basepoint*) ≤ *dist basepoint* ((*f*^^(*r n*)) *basepoint*) **if** $n \geq p$ **for** *n p*
 proof −
  have $r\ n \geq r\ p$ **using** ‹$n \geq p$› ‹*strict-mono r*› **by** (*simp add*: *strict-mono-less-eq*)
  then have ∗: *f*^^((*r n*)) = (*f*^^(*r p*)) *o* (*f*^^(*r n* − *r p*))
   unfolding *funpow-add*[*symmetric*] **by** *auto*
  have *dist* ((*f*^^(*r p*)) *basepoint*) ((*f*^^(*r n*)) *basepoint*) = *dist* ((*f*^^(*r p*)) *basepoint*) ((*f*^^(*r p*)) ((*f*^^(*r n* − *r p*)) *basepoint*))
   unfolding ∗ *comp-def* **by** *auto*
  also have ... = *dist basepoint* ((*f*^^(*r n* − *r p*)) *basepoint*)
   using *isometry-iterates*[*OF assms*(*1*), *of r p*] *isometryD* **by** *auto*
  also have ... ≤ *dist basepoint* ((*f*^^(*r n*)) *basepoint*)
   using *r*(*1*)[*of r n* − *r p n*] **unfolding** *u-def* **by** *auto*
  finally show *?thesis*
   by *simp*
 qed

 have ∗: *Gromov-product-at basepoint* ((*f*^^(*r p*)) *basepoint*) ((*f*^^(*r n*)) *basepoint*) ≥ *p* **if** $n \geq p$ **for** *n p*
 proof −
  have *2* ∗ *Gromov-product-at basepoint* ((*f*^^(*r p*)) *basepoint*) ((*f*^^(*r n*)) *basepoint*)
   = *dist basepoint* ((*f*^^(*r p*)) *basepoint*) + *dist basepoint* ((*f*^^(*r n*)) *basepoint*)
   − *dist* ((*f*^^(*r p*)) *basepoint*) ((*f*^^(*r n*)) *basepoint*)
   unfolding *Gromov-product-at-def* **by** *auto*
  also have ... ≥ *dist basepoint* ((*f*^^(*r p*)) *basepoint*)
   using *A*[*OF that*] **by** *auto*
  finally show *Gromov-product-at basepoint* ((*f*^^(*r p*)) *basepoint*) ((*f*^^(*r n*)) *basepoint*) ≥ *p*
   using *r*(*2*)[*of p*] **unfolding** *u-def* **by** *auto*
 qed
 have ∗: *Gromov-product-at basepoint* ((*f*^^(*r p*)) *basepoint*) ((*f*^^(*r n*)) *basepoint*) ≥ *N* **if** $n \geq N$ $p \geq N$ **for** *n p N*
  using ∗[*of n p*] ∗[*of p n*] *that* **by** (*auto simp add*: *Gromov-product-commute intro*: *le-cases*[*of n p*])
 have *Gromov-converging-at-boundary* (λ*n*. (*f*^^(*r n*)) *basepoint*)
  apply (*rule Gromov-converging-at-boundaryI*[*of basepoint*]) **using** ∗ **by** (*meson dual-order.trans real-arch-simple*)

with *Gromov-converging-at-boundary-converges*[*OF this*]
obtain *xi* **where** *xi*: $(\lambda n.$ *to-Gromov-completion* $((f^{\frown\frown}(r\ n))$ *basepoint*$)) \longlongrightarrow$ *xi xi* $\in$ *Gromov-boundary*
by *auto*
then have ∗: $(\lambda n.$ *Gromov-extension f* (*to-Gromov-completion* $((f^{\frown\frown}(r\ n))$ *basepoint*$))) \longrightarrow$ *xi*
apply (*simp*, *rule Gromov-converging-at-boundary-bounded-perturbation*[*of - - - dist basepoint* (*f basepoint*)])
by (*simp add*: *assms*(*1*) *funpow-swap1 isometryD*(*2*) *isometry-iterates*)
moreover have $(\lambda n.$ *Gromov-extension f* (*to-Gromov-completion* $((f^{\frown\frown}(r\ n))$ *basepoint*$))) \longrightarrow$ *Gromov-extension f xi*
using *continuous-on-tendsto-compose*[*OF Gromov-extension-isometry*(*2*)[*OF assms*(*1*)] *xi*(*1*)] **by** *auto*
ultimately have *fxi*: *Gromov-extension f xi* = *xi*
using *LIMSEQ-unique* **by** *auto*

