The Kolmogorov-Chentsov Theorem

Christian Pardillo Laursen and Simon Foster
University of York, UK

{christian.laursen,simon.foster}@york.ac.uk

February 6, 2026

Abstract

Continuous-time stochastic processes often carry the condition of having almost-surely continuous paths. If some process X satisfies certain bounds on its expectation, then the Kolmogorov-Chentsov theorem lets us construct a modification of X, i.e. a process X' such that $\forall t. X_t = X'_t$ almost surely, that has Hölder continuous paths.

In this work, we mechanise the Kolmogorov-Chentsov theorem. To get there, we develop a theory of stochastic processes, together with Hölder continuity, convergence in measure, and arbitrary intervals of dyadic rationals.

With this, we pave the way towards a construction of Brownian motion. The work is based on the exposition in Achim Klenke's probability theory text [1].

Contents

1 Supporting lemmas **2**

2 Intervals of dyadic rationals **6**

3 Hölder continuity **22**

4 Convergence in measure **30**

5 Stochastic processes **40**

6 The Kolomgorov-Chentsov theorem **55**
6.1 Supporting lemmas 55
6.2 Kolmogorov-Chentsov 56

1 Supporting lemmas

theory *Kolmogorov-Chentsov-Extras*
imports *HOL−Probability.Probability*
begin

lemma *atLeastAtMost-induct*[*consumes 1*, *case-names base Suc*]:
assumes $x \in \{n..m\}$
and $P\ n$
and $\bigwedge k.\ [\![k \geq n;\ k < m;\ P\ k]\!] \Longrightarrow P\ (Suc\ k)$
shows $P\ x$
by (*smt* (*verit*, *ccfv-threshold*) *assms Suc-le-eq atLeastAtMost-iff dec-induct le-trans*)

lemma *eventually-prodI'*:
assumes *eventually P F eventually Q G* $\forall x\ y.\ P\ x \longrightarrow Q\ y \longrightarrow R\ (x,y)$
shows *eventually R* $(F \times_F G)$
using *assms eventually-prod-filter* **by** *blast*

Analogous to $[\![$*almost-everywhere ?M ?P*; $\bigwedge N.\ [\![\bigwedge x.\ x \in$ *space ?M* $- N \Longrightarrow$ *?P x*; $N \in$ *null-sets ?M*$]\!] \Longrightarrow$ *?thesis*$]\!] \Longrightarrow$ *?thesis*

lemma *AE-I3*:
assumes $\bigwedge x.\ x \in$ *space M* $- N \Longrightarrow P\ x\ N \in$ *null-sets M*
shows *AE x in M. P x*
by (*metis* (*no-types*, *lifting*) *assms DiffI eventually-ae-filter mem-Collect-eq subsetI*)

Extends $[\![(?f \longrightarrow ?l)\ ?F;\ (?g \longrightarrow ?m)\ ?F]\!] \Longrightarrow ((\lambda x.\ dist\ (?f\ x)\ (?g\ x)) \longrightarrow dist\ ?l\ ?m)\ ?F$

lemma *tendsto-dist-prod*:
fixes *l m* :: *'a* :: *metric-space*
assumes *f*: $(f \longrightarrow l)\ F$
and *g*: $(g \longrightarrow m)\ G$
shows $((\lambda x.\ dist\ (f\ (fst\ x))\ (g\ (snd\ x))) \longrightarrow dist\ l\ m)\ (F \times_F G)$
proof (*rule tendstoI*)
fix *e* :: *real* **assume** $0 < e$
then have *e2*: $0 < e/2$ **by** *simp*
show $\forall_F\ x\ in\ F \times_F G.\ dist\ (dist\ (f\ (fst\ x))\ (g\ (snd\ x)))\ (dist\ l\ m) < e$
using *tendstoD* [*OF f e2*] *tendstoD* [*OF g e2*] **apply** (*rule eventually-prodI'*)
apply *simp*
by (*smt* (*verit*) *dist-commute dist-norm dist-triangle real-norm-def*)
qed

lemma *borel-measurable-at-within-metric* [*measurable*]:
fixes *f* :: *'a* :: *first-countable-topology* $\Rightarrow$ *'b* $\Rightarrow$ *'c* :: *metric-space*
assumes [*measurable*]: $\bigwedge t.\ t \in S \Longrightarrow f\ t \in$ *borel-measurable M*
and *convergent*: $\bigwedge \omega.\ \omega \in$ *space M* $\Longrightarrow \exists\, l.\ ((\lambda t.\ f\ t\ \omega) \longrightarrow l)\ (at\ x\ within\ S)$
and *nontrivial*: *at x within S* $\neq \bot$
shows $(\lambda \omega.\ Lim\ (at\ x\ within\ S)\ (\lambda t.\ f\ t\ \omega)) \in$ *borel-measurable M*
proof −

obtain l **where** l: $\bigwedge\omega.\ \omega \in space\ M \Longrightarrow ((\lambda t.\ f\ t\ \omega) \longrightarrow l\ \omega)\ (at\ x\ within\ S)$
using *convergent* **by** *metis*
then have *Lim-eq*: $Lim\ (at\ x\ within\ S)\ (\lambda t.\ f\ t\ \omega) = l\ \omega$
if $\omega \in space\ M$ **for** ω
using *tendsto-Lim*[*OF nontrivial*] *that* **by** *blast*
from *nontrivial* **have** *1*: $x\ islimpt\ S$
using *trivial-limit-within* **by** *blast*
then obtain $s :: nat \Rightarrow 'a$ **where** s: $\bigwedge n.\ s\ n \in S - \{x\}\ s \longrightarrow x$
using *islimpt-sequential* **by** *blast*
then have $\bigwedge n.\ f\ (s\ n) \in borel\text{-}measurable\ M$
using s **by** *simp*
moreover have $(\lambda n.\ (f\ (s\ n)\ \omega)) \longrightarrow l\ \omega$ **if** $\omega \in space\ M$ **for** ω
using $s\ l$[*unfolded tendsto-at-iff-sequentially comp-def*, *OF that*]
by *blast*
ultimately have $l \in borel\text{-}measurable\ M$
by (*rule borel-measurable-LIMSEQ-metric*)
then show *?thesis*
using *measurable-cong*[*OF Lim-eq*] **by** *fast*
qed

lemma *Max-finite-image-ex*:
assumes *finite S* $S \neq \{\}$ *P* (*MAX k∈S. f k*)
shows $\exists k \in S.\ P\ (f\ k)$
using *bexI*[*of P Max* (*f ' S*) *f ' S*] **by** (*simp add*: *assms*)

lemma *measure-leq-emeasure-ennreal*: $0 \leq x \Longrightarrow emeasure\ M\ A \leq ennreal\ x \Longrightarrow measure\ M\ A \leq x$
apply (*cases* $A \in M$)
apply (*metis Sigma-Algebra.measure-def enn2real-leI*)
apply (*auto simp*: *measure-notin-sets emeasure-notin-sets*)
done

lemma *Union-add-subset*: $(m :: nat) \leq n \Longrightarrow (\bigcup k.\ A\ (k + n)) \subseteq (\bigcup k.\ A\ (k + m))$
apply (*subst subset-eq*)
apply *simp*
apply (*metis add.commute le-iff-add trans-le-add1*)
done

lemma *floor-in-Nats* [*simp*]: $x \geq 0 \Longrightarrow \lfloor x \rfloor \in \mathbb{N}$
by (*metis nat-0-le of-nat-in-Nats zero-le-floor*)

lemma *triangle-ineq-list*:
fixes $l :: ('a :: metric\text{-}space)\ list$
assumes $l \neq []$
shows $dist\ (hd\ l)\ (last\ l) \leq (\sum\ i{=}1..length\ l - 1.\ dist\ (l!i)\ (l!(i-1)))$
using *assms* **proof** (*induct l rule*: *rev-nonempty-induct*)
case (*single x*)
then show *?case* **by** *force*

next
case (*snoc x xs*)
define $S :: 'a\ list \Rightarrow real$ **where**
$S \equiv \lambda l.\ (\sum\ i{=}1..length\ l - 1.\ dist\ (l!i)\ (l!(i{-}1)))$
have $S\ (xs\ @\ [x]) = dist\ x\ (last\ xs) + S\ xs$
unfolding *S-def* **apply** *simp*
apply (*subst sum.last-plus*)
apply (*simp add*: *Suc-leI snoc.hyps*(*1*))
apply (*rule arg-cong2*[**where** *f*=(+)])
apply (*simp add*: *last-conv-nth nth-append snoc.hyps*(*1*))
apply (*rule sum.cong*)
apply *fastforce*
by (*smt* (*verit*) *Suc-pred atLeastAtMost-iff diff-is-0-eq less-Suc-eq nat-less-le not-less nth-append*)
moreover have *dist* (*hd xs*) (*last xs*) $\leq S\ xs$
unfolding *S-def* **using** *snoc* **by** *blast*
ultimately have *dist* (*hd xs*) $x \leq S\ (xs@[x])$
by (*smt* (*verit*) *dist-triangle2*)
then show *?case*
unfolding *S-def* **using** *snoc* **by** *simp*
qed

lemma *triangle-ineq-sum*:
fixes $f :: nat \Rightarrow 'a :: metric\text{-}space$
assumes $n \leq m$
shows *dist* (*f n*) (*f m*) $\leq (\sum\ i{=}Suc\ n..m.\ dist\ (f\ i)\ (f\ (i{-}1)))$
proof (*cases n*=*m*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then have $n < m$
using *assms* **by** *simp*
define *l* **where** $l \equiv map\ (f\ o\ nat)\ [n..m]$
have [*simp*]: $l \neq []$
using ‹$n < m$› *l-def* **by** *fastforce*
have [*simp*]: *length l* = *m* − *n* +*1*
unfolding *l-def* **apply** *simp*
using *assms* **by** *linarith*
with *l-def* **have** *hd l* = *f n*
by (*simp add*: *assms upto.simps*)
moreover have *last l* = *f m*
unfolding *l-def* **apply** (*subst last-map*)
using *assms* **apply** *force*
using ‹$n < m$› *upto-rec2* **by** *force*
ultimately have *dist* (*f n*) (*f m*) = *dist* (*hd l*) (*last l*)
by *simp*
also have ... $\leq$ $(\sum\ i{=}1..length\ l - 1.\ dist\ (l!i)\ (l!(i{-}1)))$
by (*rule triangle-ineq-list*[*OF* ‹$l \neq []$›])

```
  also have ... = (∑ i=Suc n..m. dist (f i) (f (i−1)))
    apply (rule sum.reindex-cong[symmetric, where l=(+) n])
    using inj-on-add apply blast
     apply (simp add: assms)
    apply (rule arg-cong2[where f=dist])
     apply simp-all
    unfolding l-def apply (subst nth-map, fastforce)
     apply (subst nth-upto, linarith)
    subgoal by simp (insert nat-int-add, presburger)
    apply (subst nth-map, fastforce)
    apply (subst nth-upto, linarith)
    by (simp add: add-diff-eq nat-int-add nat-diff-distrib')
  finally show ?thesis
    by blast
qed

lemma (in product-prob-space) indep-vars-PiM-coordinate:
  assumes I ≠ {}
  shows prob-space.indep-vars (Π_M i∈I. M i) M (λx f. f x) I
proof −
  have distr (Pi_M I M) (Pi_M I M) (λx. restrict x I) = distr (Pi_M I M) (Pi_M I
M) (λx. x)
    by (rule distr-cong; simp add: space-PiM)
  also have ... = PiM I M
    by simp
  also have ... = Pi_M I (λi. distr (Pi_M I M) (M i) (λf. f i))
    using PiM-component by (intro PiM-cong, auto)
  finally have distr (Pi_M I M) (Pi_M I M) (λx. restrict x I) = Pi_M I (λi. distr
(Pi_M I M) (M i) (λf. f i))
    .
  then show ?thesis
    using assms by (simp add: indep-vars-iff-distr-eq-PiM')
qed

lemma (in prob-space) indep-sets-indep-set:
  assumes indep-sets F I i ∈ I j ∈ I i ≠ j
  shows indep-set (F i) (F j)
  unfolding indep-set-def
proof (rule indep-setsI)
  show (case x of True ⇒ F i | False ⇒ F j) ⊆ events for x
    using assms by (auto split: bool.split simp: indep-sets-def)
  fix A J assume ∗: J ≠ {} J ⊆ UNIV ∀ja∈J. A ja ∈ (case ja of True ⇒ F i |
False ⇒ F j)
  {
    assume J = UNIV
    then have indep-sets F I {i,j} ⊆ I {i, j} ≠ {} finite {i,j} ∀x ∈ {i,j}. (λx. if
x = i then A True else A False) x ∈ F x
      using ∗ assms apply simp-all
      by (simp add: bool.split-sel)
```

then have *prob* ($\bigcap j{\in}\{i, j\}$. *if* $j = i$ *then* A *True else* A *False*) = ($\prod j{\in}\{i, j\}$. *prob* (*if* $j = i$ *then* A *True else* A *False*))
by (*rule indep-setsD*)
then have *prob* (A *True* $\cap$ A *False*) = *prob* (A *True*) $*$ *prob* (A *False*)
using *assms* **by** (*auto simp*: *ac-simps*)
} note X = *this*
consider J = {*True*, *False*} | J = {*False*} | J = {*True*}
using $*(1,2)$ **unfolding** *UNIV-bool* **by** *blast*
then show *prob* ($\bigcap$ (A ‘ J)) = ($\prod j{\in}J$. *prob* (A j))
using X **by** (*cases*; *auto*)
qed

lemma (**in** *prob-space*) *indep-vars-indep-var*:
assumes *indep-vars* M' X I $i \in I$ $j \in I$ $i \neq j$
shows *indep-var* (M' i) (X i) (M' j) (X j)
using *assms* **unfolding** *indep-vars-def indep-var-eq*
by (*meson indep-sets-indep-set*)

end

2 Intervals of dyadic rationals

theory *Dyadic-Interval*
imports *HOL−Analysis.Analysis*
begin

In this file we describe intervals of dyadic numbers $S..T$ for reals S T. We use the floor and ceiling functions to approximate the numbers with increasing accuracy.

lemma *frac-floor*: $\lfloor x \rfloor = x - frac\ x$
by (*simp add*: *frac-def*)

lemma *frac-ceil*: $\lceil x \rceil = x + frac\ (-\ x)$
apply (*cases* x = *real-of-int* $\lfloor x \rfloor$)
unfolding *ceiling-altdef* **apply** *simp*
apply (*metis Ints-minus Ints-of-int*)
apply (*simp add*: *frac-neg frac-floor*)
done

lemma *floor-pow2-lim*: (λn. $\lfloor 2\hat{}n * T \rfloor$ / $2\ \hat{}\ n$) $\longrightarrow$ T
proof (*intro LIMSEQ-I*)
fix r :: *real* **assume** $r > 0$
obtain k **where** k: 1 / $2\hat{}k < r$
by (*metis* ‹$r > 0$› *one-less-numeral-iff power-one-over reals-power-lt-ex semiring-norm*(*76*))
then have $\forall n{\geq}k$. *norm* ($\lfloor 2\ \hat{}\ n * T \rfloor$ / $2\ \hat{}\ n - T$) $< r$
apply (*simp add*: *frac-floor field-simps*)
by (*smt* (*verit*, *ccfv-SIG*) ‹$0 < r$› *frac-lt-1 mult-left-mono power-increasing*)

then show $\exists no.\ \forall n{\geq}no.\ norm\ (real\text{-}of\text{-}int\ \lfloor 2\ \hat{}\ n * T\rfloor\ /\ 2\ \hat{}\ n - T) < r$
by *blast*
qed

lemma *floor-pow2-leq*: $\lfloor 2\hat{}n * T\rfloor\ /\ 2\ \hat{}\ n \leq T$
by (*simp add*: *frac-floor field-simps*)

lemma *ceil-pow2-lim*: $(\lambda n.\ \lceil 2\hat{}n * T\rceil\ /\ 2\ \hat{}\ n) \longrightarrow T$
proof (*intro LIMSEQ-I*)
fix r :: *real* **assume** $r > 0$
obtain k **where** k: $1\ /\ 2\hat{}k < r$
by (*metis* ‹$r > 0$› *one-less-numeral-iff power-one-over reals-power-lt-ex semiring-norm*(*76*))
then have $\forall n{\geq}k.\ norm\ (\lceil 2\ \hat{}\ n * T\rceil\ /\ 2\ \hat{}\ n - T) < r$
apply (*simp add*: *frac-ceil field-simps*)
by (*smt* (*verit*) ‹$0 < r$› *frac-lt-1 mult-left-mono power-increasing*)
then show $\exists no.\ \forall n{\geq}no.\ norm\ (\lceil 2\ \hat{}\ n * T\rceil\ /\ 2\ \hat{}\ n - T) < r$
by *blast*
qed

lemma *ceil-pow2-geq*: $\lceil 2\hat{}n * T\rceil\ /\ 2\ \hat{}\ n \geq T$
by (*simp add*: *frac-ceil field-simps*)

`dyadic_interval_step` $n\ S\ T$ is the collection of dyadic numbers in $\{S..T\}$ with denominator 2^n. As $n \to \infty$ this collection approximates $\{S..T\}$. Compare with $dyadics \equiv \bigcup_{k\ m} \{of\text{-}nat\ m\ /\ (2::?'a)^k\}$

definition *dyadic-interval-step* :: *nat* ⇒ *real* ⇒ *real* ⇒ *real set*
where *dyadic-interval-step n S T* ≡ (λ*k*. *k* / (2 ^ *n*)) ‘ {⌈2^*n* * *S*⌉..⌊2^*n* * *T*⌋}

definition *dyadic-interval* :: *real* ⇒ *real* ⇒ *real set*
where *dyadic-interval S T* ≡ (⋃*n*. *dyadic-interval-step n S T*)

lemma *dyadic-interval-step-empty*[*simp*]: $T < S \Longrightarrow dyadic\text{-}interval\text{-}step\ n\ S\ T = \{\}$
unfolding *dyadic-interval-step-def* **apply** *simp*
by (*smt* (*verit*) *ceil-pow2-geq floor-le-ceiling floor-mono floor-pow2-leq linordered-comm-semiring-strict-class.comm-mult-strict-left-mono zero-less-power*)

lemma *dyadic-interval-step-singleton*[*simp*]: $X \in \mathbb{Z} \Longrightarrow dyadic\text{-}interval\text{-}step\ n\ X\ X = \{X\}$
proof −
assume $X \in \mathbb{Z}$
then have ∗: $\lfloor 2\ \hat{}\ k * X\rfloor = 2\ \hat{}\ k * X$ **for** k :: *nat*
by *simp*
then show *?thesis*
unfolding *dyadic-interval-step-def* **apply** (*simp add*: *ceiling-altdef*)
using ∗ **by** *presburger*
qed

lemma *dyadic-interval-step-zero* [*simp*]: *dyadic-interval-step 0 S T = real-of-int ‘ {⌈S⌉ .. ⌊T⌋}*
unfolding *dyadic-interval-step-def* **by** *simp*

lemma *dyadic-interval-step-mem* [*intro*]:
assumes*x ≥ 0 T ≥ 0 x ≤ T*
shows *⌊2^n * x⌋ / 2 ^ n ∈ dyadic-interval-step n 0 T*
unfolding *dyadic-interval-step-def* **by** (*simp add*: *assms image-iff floor-mono*)

lemma *dyadic-interval-step-iff*:
x ∈ dyadic-interval-step n S T ⟷
*(∃ k. k ≥ ⌈2^n * S⌉ ∧ k ≤ ⌊2^n * T⌋ ∧ x = k/2^n)*
unfolding *dyadic-interval-step-def* **by** (*auto simp add*: *image-iff*)

lemma *dyadic-interval-step-memI* [*intro*]:
assumes *∃ k::int. x = k/2^n x ≥ S x ≤ T*
shows *x ∈ dyadic-interval-step n S T*
proof −
obtain *k :: int* **where** *x = k/2^n*
using *assms(1)* **by** *blast*
then have *k*: *k = 2^n * x*
by *simp*
then have *k ≥ ⌈2^n * S⌉*
by (*simp add*: *assms(2) ceiling-le*)
moreover from *k* **have** *k ≤ ⌊2^n * T⌋*
by (*simp add*: *assms(3) le-floor-iff*)
ultimately show *?thesis*
using *dyadic-interval-step-iff ‹x = k / 2 ^ n›* **by** *blast*
qed

lemma *mem-dyadic-interval*: *x ∈ dyadic-interval S T ⟷ (∃ n. x ∈ dyadic-interval-step n S T)*
unfolding *dyadic-interval-def* **by** *blast*

lemma *mem-dyadic-intervalI*: *∃ n. x ∈ dyadic-interval-step n S T ⟹ x ∈ dyadic-interval S T*
using *mem-dyadic-interval* **by** *fast*

lemma *dyadic-step-leq*: *x ∈ dyadic-interval-step n S T ⟹ x ≤ T*
unfolding *dyadic-interval-step-def* **apply** *clarsimp*
by (*simp add*: *divide-le-eq le-floor-iff mult.commute*)

lemma *dyadics-leq*: *x ∈ dyadic-interval S T ⟹ x ≤ T*
using *dyadic-step-leq mem-dyadic-interval* **by** *blast*

lemma *dyadic-step-geq*: *x ∈ dyadic-interval-step n S T ⟹ x ≥ S*
unfolding *dyadic-interval-step-def* **apply** *clarsimp*
by (*simp add*: *ceiling-le-iff mult.commute pos-le-divide-eq*)

lemma *dyadics-geq*: $x \in$ *dyadic-interval* $S\ T \Longrightarrow x \geq S$
using *dyadic-step-geq mem-dyadic-interval* **by** *blast*

corollary *dyadic-interval-subset-interval* [*simp*]: (*dyadic-interval* $0\ T$) $\subseteq \{0..T\}$
using *dyadics-geq dyadics-leq* **by** *force*

lemma *zero-in-dyadics*: $T \geq 0 \Longrightarrow 0 \in$ *dyadic-interval-step* $n\ 0\ T$
using *dyadic-interval-step-def* **by** *force*

The following theorem is useful for reasoning with at_within

lemma *dyadic-interval-converging-sequence*:
assumes $t \in \{0..T\}\ T \neq 0$
shows $\exists s.\ \forall n.\ s\ n \in$ *dyadic-interval* $0\ T - \{t\} \wedge s \longrightarrow t$
proof −
from *assms* **have** $T > 0$
by *auto*
consider (*eq-0*) $t = 0$ | (*dyadic*) $t \in$ *dyadic-interval* $0\ T - \{0\}$ | (*real*) $t \notin$ *dyadic-interval* $0\ T$
by *blast*
then show *?thesis*
proof *cases*
case *eq-0*
obtain n **where** $1 \leq 2\ \hat{}\ n * T$
proof −
assume $*$: $\bigwedge n.\ 1 \leq 2\ \hat{}\ n * T \Longrightarrow$ *thesis*
obtain n **where** $2\ \hat{}\ n > 1/T$
using *real-arch-pow* **by** *fastforce*
then have $2\ \hat{}\ n * T \geq 1$
using ‹$T > 0$› **by** (*simp add*: *field-simps*)
then show *?thesis*
using $*$ **by** *blast*
qed
define s :: *nat* ⇒ *real* **where** $s = (\lambda m.\ 1/2\hat{}(m + n))$
have $\forall m.\ s\ m \in$ *dyadic-interval-step* $(m + n)\ 0\ T - \{0\}$
unfolding *s-def* **apply** (*simp add*: *dyadic-interval-step-iff*)
using ‹$1 \leq 2\ \hat{}\ n * T$›
by (*smt* (*verit*, *best*) ‹$0 < T$› *le-add2 mult-right-mono power-increasing-iff*)
then have $\forall m.\ s\ m \in$ *dyadic-interval* $0\ T - \{0\}$
using *mem-dyadic-interval* **by** *auto*
moreover **{**
have $(\lambda m.\ (1::real)/2\hat{}m) \longrightarrow 0$
by (*simp add*: *divide-real-def LIMSEQ-inverse-realpow-zero*)
then have $s \longrightarrow 0$
unfolding *s-def* **using** *LIMSEQ-ignore-initial-segment* **by** *auto*
}
ultimately show *?thesis*
using *eq-0* **by** *blast*
next
case *dyadic*

```
  then have t ≠ 0
    by blast
  from dyadic obtain n where n: t ∈ dyadic-interval-step n 0 T
    by (auto simp: mem-dyadic-interval)
  then obtain k :: int where k: t = k / 2^n k ≤ ⌊2 ^ n * T⌋
    using dyadic-interval-step-iff by blast
  then have k > 0
    using ‹t ≠ 0› dyadic-interval-step-iff n by force
  define s :: nat ⇒ real where s ≡ λm. (k * 2^(m+1) - 1) / 2 ^ (m + n + 1)
  have s m ∈ dyadic-interval-step (m + n + 1) 0 T for m
  proof -
    have k * (2 ^ (m+1)) - 1 ≤ ⌊2 ^ n * T⌋ * (2 ^ (m+1)) - 1
      by (smt (verit) k(2) mult-right-mono zero-le-power)
    also have ... ≤ ⌊2 ^ n * T⌋ * ⌊(2 ^ (m+1))⌋
    by (metis add.commute add-le-cancel-left diff-add-cancel diff-self floor-numeral-power
        zero-less-one-class.zero-le-one)
    also have ⌊2 ^ n * T⌋ * ⌊(2 ^ (m+1))⌋ ≤ ⌊2 ^ n * T * (2 ^ (m+1))⌋
     by (smt (z3) ‹0 < T› floor-one floor-power le-mult-floor mult-nonneg-nonneg
of-int-1
        of-int-add one-add-floor one-add-one zero-le-power)
    also have ... = ⌊(2 ^ (m+n+1)) * T⌋
      apply (rule arg-cong[where f=floor])
      by (simp add: power-add)
    finally show ?thesis
      unfolding s-def apply (simp only: dyadic-interval-step-iff)
      apply (rule exI[where x=k * (2 ^ (m+1)) - 1])
      by (simp add: ‹0 < k›)
  qed
  then have s m ∈ dyadic-interval 0 T for m
    using mem-dyadic-interval by blast
  moreover have s m ≠ t for m
    unfolding s-def k(1) by (simp add: power-add field-simps)
  moreover have s ⟶ t
  proof
    fix e :: real assume 0 < e
    then obtain m where 1 / 2 ^ m < e
      by (metis one-less-numeral-iff power-one-over reals-power-lt-ex semir-
ing-norm(76))
    { fix m' assume m' ≥ m
      then have 1 / 2 ^ m' < e
        using ‹1/2^m < e›
      by (smt (verit) frac-less2 le-eq-less-or-eq power-strict-increasing zero-less-power)
      then have 1/ 2^(m'+n+1) < e
      by (smt (verit, ccfv-SIG) divide-less-eq-1-pos half-gt-zero-iff power-less-imp-less-exp

          power-one-over power-strict-decreasing trans-less-add1)
      have s m' - t = (k * 2^(m'+1) - 1) / 2 ^ (m' + n + 1) - k / 2 ^ n
        by (simp add: s-def k(1))
      also have ... = ((k * 2 ^ (m' + 1) - 1) - (k * 2 ^(m'+1))) / 2 ^ (m' +
```

```
n + 1)
        by (simp add: field-simps power-add)
      also have ... = -1/ 2^(m'+n+1)
        by (simp add: field-simps)
      finally have dist (s m') t < e
        unfolding s-def k(1)
        apply (simp add: dist-real-def)
        using ‹1 / 2 ^ (m' + n + 1) < e› by auto
    }
    then show ∀F x in sequentially. dist (s x) t < e
      apply (simp add: eventually-sequentially)
      apply (intro exI[where x=m])
      by simp
  qed
  ultimately show ?thesis
    by blast
 next
  case real
  then obtain n where dyadic-interval-step n 0 T ≠ {}
    by (metis ‹0 < T› empty-iff less-eq-real-def zero-in-dyadics)
  define s :: nat ⇒ real where s ≡ λm. ⌊2^(m+n) * t⌋ / 2 ^ (m+n)
  have s m ∈ dyadic-interval-step (m+n) 0 T for m
    unfolding s-def
      by (metis assms(1) atLeastAtMost-iff ceiling-zero dyadic-interval-step-iff
floor-mono
        mult.commute mult-eq-0-iff mult-right-mono zero-le-floor zero-le-numeral
zero-le-power)
  then have s m ∈ dyadic-interval 0 T for m
    using mem-dyadic-interval by blast
  moreover have s ⟶ t
      unfolding s-def using LIMSEQ-ignore-initial-segment floor-pow2-lim by
blast
  ultimately show ?thesis
    using real by blast
 qed
qed

lemma dyadic-interval-dense: closure (dyadic-interval 0 T) = {0..T}
proof (rule subset-antisym)
 have (dyadic-interval 0 T) ⊆ {0..T}
   by (fact dyadic-interval-subset-interval)
 then show closure (dyadic-interval 0 T) ⊆ {0..T}
   by (auto simp: closure-minimal)
 have {0..T} ⊆ closure (dyadic-interval 0 T) if T ≥ 0
   unfolding closure-def
 proof −
   {
    fix x assume x: 0 ≤ x x ≤ T x ∉ dyadic-interval 0 T
    then have x > 0
```

```
      unfolding dyadic-interval-def
      using zero-in-dyadics[OF that] order-le-less by blast
    have x islimpt (dyadic-interval 0 T)
      apply (simp add: islimpt-sequential)
      apply (rule exI [where x=λn. ⌊2^n * x⌋ / 2^n])
      apply safe
      using dyadic-interval-step-mem mem-dyadic-interval x(1,2) apply auto[1]
     apply (smt (verit, ccfv-threshold) dyadic-interval-step-mem mem-dyadic-interval
x)
      using floor-pow2-lim apply blast
      done
  }
  thus {0..T} ⊆ dyadic-interval 0 T ∪ {x. x islimpt dyadic-interval 0 T}
    by force
 qed
 then show {0..T} ⊆ closure (dyadic-interval 0 T)
  by (cases T ≥ 0; simp)
qed

corollary dyadic-interval-islimpt:
 assumes T > 0 t ∈ {0..T}
 shows t islimpt dyadic-interval 0 T
 using assms by (subst limpt-of-closure[symmetric], simp add: dyadic-interval-dense)

corollary at-within-dyadic-interval-nontrivial[simp]:
 assumes T > 0 t ∈ {0..T}
 shows (at t within dyadic-interval 0 T) ≠ bot
 using assms dyadic-interval-islimpt trivial-limit-within by blast

lemma dyadic-interval-step-finite[simp]: finite (dyadic-interval-step n S T)
 unfolding dyadic-interval-step-def by simp

lemma dyadic-interval-countable[simp]: countable (dyadic-interval S T)
 by (simp add: dyadic-interval-def dyadic-interval-step-def)

lemma floor-pow2-add-leq:
 fixes T :: real
 shows ⌊2^n * T⌋ / 2 ^ n ≤ ⌊2 ^ (n+k) * T⌋ / 2 ^ (n+k)
proof (induction k)
 case 0
 then show ?case by simp
next
 case (Suc k)
 let ?f = frac (2 ^ (n + k) * T)
  and ?f' = frac (2 ^ (n + (Suc k)) * T)
 show ?case
 proof (cases ?f < 1/2)
  case True
  then have ?f + ?f < 1
```

```
    by auto
  then have frac ((2 ^ (n + k) * T) + (2 ^ (n + k) * T)) = ?f + ?f
    using frac-add by meson
  then have ?f' = ?f + ?f
    by (simp add: field-simps)
  then have ⌊2 ^ (n + Suc k) * T⌋ / 2 ^ (n + Suc k) = ⌊2^(n + k) * T⌋ / 2
^ (n + k)
    by (simp add: frac-def)
  then show ?thesis
    using Suc by presburger
 next
  case False
  have ?f' = frac (2 ^ (n + k) * T + 2 ^ (n + k) * T)
    by (simp add: field-simps)
  then have ?f' = 2 * ?f - 1
    by (smt (verit, del-insts) frac-add False field-sum-of-halves)
  then have ?f' < ?f
    using frac-lt-1 by auto
  then have (2 ^ (n + k) * T - ?f) / 2 ^ (n + k) < (2 ^ (n + (Suc k)) * T
- ?f') / 2 ^ (n + Suc k)
    apply (simp add: field-simps)
    by (smt (verit, ccfv-threshold) frac-ge-0)
  then show ?thesis
    by (smt (verit, ccfv-SIG) Suc frac-def)
 qed
qed

corollary floor-pow2-mono: mono (λn. ⌊2^n * (T :: real)⌋ / 2 ^ n)
 apply (intro monoI)
 subgoal for x y
  using floor-pow2-add-leq[of x T y - x] by force
 done

lemma dyadic-interval-step-Max: T ≥ 0 ⟹ Max (dyadic-interval-step n 0 T) =
⌊2^n * T⌋ / 2^n
 apply (simp add: dyadic-interval-step-def)
 apply (subst mono-Max-commute[of λx. real-of-int x / 2^n, symmetric])
 by (auto simp: mono-def field-simps Max-eq-iff)

lemma dyadic-interval-step-subset:
 n ≤ m ⟹ dyadic-interval-step n 0 T ⊆ dyadic-interval-step m 0 T
proof (rule subsetI)
 fix x assume n ≤ m x ∈ dyadic-interval-step n 0 T
 then obtain k where k: k ≥ 0 k ≤ ⌊2^n * T⌋ x = k / 2^n
  unfolding dyadic-interval-step-def by fastforce
 then have k * 2 ^ (m - n) ∈ {0 .. ⌊2^m * T⌋}
 proof -
  have k / 2 ^ n ≤ ⌊2^m * T⌋ / 2 ^ m
   by (smt floor-pow2-mono[THEN monoD, OF ‹n ≤ m›] k(2) divide-right-mono
```

