Differential Privacy

Tetsuya Sato   Yasuhiko Minamide

March 17, 2025

Abstract

This entry contains formalization of differential privacy in a general setting, based on the seminal textbook of Dwork and Roth [3] and the papers of statistical divergence for differential privacy [2, 5].

This entry includes the following formalization:

- Measurable space of lists and measurability of list operations,
- Laplace distribution,
- The statisatical divergence for differential privacy,
- Differential privacy and its basic properties,
- Randomized Response,
- Laplace mechanism,
- The report noisy max mechanism.

Contents

theory *List-Space*
imports
HOL−Analysis.Finite-Product-Measure
Source-and-Sink-Algebras-Constructions
Measurable-Isomorphism
begin

1 Measurable space of finite lists

This entry introduces a measurable space of finite lists over a measurable space. The construction is inspired from Hirata's idea on list quasi-Borel spaces [Hirata et al., ITP 2023]. First, we construct countable coproduct space of product of n spaces. Then, we trasnsfer it to list space via bijections.

1.1 miscellaneous lemmas

lemma *measurable-pair-assoc*[*measurable*]:
shows $(\lambda((a, b), x). (a, b, x)) \in (A \bigotimes_M B) \bigotimes_M C \to_M A \bigotimes_M B \bigotimes_M C$
proof−
have $(\lambda((a,b),x). (a,b,x)) = (\lambda x. ((fst\ (fst\ x)),(snd\ (fst\ x)), snd\ x))$
by *auto*
also have ... $\in (A \bigotimes_M B) \bigotimes_M C \to_M A \bigotimes_M B \bigotimes_M C$

by *measurable*
finally show $(\lambda((a,\ b),\ x).\ (a,\ b,\ x)) \in (A \bigotimes_M B) \bigotimes_M C \to_M A \bigotimes_M B \bigotimes_M C$**.**
qed

lemma *measurable-pair-assoc'*[*measurable*]:
shows $(\lambda((a,\ b),\ x,\ l).\ (a,\ b,\ x,\ l)) \in (A \bigotimes_M B) \bigotimes_M C \bigotimes_M D \to_M A \bigotimes_M B \bigotimes_M C \bigotimes_M D$
proof−
have $(\lambda((a,b),x,l).\ (a,b,x,l)) = (\lambda x.\ ((fst\ (fst\ x)),(snd\ (fst\ x)),\ (fst\ (snd\ x)),\ (snd\ (snd\ x))))$
by *auto*
also have ... $\in (A \bigotimes_M B) \bigotimes_M C \bigotimes_M D \to_M A \bigotimes_M B \bigotimes_M C \bigotimes_M D$
by *measurable*
finally show $(\lambda((a,\ b),\ x,\ l).\ (a,\ b,\ x,\ l)) \in (A \bigotimes_M B) \bigotimes_M C \bigotimes_M D \to_M A \bigotimes_M B \bigotimes_M C \bigotimes_M D$**.**
qed

1.2 construction of list space

fun *pair-to-list* :: $(nat \times (nat \Rightarrow 'a)) \Rightarrow 'a\ list$ **where**
Zero: *pair-to-list* $(0,\text{-}) = []$
| *Suc*: *pair-to-list* $(Suc\ n,f) = (f\ 0)\ \#$ *pair-to-list* $(n,\ \lambda\ n.\ f\ (Suc\ n))$

fun *list-to-pair* :: $'a\ list \Rightarrow (nat \times (nat \Rightarrow 'a))$ **where**
Nil: *list-to-pair* $[] = (0,\lambda\text{-}.undefined)$
| *Cons*: *list-to-pair* $(x\#xs) = (Suc\ (fst($*list-to-pair* $xs)),\ \lambda n.\ if\ n = 0\ then\ x\ else\ (snd($*list-to-pair* $xs))(n - 1))$

definition *listM* :: $'a\ measure \Rightarrow 'a\ list\ measure$ **where**
listM M =
initial-source (*lists* (*space M*)) $\{($*list-to-pair*$,\ \Pi_M\ n{\in}UNIV.\ \Pi_S\ i{\in}\{..{<}n\}.\ M)\}$

lemma *space-listM*:
shows *space* (*listM M*) = (*lists* (*space M*))
by (*auto simp*: *listM-def*)

lemma *lists-listM*[*measurable*]:
shows *lists* (*space M*) $\in$ *sets* (*listM M*)
by (*metis sets.top space-listM*)

lemma *comp-list-to-list*:
shows *pair-to-list* (*list-to-pair xs*) = *xs*
by(*induction xs*, *auto*)

lemma *list-to-pair-function*:
shows *list-to-pair* $\in$ *lists* (*space M*) $\to$ *space* $(\Pi_M\ n{\in}UNIV.\ \Pi_S$

$i \in \{..<n\}$. *M*)
unfolding *space-prod-initial-source space-coprod-final-sink space-listM*
proof(*rule funcsetI*)
fix *xs*
show *xs* $\in$ *lists* (*space M*)$\Longrightarrow$ *list-to-pair xs* $\in$ (*SIGMA n*:*UNIV*. $\{..<n\} \rightarrow_E$ *space M*)
proof(*induction xs*)
case *Nil*
thus *list-to-pair* [] $\in$ (*SIGMA n*:*UNIV*. $\{..<n\} \rightarrow_E$ *space M*)
unfolding *PiE-def Sigma-def* **by** *auto*
next
case (*Cons x ys*)
hence *propxys*: *x* $\in$ (*space M*) *ys* $\in$ *lists* (*space M*)**using** *Cons-in-lists-iff* **by** *auto*
from *local.Cons* **obtain** *n f* **where** *propnf*: ((*list-to-pair ys*) = (*n*,*f*)) (*n*$\in$(*UNIV* :: *nat set*)) (*f* $\in$ $\{..<n\} \rightarrow_E$(*space M*))
unfolding *Sigma-def* **by** *auto*
have *list-to-pair* (*x* # *ys*) $\in$ ({*Suc n*} $\times$ ({..<(*Suc n*)} $\rightarrow_E$(*space M*)))
using *propxys propnf* **unfolding** *extensional-def PiE-def*
by(*casesn* = *0*,*auto*)
also have ... $\subseteq$ (*SIGMA n*:*UNIV*. $\{..<n\} \rightarrow_E$ *space M*)
unfolding *Sigma-def* **by**(*intro subsetI*, *blast*)
finally show *list-to-pair* (*x* # *ys*) $\in$ (*SIGMA n*:*UNIV*. $\{..<n\} \rightarrow_E$ *space M*)**.**
qed
qed

lemma *list-to-pair-measurable*:
shows *list-to-pair* $\in$ (*listM M*) $\rightarrow_M$ ($\amalg_M$ *n*$\in$*UNIV*. Π_S $i \in \{..<n\}$. *M*)
unfolding *listM-def* **using** *list-to-pair-function*
by (*metis insertI1 measurable-initial-source1*)

lemma *comp-pair-to-pair*:
shows (*n*,*f*) $\in$ (*SIGMA n*:*UNIV*. $\{..<n\} \rightarrow_E$(*space M*)) $\Longrightarrow$ (*list-to-pair* (*pair-to-list* (*n*,*f*)) = (*n*,*f*))
proof(*induct n arbitrary*: *f*)
case *0*
thus *?case* **by** *auto*
next
case (*Suc n*)
have (*Suc n*, *f*) $\in$ {(*n*,*f*) | *n f*. *n*$\in$ *UNIV* $\wedge$ *f* $\in$ $\{..<n\} \rightarrow_E$(*space M*)}
using *Suc.prems* **by** *blast*
hence (*n*, λ *n*. *f* (*Suc n*)) $\in$ (*SIGMA n*:*UNIV*. $\{..<n\} \rightarrow_E$ *space M*)
by (*auto simp add*: *Sigma-def*)
hence *in1*: *list-to-pair* (*pair-to-list* (*n*, λ *n*. *f* (*Suc n*))) =(*n*, λ *n*. *f*

(*Suc n*))
  by (*auto simp add*: *Suc.hyps*)
 thus *?case* **by** *auto*
qed

lemma *pair-to-list-function*:
 shows *pair-to-list* ∈ *space* (Π_M *n*∈*UNIV*. Π_S *i*∈{..<*n*}. *M*) → *lists* (*space M*)
 unfolding *space-prod-initial-source space-coprod-final-sink space-listM*
proof
 fix *x* :: (*nat* × -) **assume** *x* ∈ (*SIGMA n*:*UNIV*. {..<*n*} $\rightarrow_E$ *space M*)
 then obtain *n f* **where** *x*: *x* = (*n*,*f*) ∧ (*f* ∈ {..<*n*} $\rightarrow_E$ *space M*)
  unfolding *Sigma-def* **by** *auto*
 have ⋀*f*. (*f* ∈ {..<*n*} $\rightarrow_E$ *space M*) ⟹ *pair-to-list* (*n*,*f*) ∈ *lists* (*space M*)
 proof(*induction n*)
  case *0*
  thus *?case* **by** *auto*
 next
  case (*Suc n*)
  hence (*f 0*) ∈ *space M* **and** (λ *n*. *f* (*Suc n*))∈ {..<*n*} $\rightarrow_E$ *space M*
   and ⋀*f*. (*f* ∈ {..<*n*} $\rightarrow_E$ *space M*) ⟹ *pair-to-list* (*n*,*f*) ∈ *lists* (*space M*)
   by *auto*
  hence *pair-to-list* (*Suc n*, *f*) ∈ *lists* (*space M*)
   using *lists.Cons* **by** *auto*
  thus *?case* **by** *auto*
 qed
 thus*pair-to-list x* ∈ *lists* (*space M*)**using** *x* **by** *auto*
qed

proposition *pair-to-list-measurable*:
 shows *pair-to-list* ∈ (Π_M *n*∈*UNIV*. Π_S *i*∈{..<*n*}. *M*) $\rightarrow_M$ (*listM M*)
 unfolding *listM-def* **using** *pair-to-list-function*
proof(*rule measurable-initial-source2*)
 have *list-to-pair* ∈ *lists* (*space M*) → *space* (Π_M *n*∈*UNIV*. Π_S *i*∈{..<*n*}. *M*)
  by (*auto simp*: *list-to-pair-function*)
 thus∀ (*f*, *N*)∈{(*list-to-pair*, (Π_M *n*∈*UNIV*. Π_S *i*∈{..<*n*}. *M*))}. *f* ∈ *lists* (*space M*) → *space N*
  by *blast*
 have (λ*x*. *list-to-pair* (*pair-to-list x*)) ∈ (Π_M *n*∈*UNIV*. Π_S *i*∈{..<*n*}. *M*) $\rightarrow_M$ (Π_M *n*∈*UNIV*. Π_S *i*∈{..<*n*}. *M*)
 proof(*subst measurable-cong*[**where** *g* =*id*])
   show ⋀*w*. *w* ∈ *space* (Π_M *n*∈*UNIV*. Π_S *i*∈{..<*n*}. *M*) ⟹ *list-to-pair* (*pair-to-list w*) = *id w*

unfolding *space-prod-initial-source space-coprod-final-sink* **using** *comp-pair-to-pair* **by** *fastforce*
qed(*auto*)
thus$\forall (f, N) \in \{(\textit{list-to-pair}, (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M))\}.\ (\lambda x.\ f\ (\textit{pair-to-list}\ x)) \in (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M) \rightarrow_M N$
by *blast*
qed

proposition *measurable-iff-list-and-pair*:
shows *measurable-iff-pair-to-list*: $\bigwedge f.\ (f \in (\textit{listM}\ M) \rightarrow_M N) \longleftrightarrow (f\ o\ \textit{pair-to-list} \in (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M) \rightarrow_M N)$
and *measurable-iff-list-to-pair*: $\bigwedge f.\ (f\ o\ \textit{list-to-pair} \in (\textit{listM}\ M) \rightarrow_M N) \longleftrightarrow (f \in (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M) \rightarrow_M N)$
and *measurable-iff-pair-to-list2*: $\bigwedge f.\ (f \in N \rightarrow_M (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M)) \longleftrightarrow (\textit{pair-to-list} \circ f \in N \rightarrow_M \textit{listM}\ M \wedge f \in \textit{space}\ N \rightarrow \textit{space}\ (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M))$
and *measurable-iff-list-to-pair2*: $\bigwedge f.\ (f \in N \rightarrow_M \textit{listM}\ M) \longleftrightarrow (\textit{list-to-pair} \circ f \in N \rightarrow_M (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M) \wedge f \in \textit{space}\ N \rightarrow \textit{space}\ (\textit{listM}\ M))$
proof−
have *A*: $\forall x \in \textit{space}\ (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M).\ \textit{list-to-pair}\ (\textit{pair-to-list}\ x) = x$
and *B*: $\forall y \in \textit{space}\ (\textit{listM}\ M).\ \textit{pair-to-list}\ (\textit{list-to-pair}\ y) = y$
unfolding *space-coprod-final-sink space-prod-initial-source*
using *comp-list-to-list comp-pair-to-pair* **by** *auto*
show $\bigwedge f.\ (f \in (\textit{listM}\ M) \rightarrow_M N) \longleftrightarrow (f\ o\ \textit{pair-to-list} \in (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M) \rightarrow_M N)$
and $\bigwedge f.\ (f\ o\ \textit{list-to-pair} \in (\textit{listM}\ M) \rightarrow_M N) \longleftrightarrow (f \in (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M) \rightarrow_M N)$
and $\bigwedge f.\ (f \in N \rightarrow_M (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M)) \longleftrightarrow (\textit{pair-to-list} \circ f \in N \rightarrow_M \textit{listM}\ M \wedge f \in \textit{space}\ N \rightarrow \textit{space}\ (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M))$
and $\bigwedge f.\ (f \in N \rightarrow_M \textit{listM}\ M) \longleftrightarrow (\textit{list-to-pair} \circ f \in N \rightarrow_M (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M) \wedge f \in \textit{space}\ N \rightarrow \textit{space}\ (\textit{listM}\ M))$
using *measurable-isomorphism-iff* $[\textit{of pair-to-list}\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M\ (\textit{listM}\ M) \textit{list-to-pair - } N, \textit{OF pair-to-list-measurable list-to-pair-measurable}\ A\ B]$ **by** *blast+*
qed

proposition *measurable-iff-list-and-pair-plus*:
shows *measurable-iff-pair-to-list-plus*: $\bigwedge f.\ (f \in K \bigotimes_M (\textit{listM}\ M) \rightarrow_M N) \longleftrightarrow (f\ o\ (\lambda\ q.\ ((\textit{fst}\ q), \textit{pair-to-list}\ (\textit{snd}\ q))) \in K \bigotimes_M (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M) \rightarrow_M N)$
and *measurable-iff-list-to-pair-plus*: $\bigwedge f.\ (f \in K \bigotimes_M (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M) \rightarrow_M N) \longleftrightarrow (f\ o\ (\lambda\ p.\ ((\textit{fst}\ p), \textit{list-to-pair}\ (\textit{snd}\ p))) \in K \bigotimes_M (\textit{listM}\ M) \rightarrow_M N)$
proof−
have *1*: $(\lambda\ q.\ ((\textit{fst}\ q), \textit{pair-to-list}\ (\textit{snd}\ q))) \in K \bigotimes_M (\Pi_M\ n \in \textit{UNIV}.\ \Pi_S\ i \in \{..<n\}.\ M) \rightarrow_M K \bigotimes_M \textit{listM}\ M$

using *pair-to-list-measurable* **by** *measurable*
have *2*: $\bigwedge f.\ (\lambda\ p.\ ((fst\ p),\ list\text{-}to\text{-}pair\ (snd\ p))) \in K \bigotimes_M listM\ M \rightarrow_M K \bigotimes_M (\Pi_M\ n{\in}UNIV.\ \Pi_S\ i{\in}\{..{<}n\}.\ M)$
using *list-to-pair-measurable* **by** *measurable*
have *3*: $\forall\, x{\in}space\ (K \bigotimes_M (\Pi_M\ n{\in}UNIV.\ \Pi_S\ i{\in}\{..{<}n\}.\ M)).(fst\ (fst\ x,\ pair\text{-}to\text{-}list\ (snd\ x)),\ list\text{-}to\text{-}pair\ (snd\ (fst\ x,\ pair\text{-}to\text{-}list\ (snd\ x)))) = x$
by(*auto simp add*: *comp-pair-to-pair comp-def space-pair-measure space-prod-initial-source space-coprod-final-sink*)
have *4*: $\forall\, y{\in}space\ (K \bigotimes_M listM\ M).(fst\ (fst\ y,\ list\text{-}to\text{-}pair\ (snd\ y)),\ pair\text{-}to\text{-}list\ (snd\ (fst\ y,\ list\text{-}to\text{-}pair\ (snd\ y)))) = y$
by(*auto simp add*: *comp-list-to-list comp-def space-pair-measure space-prod-initial-source space-coprod-final-sink*)
show $\bigwedge f.\ (f \in K \bigotimes_M listM\ M \rightarrow_M N) \longleftrightarrow (f \circ (\lambda q.\ (fst\ q,\ pair\text{-}to\text{-}list\ (snd\ q))) \in K \bigotimes_M (\Pi_M\ n{\in}UNIV.\ \Pi_S\ i{\in}\{..{<}n\}.\ M) \rightarrow_M N)$
by(*rule measurable-isomorphism-iff* [*OF 1 2 3 4*])
show $\bigwedge f.\ (f \in K \bigotimes_M (\Pi_M\ n{\in}UNIV.\ \Pi_S\ i{\in}\{..{<}n\}.\ M) \rightarrow_M N) \longleftrightarrow (f\ o\ (\lambda\ p.\ ((fst\ p),\ list\text{-}to\text{-}pair\ (snd\ p))) \in K \bigotimes_M (listM\ M) \rightarrow_M N)$
by(*rule measurable-isomorphism-iff* [*OF 1 2 3 4*])
qed

lemma *measurable-iff-on-list*:
shows $\bigwedge f.\ (f \in (listM\ M) \rightarrow_M N) \longleftrightarrow (\forall\ n{\in}UNIV.\ (f\ o\ pair\text{-}to\text{-}list\ o\ coProj\ n) \in (\Pi_S\ i{\in}\{..{<}n\}.\ M) \rightarrow_M N)$
by(*auto simp add*: *measurable-iff-pair-to-list coprod-measurable-iff* [*THEN sym*])