have *Busemann-function-at* (*to-Gromov-completion* $((f^{\frown\frown}(r\ n))$ *basepoint*)) $((f^{\frown\frown}(r\ p))$ *basepoint*) *basepoint* ≤ 0 **if** $n \geq p$ **for** *n p*
unfolding *Busemann-function-inner* **using** *A*[*OF that*] **by** *auto*
then have *A*: *eventually* $(\lambda n.$ *Busemann-function-at* (*to-Gromov-completion* $((f^{\frown\frown}(r\ n))$ *basepoint*)) $((f^{\frown\frown}(r\ p))$ *basepoint*) *basepoint* ≤ 0) *sequentially* **for** *p*
unfolding *eventually-sequentially* **by** *auto*
have *B*: *eventually* $(\lambda n.$ *Busemann-function-at* (*to-Gromov-completion* $((f^{\frown\frown}(r\ n))$ *basepoint*)) $((f^{\frown\frown}(r\ p))$ *basepoint*) *basepoint* $\geq$ *Busemann-function-at xi* $((f^{\frown\frown}(r\ p))$ *basepoint*) *basepoint* $- 2 *$ *deltaG*(*TYPE*(′*a*)) $- 1$) *sequentially* **for** *p*
by (*rule eventually-mono*[*OF Busemann-function-inside-approx*[*OF - xi*(*1*), *of 1* $((f^{\frown\frown}(r\ p))$ *basepoint*) *basepoint*, *simplified*]], *simp*)
have *eventually* $(\lambda n.$ *Busemann-function-at xi* $((f^{\frown\frown}(r\ p))$ *basepoint*) *basepoint* $- 2 *$ *deltaG*(*TYPE*(′*a*)) $- 1 \leq 0$) *sequentially* **for** *p*
by (*rule eventually-mono*[*OF eventually-conj*[*OF A*[*of p*] *B*[*of p*]]], *simp*)
then have ∗: *Busemann-function-at xi* $((f^{\frown\frown}(r\ p))$ *basepoint*) *basepoint* $- 2 *$ *deltaG*(*TYPE*(′*a*)) $- 1 \leq 0$ **for** *p*
by *auto*
then have *A*: *Busemann-function-at xi* $((f^{\frown\frown}(r\ p))$ *basepoint*) *basepoint* / (*r p*) $-$ (*2* ∗ *deltaG*(*TYPE*(′*a*)) + *1*) ∗ (*1*/*r p*) ≤ 0 **if** $p \geq 1$ **for** *p*
using *order-trans*[*OF that* ‹$p \leq r\ p$›] **by** (*auto simp add*: *algebra-simps divide-simps*)
have *B*: $(\lambda p.$ *Busemann-function-at xi* $((f^{\frown\frown}(p))$ *basepoint*) *basepoint* / *p* $-$ (*2* ∗ *deltaG*(*TYPE*(′*a*)) + *1*) ∗ (*1*/*p*)) $\longrightarrow$ *additive-strength f xi* $-$ (*2* ∗ *deltaG*(*TYPE*(′*a*)) + *1*) ∗ *0*
by (*intro tendsto-intros assms fxi*)
have *additive-strength f xi* $-$ (*2* ∗ *deltaG TYPE*(′*a*) + *1*) ∗ *0* $\leq$ *0*
apply (*rule LIMSEQ-le-const2*[*OF LIMSEQ-subseq-LIMSEQ*[*OF B* ‹*strict-mono r*›]]) **using** *A* **unfolding** *comp-def* **by** *auto*
then show *?thesis* **using** *xi fxi* **by** *auto*
qed

Applying the previous result to the inverse map, we deduce that there is also a fixed point with nonnegative strength.

lemma *isometry-not-elliptic-has-repelling-fixed-point*:
  assumes *isometry f*
    $\neg$(*elliptic-isometry f*)
  shows $\exists$ *xi* $\in$ *Gromov-boundary*. *Gromov-extension f xi* = *xi* $\wedge$ *additive-strength f xi* $\geq$ *0*
proof −
  have $*$: $\neg$*elliptic-isometry* (*inv f*)
    using *elliptic-isometry-inv-iff isometry-inverse*(*2*)[*OF assms*(*1*)] *assms*(*2*) **by** *auto*
  obtain *xi* **where** *xi*: *xi* $\in$ *Gromov-boundary Gromov-extension* (*inv f*) *xi* = *xi additive-strength* (*inv f*) *xi* $\leq$ *0*
    using *isometry-not-elliptic-has-attracting-fixed-point*[*OF isometry-inverse*(*1*)[*OF assms*(*1*)] $*$] **by** *auto*
  have $*$: *Gromov-extension f xi* = *xi*
    using *Gromov-extension-inv-fixed-point*[*OF isometry-inverse*(*1*)[*OF assms*(*1*)] *xi*(*2*)] *inv-inv-eq*[*OF isometry-inverse*(*2*)[*OF assms*(*1*)]] **by** *auto*
  moreover have *additive-strength f xi* $\geq$ *0*
    using *additive-strength-inv*[*OF assms*(*1*) $*$] *xi*(*3*) **by** *auto*
  ultimately show *?thesis*
    using *xi*(*1*) **by** *auto*
qed

19.4.1 Parabolic isometries

We show that a parabolic isometry has (at least) one neutral fixed point at infinity.