of-int-le-iff zero-le-power)
    then have $k / 2 \hat{} n * 2 \hat{} m \leq \lfloor 2\hat{}m * T \rfloor$
      by (*simp add*: *field-simps*)
    moreover have $k / 2 \hat{} n * 2 \hat{} m = k * 2 \hat{} (m - n)$
      apply (*simp add*: *field-simps*)
      apply (*metis* ‹$n \leq m$› *add-diff-inverse-nat not-less power-add*)
      done
    ultimately have $k * 2 \hat{} (m - n) \leq \lfloor 2\hat{}m * T \rfloor$
      by *linarith*
    then show $k * 2 \hat{} (m - n) \in \{0 .. \lfloor 2\hat{}m * T \rfloor\}$
      using *k*(*1*) **by** *simp*
  qed
  then show $x \in$ *dyadic-interval-step m 0 T*
    apply (*subst dyadic-interval-step-iff*)
    apply (*rule exI*[**where** $x{=}k * 2 \hat{} (m - n)$])
    apply *simp*
    apply (*simp add*: ‹$n \leq m$› *k*(*3*) *power-diff*)
    done
qed

corollary *dyadic-interval-step-mono*:
  assumes $x \in$ *dyadic-interval-step n 0 T* $n \leq m$
  shows $x \in$ *dyadic-interval-step m 0 T*
  using *assms dyadic-interval-step-subset* **by** *blast*

lemma *dyadic-as-natural*:
  assumes $x \in$ *dyadic-interval-step n 0 T*
  shows $\exists! k.\ x =$ *real* $k / 2 \hat{} n$
  using *assms*
proof (*induct n*)
  case *0*
  then show *?case*
    apply *simp*
   by (*metis 0 ceiling-zero div-by-1 dyadic-interval-step-iff mult-not-zero of-nat-eq-iff of-nat-nat power.simps*(*1*))
next
  case (*Suc n*)
  then show *?case*
    by (*auto simp*: *dyadic-interval-step-iff*, *metis of-nat-nat*)
qed

lemma *dyadic-of-natural*:
  assumes *real* $k / 2 \hat{} n \leq T$
  shows *real* $k / 2 \hat{} n \in$ *dyadic-interval-step n 0 T*
  using *assms* **apply** (*induct n*)
   apply *simp*
   apply (*metis atLeastAtMost-iff imageI le-floor-iff of-int-of-nat-eq of-nat-0-le-iff*)
  apply (*simp add*: *dyadic-interval-step-iff*)
   by (*smt* (*verit*, *ccfv-SIG*) *divide-le-eq le-floor-iff mult.commute of-int-of-nat-eq*

of-nat-0-le-iff zero-less-power)

lemma *dyadic-interval-minus*:
assumes $x \in$ *dyadic-interval-step n 0 T* $y \in$ *dyadic-interval-step n 0 T* $x \leq y$
shows $y - x \in$ *dyadic-interval-step n 0 T*
proof −
obtain *kx* :: *nat* **where** $x =$ *real kx / 2* $\hat{}$ *n*
using *dyadic-as-natural assms*(*1*) **by** *blast*
obtain *ky* :: *nat* **where** $y =$ *real ky / 2* $\hat{}$ *n*
using *dyadic-as-natural assms*(*2*) **by** *blast*
then have $y - x = (ky - kx)$ */ 2*$\hat{}$*n*
by (*smt* (*verit*, *ccfv-SIG*) ‹$x =$ *real kx / 2* $\hat{}$ *n*› *add-diff-inverse-nat add-divide-distrib*

assms(*3*) *divide-strict-right-mono of-nat-add of-nat-less-iff zero-less-power*)
then show *?thesis*
using *dyadic-of-natural*
by (*smt* (*verit*, *best*) *assms*(*1*,*2*) *dyadic-step-geq dyadic-step-leq*)
qed

lemma *dyadic-times-nat*: $x \in$ *dyadic-interval-step n 0 T* $\Longrightarrow (x * 2 \hat{} n) \in \mathbb{N}$
using *dyadic-as-natural* **by** *fastforce*

definition *dyadic-expansion x n b k* $\equiv$ *set b* $\subseteq \{0,1\}$
$\wedge$ *length b* $= n \wedge x =$ *real-of-int k* $+ (\sum m \in \{1..n\}.$ *real* $(b\,!\,(m-1))$ */ 2* $\hat{}$ *m*)

lemma *dyadic-expansionI*:
assumes *set b* $\subseteq \{0,1\}$ *length b* $= n$ *x* $= k + (\sum m \in \{1..n\}.\ (b\,!\,(m-1))$ */ 2* $\hat{}$ *m*)
shows *dyadic-expansion x n b k*
unfolding *dyadic-expansion-def* **using** *assms* **by** *blast*

lemma *dyadic-expansionD*:
assumes *dyadic-expansion x n b k*
shows *set b* $\subseteq \{0,1\}$
and *length b* $= n$
and *x* $= k + (\sum m \in \{1..n\}.\ (b\,!\,(m-1))$ */ 2* $\hat{}$ *m*)
using *assms* **unfolding** *dyadic-expansion-def* **by** *simp-all*

lemma *dyadic-expansion-ex*:
assumes $x \in$ *dyadic-interval-step n 0 T*
shows $\exists$ *b k. dyadic-expansion x n b k*
using *assms*
proof (*induction n arbitrary*: *x*)
case *0*
then show *?case*
unfolding *dyadic-expansion-def* **by** *force*
next
case (*Suc n*)
then obtain *k* **where** *k*: $k \in \{0..\lfloor 2 \hat{} (Suc\ n) * T \rfloor\}$ $x = k$ */ 2* $\hat{}$ *(Suc n)*

```
    unfolding dyadic-interval-step-def by fastforce
  then have div2: k div 2 ∈ {0..⌊2 ^ n * T⌋}
    using k(1) apply simp
    by (metis divide-le-eq-numeral1(1) floor-divide-of-int-eq floor-mono le-floor-iff
mult.assoc mult.commute of-int-numeral)
  then show ?case
  proof (cases even k)
    case True
    then have x = k div 2 / 2 ^ n
      by (simp add: k(2) real-of-int-div)
    then have x ∈ dyadic-interval-step n 0 T
      using dyadic-interval-step-def div2 by force
    then obtain k' b where kb: dyadic-expansion x n b k'
      using Suc(1) by blast
    show ?thesis
      apply (rule exI[where x=b @ [0]])
      apply (rule exI[where x=k'])
      unfolding dyadic-expansion-def apply safe
      using kb unfolding dyadic-expansion-def apply simp-all
       apply (auto intro!: sum.cong simp: nth-append)
      done
  next
    case False
    then have k = 2 * (k div 2) + 1
      by force
    then have x = k div 2 / 2 ^ n + 1 / 2 ^ (Suc n)
      by (simp add: k(2) field-simps)
    then have x - 1 / 2 ^ (Suc n) ∈ dyadic-interval-step n 0 T
      using div2 by (simp add: dyadic-interval-step-def)
    then obtain k' b where kb: dyadic-expansion (x-1/2^Suc n) n b k'
      using Suc(1)[of x - 1 / 2 ^ (Suc n)] by blast
    have x: x = real-of-int k' + (∑ m = 1..n. b ! (m-1) / 2 ^ m) + 1/2^Suc n
      using dyadic-expansionD(3)[OF kb] by (simp add: field-simps)
    show ?thesis
      apply (rule exI[where x=b @ [1]])
      apply (rule exI[where x=k'])
      unfolding dyadic-expansion-def apply safe
      using kb x unfolding dyadic-expansion-def apply simp-all
       apply (auto intro!: sum.cong simp: nth-append)
      done
  qed
qed

lemma dyadic-expansion-frac-le-1:
  assumes dyadic-expansion x n b k
  shows (∑ m∈{1..n}. (b ! (m-1)) / 2 ^ m) < 1
proof -
  have b ! (m - 1) ∈ {0,1} if m ∈ {1..n} for m
  proof -
```

```
    from assms have set b ⊆ {0,1} length b = n
      unfolding dyadic-expansion-def by blast+
    then have a < n ⟹ b ! a ∈ {0,1} for a
      using nth-mem by blast
    moreover have m − 1 < n
      using that by force
    ultimately show ?thesis
      by blast
  qed
  then have (∑ m∈{1..n}. (b ! (m−1)) / 2 ^ m) ≤ (∑ m∈{1..n}. 1 / 2 ^ m)
    apply (intro sum-mono)
    using assms by fastforce
  also have ... = 1 − 1/2^n
    by (induct n, auto)
  finally show ?thesis
   by (smt (verit, ccfv-SIG) add-divide-distrib divide-strict-right-mono zero-less-power)
qed

lemma dyadic-expansion-frac-range:
  assumes dyadic-expansion x n b k m ∈ {1..n}
  shows b ! (m−1) ∈ {0,1}
proof −
  have m − 1 < length b
    using dyadic-expansionD(2)[OF assms(1)] assms(2) by fastforce
  then show ?thesis
    using nth-mem dyadic-expansionD(1)[OF assms(1)] by blast
qed

lemma dyadic-expansion-interval:
  assumes dyadic-expansion x n b k x ∈ {S..T}
  shows x ∈ dyadic-interval-step n S T
proof (subst dyadic-interval-step-iff, intro exI, safe)
  define k' where k' ≡ k * 2^n + (∑ i = 1..n. b!(i−1) * 2^(n−i))
  show x = k' / 2^n
    apply (simp add: dyadic-expansionD(3)[OF assms(1)] k'-def add-divide-distrib
sum-divide-distrib)
    apply (intro sum.cong, simp)
    apply (simp add: field-simps)
    by (metis add-diff-inverse-nat linorder-not-le power-add)
  then have k' = ⌊2^n * x⌋
    by simp
  then show k' ≤ ⌊2 ^ n * T⌋
    using assms(2) by (auto intro!: floor-mono mult-left-mono)
  from ‹x = k'/2^n› have k' = ⌈2^n * x⌉
    by force
  then show ⌈2 ^ n * S⌉ ≤ k'
    using assms(2) by (auto intro!: ceiling-mono mult-left-mono)
qed
```

lemma *dyadic-expansion-nth-geq*:
assumes *dyadic-expansion x n b k m* ∈ {*1..n*} *b* ! (*m*−*1*) = *1*
shows *x* ≥ *k* + *1*/*2*^*m*
proof −
have (∑ *i* = *1..n*. *f i*) = *f m* + (∑ *i* ∈ ({*1..n*} − {*m*}). *f i*) **for** *f* :: *nat* ⇒ *real*
by (*meson assms*(*2*) *finite-atLeastAtMost sum.remove*)
with *dyadic-expansionD*(*3*)[*OF assms*(*1*)] *assms*(*2*,*3*)
have *x* = *k* + *b*!(*m*−*1*)/*2*^*m* + (∑ *i* ∈ ({*1..n*} − {*m*}). *b* ! (*i*−*1*) / *2*^*i*)
by *simp*
moreover have (∑ *i* ∈ ({*1..n*} − {*m*}). *b* ! (*i*−*1*) / *2*^*i*) ≥ *0*
by (*simp add*: *sum-nonneg*)
ultimately show *?thesis*
using *assms*(*3*) **by** *fastforce*
qed

lemma *dyadic-expansion-frac-geq-0*:
assumes *dyadic-expansion x n b k*
shows (∑ *m*∈{*1..n*}. (*b* ! (*m*−*1*)) / *2* ^ *m*) ≥ *0*
proof −
have *b* ! (*m* − *1*) ∈ {*0*,*1*} **if** *m* ∈ {*1..n*} **for** *m*
using *dyadic-expansion-frac-range*[*OF assms*] *that* **by** *blast*
then have (∑ *m*∈{*1..n*}. (*b* ! (*m*−*1*)) / *2* ^ *m*) ≥ (∑ *m*∈{*1..n*}. *0*)
by (*intro sum-mono*, *fastforce*)
then show *?thesis*
by *auto*
qed

lemma *dyadic-expansion-frac*:
assumes *dyadic-expansion x n b k*
shows *frac x* = (∑ *m*∈{*1..n*}. (*b* ! (*m*−*1*))/ *2* ^ *m*)
apply (*simp add*: *frac-unique-iff*)
apply *safe*
using *dyadic-expansionD*(*3*)[*OF assms*] **apply** *simp*
using *dyadic-expansion-frac-geq-0*[*OF assms*] **apply** *simp*
using *dyadic-expansion-frac-le-1*[*OF assms*] **apply** *simp*
done

lemma *dyadic-expansion-floor*:
assumes *dyadic-expansion x n b k*
shows *k* = ⌊*x*⌋
proof −
have *x* = *k* + (∑ *m*∈{*1..n*}. (*b* ! (*m*−*1*))/ *2* ^ *m*)
using *assms* **by** (*rule dyadic-expansionD*(*3*))
then have *x* = *k* + *frac x*
using *dyadic-expansion-frac*[*OF assms*] **by** *linarith*
then have *k* = *x* − *frac x*
by *simp*
then show *k* = ⌊*x*⌋

```
  by (metis floor-of-int frac-floor)
qed

lemma sum-interval-pow2-inv: (∑ m∈{Suc l..n}. (1 :: real) / 2 ^ m) = 1 / 2^l
- 1/2^n if l < n
  using that proof (induct l)
  case 0
  then show ?case
    by (induct n; fastforce)
next
  case (Suc l)
  have (∑ m∈{Suc l..n} - {Suc l}. (1::real) / 2 ^ m) = (∑ m = Suc l..n. 1 / 2
^ m) - 1 / 2 ^ Suc l
    using Suc by (auto simp add: Suc sum-diff1, linarith)
  moreover have {Suc l..n} - {Suc l} = {Suc (Suc l)..n}
    by fastforce
  ultimately have (∑ m = Suc (Suc l)..n. (1::real) / 2 ^ m) = (∑ m = (Suc
l)..n. 1 / 2 ^ m) - 1 / 2^(Suc l)
    by force
  also have ... = 1 / 2 ^ l - 1 / 2 ^ n - 1 / 2^(Suc l)
    using Suc by linarith
  also have ... = 1 / 2 ^ Suc l - 1 / 2 ^ n
    by (simp add: field-simps)
  finally show ?case
    by blast
qed

lemma dyadic-expansion-unique:
  assumes dyadic-expansion x n b k
    and dyadic-expansion x n c j
  shows b = c ∧ j = k
proof (safe, rule ccontr)
  show j = k
    using assms dyadic-expansion-floor by blast
  assume b ≠ c
  have eq: (∑ m∈{1..n}. (b ! (m-1)) / 2 ^ m) = (∑ m∈{1..n}. (c ! (m-1)) / 2
^ m)
  proof -
     have k + (∑ m∈{1..n}. (b ! (m-1)) / 2 ^ m) = j + (∑ m∈{1..n}. (c !
(m-1)) / 2 ^ m)
      using assms dyadic-expansionD(3) by blast
    then show ?thesis
      using ‹j = k› by linarith
  qed
  have ex: ∃ l < n. b ! l ≠ c ! l
    by (metis list-eq-iff-nth-eq assms ‹b ≠ c› dyadic-expansionD(2))
  define l where l ≡ LEAST l. l < n ∧ b ! l ≠ c ! l
  then have l: l < n b ! l ≠ c ! l
    unfolding l-def using LeastI-ex[OF ex] by blast+
```

```
  have less-l: b ! k = c ! k if ‹k < l› for k
  proof −
    have k < n
      using that l by linarith
    then show b ! k = c ! k
      using that unfolding l-def using not-less-Least by blast
  qed
  then have l ∈ {0..n−1}
    using l by simp
  then have l < n
    apply (simp add: algebra-simps)
    using ex by fastforce
  then have b ! l ∈ {0,1} c ! l ∈ {0,1}
   by (metis assms insert-absorb insert-subset dyadic-expansionD(1,2) nth-mem)+
  then consider b ! l = 0 ∧ c ! l = 1 | b ! l = 1 ∧ c ! l = 0
    by (smt (verit) LeastI-ex emptyE insertE l-def ex)
 then have sum-ge-l-noteq:(∑ m∈{l+1..n}. (b ! (m−1)) / 2 ^ m) ≠ (∑ m∈{l+1..n}.
(c ! (m−1)) / 2 ^ m)
  proof cases
    case 1
    have ∗: ?thesis if l + 1 = n
      using that 1 by auto
    {
      assume ‹l + 1 < n›
      have (∑ m∈{l+1..n}. (c ! (m−1)) / 2 ^ m) =
          (c ! ((l+1)−1)) / 2 ^ (l+1) + (∑ m∈{Suc (l+1)..n}. (c ! (m−1)) / 2 ^
m)
        by (smt (verit, ccfv-SIG) Suc-eq-plus1 Suc-le-mono Suc-pred' ‹l ∈ {0..n −
1}›
         atLeastAtMost-iff bot-nat-0.not-eq-extremum ex order-less-trans sum.atLeast-Suc-atMost)
      also have ... ≥ 1 / 2 ^ (l+1)
        apply (simp add: 1)
        apply (rule sum-nonneg)
        using dyadic-expansion-frac-range[OF assms(2)] by simp
      finally have c-ge: (∑ m∈{l+1..n}. (c ! (m−1)) / 2 ^ m) ≥ 1/2^(l+1) .
      have (∑ m∈{l+1..n}. (b ! (m−1)) / 2 ^ m) =
          (b ! ((l+1)−1)) / 2 ^ (l+1) + (∑ m∈{Suc (l+1)..n}. (b ! (m−1)) / 2
^ m)
        by (meson ‹l + 1 < n› nat-less-le sum.atLeast-Suc-atMost)
      also have ... = (∑ m∈{Suc (l+1)..n}. (b ! (m−1)) / 2 ^ m)
        using 1 by auto
      also have ... ≤ (∑ m∈{Suc (l+1)..n}. 1 / 2 ^ m)
        apply (rule sum-mono)
      using dyadic-expansion-frac-range[OF assms(1)] apply (simp add: field-simps)
        by (metis (no-types, lifting) One-nat-def add-leE nle-le plus-1-eq-Suc)
      also have ... < 1 / 2 ^ (l+1)
        using sum-interval-pow2-inv[OF ‹l + 1 < n›] by fastforce
      finally have (∑ m∈{l+1..n}. (b ! (m−1)) / 2 ^ m) < 1 / 2 ^ (l+1) .
      with c-ge have ?thesis
```

```
      by argo
    }
    then show ?thesis
      using ∗ ‹l < n› by linarith
  next
    case 2
    have ∗: ?thesis if l + 1 = n
      using that 2 by auto
    {
      assume ‹l + 1 < n›
      have (∑ m∈{l+1..n}. (b ! (m−1)) / 2 ^ m) =
          (b ! ((l+1)−1)) / 2 ^ (l+1) + (∑ m∈{Suc (l+1)..n}. (b ! (m−1)) / 2 ^
m)
        by (meson ‹l + 1 < n› nat-less-le sum.atLeast-Suc-atMost)
      also have ... ≥ 1 / 2 ^ (l+1)
        apply (simp add: 2)
        apply (rule sum-nonneg)
        using dyadic-expansion-frac-range[OF assms(1)] by simp
      finally have b-ge: (∑ m∈{l+1..n}. (b ! (m−1)) / 2 ^ m) ≥ 1/2^(l+1) .
      have (∑ m∈{l+1..n}. (c ! (m−1)) / 2 ^ m) =
          (c ! ((l+1)−1)) / 2 ^ (l+1) + (∑ m∈{Suc (l+1)..n}. (c ! (m−1)) / 2
^ m)
        by (meson ‹l + 1 < n› nat-less-le sum.atLeast-Suc-atMost)
      also have ... = (∑ m∈{Suc (l+1)..n}. (c ! (m−1)) / 2 ^ m)
        using 2 by auto
      also have ... ≤ (∑ m∈{Suc (l+1)..n}. 1 / 2 ^ m)
        apply (intro sum-mono divide-right-mono)
        using dyadic-expansion-frac-range[OF assms(2)]
          apply (metis (no-types, opaque-lifting) One-nat-def Suc-leI Suc-le-mono
atLeastAtMost-iff
          atLeastAtMost-singleton-iff bot-nat-0.extremum bot-nat-0.not-eq-extremum

            insert-iff of-nat-eq-1-iff of-nat-le-iff)
        apply simp
        done
      also have ... < 1 / 2 ^ (l+1)
        using sum-interval-pow2-inv[OF ‹l + 1 < n›] by fastforce
      finally have (∑ m∈{l+1..n}. (c ! (m−1)) / 2 ^ m) < 1 / 2 ^ (l+1) .
      with b-ge have ?thesis
        by argo
    }
    then show ?thesis
      using ∗ ‹l < n› by linarith
  qed
  moreover have sum-upto-l-eq: (∑ m∈{1..l}. (b ! (m−1)) / 2 ^ m) =
                               (∑ m∈{1..l}. (c ! (m−1)) / 2 ^ m)
    apply (safe intro!: sum.cong)
    apply simp
   by (smt (verit, best) Suc-le-eq Suc-pred ‹l < n› l-def not-less-Least order-less-trans)
```

ultimately have $(\sum m{\in}\{1..n\}.\ (b\ !\ (m{-}1))\ /\ 2\ \hat{}\ m) \neq (\sum m{\in}\{1..n\}.\ (c\ !\ (m{-}1))\ /\ 2\ \hat{}\ m)$
proof −
have $\{1..n\} = \{1..l\} \cup \{l{<}..n\}$
using $\langle l < n \rangle$ **by** *auto*
moreover have $\{1..l\} \cap \{l{<}..n\} = \{\}$
using *ivl-disj-int-two*(8) **by** *blast*
ultimately have *split-sum*: $(\sum m \in\{1..n\}.\ (c\ !\ (m{-}1))\ /\ 2\ \hat{}\ m) =$
$(\sum m = 1..l.\ (c\ !\ (m{-}1))\ /\ 2\ \hat{}\ m) + (\sum m \in \{l{<}..n\}.\ (c\ !\ (m{-}1))\ /\ 2\ \hat{}\ m)$
for c :: *nat list*
by (*simp add*: *sum-Un*)
then show *?thesis*
using *sum-upto-l-eq sum-ge-l-noteq split-sum*[*of b*] *split-sum*[*of c*]
by (*smt* (*verit*, *del-insts*) *Suc-eq-plus1 atLeastSucAtMost-greaterThanAtMost*)
qed
then show *False*
using *eq* **by** *blast*
qed

end

3 Hölder continuity

theory *Holder-Continuous*
imports *HOL−Analysis.Analysis*
begin

Hölder continuity is a weaker version of Lipschitz continuity.

definition *holder-at-within* :: *real* ⇒ *'a set* ⇒ *'a* ⇒ (*'a* :: *metric-space* ⇒ *'b* :: *metric-space*) ⇒ *bool* **where**
holder-at-within $\gamma\ D\ r\ \varphi \equiv \gamma \in \{0{<}..1\} \wedge$
$(\exists \varepsilon > 0.\ \exists C \geq 0.\ \forall s \in D.\ dist\ r\ s < \varepsilon \longrightarrow dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma)$

definition *local-holder-on* :: *real* ⇒ *'a* :: *metric-space set* ⇒ (*'a* ⇒ *'b* :: *metric-space*) ⇒ *bool* **where**
local-holder-on $\gamma\ D\ \varphi \equiv \gamma \in \{0{<}..1\} \wedge$
$(\forall t{\in}D.\ \exists \varepsilon > 0.\ \exists C \geq 0.\ (\forall r{\in}D.\ \forall s{\in}D.\ dist\ s\ t < \varepsilon \wedge dist\ r\ t < \varepsilon \longrightarrow dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma))$

definition *holder-on* :: *real* ⇒ *'a* :: *metric-space set* ⇒ (*'a* ⇒ *'b* :: *metric-space*) ⇒ *bool* (*-−holder'-on 1000*) **where**
γ*−holder-on* $D\ \varphi \longleftrightarrow \gamma \in \{0{<}..1\} \wedge (\exists C \geq 0.\ (\forall r{\in}D.\ \forall s{\in}D.\ dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma))$

lemma *holder-onI*:
assumes $\gamma \in \{0{<}..1\}\ \exists C \geq 0.\ (\forall r{\in}D.\ \forall s{\in}D.\ dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma)$

shows γ−*holder-on* D φ
unfolding *holder-on-def* **using** *assms* **by** *blast*

We prove various equivalent formulations of local holder continuity, using open and closed balls and inequalities.

lemma *local-holder-on-cball*:
local-holder-on γ D $\varphi \longleftrightarrow \gamma \in \{0<..1\} \wedge$
$(\forall t{\in}D.\ \exists \varepsilon > 0.\ \exists C \geq 0.\ (\forall r{\in}cball\ t\ \varepsilon \cap D.\ \forall s{\in}cball\ t\ \varepsilon \cap D.\ dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma))$
(**is** *?L* $\longleftrightarrow$ *?R*)
proof
assume $*$: *?L*
{
fix t **assume** $t \in D$
then obtain ε C **where** $\varepsilon > 0$ $C \geq 0$
$\forall r{\in}ball\ t\ \varepsilon \cap D.\ \forall s{\in}ball\ t\ \varepsilon \cap D.\ dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma$
using $*$ **unfolding** *local-holder-on-def* **apply** *simp*
by (*metis Int-iff dist-commute mem-ball*)
then have $**$: $\forall r{\in}cball\ t\ (\varepsilon/2) \cap D.\ \forall s{\in}cball\ t\ (\varepsilon/2) \cap D.\ dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma$
by *auto*
have $\exists \varepsilon > 0.\ \exists C \geq 0.\ \forall r{\in}cball\ t\ \varepsilon \cap D.\ \forall s{\in}cball\ t\ \varepsilon \cap D.\ dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma$
apply (*rule exI*[**where** $x = \varepsilon/2$])
apply (*simp add*: ‹$\varepsilon > 0$›)
apply (*rule exI*[**where** $x = C$])
using $**$ ‹$C \geq 0$› **by** *blast*
}
then show *?R*
using $*$ *local-holder-on-def* **by** *blast*
next
assume $*$: *?R*
{
fix t **assume** $t \in D$
then obtain ε C **where** *eC*: $\varepsilon > 0$ $C \geq 0$
$\forall r{\in}cball\ t\ \varepsilon \cap D.\ \forall s{\in}cball\ t\ \varepsilon \cap D.\ dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma$
using $*$ **by** *blast*
then have $\forall r \in D.\ \forall s \in D.\ dist\ r\ t < \varepsilon \wedge dist\ s\ t < \varepsilon \longrightarrow dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma$
unfolding *cball-def* **by** (*simp add*: *dist-commute*)
then have $\exists \varepsilon{>}0.\ \exists C{\geq}0.\ \forall r \in D.\ \forall s \in D.\ dist\ r\ t < \varepsilon \wedge dist\ s\ t < \varepsilon \longrightarrow dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma$
using *eC* **by** *blast*
}
then show *local-holder-on* γ D φ
using $*$ **unfolding** *local-holder-on-def* **by** *metis*
qed

corollary *local-holder-on-leq-def*: *local-holder-on* γ D $\varphi \longleftrightarrow \gamma \in \{0<..1\} \wedge$