1.3 measurability of functions defined inductively on lists

1.3.1 Measurability of (#)

definition *shift-prod* :: $'a \Rightarrow (nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a)$ **where**
$shift\text{-}prod\ x\ f = (\lambda\ m.\ if\ m = 0\ then\ x\ else\ f(m - 1))$

lemma *shift-prod-function*:
shows $(\lambda\ (x,xs).(shift\text{-}prod\ x\ xs)) \in space\ M \times (\Pi_E\ i{\in}\{..{<}n\}.\ space\ M) \rightarrow (\Pi_E\ i{\in}\{..{<}(Suc\ n)\}.\ space\ M)$
proof(*clarify*)
fix *x f* **assume** *x*: $x \in (space\ M)$ **and** *f*: $f \in \{..{<}n\} \rightarrow_E space\ M$
thus $shift\text{-}prod\ x\ f \in \{..{<}Suc\ n\} \rightarrow_E space\ M$
unfolding *shift-prod-def*
proof(*induction n*)
case *0*
thus *?case* **by**(*intro PiE-I*, *auto*)
next
case (*Suc n*)
assume *f*: $f \in \{..{<}Suc\ n\} \rightarrow_E space\ M$

show $(\lambda m.\ if\ m = 0\ then\ x\ else\ f\ (m - 1)) \in \{..<Suc\ (Suc\ n)\} \rightarrow_E space\ M$
proof(*intro PiE-I*)
fix *m*
show $m \in \{..<Suc\ (Suc\ n)\} \Longrightarrow (if\ m = 0\ then\ x\ else\ f\ (m - 1)) \in space\ M$
using *PiE-mem*[*OF f*,**where** $x = m - 1$] *x* **by** *auto*
show $m \notin \{..<Suc\ (Suc\ n)\} \Longrightarrow (if\ m = 0\ then\ x\ else\ f\ (m - 1)) = undefined$
using *PiE-arb*[*OF f*,**where** $x = m - 1$] *x* **by** *auto*
qed
qed
qed

lemma *shift-prod-measurable*:
shows $(\lambda\ (x,xs).(shift\text{-}prod\ x\ xs)) \in M \bigotimes_M (prod\text{-}initial\text{-}source\ \{..<n\}\ (\lambda\text{-}.M)) \rightarrow_M (prod\text{-}initial\text{-}source\ \{..<(Suc\ n)\}\ (\lambda\text{-}.M))$
unfolding *prod-initial-source-def*
proof(*rule measurable-initial-source2*)
show $(\lambda a.\ case\ a\ of\ (x, xs) \Rightarrow shift\text{-}prod\ x\ xs) \in space\ (M \bigotimes_M initial\text{-}source\ (\{..<n\} \rightarrow_E space\ M)\ \{(\lambda f.\ f\ i, M)\ |i.\ i \in \{..<n\}\}) \rightarrow \{..<Suc\ n\} \rightarrow_E space\ M$
unfolding *space-pair-measure* **by** (*metis shift-prod-function space-initial-source*)
show $\forall (f, Ma)\in\{(\lambda f.\ f\ i, M)\ |i.\ i \in \{..<Suc\ n\}\}.\ f \in (\{..<Suc\ n\} \rightarrow_E space\ M) \rightarrow space\ Ma$
by *blast*
show $\forall (f, Ma)\in\{(\lambda f.\ f\ i, M)\ |i.\ i \in \{..<Suc\ n\}\}.$
$(\lambda x.\ f\ (case\ x\ of\ (x, xs) \Rightarrow shift\text{-}prod\ x\ xs)) \in M \bigotimes_M initial\text{-}source\ (\{..<n\} \rightarrow_E space\ M)\ \{(\lambda f.\ f\ i, M)\ |i.\ i \in \{..<n\}\} \rightarrow_M Ma$
proof(*intro ballI, elim exE conjE CollectE*)
fix $x :: ((nat \Rightarrow 'a) \Rightarrow 'a) \times 'a\ measure$ **and** *i* **assume** *x*: $x = (\lambda f.\ f\ i, M)$ **and** *i*: $i \in \{..<Suc\ n\}$
show $case\ x\ of\ (f, N) \Rightarrow (\lambda x.\ f\ (case\ x\ of\ (x, xs) \Rightarrow shift\text{-}prod\ x\ xs)) \in M \bigotimes_M initial\text{-}source\ (\{..<n\} \rightarrow_E space\ M)\ \{(\lambda f.\ f\ i, M)\ |i.\ i \in \{..<n\}\} \rightarrow_M N$
proof(*casesi* $= 0$)
case *True*
hence $*$: $case\ x\ of\ (f, N) \Rightarrow (\lambda x.\ f\ (case\ x\ of\ (x, xs) \Rightarrow shift\text{-}prod\ x\ xs)) = fst$
by (*auto simp*: *case-prod-unfold shift-prod-def x i*)
thus *?thesis*
by (*auto simp*: *x*)
next
case *False*
hence $*$: $(case\ x\ of\ (f, N) \Rightarrow (\lambda x.\ f\ (case\ x\ of\ (x, xs) \Rightarrow shift\text{-}prod\ x\ xs))) = (\lambda\ xa.\ xa(i - 1))\ o\ snd$
by (*auto simp*: *case-prod-unfold comp-def shift-prod-def x i*)
have $(\lambda\ xa.\ xa(i - 1)) \in initial\text{-}source\ (\{..<n\} \rightarrow_E space\ M)\ \{(\lambda f.\ f\ i, M)\ |i.\ i \in \{..<n\}\} \rightarrow_M M$

proof(*intro measurable-initial-source1*)
show $**$: $((\lambda\ xa.\ xa(i - 1)),M) \in \{(\lambda f.\ f\ i,\ M)\ |i.\ i \in \{..<n\}\}$
using *False i* **by** *auto*
show $(\lambda xa.\ xa\ (i - 1)) \in (\{..<n\} \rightarrow_E\ space\ M) \rightarrow space\ M$
using $i\ **$ **by** *fastforce*
qed
thus *?thesis*
using $*\ x$ **by** *force*
qed
qed
qed

lemma *shift-prod-measurable′*:
assumes $x \in (space\ M)$
shows $(shift\text{-}prod\ x) \in (prod\text{-}initial\text{-}source\ \{..<n\}\ (\lambda\text{-}.M)) \rightarrow_M (prod\text{-}initial\text{-}source\ \{..<(Suc\ n)\}\ (\lambda\text{-}.M))$
using *measurable-Pair2*[*OF shift-prod-measurable assms*] **by** *auto*

definition *shift-list* :: $'a\ measure \Rightarrow 'a \Rightarrow nat \times (nat \Rightarrow 'a) \Rightarrow nat \times (nat \Rightarrow 'a)$ **where**
shift-list M x =
coTuple (*UNIV* :: *nat set*)
$(\lambda\ n :: nat.\ (space\ (prod\text{-}initial\text{-}source\ \{..<n\}\ (\lambda\text{-}.M))))$
$(\lambda\ n :: nat.\ coProj\ (Suc\ n)\ o\ (shift\text{-}prod\ x))$

lemma *shift-list-def2*:
shows $shift\text{-}list\ M\ x\ (n,\ f) = (Suc\ n,\ shift\text{-}prod\ x\ f)$
unfolding *shift-list-def shift-prod-def coTuple-def coProj-def* **by** *auto*

lemma *shift-list-measurable*:
shows $(\lambda\ (x,(n,f)).\ (shift\text{-}list\ M\ x\ (n,f))) \in (M \bigotimes_M (\Pi_M\ n{\in}UNIV.\ \Pi_S\ i{\in}\{..<n\}.\ M)) \rightarrow_M (\Pi_M\ n{\in}UNIV.\ \Pi_S\ i{\in}\{..<n\}.\ M)$
proof(*subst measurable-iff-dist-law-A*)
show *countable* (*UNIV* :: *nat set*)
by *auto*
show $(\lambda(x,\ n,\ f).\ shift\text{-}list\ M\ x\ (n,\ f)) \circ dist\text{-}law\text{-}A\ UNIV\ (\lambda n.\ \Pi_S\ \text{-}{\in}\{..<n\}.\ M)\ M \in (\Pi_M\ i{\in}UNIV.\ M \bigotimes_M (\Pi_S\ \text{-}{\in}\{..<i\}.\ M)) \rightarrow_M (\Pi_M\ n{\in}UNIV.\ \Pi_S\ \text{-}{\in}\{..<n\}.\ M)$
proof(*subst coprod-measurable-iff*,*intro ballI*)
fix *i* :: *nat* **assume** $i{\in}UNIV$
thus $(\lambda(x,\ n,\ f).\ shift\text{-}list\ M\ x\ (n,\ f)) \circ dist\text{-}law\text{-}A\ UNIV\ (\lambda n.\ \Pi_S\ \text{-}{\in}\{..<n\}.\ M)\ M \circ coProj\ i \in M \bigotimes_M (\Pi_S\ \text{-}{\in}\{..<i\}.\ M) \rightarrow_M (\Pi_M\ n{\in}UNIV.\ \Pi_S\ \text{-}{\in}\{..<n\}.\ M)$
using *shift-prod-measurable*[*of M i*] *coProj-measurable*[*ofSuc iUNIV*$(\lambda\ i.\ (\Pi_S\ \text{-}{\in}\{..<i\}.\ M))$]
by(*subst measurable-cong*, *unfold coTuple-def shift-list-def dist-law-A-def shift-prod-def coProj-def*, *blast*,*auto*)
qed
qed

lemma *shift-list-measurable'*:
assumes $x \in (space\ M)$
shows $(shift\text{-}list\ M\ x) \in (\Pi_M\ n{\in}UNIV.\ \Pi_S\ i{\in}\{..{<}n\}.\ M) \to_M (\Pi_M\ n{\in}UNIV.\ \Pi_S\ i{\in}\{..{<}n\}.\ M)$
using *measurable-Pair2*[*OF shift-list-measurable*[*of M*] *assms*] **by** *auto*

lemma *measurable-Cons*[*measurable*]:
shows $(\lambda\ (x,xs).\ x\ \#\ xs) \in M \bigotimes_M (listM\ M) \to_M (listM\ M)$
proof−
{
fix x **fix** xs :: - *list*
have $x\ \#\ xs = (pair\text{-}to\text{-}list\ o\ (\lambda\ (x,(n,f)).\ (shift\text{-}list\ M\ x\ (n,f))))\ (x,\ list\text{-}to\text{-}pair\ xs)$
unfolding *shift-list-def coTuple-def shift-prod-def*
by(*induction xs* ,*auto simp add*: *comp-list-to-list coProj-def*)
}
hence $*$: $(\lambda\ (x,xs).\ x\ \#\ xs) = (pair\text{-}to\text{-}list\ o\ (\lambda\ (x,(n,f)).\ (shift\text{-}list\ M\ x\ (n,f))))\ o\ (\lambda\ (x,xs :: \text{-}list).\ (x,\ list\text{-}to\text{-}pair\ xs))$
by (*auto simp*: *cond-case-prod-eta*)
show *?thesis*
using *list-to-pair-measurable pair-to-list-measurable shift-list-measurable*
by(*unfold* $*$, *measurable*)
qed

lemma *listM-Nil*[*measurable*]:
shows $Nil \in space\ (listM\ M)$
unfolding *listM-def space-initial-source* **by** *auto*

lemma *measurable-Cons'*[*measurable*]:
shows $x \in space\ M \Longrightarrow Cons\ x \in (listM\ M) \to_M (listM\ M)$
using *measurable-Pair2* **by** *auto*

1.3.2 Measurability of $\lambda p.\ [p]$, the unit of list monad.

lemma *measurable-unit-ListM*[*measurable*]:
shows $(\lambda p.\ [p]) \in M \to_M listM\ M$
by *measurable*

1.3.3 Measurability of *rec-list*

Since the notion of measurable functions does not support higher-order functions in general, the following theorems for measurability of*rec-list* are restricted.