lemma *parabolic-fixed-point*:
  assumes *parabolic-isometry f*
  shows *neutral-fixed-point f* $\in$ *Gromov-boundary*
    Gromov-extension f (*neutral-fixed-point f*) = *neutral-fixed-point f*
    additive-strength f (*neutral-fixed-point f*) = *0*
proof −
  obtain *xi* **where** *xi*: *xi* $\in$ *Gromov-boundary Gromov-extension f xi* = *xi*
    using *isometry-not-elliptic-has-attracting-fixed-point parabolic-isometryD*[*OF assms*] **by** *blast*
  moreover have *additive-strength f xi* = *0*
    using *stable-translation-length-eq-additive-strength*[*OF parabolic-isometryD*(*1*)[*OF assms*] *xi*(*2*)]
    parabolic-isometryD(*3*)[*OF assms*] **by** *auto*
  ultimately have *A*: $\exists$ *xi*. *xi* $\in$ *Gromov-boundary* $\wedge$ *Gromov-extension f xi* = *xi* $\wedge$ *additive-strength f xi* = *0*
    by *auto*
  show *neutral-fixed-point f* $\in$ *Gromov-boundary*
    Gromov-extension f (*neutral-fixed-point f*) = *neutral-fixed-point f*
    additive-strength f (*neutral-fixed-point f*) = *0*
    unfolding *neutral-fixed-point-def* **using** *someI-ex*[*OF A*] **by** *auto*
qed

Parabolic isometries have exactly one fixed point, the neutral fixed point at infinity. The proof goes as follows: if it has another fixed point, then the orbit of a basepoint would stay on the horospheres centered at both fixed points. But the intersection of two horospheres based at different points is a a bounded set. Hence, the map has a bounded orbit, and is therefore elliptic.

theorem *parabolic-unique-fixed-point*:
  assumes *parabolic-isometry f*
  shows *Gromov-extension f xi* = *xi* ⟷ *xi* = *neutral-fixed-point f*
proof (*auto simp add*: *parabolic-fixed-point*[*OF assms*])
  assume *H*: *Gromov-extension f xi* = *xi*
  have ∗: *additive-strength f xi* = *0*
   using *stable-translation-length-eq-additive-strength*[*OF parabolic-isometryD*(*1*)[*OF assms*] *H*]
    parabolic-isometryD(*3*)[*OF assms*] **by** *auto*
  show *xi* = *neutral-fixed-point f*
  proof (*rule ccontr*)
    assume *N*: *xi* ≠ *neutral-fixed-point f*
    define *C* **where** *C* = *2* ∗ *real-of-ereal* (*extended-Gromov-product-at basepoint xi* (*neutral-fixed-point f*)) + *4* ∗ *deltaG*(*TYPE*(*'a*))
    have *A*: *dist basepoint* ((*f^^n*) *basepoint*) ≤ *C* **for** *n*
    proof −
     have *dist* ((*f^^n*) *basepoint*) *basepoint* − *2* ∗ *real-of-ereal* (*extended-Gromov-product-at basepoint xi* (*neutral-fixed-point f*)) − *2* ∗ *deltaG*(*TYPE*(*'a*))
        ≤ *max* (*Busemann-function-at xi* ((*f^^n*) *basepoint*) *basepoint*) (*Busemann-function-at* (*neutral-fixed-point f*) ((*f^^n*) *basepoint*) *basepoint*)
       using *dist-le-max-Busemann-functions*[*OF N*] **by** (*simp add*: *algebra-simps*)
      also have ... ≤ *max* (*n* ∗ *additive-strength f xi* + *2* ∗ *deltaG*(*TYPE*(*'a*))) (*n* ∗ *additive-strength f* (*neutral-fixed-point f*) + *2* ∗ *deltaG*(*TYPE*(*'a*)))
        apply (*intro mono-intros*)
      using *Busemann-function-eq-additive-strength*[*OF parabolic-isometryD*(*1*)[*OF assms*] *H*, *of n basepoint*]
      Busemann-function-eq-additive-strength[*OF parabolic-isometryD*(*1*)[*OF assms*] *parabolic-fixed-point*(*2*)[*OF assms*], *of n basepoint*]
        by *auto*
      also have ... = *2* ∗ *deltaG*(*TYPE*(*'a*))
        unfolding ∗ *parabolic-fixed-point*[*OF assms*] **by** *auto*
      finally show *?thesis*
        unfolding *C-def* **by** (*simp add*: *algebra-simps dist-commute*)
    qed
    have *elliptic-isometry f*
      apply (*rule elliptic-isometryI*[*of - basepoint*])
      using *parabolic-isometryD*(*1*)[*OF assms*] *A* **unfolding** *bounded-def* **by** *auto*
    then show *False*
      using *elliptic-imp-not-parabolic-loxodromic assms* **by** *auto*
  qed
qed

When one iterates a parabolic isometry, the distance to the starting point can grow at most logarithmically.

lemma *parabolic-logarithmic-growth*:
assumes *parabolic-isometry* (*f*::*'a*::*Gromov-hyperbolic-space* ⇒ *'a*) *n* ≥ *1*
shows *dist x* ((*f*^^*n*) *x*) ≤ *dist x* (*f x*) + *ceiling* (*log 2 n*) ∗ *16* ∗ *deltaG*(*TYPE*(*'a*))
using *dist-le-additive-strength*[*OF parabolic-isometryD*(*1*)[*OF assms*(*1*)] *parabolic-fixed-point*(*2*)[*OF assms*(*1*)] - *assms*(*2*)]
parabolic-isometryD(*3*)[*OF assms*(*1*)] *stable-translation-length-eq-additive-strength*[*OF parabolic-isometryD*(*1*)[*OF assms*(*1*)] *parabolic-fixed-point*(*2*)[*OF assms*(*1*)]]
by *auto*