$(\forall\, t{\in}D.\ \exists\, \varepsilon > 0.\ \exists\, C \geq 0.\ (\forall\, r{\in}D.\ \forall\, s{\in}D.\ dist\ s\ t \leq \varepsilon \wedge dist\ r\ t \leq \varepsilon \longrightarrow dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma))$
unfolding *local-holder-on-cball* **by** (*metis dist-commute Int-iff mem-cball*)

corollary *local-holder-on-ball*: *local-holder-on* $\gamma\ D\ \varphi \longleftrightarrow \gamma \in \{0<..1\} \wedge$
$(\forall\, t{\in}D.\ \exists\, \varepsilon > 0.\ \exists\, C \geq 0.\ (\forall\, r{\in}ball\ t\ \varepsilon \cap D.\ \forall\, s{\in}ball\ t\ \varepsilon \cap D.\ dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma))$
unfolding *local-holder-on-def* **by** (*metis dist-commute Int-iff mem-ball*)

lemma *local-holder-on-altdef*:
assumes $D \neq \{\}$
shows *local-holder-on* $\gamma\ D\ \varphi = (\forall\, t \in D.\ (\exists\, \varepsilon > 0.\ (\gamma{-}holder\text{-}on\ ((cball\ t\ \varepsilon) \cap D)\ \varphi)))$
unfolding *local-holder-on-cball holder-on-def* **using** *assms* **by** *blast*

lemma *local-holder-on-cong*[*cong*]:
assumes $\gamma = \varepsilon\ C = D\ \bigwedge x.\ x \in C \Longrightarrow \varphi\ x = \psi\ x$
shows *local-holder-on* $\gamma\ C\ \varphi \longleftrightarrow$ *local-holder-on* $\varepsilon\ D\ \psi$
unfolding *local-holder-on-def* **using** *assms* **by** *presburger*

lemma *local-holder-onI*:
assumes $\gamma \in \{0<..1\}$ $(\forall\, t{\in}D.\ \exists\, \varepsilon > 0.\ \exists\, C \geq 0.\ (\forall\, r{\in}D.\ \forall\, s{\in}D.\ dist\ s\ t < \varepsilon \wedge dist\ r\ t < \varepsilon \longrightarrow dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma))$
shows *local-holder-on* $\gamma\ D\ \varphi$
using *assms* **unfolding** *local-holder-on-def* **by** *blast*

lemma *local-holder-ballI*:
assumes $\gamma \in \{0<..1\}$
and $\bigwedge t.\ t \in D \Longrightarrow \exists\, \varepsilon > 0.\ \exists\, C \geq 0.\ \forall\, r{\in}ball\ t\ \varepsilon \cap D.\ \forall\, s{\in}ball\ t\ \varepsilon \cap D.$
$dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma$
shows *local-holder-on* $\gamma\ D\ \varphi$
using *assms* **unfolding** *local-holder-on-ball* **by** *blast*

lemma *local-holder-onE*:
assumes *local-holder*: *local-holder-on* $\gamma\ D\ \varphi$
and *gamma*: $\gamma \in \{0<..1\}$
and $t \in D$
obtains $\varepsilon\ C$ **where** $\varepsilon > 0\ C \geq 0$
$\bigwedge r\ s.\ r \in ball\ t\ \varepsilon \cap D \Longrightarrow s \in ball\ t\ \varepsilon \cap D \Longrightarrow dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s\ powr\ \gamma$
using *assms* **unfolding** *local-holder-on-ball* **by** *auto*

Holder continuity matches up with the existing definitions in *HOL−Analysis.Lipschitz*

lemma *holder-1-eq-lipschitz*: $1{-}holder\text{-}on\ D\ \varphi = (\exists\, C.\ lipschitz\text{-}on\ C\ D\ \varphi)$
unfolding *holder-on-def lipschitz-on-def* **by** (*auto simp*: *fun-eq-iff dist-commute*)

lemma *local-holder-1-eq-local-lipschitz*:
assumes $T \neq \{\}$
shows *local-holder-on* $1\ D\ \varphi =$ *local-lipschitz* $T\ D\ (\lambda\text{-}.\ \varphi)$

proof
assume $*$: *local-holder-on 1 D* φ
{
fix t **assume** $t \in D$
then obtain ε C **where** eC: $\varepsilon > 0$ $C \geq 0$
$(\forall\, r{\in}D.\ \forall\, s{\in}D.\ dist\ s\ t \leq \varepsilon \wedge dist\ r\ t \leq \varepsilon \longrightarrow dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s)$
using $*$ *powr-to-1* **unfolding** *local-holder-on-cball* **apply** *simp*
by (*metis Int-iff dist-commute mem-cball*)
{
fix r s **assume** rs: $r \in D$ $s \in D$ $dist\ s\ t \leq \varepsilon \wedge dist\ r\ t \leq \varepsilon$
then have $r \in cball\ t\ \varepsilon \cap D$ $s{\in}cball\ t\ \varepsilon \cap D$ $dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s$
unfolding *cball-def* **using** *rs eC* **by** (*auto simp*: *dist-commute*)
}
then have $\forall\, r{\in}cball\ t\ \varepsilon \cap D.\ \forall\, s{\in}cball\ t\ \varepsilon \cap D.\ dist\ (\varphi\ r)\ (\varphi\ s) \leq C * dist\ r\ s$
by (*simp add*: *dist-commute*)
then have $C{-}lipschitz\text{-}on\ ((cball\ t\ \varepsilon) \cap D)\ \varphi$
using *eC lipschitz-on-def* **by** *blast*
then have $\exists\, \varepsilon{>}0.\ \exists\, C.\ C{-}lipschitz\text{-}on\ ((cball\ t\ \varepsilon) \cap D)\ \varphi$
using $eC(1)$ **by** *blast*
}
then show *local-lipschitz T D* (λ-. φ)
unfolding *local-lipschitz-def* **by** *blast*
next
assume $*$: *local-lipschitz T D* (λ-. φ)
{
fix x **assume** x: $x \in D$
fix t **assume** t: $t \in T$
then obtain u L **where** uL: $u > 0$ $\forall\, t{\in}cball\ t\ u \cap T.\ L{-}lipschitz\text{-}on\ (cball\ x\ u \cap D)\ \varphi$
using $*$ x t **unfolding** *local-lipschitz-def* **by** *blast*
then have $L{-}lipschitz\text{-}on\ (cball\ x\ u \cap D)\ \varphi$
using t **by** *force*
then have $1{-}holder\text{-}on\ (cball\ x\ u \cap D)\ \varphi$
using *holder-1-eq-lipschitz* **by** *blast*
then have $\exists\, \varepsilon > 0.\ (1{-}holder\text{-}on\ ((cball\ x\ \varepsilon) \cap D)\ \varphi)$
using uL **by** *blast*
}
then have $x \in D \Longrightarrow \exists\, \varepsilon{>}0.\ (1{-}holder\text{-}on\ (cball\ x\ \varepsilon \cap D)\ \varphi)$ **for** x
using *assms* **by** *blast*
then show *local-holder-on 1 D* φ
unfolding *local-holder-on-cball holder-on-def* **by** (*auto simp*: *dist-commute*)
qed

lemma *local-holder-refine*:
assumes g: *local-holder-on g D* φ $g \leq 1$
and h: $h \leq g$ $h > 0$
shows *local-holder-on h D* φ
proof −
{

```
    fix t assume t: t ∈ D
    then have ∃ε>0. ∃ C≥0. (∀ r∈D. ∀ s∈D. dist s t ≤ ε ∧ dist r t ≤ ε ⟶ dist
(φ r) (φ s) ≤ C ∗ dist r s powr g)
      using g(1) unfolding local-holder-on-leq-def by blast
    then obtain ε C where eC: ε > 0 C ≥ 0
      (∀ s∈D. ∀ r∈D. dist s t ≤ ε ∧ dist r t ≤ ε ⟶ dist (φ r) (φ s) ≤ C ∗ dist r s
powr g)
      by blast
    let ?e = min ε (1/2)
    {
      fix s r assume ∗: s ∈ D r ∈ D dist s t ≤ ?e dist r t ≤ ?e
      then have dist (φ r) (φ s) ≤ C ∗ dist r s powr g
        using eC by simp
      moreover have dist r s ≤ 1
        by (smt (verit) ∗ dist-triangle2 half-bounded-equal)
      ultimately have dist (φ r) (φ s) ≤ C ∗ dist r s powr h
      by (metis dual-order.trans zero-le-dist powr-mono' assms(3) eC(2) mult-left-mono
)
    }
    then have (∀ s∈D. ∀ r∈D. dist s t ≤ ?e ∧ dist r t ≤ ?e ⟶ dist (φ r) (φ s)
≤ C ∗ dist r s powr h)
      by blast
    moreover have ?e > 0 C ≥ 0
      using eC by linarith+
    ultimately have ∃ε>0. ∃ C≥0. (∀ r∈D. ∀ s∈D. dist s t ≤ ε ∧ dist r t ≤ ε
⟶ dist (φ r) (φ s) ≤ C ∗ dist r s powr h)
      by blast
  }
  then show ?thesis
    unfolding local-holder-on-leq-def using assms by force
qed

lemma holder-uniform-continuous:
  assumes γ−holder-on X φ
  shows uniformly-continuous-on X φ
  unfolding uniformly-continuous-on-def
proof safe
  fix e::real
  assume 0 < e
  from assms obtain C where C: C ≥ 1 (∀ r∈X. ∀ s∈X. dist (φ r) (φ s) ≤ C ∗
dist r s powr γ)
    unfolding holder-on-def
    by (smt (verit) dist-eq-0-iff mult-le-cancel-right1 powr-0 powr-gt-zero)
  {
    fix r s assume r ∈ X s ∈ X
    have dist-0: dist (φ r) (φ s) = 0 ⟹ dist (φ r) (φ s) < e
      using ‹0 < e› by linarith
    then have holder-neq-0: dist (φ r) (φ s) < (C + 1) ∗ dist r s powr γ if dist
(φ r) (φ s) > 0
```

using *C(2) that*
by (*smt* (*verit*, *ccfv-SIG*) ‹*r* ∈ *X*› ‹*s* ∈ *X*› *dist-eq-0-iff mult-le-cancel-right powr-gt-zero*)
have *gamma*: $\gamma \in \{0<..1\}$
using *assms holder-on-def* **by** *blast+*
assume *dist r s* < (*e*/*C*) *powr* (*1* / γ)
then have *C* ∗ *dist r s powr* γ < *C* ∗ ((*e*/*C*) *powr* (*1* / γ)) *powr* γ **if** *dist* (φ *r*) (φ *s*) > *0*
using *holder-neq-0 C(1) powr-less-mono2 gamma* **by** *fastforce*
also have ... = *e*
using *C(1) gamma* ‹*0* < *e*› *powr-powr* **by** *auto*
finally have *dist* (φ *r*) (φ *s*) < *e*
using *dist-0 holder-neq-0 C(2)* ‹*r* ∈ *X*› ‹*s* ∈ *X*› **by** *fastforce*
}
then show ∃ *d*>*0*. ∀ *x*∈*X*. ∀ *x*′∈*X*. *dist x*′ *x* < *d* ⟶ *dist* (φ *x*′) (φ *x*) < *e*
by (*metis C(1)* ‹*0* < *e*› *divide-eq-0-iff linorder-not-le order-less-irrefl powr-gt-zero zero-less-one*)
qed

corollary *holder-on-continuous-on*: γ−*holder-on X* φ ⟹ *continuous-on X* φ
using *holder-uniform-continuous uniformly-continuous-imp-continuous* **by** *blast*

lemma *holder-implies-local-holder*: γ−*holder-on D* φ ⟹ *local-holder-on* γ *D* φ
apply (*cases D* = {})
apply (*simp add*: *holder-on-def local-holder-on-def*)
apply (*simp add*: *local-holder-on-altdef holder-on-def*)
apply (*metis IntD1 inf.commute*)
done

lemma *local-holder-imp-continuous*:
assumes *local-holder*: *local-holder-on* γ *X* φ
shows *continuous-on X* φ
unfolding *continuous-on-def*
proof *safe*
fix *x* **assume** *x* ∈ *X*
{
assume *X* ≠ {}
from *local-holder* **obtain** ε **where** *0* < ε **and** *holder*: γ−*holder-on* ((*cball x* ε) ∩ *X*) φ
unfolding *local-holder-on-altdef*[*OF* ‹*X* ≠ {}›] **using** ‹*x* ∈ *X*› **by** *blast*
have *x* ∈ *ball x* ε **using** ‹*0* < ε› **by** *simp*
then have (φ ⟶ φ *x*) (*at x within cball x* ε ∩ *X*)
using *holder-on-continuous-on*[*OF holder*] ‹*x* ∈ *X*› **unfolding** *continuous-on-def* **by** *simp*
moreover have ∀ *F xa in at x*. (*xa* ∈ *cball x* ε ∩ *X*) = (*xa* ∈ *X*)
using *eventually-at-ball*[*OF* ‹*0* < ε›, *of x UNIV*]
by *eventually-elim auto*
ultimately have (φ ⟶ φ *x*) (*at x within X*)

```
    by (rule Lim-transform-within-set)
  }
  then show (φ ⟶ φ x) (at x within X)
    by fastforce
qed

lemma local-holder-compact-imp-holder:
  assumes compact I local-holder-on γ I φ
  shows γ−holder-on I φ
proof −
  have ∗: γ ∈ {0<..1} (∀ t∈I. ∃ε. ∃ C. ε > 0 ∧ C ≥ 0 ∧
    (∀ r ∈ ball t ε ∩ I. ∀ s ∈ ball t ε ∩ I. dist (φ r) (φ s) ≤ C ∗ dist r s powr γ))
    using assms(2) unfolding local-holder-on-ball by simp-all
  obtain ε C where eC: t ∈ I ⟹ ε t > 0 ∧ C t ≥ 0 ∧ (∀ r ∈ ball t (ε t) ∩ I.
∀ s ∈ ball t (ε t) ∩ I. dist (φ r) (φ s) ≤ C t ∗ dist r s powr γ) for t
    by (metis ∗(2))
  have I ⊆ (⋃ t ∈ I. ball t (ε t))
    apply (simp add: subset-iff)
    using eC by force
  then obtain D where D: D ⊆ (λt. ball t (ε t)) ‘ I finite D I ⊆ ⋃ D
    using compact-eq-Heine-Borel[of I] apply (simp add: assms(1))
    by (smt (verit, ccfv-SIG) open-ball imageE mem-Collect-eq subset-iff)
  then obtain T where T: D = (λt. ball t (ε t)) ‘ T T ⊆ I finite T
    by (meson finite-subset-image subset-image-iff)
```

ϱ is the Lebesgue number of the cover

```
  from D obtain ϱ :: real where ϱ: ∀ t ∈ I. ∃ U ∈ D. ball t ϱ ⊆ U ϱ > 0
   by (smt (verit, del-insts) Elementary-Metric-Spaces.open-ball Heine-Borel-lemma
assms(1) imageE subset-image-iff)
  have bounded (φ ‘ I)
   by (metis compact-continuous-image compact-imp-bounded assms local-holder-imp-continuous)
  then obtain l where l: ∀ x ∈ I. ∀ y ∈ I. dist (φ x) (φ y) ≤ l
    by (metis bounded-two-points image-eqI)
```

Simply need to construct C_bar such that it is greater than any of these

```
  define C-bar where C-bar ≡ max ((∑ t ∈ T. C t)) (l ∗ ϱ powr (− γ))
  have C-bar-le: C-bar ≥ C t if t ∈ T for t
  proof −
    have ge-0: t ∈ T ⟹ C t ≥ 0 for t
      using T(2) eC by blast
    then have ∑ (C ‘ (T − {t})) ≥ 0
      by (metis (mono-tags, lifting) Diff-subset imageE subset-eq sum-nonneg)
    then have (∑ t ∈ T. C t) ≥ C t
      by (metis T(3) ge-0 sum-nonneg-leq-bound that)
    then have max ((∑ t ∈ T. C t)) S ≥ C t for S
      by argo
    then show C-bar ≥ C t
      unfolding C-bar-def by blast
  qed
```

```
  {
    fix s r assume sr: s ∈ I r ∈ I
    {
      assume dist s r < ϱ
      then obtain t where t: t ∈ T s ∈ ball t (ε t) r ∈ ball t (ε t)
          by (smt (verit) sr D T ϱ ball-eq-empty centre-in-ball imageE mem-ball
subset-iff)
      then have dist (φ s) (φ r) ≤ C t * dist s r powr γ
        using eC[of t] T(2) sr by blast
      then have dist (φ s) (φ r) ≤ C-bar * dist s r powr γ
        by (smt (verit, best) t C-bar-le mult-right-mono powr-non-neg)
    } note le-rho = this
    {
      assume dist s r ≥ ϱ
      then have dist (φ s) (φ r) ≤ l * (dist s r / ϱ) powr γ
      proof −
        have (dist s r / ϱ) ≥ 1
          using ‹dist s r ≥ ϱ› ‹ϱ > 0› by auto
        then have (dist s r / ϱ) powr γ ≥ 1
          using *(1) ge-one-powr-ge-zero by auto
        then show dist (φ s) (φ r) ≤ l * (dist s r / ϱ) powr γ
          using l
       by (metis dist-self dual-order.strict-trans1 leI less-le-not-le mult-less-cancel-left2
sr)
      qed
      also have ... ≤ C-bar * dist s r powr γ
      proof −
        have l * (dist s r / ϱ) powr γ = l * ϱ powr (− γ) * dist s r powr γ
          using ϱ(2) divide-powr-uminus powr-divide by force
        also have ... ≤ C-bar * dist s r powr γ
          unfolding C-bar-def by (simp add: mult-right-mono)
        finally show l * (dist s r / ϱ) powr γ ≤ C-bar * dist s r powr γ
          .
      qed
      finally have dist (φ s) (φ r) ≤ C-bar * dist s r powr γ
        .
    }
    then have dist (φ s) (φ r) ≤ C-bar * dist s r powr γ
      using le-rho by argo
  }
  then have ∀ r∈I. ∀ s∈I. dist (φ r) (φ s) ≤ C-bar * dist r s powr γ
    by simp
  then show ?thesis
    unfolding holder-on-def
   by (metis *(1) C-bar-def dist-self div-by-0 divide-nonneg-pos divide-powr-uminus

      dual-order.trans l max.cobounded2 powr-0 powr-gt-zero)
qed
```

lemma *holder-const*: γ−*holder-on* C (λ-. c) $\longleftrightarrow$ $\gamma \in \{0<..1\}$
unfolding *holder-on-def* **by** *auto*

lemma *local-holder-const*: *local-holder-on* γ C (λ-. c) $\longleftrightarrow$ $\gamma \in \{0<..1\}$
using *holder-const holder-implies-local-holder local-holder-on-def* **by** *blast*

end

4 Convergence in measure

theory *Measure-Convergence*
imports *HOL−Probability.Probability*
begin

We use measure rather than emeasure because ennreal is not a metric space, which we need to reason about convergence. By intersecting with the set of finite measure A, we don't run into issues where infinity is collapsed to 0.

For finite measures this definition is equal to the definition without set A – see below.

definition *tendsto-measure* :: *'b measure* ⇒ (*'a* ⇒ *'b* ⇒ (*'c* :: {*second-countable-topology*,*metric-space*}))
⇒ (*'b* ⇒ *'c*) ⇒ *'a filter* ⇒ *bool*
where *tendsto-measure M X l F* ≡ (∀ *n*. *X n* ∈ *borel-measurable M*) ∧ *l* ∈ *borel-measurable M* ∧
(∀ $\varepsilon > 0$. ∀ *A* ∈ *fmeasurable M*.
((λ*n*. *measure M* ({ω ∈ *space M*. *dist* (*X n* ω) (*l* ω) > ε} ∩ *A*)) ⟶ 0) *F*)

abbreviation (**in** *prob-space*) *tendsto-prob* (**infixr** ⟶$_P$ 55) **where**
(*f* ⟶$_P$ *l*) *F* ≡ *tendsto-measure M f l F*

lemma *tendsto-measure-measurable*[*measurable-dest*]:
tendsto-measure M X l F ⟹ *X n* ∈ *borel-measurable M*
unfolding *tendsto-measure-def* **by** *meson*

lemma *tendsto-measure-measurable-lim*[*measurable-dest*]:
tendsto-measure M X l F ⟹ *l* ∈ *borel-measurable M*
unfolding *tendsto-measure-def* **by** *meson*

lemma *tendsto-measure-mono*: *F* ≤ *F'* ⟹ *tendsto-measure M f l F'* ⟹ *tendsto-measure M f l F*
unfolding *tendsto-measure-def* **by** (*simp add*: *tendsto-mono*)

lemma *tendsto-measureI*:
assumes [*measurable*]: ⋀*n*. *X n* ∈ *borel-measurable M l* ∈ *borel-measurable M*
and ⋀ε *A*. $\varepsilon > 0$ ⟹ *A* ∈ *fmeasurable M* ⟹
((λ*n*. *measure M* ({ω ∈ *space M*. *dist* (*X n* ω) (*l* ω) > ε} ∩ *A*)) ⟶ 0) *F*
shows *tendsto-measure M X l F*
unfolding *tendsto-measure-def* **using** *assms* **by** *fast*

lemma (**in** *finite-measure*) *finite-tendsto-measureI*:
assumes [*measurable*]: $\bigwedge n.\ f'\ n \in$ *borel-measurable* $M\ f \in$ *borel-measurable* M
and $\bigwedge \varepsilon.\ \varepsilon > 0 \Longrightarrow ((\lambda n.\ \textit{measure}\ M\ \{\omega \in \textit{space}\ M.\ \textit{dist}\ (f'\ n\ \omega)\ (f\ \omega) > \varepsilon\}) \longrightarrow 0)\ F$
shows *tendsto-measure* $M\ f'\ f\ F$
proof (*intro tendsto-measureI*)
fix ε :: *real* **assume** $\varepsilon > 0$
then have *prob-conv*: $((\lambda n.\ \textit{measure}\ M\ \{\omega \in \textit{space}\ M.\ \varepsilon < \textit{dist}\ (f'\ n\ \omega)\ (f\ \omega)\}) \longrightarrow 0)\ F$
using *assms* **by** *simp*
fix A **assume** $A \in$ *fmeasurable* M
have $\bigwedge n.\ \textit{measure}\ M\ (\{\omega \in \textit{space}\ M.\ \varepsilon < \textit{dist}\ (f'\ n\ \omega)\ (f\ \omega)\}) \geq$
measure $M\ (\{\omega \in \textit{space}\ M.\ \varepsilon < \textit{dist}\ (f'\ n\ \omega)\ (f\ \omega)\} \cap A)$
by (*rule finite-measure-mono*; *measurable*)
then show $((\lambda n.\ \textit{measure}\ M\ (\{\omega \in \textit{space}\ M.\ \varepsilon < \textit{dist}\ (f'\ n\ \omega)\ (f\ \omega)\} \cap A)) \longrightarrow 0)\ F$
by (*simp add*: *tendsto-0-le*[*OF prob-conv*, **where** *K*=*1*])
qed *measurable*

lemma (**in** *finite-measure*) *finite-tendsto-measureD*:
assumes [*measurable*]: *tendsto-measure* $M\ f'\ f\ F$
shows $(\forall \varepsilon > 0.\ ((\lambda n.\ \textit{measure}\ M\ \{\omega \in \textit{space}\ M.\ \textit{dist}\ (f'\ n\ \omega)\ (f\ \omega) > \varepsilon\}) \longrightarrow 0)\ F)$
proof −
from *assms* **have** $((\lambda n.\ \textit{measure}\ M\ (\{\omega \in \textit{space}\ M.\ \textit{dist}\ (f'\ n\ \omega)\ (f\ \omega) > \varepsilon\} \cap \textit{space}\ M)) \longrightarrow 0)\ F$
if $\varepsilon > 0$ **for** ε
unfolding *tendsto-measure-def* **using** *that fmeasurable-eq-sets* **by** *blast*
then show *?thesis*
by (*simp add*: *sets.Int-space-eq2*[*symmetric*, **where** *M*=*M*])
qed

lemma (**in** *finite-measure*) *tendsto-measure-leq*:
assumes [*measurable*]: $\bigwedge n.\ f'\ n \in$ *borel-measurable* $M\ f \in$ *borel-measurable* M
shows *tendsto-measure* $M\ f'\ f\ F \longleftrightarrow$
$(\forall \varepsilon > 0.\ ((\lambda n.\ \textit{measure}\ M\ \{\omega \in \textit{space}\ M.\ \textit{dist}\ (f'\ n\ \omega)\ (f\ \omega) \geq \varepsilon\}) \longrightarrow 0)\ F)$ (**is** *?L* $\longleftrightarrow$ *?R*)
proof (*rule iffI*, *goal-cases*)
case *1*
{
fix ε :: *real* **assume** $\varepsilon > 0$
then have $((\lambda n.\ \textit{measure}\ M\ \{\omega \in \textit{space}\ M.\ \textit{dist}\ (f'\ n\ \omega)\ (f\ \omega) > \varepsilon/2\}) \longrightarrow 0)\ F$
using *finite-tendsto-measureD*[*OF 1*] *half-gt-zero* **by** *blast*
then have $((\lambda n.\ \textit{measure}\ M\ \{\omega \in \textit{space}\ M.\ \textit{dist}\ (f'\ n\ \omega)\ (f\ \omega) \geq \varepsilon\}) \longrightarrow 0)\ F$
apply (*rule metric-tendsto-imp-tendsto*)
using ‹$\varepsilon > 0$› **by** (*auto intro*!: *eventuallyI finite-measure-mono*)
}

```
  then show ?case
    by simp
next
  case 2
  {
    fix ε :: real assume ε > 0
    then have ∗: ((λn. P(ω in M. ε ≤ dist (f' n ω) (f ω))) ⟶ 0) F
      using 2 by blast
    then have ((λn. P(ω in M. ε < dist (f' n ω) (f ω))) ⟶ 0) F
      apply (rule metric-tendsto-imp-tendsto)
      using ‹ε > 0› by (auto intro!: eventuallyI finite-measure-mono)
  }
  then show ?case
    by (simp add: finite-tendsto-measureI[OF assms])
qed

abbreviation LIMSEQ-measure M f l ≡ tendsto-measure M f l sequentially

lemma LIMSEQ-measure-def: LIMSEQ-measure M f l ⟷
  (∀ n. f n ∈ borel-measurable M) ∧ (l ∈ borel-measurable M) ∧
   (∀ ε > 0. ∀ A ∈ fmeasurable M.
   (λn. measure M ({ω ∈ space M. dist (f n ω) (l ω) > ε} ∩ A)) ⟶ 0)
  unfolding tendsto-measure-def ..

lemma LIMSEQ-measureD:
  assumes LIMSEQ-measure M f l ε > 0 A ∈ fmeasurable M
  shows (λn. measure M ({ω ∈ space M. dist (f n ω) (l ω) > ε} ∩ A)) ⟶ 0
  using assms LIMSEQ-measure-def by blast

lemma fmeasurable-inter: ⟦A ∈ sets M; B ∈ fmeasurable M⟧ ⟹ A ∩ B ∈ fmeasurable M
proof (intro fmeasurableI, goal-cases measurable finite)
  case measurable
  then show ?case by simp
next
  case finite
  then have emeasure M (A ∩ B) ≤ emeasure M B
    by (simp add: emeasure-mono)
  also have emeasure M B < ∞
    using finite(2)[THEN fmeasurableD2] by (simp add: top.not-eq-extremum)
  finally show ?case .
qed

lemma LIMSEQ-measure-emeasure:
  assumes LIMSEQ-measure M f l ε > 0 A ∈ fmeasurable M
    and [measurable]: ⋀i. f i ∈ borel-measurable M l ∈ borel-measurable M
  shows (λn. emeasure M ({ω ∈ space M. dist (f n ω) (l ω) > ε} ∩ A)) ⟶ 0
proof −
  have fmeasurable: {ω ∈ space M. dist (f n ω) (l ω) > ε} ∩ A ∈ fmeasurable M
```

for *n*
by (*rule fmeasurable-inter*; *simp add*: *assms*(*3*))
then show *?thesis*
apply (*simp add*: *emeasure-eq-measure2 ennreal-tendsto-0-iff*)
using *LIMSEQ-measure-def assms* **by** *blast*
qed

lemma *measure-Lim-within-LIMSEQ*:
fixes *a* :: *'a* :: *first-countable-topology*
assumes ⋀*t*. *X t* ∈ *borel-measurable M L* ∈ *borel-measurable M*
assumes ⋀*S*. ⟦(∀ *n*. *S n* ≠ *a* ∧ *S n* ∈ *T*); *S* ⟶ *a*⟧ ⟹ *LIMSEQ-measure M* (λ*n*. *X* (*S n*)) *L*
shows *tendsto-measure M X L* (*at a within T*)
apply (*intro tendsto-measureI*[*OF assms*(*1*,*2*)])
unfolding *tendsto-measure-def*[**where** *l*=*L*] *tendsto-def* **apply** *safe*
apply (*rule sequentially-imp-eventually-within*)
using *assms* **unfolding** *LIMSEQ-measure-def tendsto-def* **by** *presburger*

definition *tendsto-AE* :: *'b measure* ⇒ (*'a* ⇒ *'b* ⇒ *'c* :: *topological-space*) ⇒ (*'b* ⇒ *'c*) ⇒ *'a filter* ⇒ *bool* **where**
tendsto-AE M f' l F ⟷ (*AE* ω *in M*. ((λ*n*. *f' n* ω) ⟶ *l* ω) *F*)

lemma *LIMSEQ-ae-pointwise*: (⋀*x*. (λ*n*. *f n x*) ⟶ *l x*) ⟹ *tendsto-AE M f l sequentially*
unfolding *tendsto-AE-def* **by** *simp*

lemma *tendsto-AE-within-LIMSEQ*:
fixes *a* :: *'a* :: *first-countable-topology*
assumes ⋀*S*. ⟦(∀ *n*. *S n* ≠ *a* ∧ *S n* ∈ *T*); *S* ⟶ *a*⟧ ⟹ *tendsto-AE M* (λ*n*. *X* (*S n*)) *L sequentially*
shows *tendsto-AE M X L* (*at a within T*)
oops

lemma *LIMSEQ-dominated-convergence*:
fixes *X* :: *nat* ⇒ *real*
assumes *X* ⟶ *L* (⋀*n*. *Y n* ≤ *X n*) (⋀*n*. *Y n* ≥ *L*)
shows *Y* ⟶ *L*
proof (*rule metric-LIMSEQ-I*)
have *X n* ≥ *L* **for** *n*
using *assms*(*2*,*3*)[*of n*] **by** *linarith*
fix *r* :: *real* **assume** *0* < *r*
then obtain *N* **where** ∀ *n*≥*N*. *dist* (*X n*) *L* < *r*
using *metric-LIMSEQ-D*[*OF assms*(*1*) ‹*0* < *r*›] **by** *blast*
then have *N*: ∀ *n*≥*N*. *X n* − *L* < *r*
using ‹⋀*n*. *L* ≤ *X n*› **by** (*auto simp*: *dist-real-def*)
have ∀ *n*≥*N*. *Y n* − *L* < *r*
proof *clarify*
fix *n* **assume** ‹*n* ≥ *N*›
then have *X n* − *L* < *r*

```
    using N by blast
  then show Y n − L < r
    using assms(2)[of n] by auto
 qed
 then show ∃ no. ∀ n≥no. dist (Y n) L < r
  apply (intro exI[where x=N])
  using assms(3) dist-real-def by auto
qed
```