lemma *measurable-rec-list-func2'*:
fixes $T :: 'a \Rightarrow ('b \Rightarrow 'd)$
and $F :: 'a \Rightarrow 'c \Rightarrow 'c\ list \Rightarrow ('b \Rightarrow 'd) \Rightarrow ('b \Rightarrow 'd)$
assumes $(\lambda(a,b).\ T\ a\ b) \in K \bigotimes_M L \to_M N$

and $\bigwedge i\ g.\ (\lambda\ ((a,\ q),\ l).\ g\ a\ q\ l) \in (K \bigotimes_M L) \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
$\Longrightarrow (\lambda((a,b),x,l).\ (F\ a\ x\ ((\textit{pair-to-list}\ o\ (\textit{coProj}\ i))\ l))\ (\lambda q.\ (g\ a\ q\ l))\ b) \in (K \bigotimes_M L) \bigotimes_M M \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$

shows $(\lambda((a,b),xs).\ (\textit{rec-list}\ (T\ a)\ (F\ a))\ xs\ b) \in (K \bigotimes_M L) \bigotimes_M (\textit{listM}\ M) \rightarrow_M N$
proof(*subst measurable-iff-pair-to-list-plus*,*subst measurable-iff-dist-laws*)
show *countable* (*UNIV* :: *nat set*)**by** *auto*
show $(\lambda((a,\ b),\ xs).\ \textit{rec-list}\ (T\ a)\ (F\ a)\ xs\ b) \circ (\lambda q.\ (\textit{fst}\ q,\ \textit{pair-to-list}\ (\textit{snd}\ q))) \circ \textit{dist-law-A}\ \textit{UNIV}\ (\lambda n.\ \Pi_S\ i{\in}\{..{<}n\}.\ M)\ (K \bigotimes_M L) \in (\amalg_M\ i{\in}\textit{UNIV}.\ (K \bigotimes_M L) \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M)) \rightarrow_M N$
unfolding *dist-law-A-def2*
proof(*subst coprod-measurable-iff*, *intro ballI*)
fix *i* :: *nat* **assume** *i*: $i \in \textit{UNIV}$
show $(\lambda((a,\ b),\ xs).\ \textit{rec-list}\ (T\ a)\ (F\ a)\ xs\ b) \circ (\lambda q.\ (\textit{fst}\ q,\ \textit{pair-to-list}\ (\textit{snd}\ q))) \circ (\lambda p.\ (\textit{fst}\ (\textit{snd}\ p),\ \textit{fst}\ p,\ \textit{snd}\ (\textit{snd}\ p))) \circ \textit{coProj}\ i \in (K \bigotimes_M L) \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
proof(*induction i*)
case *0*
have $((\lambda a.\ \textit{case}\ a\ \textit{of}\ (a,\ b) \Rightarrow (\textit{case}\ a\ \textit{of}\ (a,\ b) \Rightarrow \lambda xs.\ \textit{rec-list}\ (T\ a)\ (F\ a)\ xs\ b)\ b) \circ (\lambda q.\ (\textit{fst}\ q,\ \textit{pair-to-list}\ (\textit{snd}\ q))) \circ (\lambda p.\ (\textit{fst}\ (\textit{snd}\ p),\ \textit{fst}\ p,\ \textit{snd}\ (\textit{snd}\ p))) \circ \textit{coProj}\ 0) = ((\lambda\ (a,b).\ T\ a\ b)\ o\ \textit{fst})$
by (*auto simp*: *coProj-def*)
also have ... $\in (K \bigotimes_M L) \bigotimes_M (\Pi_S\ i{\in}\{..{<}0\}.\ M) \rightarrow_M N$
using *assms*(*1*) **by** *auto*
finally show *?case* **.**
next
case (*Suc i*)
have $((\lambda a.\ \textit{case}\ a\ \textit{of}\ (a,\ b) \Rightarrow (\textit{case}\ a\ \textit{of}\ (a,\ b) \Rightarrow \lambda xs.\ \textit{rec-list}\ (T\ a)\ (F\ a)\ xs\ b)\ b) \circ (\lambda q.\ (\textit{fst}\ q,\ \textit{pair-to-list}\ (\textit{snd}\ q))) \circ (\lambda p.\ (\textit{fst}\ (\textit{snd}\ p),\ \textit{fst}\ p,\ \textit{snd}\ (\textit{snd}\ p))) \circ \textit{coProj}\ (\textit{Suc}\ i)) = ((\lambda((a,b),x,l).\ (F\ a)\ x\ ((\textit{pair-to-list}\ o\ (\textit{coProj}\ i))\ l)\ (\textit{rec-list}\ (T\ a)\ (F\ a)\ ((\textit{pair-to-list}\ o\ (\textit{coProj}\ i))\ l))\ b)\ o\ (\lambda\ ((a,b),l).\ ((a,b),l\ 0,\lambda n.\ l(\textit{Suc}\ n))))$
using *List.list.rec*(*2*)
by (*auto simp*: *coProj-def*)
also have ... $\in (K \bigotimes_M L) \bigotimes_M (\Pi_S\ i{\in}\{..{<}\textit{Suc}\ i\}.\ M) \rightarrow_M N$
proof(*rule measurable-comp*)
show $(\lambda((a,\ b),\ l).\ ((a,\ b),\ l\ 0,\ \lambda n.\ l\ (\textit{Suc}\ n))) \in (K \bigotimes_M L) \bigotimes_M (\Pi_S\ i{\in}\{..{<}\textit{Suc}\ i\}.\ M) \rightarrow_M (K \bigotimes_M L) \bigotimes_M M \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M)$
proof(*intro measurable-pair*)
have *1*: $(\lambda\ l.\ l\ (0\ ::\ \textit{nat})) \in (\Pi_S\ i{\in}\{..{<}\textit{Suc}\ i\}.\ M) \rightarrow_M M$**using**
measurable-prod-initial-source-projections **by** *measurable*
have $\textit{fst} \circ (\textit{snd} \circ (\lambda((a,\ b),\ l).\ ((a,\ b),\ l\ 0,\ \lambda n.\ l\ (\textit{Suc}\ n)))) = (\lambda\ l.\ l\ 0)\ o\ \textit{snd}$**by** *auto*
also have ... $\in (K \bigotimes_M L) \bigotimes_M (\Pi_S\ i{\in}\{..{<}\textit{Suc}\ i\}.\ M) \rightarrow_M$

M

using *1* **by** *auto*

finally show *fst* $\circ$ (*snd* $\circ$ (λ((*a*, *b*), *l*). ((*a*, *b*), *l 0*, λ*n*. *l* (*Suc n*)))) $\in$ ($K \bigotimes_M L$) $\bigotimes_M$ (Π_S *i*$\in${..<*Suc i*}. M) $\to_M$ M**.**

have *snd* $\circ$ (*snd* $\circ$ (λ((*a*, *b*), *l*). ((*a*, *b*), *l 0*, λ*n*. *l* (*Suc n*)))) = (λ *l*. λ*n*. *l* (*Suc n*)) *o snd***by** *auto*

also have ... $\in$ ($K \bigotimes_M L$) $\bigotimes_M$ (Π_S *i*$\in${..<*Suc i*}. M) $\to_M$ (Π_S *i*$\in${..<*i*}. M)

using *measurable-projection-shift* **by** *measurable*

finally show *snd* $\circ$ (*snd* $\circ$ (λ((*a*, *b*), *l*). ((*a*, *b*), *l 0*, λ*n*. *l* (*Suc n*)))) $\in$ ($K \bigotimes_M L$) $\bigotimes_M$ (Π_S *i*$\in${..<*Suc i*}. M) $\to_M$ (Π_S *i*$\in${..<*i*}. M)**.**

have *fst* $\circ$ (*fst* $\circ$ (λ((*a*, *b*), *l*). ((*a*, *b*), *l 0*, λ*n*. *l* (*Suc n*)))) = *fst* $\circ$ *fst***by** *auto*

also have ... $\in$ ($K \bigotimes_M L$) $\bigotimes_M$ (Π_S *i*$\in${..<*Suc i*}. M) $\to_M$ K**by** *auto*

finally show *fst* $\circ$ (*fst* $\circ$ (λ((*a*, *b*), *l*). ((*a*, *b*), *l 0*, λ*n*. *l* (*Suc n*)))) $\in$ ($K \bigotimes_M L$) $\bigotimes_M$ (Π_S *i*$\in${..<*Suc i*}. M) $\to_M$ K**.**

have *snd* $\circ$ (*fst* $\circ$ (λ((*a*, *b*), *l*). ((*a*, *b*), *l 0*, λ*n*. *l* (*Suc n*)))) = *snd* $\circ$ *fst***by** *auto*

also have ... $\in$ ($K \bigotimes_M L$) $\bigotimes_M$ (Π_S *i*$\in${..<*Suc i*}. M) $\to_M$ L**by** *auto*

finally show *snd* $\circ$ (*fst* $\circ$ (λ((*a*, *b*), *l*). ((*a*, *b*), *l 0*, λ*n*. *l* (*Suc n*)))) $\in$ ($K \bigotimes_M L$) $\bigotimes_M$ (Π_S *i*$\in${..<*Suc i*}. M) $\to_M$ L**.**

qed

show (λ((*a*, *b*), *x*, *l*). *F a x* ((*pair-to-list* $\circ$ *coProj i*) *l*) (*rec-list* (*T a*) (*F a*) ((*pair-to-list* $\circ$ *coProj i*) *l*)) *b*)
$\in$ ($K \bigotimes_M L$) $\bigotimes_M$ $M \bigotimes_M$ (Π_S *i*$\in${..<*i*}. M) $\to_M$ N

proof(*intro assms*(*2*)[*of* λ *a b l*. (*rec-list* (*T a*) (*F a*) ((*pair-to-list* $\circ$ *coProj i*) *l*)) *bi*])

have (λ((*a*, *q*), *l*). *rec-list* (*T a*) (*F a*) ((*pair-to-list* $\circ$ *coProj i*) *l*) *q*) = (λ*a*. *case a of* (*a*, *b*) $\Rightarrow$ (*case a of* (*a*, *b*) $\Rightarrow$ λ*xs*. *rec-list* (*T a*) (*F a*) *xs b*) *b*) $\circ$ (λ*q*. (*fst q*, *pair-to-list* (*snd q*))) $\circ$
(λ*p*. (*fst* (*snd p*), *fst p*, *snd* (*snd p*))) $\circ$
coProj i

by (*auto simp*: *coProj-def*)

also have ... $\in$ ($K \bigotimes_M L$) $\bigotimes_M$ (Π_S *i*$\in${..<*i*}. M) $\to_M$ N**using** *Suc* **by** *measurable*

finally show (λ((*a*, *q*), *l*). *rec-list* (*T a*) (*F a*) ((*pair-to-list* $\circ$ *coProj i*) *l*) *q*) $\in$ ($K \bigotimes_M L$) $\bigotimes_M$ (Π_S *i*$\in${..<*i*}. M) $\to_M$ N**.**

qed

qed

finally show *?case* **.**

qed

qed

qed

lemma *measurable-rec-list-func2*:
fixes $T :: 'a \Rightarrow ('b \Rightarrow 'd)$
and $F :: 'a \Rightarrow 'c \Rightarrow 'c\ list \Rightarrow ('b \Rightarrow 'd) \Rightarrow ('b \Rightarrow 'd)$
assumes $(\lambda(a,b).\ T\ a\ b) \in K \bigotimes_M L \rightarrow_M N$
and $\bigwedge i\ g.\ (\lambda\ (a,\ q,\ l).\ g\ a\ q\ l) \in K \bigotimes_M L \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
$\Longrightarrow (\lambda(a,b,x,l).\ (F\ a\ x\ ((pair\text{-}to\text{-}list\ o\ (coProj\ i))\ l))\ (\lambda q.\ (g\ a\ q\ l))\ b) \in K \bigotimes_M L \bigotimes_M M \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$

shows $(\lambda(a,b,xs).\ (rec\text{-}list\ (T\ a)\ (F\ a))\ xs\ b) \in K \bigotimes_M L \bigotimes_M (listM\ M) \rightarrow_M N$
proof−
{
fix $i :: nat$ **and** g
assume g: $(\lambda\ ((a,\ q),\ l).\ g\ a\ q\ l) \in (K \bigotimes_M L) \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
have $(\lambda\ (a,\ q,\ l).\ g\ a\ q\ l) = (\lambda\ ((a,\ q),\ l).\ g\ a\ q\ l)\ o\ (\lambda\ (a,\ q,\ l).\ ((a,\ q),\ l))$
by *auto*
also have ... $\in K \bigotimes_M L \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
using g **by** *measurable*
finally have *g2*: $(\lambda(a,\ q,\ l).\ g\ a\ q\ l) \in K \bigotimes_M L \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$.

hence m: $(\lambda(a,b,x,l).\ (F\ a\ x\ ((pair\text{-}to\text{-}list\ o\ (coProj\ i))\ l))\ (\lambda q.\ (g\ a\ q\ l))\ b) \in K \bigotimes_M L \bigotimes_M M \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
using *assms*(2) **by** *measurable*
have $(\lambda((a,b),x,l).\ (F\ a\ x\ ((pair\text{-}to\text{-}list\ o\ (coProj\ i))\ l))\ (\lambda q.\ (g\ a\ q\ l))\ b) = ((\lambda(a,b,x,l).\ (F\ a\ x\ ((pair\text{-}to\text{-}list\ o\ (coProj\ i))\ l))\ (\lambda q.\ (g\ a\ q\ l))\ b))\ o\ (\lambda((a,b),x,l).(a,b,x,l))$**by** *auto*
also have ... $\in (K \bigotimes_M L) \bigotimes_M M \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
using m **by** *measurable*
finally have F: $(\lambda((a,\ b),\ x,\ l).\ F\ a\ x\ ((pair\text{-}to\text{-}list \circ coProj\ i)\ l)\ (\lambda q.\ g\ a\ q\ l)\ b) \in (K \bigotimes_M L) \bigotimes_M M \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$.
}
note *2* = *this*
have *3*: $(\lambda((a,b),xs).\ (rec\text{-}list\ (T\ a)\ (F\ a))\ xs\ b) \in (K \bigotimes_M L) \bigotimes_M (listM\ M) \rightarrow_M N$
by(*rule measurable-rec-list-func2'*[*OF assms*(*1*) *2*],*auto*)
have $(\lambda(a,b,xs).\ (rec\text{-}list\ (T\ a)\ (F\ a))\ xs\ b) = (\lambda((a,b),xs).\ (rec\text{-}list\ (T\ a)\ (F\ a))\ xs\ b)\ o\ (\lambda(a,b,xs).((a,b),xs))$
by *auto*
also have ... $\in K \bigotimes_M L \bigotimes_M (listM\ M) \rightarrow_M N$**using** *3* **by** *measurable*
finally show $(\lambda(a,\ b,\ xs).\ rec\text{-}list\ (T\ a)\ (F\ a)\ xs\ b) \in K \bigotimes_M L \bigotimes_M listM\ M \rightarrow_M N$.

qed

lemma *measurable-rec-list$'$-func*:
fixes $T :: ('c \Rightarrow 'd)$
and $F :: 'b \Rightarrow 'b\ list \Rightarrow ('c \Rightarrow 'd) \Rightarrow ('c \Rightarrow 'd)$
assumes $T \in K \rightarrow_M N$
and $\bigwedge i\ g.\ g \in K \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
$\Longrightarrow (\lambda p.\ (F\ (fst\ (snd\ p))\ (pair\text{-}to\text{-}list\ (coProj\ i(snd\ (snd\ (p)))))\ (\lambda q.\ (g\ (q,snd\ (snd\ (p))))))\ (fst\ p)) \in K \bigotimes_M M \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$

shows $(\lambda p.\ (rec\text{-}list\ T\ F)\ (snd\ p)\ (fst\ p)) \in K \bigotimes_M (listM\ M) \rightarrow_M N$
proof(*subst measurable-iff-pair-to-list-plus*,*subst measurable-iff-dist-laws*)
show *countable* (*UNIV* :: *nat set*)**by** *auto*
show $(\lambda p.\ rec\text{-}list\ T\ F\ (snd\ p)\ (fst\ p)) \circ (\lambda q.\ (fst\ q,\ pair\text{-}to\text{-}list\ (snd\ q))) \circ dist\text{-}law\text{-}A\ UNIV\ (\lambda n.\ \Pi_S\ i{\in}\{..{<}n\}.\ M)\ K \in (\Pi_M\ i{\in}UNIV.\ K \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M)) \rightarrow_M N$
unfolding *dist-law-A-def2*
proof(*subst coprod-measurable-iff*, *intro ballI*)
fix i :: *nat* **assume** $i \in UNIV$
show $(\lambda p.\ rec\text{-}list\ T\ F\ (snd\ p)\ (fst\ p)) \circ (\lambda q.\ (fst\ q,\ pair\text{-}to\text{-}list\ (snd\ q))) \circ (\lambda p.\ (fst\ (snd\ p),\ fst\ p,\ snd\ (snd\ p))) \circ coProj\ i \in K \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
proof(*induction i*)
case *0*
{
fix p :: (- × (*nat*⇒-)) **assume** $p \in space\ (K \bigotimes_M (\Pi_S\ i{\in}\{..{<}0\}.\ M))$

hence $(snd\ p) = (\lambda\ n.\ undefined)$
unfolding *space-pair-measure space-prod-initial-source* **by** *auto*
hence $((\lambda p.\ rec\text{-}list\ T\ F\ (snd\ p)\ (fst\ p)) \circ (\lambda q.\ (fst\ q,\ pair\text{-}to\text{-}list\ (snd\ q))) \circ (\lambda p.\ (fst\ (snd\ p),\ fst\ p,\ snd\ (snd\ p))) \circ coProj\ 0)\ p = T\ (fst\ p)$
by (*auto simp*: *case-prod-beta comp-def coProj-def*)
}note $*$ = *this*
thus *?case*
by(*subst measurable-cong*[*OF* $*$],*auto intro*: *assms measurable-compose*)
next
case (*Suc i*)
have *eq1*: $(\lambda p.\ (rec\text{-}list\ T\ F\ (pair\text{-}to\text{-}list\ (i,(snd\ p))))\ (fst\ p)) = (\lambda p.\ rec\text{-}list\ T\ F\ (snd\ p)\ (fst\ p)) \circ (\lambda q.\ (fst\ q,\ pair\text{-}to\text{-}list\ (snd\ q))) \circ (\lambda p.\ (fst\ (snd\ p),\ fst\ p,\ snd\ (snd\ p))) \circ coProj\ i$
by (*auto simp*: *coProj-def comp-def*)
also have $... \in K \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$**using** *local.Suc* **by** *blast*

finally have *meas1*: $(\lambda p.\ (\mathit{rec\text{-}list}\ T\ F\ (\mathit{pair\text{-}to\text{-}list}\ (i,(\mathit{snd}\ p))))\ (\mathit{fst}\ p)) \in K \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$**.**