It follows that there is no parabolic isometry in trees, since the formula in the previous lemma shows that there is no orbit growth as $\delta = 0$, and therefore orbits are bounded, contradicting the parabolicity of the isometry.

lemma *tree-no-parabolic-isometry*:
assumes *isometry* (*f*::*'a*::*Gromov-hyperbolic-space-0* ⇒ *'a*)
shows *elliptic-isometry f* ∨ *loxodromic-isometry f*
proof −
have ¬*parabolic-isometry f*
proof
assume *P*: *parabolic-isometry f*
have *dist basepoint* ((*f*^^*n*) *basepoint*) ≤ *dist basepoint* (*f basepoint*) **if** *n* ≥ *1* **for** *n*
using *parabolic-logarithmic-growth*[*OF P that*, *of basepoint*] **by** *auto*
then have ∗: *dist basepoint* ((*f*^^*n*) *basepoint*) ≤ *dist basepoint* (*f basepoint*) **for** *n*
by (*cases n* ≥ *1*, *auto simp add*: *not-less-eq-eq*)
have *elliptic-isometry f*
apply (*rule elliptic-isometryI*[*OF - assms*, *of basepoint*]) **using** ∗ **unfolding** *bounded-def* **by** *auto*
then show *False*
using *elliptic-imp-not-parabolic-loxodromic P* **by** *auto*
qed
then show *?thesis*
using *elliptic-or-parabolic-or-loxodromic*[*OF assms*] **by** *auto*
qed

19.4.2 Loxodromic isometries

A loxodromic isometry has (at least) two fixed points at infinity, one attracting and one repelling. We have already constructed fixed points with nonnegative and nonpositive strengths. Since the strength is nonzero (its absolute value is the stable translation length), then these fixed points correspond to what we want.

lemma *loxodromic-attracting-fixed-point*:
assumes *loxodromic-isometry f*
shows *attracting-fixed-point f* ∈ *Gromov-boundary*
Gromov-extension f (*attracting-fixed-point f*) = *attracting-fixed-point f*
additive-strength f (*attracting-fixed-point f*) < *0*
proof −

obtain *xi* **where** *xi*: *xi* $\in$ *Gromov-boundary Gromov-extension f xi* = *xi additive-strength f xi* $\leq$ *0*
using *isometry-not-elliptic-has-attracting-fixed-point loxodromic-isometryD*[*OF assms*] **by** *blast*
moreover have *additive-strength f xi* < *0*
using *stable-translation-length-eq-additive-strength*[*OF loxodromic-isometryD*(*1*)[*OF assms*] *xi*(*2*)]
loxodromic-isometryD(*3*)[*OF assms*] *xi*(*3*) **by** *auto*
ultimately have *A*: $\exists$ *xi*. *xi* $\in$ *Gromov-boundary* $\wedge$ *Gromov-extension f xi* = *xi* $\wedge$ *additive-strength f xi* < *0*
by *auto*
show *attracting-fixed-point f* $\in$ *Gromov-boundary*
Gromov-extension f (*attracting-fixed-point f*) = *attracting-fixed-point f*
additive-strength f (*attracting-fixed-point f*) < *0*
unfolding *attracting-fixed-point-def* **using** *someI-ex*[*OF A*] **by** *auto*
qed

lemma *loxodromic-repelling-fixed-point*:
assumes *loxodromic-isometry f*
shows *repelling-fixed-point f* $\in$ *Gromov-boundary*
Gromov-extension f (*repelling-fixed-point f*) = *repelling-fixed-point f*
additive-strength f (*repelling-fixed-point f*) > *0*
proof −
obtain *xi* **where** *xi*: *xi* $\in$ *Gromov-boundary Gromov-extension f xi* = *xi additive-strength f xi* $\geq$ *0*
using *isometry-not-elliptic-has-repelling-fixed-point loxodromic-isometryD*[*OF assms*] **by** *blast*
moreover have *additive-strength f xi* > *0*
using *stable-translation-length-eq-additive-strength*[*OF loxodromic-isometryD*(*1*)[*OF assms*] *xi*(*2*)]
loxodromic-isometryD(*3*)[*OF assms*] *xi*(*3*) **by** *auto*
ultimately have *A*: $\exists$ *xi*. *xi* $\in$ *Gromov-boundary* $\wedge$ *Gromov-extension f xi* = *xi* $\wedge$ *additive-strength f xi* > *0*
by *auto*
show *repelling-fixed-point f* $\in$ *Gromov-boundary*
Gromov-extension f (*repelling-fixed-point f*) = *repelling-fixed-point f*
additive-strength f (*repelling-fixed-point f*) > *0*
unfolding *repelling-fixed-point-def* **using** *someI-ex*[*OF A*] **by** *auto*
qed

The attracting and repelling fixed points of a loxodromic isometry are distinct – precisely since one is attracting and the other is repelling.

lemma *attracting-fixed-point-neq-repelling-fixed-point*:
assumes *loxodromic-isometry f*
shows *attracting-fixed-point f* $\neq$ *repelling-fixed-point f*
using *loxodromic-repelling-fixed-point*[*OF assms*] *loxodromic-attracting-fixed-point*[*OF assms*] **by** *auto*

The attracting fixed point of a loxodromic isometry is indeed attracting.