Klenke remark 6.4

```
lemma measure-conv-imp-AE-sequentially:
 assumes [measurable]: ⋀n. f' n ∈ borel-measurable M f ∈ borel-measurable M
  and tendsto-AE M f' f sequentially
 shows LIMSEQ-measure M f' f
proof (unfold tendsto-measure-def, safe)
 fix ε :: real assume 0 < ε
 fix A assume A[measurable]: A ∈ fmeasurable M
```

From AE convergence we know there's a null set where f' doesn't converge

```
  obtain N where N: N ∈ null-sets M {ω ∈ space M. ¬ (λn. f' n ω) ⟶ f
ω} ⊆ N
    using assms unfolding tendsto-AE-def by (simp add: eventually-ae-filter,
blast)
 then have measure-0: measure M {ω ∈ space M. ¬ (λn. f' n ω) ⟶ f ω} =
0
   by (meson measure-eq-0-null-sets measure-notin-sets null-sets-subset)
```

D is a sequence of sets that converges to N

```
 define D where D ≡ λn. {ω ∈ space M. ∃ m ≥ n. dist (f' m ω) (f ω) > ε}
 have ⋀n. D n ∈ sets M
  unfolding D-def by measurable
 then have [measurable]: ⋀n. D n ∩ A ∈ sets M
  by simp
 have (⋂ n. D n) ∈ sets M
  unfolding D-def by measurable
 then have measurable-D-A: (⋂ n. D n ∩ A) ∈ sets M
  by simp
 have (⋂ n. D n) ⊆ {ω ∈ space M. ¬ (λn. (f' n ω)) ⟶ f ω}
 proof (intro subsetI)
  fix x assume x ∈ (⋂ n. D n)
  then have x ∈ space M ⋀n. ∃ m ≥ n. ε < dist (f' m x) (f x)
   unfolding D-def by simp-all
  then have ¬ (λn. f' n x) ⟶ f x
   by (simp add: LIMSEQ-def) (meson ‹0 < ε› not-less-iff-gr-or-eq order-less-le)
  then show x ∈ {ω ∈ space M. ¬ (λn. f' n ω) ⟶ f ω}
   using ‹x ∈ space M› by blast
 qed
 then have measure M (⋂ n. D n) = 0
```

by (*metis* (*no-types*, *lifting*) *N* ‹$\bigcap$ (*range D*) ∈ *sets M*› *measure-eq-0-null-sets null-sets-subset subset-trans*)
then have *measure M* ($\bigcap n.$ *D n* ∩ *A*) = *0*
proof −
have *emeasure M* ($\bigcap n.$ *D n* ∩ *A*) ≤ *emeasure M* ($\bigcap n.$ *D n*)
apply (*rule emeasure-mono*)
apply *blast*
unfolding *D-def* **apply** *measurable*
done
then show *?thesis*
by (*smt* (*verit*, *del-insts*) *N Sigma-Algebra.measure-def* ‹*measure M* ($\bigcap$ (*range D*)) = *0*›
‹$\bigcap$ (*range D*) ∈ *sets M*› ‹$\bigcap$ (*range D*) ⊆ {ω ∈ *space M*. ¬ (λ*n*. *f* ′ *n* ω) ⟶ *f* ω}›
dual-order.trans enn2real-mono ennreal-zero-less-top measure-nonneg null-setsD1 null-sets-subset)
qed
moreover have (λ*n*. *measure M* (*D n* ∩ *A*)) ⟶ *measure M* ($\bigcap n.$ *D n* ∩ *A*)
apply (*rule Lim-measure-decseq*)
using *A*(*1*) ‹$\bigwedge n.$ *D n* ∈ *sets M*› **apply** *blast*
subgoal
apply (*intro monotoneI*)
apply *clarsimp*
apply (*simp add*: *D-def*)
by (*meson order-trans*)
apply (*simp add*: *A* ‹$\bigwedge n.$ *D n* ∈ *sets M*› *fmeasurableD2 fmeasurable-inter*)
done
ultimately have *measure-D-0*: (λ*n*. *measure M* (*D n* ∩ *A*)) ⟶ *0*
by *presburger*
have $\bigwedge n.$ {ω ∈ *space M*. ε < *dist* (*f* ′ *n* ω) (*f* ω)} ∩ *A* ⊆ (*D n* ∩ *A*)
unfolding *D-def* **by** *blast*
then have $\bigwedge n.$ *emeasure M* ({ω ∈ *space M*. ε < *dist* (*f* ′ *n* ω) (*f* ω)} ∩ *A*) ≤ *emeasure M* (*D n* ∩ *A*)
by (*rule emeasure-mono*) *measurable*
then have $\bigwedge n.$ *measure M* ({ω ∈ *space M*. ε < *dist* (*f* ′ *n* ω) (*f* ω)} ∩ *A*) ≤ *measure M* (*D n* ∩ *A*)
unfolding *measure-def* **apply** (*rule enn2real-mono*)
by (*meson A* ‹$\bigwedge n.$ *D n* ∈ *sets M*› *fmeasurableD2 fmeasurable-inter top.not-eq-extremum*)
then show (λ*n*. *measure M* ({ω ∈ *space M*. ε < *dist* (*f* ′ *n* ω) (*f* ω)} ∩ *A*)) ⟶ *0*
by (*simp add*: *LIMSEQ-dominated-convergence*[*OF measure-D-0*])
qed *simp-all*

corollary *LIMSEQ-measure-pointwise*:
assumes $\bigwedge x.$ (λ*n*. *f n x*) ⟶ *f* ′ *x* $\bigwedge n.$ *f n* ∈ *borel-measurable M f* ′ ∈ *borel-measurable M*
shows *LIMSEQ-measure M f f* ′
by (*simp add*: *LIMSEQ-ae-pointwise measure-conv-imp-AE-sequentially assms*)

lemma *Lim-measure-pointwise*:
fixes $a :: 'a :: \textit{first-countable-topology}$
assumes $\bigwedge x.\ ((\lambda n.\ f\ n\ x) \longrightarrow f'\ x)(\textit{at}\ a\ \textit{within}\ T)\ \bigwedge n.\ f\ n \in \textit{borel-measurable}\ M\ f' \in \textit{borel-measurable}\ M$
shows *tendsto-measure* $M\ f\ f'\ (\textit{at}\ a\ \textit{within}\ T)$
proof (*intro measure-Lim-within-LIMSEQ*)
fix S **assume** $\forall n.\ S\ n \neq a \wedge S\ n \in T\ S \longrightarrow a$
then have $(\lambda n.\ f\ (S\ n)\ x) \longrightarrow f'\ x$ **for** x
using *assms*(*1*) **by** (*simp add*: *tendsto-at-iff-sequentially o-def*)
then show *LIMSEQ-measure* $M\ (\lambda n.\ f\ (S\ n))\ f'$
by (*simp add*: *LIMSEQ-measure-pointwise assms*(*2*,*3*))
qed (*simp-all add*: *assms*)

corollary *measure-conv-imp-AE-at-within*:
fixes $x :: 'a :: \textit{first-countable-topology}$
assumes [*measurable*]: $\bigwedge n.\ f'\ n \in \textit{borel-measurable}\ M\ f \in \textit{borel-measurable}\ M$
and *tendsto-AE* $M\ f'\ f\ (\textit{at}\ x\ \textit{within}\ S)$
shows *tendsto-measure* $M\ f'\ f\ (\textit{at}\ x\ \textit{within}\ S)$
proof (*rule measure-Lim-within-LIMSEQ*[*OF assms*(*1*,*2*)])
fix s **assume** $*$: $\forall n.\ s\ n \neq x \wedge s\ n \in S\ s \longrightarrow x$
have *AE-seq*: $\textit{AE}\ \omega\ \textit{in}\ M.\ \forall X.\ (\forall i.\ X\ i \in S - \{x\}) \longrightarrow X \longrightarrow x \longrightarrow ((\lambda n.\ f'\ n\ \omega) \circ X) \longrightarrow f\ \omega$
using *assms*(*3*) **by** (*simp add*: *tendsto-AE-def tendsto-at-iff-sequentially*)
then have $\textit{AE}\ \omega\ \textit{in}\ M.\ (\forall i.\ s\ i \in S - \{x\}) \longrightarrow s \longrightarrow x \longrightarrow ((\lambda n.\ f'\ n\ \omega) \circ s) \longrightarrow f\ \omega$
by *force*
then have $\textit{AE}\ \omega\ \textit{in}\ M.\ ((\lambda n.\ f'\ n\ \omega) \circ s) \longrightarrow f\ \omega$
using $*$ **by** *force*
then have *tendsto-AE* $M\ (\lambda n.\ f'\ (s\ n))\ f\ \textit{sequentially}$
unfolding *tendsto-AE-def comp-def* **by** *blast*
then show *LIMSEQ-measure* $M\ (\lambda n.\ f'\ (s\ n))\ f$
by (*rule measure-conv-imp-AE-sequentially*[*OF assms*(*1*,*2*)])
qed

Klenke remark 6.5

lemma (**in** *sigma-finite-measure*) *LIMSEQ-measure-unique-AE*:
fixes $f :: \textit{nat} \Rightarrow 'a \Rightarrow 'b :: \{\textit{second-countable-topology},\textit{metric-space}\}$
assumes [*measurable*]: $\bigwedge n.\ f\ n \in \textit{borel-measurable}\ M\ l \in \textit{borel-measurable}\ M\ l' \in \textit{borel-measurable}\ M$
and *LIMSEQ-measure* $M\ f\ l$ *LIMSEQ-measure* $M\ f\ l'$
shows $\textit{AE}\ x\ \textit{in}\ M.\ l\ x = l'\ x$
proof −
obtain $A :: \textit{nat} \Rightarrow 'a\ \textit{set}$ **where** A: $\bigwedge i.\ A\ i \in \textit{fmeasurable}\ M\ (\bigcup i.\ A\ i) = \textit{space}\ M$
by (*metis fmeasurableI infinity-ennreal-def rangeI sigma-finite subset-eq top.not-eq-extremum*)
have $\bigwedge m\ \varepsilon.\ \{x \in \textit{space}\ M.\ \textit{dist}\ (l\ x)\ (l'\ x) > \varepsilon\} \cap A\ m \in \textit{fmeasurable}\ M$
by (*intro fmeasurable-inter*; *simp add*: *A*)
then have *emeasure-leq*: $\textit{emeasure}\ M\ (\{x \in \textit{space}\ M.\ \textit{dist}\ (l\ x)\ (l'\ x) > \varepsilon\} \cap A$

$m) \leq$
emeasure M ({$x \in$ *space M*. *dist* (*l x*) (*f n x*) $> \varepsilon/2$} $\cap$ *A m*) +
emeasure M ({$x \in$ *space M*. *dist* (*f n x*) (*l' x*) $> \varepsilon/2$} $\cap$ *A m*) **if** $\varepsilon > 0$ **for** *n m* ε
proof −
have [*measurable*]:
{$x \in$ *space M*. ε / 2 < *dist* (*l x*) (*f n x*)} $\cap$ *A m* $\in$ *sets M*
{$x \in$ *space M*. ε / 2 < *dist* (*f n x*) (*l' x*)} $\cap$ *A m* $\in$ *sets M*
using *A* **by** (*measurable*; *auto*)+
have *dist* (*l x*) (*l' x*) $\leq$ *dist* (*l x*) (*f n x*) + *dist* (*f n x*) (*l' x*) **for** *x*
by (*simp add*: *dist-triangle*)
then have {*x*. *dist* (*l x*) (*l' x*) $> \varepsilon$} $\subseteq$ {*x*. *dist* (*l x*) (*f n x*) $> \varepsilon/2$} $\cup$ {*x*. *dist* (*f n x*) (*l' x*) $> \varepsilon/2$}
by (*safe*, *smt* (*verit*, *best*) *field-sum-of-halves*)
then have {$x \in$ *space M*. *dist* (*l x*) (*l' x*) $> \varepsilon$} $\cap$ *A m* $\subseteq$
({$x \in$ *space M*. *dist* (*l x*) (*f n x*) $> \varepsilon/2$} $\cap$ *A m*) $\cup$ ({$x \in$ *space M*. *dist* (*f n x*) (*l' x*) $> \varepsilon/2$} $\cap$ *A m*)
by *blast*
then have *emeasure M* ({$x \in$ *space M*. *dist* (*l x*) (*l' x*) $> \varepsilon$} $\cap$ *A m*) $\leq$
emeasure M ({$x \in$ *space M*. *dist* (*l x*) (*f n x*) $> \varepsilon/2$} $\cap$ *A m* $\cup$ {$x \in$ *space M*. *dist* (*f n x*) (*l' x*) $> \varepsilon/2$} $\cap$ *A m*)
apply (*rule emeasure-mono*)
using *A* **by** *measurable*
also have ... $\leq$ *emeasure M* ({$x \in$ *space M*. *dist* (*l x*) (*f n x*) $> \varepsilon/2$} $\cap$ *A m*) +
emeasure M ({$x \in$ *space M*. *dist* (*f n x*) (*l' x*) $> \varepsilon/2$} $\cap$ *A m*)
apply (*rule emeasure-subadditive*)
using *A* **by** *measurable*
finally show *?thesis* **.**
qed

moreover have *tendsto-zero*: (λn. *emeasure M* ({$x \in$ *space M*. ε / 2 < *dist* (*f n x*) (*l x*)} $\cap$ *A m*)
+ *emeasure M* ({$x \in$ *space M*. ε / 2 < *dist* (*f n x*) (*l' x*)} $\cap$ *A m*)) $\longrightarrow$ *0*
if ‹$\varepsilon > 0$› **for** ε *m*
apply (*rule tendsto-add-zero*)
apply (*rule LIMSEQ-measure-emeasure*[*OF assms*(*4*)])
prefer *5* **apply** (*rule LIMSEQ-measure-emeasure*[*OF assms*(*5*)])
using *that A* **apply** *simp-all*
done
have *dist-ε-emeasure*: *emeasure M* ({$x \in$ *space M*. $\varepsilon <$ *dist* (*l x*) (*l' x*)} $\cap$ *A m*) = *0*
if ‹$\varepsilon > 0$› **for** ε *m*
proof (*rule ccontr*)
assume *emeasure M* ({$x \in$ *space M*. $\varepsilon <$ *dist* (*l x*) (*l' x*)} $\cap$ *A m*) $\neq$ *0*
then obtain *e* **where** *e*: *e* > *0 emeasure M* ({$x \in$ *space M*. $\varepsilon <$ *dist* (*l x*) (*l' x*)} $\cap$ *A m*) $\geq$ *e*
using *not-gr-zero* **by** *blast*
have $\exists$ *no*. $\forall$ *n*$\geq$*no*. (*emeasure M* ({$x \in$ *space M*. ε / 2 < *dist* (*f n x*) (*l x*)} $\cap$

A m)
+ *emeasure M* ({*x* ∈ *space M*. *ε* / *2* < *dist* (*f n x*) (*l' x*)} ∩ *A m*)) < *e*
proof −
have *measure-tendsto-0*: (λ*n*. *measure M* ({*x* ∈ *space M*. *ε* / *2* < *dist* (*f n x*) (*l x*)} ∩ *A m*)
+ *measure M* ({*x* ∈ *space M*. *ε* / *2* < *dist* (*f n x*) (*l' x*)} ∩ *A m*)) ⟶ *0*
apply (*rule tendsto-add-zero*)
using *A*(*1*) *LIMSEQ-measure-def assms*(*4*,*5*) *half-gt-zero that* **by** *blast*+
have *enn2real e* > *0*
by (*metis* (*no-types*, *lifting*) *A*(*1*) *e*(*1*) *e*(*2*) *emeasure-neq-0-sets enn2real-eq-0-iff*
enn2real-nonneg fmeasurableD2 fmeasurable-inter inf-right-idem linorder-not-less nless-le top.not-eq-extremum)
then obtain *no* **where** ∀ *n*≥*no*. (*measure M* ({*x* ∈ *space M*. *ε* / *2* < *dist* (*f n x*) (*l x*)} ∩ *A m*)
+ *measure M* ({*x* ∈ *space M*. *ε* / *2* < *dist* (*f n x*) (*l' x*)} ∩ *A m*)) < *enn2real e*
using *LIMSEQ-D*[*OF measure-tendsto-0* ‹*enn2real e* > *0*›] **by** (*simp*) *blast*
then show *?thesis*
apply (*safe intro*!: *exI*[**where** *x*=*no*])
by (*smt* (*verit*, *del-insts*) *A*(*1*) *Sigma-Algebra.measure-def add-eq-0-iff-both-eq-0*
emeasure-eq-measure2 emeasure-neq-0-sets enn2real-mono enn2real-plus enn2real-top
ennreal-0 ennreal-zero-less-top fmeasurable-inter inf-sup-ord(*2*) *le-iff-inf*
linorder-not-less top.not-eq-extremum zero-less-measure-iff)
qed
then obtain *N* **where** *N*: *emeasure M* ({*x* ∈ *space M*. *ε* / *2* < *dist* (*f N x*) (*l x*)} ∩ *A m*)
+ *emeasure M* ({*x* ∈ *space M*. *ε* / *2* < *dist* (*f N x*) (*l' x*)} ∩ *A m*) < *e*
by *auto*
then have *emeasure M* ({*x* ∈ *space M*. *ε* < *dist* (*l x*) (*l' x*)} ∩ *A m*) < *e*
by (*smt* (*verit*, *del-insts*) *emeasure-leq*[*OF that*] *Collect-cong dist-commute e*(*2*) *leD order-less-le-trans*)
then show *False*
using *e*(*2*) **by** *auto*
qed
have *emeasure M* ({*x* ∈ *space M*. *0* < *dist* (*l x*) (*l' x*)} ∩ *A m*) = *0* **for** *m*
proof −
have *sets*: *range* (λ*n*. {*x* ∈ *space M*. *1*/*2*^*n* < *dist* (*l x*) (*l' x*)} ∩ *A m*) ⊆ *sets M*
using *A* **by** *force*
have (⋃*n*. {*x* ∈ *space M*. *1*/*2*^*n* < *dist* (*l x*) (*l' x*)}) = {*x* ∈ *space M*. *0* < *dist* (*l x*) (*l' x*)}
apply (*intro subset-antisym subsetI*)
apply *force*
apply *simp*
by (*metis one-less-numeral-iff power-one-over reals-power-lt-ex semiring-norm*(*76*) *zero-less-dist-iff*)

moreover have *emeasure M* ({*x* ∈ *space M*. *1/2^n* < *dist* (*l x*) (*l' x*)} ∩ *A m*) = *0* **for** *n*
using *dist-ε-emeasure* **by** *simp*
then have *suminf-zero*: (∑ *n*. *emeasure M* ({*x* ∈ *space M*. *1/2^n* < *dist* (*l x*) (*l' x*)} ∩ *A m*)) = *0*
by *auto*
then have *emeasure M* (⋃ *n*. ({*x* ∈ *space M*. *1/2^n* < *dist* (*l x*) (*l' x*)} ∩ *A m*)) ≤ *0*
apply (*subst suminf-zero*[*symmetric*])
apply (*rule emeasure-subadditive-countably*)
by (*simp add*: *emeasure-subadditive-countably sets*)
ultimately show *?thesis*
by *simp*
qed
then have (∑ *m*. *emeasure M* ({*x* ∈ *space M*. *0* < *dist* (*l x*) (*l' x*)} ∩ *A m*)) = *0*
by *simp*
then have *emeasure M* (⋃ *m*. {*x* ∈ *space M*. *0* < *dist* (*l x*) (*l' x*)} ∩ *A m*) = *0*
proof −
let *?S* = λ*m*. {*x* ∈ *space M*. *0* < *dist* (*l x*) (*l' x*)} ∩ *A m*
have *emeasure M* (⋃ *m*. *?S m*) ≤ (∑ *m*. *emeasure M* (*?S m*))
apply (*rule emeasure-subadditive-countably*)
using ‹⋀*m ε*. {*x* ∈ *space M*. *ε* < *dist* (*l x*) (*l' x*)} ∩ *A m* ∈ *fmeasurable M*› **by** *blast*
then show *?thesis*
using ‹(∑ *m*. *emeasure M* (*?S m*)) = *0*› **by** *force*
qed
then have *emeasure M* {*x* ∈ *space M*. *0* < *dist* (*l x*) (*l' x*)} = *0*
using *A* **by** *simp*
then show *?thesis*
by (*auto simp add*: *AE-iff-null*)
qed

corollary (**in** *sigma-finite-measure*) *LIMSEQ-ae-unique-AE*:
fixes *f* :: *nat* ⇒ *'a* ⇒ *'b* :: {*second-countable-topology*,*metric-space*}
assumes ⋀*n*. *f n* ∈ *borel-measurable M l* ∈ *borel-measurable M l'* ∈ *borel-measurable M*
and *tendsto-AE M f l sequentially tendsto-AE M f l' sequentially*
shows *AE x in M*. *l x* = *l' x*
proof −
have *LIMSEQ-measure M f l LIMSEQ-measure M f l'*
using *assms measure-conv-imp-AE-sequentially* **by** *blast+*
then show *?thesis*
using *assms*(*1*−*3*) *LIMSEQ-measure-unique-AE* **by** *blast*
qed

lemma (**in** *sigma-finite-measure*) *tendsto-measure-at-within-eq-AE*:
fixes *f* :: *'b* :: *first-countable-topology* ⇒ *'a* ⇒ *'c* :: {*second-countable-topology*,*metric-space*}
assumes *f-measurable*: ⋀*x*. *x* ∈ *S* ⟹ *f x* ∈ *borel-measurable M*

and *l-measurable*: $l \in$ *borel-measurable* M $l' \in$ *borel-measurable* M
and *tendsto*: *tendsto-measure* M f l (*at* t *within* S) *tendsto-measure* M f l' (*at* t *within* S)
and *not-bot*: (*at* t *within* S) $\neq \bot$
shows *AE* x *in* M. $l\ x = l'\ x$
proof −
from *not-bot* **have** t *islimpt* S
using *trivial-limit-within* **by** *blast*
then obtain $s :: nat \Rightarrow 'b$ **where** s: $\bigwedge i.\ s\ i \in S - \{t\}\ s \longrightarrow t$
using *islimpt-sequential* **by** *meson*
then have *fs-measurable*: $\bigwedge n.\ f\ (s\ n) \in$ *borel-measurable* M
using *f-measurable* **by** *blast*
have $*$: *LIMSEQ-measure* M $(\lambda n.\ f\ (s\ n))$ l
if $l \in$ *borel-measurable* M *tendsto-measure* M f l (*at* t *within* S) **for** l
proof (*intro tendsto-measureI*[*OF fs-measurable that*(*1*)], *goal-cases*)
case (*1* ε A)
then have ((*λn. measure* M ({$\omega \in$ *space* M. $\varepsilon <$ *dist* (f n ω) (l ω)} $\cap$ A)) $\longrightarrow$ *0*)(*at* t *within* S)
using *that*(*2*) *1 tendsto-measure-def* **by** *blast*
then show *?case*
apply (*rule filterlim-compose*[**where** f=s])
by (*smt* (*verit*, *del-insts*) *DiffD1 DiffD2 eventuallyI filterlim-at insertI1 s*)
qed
show *?thesis*
apply (*rule LIMSEQ-measure-unique-AE*[*OF fs-measurable l-measurable*])
using $*$ *tendsto l-measurable* **by** *simp-all*
qed
end

5 Stochastic processes

theory *Stochastic-Processes*
imports *Kolmogorov-Chentsov-Extras Dyadic-Interval*
begin

A stochastic process is an indexed collection of random variables. For compatibility with `product_prob_space` we don't enforce conditions on the index set I in the assumptions.

locale *stochastic-process* = *prob-space* +
fixes $M' :: 'b$ *measure*
and $I :: 't$ *set*
and $X :: 't \Rightarrow 'a \Rightarrow 'b$
assumes *random-process*[*measurable*]: $\bigwedge i.$ *random-variable* M' (X i)

sublocale *stochastic-process* $\subseteq$ *product*: *product-prob-space* $(\lambda t.$ *distr* M M' (X t))
using *prob-space-distr random-process* **by** (*blast intro*: *product-prob-spaceI*)

lemma (**in** *prob-space*) *stochastic-processI*:

assumes $\bigwedge i.$ *random-variable* M' $(X\ i)$
shows *stochastic-process* $M\ M'\ X$
by (*simp add*: *assms prob-space-axioms stochastic-process-axioms.intro stochastic-process-def*)

typedef $('t, 'a, 'b)$ *stochastic-process* =
$\{(M :: 'a\ measure, M' :: 'b\ measure, I :: 't\ set, X :: 't \Rightarrow 'a \Rightarrow 'b).$
stochastic-process $M\ M'\ X\}$
proof
show (*return* (*sigma UNIV* $\{\{\}, UNIV\}$) x, *sigma UNIV UNIV*, *UNIV*, λ- -. c) $\in$
$\{(M, M', I, X).$ *stochastic-process* $M\ M'\ X\}$ **for** $x :: 'a$ **and** $c :: 'b$
by (*simp add*: *prob-space-return prob-space.stochastic-processI*)
qed

setup-lifting *type-definition-stochastic-process*

lift-definition *proc-source* :: $('t,'a,'b)$ *stochastic-process* $\Rightarrow$ $'a$ *measure*
is *fst* .

interpretation *proc-source*: *prob-space proc-source X*
by (*induction*, *simp add*: *proc-source-def Abs-stochastic-process-inverse case-prod-beta' stochastic-process-def*)

lift-definition *proc-target* :: $('t,'a,'b)$ *stochastic-process* $\Rightarrow$ $'b$ *measure*
is *fst* $\circ$ *snd* .

lift-definition *proc-index* :: $('t,'a,'b)$ *stochastic-process* $\Rightarrow$ $'t$ *set*
is *fst* $\circ$ *snd* $\circ$ *snd* .

lift-definition *process* :: $('t,'a,'b)$ *stochastic-process* $\Rightarrow$ $'t \Rightarrow 'a \Rightarrow 'b$
is *snd* $\circ$ *snd* $\circ$ *snd* .

declare [[*coercion process*]]

lemma *stochastic-process-construct* [*simp*]: *stochastic-process* (*proc-source X*) (*proc-target X*) (*process X*)
by (*transfer*, *force*)

interpretation *stochastic-process proc-source X proc-target X proc-index X process X*
by *simp*

lemma *stochastic-process-measurable* [*measurable*]: *process X t* $\in$ (*proc-source X*) $\rightarrow_M$ (*proc-target X*)
by (*meson random-process*)

Here we construct a process on a given index set. For this we need to produce measurable functions for indices outside the index set; we use the

constant function, but it needs to point at an element of the target set to be measurable.

context *prob-space*
begin

lift-definition *process-of* :: *'b measure* ⇒ *'t set* ⇒ (*'t* ⇒ *'a* ⇒ *'b*) ⇒ *'b* ⇒ (*'t*,*'a*,*'b*) *stochastic-process*
  is λ *M' I X ω*. *if* (∀ *t* ∈ *I*. *X t* ∈ *M* →$_M$ *M'*) ∧ *ω* ∈ *space M'*
  then (*M*, *M'*, *I*, (λ*t*. *if t* ∈ *I then X t else* (λ-. *ω*)))
  else (*return* (*sigma UNIV* {{}, *UNIV*}) (*SOME x*. *True*), *sigma UNIV UNIV*, *I*, λ- -. *ω*)
 by (*simp add*: *stochastic-processI prob-space-return prob-space.stochastic-processI*)

lemma *index-process-of*[*simp*]: *proc-index* (*process-of M' I X ω*) = *I*
  by (*transfer*, *auto*)

lemma
  assumes ∀ *t* ∈ *I*. *X t* ∈ *M* →$_M$ *M' ω* ∈ *space M'*
  shows
    source-process-of[*simp*]: *proc-source* (*process-of M' I X ω*) = *M* **and**
    target-process-of[*simp*]: *proc-target* (*process-of M' I X ω*) = *M'* **and**
    process-process-of[*simp*]: *process* (*process-of M' I X ω*) = (λ*t*. *if t* ∈ *I then X t else* (λ-. *ω*))
  using *assms* **by** (*transfer*, *auto*)+

lemma *process-of-apply*:
  assumes ∀ *t* ∈ *I*. *X t* ∈ *M* →$_M$ *M' ω* ∈ *space M' t* ∈ *I*
  shows *process* (*process-of M' I X ω*) *t* = *X t*
  using *assms* **by** (*meson process-process-of*)
end

We define the finite-dimensional distributions of our process.

lift-definition *distributions* :: (*'t*, *'a*, *'b*) *stochastic-process* ⇒ *'t set* ⇒ (*'t* ⇒ *'b*) *measure*
  is λ(*M*, *M'*, -, *X*) *T*. (Π$_M$ *t*∈*T*. *distr M M'* (*X t*)) **.**

lemma *distributions-altdef*: *distributions X T* = (Π$_M$ *t*∈*T*. *distr* (*proc-source X*) (*proc-target X*) (*X t*))
  by (*transfer*, *auto*)

lemma *prob-space-distributions*: *prob-space* (*distributions X J*)
  unfolding *distributions-altdef*
  by (*simp add*: *prob-space-PiM proc-source.prob-space-distr random-process*)

lemma *sets-distributions*: *sets* (*distributions X J*) = *sets* (*PiM J* (λ-. (*proc-target X*)))
  by (*transfer*, *auto cong*: *sets-PiM-cong*)

lemma *space-distributions*: *space* (*distributions X J*) = (Π$_E$ *i*∈*J*. *space* (*proc-target*