{

fix $p :: (\text{-} \times (\mathit{nat} \Rightarrow \text{-}))$ **assume** $p \in \mathit{space}\ (K \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M))$

hence $(\mathit{pair\text{-}to\text{-}list}\ (\mathit{Suc}\ i,\mathit{snd}\ p)) = ((\mathit{snd}\ p\ 0)\ \#\ (\mathit{pair\text{-}to\text{-}list}\ (i,\lambda n.\ (\mathit{snd}\ p)(\mathit{Suc}\ n))))$

by *auto*

hence $((\lambda p.\ \mathit{rec\text{-}list}\ T\ F\ (\mathit{snd}\ p)\ (\mathit{fst}\ p)) \circ (\lambda q.\ (\mathit{fst}\ q,\ \mathit{pair\text{-}to\text{-}list}\ (\mathit{snd}\ q))) \circ (\lambda p.\ (\mathit{fst}\ (\mathit{snd}\ p),\ \mathit{fst}\ p,\ \mathit{snd}\ (\mathit{snd}\ p))) \circ \mathit{coProj}\ (\mathit{Suc}\ i))\ p =$

$\mathit{rec\text{-}list}\ T\ F\ ((\mathit{snd}\ p\ 0)\ \#\ (\mathit{pair\text{-}to\text{-}list}\ (i,\lambda n.\ (\mathit{snd}\ p)(\mathit{Suc}\ n))))(\mathit{fst}\ p)$

by (*auto simp*: *coProj-def*)

also have ... $= F\ (\mathit{snd}\ p\ 0)\ (\mathit{pair\text{-}to\text{-}list}\ (i,\lambda n.\ (\mathit{snd}\ p)(\mathit{Suc}\ n)))\ (\mathit{rec\text{-}list}\ T\ F\ (\mathit{pair\text{-}to\text{-}list}\ (i,\lambda n.\ (\mathit{snd}\ p)(\mathit{Suc}\ n))))\ (\mathit{fst}\ p)$

by *auto*

finally have $((\lambda p.\ \mathit{rec\text{-}list}\ T\ F\ (\mathit{snd}\ p)\ (\mathit{fst}\ p)) \circ (\lambda q.\ (\mathit{fst}\ q,\ \mathit{pair\text{-}to\text{-}list}\ (\mathit{snd}\ q))) \circ (\lambda p.\ (\mathit{fst}\ (\mathit{snd}\ p),\ \mathit{fst}\ p,\ \mathit{snd}\ (\mathit{snd}\ p))) \circ \mathit{coProj}\ (\mathit{Suc}\ i))\ p = ((\lambda p.\ F\ (\mathit{fst}\ (\mathit{snd}\ p))\ (\mathit{pair\text{-}to\text{-}list}\ (\mathit{coProj}\ i\ (\mathit{snd}\ (\mathit{snd}\ p))))\ (\lambda q.\ \mathit{rec\text{-}list}\ T\ F\ (\mathit{pair\text{-}to\text{-}list}\ (i,\ \mathit{snd}\ (q,\ \mathit{snd}\ (\mathit{snd}\ p))))\ (\mathit{fst}\ (q,\ \mathit{snd}\ (\mathit{snd}\ p))))\ (\mathit{fst}\ p))\ o\ (\lambda\ p.\ (\mathit{fst}\ p,\mathit{snd}\ p\ 0,\ \lambda n.\ (\mathit{snd}\ p)(\mathit{Suc}\ n))))\ p$

by (*auto simp*: *coProj-def*)

}note $* = \mathit{this}$

show *?case*

proof(*subst measurable-cong*[*OF* $*$])

show $\bigwedge\ w.\ w \in \mathit{space}\ (K \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M)) \Longrightarrow w \in \mathit{space}\ (K \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M))$**by** *auto*

show $(\lambda p.\ F\ (\mathit{fst}\ (\mathit{snd}\ p))\ (\mathit{pair\text{-}to\text{-}list}\ (\mathit{coProj}\ i\ (\mathit{snd}\ (\mathit{snd}\ p))))\ (\lambda q.\ \mathit{rec\text{-}list}\ T\ F\ (\mathit{pair\text{-}to\text{-}list}\ (i,\ \mathit{snd}\ (q,\ \mathit{snd}\ (\mathit{snd}\ p))))\ (\mathit{fst}\ (q,\ \mathit{snd}\ (\mathit{snd}\ p))))\ (\mathit{fst}\ p)) \circ (\lambda p.\ (\mathit{fst}\ p,\ \mathit{snd}\ p\ 0,\ \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)))$
$\in K \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M N$

proof(*rule measurable-comp*)

show $(\lambda p.\ F\ (\mathit{fst}\ (\mathit{snd}\ p))\ (\mathit{pair\text{-}to\text{-}list}\ (\mathit{coProj}\ i\ (\mathit{snd}\ (\mathit{snd}\ p))))\ (\lambda q.\ \mathit{rec\text{-}list}\ T\ F\ (\mathit{pair\text{-}to\text{-}list}\ (i,\ \mathit{snd}\ (q,\ \mathit{snd}\ (\mathit{snd}\ p))))\ (\mathit{fst}\ (q,\ \mathit{snd}\ (\mathit{snd}\ p))))\ (\mathit{fst}\ p)) \in K \bigotimes_M M \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$

using *assms*(*2*)[*OF meas1*] **by** *auto*

show $(\lambda p.\ (\mathit{fst}\ p,\ \mathit{snd}\ p\ 0,\ \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n))) \in K \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M K \bigotimes_M M \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M)$

proof(*intro measurable-pair*)

have *eq1*: $\mathit{fst} \circ (\mathit{snd} \circ (\lambda p.\ (\mathit{fst}\ p,\ \mathit{snd}\ p\ 0,\ \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)))) = (\lambda p.\ p\ 0)\ o\ \mathit{snd}$**by** *auto*

have *eq2*: $\mathit{snd} \circ (\mathit{snd} \circ (\lambda p.\ (\mathit{fst}\ p,\ \mathit{snd}\ p\ 0,\ \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)))) = (\lambda p.\ \lambda n.\ p\ (\mathit{Suc}\ n))\ o\ \mathit{snd}$**by** *auto*

show $\mathit{fst} \circ (\lambda p.\ (\mathit{fst}\ p,\ \mathit{snd}\ p\ 0,\ \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n))) \in K \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M K$**by**(*auto simp*: *comp-def*)

show $\mathit{fst} \circ (\mathit{snd} \circ (\lambda p.\ (\mathit{fst}\ p,\ \mathit{snd}\ p\ 0,\ \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)))) \in K \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M M$

by(*subst eq1*, *auto intro*: *measurable-comp measur-*

able-prod-initial-source-projections)
show *snd* ∘ (*snd* ∘ (λ*p*. (*fst p*, *snd p 0*, λ*n*. *snd p* (*Suc n*)))) ∈ *K* $\bigotimes_M$ (Π_S *i*∈{..<*Suc i*}. *M*) $\rightarrow_M$ (Π_S *i*∈{..<*i*}. *M*)
proof(*subst eq2*,*rule measurable-comp*)
show *snd* ∈ *K* $\bigotimes_M$ (Π_S *i*∈{..<*Suc i*}. *M*) $\rightarrow_M$ (Π_S *i*∈{..<*Suc i*}. *M*)**by** *auto*
show (λ*p n*. *p* (*Suc n*)) ∈ (Π_S *i*∈{..<*Suc i*}. *M*) $\rightarrow_M$ (Π_S *i*∈{..<*i*}. *M*)
by(*rule measurable-prod-initial-sourceI*,*unfold space-prod-initial-source*, *auto intro* :*measurable-prod-initial-source-projections*)
qed
qed
qed
qed
qed
qed
qed

lemma *measurable-rec-list''-func*:
fixes *T* :: ('*c* ⇒ '*d*)
and *F* :: '*b* ⇒ '*b list* ⇒ ('*c* ⇒ '*d*) ⇒ ('*c* ⇒ '*d*)
assumes *T* ∈ *K* $\rightarrow_M$ *N*
and ⋀*i g*. (λ(*q*,*v*). *g q v*) ∈ *K* $\bigotimes_M$ (Π_S *i*∈{..<*i*}. *M*) $\rightarrow_M$ *N* ⟹ (λ(*x*,*y*,*xs*). (*F y* (*pair-to-list* (*coProj i xs*)) (λ*q*. (*g q xs*))) *x*) ∈ *K* $\bigotimes_M$ *M* $\bigotimes_M$ (Π_S *i*∈{..<*i*}. *M*) $\rightarrow_M$ *N*
shows (λ(*x*,*xs*). (*rec-list T F*) *xs x*) ∈ *K* $\bigotimes_M$ (*listM M*) $\rightarrow_M$ *N*
proof−
{
fix *i* :: *nat* **and** *g* **assume** *g* ∈ *K* $\bigotimes_M$ (Π_S *i*∈{..<*i*}. *M*) $\rightarrow_M$ *N*
hence (λ*x*. *F* (*fst* (*snd x*)) (*pair-to-list* (*coProj i* (*snd* (*snd x*)))) (λ*q*. *g* (*q*, *snd* (*snd x*))) (*fst x*)) ∈ *K* $\bigotimes_M$ *M* $\bigotimes_M$ (Π_S *i*∈{..<*i*}. *M*) $\rightarrow_M$ *N*
using *assms*(*2*)[*of*(λ*q xs*. *g* (*q*, *xs*))] **by** (*auto simp*: *case-prod-beta'*)
}
hence ⋀*i g*. *g* ∈ *K* $\bigotimes_M$ (Π_S *i*∈{..<*i*}. *M*) $\rightarrow_M$ *N* ⟹ (λ*x*. *F* (*fst* (*snd x*)) (*pair-to-list* (*coProj i* (*snd* (*snd x*)))) (λ*q*. *g* (*q*, *snd* (*snd x*))) (*fst x*)) ∈ *K* $\bigotimes_M$ *M* $\bigotimes_M$ (Π_S *i*∈{..<*i*}. *M*) $\rightarrow_M$ *N***by** *auto*
thus *?thesis* **using** *assms*(*1*) *measurable-rec-list'-func* **by** *measurable*
qed

lemma *measurable-rec-list'*:
assumes *F* ∈ (*N* $\bigotimes_M$ *M* $\bigotimes_M$ (*listM M*) $\rightarrow_M$ *N*)
shows (λ*p*. (*rec-list* (*fst p*) (λ *y xs x*. *F*(*x*,*y*,*xs*)) (*snd p*))) ∈ *N* $\bigotimes_M$ (*listM M*) $\rightarrow_M$ *N*
proof(*subst measurable-iff-list-and-pair-plus*, *subst measurable-iff-dist-laws measurable-iff-dist-laws*)
show *countable* (*UNIV* :: *nat set*)**by** *auto*
show (λ*p*. *rec-list* (*fst p*) (λ*y xs x*. *F* (*x*, *y*, *xs*)) (*snd p*)) ∘ (λ*q*. (*fst q*, *pair-to-list* (*snd q*))) ∘ *dist-law-A UNIV* (λ*n*. Π_S *i*∈{..<*n*}. *M*) *N*

$\in (\Pi_M\ i{\in}UNIV.\ N \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M)) \rightarrow_M N$
unfolding *dist-law-A-def2*
proof(*subst coprod-measurable-iff*, *intro ballI*)
fix *i* :: *nat* **assume** $i \in UNIV$
show (λp. *rec-list* (*fst p*) (λy *xs x*. *F* (*x*, *y*, *xs*)) (*snd p*)) $\circ$ (λq. (*fst q*, *pair-to-list* (*snd q*))) $\circ$ (λp. (*fst* (*snd p*), *fst p*, *snd* (*snd p*))) $\circ$ *coProj i* $\in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
proof(*induction i*)
case *0*
{
fix *p* :: (- $\times$ (*nat*$\Rightarrow$-)) **assume** $p \in$ *space* ($N \bigotimes_M (\Pi_S\ i{\in}\{..{<}0\}.\ M)$)
hence (*snd p*) = (λ *n*. *undefined*)
unfolding *space-pair-measure space-prod-initial-source* **by** *auto*
hence ((λp. *rec-list* (*fst p*) (λy *xs x*. *F* (*x*, *y*, *xs*)) (*snd p*)) $\circ$ (λq. (*fst q*, *pair-to-list* (*snd q*))) $\circ$ (λp. (*fst* (*snd p*), *fst p*, *snd* (*snd p*))) $\circ$ *coProj 0*) *p* = *fst p*
by (*auto simp*: *case-prod-beta comp-def coProj-def*)
}note $*$ = *this*
show *?case*
by(*subst measurable-cong*[*OF* $*$],*auto*)
next
case (*Suc i*)
define *X* **where** *X* = (λp. *rec-list* (*fst p*) (λy *xs x*. *F* (*x*, *y*, *xs*)) (*snd p*)) $\circ$ (λq. (*fst q*, *pair-to-list* (*snd q*))) $\circ$ (λp. (*fst* (*snd p*), *fst p*, *snd* (*snd p*))) $\circ$ *coProj i*
hence *X*[*measurable*]: $X \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$
by (*auto simp*: *local.Suc*)
{
fix *p* :: (- $\times$ (*nat*$\Rightarrow$-)) **assume** $p \in$ *space* ($N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ M)$)
hence
((λp. *rec-list* (*fst p*) (λy *xs x*. *F* (*x*, *y*, *xs*)) (*snd p*)) $\circ$ (λq. (*fst q*, *pair-to-list* (*snd q*))) $\circ$ (λp. (*fst* (*snd p*), *fst p*, *snd* (*snd p*))) $\circ$ *coProj* (*Suc i*)) *p*
= *rec-list* (*fst p*) (λy *xs x*. *F* (*x*, *y*, *xs*)) (*pair-to-list* (*Suc i*, *snd p*))
by (*auto simp*: *case-prod-beta comp-def coProj-def*)
also have ... =*F*((((λp. *rec-list* (*fst p*) (λy *xs x*. *F* (*x*, *y*, *xs*)) (*snd p*)) $\circ$ (λq. (*fst q*, *pair-to-list* (*snd q*))) $\circ$ (λp. (*fst* (*snd p*), *fst p*, *snd* (*snd p*))) $\circ$ *coProj i*)(*fst p* , λn. *snd p* (*Suc n*)), *snd p 0*, *pair-to-list* (*i*, λn. *snd p* (*Suc n*)))
by(*auto simp*: *case-prod-beta comp-def coProj-def*)
also have ... = (*F o* (λ *p*. (*X* (*fst p* , λn. *snd p* (*Suc n*)), *snd p 0*, *pair-to-list* (*i*, λn. *snd p* (*Suc n*))))) *p*
unfolding *X-def* **by** *auto*
finally have ((λp. *rec-list* (*fst p*) (λy *xs x*. *F* (*x*, *y*, *xs*)) (*snd p*)) $\circ$ (λq. (*fst q*, *pair-to-list* (*snd q*))) $\circ$ (λp. (*fst* (*snd p*), *fst p*, *snd* (*snd p*))) $\circ$ *coProj* (*Suc i*)) *p* = (*F o* (λ *p*. (*X* (*fst p* , λn. *snd p* (*Suc*