Moreover, the convergence is uniform away from the repelling fixed point. This is expressed in the following proposition, where neighborhoods of the repelling and attracting fixed points are given by the property that the Gromov product with the fixed point is large.

The proof goes as follows. First, the Busemann function with respect to the fixed points at infinity evolves like the strength. Therefore, $f^n e$ tends to the repulsive fixed point in negative time, and to the attracting one in positive time. Consider now a general point x with $(\xi^-, x)_e \leq K$. This means that the geodesics from e to x and ξ^- diverge before time K. For large n, since $f^{-n}e$ is close to ξ^-, we also get the inequality $(f^{-n}e, x)_e \leq K$. Applying f^n and using the invariance of the Gromov product under isometries yields $(e, f^n x)_{f^n e} \leq K$. But this Gromov product is equal to $d(e, f^n e) - (f^n e, f^n x)_e$ (this is a general property of Gromov products). In particular, $(f^n e, f^n x) \geq d(e, f^n e) - K$, and moreover $d(e, f^n e)$ is large. Since $f^n e$ is close to ξ^+, it follows that $f^n x$ is also close to ξ^+, as desired.

The real proof requires some more care as everything should be done in ereal, and moreover every inequality is only true up to some multiple of δ. But everything works in the way just described above.

proposition *loxodromic-attracting-fixed-point-attracts-uniformly*:
  assumes *loxodromic-isometry f*
  shows $\exists N. \forall n \geq N. \forall x.$ *extended-Gromov-product-at basepoint x* (*repelling-fixed-point f*) $\leq$ *ereal K*
    $\longrightarrow$ *extended-Gromov-product-at basepoint* (((*Gromov-extension f*)^^*n*) *x*) (*attracting-fixed-point f*) $\geq$ *ereal M*
proof $-$
  have *I*: *isometry f*
    using *loxodromic-isometryD*(*1*)[*OF assms*] **by** *simp*
  have *J*: $|$*ereal r*$| \neq \infty$ **for** *r* **by** *auto*

We show that $f^n e$ tends to the repelling fixed point in negative time.

  have ($\lambda n.$ *ereal* (*Busemann-function-at* (*repelling-fixed-point f*) ((*inv f* ^^ *n*) *basepoint*) *basepoint*)) $\longrightarrow - \infty$
  proof (*rule tendsto-sandwich*[*of* $\lambda n. -\infty$ - - $\lambda n.$ *ereal*($-$ *real n* $*$ *additive-strength f* (*repelling-fixed-point f*) + *2* $*$ *deltaG*(*TYPE*('*a*))), *OF* - *always-eventually*], *auto*)
    fix *n*
    have *Busemann-function-at* (*repelling-fixed-point f*) ((*inv f* ^^ *n*) *basepoint*) *basepoint* $\leq$ *real n* $*$ *additive-strength* (*inv f*) (*repelling-fixed-point f*) + *2* $*$ *deltaG*(*TYPE*('*a*))
      using *Busemann-function-eq-additive-strength*[*OF isometry-inverse*(*1*)[*OF I*] *Gromov-extension-inv-fixed-point*[*OF I loxodromic-repelling-fixed-point*(*2*)[*OF assms*]], *of n basepoint*] **by** *auto*
    then show *Busemann-function-at* (*repelling-fixed-point f*) ((*inv f* ^^ *n*) *basepoint*) *basepoint* $\leq$ *2* $*$ *deltaG*(*TYPE*('*a*)) $-$ *real n* $*$ *additive-strength f* (*repelling-fixed-point f*)
      by (*simp add*: *I additive-strength-inv assms loxodromic-repelling-fixed-point*(*2*))
  next
    have ($\lambda n.$ *ereal* (*2* $*$ *deltaG TYPE*('*a*)) + *ereal* ($-$ *real n*) $*$ *additive-strength*

f (*repelling-fixed-point f*)) $\longrightarrow$ *ereal* (*2* $*$ *deltaG* (*TYPE*('a))) + ($-\infty$) $*$ *additive-strength f* (*repelling-fixed-point f*)
apply (*intro tendsto-intros*) **using** *loxodromic-repelling-fixed-point*(*3*)[*OF assms*] **by** *auto*
then show (λ*n*. *ereal* (*2* $*$ *deltaG TYPE*('a) $-$ *real n* $*$ *additive-strength f* (*repelling-fixed-point f*))) $\longrightarrow$ $-\infty$
using *loxodromic-repelling-fixed-point*(*3*)[*OF assms*] **by** *auto*
qed
then have (λ*n*. *to-Gromov-completion* (((*inv f*)^^*n*) *basepoint*)) $\longrightarrow$ *repelling-fixed-point f*
by (*rule Busemann-function-minus-infinity-imp-convergent*[*of - - basepoint*])
then have (λ*n*. *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (((*inv f*)^^*n*) *basepoint*)) (*repelling-fixed-point f*)) $\longrightarrow$ ∞
unfolding *Gromov-completion-boundary-limit*[*OF loxodromic-repelling-fixed-point*(*1*)[*OF assms*]] **by** *simp*
then obtain *Nr* **where** *Nr*: $\bigwedge$*n*. *n* $\geq$ *Nr* $\Longrightarrow$ *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (((*inv f*)^^*n*) *basepoint*)) (*repelling-fixed-point f*) $\geq$ *ereal* (*K* + *deltaG*(*TYPE*('a)) + *1*)
unfolding *Lim-PInfty* **by** *auto*