X))
by (*transfer*, *auto simp add*: *space-PiM*)

lemma *emeasure-distributions*:
assumes *finite J* $\bigwedge j.\ j \in J \Longrightarrow A\ j \in sets\ (proc\text{-}target\ X)$
shows *emeasure* (*distributions X J*) (Pi_E *J A*) = ($\prod j \in J.$ *emeasure* (*distr* (*proc-source X*) (*proc-target X*) (*X j*)) (*A j*))
by (*simp add*: *assms*(*1*) *assms*(*2*) *distributions-altdef product.emeasure-PiM*)

interpretation *projective-family* (*proc-index X*) *distributions X* (λ-. *proc-target X*)
proof (*intro projective-family.intro*)
fix *J* **and** *H*
let *?I* = *proc-index X*
and *?M* = *proc-source X*
and *?M′* = *proc-target X*
assume ∗: $J \subseteq H$ *finite H* $H \subseteq$ *?I*
then have $J \subseteq$ *?I*
by *simp*
show *distributions X J* = *distr* (*distributions X H*) (Pi_M *J* (λ-. *?M′*)) (λ*f*. *restrict f J*)
proof (*rule measure-eqI*)
show *sets* (*distributions X J*) = *sets* (*distr* (*distributions X H*) (Pi_M *J* (λ-. *?M′*)) (λ*f*. *restrict f J*))
by (*simp add*: *sets-distributions*)
fix *S* **assume** $S \in$ *sets* (*distributions X J*)
then have *in-sets*: $S \in$ *sets* (*PiM J* (λ-. *?M′*))
by (*simp add*: *sets-distributions*)
have *prod-emb-distr*: (*prod-emb H* (λ-. *?M′*) *J S*) = (*prod-emb H* (λ*t*. *distr ?M ?M′* (*X t*)) *J S*)
by (*simp add*: *prod-emb-def*)
have *emeasure* (*distr* (*distributions X H*) (Pi_M *J* (λ-. *?M′*)) (λ*f*. *restrict f J*)) *S* =
emeasure (*distributions X H*) (*prod-emb H* (λ-. *?M′*) *J S*)
apply (*rule emeasure-distr-restrict*)
by (*simp-all add*: ∗ *sets-distributions in-sets*)
also have ... = *emeasure* (*distributions X J*) *S*
unfolding *distributions-altdef*
using ∗(*1*,*2*) *in-sets prod-emb-distr* **by** *force*
finally show *emeasure* (*distributions X J*) *S*
= *emeasure* (*distr* (*distributions X H*) (Pi_M *J* (λ-. *?M′*)) (λ*f*. *restrict f J*)) *S*
by *argo*
qed
qed (*rule prob-space-distributions*)

locale *polish-stochastic* = *stochastic-process M borel* :: *′b*::*polish-space measure I X*
for *M* **and** *I* **and** *X*

lemma *distributed-cong-random-variable*:
assumes $M = K$ $N = L$ *AE x in M*. $X\ x = Y\ x$ $X \in M \to_M N$ $Y \in K \to_M L$ $f \in$ *borel-measurable N*
shows *distributed M N X f* $\longleftrightarrow$ *distributed K L Y f*
using *assms* **by** (*auto simp add*: *distributed-def distr-cong-AE*)

For all sorted lists of indices, the increments specified by this list are independent

lift-definition *indep-increments* :: ('*t* :: *linorder*, '*a*, '*b* :: *minus*) *stochastic-process* $\Rightarrow$ *bool* **is**
$\lambda(M, M', I, X)$.
$(\forall\, l.\ set\ l \subseteq I \wedge sorted\ l \wedge length\ l \geq 2 \longrightarrow$
prob-space.indep-vars M (λ-. *M'*) (λ*k v*. *X* (*l*!*k*) *v* − *X* (*l*!(*k*−1)) *v*) {*1*..<*length l*}) **.**

lemma *indep-incrementsE*:
assumes *indep-increments X*
and *set l* ⊆ *proc-index X* ∧ *sorted l* ∧ *length l* ≥ *2*
shows *prob-space.indep-vars* (*proc-source X*) (λ-. *proc-target X*)
(λ*k v*. *X* (*l*!*k*) *v* − *X* (*l*!(*k*−*1*)) *v*) {*1*..<*length l*}
using *assms* **by** (*transfer*, *auto*)

lemma *indep-incrementsI*:
assumes ⋀*l*. *set l* ⊆ *proc-index X* ⟹ *sorted l* ⟹ *length l* ≥ *2* ⟹
prob-space.indep-vars (*proc-source X*) (λ-. *proc-target X*) (λ*k v*. *X* (*l*!*k*) *v* − *X* (*l*!(*k*−*1*)) *v*) {*1*..<*length l*}
shows *indep-increments X*
using *assms* **by** (*transfer*, *auto*)

lemma *indep-increments-indep-var*:
assumes *indep-increments X h* ∈ *proc-index X j* ∈ *proc-index X k* ∈ *proc-index X h* ≤ *j j* ≤ *k*
shows *prob-space.indep-var* (*proc-source X*) (*proc-target X*) (λ*v*. *X j v* − *X h v*) (*proc-target X*) (λ*v*. *X k v* − *X j v*)
proof −
let *?l* = [*h*,*j*,*k*]
have *set ?l* ⊆ *proc-index X* ∧ *sorted ?l* ∧ *2* ≤ *length ?l*
using *assms* **by** *auto*
then have *prob-space.indep-vars* (*proc-source X*) (λ-. *proc-target X*) (λ*k v*. *X* (*?l*!*k*) *v* − *X* (*?l*!(*k*−*1*)) *v*) {*1*..<*length ?l*}
by (*rule indep-incrementsE*[*OF assms*(*1*)])
then show *?thesis*
using *proc-source.indep-vars-indep-var* **by** *fastforce*
qed

definition *stationary-increments X* $\longleftrightarrow$ (∀ *t1 t2 k*. *t1* > *0* ∧ *t2* > *0* ∧ *k* > *0* ⟶

distr (*proc-source X*) (*proc-target X*) (λ*v*. *X* (*t1* + *k*) *v* − *X t1 v*) =
distr (*proc-source X*) (*proc-target X*) (λ*v*. *X* (*t2* + *k*) *v* − *X t2 v*))

Processes on the same source measure space, with the same index space, but not necessarily the same target measure since we only care about the measurable target space, not the measure

lift-definition *compatible* :: ('*t*,'*a*,'*b*) *stochastic-process* ⇒ ('*t*,'*a*,'*b*) *stochastic-process* ⇒ *bool*
is λ(*Mx*, *M'x*, *Ix*, *X*) (*My*, *M'y*, *Iy*, -). *Mx* = *My* ∧ *sets M'x* = *sets M'y* ∧ *Ix* = *Iy* **.**

lemma *compatibleI*:
assumes *proc-source X* = *proc-source Y sets* (*proc-target X*) = *sets* (*proc-target Y*)
proc-index X = *proc-index Y*
shows *compatible X Y*
using *assms* **by** (*transfer*, *auto*)

lemma
assumes *compatible X Y*
shows
compatible-source [*dest*]: *proc-source X* = *proc-source Y* **and**
compatible-target [*dest*]: *sets* (*proc-target X*) = *sets* (*proc-target Y*) **and**
compatible-index [*dest*]: *proc-index X* = *proc-index Y*
using *assms* **by** (*transfer*, *auto*)+

lemma *compatible-refl* [*simp*]: *compatible X X*
by (*transfer*, *auto*)

lemma *compatible-sym*: *compatible X Y* ⟹ *compatible Y X*
by (*transfer*, *auto*)

lemma *compatible-trans*:
assumes *compatible X Y compatible Y Z*
shows *compatible X Z*
using *assms* **by** (*transfer*, *auto*)

lemma (**in** *prob-space*) *compatible-process-of*:
assumes *measurable*: ∀ *t* ∈ *I*. *X t* ∈ *M* →$_M$ *M'* ∀ *t* ∈ *I*. *Y t* ∈ *M* →$_M$ *M'*
and *a* ∈ *space M' b* ∈ *space M'*
shows *compatible* (*process-of M' I X a*) (*process-of M' I Y b*)
using *assms* **by** (*transfer*, *auto*)

definition *modification* :: ('*t*,'*a*,'*b*) *stochastic-process* ⇒ ('*t*,'*a*,'*b*) *stochastic-process* ⇒ *bool* **where**
modification X Y ⟷ *compatible X Y* ∧ (∀ *t* ∈ *proc-index X*. *AE x in proc-source X*. *X t x* = *Y t x*)

lemma *modificationI* [*intro*]:

assumes *compatible X Y* $\bigwedge t.\ t \in$ *proc-index X* $\Longrightarrow$ *AE x in proc-source X. X t x = Y t x*
shows *modification X Y*
unfolding *modification-def* **using** *assms* **by** *blast*

lemma *modificationD* [*dest*]:
assumes *modification X Y*
shows *compatible X Y*
and $\bigwedge t.\ t \in$ *proc-index X* $\Longrightarrow$ *AE x in proc-source X. X t x = Y t x*
using *assms* **unfolding** *modification-def* **by** *blast+*

lemma *modification-null-set*:
assumes *modification X Y t* $\in$ *proc-index X*
obtains *N* **where** $\{x \in$ *space* (*proc-source X*)*. X t x* $\neq$ *Y t x*$\} \subseteq N\ N \in$ *null-sets* (*proc-source X*)
proof −
from *assms* **have** *AE x in proc-source X. X t x = Y t x*
by (*rule modificationD*(*2*))
then have $\exists N \in$*null-sets* (*proc-source X*)*.* $\{x \in$ *space* (*proc-source X*)*. X t x* $\neq$ *Y t x*$\} \subseteq N$
by (*simp add*: *eventually-ae-filter*)
then show *?thesis*
using *that* **by** *blast*
qed

lemma *modification-refl* [*simp*]: *modification X X*
by (*simp add*: *modificationI*)

lemma *modification-sym*: *modification X Y* $\Longrightarrow$ *modification Y X*
proof (*rule modificationI*)
assume $*$: *modification X Y*
then show *compat*: *compatible Y X*
using *compatible-sym modificationD*(*1*) **by** *blast*
fix *t* **assume** *t* $\in$ *proc-index Y*
then have *t* $\in$ *proc-index X*
using *compatible-index*[*OF compat*] **by** *blast*
have *AE x in proc-source Y. X t x = Y t x*
using *modificationD*(*2*)[*OF* $*$ ‹*t* $\in$ *proc-index X*›]
compatible-source[*OF compat*] **by** *argo*
then show *AE x in proc-source Y. Y t x = X t x*
by *force*
qed

lemma *modification-trans*:
assumes *modification X Y modification Y Z*
shows *modification X Z*
proof (*intro modificationI*)
show *compatible X Z*
using *compatible-trans modificationD*(*1*) *assms* **by** *blast*

fix t **assume** t: $t \in$ *proc-index* X
have XY: $AE\ x$ *in proc-source* X. *process* $X\ t\ x =$ *process* $Y\ t\ x$
by (*fact modificationD*(*2*)[*OF assms*(*1*) t])
have $t \in$ *proc-index* Y *proc-source* $X =$ *proc-source* Y
using *compatible-index compatible-source assms*(*1*) *modificationD*(*1*) t **by** *blast+*
then have $AE\ x$ *in proc-source* X. *process* $Y\ t\ x =$ *process* $Z\ t\ x$
using *modificationD*(*2*)[*OF assms*(*2*)] **by** *presburger*
then show $AE\ x$ *in proc-source* X. *process* $X\ t\ x =$ *process* $Z\ t\ x$
using XY **by** *fastforce*
qed

lemma *modification-imp-identical-distributions*:
assumes *modification*: *modification* $X\ Y$
and *index*: $T \subseteq$ *proc-index* X
shows *distributions* $X\ T =$ *distributions* $Y\ T$
proof −
have *proc-source* $X =$ *proc-source* Y
using *modification* **by** *blast*
moreover have *sets* (*proc-target* X) = *sets* (*proc-target* Y)
using *modification* **by** *blast*
ultimately have *distr* (*proc-source* X) (*proc-target* X) ($X\ x$) =
distr (*proc-source* Y) (*proc-target* Y) ($Y\ x$)
if $x \in T$ **for** x
apply (*rule distr-cong-AE*)
apply (*metis assms modificationD*(*2*) *subset-eq that*)
apply *simp-all*
done
then show *?thesis*
by (*auto simp*: *distributions-altdef cong*: *PiM-cong*)
qed

definition *indistinguishable* :: ('*t*,'*a*,'*b*) *stochastic-process* ⇒ ('*t*,'*a*,'*b*) *stochastic-process* ⇒ *bool* **where**
indistinguishable $X\ Y \longleftrightarrow$ *compatible* $X\ Y \wedge$
($\exists N \in$ *null-sets* (*proc-source* X). $\forall t \in$ *proc-index* X. $\{x \in$ *space* (*proc-source* X). $X\ t\ x \neq Y\ t\ x\} \subseteq N$)

lemma *indistinguishableI*:
assumes *compatible* $X\ Y$
and $\exists N \in$ *null-sets* (*proc-source* X). ($\forall t \in$ *proc-index* X. $\{x \in$ *space* (*proc-source* X). $X\ t\ x \neq Y\ t\ x\} \subseteq N$)
shows *indistinguishable* $X\ Y$
unfolding *indistinguishable-def* **using** *assms* **by** *blast*

lemma *indistinguishable-null-set*:
assumes *indistinguishable* $X\ Y$
obtains N **where**
$N \in$ *null-sets* (*proc-source* X)

$\bigwedge t.\ t \in$ *proc-index* $X \Longrightarrow \{x \in$ *space* (*proc-source* X). $X\ t\ x \neq Y\ t\ x\} \subseteq N$
using *assms* **unfolding** *indistinguishable-def* **by** *force*

lemma *indistinguishableD*:
assumes *indistinguishable* $X\ Y$
shows *compatible* $X\ Y$
and $\exists\, N \in$ *null-sets* (*proc-source* X). ($\forall\, t \in$ *proc-index* X. $\{x \in$ *space* (*proc-source* X). $X\ t\ x \neq Y\ t\ x\} \subseteq N$)
using *assms* **unfolding** *indistinguishable-def* **by** *blast+*

lemma *indistinguishable-eq-AE*:
assumes *indistinguishable* $X\ Y$
shows *AE x in proc-source* X. $\forall\, t \in$ *proc-index* X. $X\ t\ x = Y\ t\ x$
using *assms*[*THEN indistinguishableD*(*2*)] **by** (*auto simp add*: *eventually-ae-filter*)

lemma *indistinguishable-null-ex*:
assumes *indistinguishable* $X\ Y$
shows $\exists\, N \in$ *null-sets* (*proc-source* X). $\{x \in$ *space* (*proc-source* X).$\exists\, t \in$ *proc-index* X. $X\ t\ x \neq Y\ t\ x\} \subseteq N$
using *indistinguishableD*(*2*)[*OF assms*] **by** *blast*

lemma *indistinguishable-refl* [*simp*]: *indistinguishable* $X\ X$
by (*auto intro*: *indistinguishableI*)

lemma *indistinguishable-sym*: *indistinguishable* $X\ Y \Longrightarrow$ *indistinguishable* $Y\ X$
unfolding *indistinguishable-def* **apply** (*simp add*: *compatible-sym*)
by (*smt* (*verit*, *ccfv-SIG*) *Collect-cong compatible-index compatible-source indistinguishable-def*)

lemma *indistinguishable-trans*:
assumes *indistinguishable* $X\ Y$ *indistinguishable* $Y\ Z$
shows *indistinguishable* $X\ Z$
proof (*intro indistinguishableI*)
show *compatible* $X\ Z$
using *assms indistinguishableD*(*1*) *compatible-trans* **by** *blast*
have *eq*: *proc-index* $X =$ *proc-index* Y *proc-source* $X =$ *proc-source* Y
using *compatible-index compatible-source indistinguishableD*(*1*)[*OF assms*(*1*)] **by** *blast+*
have *AE x in proc-source* X. $\forall\, t \in$ *proc-index* X. $X\ t\ x = Y\ t\ x$
by (*fact indistinguishable-eq-AE*[*OF assms*(*1*)])
moreover have *AE x in proc-source* X. $\forall\, t \in$ *proc-index* X. $Y\ t\ x = Z\ t\ x$
apply (*subst eq*)+
by (*fact indistinguishable-eq-AE*[*OF assms*(*2*)])
ultimately have *AE x in proc-source* X. $\forall\, t \in$ *proc-index* X. $X\ t\ x = Z\ t\ x$
using *assms* **by** *fastforce*
then obtain N **where** $N \in$ *null-sets* (*proc-source* X)
$\{x \in$ *space* (*proc-source* X).$\exists\, t \in$*proc-index* X. *process* $X\ t\ x \neq$ *process* $Z\ t\ x\} \subseteq N$
using *eventually-ae-filter* **by** (*smt* (*verit*) *Collect-cong eventually-ae-filter*)

then show $\exists N{\in}$*null-sets* (*proc-source X*). $\forall t{\in}$*proc-index X*. {$x \in$ *space* (*proc-source X*). *process X t x* $\neq$ *process Z t x*} $\subseteq N$
by *blast*
qed

lemma *indistinguishable-modification*: *indistinguishable X Y* $\Longrightarrow$ *modification X Y*
apply (*intro modificationI*)
apply (*erule indistinguishableD*(*1*))
apply (*drule indistinguishableD*(*2*))
using *eventually-ae-filter* **by** *blast*

Klenke 21.5(i)

lemma *modification-countable*:
assumes *modification X Y countable* (*proc-index X*)
shows *indistinguishable X Y*
proof (*rule indistinguishableI*)
show *compatible X Y*
using *assms*(*1*) *modification-def* **by** *auto*
let *?N* = (λt. {$x \in$ *space* (*proc-source X*). *X t x* $\neq$ *Y t x*})
from *assms*(*1*) **have** $\forall\, t \in$ *proc-index X*. *AE x in proc-source X*. *X t x* = *Y t x*
unfolding *modification-def* **by** *argo*
then have $\bigwedge t$. *t* $\in$ *proc-index X* $\Longrightarrow$ $\exists\, N \in$ *null-sets* (*proc-source X*). *?N t* $\subseteq N$
by (*subst eventually-ae-filter*[*symmetric*], *blast*)
then have $\exists\, N$. $\forall\, t \in$ *proc-index X*. *N t* $\in$ *null-sets* (*proc-source X*) $\wedge$ *?N t* $\subseteq N$ *t*
by *meson*
then obtain *N* **where** *N*: $\forall\, t \in$ *proc-index X*. (*N t*) $\in$ *null-sets* (*proc-source X*) $\wedge$ *?N t* $\subseteq$ *N t*
by *blast*
then have *null*: ($\bigcup t \in$ *proc-index X*. *N t*) $\in$ *null-sets* (*proc-source X*)
by (*simp add*: *null-sets-UN′ assms*(*2*))
moreover have $\forall\, t{\in}$*proc-index X*. *?N t* $\subseteq$ ($\bigcup t \in$ *proc-index X*. *N t*)
using *N* **by** *blast*
ultimately show $\exists\, N{\in}$ *null-sets* (*proc-source X*). ($\forall\, t{\in}$*proc-index X*. *?N t* $\subseteq N$)
by *blast*
qed

Klenke 21.5(ii). The textbook statement is more general - we reduce right continuity to regular continuity

lemma *modification-continuous-indistinguishable*:
fixes *X* :: (*real*, *'a*, *'b* :: *metric-space*) *stochastic-process*
assumes *modification*: *modification X Y*
and *interval*: $\exists\, T > 0$. *proc-index X* = {*0*..*T*}
and *rc*: *AE* ω *in proc-source X*. *continuous-on* (*proc-index X*) (λt. *X t* ω)
(**is** *AE* ω *in proc-source X*. *?cont-X* ω)
AE ω *in proc-source Y*. *continuous-on* (*proc-index Y*) (λt. *Y t* ω)
(**is** *AE* ω *in proc-source Y*. *?cont-Y* ω)
shows *indistinguishable X Y*

proof (*rule indistinguishableI*)
show *compatible X Y*
using *modification modification-def* **by** *blast*
obtain *T* **where** *T*: *proc-index X* = {*0..T*} *T* > *0*
using *interval* **by** *blast*
define *N* **where** *N* ≡ λ*t*. {*x* ∈ *space* (*proc-source X*). *X t x* ≠ *Y t x*}
have *1*: ∀ *t* ∈ *proc-index X*. ∃ *S*. *N t* ⊆ *S* ∧ *S* ∈ *null-sets* (*proc-source X*)
using *modificationD*(*2*)[*OF modification*] **by** (*auto simp add*: *N-def eventually-ae-filter*)

S is a null set such that $X_t(x) \neq Y_t(x) \Longrightarrow x \in S_t$

obtain *S* **where** *S*: ∀ *t* ∈ *proc-index X*. *N t* ⊆ *S t* ∧ *S t* ∈ *null-sets* (*proc-source X*)
using *bchoice*[*OF 1*] **by** *blast*
have *eq*: *proc-source X* = *proc-source Y proc-index X* = *proc-index Y*
using ‹*compatible X Y*› *compatible-source compatible-index* **by** *blast*+
have *AE p in proc-source X*. *?cont-X p* ∧ *?cont-Y p*
apply (*rule AE-conjI*)
using *eq rc* **by** *argo*+

R is a set of measure 1 such that if $x \in R$ then the paths at x are continuous for X and Y

then obtain *R* **where** *R*: *R* ⊆ {*p* ∈ *space* (*proc-source X*). *?cont-X p* ∧ *?cont-Y p*}
R ∈ *sets* (*proc-source X*) *measure* (*proc-source X*) *R* = *1*
using *proc-source.AE-E-prob* **by** *blast*

We use an interval of dyadic rationals because we need to produce a countable dense set for $\{0..T\}$, which we have by *closure* (*dyadic-interval 0 ?T*) = {*0..?T*}.

let *?I* = *dyadic-interval 0 T*
let *?N′* = ⋃ *n* ∈ *?I*. *N n*
have *N-subset*: ⋀*t*. *t* ∈ *proc-index X* ⟹ *N t* ∩ *R* ⊆ *?N′*
proof
fix *t* **assume** *t* ∈ *proc-index X*
fix *p* **assume** ∗: *p* ∈ *N t* ∩ *R*
then obtain ε **where** ε: *0* < ε ε = *dist* (*X t p*) (*Y t p*)
by (*simp add*: *N-def*)
have *cont-p*: *continuous-on* {*0..T*} (λ*t*. *Y t p*) *continuous-on* {*0..T*} (λ*t*. *X t p*)
using *R* ∗(*1*) *T*(*1*)[*symmetric*] *eq*(*2*) **by** *auto*
then have *continuous-dist*: *continuous-on* {*0..T*} (λ*t*. *dist* (*X t p*) (*Y t p*))
using *continuous-on-dist* **by** *fast*
{
assume ∀ *r*∈ *?I*. *X r p* = *Y r p*
then have *dist-0*: ⋀*r*. *r* ∈ *?I* ⟹ *dist* (*X r p*) (*Y r p*) = *0*
by *auto*
have *dist* (*X t p*) (*Y t p*) = *0*

proof −
have *dist-tendsto-0*: $((\lambda t.\ dist\ (X\ t\ p)\ (Y\ t\ p)) \longrightarrow 0)(at\ t\ within\ ?I)$
using *dist-0 continuous-dist*
by (*smt* (*verit*, *best*) *Lim-transform-within* ε *tendsto-const*)
have *XY*: $((\lambda t.\ X\ t\ p) \longrightarrow X\ t\ p)(at\ t\ within\ ?I)\ ((\lambda t.\ Y\ t\ p) \longrightarrow Y\ t\ p)(at\ t\ within\ ?I)$
by (*metis cont-p T*(*1*) ‹$t \in$ *proc-index X*› *continuous-on-def tendsto-within-subset dyadic-interval-subset-interval*)+
show *?thesis*
apply (*rule tendsto-unique*[*of at t within ?I*])
apply (*simp add*: *trivial-limit-within*)
apply (*metis T*(*1*) *T*(*2*) ‹$t \in$ *proc-index X*› *dyadic-interval-dense islimpt-Icc limpt-of-closure*)
using *tendsto-dist*[*OF XY*] *dist-tendsto-0*
by *simp-all*
qed
then have *False*
using ε **by** *force*
}
then have $\exists r{\in}$*dyadic-interval 0 T*. $p \in N\ r$
unfolding *N-def* **using** $*$ *R*(*2*) *sets.sets-into-space* **by** *auto*
then show $p \in \bigcup\ (N\ `\ ?I)$
by *simp*
qed
have *null*: (*space* (*proc-source X*) − *R*) ∪ ($\bigcup r \in ?I.\ S\ r$) ∈ *null-sets* (*proc-source X*)
apply (*rule null-sets.Un*)
apply (*smt* (*verit*) *R*(*2*,*3*) *AE-iff-null-sets proc-source.prob-compl proc-source.prob-eq-0 sets.Diff sets.top*)
by (*metis* (*no-types*, *lifting*) *S T*(*1*) *dyadic-interval-countable dyadic-interval-subset-interval in-mono null-sets-UN′*)
have ($\bigcup r \in$ *proc-index X*. *N r*) ⊆ (*space* (*proc-source X*) − *R*) ∪ ($\bigcup r \in$ *proc-index X*. *N r*)
by *blast*
also have ... ⊆ (*space* (*proc-source X*) − *R*) ∪ ($\bigcup r \in ?I.\ N\ r$)
using *N-subset N-def* **by** *blast*
also have ... ⊆ (*space* (*proc-source X*) − *R*) ∪ ($\bigcup r \in ?I.\ S\ r$)
by (*smt* (*verit*, *ccfv-threshold*)*S T*(*1*) *UN-iff Un-iff dyadic-interval-subset-interval in-mono subsetI*)
finally show $\exists N{\in}$*null-sets* (*proc-source X*). $\forall t{\in}$*proc-index X*. {$x \in$ *space* (*proc-source X*). *process X t x* ≠ *process Y t x*} ⊆ *N*
by (*smt* (*verit*) *N-def null S SUP-le-iff order-trans*)
qed

lift-definition *restrict-index* :: ('*t*, '*a*, '*b*) *stochastic-process* ⇒ '*t set* ⇒ ('*t*, '*a*, '*b*) *stochastic-process*
is $\lambda(M,\ M',\ I,\ X)\ T.\ (M,\ M',\ T,\ X)$ **by** *fast*

lemma

shows
restrict-index-source[*simp*]: *proc-source* (*restrict-index X T*) = *proc-source X*
and
restrict-index-target[*simp*]: *proc-target* (*restrict-index X T*) = *proc-target X* **and**
restrict-index-index[*simp*]: *proc-index* (*restrict-index X T*) = *T* **and**
restrict-index-process[*simp*]: *process* (*restrict-index X T*) = *process X*
by (*transfer*, *force*)+

lemma *restrict-index-override*[*simp*]: *restrict-index* (*restrict-index X T*) *S* = *restrict-index X S*
by (*transfer*, *auto*)

lemma *compatible-restrict-index*:
assumes *compatible X Y*
shows *compatible* (*restrict-index X S*) (*restrict-index Y S*)
using *assms* **unfolding** *compatible-def* **by** (*transfer*, *auto*)

lemma *modification-restrict-index*:
assumes *modification X Y S* ⊆ *proc-index X*
shows *modification* (*restrict-index X S*) (*restrict-index Y S*)
using *assms* **unfolding** *modification-def*
apply (*simp add*: *compatible-restrict-index*)
apply (*metis restrict-index-source subsetD*)
done

lemma *indistinguishable-restrict-index*:
assumes *indistinguishable X Y S* ⊆ *proc-index X*
shows *indistinguishable* (*restrict-index X S*) (*restrict-index Y S*)
using *assms* **unfolding** *indistinguishable-def* **by** (*auto simp*: *compatible-restrict-index*)

lemma *AE-eq-minus* [*intro*]:
fixes *a* :: *'a* ⇒ (*'b* :: *real-normed-vector*)
assumes *AE x in M*. *a x* = *b x AE x in M*. *c x* = *d x*
shows *AE x in M*. *a x* − *c x* = *b x* − *d x*
using *assms* **by** *fastforce*

lemma *modification-indep-increments*:
fixes *X Y* :: (*'a* :: *linorder*, *'b*, *'c* :: {*second-countable-topology*, *real-normed-vector*}) *stochastic-process*
assumes *modification X Y sets* (*proc-target Y*) = *sets borel*
shows *indep-increments X* ⟹ *indep-increments Y*
proof (*intro indep-incrementsI*, *subst proc-source.indep-vars-iff-distr-eq-PiM*, *goal-cases*)
case (*1 l*)
then show *?case* **by** *simp*
next
case (*2 l i*)
then show *?case*
using *assms* **apply** *measurable*
using *modificationD*(*1*)[*OF assms*(*1*), *THEN compatible-source*] *assms*(*2*)

```
    by (metis measurable-cong-sets random-process)+
next
  case (3 l)
  have target-X [measurable]: sets (proc-target X) = sets borel
    using assms by auto
  then have measurable-target: f ∈ M →M proc-target X = (f ∈ borel-measurable
M) for f and M :: 'b measure
    using measurable-cong-sets by blast
  have AE ω in proc-source X. X (l ! i) ω = Y (l ! i) ω
    if i ∈ {0..<length l} for i
    apply (rule assms(1)[THEN modificationD(2)])
    by (metis 3(2) that assms(1) atLeastLessThan-iff basic-trans-rules(31)
        compatible-index modificationD(1) nth-mem)
  then have AE-eq: AE ω in proc-source X. X (l!i) ω - X (l!(i-1)) ω = Y (l!i)
ω - Y (l!(i-1)) ω
    if i ∈ {1..<length l} for i
    using AE-eq-minus that by auto
  have AE-eq': AE x in proc-source X. (λi∈{1..<length l}. X (l ! i) x - X (l ! (i
- 1)) x) =
        (λi∈{1..<length l}. Y (l ! i) x - Y (l ! (i - 1)) x)
  proof (rule AE-mp)
    show AE ω in proc-source X. ∀ i ∈{1..<length l}. X (l!i) ω - X (l!(i-1)) ω
= Y (l!i) ω - Y (l!(i-1)) ω
    proof -
      {
        fix i assume *: i ∈ {1..<length l}
        obtain N where
          {ω ∈ space (proc-source X). X (l!i) ω - X (l!(i-1)) ω ≠ Y (l!i) ω - Y
(l!(i-1)) ω} ⊆ N
          N ∈ proc-source X emeasure (proc-source X) N = 0
          using AE-eq[OF *, THEN AE-E] .
        then have ∃ N ∈ null-sets (proc-source X).
          {ω ∈ space (proc-source X). X (l!i) ω - X (l!(i-1)) ω ≠ Y (l!i) ω - Y
(l!(i-1)) ω} ⊆ N
          by blast
      } then obtain N where N: N i ∈ null-sets (proc-source X)
          {ω ∈ space (proc-source X). X (l!i) ω - X (l!(i-1)) ω ≠ Y (l!i) ω - Y
(l!(i-1)) ω} ⊆ N i
      if i ∈ {1..< length l} for i
        by (metis (lifting) ext)
      have {ω ∈ space (proc-source X). ¬ (∀ i∈{1..<length l}. X (l ! i) ω - X (l !
(i - 1)) ω = Y (l ! i) ω - Y (l ! (i - 1)) ω)} ⊆ (⋃ i ∈ {1..< length l}. N i)
        using N by blast
      moreover have (⋃ i ∈ {1..< length l}. N i) ∈ null-sets (proc-source X)
        apply (rule null-sets.finite-UN)
        using 3 N by simp-all
      ultimately show ?thesis
        by (blast intro: AE-I)
    qed
```