$n)), \mathit{snd}\ p\ 0, \mathit{pair\text{-}to\text{-}list}\ (i, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n))))) \ p$.
}**note** $* = \mathit{this}$

show $(\lambda p.\ \mathit{rec\text{-}list}\ (\mathit{fst}\ p)\ (\lambda y\ xs\ x.\ F\ (x, y, xs))\ (\mathit{snd}\ p)) \circ (\lambda q.\ (\mathit{fst}\ q, \mathit{pair\text{-}to\text{-}list}\ (\mathit{snd}\ q))) \circ (\lambda p.\ (\mathit{fst}\ (\mathit{snd}\ p), \mathit{fst}\ p, \mathit{snd}\ (\mathit{snd}\ p))) \circ \mathit{coProj}\ (\mathit{Suc}\ i) \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M N$
proof(*subst measurable-cong*[*OF* ∗])
show $\bigwedge w.\ w \in \mathit{space}\ (N \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M)) \Longrightarrow w \in \mathit{space}\ (N \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M))$**by** *auto*
show $F \circ (\lambda p.\ (X\ (\mathit{fst}\ p, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)), \mathit{snd}\ p\ 0, \mathit{pair\text{-}to\text{-}list}\ (i, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)))) \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M N$
proof(*intro measurable-comp*)
show $F \in (N \bigotimes_M M \bigotimes_M (\mathit{listM}\ M) \rightarrow_M N)$**by** (*auto simp*: *assms*(*1*))
show $(\lambda p.\ (X\ (\mathit{fst}\ p, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)), \mathit{snd}\ p\ 0, \mathit{pair\text{-}to\text{-}list}\ (i, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)))) \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M N \bigotimes_M M \bigotimes_M \mathit{listM}\ M$
proof(*intro measurable-pair*)
have $\mathit{fst} \circ (\mathit{snd} \circ (\lambda p.\ (X\ (\mathit{fst}\ p, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)), \mathit{snd}\ p\ 0, \mathit{pair\text{-}to\text{-}list}\ (i, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n))))) = (\lambda p.\ p\ 0)\ o\ \mathit{snd}$**by** *auto*
also have ... $\in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M M$
proof(*rule measurable-comp*)
show $\mathit{snd} \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M)$**by** *auto*
show $(\lambda p.\ p\ 0) \in (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M M$
unfolding *prod-initial-source-def* **by**(*rule measurable-initial-source1*,*auto*)
qed
finally show $\mathit{fst} \circ (\mathit{snd} \circ (\lambda p.\ (X\ (\mathit{fst}\ p, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)), \mathit{snd}\ p\ 0, \mathit{pair\text{-}to\text{-}list}\ (i, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n))))) \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M M$.

have $\mathit{snd} \circ (\mathit{snd} \circ (\lambda p.\ (X\ (\mathit{fst}\ p, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n)), \mathit{snd}\ p\ 0, \mathit{pair\text{-}to\text{-}list}\ (i, \lambda n.\ \mathit{snd}\ p\ (\mathit{Suc}\ n))))) = (\lambda p.\ \mathit{pair\text{-}to\text{-}list}\ (i, \lambda n.\ p\ (\mathit{Suc}\ n)))\ o\ \mathit{snd}$**by** *auto*
also have ... $\in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M \mathit{listM}\ M$
proof(*rule measurable-comp*)
show $\mathit{snd} \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M)$**by** *auto*
have $(\lambda p.\ \mathit{pair\text{-}to\text{-}list}\ (i, \lambda n.\ p\ (\mathit{Suc}\ n))) = \mathit{pair\text{-}to\text{-}list}\ o\ (\mathit{coProj}\ i)\ o\ (\lambda p.\ (\lambda n.\ p\ (\mathit{Suc}\ n)))$
by(*auto simp*: *comp-def coProj-def*)
also have ... $\in (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M) \rightarrow_M \mathit{listM}\ M$
proof(*subst measurable-iff-prod-initial-source*,*intro measurable-comp*)
show $(\lambda p\ n.\ \mathit{if}\ n = 0\ \mathit{then}\ \mathit{fst}\ p\ \mathit{else}\ \mathit{snd}\ p\ (n - 1)) \in M \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M (\Pi_S\ i{\in}\{..{<}\mathit{Suc}\ i\}.\ M)$
by(*auto simp*: *measurable-integrate-prod-initial-source-Suc-n*)
show $\mathit{pair\text{-}to\text{-}list} \in (\amalg_M\ n{\in}\mathit{UNIV}.\ \Pi_S\ i{\in}\{..{<}n\}.\ M)$

$\rightarrow_M$ (*listM M*)
by(*auto simp*: *pair-to-list-measurable*)
show *coProj i* $\in$(Π_S $i\in\{..<i\}$. M) $\rightarrow_M$ ($\amalg_M$ $n\in UNIV$. Π_S $i\in\{..<n\}$. M)
by *auto*
show (λp n. p (*Suc n*)) $\in$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ (Π_S $i\in\{..<i\}$. M)
by(*auto simp*: *measurable-projection-shift*)
qed
finally show (λp. *pair-to-list* (i, λn. p (*Suc n*))) $\in$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ *listM M*.
qed
finally show *snd* $\circ$ (*snd* $\circ$ (λp. (X (*fst p*, λn. *snd p* (*Suc n*)), *snd p 0*, *pair-to-list* (i, λn. *snd p* (*Suc n*)))))) $\in$ $N \bigotimes_M$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ *listM M*.
have *fst* $\circ$ (λp. (X (*fst p*, λn. *snd p* (*Suc n*)), *snd p 0*, *pair-to-list* (i, λn. *snd p* (*Suc n*)))) = X *o* (λp. (*fst p*, λn. *snd p* (*Suc n*)))**by** *auto*
also have ... $\in$ $N \bigotimes_M$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ N
proof(*rule measurable-comp*)
show $X \in N \bigotimes_M$ (Π_S $i\in\{..<i\}$. M) $\rightarrow_M$ N**by**(*auto simp*: X)
show (λp. (*fst p*, λn. *snd p* (*Suc n*))) $\in$ $N \bigotimes_M$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ $N \bigotimes_M$ (Π_S $i\in\{..<i\}$. M)
proof(*rule measurable-pair*)
show *fst* $\circ$ (λp. (*fst p*, λn. *snd p* (*Suc n*))) $\in$ $N \bigotimes_M$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ N
by (*auto simp*: *measurable-cong-simp*)
have *snd* $\circ$ (λp. (*fst p*, λn. *snd p* (*Suc n*))) = (λp n. p (*Suc n*)) *o snd***by** *auto*
also have ... $\in$ $N \bigotimes_M$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ (Π_S $i\in\{..<i\}$. M)
proof(*rule measurable-comp*)
show *snd* $\in$ $N \bigotimes_M$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ (Π_S $i\in\{..<Suc\ i\}$. M)**by** *auto*
show (λp n. p (*Suc n*)) $\in$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ (Π_S $i\in\{..<i\}$. M)
by(*auto simp*: *measurable-projection-shift*)
qed
finally show *snd* $\circ$ (λp. (*fst p*, λn. *snd p* (*Suc n*))) $\in$ $N \bigotimes_M$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ (Π_S $i\in\{..<i\}$. M).
qed
qed
finally show *fst* $\circ$ (λp. (X (*fst p*, λn. *snd p* (*Suc n*)), *snd p 0*, *pair-to-list* (i, λn. *snd p* (*Suc n*)))) $\in$ $N \bigotimes_M$ (Π_S $i\in\{..<Suc\ i\}$. M) $\rightarrow_M$ N.
qed
qed
qed

qed
qed
qed

lemma *measurable-rec-list'2*:
assumes $(\lambda p.\ (F\ (fst\ (snd\ p))\ (snd\ (snd\ p))\ (fst\ p))) \in (N \bigotimes_M M \bigotimes_M (listM\ M) \rightarrow_M N)$
shows $(\lambda p.\ (rec\text{-}list\ (fst\ p)\ F\ (snd\ p))) \in N \bigotimes_M (listM\ M) \rightarrow_M N$
using *measurable-rec-list'*[*OF assms*] **by** *auto*

lemma *measurable-rec-list*:
assumes $F \in (N \bigotimes_M M \bigotimes_M (listM\ M) \rightarrow_M N)$
and $T \in space\ N$
shows $rec\text{-}list\ T\ (\lambda\ y\ xs\ x.\ F(x,y,xs)) \in (listM\ M) \rightarrow_M N$
proof−
have $rec\text{-}list\ T\ (\lambda\ y\ xs\ x.\ F(x,y,xs)) = (\lambda p.\ (rec\text{-}list\ (fst\ p)\ (\lambda\ y\ xs\ x.\ F(x,y,xs))\ (snd\ p)))\ o\ (\lambda\ r.(T,\ r))$
by *auto*
also have ... $\in (listM\ M) \rightarrow_M N$
using *measurable-rec-list' assms* **by** *measurable*
finally show *?thesis* **.**
qed

lemma *measurable-rec-list2*:
assumes $(\lambda p.\ (F\ (fst\ (snd\ p))\ (snd\ (snd\ p))\ (fst\ p))) \in (N \bigotimes_M M \bigotimes_M (listM\ M) \rightarrow_M N)$
and $T \in space\ N$
shows $rec\text{-}list\ T\ F \in (listM\ M) \rightarrow_M N$
using *measurable-rec-list*[*OF assms*] **by** *auto*

lemma *measurable-rec-list'''*:
assumes $(\lambda(x,y,xs).\ F\ x\ y\ xs) \in N \bigotimes_M M \bigotimes_M (listM\ M) \rightarrow_M N$
and $T \in space\ N$
shows $rec\text{-}list\ T\ (\lambda\ y\ xs\ x.\ F\ x\ y\ xs) \in (listM\ M) \rightarrow_M N$
using *measurable-rec-list*[*OF assms*] **by** *fastforce*

1.3.4 Measurability of *case-list*

lemma *measurable-case-list*[*measurable*(*raw*)]:
assumes [*measurable*]:$(\lambda p.\ (F\ (fst\ p)\ (snd\ p))) \in M \bigotimes_M (listM\ M) \rightarrow_M N$
and $T \in space\ N$
shows $case\text{-}list\ T\ F \in (listM\ M) \rightarrow_M N$
proof(*subst measurable-iff-list-and-pair*, *subst coprod-measurable-iff*, *rule ballI*)
note [*measurable*] = *measurable-decompose-prod-initial-source-Suc-n*
fix $i :: nat$ **assume** i: $i \in UNIV$
show $case\text{-}list\ T\ F \circ pair\text{-}to\text{-}list \circ coProj\ i \in (\Pi_S\ i{\in}\{..{<}i\}.\ M) \rightarrow_M N$

unfolding *comp-def coProj-def*
proof(*induction i*)
case *0*
thus ($\lambda x.$ *case pair-to-list* (*0*, *x*) *of* [] $\Rightarrow$ *T* | *z* # *zs* $\Rightarrow$ *F z zs*) $\in$ (Π_S *i*$\in${..<*0*}. *M*) $\rightarrow_M$ *N*
using *assms*(*2*) **by** *auto*
next
case (*Suc i*)
{
fix *y* **assume** *y* $\in$ *space* (Π_S *i*$\in${..<*Suc i*}. *M*)
then obtain *z zs* **where** *pair*: *pair-to-list* (*Suc i*, *y*) = *z* # *zs* **and** *z* : *z* = *y 0* **and** *zs*: *zs* = *pair-to-list*(*i*, (λn :: *nat*. *y*(*Suc n*)))
by *auto*
hence ($\lambda x.$ *case pair-to-list* (*Suc i*, *x*) *of* [] $\Rightarrow$ *T* | *z* # *zs* $\Rightarrow$ *F z zs*) *y* = *F z zs*
by *auto*
also have ... = (($\lambda p.$ (*F* (*fst p*) (*snd p*))) *o* (λ *p*. ((*fst p*),(*pair-to-list o* (*coProj i*))(*snd p*))) *o* (λ *f*. (*f 0*, (λ *n*. *f* (*Suc n*))))) *y*
by (*auto simp*: *z zs coProj-def*)
finally have (*case pair-to-list* (*Suc i*, *y*) *of* [] $\Rightarrow$ *T* | *z* # *zs* $\Rightarrow$ *F z zs*) = (($\lambda p.$ *F* (*fst p*) (*snd p*)) $\circ$ ($\lambda p.$ (*fst p*, (*pair-to-list* $\circ$ *coProj i*) (*snd p*))) $\circ$ ($\lambda f.$ (*f 0*, $\lambda n.$ *f* (*Suc n*)))) *y*.
}note $*$ = *this*
show *?case*
unfolding *coProj-def*
proof(*subst measurable-cong*[*OF* $*$])
show $\bigwedge w.$ *w* $\in$ *space* (Π_S *i*$\in${..<*Suc i*}. *M*) $\Longrightarrow$ *w* $\in$ *space* (Π_S *i*$\in${..<*Suc i*}. *M*)
by *auto*
show ($\lambda p.$ *F* (*fst p*) (*snd p*)) $\circ$ ($\lambda p.$ (*fst p*, (*pair-to-list* $\circ$ *coProj i*) (*snd p*))) $\circ$ ($\lambda f.$ (*f 0*, $\lambda n.$ *f* (*Suc n*))) $\in$ (Π_S *i*$\in${..<*Suc i*}. *M*) $\rightarrow_M$ *N*
unfolding *comp-def* **using** *pair-to-list-measurable assms* **by** *measurable*
qed
qed
qed

lemma *measurable-case-list2*[*measurable*]:
assumes [*measurable*]:(λ(*x*,*b*,*bl*). *g x b bl*) $\in$ *N* $\bigotimes_M$ *L* $\bigotimes_M$ (*listM L*) $\rightarrow_M$ *M*
and [*measurable*]:*a* $\in$ *N* $\rightarrow_M$ *M*
shows (λ(*x*,*xs*). *case-list* (*a x*) (*g x*) *xs*) $\in$ *N* $\bigotimes_M$ (*listM L*) $\rightarrow_M$ *M*
proof(*subst measurable-iff-list-and-pair-plus*, *subst measurable-iff-dist-laws*)
note [*measurable*] = *pair-to-list-measurable list-to-pair-measurable dist-law-A-measurable measurable-projection-shift*
show *countable* (*UNIV* :: *nat set*)
by *auto*
show (λ(*x*, *y*). *case-list* (*a x*) (*g x*) *y*) $\circ$ ($\lambda q.$ (*fst q*, *pair-to-list* (*snd*

q))) $\circ$ *dist-law-A UNIV* ($\lambda n.\ \Pi_S\ i{\in}\{..{<}n\}.\ L$) $N \in (\Pi_M\ i{\in}UNIV.\ N \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ L)) \rightarrow_M M$
proof(*subst coprod-measurable-iff*,*rule ballI*)
fix i :: *nat* **assume** i: $i \in$ (*UNIV* :: *nat set*)
show ($\lambda(x,\ y)$. *case-list* ($a\ x$) ($g\ x$) y) $\circ$ (λq. (*fst q*, *pair-to-list* (*snd q*))) $\circ$ *dist-law-A UNIV* ($\lambda n.\ \Pi_S\ i{\in}\{..{<}n\}.\ L$) $N \circ$ *coProj* $i \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}i\}.\ L) \rightarrow_M M$
proof(*induction i*)
case *0*
{
fix w :: $'a \times (nat \Rightarrow\ 'b)$ **assume** $w \in$ *space* ($N \bigotimes_M (\Pi_S\ i{\in}\{..{<}0\}.\ L)$)
have ((λb. *case b of* (x, []) $\Rightarrow a\ x \mid (x,\ a\ \#\ b) \Rightarrow g\ x\ a\ b$) $\circ$ (λq. (*fst q*, *pair-to-list* (*snd q*))) $\circ$ *dist-law-A UNIV* ($\lambda n.\ \Pi_S\ i{\in}\{..{<}n\}.\ L$) $N \circ$ *coProj 0*) w = (a (*fst w*))
unfolding *comp-def coProj-def dist-law-A-def2 pair-to-list.simps* **by** *auto*
}note $*$ = *this*
show *?case*
using *assms* **by**(*subst measurable-cong*[*OF* $*$],*measurable*)
next
case (*Suc i*)
{
fix w :: $'a \times (nat \Rightarrow\ 'b)$ **assume** $w \in$ *space* ($N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L)$)
have ((λb. *case b of* (x, []) $\Rightarrow a\ x \mid (x,\ a\ \#\ b) \Rightarrow g\ x\ a\ b$) $\circ$ (λq. (*fst q*, *pair-to-list* (*snd q*))) $\circ$ *dist-law-A UNIV* ($\lambda n.\ \Pi_S\ i{\in}\{..{<}n\}.\ L$) $N \circ$ *coProj* (*Suc i*)) w = (($\lambda(x,b,bl)$. $g\ x\ b\ bl$) $\circ$ (λ w. (*fst w*, (((*snd w*) *0*) , (*pair-to-list* (i, λ n. (*snd w*)(*Suc n*))))))) w
unfolding *comp-def coProj-def dist-law-A-def2 pair-to-list.simps* **by** *auto*
}note $*$ = *this*
show *?case*
proof(*subst measurable-cong*[*OF* $*$])
show $\bigwedge w.\ w \in$ *space* ($N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L)$) $\Longrightarrow w \in$ *space* ($N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L)$)
by *auto*
show ($\lambda(x,\ xa,\ y)$. $g\ x\ xa\ y$) $\circ$ (λw. (*fst w*, *snd w 0*, *pair-to-list* (i, λn. *snd w* (*Suc n*)))) $\in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L) \rightarrow_M M$
proof(*rule measurable-comp*)
show (λw. (*fst w*, *snd w 0*, *pair-to-list* (i, λn. *snd w* (*Suc n*)))) $\in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L) \rightarrow_M N \bigotimes_M (L \bigotimes_M$ (*listM L*))
proof(*intro measurable-pair*)
show *fst* $\circ$ (λw. (*fst w*, *snd w 0*, *pair-to-list* (i, λn. *snd w* (*Suc n*)))) $\in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L) \rightarrow_M N$
by(*auto simp*: *comp-def*)
have (λx. *snd x 0*) = ((λf. *f 0*) *o snd*)
by *auto*
also have ... $\in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L) \rightarrow_M L$