We show that $f^n e$ tends to the attracting fixed point in positive time.

have (λ*n*. *ereal* (*Busemann-function-at* (*attracting-fixed-point f*) ((*f* ^^ *n*) *basepoint*) *basepoint*)) $\longrightarrow$ $-\infty$
proof (*rule tendsto-sandwich*[*of* λ*n*. $-\infty$ *- -* λ*n*. *ereal*(*real n* $*$ *additive-strength f* (*attracting-fixed-point f*) + *2* $*$ *deltaG*(*TYPE*('a))), *OF - always-eventually*], *auto*)
fix *n*
show *Busemann-function-at* (*attracting-fixed-point f*) ((*f* ^^ *n*) *basepoint*) *basepoint* $\leq$ *real n* $*$ *additive-strength f* (*attracting-fixed-point f*) + *2* $*$ *deltaG*(*TYPE*('a))
using *Busemann-function-eq-additive-strength*[*OF I loxodromic-attracting-fixed-point*(*2*)[*OF assms*], *of n basepoint*] **by** *auto*
next
have (λ*n*. *ereal* (*2* $*$ *deltaG TYPE*('a)) + *ereal* (*real n*) $*$ *additive-strength f* (*attracting-fixed-point f*)) $\longrightarrow$ *ereal* (*2* $*$ *deltaG* (*TYPE*('a))) + (∞) $*$ *additive-strength f* (*attracting-fixed-point f*)
apply (*intro tendsto-intros*) **using** *loxodromic-attracting-fixed-point*(*3*)[*OF assms*] **by** *auto*
then show (λ*n*. *ereal* (*real n* $*$ *additive-strength f* (*attracting-fixed-point f*) + *2* $*$ *deltaG TYPE*('a))) $\longrightarrow$ $-\infty$
using *loxodromic-attracting-fixed-point*(*3*)[*OF assms*] **by** (*auto simp add*: *algebra-simps*)
qed
then have $*$: (λ*n*. *to-Gromov-completion* (((*f*)^^*n*) *basepoint*)) $\longrightarrow$ *attracting-fixed-point f*
by (*rule Busemann-function-minus-infinity-imp-convergent*[*of - - basepoint*])
then have (λ*n*. *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (((*f*)^^*n*) *basepoint*)) (*attracting-fixed-point f*)) $\longrightarrow$ ∞
unfolding *Gromov-completion-boundary-limit*[*OF loxodromic-attracting-fixed-point*(*1*)[*OF assms*]] **by** *simp*
then obtain *Na* **where** *Na*: $\bigwedge$*n*. *n* $\geq$ *Na* $\Longrightarrow$ *extended-Gromov-product-at*

basepoint (*to-Gromov-completion* (((*f*)^^*n*) *basepoint*)) (*attracting-fixed-point f*) ≥ *ereal* (*M* + *deltaG*(*TYPE*('*a*)))
unfolding *Lim-PInfty* **by** *auto*

We show that the distance between e and $f^n e$ tends to infinity.

have (λ*n*. *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*to-Gromov-completion* ((*f*^^*n*) *basepoint*))) ⟶
extended-Gromov-distance (*to-Gromov-completion basepoint*) (*attracting-fixed-point f*)
using * *extended-Gromov-distance-continuous*[*of to-Gromov-completion basepoint*]
by (*metis UNIV-I continuous-on filterlim-compose tendsto-at-iff-tendsto-nhds*)
then have (λ*n*. *extended-Gromov-distance* (*to-Gromov-completion basepoint*) (*to-Gromov-completion* ((*f*^^*n*) *basepoint*))) ⟶ ∞
using *loxodromic-attracting-fixed-point*(*1*)[*OF assms*] **by** *simp*
then obtain *Nd* **where** *Nd*: ⋀*n*. *n* ≥ *Nd* ⟹ *dist basepoint* ((*f*^^*n*) *basepoint*) ≥ *M* + *K* + *3* * *deltaG*(*TYPE*('*a*))
unfolding *Lim-PInfty* **by** *auto*

Now, if n is large enough so that all the convergences to infinity above are almost realized, we show that a point x which is not too close to ξ^- is sent by f^n to a point close to ξ^+, uniformly.