show *AE x in proc-source*
X. (∀ *i*∈{*1*..<*length l*}. *process X* (*l* ! *i*) *x* − *process X* (*l* ! (*i* − *1*)) *x* = *process Y* (*l* ! *i*) *x* − *process Y* (*l* ! (*i* − *1*)) *x*) ⟶
(λ*i*∈{*1*..<*length l*}. *process X* (*l* ! *i*) *x* − *process X* (*l* ! (*i* − *1*)) *x*) = (λ*i*∈{*1*..<*length l*}. *process Y* (*l* ! *i*) *x* − *process Y* (*l* ! (*i* − *1*)) *x*)
by (*rule AE-I2*, *auto*)
qed
have *distr* (*proc-source Y*) (*Pi*$_M$ {*1*..<*length l*} (λ*i*. *proc-target Y*))
(λ*x*. λ*i*∈{*1*..<*length l*}. *Y* (*l* ! *i*) *x* − *Y* (*l* ! (*i* − *1*)) *x*) =
distr (*proc-source X*) (*Pi*$_M$ {*1*..<*length l*} (λ*i*. *proc-target X*))
(λ*x*. λ*i*∈{*1*..<*length l*}. *X* (*l* ! *i*) *x* − *X* (*l* ! (*i* − *1*)) *x*)
apply (*rule sym*)
apply (*rule distr-cong-AE*)
using *assms*(*1*) **apply** *blast*
apply (*metis assms*(*2*) *sets-PiM-cong target-X*)
apply (*fact AE-eq*′)
apply *simp*
apply (*rule measurable-restrict*)
apply (*simp add*: *measurable-target*)
subgoal by *measurable* (*meson measurable-target random-process*)+
apply (*rule measurable-restrict*)
by (*metis* (*full-types*) *assms*(*2*) *borel-measurable-diff measurable-cong-sets stochastic-process-measurable*)
also have ... = *Pi*$_M$ {*1*..<*length l*} (λ*i*. *distr* (*proc-source X*) (*proc-target X*) (λ*v*. *X* (*l* ! *i*) *v* − *X* (*l* ! (*i* − *1*)) *v*))
apply (*subst proc-source.indep-vars-iff-distr-eq-PiM*[*symmetric*])
subgoal using *3* **by** *simp*
apply *simp*
apply (*metis* (*full-types*) *borel-measurable-diff measurable-cong-sets stochastic-process-measurable target-X*)
apply (*rule indep-incrementsE*)
apply (*fact 3*(*1*))
using *3*(*2*−) *assms*(*1*) **by** *blast*
also have ... = *Pi*$_M$ {*1*..<*length l*} (λ*i*. *distr* (*proc-source Y*) (*proc-target Y*) (λ*v*. *Y* (*l* ! *i*) *v* − *Y* (*l* ! (*i* − *1*)) *v*))
apply (*safe intro*!: *PiM-cong*)
apply (*rule distr-cong-AE*)
subgoal using *assms*(*1*) **by** *blast*
subgoal using *assms*(*1*) **by** *blast*
subgoal using *AE-eq* **by** *presburger*
subgoal by (*metis* (*mono-tags*) *borel-measurable-diff measurable-target random-process*)
by (*metis* (*full-types*) *assms*(*2*) *borel-measurable-diff measurable-cong-sets random-process*)
finally show *?case* **.**
qed

end

6 The Kolomgorov-Chentsov theorem

theory *Kolmogorov-Chentsov*
imports *Stochastic-Processes Holder-Continuous Dyadic-Interval Measure-Convergence*
begin

6.1 Supporting lemmas

The main contribution of this file is the Kolmogorov-Chentsov theorem: given a stochastic process that satisfies some continuity properties, we can construct a Hölder continuous modification. We first prove some auxiliary lemmas before moving on to the main construction.

Klenke 5.11: Markov inequality. Compare with ⟦(λx. *?u x* ∗ *indicator ?A x*) ∈ *borel-measurable ?M*; *?A* ∈ *sets ?M*⟧ ⟹ *emeasure ?M* {x ∈ *?A*. *1* ≤ *?c* ∗ *?u x*} ≤ *?c* ∗ *set-nn-integral ?M ?A ?u*

lemma *nn-integral-Markov-inequality-extended*:
fixes *f* :: *real* ⇒ *ennreal* **and** *ε* :: *real* **and** *X* :: *'a* ⇒ *real*
assumes *mono*: *mono-on* (*range X* ∪ {*0*<..}) *f*
and *finite*: ⋀x. *f x* < ∞
and *e*: *ε* > *0 f ε* > *0*
and [*measurable*]: *X* ∈ *borel-measurable M*
shows *emeasure M* {*p* ∈ *space M*. (*X p*) ≥ *ε*} ≤ (∫⁺ *x*. *f* (*X x*) ∂*M*) / *f ε*
proof −
have *f-eq*: *f* = (λx. *ennreal* (*enn2real* (*f x*)))
using *finite* **by** *simp*
have *mono-on* (*range X*) (λx. *enn2real* (*f x*))
apply (*intro mono-onI*)
using *mono*[*THEN mono-onD*] *finite* **by** (*simp add*: *enn2real-mono*)
then have *f* ∈ *borel-measurable* (*restrict-space borel* (*range X*))
apply (*subst f-eq*)
apply (*intro measurable-compose*[**where** *f*=λx. *enn2real* (*f x*) **and** *g*=*ennreal*])
using *borel-measurable-mono-on-fnc* **apply** *blast*
apply *simp*
done
then have (λx. *f* (*X x*)) ∈ *borel-measurable M*
apply (*intro measurable-compose*[**where** *g*=*f* **and** *f*=*X* **and** *N*=*restrict-space borel* (*range X*)])
apply (*simp-all add*: *measurable-restrict-space2*)
done
then have {x ∈ *space M*. *f* (*X x*) ≥ *f ε*} ∈ *sets M*
by *measurable*
then have *f ε* ∗ *emeasure M* {x ∈ *space M*. *X x* ≥ *ε*} ≤ (∫⁺x∈{x ∈ *space M*. *f ε* ≤ *f* (*X x*)}. *f ε* ∂*M*)
apply (*simp add*: *nn-integral-cmult-indicator*)
using *e mono-onD*[*OF mono*] *zero-le* **apply** (*blast intro*: *mult-left-mono emeasure-mono*)
done
also have ... ≤ (∫⁺x∈{x ∈ *space M*. *f ε* ≤ *f* (*X x*)}. *f* (*X x*) ∂*M*)

apply (*rule nn-integral-mono*)
subgoal for *x*
apply (*cases* $f\ \varepsilon \leq f\ (X\ x)$)
using *ennreal-leI* **by** *auto*
done
also have ... $\leq \int^+ x.\ f\ (X\ x)\ \partial M$
by (*simp add*: *nn-integral-mono indicator-def*)
finally have *emeasure M* $\{p \in$ *space M*. $\varepsilon \leq X\ p\} * f\ \varepsilon\ /\ f\ \varepsilon \leq (\int^+ x.\ f\ (X\ x)\ \partial M)\ /\ f\ \varepsilon$
by (*simp add*: *divide-right-mono-ennreal field-simps*)
then show *?thesis*
using *mult-divide-eq-ennreal finite*[*of* ε] *e*(*2*) **by** *simp*
qed

lemma *nn-integral-Markov-inequality-extended-rnv*:
fixes $f :: real \Rightarrow real$ **and** $\varepsilon :: real$ **and** $X :: 'a \Rightarrow 'b :: real\text{-}normed\text{-}vector$
assumes [*measurable*]: $X \in$ *borel-measurable M*
and *mono*: *mono-on* $\{0..\}\ f$
and *e*: $\varepsilon > 0\ f\ \varepsilon > 0$
shows *emeasure M* $\{p \in$ *space M*. *norm* $(X\ p) \geq \varepsilon\} \leq (\int^+ x.\ f\ (norm\ (X\ x))\ \partial M)\ /\ f\ \varepsilon$
apply (*rule nn-integral-Markov-inequality-extended*)
using *mono ennreal-leI* **unfolding** *mono-on-def* **apply** *force*
apply (*simp-all add*: *e*)
done

6.2 Kolmogorov-Chentsov

Klenke theorem 21.6 - Kolmorogov-Chentsov

locale *Kolmogorov-Chentsov* =
fixes $X :: (real, 'a, 'b :: polish\text{-}space)\ stochastic\text{-}process$
and $a\ b\ C\ \gamma :: real$
assumes *index*[*simp*]: *proc-index X* $= \{0..\}$
and *target-borel*[*simp*]: *proc-target X* $=$ *borel*
and *gt-0*: $a > 0\ b > 0\ C > 0$
and *b-leq-a*: $b \leq a$
and *gamma*: $\gamma \in \{0<..<b/a\}$
and *expectation*: $\bigwedge s\ t.\ [\![s \geq 0;\ t \geq 0]\!] \Longrightarrow$
$(\int^+ x.\ dist\ (X\ t\ x)\ (X\ s\ x)\ powr\ a\ \partial proc\text{-}source\ X) \leq C * dist\ t\ s\ powr\ (1+b)$
begin

lemma *gamma-0-1*[*simp*]:$\gamma \in \{0<..1\}$
using *gt-0 b-leq-a gamma*
by (*metis divide-less-eq-1-pos divide-self greaterThanAtMost-iff greaterThanLessThan-iff nless-le order-less-trans*)

lemma *gamma-gt-0*[*simp*]: $\gamma > 0$
using *gamma greaterThanLessThan-iff* **by** *blast*

lemma *gamma-le-1*[*simp*]: $\gamma \leq 1$
using *gamma-0-1* **by** *auto*

abbreviation *source* $\equiv$ *proc-source X*

lemma *X-borel-measurable*[*measurable*]: *X t* $\in$ *borel-measurable source* **for** *t*
by (*metis random-process target-borel*)

lemma *markov*: $\mathcal{P}(x$ *in source.* $\varepsilon \leq$ *dist* (*X t x*) (*X s x*)) $\leq$ (*C* $*$ *dist t s powr* (*1* + *b*)) / ε *powr a*
if $s \geq 0$ $t \geq 0$ $\varepsilon > 0$ **for** *s t* ε
proof −
let *?inc* = λx. *dist* (*X t x*) (*X s x*) *powr a*
have *emeasure source* {$x \in$ *space source.* $\varepsilon \leq$ *dist* (*X t x*) (*X s x*)}
$\leq$ *integral*^{N} *source ?inc* / ε *powr a*
apply (*rule nn-integral-Markov-inequality-extended*)
using *that*(*1*,*2*) **apply** *measurable*
subgoal using *gt-0*(*1*) *imageE powr-mono2* **by** (*auto intro*: *mono-onI*)
using *that* **apply** *simp-all*
done
also have ... $\leq$ (*C* $*$ *dist t s powr* (*1* + *b*)) / *ennreal* (ε *powr a*)
apply (*rule divide-right-mono-ennreal*)
using *expectation*[*OF that*(*1*,*2*)] *ennreal-leI* **by** *simp*
finally have *emeasure source* {$x \in$ *space source.* $\varepsilon \leq$ *dist* (*X t x*) (*X s x*)}
$\leq$ (*C* $*$ *dist t s powr* (*1* + *b*)) / ε *powr a*
using *that*(*3*) *divide-ennreal gt-0*(*3*) **by** *simp*
moreover have *C* $*$ *dist t s powr* (*1* + *b*) / ε *powr a* ≥ 0
using *gt-0*(*3*) **by** *auto*
ultimately show *?thesis*
by (*simp add*: *proc-source.emeasure-eq-measure*)
qed

lemma *conv-in-prob*:
assumes $t \geq 0$
shows *tendsto-measure* (*proc-source X*) *X* (*X t*) (*at t within* {*0*..})
proof −
{
fix p ε :: *real* **assume** $0 < p$ $0 < \varepsilon$
let *?q* = (p $*$ ε *powr a* / *C*) *powr* (*1*/(*1*+*b*))
have $0 <$ *?q*
using ‹$0 < p$› *gt-0*(*3*) ‹$0 < \varepsilon$› **by** *simp*
have *p-eq*: p = (*C* $*$ *?q powr* (*1* + *b*)) / ε *powr a*
using *gt-0* ‹$0 <$ *?q*› ‹$0 < p$› **by** (*simp add*: *field-simps powr-powr*)
have $0 <$ *dist r t* $\wedge$ *dist r t* $<$ *?q* $\longrightarrow$ *dist* $\mathcal{P}(x$ *in source.* $\varepsilon \leq$ *dist* (*X t x*) (*X r x*)) $0 \leq p$
if $r \in$ {*0*..} **for** *r*
proof *safe*
assume $0 <$ *dist r t dist r t* $<$ *?q*

```
    have 0 ≤ r
      using that by auto
    from ‹dist r t < ?q› have C * dist r t powr (1 + b) / ε powr a ≤ p
      apply (subst p-eq)
      using gt-0(2) gt-0(3) apply (simp add: divide-le-cancel powr-mono2)
      done
    then show dist P(x in source. ε ≤ dist (process X t x) (process X r x)) 0 ≤
p
      using markov[OF ‹0 ≤ r› assms ‹0 < ε›] by (simp add: dist-commute)
  qed
  then have ∃ d>0. ∀ r∈{0..}. 0 < dist r t ∧ dist r t < d ⟶
          dist P(x in source. ε ≤ dist (X r x) (X t x)) 0 ≤ p
    apply (intro exI[where x=?q])
    apply (subst(3) dist-commute)
    using ‹0 < p› gt-0(3) ‹0 < ε› dist-commute by fastforce
 } then show ?thesis
  by (simp add: finite-measure.tendsto-measure-leq, safe, intro Lim-withinI)
qed

lemma conv-in-prob-finite:
 assumes t ≥ 0
 shows tendsto-measure (proc-source X) X (X t) (at t within {0..T})
proof −
 have at t within {0..T} ≤ at t within {0..}
  by (simp add: at-le)
 then show ?thesis
  apply (rule tendsto-measure-mono)
  using assms by (rule conv-in-prob)
qed

lemma incr: P(x in source. 2 powr (− γ * n) ≤ dist (X ((k − 1) * 2 powr − n)
x) (X (k * 2 powr − n) x))
       ≤ C * 2 powr (−n * (1+b−a*γ))
 if k ≥ 1 n ≥ 0 for k n
proof −
 have P(x in source. 2 powr (− γ * n) ≤ dist (X ((k − 1) * 2 powr − n) x) (X
(k * 2 powr − n) x))
      ≤ C * dist ((k − 1) * 2 powr − n) (k * 2 powr − n) powr (1 + b) / (2
powr (− γ * n)) powr a
   using that by (auto intro: markov)
 also have ... = C * 2 powr (− n − b * n) / 2 powr (− γ * n * a)
   by (auto simp: dist-real-def powr-powr field-simps)
 also have ... =  C * 2 powr (−n * (1+b−a*γ))
   by (simp add: field-simps powr-add[symmetric])
 finally show ?thesis .
qed

end
```

In order to construct the modification of X, it suffices to construct a modi-

fication of X on $\{0..T\}$ for all finite T, from which we construct the modification on $\{0..\}$ via a countable union.

locale *Kolmogorov-Chentsov-finite* = *Kolmogorov-Chentsov* +
  fixes T :: *real*
  assumes *zero-le-T*: $0 < T$
begin

A_n will characterise the set of states with increments that exceed the bounds required for Hölder continuity. As $n \longrightarrow \infty$, this approaches the set of states for which X is not Hölder continuous. We define N as this limit, and show that N is a null set. On $\omega \in \Omega - N$, we show that $X(\omega)$ is Hölder continuous (and therefore uniformly continuious) on the dyadic rationals, and construct a modification by taking the continuous extension on the reals.

definition $A \equiv \lambda n.$ *if* $2 \hat{} n * T < 1$ *then space source else*
  $\{x \in$ *space source*.
    Max $\{$*dist* $(X$ (*real-of-int* $(k - 1) * 2$ *powr* $-$ *real* $n)$ $x)$ $(X$ (*real-of-int* $k * 2$ *powr* $-$ *real* $n)$ $x)$
      $\mid k.\ k \in \{1..\lfloor 2\hat{}n * T\rfloor\}\} \geq 2$ *powr* $(-\gamma *$ *real* $n)\}$

abbreviation $B \equiv \lambda n.\ (\bigcup m.\ A\ (m + n))$

abbreviation $N \equiv \bigcap$ (*range* B)

lemma *A-geq*: $2 \hat{} n * T \geq 1 \Longrightarrow A\ n = \{x \in$ *space source*.
    Max $\{$*dist* $(X$ (*real-of-int* $(k - 1) * 2$ *powr* $-$ *real* $n)$ $x)$ $(X$ (*real-of-int* $k * 2$ *powr* $-$ *real* $n)$ $x)$
  $\mid k.\ k \in \{1..\lfloor 2\hat{}n * T\rfloor\}\} \geq 2$ *powr* $(-\gamma *$ *real* $n)\}$ **for** n
  by (*simp add*: *A-def*)

lemma *A-measurable*[*measurable*]: $A\ n \in$ *sets source*
  unfolding *A-def* **apply** (*cases* $2 \hat{} n * T < 1$)
   apply *simp*
  apply (*simp only*: *if-False*)
  apply *measurable*
  done

lemma *emeasure-A-leq*:
  fixes n :: *nat*
  assumes [*simp*]: $2\hat{}n * T \geq 1$
  shows *emeasure source* $(A\ n) \leq C * T * 2$ *powr* $(-\ n * (b - a * \gamma))$
proof $-$
  have *nonempty*: $\{1..\lfloor 2\hat{}n * T\rfloor\} \neq \{\}$
    using *assms* **by** *fastforce*
  have *finite*: *finite* $\{1..\lfloor 2\hat{}n * T\rfloor\}$
    by *simp*
  have *emeasure source* $(A\ n) \leq$ *emeasure source* $(\bigcup k \in \{1..\lfloor 2\hat{}n * T\rfloor\}.$
   $\{x \in$*space source*. *dist* $(X$ (*real-of-int* $(k - 1) * 2$ *powr* $-$ *real* $n)$ $x)$ $(X$ (*real-of-int* $k * 2$ *powr* $-$ *real* $n)$ $x) \geq 2$ *powr* $(-\ \gamma *$ *real* $n)\})$

(**is** *emeasure source* $(A\ n) \leq$ *emeasure source ?R*)
proof (*rule emeasure-mono*, *intro subsetI*)
fix x **assume** $*$: $x \in A\ n$
from $*$ **have** *in-space*: $x \in$ *space source*
using *A-measurable sets.sets-into-space* **by** *blast*
from $*$ **have** *2 powr* $(-\gamma * real\ n) \leq Max\ \{dist\ (X\ (real\text{-}of\text{-}int\ (k-1) * 2\ powr - real\ n)\ x)\ (X\ (real\text{-}of\text{-}int\ k * 2\ powr - real\ n)\ x) \mid k.\ k \in \{1..\lfloor 2\ \hat{}\ n * T \rfloor\}\}$
using *A-geq assms* **by** *blast*
then have $\exists k \in \{1..\lfloor 2\ \hat{}\ n * T \rfloor\}.$ *2 powr* $(-\gamma * real\ n) \leq dist\ (X\ (real\text{-}of\text{-}int\ (k-1) * 2\ powr - real\ n)\ x)\ (X\ (real\text{-}of\text{-}int\ k * 2\ powr - real\ n)\ x)$
apply (*simp only*: *setcompr-eq-image*)
apply (*rule Max-finite-image-ex*[**where** $P{=}\lambda x.$ *2 powr* $(-\gamma * real\ n) \leq x$, *OF finite nonempty*])
apply (*metis Collect-mem-eq*)
done
then show $x \in$ *?R*
using *in-space* **by** *simp*
next
show *?R* $\in$ *sets source*
by *measurable*
qed
also have ... $\leq (\sum k{\in}\{1..\lfloor 2\hat{}n * T \rfloor\}.$ *emeasure source*
$\{x{\in}space\ source.\ dist\ (X\ (real\text{-}of\text{-}int\ (k-1) * 2\ powr - real\ n)\ x)\ (X\ (real\text{-}of\text{-}int\ k * 2\ powr - real\ n)\ x) \geq 2\ powr\ (-\gamma * real\ n)\}$)
apply (*rule emeasure-subadditive-finite*)
apply *blast*
apply (*subst image-subset-iff*)
apply (*intro ballI*)
apply *measurable*
done
also have ... $\leq C * 2\ powr\ (-\ n * (1 + b - a * \gamma)) * (card\ \{1..\lfloor 2\ \hat{}\ n * T \rfloor\})$
proof −
{
fix k **assume** $k \in \{1..\lfloor 2\ \hat{}\ n * T \rfloor\}$
then have *real-of-int* $k \geq 1$
by *presburger*
then have $\mathcal{P}(x$ *in source. 2 powr* $(-\gamma * real\ n) \leq dist\ (X\ (real\text{-}of\text{-}int\ (k-1) * 2\ powr - real\ n)\ x)\ (X\ (real\text{-}of\text{-}int\ k * 2\ powr - real\ n)\ x))$
$\leq C * 2\ powr\ (-(real\ n) * (1{+}b{-}a{*}\gamma))$
using *incr gamma* **by** *force*
} note $X =$ *this*
then have *sum* $(\lambda k.\ \mathcal{P}(x$ *in source. 2 powr* $(-\gamma * real\ n) \leq dist\ (X\ (real\text{-}of\text{-}int\ (k-1) * 2\ powr - real\ n)\ x)\ (X\ (real\text{-}of\text{-}int\ k * 2\ powr - real\ n)\ x)))$
$\{1..\lfloor 2\ \hat{}\ n * T \rfloor\} \leq of\text{-}nat\ (card\ \{1..\lfloor 2\ \hat{}\ n * T \rfloor\}) * (C * 2\ powr\ (-(real\ n) * (1{+}b{-}a{*}\gamma)))$
by (*fact sum-bounded-above*)
then show *?thesis*
using *ennreal-leI* **by** (*auto simp*: *proc-source.emeasure-eq-measure mult.commute*)
qed

```
  also have ... ≤ C * 2 powr (- n * (1 + b - a * γ)) * ⌊2 ^ n * T⌋
    using nonempty zle-iff-zadd by force
  also have ... ≤ C * 2 powr (- n * (1+b - a * γ)) * 2^n * T
    by (simp add:ennreal-leI gt-0(3))
  also have ... = C * 1/(2^n) * 2 powr (- n * (b - a * γ)) * 2^n * T
    apply (intro ennreal-cong)
    apply (simp add: scale-left-imp-eq field-simps)
    by (smt (verit) powr-add powr-realpow)
  also have ... = C * T * 2 powr (- n * (b - a * γ))
    by (simp add: field-simps)
  finally show ?thesis .
qed

lemma measure-A-leq:
  assumes 2^n * T ≥ 1
  shows measure source (A n) ≤ C * T * 2 powr (- n * (b - a * γ))
  apply (intro measure-leq-emeasure-ennreal)
  subgoal using gt-0(3) zero-le-T by auto
  using emeasure-A-leq apply (simp add: A-geq assms)
  done

lemma summable-A: summable (λm. measure source (A m))
proof -
  have b - a * γ > 0
   by (metis diff-gt-0-iff-gt gamma greaterThanLessThan-iff gt-0(1) mult.commute
pos-less-divide-eq)
  have 1: 2 powr (- real x * (b - a * γ)) = (1 / 2 powr (b - a * γ)) ^ x for x
    apply (cases x = 0)
     by (simp-all add: field-simps powr-add[symmetric] powr-realpow[symmetric]
powr-powr)
  have 2: summable (λn. 2 powr (- n * (b - a * γ))) (is summable ?C)
  proof -
    have summable (λn. (1 / 2 powr (b - a * γ)) ^ n)
      using ‹b - a * γ > 0› by auto
    then show summable (λx. 2 powr (- real x * (b - a * γ)))
      using 1 by simp
  qed
  from zero-le-T obtain N where 2^N * T ≥ 1
   by (metis dual-order.order-iff-strict mult.commute one-less-numeral-iff pos-divide-le-eq
      power-one-over reals-power-lt-ex semiring-norm(76) zero-less-numeral zero-less-power)
  then have ⋀n. n ≥ N ⟹ 2^n * T ≥ 1
    by (smt (verit, best) ‹0 < T› mult-right-mono power-increasing-iff)
  then have ⋀n. n ≥ N ⟹ norm (measure source (A n)) ≤ C * T * 2 powr (-
n * (b - a * γ))
    using measure-A-leq by simp
  moreover have summable (λn. C * T * 2 powr (- n * (b - a * γ)))
    using 2 summable-mult by simp
  ultimately show ?thesis
    using summable-comparison-test' by fast
```

qed

lemma *lim-B*: $(\lambda n.\ \textit{measure source}\ (B\ n)) \longrightarrow 0$
proof −
have *measure-B-le*: $\textit{measure source}\ (B\ n) \leq (\sum m.\ \textit{measure source}\ (A\ (m + n)))$
for n
apply (*rule proc-source.finite-measure-subadditive-countably*)
subgoal by *auto*
apply (*subst summable-iff-shift*)
using *summable-A* **by** *blast*
have *lim-A*: $(\lambda n.\ (\sum m.\ \textit{measure source}\ (A\ (m + n)))) \longrightarrow 0$
by (*fact suminf-exist-split2*[*OF summable-A*])
have *convergent* $(\lambda n.\ \textit{measure source}\ (B\ n))$
proof (*intro Bseq-monoseq-convergent*)
show *Bseq* $(\lambda n.\ \textit{Sigma-Algebra.measure source}\ (\bigcup m.\ A\ (m + n)))$
apply (*rule BseqI'*[**where** K=*measure source* $(\bigcup\ (\textit{range}\ A))$])
apply (*auto intro*!: *proc-source.finite-measure-mono*)
done
show *monoseq* $(\lambda n.\ \textit{Sigma-Algebra.measure source}\ (\bigcup m.\ A\ (m + n)))$
apply (*intro decseq-imp-monoseq*[*unfolded decseq-def*] *allI impI proc-source.finite-measure-mono*)
apply (*simp-all add*: *Union-add-subset*)
done
qed
then obtain L **where** *lim-B*: $(\lambda n.\ \textit{measure source}\ (B\ n)) \longrightarrow L$
unfolding *convergent-def* **by** *auto*
then have $L \geq 0$
by (*simp add*: *LIMSEQ-le-const*)
moreover have $L \leq 0$
using *measure-B-le* **by** (*simp add*: *LIMSEQ-le*[*OF lim-B lim-A*])
ultimately show *?thesis*
using *lim-B* **by** *simp*
qed

lemma *N-null*: $N \in \textit{null-sets source}$
proof −
have $(\lambda n.\ \textit{measure source}\ (B\ n)) \longrightarrow \textit{measure source}\ N$
apply (*rule proc-source.finite-Lim-measure-decseq*)
using *A-measurable* **apply** *fast*
apply (*intro monotoneI*, *simp add*: *Union-add-subset*)
done
then have *measure source* $N = 0$
using *lim-B LIMSEQ-unique* **by** *blast*
then show *?thesis*
by (*auto simp add*: *emeasure-eq-ennreal-measure*)
qed

lemma *notin-N-index*:
assumes $\omega \in \textit{space source} - N$
obtains n_0 **where** $\omega \notin (\bigcup n.\ A\ (n + n_0))$

using *assms* **by** *blast*

context
fixes ω
assumes ω: $\omega \in$ *space source* $-$ *N*
begin

definition $n_0 \equiv$ *SOME* $m.\ \omega \notin (\bigcup n.\ A\ (n + m)) \wedge m > 0$

lemma
shows *n-zero*: $\omega \notin (\bigcup n.\ A\ (n + n_0))$
and *n-zero-nonzero*: $n_0 > 0$
proof $-$
have $\exists m.\ \omega \notin (\bigcup n.\ A\ (n + m))$
using ω **by** *blast*
then have $\exists m.\ \omega \notin (\bigcup n.\ A\ (n + m)) \wedge m > 0$
by (*metis* (*no-types*, *lifting*) *UNIV-I UN-iff add.comm-neutral not-gr-zero zero-less-Suc*)
then have $\omega \notin (\bigcup n.\ A\ (n + n_0)) \wedge n_0 > 0$
unfolding n_0*-def* **by** (*rule someI-ex*)
then show $\omega \notin (\bigcup n.\ A\ (n + n_0))\ n_0 > 0$
by *blast*+
qed

lemma *nzero-ge*: $\bigwedge n.\ n \geq n_0 \Longrightarrow 2\hat{}n * T \geq 1$
proof (*rule ccontr*)
fix *n* **assume** $n_0 \leq n \neg\ 1 \leq 2\ \hat{}\ n * T$
then have *A n* = *space source*
unfolding *A-def* **by** *simp*
then have *space source* $\subseteq (\bigcup m.\ A\ (m + n))$
by (*smt* (*verit*, *del-insts*) *UNIV-I UN-upper add-0*)
also have $(\bigcup m.\ A\ (m + n)) \subseteq (\bigcup m.\ A\ (m + n_0))$
by (*simp add*: *Union-add-subset* ‹$n_0 \leq n$›)
finally show *False*
using ω *n-zero* **by** *blast*
qed

lemma *omega-notin*: $\bigwedge n.\ n \geq n_0 \Longrightarrow \omega \notin A\ n$
by (*metis n-zero UNIV-I UN-iff add.commute le-Suc-ex*)

Klenke 21.7

lemma *X-dyadic-incr*:
assumes $k \in \{1..\lfloor 2\hat{}n * T\rfloor\}$ $n \geq n_0$
shows *dist* (*X* ((*real-of-int k*-1)/$2\hat{}n$) ω) (*X* (*real-of-int k*/$2\hat{}n$) ω) $<$ *2 powr* ($-\ \gamma * n$)
proof $-$
have *finite* $\{1..\lfloor 2\hat{}n * T\rfloor\}$ $\{1..\lfloor 2\hat{}n * T\rfloor\} \neq \{\}$
using *assms nzero-ge* **by** *blast*+
then have *fin-nonempty*: *finite* $\{$*dist* (*X* (*real-of-int* $(k - 1) * 2$ *powr* $-$ *real n*) ω) (*X* (*real-of-int k* $* 2$ *powr* $-$ *real n*) ω) $|k$.