by(*auto intro*: *measurable-comp measurable-prod-initial-source-projections*)
finally have $(\lambda x.\ snd\ x\ 0) \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L) \rightarrow_M L$.
thus$fst \circ (snd \circ (\lambda w.\ (fst\ w,\ snd\ w\ 0,\ pair\text{-}to\text{-}list\ (i,\ \lambda n.\ snd\ w\ (Suc\ n))))) \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L) \rightarrow_M L$
unfolding *comp-def prod-initial-source-def* **using** *pair-to-list-measurable*[*of* *L*] **by** *auto*
have $(\lambda x.\ pair\text{-}to\text{-}list\ (i,\ \lambda n.\ snd\ x\ (Suc\ n))) = pair\text{-}to\text{-}list\ o\ (coProj\ i)\ o\ (\lambda\ f.\ \lambda n.\ f\ (Suc\ n))\ o\ snd$
by(*auto simp*: *comp-def coProj-def*)
also have ... $\in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L) \rightarrow_M (listM\ L)$
proof(*intro measurable-comp*)
show $coProj\ i \in (\Pi_S\ i{\in}\{..{<}i\}.\ L) \rightarrow_M ((\Pi_M\ i{\in}UNIV.\ \Pi_S\ i{\in}\{..{<}i\}.\ L))$
by *auto*
qed(*auto*)
finally have $(\lambda x.\ pair\text{-}to\text{-}list\ (i,\ \lambda n.\ snd\ x\ (Suc\ n))) \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L) \rightarrow_M listM\ L$.
thus$snd \circ (snd \circ (\lambda w.\ (fst\ w,\ snd\ w\ 0,\ pair\text{-}to\text{-}list\ (i,\ \lambda n.\ snd\ w\ (Suc\ n))))) \in N \bigotimes_M (\Pi_S\ i{\in}\{..{<}Suc\ i\}.\ L) \rightarrow_M listM\ L$
unfolding *comp-def* **by** *auto*
qed
show $(\lambda(x,\ xa,\ y).\ g\ x\ xa\ y) \in N \bigotimes_M L \bigotimes_M listM\ L \rightarrow_M M$
by (*auto simp*: *assms*)
qed
qed
qed
qed
qed

lemma *measurable-case-list'*:
assumes $(\lambda(x,b,bl).\ g\ x\ b\ bl) \in N \bigotimes_M L \bigotimes_M (listM\ L) \rightarrow_M M$
and $a \in N \rightarrow_M M$
and $l \in N \rightarrow_M listM\ L$
shows $(\lambda x.\ case\text{-}list\ (a\ x)\ (g\ x)\ (l\ x)) \in N \rightarrow_M M$
using *measurable-case-list2 assms* **by** *measurable*

1.3.5 Measurability of *foldr*

lemma *measurable-foldr*:
assumes $(\lambda\ (x,y).\ F\ x\ y) \in (N \bigotimes_M M \rightarrow_M N)$
shows $(\lambda\ (T,xs).\ foldr\ (\lambda\ x\ y.\ F\ y\ x)\ xs\ T) \in N \bigotimes_M (listM\ M) \rightarrow_M N$
proof−
have *1*: $(\lambda\ (x\ ,y\ ,\ xs).\ (\lambda\ y\ xs\ x.\ F\ x\ y)\ y\ xs\ x) \in N \bigotimes_M M \bigotimes_M listM\ M \rightarrow_M N$
using *assms* **by** *measurable*
{
fix *T xs*

have (*foldr* (λ *x y*. *F y x*) *xs T*) = (*rec-list T* (λ *y xs x*. *F x y*) *xs*)
by(*induction xs*,*auto*)
}
hence (λ (*T*,*xs*). *foldr* (λ *x y*. *F y x*) *xs T*) = (λ (*T*,*xs*). (*rec-list T* (λ *y xs x*. (λ (*x* ,*y* , *xs*). (λ *y xs x*. *F x y*) *y xs x*)(*x*,*y*,*xs*)) *xs*))
by *auto*
also have ... $\in N \bigotimes_M$ (*listM M*) $\rightarrow_M N$
using *measurable-rec-list'*[*OF 1*] **by** *measurable*
finally show *?thesis* **.**
qed

1.3.6 Measurability of (@)

lemma *measurable-append*[*measurable*]:
shows (λ (*xs*,*ys*). *xs* @ *ys*) $\in$ (*listM M*) $\bigotimes_M$ (*listM M*) $\rightarrow_M$ (*listM M*)
proof−
{
fix *xs ys* :: *'a list*
have *xs* @ *ys* = (*rec-list ys* (λ*x xs ys*. *x* # *ys*)) *xs*
by(*induction xs*,*auto*)
}
hence (λ (*xs* :: *'a list*,*ys* :: *'a list*). *xs* @ *ys*) = (λ*p*. *rec-list* (*fst p*) (λ*x xs ys*. *x* # *ys*) (*snd p*)) *o* (λ(*xs*, *ys*). (*ys*, *xs*))
by *auto*
also have ... $\in$ (*listM M*) $\bigotimes_M$ (*listM M*) $\rightarrow_M$ (*listM M*)
using *measurable-rec-list'2* **by** *measurable*
finally show *?thesis* **.**
qed

1.3.7 Measurability of *rev*

lemma *measurable-rev*[*measurable*]:
shows *rev* $\in$ (*listM M*) $\rightarrow_M$ (*listM M*)
proof−
note [*measurable*] = *measurable-pair listM-Nil*
have *rev* = (λ *xs* :: *'a list*. *rec-list* [] (λ*y xs x*. (((λ(*xs*, *ys*). *xs* @ *ys*)*o* (λ*p*. (*fst p*,[*fst* (*snd p*)]))) (*x*, *y*, *xs*))) *xs*)
by (*auto simp add*: *rev-def*)
also have ... $\in$ *listM M* $\rightarrow_M$ *listM M*
by(*rule measurable-rec-list*,*measurable*)
finally show *?thesis* **.**
qed

1.3.8 Measurability of *concat*, that is, the bind of the list monad

lemma *measurable-concat*[*measurable*]:
shows *concat* $\in$ *listM* (*listM M*) $\rightarrow_M$ (*listM M*)
proof−

note [*measurable*] = *measurable-append listM-Nil*
have *concat* = (λ *xs* :: *'a list list*. *rec-list* [] (λ*y xs x*. (((λ(*xs*, *ys*). *xs* @ *ys*) *o* (λ*p*. (*fst* (*snd p*), *fst p*))) (*x*, *y*, *xs*))) *xs*)
by (*auto simp add*: *concat-def comp-def*)
also have ... ∈ *listM* (*listM M*) $\rightarrow_M$ *listM M*
by(*rule measurable-rec-list*,*measurable*)
finally show *?thesis* **.**
qed

1.3.9 Measurability of *map*, the functor part of the list monad

lemma *measurable-map*[*measurable*]:
assumes [*measurable*]: *f* ∈ *M* $\rightarrow_M$ *N*
shows (*map f*) ∈ (*listM M*) $\rightarrow_M$ (*listM N*)
proof−
note [*measurable*] = *measurable-pair listM-Nil*
{
fix *xs* :: *'a list*
have *map f xs* = (*rec-list* [] (λ*x xs ys*. (*f x*) # *ys*)) *xs*
by(*induction xs*,*auto*)
}
hence (*map f*) = (λ*p*. (*rec-list* [] (λ*y' xs' x'*. ((λ (*x*,*xs*). *x* # *xs*) *o* (λ*p*. (*f* (*fst* (*snd p*)), *fst p*))) (*x'*, *y'*, *xs'*)) *p*))
by *auto*
also have ... ∈ *listM M* $\rightarrow_M$ *listM N*
by(*rule measurable-rec-list*, *measurable*)
finally show *?thesis***.**
qed

1.3.10 Measurability of *length*

lemma *measurable-length*[*measurable*]:
shows *length* ∈ *listM M* $\rightarrow_M$ *count-space* (*UNIV* :: *nat set*)
proof −
{
fix *xs* :: *'a list*
have *length xs* = (*rec-list 0* (λ*y xs x*. (*Suc o fst*) (*x*, *y* ,*xs*))) *xs*
by(*induction xs*,*auto*)
}
hence *length* = (λ *xs* :: *'a list*.(*rec-list 0* (λ*y xs x* :: *nat*. (*Suc o fst*) (*x*, *y* ,*xs*)) *xs*))
by *auto*
also have ... ∈ *listM M* $\rightarrow_M$ *count-space UNIV*
by(*rule measurable-rec-list*, *auto*)
finally show *?thesis* **.**
qed

lemma *sets-listM-length*[*measurable*]:
shows {*xs*. *xs* ∈ *space* (*listM M*) ∧ *length xs* = *n*} ∈ *sets* (*listM M*)

by *measurable*

1.3.11 Measurability of *foldl*

lemma *measurable-foldl* [*measurable*]:
assumes [*measurable*]:$(\lambda(x,y).\ F\ y\ x) \in (M \bigotimes_M N \to_M N)$
shows $(\lambda\ (T,xs).\ foldl\ F\ T\ xs) \in N \bigotimes_M (listM\ M) \to_M N$
proof(*subst foldl-conv-foldr*)
have $(\lambda(T,\ xs).\ foldr\ (\lambda x\ y.\ F\ y\ x)\ (rev\ xs)\ T) = (\lambda(T,\ xs).\ foldr\ (\lambda x\ y.\ F\ y\ x)\ xs\ T)\ o\ (\lambda(T,\ xs).\ (T,\ rev\ xs))$
by *auto*
also have $... \in N \bigotimes_M listM\ M \to_M N$
using *measurable-foldr* **by** *measurable*
finally show $(\lambda(T,\ xs).\ foldr\ (\lambda x\ y.\ F\ y\ x)\ (rev\ xs)\ T) \in N \bigotimes_M listM\ M \to_M N$.
qed

1.3.12 Measurability of *fold*

lemma *measurable-fold* [*measurable*]:
assumes [*measurable*]: $(\lambda(x,y).\ F\ x\ y) \in (M \bigotimes_M N \to_M N)$
shows $(\lambda\ (T,xs).\ fold\ F\ xs\ T) \in N \bigotimes_M (listM\ M) \to_M N$
proof−
have $(\lambda\ (T,xs).\ fold\ F\ xs\ T) = (\lambda\ (T,xs).\ foldr\ F\ xs\ T)\ o\ (\lambda\ (T,xs).\ (T,\ rev\ xs))$
by (*auto simp add*: *foldr-conv-fold*)
also have $... \in N \bigotimes_M listM\ M \to_M N$
using *measurable-foldr* **by** *measurable*
finally show $(\lambda(T,\ xs).\ fold\ F\ xs\ T) \in N \bigotimes_M listM\ M \to_M N$.
qed

1.3.13 Measurability of *drop*

lemma *measurable-drop*[*measurable*]:
shows $drop\ n \in listM\ M \to_M listM\ M$
proof−
have $(\lambda a.\ []) \in space\ (Pi_M\ UNIV\ (\lambda a.\ listM\ M))$
using *listM-Nil space-PiM* **by** *fastforce*
with *measurable-rec-list'''* *listM-Nil space-PiM*
show *?thesis* **unfolding** *drop-def* **by** *measurable*
qed

lemma *measurable-drop'*[*measurable*]:
shows $(\lambda\ (n,xs).\ drop\ n\ xs) \in count\text{-}space\ (UNIV :: nat\ set) \bigotimes_M listM\ M \to_M listM\ M$
by *measurable*

1.3.14 Measurability of *take*

lemma *measurable-take*[*measurable*]:

shows *take n* $\in$ *listM M* $\rightarrow_M$ *listM M*
proof−
have (λ (*n*,*xs*). *take n xs*) = (λ (*n*,*xs*). *rev* (*drop* (*length xs* − *n*) (*rev xs*)))
by (*metis rev-swap rev-take*)
also have ... $\in$ *count-space* (*UNIV* :: *nat set*) $\bigotimes_M$ *listM M* $\rightarrow_M$ *listM M*
using *measurable-drop* **by** *measurable*
finally show *?thesis* **by** *measurable*
qed

lemma *measurable-take'*[*measurable*]:
shows (λ (*n*,*xs*). *take n xs*) $\in$ *count-space* (*UNIV* :: *nat set*) $\bigotimes_M$ *listM M* $\rightarrow_M$ *listM M*
by *measurable*

1.3.15 Measurability of (!) plus a default value

Since the @term undefined is harmful to prove measurability, we introduce the total function version of (!) that returns a fixed default value when the nth element is not found.

primrec *nth-total* :: *'a* $\Rightarrow$ *'a list* $\Rightarrow$ *nat* $\Rightarrow$ *'a* **where**
nth-total d [] *n* = *d* |
nth-total d (*x* # *xs*) *n* = (*case n of 0* $\Rightarrow$ *x* | *Suc k* $\Rightarrow$ *nth-total d xs k*)

value*nth-total* (*15* :: *nat*) [] (*0* :: *nat*) :: *nat*
value*nth-total* (*15* :: *nat*) [*1*] (*0* :: *nat*) :: *nat*
value*nth-total* (*15* :: *nat*) [*1*,*2*,*3*] (*0* :: *nat*) :: *nat*
value*nth-total* (*15* :: *nat*) [*1*,*2*,*3*] (*1* :: *nat*) :: *nat*
value*nth-total* (*15* :: *nat*) [*1*,*2*,*3*] (*2* :: *nat*) :: *nat*
value*nth-total* (*15* :: *nat*) [*1*,*2*,*3*] (*3* :: *nat*) :: *nat*

value*nth* [*1*] (*0* :: *nat*) :: *nat*
value*nth* [*1*,*2*,*3*] (*0* :: *nat*) :: *nat*
value*nth* [*1*,*2*,*3*] (*1* :: *nat*) :: *nat*
value*nth* [*1*,*2*,*3*] (*2* :: *nat*) :: *nat*

The total version *nth-total* is the same as (!) if the nth element exists and otherwise it returns the default value.