define *N* **where** *N* = *Nr* + *Na* + *Nd*
have *extended-Gromov-product-at basepoint* (((*Gromov-extension f*)^^*n*) *x*) (*attracting-fixed-point f*) ≥ *ereal M* **if** *H*: *extended-Gromov-product-at basepoint x* (*repelling-fixed-point f*) ≤ *K n* ≥ *N* **for** *n x*
proof −
have *n*: *n* ≥ *Nr n* ≥ *Na n* ≥ *Nd* **using** *that* **unfolding** *N-def* **by** *auto*
have *min* (*extended-Gromov-product-at basepoint x* (*to-Gromov-completion* (((*inv f*)^^*n*) *basepoint*)))
(*extended-Gromov-product-at basepoint* (*to-Gromov-completion* (((*inv f*)^^*n*) *basepoint*)) (*repelling-fixed-point f*))
≤ *extended-Gromov-product-at basepoint x* (*repelling-fixed-point f*) + *deltaG*(*TYPE*('*a*))
by (*intro mono-intros*)
also have ... ≤ *ereal K* + *deltaG*(*TYPE*('*a*))
apply (*intro mono-intros*) **using** *H* **by** *auto*
finally have *min* (*extended-Gromov-product-at basepoint x* (*to-Gromov-completion* (((*inv f*)^^*n*) *basepoint*)))
(*extended-Gromov-product-at basepoint* (*to-Gromov-completion* (((*inv f*)^^*n*) *basepoint*)) (*repelling-fixed-point f*))
≤ *ereal K* + *deltaG*(*TYPE*('*a*))
by *simp*
moreover have *extended-Gromov-product-at basepoint* (*to-Gromov-completion* (((*inv f*)^^*n*) *basepoint*)) (*repelling-fixed-point f*) > *ereal K* + *deltaG*(*TYPE*('*a*))
using *Nr*[*OF n*(*1*)] *ereal-le-less* **by** *auto*
ultimately have *A*: *extended-Gromov-product-at basepoint x* (*to-Gromov-completion* (((*inv f*)^^*n*) *basepoint*)) ≤ *ereal K* + *deltaG*(*TYPE*('*a*))
using *not-le* **by** *fastforce*

have ∗: ((*inv f*) ^^ *n*) ((*f* ^^ *n*) *z*) = *z* **for** *z*
by (*metis I bij-is-inj inj-fn inv-f-f inv-fn isometry-inverse*(*2*))
have ∗∗: *x* = *Gromov-extension* ((*inv f*) ^^ *n*) (*Gromov-extension* (*f* ^^ *n*) *x*)
using *Gromov-extension-isometry-composition*[*OF isometry-iterates*[*OF I*] *isometry-iterates*[*OF isometry-inverse*(*1*)[*OF I*]], *of n n*]
unfolding *comp-def* ∗ **apply** *auto* **by** *meson*
have *extended-Gromov-product-at* (((*inv f*) ^^ *n*) ((*f* ^^ *n*) *basepoint*)) (*Gromov-extension* ((*inv f*) ^^ *n*) (*Gromov-extension* (*f* ^^ *n*) *x*)) (*Gromov-extension* (((*inv f*) ^^ *n*)) (*to-Gromov-completion basepoint*)) ≤ *ereal K* + *deltaG*(*TYPE*('*a*))
using *A* **by** (*simp add*: ∗ ∗∗[*symmetric*])
then have *B*: *extended-Gromov-product-at* ((*f* ^^ *n*) *basepoint*) (*Gromov-extension* (*f* ^^ *n*) *x*) (*to-Gromov-completion basepoint*) ≤ *ereal K* + *deltaG*(*TYPE*('*a*))
unfolding *Gromov-extension-preserves-extended-Gromov-product*[*OF isometry-iterates*[*OF isometry-inverse*(*1*)[*OF I*]]] **by** *simp*

have *dist basepoint* ((*f* ^^ *n*) *basepoint*) − *deltaG*(*TYPE*('*a*)) ≤
extended-Gromov-product-at ((*f* ^^ *n*) *basepoint*) (*Gromov-extension* (*f* ^^ *n*) *x*) (*to-Gromov-completion basepoint*) + *extended-Gromov-product-at basepoint* (*Gromov-extension* (*f* ^^ *n*) *x*) (*to-Gromov-completion* ((*f* ^^ *n*) *basepoint*))
using *extended-Gromov-product-add-ge*[*of basepoint* (*f* ^^ *n*) *basepoint Gromov-extension* (*f* ^^ *n*) *x*] **by** (*auto simp add*: *algebra-simps*)
also have ... ≤ (*ereal K* + *deltaG*(*TYPE*('*a*))) + *extended-Gromov-product-at basepoint* (*Gromov-extension* (*f* ^^ *n*) *x*) (*to-Gromov-completion* ((*f* ^^ *n*) *basepoint*))
by (*intro mono-intros B*)
finally have *extended-Gromov-product-at basepoint* (*Gromov-extension* (*f* ^^ *n*) *x*) (*to-Gromov-completion* ((*f* ^^ *n*) *basepoint*)) ≥ *dist basepoint* ((*f* ^^ *n*) *basepoint*) − (*2* ∗ *deltaG*(*TYPE*('*a*)) + *K*)
apply (*simp only*: *ereal-minus-le* [*OF J*] *ereal-minus*(*1*) [*symmetric*])
apply (*auto simp add*: *algebra-simps*)
apply (*metis* (*no-types*, *opaque-lifting*) *add.assoc add.left-commute mult-2-right plus-ereal.simps*(*1*))
done
moreover have *dist basepoint* ((*f* ^^ *n*) *basepoint*) − (*2* ∗ *deltaG TYPE*('*a*) + *K*) ≥ *M* + *deltaG*(*TYPE*('*a*))
using *Nd*[*OF n*(*3*)] **by** *auto*
ultimately have *extended-Gromov-product-at basepoint* (*Gromov-extension* (*f* ^^ *n*) *x*) (*to-Gromov-completion* ((*f* ^^ *n*) *basepoint*)) ≥ *ereal* (*M* + *deltaG*(*TYPE*('*a*)))
using *order-trans ereal-le-le* **by** *auto*
then have *ereal* (*M* + *deltaG*(*TYPE*('*a*))) ≤ *min* (*extended-Gromov-product-at basepoint* (*Gromov-extension* (*f* ^^ *n*) *x*) (*to-Gromov-completion* ((*f* ^^ *n*) *basepoint*)))
(*extended-Gromov-product-at basepoint* (*to-Gromov-completion* ((*f* ^^ *n*) *basepoint*)) (*attracting-fixed-point f*))
using *Na*[*OF n*(*2*)] **by** (*simp add*: *extended-Gromov-product-at-commute*)
also have ... ≤ *extended-Gromov-product-at basepoint* (*Gromov-extension* (*f* ^^ *n*) *x*) (*attracting-fixed-point f*) + *deltaG*(*TYPE*('*a*))
by (*intro mono-intros*)
finally have *ereal M* ≤ *extended-Gromov-product-at basepoint* (*Gromov-extension* (*f* ^^ *n*) *x*) (*attracting-fixed-point f*)
unfolding *plus-ereal.simps*(*1*)[*symmetric*] *ereal-add-le-add-iff2* **by** *auto*