$k \in \{1..\lfloor 2 \hat{} n * T \rfloor\}\}$ $\{dist\ (X\ (real\text{-}of\text{-}int\ (k - 1) * 2\ powr - real\ n)\ \omega)\ (X\ (real\text{-}of\text{-}int\ k * 2\ powr - real\ n)\ \omega)\ |k.$
$k \in \{1..\lfloor 2 \hat{} n * T \rfloor\}\} \neq \{\}$
by *fastforce+*
have $2\ powr\ (-\gamma * real\ n)$
$> Max\ \{dist\ (X\ (real\text{-}of\text{-}int\ (k - 1) * 2\ powr - real\ n)\ \omega)\ (X\ (real\text{-}of\text{-}int\ k * 2\ powr - real\ n)\ \omega)\ |k.$
$k \in \{1..\lfloor 2 \hat{} n * T \rfloor\}\}$
using *nzero-ge*[*OF assms*(2)] *omega-notin*[*OF assms*(2)] ω *A-def* **by** *auto*
then have $2\ powr\ (-\gamma * real\ n) > dist\ (X\ (real\text{-}of\text{-}int\ (k - 1) * 2\ powr - real\ n)\ \omega)\ (X\ (real\text{-}of\text{-}int\ k * 2\ powr - real\ n)\ \omega)$
using *Max-less-iff*[*OF fin-nonempty*] *assms*(1) **by** *blast*
then show *?thesis*
by (*simp*, *smt* (*verit*, *ccfv-threshold*) *divide-powr-uminus powr-realpow*)
qed

Klenke (21.8)

lemma *dist-dyadic-mn*:
assumes *mn*: $n_0 \leq n\ n \leq m$
and *t-dyadic*: $t \in dyadic\text{-}interval\text{-}step\ m\ 0\ T$
and *u-dyadic-n*: $u \in dyadic\text{-}interval\text{-}step\ n\ 0\ T$
and *ut*: $u \leq t\ t - u < 2/2\hat{}n$
shows $dist\ (X\ u\ \omega)\ (X\ t\ \omega) \leq 2\ powr\ (-\gamma * n)\ /\ (1 - 2\ powr - \gamma)$
proof −
have *u-dyadic*: $u \in dyadic\text{-}interval\text{-}step\ m\ 0\ T$
using *mn*(2) *dyadic-interval-step-subset u-dyadic-n* **by** *fast*
have $0 < n$
using *mn*(1) *n-zero-nonzero* **by** *linarith*
then have $t - u < 1$
by (*smt* (*verit*) *ut*(2) *One-nat-def Suc-le-eq divide-le-eq-1-pos power-increasing power-one-right*)
obtain *b-tu k-tu* **where** *tu-exp*: *dyadic-expansion* $(t-u)\ m$ *b-tu k-tu*
using *dyadic-expansion-ex dyadic-interval-minus*[*OF u-dyadic t-dyadic* ‹$u \leq t$›]
by *blast*
then have *k-tu* = *0*
using *dyadic-expansion-floor*[*OF tu-exp*] ‹$t - u < 1$› ‹$u \leq t$› **by** *linarith*
have *b-tu-0-1*: *b-tu* ! $i \in \{0,1\}$ **if** $i \in \{0..m-1\}$ **for** i
using *dyadic-expansionD*(1,2)[*OF tu-exp*] *that*
by (*metis Suc-pred'* ‹$0 < n$› *atLeastAtMost-iff le-imp-less-Suc le-trans less-eq-Suc-le mn*(2) *nth-mem subsetD*)

And hence $b_i(t-u) = b_i(s-u) = 0$ for $i < n$.

have *b-t-zero*: *b-tu* ! *i* = *0* **if** *i+1* < *n* **for** *i*
proof (*rule ccontr*)
assume *b-tu* ! $i \neq 0$
then have *b-tu* ! *i* = *1*
by (*smt* (*verit*) *add-lessD1 dyadic-expansionD*(1,2) *insertE mn*(2) *nth-mem order-less-le-trans*
singletonD subset-iff that tu-exp)

```
    then have t − u ≥ (real-of-int 0) + 1/2^(i+1)
      apply (intro dyadic-expansion-nth-geq)
      using tu-exp ‹k-tu = 0› apply blast
     apply (metis One-nat-def Suc-eq-plus1 Suc-le-mono atLeastAtMost-iff le-trans
less-Suc-eq linorder-not-le mn(2) nat-less-le that zero-order(1))
      apply simp
      done
    moreover have 1/2^(n−1) ≤ 1/(2^(i+1) :: real)
      apply (intro divide-left-mono)
           apply (metis that Suc-eq-plus1 Suc-leI less-diff-conv power-increasing
power-one-right two-realpow-ge-one)
      by simp-all
    ultimately have t − u ≥ 1 / 2 ^ (n − 1)
      by linarith
    then show False
      using ‹t − u < 2/2^n› ‹n > 0› by (auto simp: power-diff)
  qed
  define t' where t' ≡ λl. (u + (∑ i = n..l. b-tu!(i−1) / 2 ^ i))
  have t' (n−1) = u
    unfolding t'-def using ‹n > 0› by simp
  have t' m = t
  proof −
    have b-tu-eq-0: (∑ i = 1..n−1. b-tu!(i−1) / 2 ^ i) = 0
      by (subst sum-nonneg-eq-0-iff, auto simp add: sum-nonneg-eq-0-iff b-t-zero)
    have t − u = (∑ i = 1..m. b-tu!(i−1) / 2 ^ i)
      using tu-exp[THEN dyadic-expansionD(3)] ‹k-tu = 0› by linarith
   also have ... = (∑ i = 1..n−1. b-tu!(i−1) / 2 ^ i) + (∑ i = n..m. b-tu!(i−1)
/ 2 ^ i)
    proof −
      have 1: {1..m} = {1..n−1} ∪ {n..m}
        using ‹n > 0› mn(2) by fastforce
      show ?thesis
        by (subst 1, auto simp: sum.union-disjoint)
    qed
    finally have t−u = (∑ i = n..m. b-tu!(i−1) / 2 ^ i)
      using b-tu-eq-0 by algebra
    then show ?thesis
      unfolding t'-def by argo
  qed
  have t-pos: t' l ≥ 0 if ‹l ∈ {n..m}› for l
    unfolding t'-def apply (rule add-nonneg-nonneg)
    using dyadic-step-geq u-dyadic apply blast
    by (simp add: sum-nonneg)
  have t'-Suc: t' (Suc l) = t' l + b-tu ! l / 2^(Suc l) if l ∈{n−1..m−1} for l
    unfolding t'-def by (simp add: b-t-zero)
  have le-add-diff: b ≤ c − a ⟹ a + b ≤ c for a b c :: real
    by argo
  have t'-leq: t' l ≤ t if ‹l ∈ {n..m}› for l
    unfolding t'-def apply (intro le-add-diff)
```

```
    apply (simp only: tu-exp[THEN dyadic-expansionD(3)] ‹k-tu = 0› of-int-0 add-0)
   apply (rule sum-mono2)
   using ‹0 < n› that by auto
 have t'-dyadic: t' l ∈ dyadic-interval-step l 0 T if l ∈ {n..m} for l
   using that
 proof (induct l rule: atLeastAtMost-induct)
   case base
   consider b-tu ! (n − 1) = 0 | b-tu ! (n−1) = 1
     using dyadic-expansion-frac-range[OF tu-exp(1), of n] ‹0 < n› mn(2) by auto
   then show ?case
     apply cases
     using ‹0 < n› t'-def apply simp
     using u-dyadic-n apply blast
     apply (rule dyadic-interval-step-memI)
      apply (simp add: t'-def)
     using u-dyadic-n
      apply (metis add-divide-distrib dyadic-interval-step-iff of-int-1 of-int-add)
     apply (simp add: mn(2) t-pos)
     by (meson t'-leq[OF that] atLeastAtMost-iff dual-order.refl dual-order.trans dyadic-step-leq mn(2) t'-leq t-dyadic)
 next
   case (Suc l)
   then have t'-dyadic-Suc: t' l ∈ dyadic-interval-step (Suc l) 0 T
     using dyadic-interval-step-mono le-SucI by blast
   from Suc have l ∈ {0..m−1}
     by force
   then consider b-tu!l = 0 | b-tu!l = 1
     using b-tu-0-1 by fastforce
   then obtain k :: int where k: t' l + (b-tu ! l) / 2 ^ Suc l = k / 2 ^ Suc l
     apply cases
     subgoal using dyadic-interval-step-iff t'-dyadic-Suc by auto
      by (metis add.commute add-divide-distrib dyadic-as-natural of-int-of-nat-eq of-nat-1 of-nat-Suc t'-dyadic-Suc)
   have t' (Suc l) ≤ t
     by (meson Suc atLeastAtMost-iff le-SucI less-eq-Suc-le t'-leq)
   with Suc(1,2) show ?case
     apply (subst t'-Suc)
       apply (metis Suc-leD Suc-pred' ‹0 < n› atLeastAtMost-iff less-Suc-eq-le mn(2) order-less-le-trans)
     apply (intro dyadic-interval-step-memI)
      apply (rule exI[where x=k])
     using k apply blast
     using dyadic-step-geq t'-dyadic-Suc apply force
     apply (subst t'-Suc[symmetric])
      apply force
     using dyadic-step-leq order-trans t-dyadic by blast
 qed
```

```
  have dist (X (t' (n−1)) ω) (X (t' m) ω) ≤ (∑ l=Suc (n−1)..m. dist (X (t' l)
ω) (X (t' (l−1)) ω))
    apply (rule triangle-ineq-sum)
    using diff-le-self dual-order.trans mn(2) by blast
  also have ... = (∑ l=n..m. dist (X (t' l) ω) (X (t' (l−1)) ω))
    using Suc-diff-1 ‹0 < n› by presburger
  also have ... ≤ (∑ l=n..m. 2 powr (−γ * l))
  proof (rule sum-mono)
    fix l assume *: l ∈ {n..m}
    then have l ∈ {n−1..m}
      by (metis atLeastAtMost-iff less-imp-diff-less linorder-not-less order-le-less)
    from * have [simp]: 0 < l
      using ‹0 < n› by fastforce
    from ‹l ∈ {n..m}› have b-tu ! (l−1) ∈ {0, 1}
      apply (intro dyadic-expansion-frac-range)
       apply (rule tu-exp)
      using ‹n > 0› by simp
    then consider (zero) b-tu ! (l−1) = 0 | (one) b-tu ! (l−1) = 1
      by fast
    then show dist (X (t' l) ω) (X (t' (l − 1)) ω) ≤ 2 powr (− γ * l)
    proof cases
      case zero
      have {n..l} = insert l {n..l−1}
        using ‹0 < n› ‹l ∈ {n..m}› by auto
      then have sum f {n..l} = sum f {n..l−1} + f l for f :: nat ⇒ real
        by (metis (no-types, opaque-lifting) Groups.add-ac(3) Suc-le-eq Suc-pred'
‹0 < n› atLeastAtMost-iff finite-atLeastAtMost group-cancel.rule0 linorder-not-le
nle-le sum.insert zero-diff zero-less-diff)
      then have t' l = t' (l−1)
        unfolding t'-def using zero by simp
      then show ?thesis
        by simp
    next
      case one
      then have [simp]: b-tu ! (l − Suc 0) = 1
        by simp
      obtain k where k: k ≥ 0 k ≤ ⌊2^l * T⌋ t' l = k/2^l
        using t'-dyadic ‹l ∈ {n..m}› dyadic-interval-step-iff by force
      have t' (l−1) ∈ dyadic-interval-step l 0 T
      proof (cases l = n)
        case True
        then have t' (l−1) = u
          using ‹t' (n − 1) = u› by presburger
        then show t' (l−1) ∈ dyadic-interval-step l 0 T
          using True u-dyadic-n by blast
      next
        case False
        then have l−1 ∈ {n..m}
       by (metis * Suc-eq-plus1 add-leD2 atLeastAtMost-iff diff-le-self dual-order.trans
```

```
      le-antisym not-less-eq-eq ordered-cancel-comm-monoid-diff-class.le-diff-conv2)
    then show t' (l−1) ∈ dyadic-interval-step l 0 T
      using t'-dyadic dyadic-interval-step-subset diff-le-self by blast
  qed
  then obtain k' where k': k' ≥ 0 k' ≤ ⌊2^l * T⌋ t' (l−1) = k'/2^l
    using dyadic-interval-step-iff by auto
  then have t'-k: t' (l−1) = (k−1) / 2^l
  proof −
    have t' l = t' (l−1) + real (b-tu ! (l−1)) / 2 ^ l
      using t'-Suc[of l−1] apply simp
      using * diff-le-mono by presburger
    then have k / 2^l = t' (l−1) + 1/2^l
      by (simp add: k(3))
    then show ?thesis
      by (simp add: diff-divide-distrib)
  qed
  then have k ≥ 1
    using k'(1) k'(3) by auto
  then show ?thesis
    apply (simp only: k(3) t'-k)
    apply (subst dist-commute)
    apply (intro less-imp-le)
    apply (simp only: of-int-diff of-int-1)
    apply (rule X-dyadic-incr[of k l])
    using k(2) apply presburger
    using ‹l ∈ {n..m}› mn(1) by auto
 qed
qed
also have ... = (∑ l=n..m. (2 powr −γ) ^ l)
 apply (intro sum.cong; simp add: field-simps)
  by (smt (verit, ccfv-SIG) powr-powr[symmetric] mult-minus-left powr-gt-zero
powr-realpow)
also have ... ≤ 2 powr (−γ * n) / (1 − 2 powr −γ)
 apply (subst sum-gp)
 using ‹m ≥ n› apply (simp add: field-simps)
 apply safe
 using gamma-gt-0 apply force
 apply (rule divide-right-mono)
  apply (simp only: minus-mult-left)
  apply (subst powr-powr[symmetric])
  apply (subst powr-realpow[symmetric]; simp)+
 by (metis diff-ge-0-iff-ge gamma-gt-0 less-eq-real-def
     neg-le-0-iff-le one-le-numeral powr-mono powr-nonneg-iff powr-zero-eq-one)
finally show dist (X u ω) (X t ω) ≤ 2 powr (− γ * real n) / (1 − 2 powr − γ)
 using ‹t' (n − 1) = u› ‹t' m = t› by blast
qed

lemma dist-dyadic-fixed:
 assumes mn: n_0 ≤ n n ≤ m
```

and *s-dyadic*: $s \in$ *dyadic-interval-step m 0 T*
and *t-dyadic*: $t \in$ *dyadic-interval-step m 0 T*
and *st*: $s \leq t\ t - s \leq 1/2\hat{}n$
shows *dist* $(X\ t\ \omega)\ (X\ s\ \omega) \leq 2 * 2\ powr\ (-\gamma * n)\ /\ (1 - 2\ powr - \gamma)$
proof −
have $n > 0$
using *mn(1) n-zero-nonzero* **by** *linarith*
define u **where** $u \equiv \lfloor 2\hat{}n * s \rfloor\ /\ 2\hat{}n$
have $u = Max\ (dyadic\text{-}interval\text{-}step\ n\ 0\ s)$
unfolding *u-def* **using** *dyadic-interval-step-Max*[*symmetric*] *dyadic-step-geq*[*OF s-dyadic*]
by *blast*
then have *u-dyadic-n*: $u \in$ *dyadic-interval-step n 0 T*
using *dyadic-interval-step-mem dyadic-step-geq dyadic-step-leq s-dyadic u-def* **by** *force*

Then, $u \leq s < u + 2^{-n}$ and $u \leq t < u + 2^{1-n}$

have $u \leq s$
unfolding *u-def* **using** *floor-pow2-leq* **by** *blast*
have $s < u + 1/2\hat{}n$
unfolding *u-def* **apply** (*simp add*: *field-simps*)
using *floor-le-iff* **apply** *linarith*
done
then have $s - u < 2/2\hat{}n$
using ‹$u \leq s$› **by** *auto*
then have *dist-us*: *dist* $(X\ u\ \omega)\ (X\ s\ \omega) \leq 2\ powr\ (-\gamma * real\ n)\ /\ (1 - 2\ powr - \gamma)$
by (*rule dist-dyadic-mn*[*OF mn s-dyadic u-dyadic-n* ‹$u \leq s$›])
have $u \leq t$
using ‹$u \leq s$› *st(1)* **by** *linarith*
have $t < u + 2/2\hat{}n$
using ‹$s <u + 1/2\hat{}n$› *st(2)* **by** *force*
then have $t - u < 2/2\hat{}n$
by *force*
then have *dist* $(X\ u\ \omega)\ (X\ t\ \omega) \leq 2\ powr\ (-\gamma * real\ n)\ /\ (1 - 2\ powr - \gamma)$
by (*rule dist-dyadic-mn*[*OF mn t-dyadic u-dyadic-n* ‹$u \leq t$›])
then show *dist* $(X\ t\ \omega)\ (X\ s\ \omega) \leq 2 * (2\ powr\ (-\gamma * n))\ /\ (1 - 2\ powr - \gamma)$
using *dist-us* **by** *metric*
qed

definition $C_0 \equiv 2 * 2\ powr\ \gamma\ /\ (1 - 2\ powr - \gamma)$

lemma *C-zero-ge*[*simp*]: $C_0 > 0$
by (*smt* (*verit*, *ccfv-SIG*) C_0*-def divide-pos-pos gamma-gt-0 powr-eq-one-iff powr-less-mono*)

Klenke (21.9)

Let $s, t \in D$ with $|s - t| \leq \frac{1}{2_0^n}$. By choosing the minimal $n \geq n_0$ such that $|t - s| \geq 2^{-n}$, we obtain by ⟦$n_0 \leq$ *?n*; *?n* $\leq$ *?m*; *?s* $\in$ *dyadic-interval-step*

?m 0 T; *?t* ∈ *dyadic-interval-step ?m 0 T*; *?s* ≤ *?t*; *?t* − *?s* ≤ *1 / 2 ^ ?n*⟧ ⟹ *dist* (*process X ?t* ω) (*process X ?s* ω) ≤ *2* ∗ *2 powr* (− γ ∗ *real ?n*) / (*1* − *2 powr* − γ):

$$|X_t(\omega) - X_s(\omega)| \leq C_0 |t - s|^\gamma$$

lemma *dist-dyadic*:
assumes *t*: *t* ∈ *dyadic-interval 0 T*
and *s*: *s* ∈ *dyadic-interval 0 T*
and *st-dist*: *dist t s* ≤ *1 / 2 ^ n*$_0$
shows *dist* (*X t* ω) (*X s* ω) ≤ C_0 ∗ (*dist t s*) *powr* γ
proof (*cases s* = *t*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
define *n* **where** *n* ≡ *LEAST n. dist t s* ≥ *1/2^n*
have *dist t s* > *0*
using *False* **by** *simp*
then have ∃ *n. dist t s* ≥ *1 / 2^n*
by (*metis less-eq-real-def one-less-numeral-iff power-one-over reals-power-lt-ex semiring-norm*(*76*))
then have *dist t s* ≥ *1/2^n*
unfolding *n-def* **by** (*meson LeastI-ex*)
then have *n* ≥ *n*$_0$
using *order-trans*[*OF* ‹*dist t s* ≥ *1/2^n*› *st-dist*]
by (*simp add*: *field-simps*)
have *dist t s* ≤ *1/2^*(*n*−*1*)
proof −
have *n*−*1* < (*LEAST n. dist t s* ≥ *1/2^n*)
using ‹*n*$_0$ ≤ *n*› *n-zero-nonzero n-def* **by** *fastforce*
then have ¬ (*dist t s* ≥ *1/2^*(*n*−*1*))
by (*rule not-less-Least*)
then show *?thesis*
by *auto*
qed
obtain *m* **where** *m*: *m* ≥ *n s* ∈ *dyadic-interval-step m 0 T t* ∈ *dyadic-interval-step m 0 T*
by (*metis dyadic-interval-step-mono linorder-not-le mem-dyadic-interval order.asym s t*)
from ‹*n* ≥ *n*$_0$› **consider** (*eq*) *n* = *n*$_0$ | (*gt*) *n* > *n*$_0$
using *less-eq-real-def* **by** *linarith*
then show *?thesis*
proof *cases*
case *eq*
consider *t* ≤ *s* | *s* ≤ *t*
by *fastforce*
then have *dist* (*X t* ω) (*X s* ω) ≤ *2* ∗ *2 powr* (− γ ∗ *n*) / (*1* − *2 powr* − γ)
apply *cases*

```
      apply (subst dist-commute)
      apply (rule dist-dyadic-fixed[OF ‹n ≥ n₀› m(1,3,2)])
       apply simp
      using dist-real-def eq st-dist apply force
      apply (rule dist-dyadic-fixed[OF ‹n ≥ n₀› m])
       apply simp
      using dist-real-def eq st-dist apply force
      done
    also have ... ≤ C₀ * 2 powr (− γ * n)
      unfolding C₀-def apply (simp add: field-simps)
      apply (intro divide-right-mono mult-left-mono)
        apply (simp add: less-eq-real-def)
       apply simp
      by (smt (verit) gamma-gt-0 powr-le-cancel-iff powr-zero-eq-one)
    also have ... = C₀ * (1/2^n) powr γ
     by (smt (verit, del-insts) powr-minus-divide powr-powr powr-powr-swap powr-realpow)
    also have ... ≤ C₀ * (dist t s) powr γ
      using ‹1 / 2 ^ n ≤ dist t s› eq st-dist by auto
    finally show ?thesis .
  next
    case gt
    consider t ≤ s | s ≤ t
      by fastforce
    then have dist (X t ω) (X s ω) ≤ 2 * 2 powr (− γ * (n−1)) / (1 − 2 powr
− γ)
      apply cases
       apply (subst dist-commute)
       apply (rule dist-dyadic-fixed[where m=m])
           prefer 7 apply (rule dist-dyadic-fixed[where m=m])
      using gt m apply simp-all
      using ‹dist t s ≤ 1 / 2 ^ (n − 1)› dist-real-def apply force+
      done
    also have ... ≤ C₀ * 2 powr (− γ * n)
      unfolding C₀-def apply simp
      apply (intro divide-right-mono)
       apply (simp add: powr-add[symmetric])
         apply (metis One-nat-def Suc-leI dual-order.refl gt less-imp-Suc-add mi-
nus-diff-eq mult.right-neutral of-nat-1 of-nat-diff right-diff-distrib zero-less-Suc)
    by (metis gamma-gt-0 ge-iff-diff-ge-0 less-eq-real-def neg-le-0-iff-le one-le-numeral
powr-mono powr-zero-eq-one zero-neq-numeral)
    also have ... = C₀ * (1/2^n) powr γ
    by (smt (verit, best) powr-minus-divide powr-powr powr-powr-swap powr-realpow)
    also have C₀ * (1/2^n) powr γ ≤ C₀ * dist t s powr γ
      apply (rule mult-left-mono)
      using ‹1 / 2 ^ n ≤ dist t s› less-eq-real-def powr-mono2 apply force
      using C-zero-ge by linarith
    finally show ?thesis .
  qed
qed
```

definition $K \equiv C_0 * (2\hat{}nat\ \lceil 2\hat{}n_0 * T\rceil)\ powr\ (1 - \gamma)$

lemma C_0*-le-K*: $C_0 \leq K$
unfolding *K-def* **using** *nzero-ge*[*of* n_0] *ge-one-powr-ge-zero* **by** *force*

lemma *K-pos*: $0 < K$
using C_0*-le-K C-zero-ge* **by** *linarith*

Klenke (21.10)

lemma *X-dyadic-le-K'*:
assumes *dyadic*: $s \in$ *dyadic-interval 0 T* $t \in$ *dyadic-interval 0 T*
and *st*: $s \leq t$
shows *dist* $(X\ s\ \omega)\ (X\ t\ \omega) \leq K * dist\ s\ t\ powr\ \gamma$
proof (*cases dist s t* $\leq$ *1 / 2*$\hat{}n_0$)
case *True*
then have $C_0 * dist\ t\ s\ powr\ \gamma \leq K * dist\ t\ s\ powr\ \gamma$
by (*simp add*: C_0*-le-K powr-def*)
then show *?thesis*
using *dist-dyadic*[*OF assms*(*1*,*2*) *True*] **by** (*simp add*: *dist-commute*)
next
case *False*
define n :: *nat* **where** $n \equiv nat\ \lceil 2\hat{}n_0 * T\rceil$
have *dist s t / n* $\leq$ *1/2*$\hat{}n_0$
apply (*simp add*: *n-def field-simps*)
by (*smt* (*verit*, *best*) *dyadic dist-real-def divide-le-eq-1 dyadics-geq dyadics-leq*
mem-dyadic-interval mult-mono of-nat-eq-0-iff of-nat-le-0-iff real-nat-ceiling-ge
zero-le-power)
have *dist-st*: *dist s t / 2*$\hat{}n \leq$ *1/2*$\hat{}n_0$
apply (*rule order-trans*[**where** *y=dist s t / n*])
apply (*rule divide-left-mono*; *simp?*)
apply (*simp add*: *n-def zero-le-T*)
apply (*fact* ‹*dist s t / n* $\leq$ *1/2*$\hat{}n_0$›)
done
define f **where** $f \equiv \lambda i{::}nat.\ (s + (t-s) * i/2\hat{}n)$
have *f-inc*: $f\ k = f\ (k - 1) + (t - s)$ */ 2*$\hat{}n$ **if** $k > 0$ **for** k
proof −
have $f\ (Suc\ k) = f\ k + (t - s)$ */ 2*$\hat{}n$ **for** k
by (*simp add*: *f-def field-simps*)
then show *?thesis*
by (*metis Suc-pred' that*)
qed
have *f-inc-le*: *dist* $(f\ i)\ (f\ (i - 1)) \leq$ *1 / 2* $\hat{}$ n_0 **for** i
proof (*cases i=0*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*

```
    then show ?thesis
      using f-inc dist-real-def dist-st st by auto
  qed
  have f-ge-s: ⋀i. i ≤ 2^n ⟹ f i ≥ s
    unfolding f-def using st by auto
  have f-le-t: ⋀i. i ≤ 2^n ⟹ f i ≤ t
     by (smt (verit, del-insts) f-def st divide-le-eq-1 mult-less-cancel-left1 of-nat-1
of-nat-add
        of-nat-le-iff of-nat-power one-add-one times-divide-eq-right zero-less-power)
  have f-dyadic: f i ∈ dyadic-interval 0 T if i ≤ 2^n for i
  proof (rule mem-dyadic-intervalI)
    have f i ≤ T
    proof −
      have f i ≤ s + t − s
        using f-le-t[OF that] by simp
      also have ... ≤ T
        using dyadic(2) dyadics-leq by simp
      finally show ?thesis .
    qed
    obtain m where s ∈ dyadic-interval-step m 0 T t ∈ dyadic-interval-step m 0
T
      by (metis dyadic dyadic-interval-step-mono mem-dyadic-interval nle-le)
    then obtain ks kt where ks: s = real ks / 2^m and kt: t = real kt / 2^m
      using dyadic-as-natural by metis
    then have ks ≤ kt
      using st by (simp add: divide-le-cancel)
    from ks kt have ks = 2^m * s kt = 2^m * t
      by simp-all
    from ks kt have f i = (ks / 2^m) + (kt / 2^m − ks / 2^m) * i / 2^n
      unfolding f-def by auto
    also have ... = (ks * (2^n − i) + kt * i) / 2^(m+n)
      using ‹ks ≤ kt› apply (simp add: right-diff-distrib field-simps power-add)
     by (metis distrib-left le-add-diff-inverse mult.commute of-nat-add of-nat-numeral
of-nat-power that)
    finally have f i ∈ dyadic-interval-step (m+n) 0 T
      apply (intro dyadic-interval-step-memI)
        apply (rule exI[where x=int (ks * (2^n − i) + kt * i)])
        prefer 3 apply (rule ‹f i ≤ T›)
      by simp-all
    then show ∃ n. f i ∈ dyadic-interval-step n 0 T
      by blast
  qed
  have dist (X s ω) (X t ω) ≤ (∑ i=1..2^n. dist (X (f i) ω) (X (f (i−1)) ω))
  proof −
    have f 0 = s
      unfolding f-def by (simp add: field-simps)
    moreover have f (2^n) = t
      unfolding f-def by (simp add: field-simps)
    moreover have dist (X (f 0) ω) (X (f (2^n)) ω) ≤ (∑ i=Suc 0..2^n. dist
```

```
(X (f i) ω) (X (f (i−1)) ω))
      by (rule triangle-ineq-sum, simp)
    ultimately show ?thesis
      by simp
  qed
  also have ... ≤ (∑ i=1..2^n::nat. C₀ * (dist (f i) (f (i−1))) powr γ)
    apply (rule sum-mono)
    apply (intro dist-dyadic f-dyadic)
     apply fastforce
    apply fastforce
    using f-inc-le .
  also have ... ≤ (∑ i=1..2^n::nat. C₀ * (dist t s / 2^n) powr γ)
    apply (rule sum-mono)
    using f-inc by (simp add: dist-real-def)
  also have ... ≤ 2^n * C₀ * (dist t s / 2^n) powr γ
    by (subst sum-constant, force)
  also have ... ≤ K * dist t s powr γ
    unfolding K-def n-def apply (simp add: powr-divide field-simps)
    apply (rule mult-left-mono)
     apply (smt (verit, ccfv-threshold) powr-add powr-one zero-le-power)
    by simp
  finally show ?thesis
    by (metis dist-commute)
qed

lemma X-dyadic-le-K:
  assumes s ∈ dyadic-interval 0 T
    and t ∈ dyadic-interval 0 T
  shows dist (X s ω) (X t ω) ≤ K * dist s t powr γ
  by (metis nle-le assms X-dyadic-le-K' dist-commute)

corollary holder-dyadic: γ−holder-on (dyadic-interval 0 T) (λt. X t ω)
  apply (intro holder-onI[OF gamma-0-1] exI[where x=K])
  using K-pos X-dyadic-le-K by force

lemma uniformly-continuous-dyadic: uniformly-continuous-on (dyadic-interval 0
T) (λt. X t ω)
  using holder-dyadic by (fact holder-uniform-continuous)

lemma Lim-exists: ∃ L. ((λs. X s ω) ⟶ L) (at t within (dyadic-interval 0 T))
  if t ∈ {0..T}
  apply (rule uniformly-continuous-on-extension-at-closure[where x = t])
  using that dyadic-interval-dense uniformly-continuous-dyadic apply fast
  using that apply (simp add: dyadic-interval-dense)
  by blast

lemma Lim-unique: ∃!L. ((λs. X s ω) ⟶ L) (at t within (dyadic-interval 0 T))

  if t ∈ {0..T}
```

by (*metis that Lim-exists dyadic-interval-islimpt tendsto-Lim trivial-limit-within zero-le-T*)

definition $L \equiv (\lambda t.\ (Lim\ (at\ t\ within\ dyadic\text{-}interval\ 0\ T)\ (\lambda s.\ X\ s\ \omega)))$

lemma *X-tendsto-L*:
assumes $t \in \{0..T\}$
shows $((\lambda s.\ X\ s\ \omega) \longrightarrow L\ t)\ (at\ t\ within\ (dyadic\text{-}interval\ 0\ T))$
proof −
have *at t within dyadic-interval 0 T* $\neq \bot$
by (*simp add*: *trivial-limit-within dyadic-interval-islimpt*[*OF zero-le-T assms*])
moreover obtain L' **where** L': $((\lambda s.\ X\ s\ \omega) \longrightarrow L')\ (at\ t\ within\ (dyadic\text{-}interval\ 0\ T))$
using *Lim-exists*[*OF assms*] **by** *blast*
ultimately have $L\ t = L'$
unfolding *L-def* **by** (*rule tendsto-Lim*)
then show *?thesis*
using L' **by** *blast*
qed

lemma *L-dist-K*:
assumes *s*: $s \in \{0..T\}$
and *t*: $t \in \{0..T\}$
shows $dist\ (L\ s)\ (L\ t) \leq K * dist\ s\ t\ powr\ \gamma$
proof (*cases* $s = t$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
let $?F = \lambda x.\ at\ x\ within\ dyadic\text{-}interval\ 0\ T$
have $(?F\ s \times_F\ ?F\ t) \neq \bot$
by (*meson dyadic-interval-islimpt prod-filter-eq-bot s t trivial-limit-within zero-le-T*)
moreover have $((\lambda x.\ K * dist\ (fst\ x)\ (snd\ x)\ powr\ \gamma) \longrightarrow K * dist\ s\ t\ powr\ \gamma)\ (?F\ s \times_F\ ?F\ t)$
apply (*rule tendsto-mult-left*)
apply (*rule tendsto-powr*)
using *tendsto-dist-prod* **apply** *blast*
apply *simp*
using *False* **by** *simp*
moreover have $((\lambda x.\ dist\ (X\ (fst\ x)\ \omega)\ (X\ (snd\ x)\ \omega)) \longrightarrow dist\ (L\ s)\ (L\ t))\ (?F\ s \times_F\ ?F\ t)$
using *X-tendsto-L t s tendsto-dist-prod* **by** *blast*
moreover have $\forall_F\ x\ in\ ?F\ s \times_F\ ?F\ t.\ dist\ (process\ X\ (fst\ x)\ \omega)\ (process\ X\ (snd\ x)\ \omega) \leq K * dist\ (fst\ x)\ (snd\ x)\ powr\ \gamma$
apply (*rule eventually-prodI'*[**where** $P = \lambda x.\ x \in dyadic\text{-}interval\ 0\ T$
and $Q = \lambda x.\ x \in dyadic\text{-}interval\ 0\ T$])
using *eventually-at-topological* **apply** *blast*
using *eventually-at-topological* **apply** *blast*

```
    using X-dyadic-le-K by simp
  ultimately show ?thesis
    by (rule tendsto-le)
qed

corollary L-holder: γ−holder-on {0..T} L
  using K-pos L-dist-K by (auto intro!: holder-onI[OF gamma-0-1] exI[where
x=K])

corollary L-local-holder: local-holder-on γ {0..T} L
  using holder-implies-local-holder[OF L-holder] by blast

lemma X-dyadic-eq-L:
  assumes t ∈ dyadic-interval 0 T
  shows X t ω = L t
proof −
  have ((λx. X x ω) ⟶ X t ω) (at t within dyadic-interval 0 T)
   using continuous-within[symmetric] uniformly-continuous-dyadic uniformly-continuous-imp-continuous
      continuous-on-eq-continuous-within assms by fast
  then show ?thesis
     by (metis L-def assms dyadic-interval-islimpt dyadic-interval-subset-interval
subsetD
       tendsto-Lim trivial-limit-within zero-le-T)
qed
end

definition default :: 'b where default = (SOME x. True)

definition X-tilde :: real ⇒ 'a ⇒ 'b where
  X-tilde ≡ (λt ω. if ω ∈ N then default else (Lim (at t within dyadic-interval 0
T) (λs. X s ω)))

lemma X-tilde-not-N-Lim:
  assumes ω ∈ space source − N
  shows X-tilde t ω = Lim (at t within dyadic-interval 0 T) (λs. X s ω)
  using assms X-tilde-def by auto

lemma X-tilde-not-N-L:
  assumes ω ∈ space source − N
  shows X-tilde t ω = L ω t
  using assms X-tilde-def L-def[OF assms] by auto

lemma local-holder-X-tilde: local-holder-on γ {0..T} (λt. X-tilde t ω)
  if ω ∈ space source for ω
proof (cases ω ∈ N)
  case True
  then show ?thesis
    unfolding X-tilde-def using local-holder-const by fastforce
next
```

case *False*
then have *1*: $\omega \in$ *space source* $-$ *N*
using *that* **by** *blast*
show *?thesis*
using *L-local-holder*[*OF 1*] *X-tilde-not-N-L*[*OF 1*]
by (*simp only*: *False if-False*)
qed

corollary *X-tilde-eq-L-AE*: *AE* ω *in source. X-tilde t* ω = *L* ω *t*
apply (*rule AE-I*[**where** *N*=*N*])
apply (*smt* (*verit*, *del-insts*) *X-tilde-def Diff-iff L-def mem-Collect-eq subsetI*)
using *N-null* **apply** *blast+*
done

corollary *X-tilde-eq-Lim-AE*:
AE ω *in source. X-tilde t* ω = *Lim* (*at t within dyadic-interval 0 T*) (λ*s. X s* ω)
apply (*rule AE-I*[**where** *N*=*N*])
apply (*smt* (*verit*, *del-insts*) *X-tilde-def Diff-iff L-def mem-Collect-eq subsetI*)
using *N-null* **apply** *blast+*
done

lemma *X-tilde-tendsto-AE*: $t \in \{0..T\} \Longrightarrow$ *tendsto-AE source X* (*X-tilde t*) (*at t within dyadic-interval 0 T*)
apply (*unfold tendsto-AE-def*)
apply (*rule AE-I3*[**where** *N*=*N*])
apply (*subst X-tilde-not-N-Lim*, *argo*)
unfolding *t2-space-class.Lim-def* **apply** (*rule the1I2*)
using *Lim-unique* **apply** *presburger*
apply *blast*
using *N-null* **by** *blast*

end

context *Kolmogorov-Chentsov-finite*
begin

By (21.5) $0 \leq ?t \Longrightarrow$ *tendsto-measure source* (*process X*) (*process X ?t*) (*at ?t within* $\{0..?T\}$) and (21.11) $?\omega \in$ *space source* $-$ $(\bigcap_n \bigcup_m A\ (m + n)) \Longrightarrow L\ ?\omega \equiv \lambda t.$ *Lim* (*at t within dyadic-interval 0 T*) (λ*s. process X s ?*ω), $P[X \neq \tilde{X}] = 0$

lemma *X-tilde-measurable*[*measurable*]:
assumes $t \in \{0..T\}$
shows *X-tilde t* $\in$ *borel-measurable source*
proof $-$
let *?Lim* = ($\lambda\omega$. *Lim* (*at t within dyadic-interval 0 T*) (λ*s. process X s* ω))
have *?Lim* $\in$ *borel-measurable* (*restrict-space source* (*space source* $-$ *N*))
unfolding *X-tilde-def* **apply** *measurable*
using *measurable-id measurable-restrict-space1* **apply** *blast*
using *assms Lim-exists space-restrict-space* **apply** *simp*

using *assms dyadic-interval-islimpt trivial-limit-within zero-le-T* **by** *blast*
then have (*λω. if ω ∈ space source − N then ?Lim ω else default*) ∈ *borel-measurable source*
by (*subst measurable-restrict-space-iff*[*symmetric*]; *simp*)
then show *?thesis*
apply (*subst measurable-cong*[**where** *g*=(*λω. if ω ∈ space source − N then ?Lim ω else default*)])
unfolding *X-tilde-def* **by** *auto*
qed

lemma *X-eq-X-tilde-AE*: *AE ω in source. X t ω = X-tilde t ω* **if** *t* ∈ {*0..T*} **for** *t*
apply (*rule sigma-finite-measure.tendsto-measure-at-within-eq-AE*[**where** *f*=*process X* **and** *S*=*dyadic-interval 0 T*])
using *proc-source.sigma-finite-measure-axioms* **apply** *blast*
using *X-borel-measurable* **apply** *blast*
apply *measurable*
using *X-tilde-def* **apply** *simp*
using *that* **apply** *simp*
using *tendsto-measure-mono*[*OF at-le*[*OF dyadic-interval-subset-interval*] *conv-in-prob-finite*] *that*
apply *force*
using *X-borel-measurable X-tilde-measurable X-tilde-tendsto-AE measure-conv-imp-AE-at-within that* **apply** *blast*
using *dyadic-interval-islimpt that trivial-limit-within zero-le-T* **by** *blast*

lemma *X-tilde-modification*: *modification* (*restrict-index X* {*0..T*})
(*prob-space.process-of source* (*proc-target X*) {*0..T*} *X-tilde default*)
apply (*intro modificationI compatibleI*)
apply *simp-all*
apply (*subst restrict-index-source*)
apply (*auto simp*: *X-eq-X-tilde-AE*)
done
end

We have now shown that we can construct a modification of X for any interval $\{0..T\}$. We want to extend this result to construct a modification on the interval $\{0..\}$ - this can be constructed by gluing together all modifications with natural-valued T which results in a countable union of modifications, which itself is a modification.

context *Kolmogorov-Chentsov*
begin

lemma *Kolmogorov-Chentsov-finite*: *T* > *0* ⟹ *Kolmogorov-Chentsov-finite X a b C γ T*
by (*simp add*: *Kolmogorov-Chentsov-axioms Kolmogorov-Chentsov-finite.intro Kolmogorov-Chentsov-finite-axioms-def*)

definition *Mod* ≡ *λT. SOME Y. modification* (*restrict-index X* {*0..T*}) *Y* ∧
(∀ *x*∈*space source. local-holder-on γ* {*0..T*} (*λt. Y t x*))

lemma *Mod*: *modification* (*restrict-index X* {*0..T*}) (*Mod T*)
(∀ *x*∈*space source*. *local-holder-on γ* {*0..T*} (λ*t*. (*Mod T*) *t x*)) **if** *0* < *T* **for** *T*
proof −
interpret *Kolmogorov-Chentsov-finite X a b C γ T*
using *that* **by** (*simp add*: *Kolmogorov-Chentsov-finite*)
have *modification* (*restrict-index X* {*0..T*}) (*Mod T*) ∧
(∀ *x*∈*space source*. *local-holder-on γ* {*0..T*} (λ*t*. (*Mod T*) *t x*))
unfolding *Mod-def* **apply** (*rule someI-ex*)
apply (*rule exI*[**where** *x*=*prob-space.process-of source* (*proc-target X*) {*0..T*}
X-tilde default])
apply *safe*
apply (*fact X-tilde-modification*)
apply (*subst local-holder-on-cong*[*OF refl refl*])
using *local-holder-X-tilde X-tilde-measurable* **apply** (*auto cong*: *local-holder-on-cong*)
done
then show *modification* (*restrict-index X* {*0..T*}) (*Mod T*)
(∀ *x*∈*space source*. *local-holder-on γ* {*0..T*} (λ*t*. (*Mod T*) *t x*))
by *blast*+
qed

lemma *compatible-Mod*: *compatible* (*restrict-index X* {*0..T*}) (*Mod T*) **if** *0* < *T*
for *T*
using *Mod that modificationD*(*1*) **by** *blast*

lemma *Mod-source*[*simp*]: *proc-source* (*Mod T*) = *source* **if** *0* < *T* **for** *T*
by (*metis compatible-Mod compatible-source restrict-index-source that*)

lemma *Mod-target*: *sets* (*proc-target* (*Mod T*)) = *sets* (*proc-target X*) **if** *0* < *T*
for *T*
by (*metis compatible-Mod*[*OF that*] *compatible-target restrict-index-target*)

lemma *Mod-index*[*simp*]: *0* < *T* ⟹ *proc-index* (*Mod T*) = {*0..T*}
using *compatible-Mod*[*THEN compatible-index*] **by** *simp*

lemma *indistinguishable-mod*:
indistinguishable (*restrict-index* (*Mod S*) {*0..min S T*}) (*restrict-index* (*Mod T*)
{*0..min S T*})
if *S* > *0* *T* > *0* **for** *S T*
proof −
have ∗: *modification* (*restrict-index* (*Mod S*) {*0..min S T*}) (*restrict-index* (*Mod*
T) {*0..min S T*})
proof −
have *modification* (*restrict-index X* {*0..min S T*}) (*restrict-index* (*Mod S*)
{*0..min S T*})
apply (*cases S* ≤ *T*)
using *Mod*(*1*)[*OF that*(*1*)] **apply** *clarsimp*
apply (*metis modification-restrict-index order-refl restrict-index-index re-*
strict-index-override)

using *Mod*(*1*)[*OF that*(*2*)] **apply** *clarsimp*
by (*metis* (*mono-tags*, *opaque-lifting*) ‹*modification* (*restrict-index X* {*0*..*S*}) (*Mod S*)› *atLeastatMost-subset-iff modification-restrict-index nle-le restrict-index-index restrict-index-override*)
moreover have *modification* (*restrict-index X* {*0*..*min S T*}) (*restrict-index* (*Mod T*) {*0*..*min S T*})
apply (*cases S* ≤ *T*)
using *Mod*(*1*)[*OF that*(*1*)] **apply** *clarsimp*
apply (*metis Mod*(*1*) *atLeastatMost-subset-iff modification-restrict-index order.refl restrict-index-index restrict-index-override that*(*2*))
using *Mod*(*1*)[*OF that*(*2*)] **apply** *clarsimp*
by (*metis* (*mono-tags*, *opaque-lifting*) *atLeastatMost-subset-iff modification-restrict-index nle-le restrict-index-index restrict-index-override*)
ultimately show *?thesis*
using *modification-sym modification-trans* **by** *metis*
qed
then show *?thesis*
proof (*rule modification-continuous-indistinguishable*,
goal-cases - continuous-S continuous-T)
show ∃ *Ta*>*0*. *proc-index* (*restrict-index* (*Mod S*) {*0*..*min S T*}) = {*0*..*Ta*}
by (*metis that min-def restrict-index-index*)
next
case (*continuous-S*)
have ∀ *x* ∈ *space source*. *continuous-on* {*0*..*S*} (λ*t*. (*Mod S*) *t x*)
using *Mod*[*OF that*(*1*)] *local-holder-imp-continuous* **by** *blast*
then have ∀ *x* ∈ *space source*. *continuous-on* {*0*..*min S T*} (λ*t*. (*Mod S*) *t x*)
using *continuous-on-subset* **by** *fastforce*
then show *?case*
by (*auto simp*: *that*(*1*))
next
case (*continuous-T*)
have ∀ *x* ∈ *space source*. *continuous-on* {*0*..*T*} (λ*t*. (*Mod T*) *t x*)
using *Mod*[*OF that*(*2*)] *local-holder-imp-continuous* **by** *fast*
then have *2*: ∀ *x* ∈ *space source*. *continuous-on* {*0*..*min S T*} (λ*t*. (*Mod T*) *t x*)
using *continuous-on-subset* **by** *fastforce*
then show *?case*
by (*auto simp*: *that*(*2*))
qed
qed

definition *N S T* ≡ *SOME N*. *N* ∈ *null-sets source* ∧ {ω ∈ *space source*. ∃ *t* ∈ {*0*..*min S T*}. (*Mod S*) *t* ω ≠ (*Mod T*) *t* ω} ⊆ *N*

lemma *N*:
assumes *S* > *0 T* > *0*
shows *N S T* ∈ *null-sets source* ∧ {ω ∈ *space source*. ∃ *t* ∈ {*0*..*min S T*}. (*Mod S*) *t* ω ≠ (*Mod T*) *t* ω} ⊆ *N S T*
proof −

have *ex*: $\forall S > 0.\ \forall T > 0.\ \exists N.\ N \in$ *null-sets source* $\wedge \{\omega \in$ *space source*. $\exists t \in \{0..min\ S\ T\}.$
$(Mod\ S)\ t\ \omega \neq (Mod\ T)\ t\ \omega\} \subseteq N$
apply (*fold Bex-def*)
using *indistinguishable-mod*[*THEN indistinguishable-null-ex*] **by** *simp*
show *?thesis*
unfolding *N-def* **apply** (*rule someI-ex*)
using *ex assms* **by** *blast*
qed

definition *N-inf* **where** $N\text{-}inf \equiv (\bigcup S \in \mathbb{N} - \{0\}.\ (\bigcup\ T \in \mathbb{N} - \{0\}.\ N\ S\ T))$

lemma *N-inf-null*: *N-inf* $\in$ *null-sets source*
unfolding *N-inf-def*
apply (*intro null-sets-UN'*)
apply (*rule countable-Diff*)
apply (*simp add*: *Nats-def*)+
using *N* **by** *force*

lemma *Mod-eq-N-inf*: $(Mod\ S)\ t\ \omega = (Mod\ T)\ t\ \omega$
if $\omega \in$ *space source* $-$ *N-inf* $t \in \{0..min\ S\ T\}\ S \in \mathbb{N} - \{0\}\ T \in \mathbb{N} - \{0\}$ **for** $\omega\ t\ S\ T$
proof $-$
have $\omega \in$ *space source* $-\ N\ S\ T$
using *that*(*1*,*3*,*4*) **unfolding** *N-inf-def* **by** *blast*
moreover have $S > 0\ T > 0$
using *that*(*2*,*3*,*4*) **by** *auto*
ultimately have $\omega \in \{\omega \in$ *space source*. $\forall t \in \{0..min\ S\ T\}.\ (Mod\ S)\ t\ \omega = (Mod\ T)\ t\ \omega\}$
using *N* **by** *auto*
then show *?thesis*
using *that*(*2*) **by** *blast*
qed

definition *default* :: *'b* **where** $default = (SOME\ x.\ True)$

definition $X\text{-}mod \equiv \lambda t\ \omega.$ *if* $\omega \in$ *space source* $-$ *N-inf then* $(Mod\ \lfloor t+1 \rfloor)\ t\ \omega$ *else default*

definition *X-mod-process* $\equiv$ *prob-space.process-of source* (*proc-target X*) $\{0..\}$ *X-mod default*

lemma *Mod-measurable*[*measurable*]: $\forall t \in \{0..\}.$ *X-mod* $t \in$ *source* $\rightarrow_M$ *proc-target X*
proof (*intro ballI*)
fix *t* :: *real* **assume** $t \in \{0..\}$
then have $0 < \lfloor t + 1 \rfloor$
by *force*
then show *X-mod* $t \in$ *source* $\rightarrow_M$ *proc-target X*

```
    unfolding X-mod-def apply measurable
      apply (subst measurable-cong-sets[where M'= proc-source (Mod ⌊t + 1⌋)
and N' = proc-target (Mod ⌊t + 1⌋)])
    using Mod-source ‹0 < ⌊t + 1⌋› apply presburger
    using Mod-target ‹0 < ⌊t + 1⌋› apply presburger
     apply (meson random-process)
    apply simp
    apply (metis N-inf-null Int-def null-setsD2 sets.Int-space-eq1 sets.compl-sets)
    done
qed

lemma modification-X-mod-process: modification X X-mod-process
proof (intro modificationI ballI)
  show compatible X X-mod-process
    apply (intro compatibleI)
    unfolding X-mod-process-def
      apply (subst proc-source.source-process-of)
    using Mod-measurable proc-source.prob-space-axioms apply auto
    done
  fix t assume t ∈ proc-index X
  then have t ∈ {0..}
    by simp
  then have real-of-int ⌊t⌋ + 1 > 0
    by (simp add: add.commute add-pos-nonneg)
  then have ∃ N ∈ null-sets source. ∀ ω ∈ space source − N.
      X t ω = (prob-space.process-of source (proc-target X) {0..} X-mod default) t
ω
  proof −
    have 1: (prob-space.process-of source (proc-target X) {0..} X-mod default) t ω
= (Mod (real-of-int ⌊t⌋ + 1)) t ω
      if ω ∈ space source − N-inf for ω
      apply (subst proc-source.process-process-of)
        apply measurable
      unfolding X-mod-def using that ‹t ∈ {0..}› apply simp
      done
   have ∃ N ∈ null-sets source. ∀ ω ∈ space source − N. X t ω = (Mod (real-of-int
⌊t⌋ + 1)) t ω
    proof −
      obtain N where {x ∈ space source. (restrict-index X {0..real-of-int ⌊t⌋ +
1}) t x ≠ (Mod (real-of-int ⌊t⌋ + 1)) t x} ⊆ N ∧
    N ∈ null-sets (proc-source (restrict-index X {0..(real-of-int ⌊t⌋ + 1)}))
        using Mod(1)[OF ‹real-of-int ⌊t⌋ + 1 > 0›, THEN modification-null-set,
of t]
        using ‹t ∈ {0..}› by fastforce
      then show ?thesis
        by (smt (verit, ccfv-threshold) DiffE mem-Collect-eq restrict-index-process
restrict-index-source subset-eq)
    qed
    then obtain N where N ∈ null-sets source ∀ ω ∈ space source − N. X t ω =
```

```
(Mod (real-of-int ⌊t⌋ + 1)) t ω
      by blast
    then show ?thesis
      apply (intro bexI[where x=N ∪ N-inf])
       apply (metis 1 DiffE DiffI UnCI)
      using N-inf-null by blast
  qed
  then show AE x in source. X t x = X-mod-process t x
   by (smt (verit, del-insts) X-mod-process-def DiffI eventually-ae-filter mem-Collect-eq
subsetI)
qed

lemma local-holder-X-mod: local-holder-on γ {0..} (λt. X-mod t ω) for ω
proof (cases ω ∈ space source − N-inf)
  case False
  then show ?thesis
    apply (simp only: X-mod-def)
    apply (metis local-holder-const gamma-0-1)
    done
next
  case True
  then have ω: ω ∈ space source ω ∉ N-inf
    by simp-all
  then show ?thesis
  proof (intro local-holder-ballI[OF gamma-0-1] ballI)
    fix t::real assume t ∈ {0..}
    then have real-of-int ⌊t⌋ + 1 > 0
      using floor-le-iff by force
    have t < real-of-int ⌊t⌋ + 1
      by simp
    then have local-holder-on γ {0..real-of-int ⌊t⌋ + 1} (λs. Mod (real-of-int ⌊t⌋
+ 1) s ω)
      using Mod(2) ω(1) ‹real-of-int ⌊t⌋ + 1 > 0› by blast
    then obtain ε C where eC: ε > 0 C ≥ 0 ⋀r s. r ∈ ball t ε ∩ {0..real-of-int
⌊t⌋ + 1} ⟹
       s ∈ ball t ε ∩ {0..real-of-int ⌊t⌋ + 1} ⟹
       dist (Mod (real-of-int ⌊t⌋ + 1) r ω) (Mod (real-of-int ⌊t⌋ + 1) s ω) ≤ C *
dist r s powr γ
      apply −
      apply (erule local-holder-onE)
       apply (rule gamma-0-1)
      using ‹t ∈ {0..}› ‹t < real-of-int ⌊t⌋ + 1› apply fastforce
      apply blast
      done
    define ε' where ε' = min ε (if frac t = 0 then 1/2 else 1 − frac t)
    have e': ε' ≤ ε ∧ ε' > 0 ∧ ball t ε' ∩ {0..} ⊆ {0..real-of-int ⌊t⌋ + 1}
      unfolding ε'-def apply (simp add: eC(1))
      apply safe
       apply (simp-all add: eC(1) dist-real-def frac-lt-1 frac-floor)
```

apply *argo+*
done
{
fix *r s*
assume *r*: $r \in ball\ t\ \varepsilon' \cap \{0..\}$
assume *s*: $s \in ball\ t\ \varepsilon' \cap \{0..\}$

then have *rs-ball*: $r \in ball\ t\ \varepsilon \cap \{0..real\text{-}of\text{-}int\ \lfloor t \rfloor + 1\}$
$s \in ball\ t\ \varepsilon \cap \{0..real\text{-}of\text{-}int\ \lfloor t \rfloor + 1\}$
using $e'\ r\ s$ **by** *auto*
then have $r \in \{0..min\ (real\text{-}of\text{-}int\ \lfloor t \rfloor + 1)\ (real\text{-}of\text{-}int\ \lfloor r + 1 \rfloor)\}$
by *auto*
then have $(Mod\ (real\text{-}of\text{-}int\ \lfloor t \rfloor + 1))\ r\ \omega = X\text{-}mod\ r\ \omega$
unfolding *X-mod-def* **apply** (*simp only*: *True if-True*)
apply (*intro Mod-eq-N-inf*[*OF True*])
apply *simp*
using ‹$t \in \{0..\}$› **by** *auto*
(*metis* (*no-types*, *opaque-lifting*) *floor-in-Nats Nats-1 Nats-add Nats-cases of-int-of-nat-eq of-nat-in-Nats*, *linarith*)+
moreover have $(Mod\ (real\text{-}of\text{-}int\ \lfloor t \rfloor + 1))\ s\ \omega = X\text{-}mod\ s\ \omega$
unfolding *X-mod-def* **apply** (*simp only*: *True if-True*)
apply (*intro Mod-eq-N-inf*[*OF True*])
using ‹$s \in ball\ t\ \varepsilon \cap \{0..real\text{-}of\text{-}int\ \lfloor t \rfloor + 1\}$› **apply** *simp*
using ‹$t \in \{0..\}$› **apply** *safe*
apply (*metis* (*no-types*, *opaque-lifting*) *floor-in-Nats Nats-1 Nats-add Nats-cases of-int-of-nat-eq of-nat-in-Nats*)
apply *linarith*
apply (*smt* (*verit*) *Int-iff Nats-1 Nats-add Nats-altdef1 atLeast-iff mem-Collect-eq s zero-le-floor*)
apply (*metis Int-iff atLeast-iff floor-correct linorder-not-less one-add-floor s*)
done
ultimately have $dist\ (X\text{-}mod\ r\ \omega)\ (X\text{-}mod\ s\ \omega) \leq C * dist\ r\ s\ powr\ \gamma$
using $eC(3)$[*OF rs-ball*] **by** *simp*
}
then show $\exists \varepsilon{>}0.\ \exists\ C{\geq}0.\ \forall\ r{\in}ball\ t\ \varepsilon \cap \{0..\}.\ \forall\ s{\in}ball\ t\ \varepsilon \cap \{0..\}.\ dist\ (X\text{-}mod\ r\ \omega)\ (X\text{-}mod\ s\ \omega) \leq C * dist\ r\ s\ powr\ \gamma$
using $e'\ eC(2)$ **by** *blast*
qed
qed

lemma *local-holder-X-mod-process*: $local\text{-}holder\text{-}on\ \gamma\ \{0..\}\ (\lambda t.\ X\text{-}mod\text{-}process\ t\ \omega)$
for ω
unfolding *X-mod-process-def*
by (*smt* (*verit*, *best*) *Mod-measurable UNIV-I local-holder-X-mod local-holder-on-cong proc-source.process-process-of space-borel target-borel*)

theorem *continuous-modification*:
$\exists X'.\ modification\ X\ X' \wedge (\forall \omega.\ local\text{-}holder\text{-}on\ \gamma\ \{0..\}\ (\lambda t.\ X'\ t\ \omega))$

using *local-holder-X-mod-process modification-X-mod-process* **by** *blast*

end

theorem *Kolmogorov-Chentsov*:
fixes X :: (*real*, *'a*, *'b* :: *polish-space*) *stochastic-process*
and a b C γ :: *real*
assumes *index*[*simp*]: *proc-index* $X = \{0..\}$
and *target-borel*[*simp*]: *proc-target* $X = borel$
and *gt-0*: $a > 0$ $b > 0$ $C > 0$
and *b-leq-a*: $b \leq a$
and *gamma*: $\gamma \in \{0<..<b/a\}$
and *expectation*: $\bigwedge s\ t.\ [\![s \geq 0; t \geq 0]\!] \Longrightarrow$
$(\int^+ x.\ dist\ (X\ t\ x)\ (X\ s\ x)\ powr\ a\ \partial proc\text{-}source\ X) \leq C * dist\ t\ s\ powr$
$(1+b)$
shows $\exists X'.\ modification\ X\ X' \wedge (\forall \omega.\ local\text{-}holder\text{-}on\ \gamma\ \{0..\}\ (\lambda t.\ X'\ t\ \omega))$
using *Kolmogorov-Chentsov.continuous-modification Kolmogorov-Chentsov.intro*[*OF assms*]
by *blast*
end

References

[1] A. Klenke. *Probability theory: a comprehensive course*. Springer Science & Business Media, 2020.