then show *?thesis*
by (*simp add*: *Gromov-extension-isometry-iterates I*)
qed
then show *?thesis*
by *auto*
qed

We deduce pointwise convergence from the previous result.

lemma *loxodromic-attracting-fixed-point-attracts*:
assumes *loxodromic-isometry f*
xi $\neq$ *repelling-fixed-point f*
shows $(\lambda n.\ ((\mathit{Gromov\text{-}extension}\ f)\widehat{\ }\widehat{\ }n)\ \mathit{xi}) \longrightarrow$ *attracting-fixed-point f*
proof −
have (λn. *extended-Gromov-product-at basepoint* (((*Gromov-extension f*)^^*n*) *xi*) (*attracting-fixed-point f*)) $\longrightarrow \infty$
unfolding *Lim-PInfty* **using** *loxodromic-attracting-fixed-point-attracts-uniformly*[*OF assms*(*1*)]
by *auto* (*metis Gromov-boundary-extended-product-PInf assms*(*2*) *dual-order.refl real-le-ereal-iff real-of-ereal-le-0 zero-ereal-def*)
then show *?thesis*
unfolding *Gromov-completion-boundary-limit*[*OF loxodromic-attracting-fixed-point*(*1*)[*OF assms*(*1*)]] **by** *simp*
qed

Finally, we show that a loxodromic isometry has exactly two fixed points, its attracting and repelling fixed points defined above. Indeed, we already know that these points are fixed. It remains to see that there is no other fixed point. But a fixed point which is not the repelling one is both stationary and attracted to the attracting fixed point by the previous lemma, hence it has to coincide with the attracting fixed point.

theorem *loxodromic-unique-fixed-points*:
assumes *loxodromic-isometry f*
shows *Gromov-extension f xi* = *xi* $\longleftrightarrow$ *xi* = *attracting-fixed-point f* $\vee$ *xi* = *repelling-fixed-point f*
proof −
have *xi* = *attracting-fixed-point f* **if** *H*: *Gromov-extension f xi* = *xi xi* $\neq$ *repelling-fixed-point f* **for** *xi*
proof −
have ((*Gromov-extension f*)^^*n*) *xi* = *xi* **for** *n*
apply (*induction n*) **using** *H*(*1*) **by** *auto*
then have (λn. ((*Gromov-extension f*)^^*n*) *xi*) $\longrightarrow$ *xi*
by *auto*
then show *?thesis*
using *loxodromic-attracting-fixed-point-attracts*[*OF assms H*(*2*)] *LIMSEQ-unique* **by** *auto*
qed
then show *?thesis*
using *loxodromic-attracting-fixed-point*(*2*)[*OF assms*] *loxodromic-repelling-fixed-point*(*2*)[*OF assms*] **by** *auto*

qed

end

References

[BH99] Martin R. Bridson and André Haefliger, *Metric spaces of non-positive curvature*, Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], vol. 319, Springer-Verlag, Berlin, 1999. MR MR1744486

[BS00] Mario Bonk and Oded Schramm, *Embeddings of Gromov hyperbolic spaces*, Geom. Funct. Anal. **10** (2000), 266–306. MR MR1771428

[CDP90] Michel Coornaert, Thomas Delzant, and Athanase Papadopoulos, *Parties quasi-convexes d'un espace hyperbolique*, pp. 106–123, Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[GdlH90] Étienne Ghys and Pierre de la Harpe (eds.), *Sur les groupes hyperboliques d'après Mikhael Gromov*, Progress in Mathematics, vol. 83, Birkhäuser Boston Inc., Boston, MA, 1990, Papers from the Swiss Seminar on Hyperbolic Groups held in Bern, 1988. MR MR1086648

[Min05] Igor Mineyev, *Flows and joins of metric spaces*, Geom. Topol. **9** (2005), 403–482. MR 2140987

[Shc13] Vladimir Shchur, *A quantitative version of the Morse lemma and quasi-isometries fixing the ideal boundary*, J. Funct. Anal. **264** (2013), no. 3, 815–836. MR 3003738