lemma *cong-nth-total-nth*:
shows ((*n* :: *nat*) < *length xs* $\wedge$ *0* < *length xs*) $\Longrightarrow$ *nth-total d xs n* = *nth xs n*
proof(*induction xs arbitrary* :*n*)
case *Nil*
thus *?case* **by** *auto*
next
case (*Cons a xs*)

show *nth-total d* (*a* # *xs*) *n* = (*a* # *xs*) ! *n*
proof(*casesn=0*)
case *True*
thus *?thesis* **by** *auto*
next
case *False*
hence *1*: *n* = *Suc* (*n*−*1*) **and** *3*: *n*−*1* < *length xs*
using *Cons*(*2*) **by** *auto*
hence *2*: *nth-total d xs* (*n*−*1*) = *xs* ! (*n*−*1*)
using *Cons*(*1*) **by** *fastforce*
thus *?thesis* **unfolding** *nth-total.simps*(*2*) *nth.simps*
by (*metis2Nitpick.case-nat-unfold*)
qed
qed

lemma *cong-nth-total-default*:
shows ¬((*n* :: *nat*) < *length xs* ∧ *0* < *length xs*) ⟹ *nth-total d xs n* = *d*
proof(*induction xs arbitrary* :*n*)
case *Nil*
thus *?case* **by** *auto*
next
case (*Cons a xs*)
hence *0* < *length* (*a* # *xs*) **by** *auto*
then obtain *m* **where** *n*: *n* = *Suc m*
using *Cons.prems not0-implies-Suc* **by** *force*
thus *?case* **unfolding** *n nth-total.simps* **using** *Cons.IH*[*of m*]
using *Cons.prems n* **by** *auto*
qed

lemma *nth-total-def2*:
shows *nth-total d xs n* = (*if* (*n* < *length xs* ∧ *0* < *length xs*) *then* *xs* ! *n else d*)
using *cong-nth-total-nth cong-nth-total-default* **by** (*metis* (*no-types*))

context
fixes *d*
begin

primrec *nth-total′* :: ′*a list* ⇒ *nat* ⇒ ′*a* **where**
nth-total′ [] *n* = *d* |
nth-total′ (*x* # *xs*) *n* = (*case n of 0* ⇒ *x* | *Suc k* ⇒ *nth-total′ xs k*)

lemma *measurable-nth-total′*:
assumes *d*: *d* ∈ *space M*
shows (λ (*n*,*xs*). *nth-total′ xs n*) ∈ (*count-space UNIV*) $\bigotimes_M$ *listM M* $\rightarrow_M$ *M*
unfolding *nth-total′-def* **using** *assms*
by(*intro measurable-rec-list′′-func*, *auto simp add*: *d*)

lemma *nth-total-nth-total′*:
  shows *nth-total d xs n* = *nth-total′ xs n*
proof(*induction xs arbitrary*: *n*)
  case *Nil*
  thus *?case* **by** *auto*
next
  case (*Cons a xs*)
  thus *?case* **unfolding** *nth-total′.simps nth-total.simps*
    by *meson*
qed

end

lemma *measurable-nth-total*[*measurable*]:
  assumes $d \in$ *space M*
  shows (λ (*n*,*xs*). *nth-total d xs n*) $\in$ (*count-space UNIV*) $\bigotimes_M$ *listM M* $\to_M$ *M*
  by (*auto simp*: *measurable-nth-total′*[*OF assms*] *nth-total-nth-total′*)

1.3.16 Definition and Measurability of list insert.

definition *list-insert* :: *′a* ⇒ *′a list* ⇒ *nat* ⇒ *′a list* **where**
  list-insert x xs i = (*take i xs*) @ [*x*] @ (*drop i xs*)

value*list-insert* (*15* :: *nat*) [] (*0* :: *nat*) :: *nat list*
value*list-insert* (*15* :: *nat*) [] (*1* :: *nat*) :: *nat list*
value*list-insert* (*15* :: *nat*) [*1*,*2*,*3*] (*0* :: *nat*) :: *nat list*
value*list-insert* (*15* :: *nat*) [*1*,*2*,*3*] (*1* :: *nat*) :: *nat list*
value*list-insert* (*15* :: *nat*) [*1*,*2*,*3*] (*2* :: *nat*) :: *nat list*
value*list-insert* (*15* :: *nat*) [*1*,*2*,*3*] (*3* :: *nat*) :: *nat list*
value*list-insert* (*15* :: *nat*) [*1*,*2*,*3*] (*4* :: *nat*) :: *nat list*

lemma *measurable-list-insert*[*measurable*]:
  fixes *n* :: *nat*
  shows (λ (*d*,*xs*). *list-insert d xs n*) $\in M \bigotimes_M$ *listM M* $\to_M$ *listM M*
  unfolding *list-insert-def* **by** *measurable*

primrec *list-insert′* :: *′a* ⇒ *′a list* ⇒ *nat* ⇒ *′a list* **where**
  list-insert′ d [] *n* = [*d*]
| *list-insert′ d* (*x* # *xs*) *n* = (*case n of 0* ⇒ *d* # (*x* # *xs*) | *Suc k* ⇒ *x* # (*list-insert′ d xs k*))

lemma *list-insert′-is-list-insert*:
  shows *list-insert′ x xs i* = *list-insert x xs i*
proof(*induction xs arbitrary*: *x i*)
  case *Nil*
  thus *?case* **unfolding** *list-insert-def* **by** *auto*
next

case (*Cons a xs*)
thus *?case*
proof(*casesi* = *0*)
case *True*
thus *?thesis* **unfolding** *list-insert-def list-insert'.simps*(*2*) *take.simps*(*2*) *drop.simps*(*2*) **by** *auto*
next
case *False*
then obtain *j* :: *nat* **where** *i*: *i* = *Suc j*
using *not0-implies-Suc* **by** *auto*
thus *?thesis* **unfolding** *i list-insert-def list-insert'.simps*(*2*) *take.simps*(*2*) *drop.simps*(*2*) **using** *Cons*
using *list-insert-def* **by** *fastforce*
qed
qed

definition *list-exclude* :: *nat*⇒ *'a list* ⇒ *'a list* **where**
list-exclude n xs = ((*take n xs*) @ (*drop* (*Suc n*) *xs*))

value*list-exclude* (*0* :: *nat*) [] :: *nat list*
value*list-exclude* (*0* :: *nat*) [*1*,*2*] :: *nat list*
value*list-exclude* (*1* :: *nat*) [*1*,*2*] :: *nat list*
value*list-exclude* (*2* :: *nat*) [*1*,*2*] :: *nat list*

lemma *measurable-list-exclude*[*measurable*]:
shows *list-exclude n* ∈ *listM M* $\rightarrow_M$ *listM M*
unfolding *list-exclude-def* **by** *measurable*

lemma *insert-exclude*:
shows *n* < *length xs* ⟹ *xs* ≠ [] ⟹ *list-insert'* (*nth-total d xs n*) (*list-exclude n xs*) *n* = *xs*
unfolding *list-exclude-def*
proof(*induction xs arbitrary* : *d n*)
case *Nil*
thus *?case* **by** *auto*
next
case (*Cons a xs*)
thus *n* < *length* (*a* # *xs*) ⟹ *a* # *xs* ≠ [] ⟹ *list-insert'* (*nth-total d* (*a* # *xs*) *n*) (*take n* (*a* # *xs*) @ *drop* (*Suc n*) (*a* # *xs*)) *n* = *a* # *xs*
proof(*casesxs* = [])
case *True*
then have *q*: *a* # *xs* = [*a*] **and** *n* = *0*
using *Cons* **by** *auto*
thus *?thesis*
by *auto*
next
case *False*
thus *n* < *length* (*a* # *xs*) ⟹ *list-insert'* (*nth-total d* (*a* # *xs*) *n*)

$(take\ n\ (a\ \#\ xs)\ @\ drop\ (Suc\ n)\ (a\ \#\ xs))\ n = a\ \#\ xs$
proof(*induction n*)
case *0*
then obtain *b ys* **where** *xs*: $xs = b\ \#\ ys$
using *list-to-pair.cases* **by** *auto*
thus *?case*
by *auto*
next
case (*Suc n*)
have $n < length\ xs$
using *Suc* **by** *auto*
thus *?case*
using *False Cons*(*1*) **by** *auto*
qed
qed
qed

1.3.17 Measurability of *list-update*

lemma *update-insert-exclude*:
shows $n < length\ xs \Longrightarrow xs \neq [] \Longrightarrow list\text{-}update\ xs\ n\ x = list\text{-}insert'\ x\ (list\text{-}exclude\ n\ xs)\ n$
proof(*induction xs arbitrary* : *x n*)
case *Nil*
thus *?case* **by** *auto*
next
case (*Cons a xs*)
thus $n < length\ (a\ \#\ xs) \Longrightarrow (a\ \#\ xs) \neq [] \Longrightarrow list\text{-}update\ (a\ \#\ xs)\ n\ x = list\text{-}insert'\ x\ (list\text{-}exclude\ n\ (a\ \#\ xs))\ n$
proof(*casesxs* = [])
case *True*
with *Cons* **have** *q*: $a\ \#\ xs = [a]$ **and** *q2*: $n = 0$**by** *auto*
show *?thesis* **unfolding** *q q2 list-exclude-def* **by** *auto*
next
case *False*
thus $n < length\ (a\ \#\ xs) \Longrightarrow (a\ \#\ xs) \neq [] \Longrightarrow list\text{-}update\ (a\ \#\ xs)\ n\ x = list\text{-}insert'\ x\ (list\text{-}exclude\ n\ (a\ \#\ xs))\ n$
proof(*induction n*)
case *0*
then obtain *b ys* **where** *xs*: $xs = b\ \#\ ys$
using *list-to-pair.cases* **by** *auto*
show *?case* **unfolding** *xs list-exclude-def* **by** *auto*
next
case (*Suc n*)
have *nn*: $n < length\ xs$**using** *Suc* **by** *auto*
thus *?case* **unfolding** *list-exclude-def list-insert'.simps*(*2*) *take.simps*(*2*) *drop.simps*(*2*) *list-update.simps*(*2*)
using *Cons*(*1*)
by (*metis Cons-eq-appendI False list-exclude-def list-insert'.simps*(*2*)

old.nat.simps(*5*))
  qed
 qed
qed

lemma *list-update-def2*:
 shows *list-update xs n x* = *If* (*n* < *length xs* ∧ *xs* ≠ []) (*list-insert′ x* (*list-exclude n xs*) *n*) *xs*
proof(*cases*(*n* < *length xs* ∧ *xs* ≠ []))
 case *True*
 thus *?thesis* **by**(*subst update-insert-exclude*,*auto*)
next
 case *False*
 thus *?thesis* **by** *auto*
qed

lemma *measurable-list-update*[*measurable*]:
 shows (λ (*x*,*xs*). *list-update xs* (*n* :: *nat*) *x*) ∈ $M \bigotimes_M$ (*listM M*) $\to_M$ (*listM M*)
proof−
 have *1* : (λ*x*. *length* (*snd x*)) ∈ ($M \bigotimes_M$ *listM M*) $\to_M$ (*count-space UNIV*)
  by *auto*
 have (λ*x*. *snd x* ≠ []) = (λ*xs*. (*length xs* ≠ *0*)) *o snd*
  by *auto*
 also have ... ∈ ($M \bigotimes_M$ *listM M*) $\to_M$ (*count-space* (*UNIV* :: *bool set*))
  by *measurable*
 finally have *Measurable.pred* ($M \bigotimes_M$ *listM M*) (λ*x*. *snd x* ≠ []).
 thus *?thesis*
  unfolding *list-insert′-is-list-insert list-update-def2*
  using *1* **by** *auto*
qed

1.3.18 Measurability of *zip*

lemma *measurable-zip*[*measurable*]:
 shows (λ(*xs*,*ys*). (*zip xs ys*)) ∈ (*listM M*) $\bigotimes_M$ (*listM N*) $\to_M$ (*listM* ($M \bigotimes_M N$))
 unfolding *zip-def*
 by(*rule measurable-rec-list″-func*,*auto*)

1.3.19 Measurability of *map2*

lemma *measurable-map2*[*measurable*]:
 assumes [*measurable*]: (λ(*x*,*y*). *f x y*) ∈ $M \bigotimes_M M' \to_M N$
 shows (λ(*xs*,*ys*). *map2 f xs ys*) ∈ (*listM M*) $\bigotimes_M$ (*listM M′*) $\to_M$ (*listM N*)
 by *auto*

end

theory *L1-norm-list*
imports *HOL−Analysis.Abstract-Metric-Spaces*
HOL−Library.Extended-Real
begin

2 L1-norm on Lists with fixed length

lemma *list-sum-dist-Cons*:
fixes *a b xs ys n* **assumes** *xn*: *length xs* = *n* **and** *yn*: *length ys* = *n*
shows $(\sum i = 1..Suc\ n.\ d\ ((a\ \#\ xs)\ !\ (i - 1))\ ((b\ \#\ ys)\ !\ (i - 1)))$
$= d\ a\ b + (\sum i = 1..n.\ d\ (xs\ !\ (i - 1))\ (ys\ !\ (i - 1)))$
proof−
have *0*: $\{\ 0..\ n\} = \{0\} \cup \{1..\ n\}$
by *auto*
show $(\sum i = 1..(Suc\ n).\ d\ ((a\ \#\ xs)\ !\ (i - 1))\ ((b\ \#\ ys)\ !\ (i - 1))) = d\ a\ b + (\sum i = 1..\ n.\ d\ (xs\ !\ (i - 1))\ (ys\ !\ (i - 1)))$
by(*subst sum.reindex-bij-witness*[*of - Suc* λ*i. i* − *1* {*0*..*n*}],*auto simp*: *0*)
qed

2.1 A locale for L1-norm of Lists

locale *L1-norm-list* = *Metric-space* +
fixes *n*::*nat*
begin

definition *space-L1-norm-list* ::(′*a list*) *set* **where**
space-L1-norm-list = {*xs. xs*∈ *lists M* ∧ *length xs* = *n*}

lemma *in-space-L1-norm-list-iff*:
shows *x* ∈ *space-L1-norm-list* ⟷(*x*∈ *lists M* ∧ *length x* = *n*)
unfolding *space-L1-norm-list-def* **by** *auto*

definition *dist-L1-norm-list* ::(′*a list*) ⇒ (′*a list*) ⇒ *real* **where**
dist-L1-norm-list xs ys = $(\sum\ i \in \{1..n\}.\ d\ (nth\ xs\ (i-1))\ (nth\ ys\ (i-1)))$

lemma *dist-L1-norm-list-nonneg*:
shows ⋀ *x y. 0* ≤ *dist-L1-norm-list x y*
unfolding *dist-L1-norm-list-def* **by**(*auto intro*!: *sum-nonneg*)

lemma *dist-L1-norm-list-commute*:
shows ⋀ *x y. dist-L1-norm-list x y* = *dist-L1-norm-list y x*
unfolding *dist-L1-norm-list-def* **using** *commute* **by** *presburger*

lemma *dist-L1-norm-list-zero*:

shows $\bigwedge$ *x y. length x = n* $\Longrightarrow$ *length y = n* $\Longrightarrow$ *x* $\in$ *lists M* $\Longrightarrow$ *y* $\in$ *lists M* $\Longrightarrow$ (*dist-L1-norm-list x y = 0* = (*x = y*))
proof−
fix *x y*::*'a list* **assume** *xm*: *length x = n* **and** *x*: *x* $\in$ *lists M* **and** *ym*: *length y = n* **and** *y*: *y* $\in$ *lists M*
show (*dist-L1-norm-list x y = 0*) = (*x = y*)
using *xm ym x y*
proof(*intro iffI*)
show *length x = n* $\Longrightarrow$ *length y = n* $\Longrightarrow$ *x* $\in$ *lists M* $\Longrightarrow$ *y* $\in$ *lists M* $\Longrightarrow$ *x = y* $\Longrightarrow$ *dist-L1-norm-list x y = 0*
unfolding *dist-L1-norm-list-def*
proof(*induction x arbitrary*: *n y*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a x2*)
then obtain *m* **where** *n*: *n = Suc m* **and** *y*: *y* = (*a # x2*) **and** *x2m*: *length x2 = m* **and** *x2M*: *x2* $\in$ *lists M* **and** *aM*: *a* $\in$ *M*
by *fastforce*
have ∗: ($\sum$ *i = 1..n. d* ((*a # x2*) ! (*i* − *1*)) (*y* ! (*i* − *1*))) = (*d a a*) + ($\sum$ *i = 1..m. d* (*x2* ! (*i*−*1*)) (*x2* ! (*i*−*1*)))
unfolding *n y* **using** *list-sum-dist-Cons*[*OF x2m x2m*] **by** *blast*
then show *?case*
using *Cons.IH*[*of m x2*, *OF x2m x2m x2M x2M*] *aM* **by** *auto*
qed
next
show *length x = n* $\Longrightarrow$ *length y = n* $\Longrightarrow$ *x* $\in$ *lists M* $\Longrightarrow$ *y* $\in$ *lists M* $\Longrightarrow$ *dist-L1-norm-list x y = 0* $\Longrightarrow$ *x = y*
unfolding *dist-L1-norm-list-def*
proof(*induction x arbitrary*: *n y*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a x2*)
then obtain *m b y2* **where** *n*: *n = Suc m* **and** *y*: *y* = (*b # y2*) **and** *x2m*: *length x2 = m* **and** *y2m*: *length y2 = m* **and** *aM*: *a* $\in$ *M* **and** *bM*: *b* $\in$ *M* **and** *x2M*: *x2* $\in$ *lists M* **and** *y2M*:*y2* $\in$*lists M*
by (*metis* (*no-types*, *opaque-lifting*) *Cons-in-lists-iff Suc-length-conv*)
have *0* ≤ ($\sum$ *i = 1..n. d* ((*a # x2*) ! (*i* − *1*)) (*y* ! (*i* − *1*))) **and** *1*: *0* ≤ ($\sum$ *i = 1..m. d* (*x2* ! (*i* − *1*)) (*y2* ! (*i* − *1*)))
by(*rule sum-nonneg*,*auto*)+
hence *0*: ($\sum$ *i = 1..n. d* ((*a # x2*) ! (*i* − *1*)) (*y* ! (*i* − *1*))) = *0*
using *Cons.prems*(*5*) **by** *blast*
have ∗: ($\sum$ *i = 1..n. d* ((*a # x2*) ! (*i* − *1*)) (*y* ! (*i* − *1*))) = (*d a b*) + ($\sum$ *i = 1..m. d* (*x2* ! (*i*−*1*)) (*y2* ! (*i*−*1*)))
unfolding *y n* **using** *list-sum-dist-Cons*[*of x2 m y2 d a b* , *OF x2m y2m*] **by** *blast*
have *2*: ($\sum$ *i = 1..m. d* (*x2* ! (*i* − *1*)) (*y2* ! (*i* − *1*))) ≤ *0*
using *nonneg*[*of a b*] *Cons.prems*(*5*) **unfolding** ∗ **by** *linarith*

with *1* **have** *3*: $(\sum i = 1..m.\ d\ (x2 \ !\ (i - 1))\ (y2 \ !\ (i - 1)))$
$= 0$
by *auto*
have $a = b$ **and** $y2 = x2$
unfolding *y* **using** *0 ∗ nonneg*[*of a b*] *zero*[*of a b*,*OF aM bM*]
Cons.IH[*of m y2*,*OF x2m y2m x2M y2M*] *3* **by** *auto*
thus *?case* **by**(*auto simp*: *y*)
qed
qed
qed

lemma *dist-L1-norm-list-trans*:
shows$\bigwedge$ *x y z*. *length x* = *n* $\Longrightarrow$ *length y* = *n* $\Longrightarrow$ *length z* = *n*$\Longrightarrow$
x ∈ *lists M* $\Longrightarrow$ *y* ∈ *lists M* $\Longrightarrow$ *z* ∈ *lists M* $\Longrightarrow$ (*dist-L1-norm-list x*
z ≤ *dist-L1-norm-list x y* + *dist-L1-norm-list y z*)
proof−
fix *x y z*::′*a list* **assume** *xm*: *length x* = *n* **and** *x*: *x* ∈ *lists M* **and**
ym: *length y* = *n* **and** *y*: *y* ∈ *lists M* **and** *zm*: *length z* = *n* **and** *z*: *z*
∈ *lists M*
show *dist-L1-norm-list x z* ≤ *dist-L1-norm-list x y* + *dist-L1-norm-list*
y z
using *xm ym zm x y z triangle* **unfolding** *dist-L1-norm-list-def*
proof(*induction n arbitrary*:*x y z*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then obtain *a b c*::′*a* **and** *xs ys zs*::′*a list* **where** *x*: *x* = *a*#*xs*
and *y*: *y* = *b*#*ys* **and** *z*:*z* = *c*#*zs*
and *xn*: *length xs* = *n* **and** *yn*: *length ys* = *n* **and** *zn*: *length zs*
= *n*
using *Suc-length-conv*[*of n x*] *Suc-length-conv*[*of n y*] *Suc-length-conv*[*of*
n z] **by** *auto*
hence *aM*: *a* ∈ *M* **and** *bM*: *b* ∈ *M* **and** *cM*: *c* ∈ *M* **and** *xsM*: *xs*
∈ *lists M* **and** *ysM*: *ys* ∈ *lists M* **and** *zsM*: *zs* ∈ *lists M*
using *listsE Suc.prems*(*4*,*5*,*6*) **unfolding** *x y z* **by** *auto*
have $(\sum i = 1..Suc\ n.\ d\ (x \ !\ (i - 1))\ (z \ !\ (i - 1))) = d\ a\ c\ +$
$(\sum i = 1..n.\ d\ (xs \ !\ (i - 1))\ (zs \ !\ (i - 1)))$
unfolding *x z* **using** *list-sum-dist-Cons*[*of xs n zs d a c*, *OF xn*
zn] **by** *blast*
also have ... $\leq (d\ a\ b + d\ b\ c) + (\ (\sum i = 1..\ n.\ d\ (xs \ !\ (i - 1))$
$(ys \ !\ (i - 1))) + (\sum i = 1..\ n.\ d\ (ys \ !\ (i - 1))\ (zs \ !\ (i - 1))))$
using *Suc.IH*[*of xs ys zs*, *OF xn yn zn xsM ysM zsM*] *Suc.prems*(*7*)
triangle[*of a b c*, *OF aM bM cM*] **by** *auto*
also have ... $\leq (d\ a\ b + (\sum i = 1..\ n.\ d\ (xs \ !\ (i - 1))\ (ys \ !\ (i -$
$1)))) + (\ d\ b\ c + (\sum i = 1..\ n.\ d\ (ys \ !\ (i - 1))\ (zs \ !\ (i - 1))))$
by *auto*
also have ... $= (\sum i = 1..Suc\ n.\ d\ (x \ !\ (i - 1))\ (y \ !\ (i - 1))) +$
$(\sum i = 1..Suc\ n.\ d\ (y \ !\ (i - 1))\ (z \ !\ (i - 1)))$

unfolding *x y z* **using** *list-sum-dist-Cons*[*of ys n zs d b c*, *OF yn zn*,*THEN sym*] *list-sum-dist-Cons*[*of xs n ys d a b*, *OF xn yn*,*THEN sym*] **by** *presburger*
finally show *?case*.
qed
qed

lemma *Metric-L1-norm-list* : *Metric-space space-L1-norm-list dist-L1-norm-list*
by (*simp add*: *Metric-space-def dist-L1-norm-list-commute dist-L1-norm-list-nonneg dist-L1-norm-list-trans dist-L1-norm-list-zero in-space-L1-norm-list-iff*)

end

sublocale *L1-norm-list* ⊆ *MetL1*: *Metric-space space-L1-norm-list dist-L1-norm-list*
by (*simp add*: *Metric-L1-norm-list*)

2.2 Lemmas on L1-norm on lists of natural numbers

lemma *L1-dist-k*:
fixes *x y k*::*nat*
assumes *real-of-int* $|int\ x - int\ y| \leq k$
shows $(x,y) \in \{(x,y) \mid x\ y.\ |int\ x - int\ y| \leq 1\}$ ^^ *k*
using *assms* **proof**(*induction k arbitrary*: *x y*)
case *0*
then have $x = y$
by *auto*
then show *?case*
by *auto*
next
case (*Suc k*)
then show *?case*
proof(*cases real-of-int* $|int\ x - int\ y| \leq k$)
case *True*
then have $(x,y) \in \{(x,y) \mid x\ y.\ |int\ x - int\ y| \leq 1\}$ ^^ *k*
using *Suc.IH*[*of x y*] **by** *auto*
thus *?thesis*
by *auto*
next
case *False*
hence $|int\ x - int\ y| > k$
by *auto*
moreover have $|int\ x - int\ y| \leq Suc\ k$
using *Suc.prems* **by** *linarith*
ultimately have *absSuck*: $|\ int\ x - int\ y| = Suc\ k$
unfolding *Int.zabs-def* **using** *int-ops*(*4*) *of-int-of-nat-eq of-nat-less-of-int-iff* **by** *linarith*
have $\exists\, z::nat.\ |\ int\ x - int\ z| = k \land |\ int\ z - int\ y| = 1$
proof(*cases* $x \leq y$)

case *True*
thus *?thesis*
using *absSuck* **by**(*intro exI*[*of* - *x* + *k*],*auto*)
next
case *False*
thus *?thesis*
using *absSuck* **by**(*intro exI*[*of* - *y* + *1*],*auto*)
qed
then obtain *z*::*nat* **where** *a*: | *int x* − *int z*| = *k* **and** *b*: | *int z* − *int y*| = *1*
by *blast*
then have (*x*,*z*) ∈ {(*x*,*y*) | *x y*. |*int x* − *int y*| ≤ *1*} ^^ *k* **and** (*z*,*y*) ∈ {(*x*,*y*) | *x y*. |*int x* − *int y*| ≤ *1*} ^^ *1*
using *Suc.IH*[*of x z*] **by** *auto*
thus *?thesis*
by *auto*
qed
qed

context
fixes *n*::*nat*
begin

interpretation *L11nat*: *L1-norm-list* (*UNIV*::*nat set*) (λ*x y*. |*int x* − *int y*|) *n*
by(*unfold-locales*,*auto*)
interpretation *L11nat2*: *L1-norm-list* (*UNIV*::*nat set*) (λ*x y*. |*int x* − *int y*|) *Suc n*
by(*unfold-locales*,*auto*)

abbreviation *adj-L11nat* ≡ {(*xs*, *ys*) |*xs ys*. *xs* ∈ *L11nat.space-L1-norm-list* ∧ *ys* ∈ *L11nat.space-L1-norm-list* ∧ *L11nat.dist-L1-norm-list xs ys* ≤ *1*}
abbreviation *adj-L11nat2* ≡ {(*xs*, *ys*) |*xs ys*. *xs* ∈ *L11nat2.space-L1-norm-list* ∧ *ys* ∈ *L11nat2.space-L1-norm-list* ∧ *L11nat2.dist-L1-norm-list xs ys* ≤ *1*}

lemma *L1-adj-iterate-Cons1*:
assumes *xs* ∈ *L11nat.space-L1-norm-list*
and *ys* ∈ *L11nat.space-L1-norm-list*
and (*xs*, *ys*) ∈ *adj-L11nat* ^^ *k*
shows (*x*#*xs*, *x*#*ys*) ∈ *adj-L11nat2* ^^ *k*
using *assms*
proof(*induction k arbitrary*: *xs ys*)
case *0*
then have *1*: *x* # *xs* = *x* # *ys* **and** *x* # *xs* ∈ *L11nat2.space-L1-norm-list*
by (*auto simp*: *L11nat.space-L1-norm-list-def L11nat2.space-L1-norm-list-def*)
then show *?case*
by *auto*

next
case (*Suc k*)
then obtain *zs* **where** *k*: (*xs* , *zs*) ∈ *adj-L11nat* ^^ *k* **and** *1*: (*zs* , *ys*) ∈ *adj-L11nat* **and** *zs*: *zs* ∈ *L11nat.space-L1-norm-list*
by *auto*
have *A*: (*x* # *xs* , *x* # *zs*) ∈ *adj-L11nat2* ^^ *k*
by(*rule Suc.IH* [*of xs zs*,*OF Suc*(*2*) *zs*, *OF k*])
have *x* # *zs* ∈ *L11nat2.space-L1-norm-list* **and** *x* # *ys* ∈ *L11nat2.space-L1-norm-list*

using *Suc*(*3*) *zs* **by**(*auto simp*: *L11nat2.space-L1-norm-list-def L11nat.space-L1-norm-list-def*)
moreover have *L11nat2.dist-L1-norm-list* (*x* # *zs*)(*x* # *ys*) ≤ *1*
using *1 L1-norm-list.list-sum-dist-Cons*[*of zs n ys* (λ*x y. real-of-int* |*int x* − *int y*|) *x x*] *Suc*(*3*) *zs L11nat.space-L1-norm-list-def L11nat2.dist-L1-norm-list-def L11nat.dist-L1-norm-list-def*
by *auto*
ultimately show *?case*
using *A* **by** *auto*
qed

lemma *L1-adj-Cons1*:
assumes *xs* ∈ *L11nat.space-L1-norm-list*
and *real-of-int* |*int x* − *int y*| ≤ *1*
shows (*x* # *xs*, *y* # *xs*) ∈ *adj-L11nat2*
using *L1-norm-list.list-sum-dist-Cons*[*of xs n xs* (λ*x y. real-of-int* |*int x* − *int y*|) *x y*] *assms*
unfolding *L11nat.dist-L1-norm-list-def L11nat2.space-L1-norm-list-def L11nat2.dist-L1-norm-list-def L11nat.space-L1-norm-list-def lists-UNIV*
by *auto*

lemma *L1-adj-iterate-Cons2*:
assumes *xs* ∈ *L11nat.space-L1-norm-list* **and** *real-of-int* |*int x* − *int y*| ≤ *k*
shows (*x* # *xs*, *y* # *xs*) ∈ *adj-L11nat2* ^^ *k*
proof−
have (*x*,*y*) ∈ {(*x*,*y*) | *x y*. |*int x* − *int y*| ≤ *1*} ^^ *k*
using *L1-dist-k*[*of x y k*] *assms*(*2*) **by** *auto*
thus *?thesis*
proof(*induction k arbitrary*: *x y*)
case *0*
then have *x* = *y*
by *auto*
then show *?case*
by *auto*
next
case (*Suc k*)
then obtain *z* **where** *k*: (*x*, *z*) ∈ {(*x*, *y*) |*x y*. |*int x* − *int y*| ≤ *1*} ^^ *k* **and** (*z*, *y*) ∈ {(*x*, *y*) |*x y*. |*int x* − *int y*| ≤ *1*}
using *L1-dist-k*[*of x y k*] **by** *auto*

```
    then have (x#xs, z#xs) ∈ adj-L11nat2 ^^ k and (z#xs, y#xs)
∈ adj-L11nat2
      using Suc.IH[of x z, OF k] L1-adj-Cons1[of xs z y, OF assms(1)]
      by auto
    then show ?case
      by auto
  qed
qed

lemma L1-adj-iterate:
  fixes k::nat
  assumes xs ∈ L11nat.space-L1-norm-list
    and ys ∈ L11nat.space-L1-norm-list
    and  L11nat.dist-L1-norm-list xs ys ≤ k
  shows (xs,ys) ∈ adj-L11nat ^^ k
  using assms
proof(induction n arbitrary: xs ys k)
  case 0
  interpret L1-norm-list UNIV::nat set (λx y. real-of-int |int x − int
y|) 0
    by(unfold-locales,auto)
  let ?R = {(xs, ys) |xs ys. xs ∈ space-L1-norm-list ∧ ys ∈ space-L1-norm-list
∧ dist-L1-norm-list xs ys ≤ 1}
  from 0 have 1: xs = [] and 2: ys = []
    by (auto simp: space-L1-norm-list-def)
  then show ?case
  proof(induction k)
    case 0
    then show ?case
      by auto
  next
    case (Suc k)
    then have 1: (xs, ys) ∈ ?R ^^ k
      by auto
    moreover from Suc have dist-L1-norm-list ys ys = 0
      by(subst MetL1.mdist-zero,auto simp: space-L1-norm-list-def)
    moreover hence 2: (ys, ys) ∈ ?R
      using space-L1-norm-list-def 2 unfolding lists-UNIV by auto
    ultimately show ?case
      by auto
  qed
next
  case (Suc n)
  interpret sp: L1-norm-list UNIV::nat set (λx y. real-of-int |int x −
int y|) n
    by(unfold-locales,auto)
  interpret sp2: L1-norm-list UNIV::nat set (λx y. real-of-int |int x
− int y|) Suc n
    by(unfold-locales,auto)
```

let *?R* = {(*xs*, *ys*) |*xs ys*. *xs* ∈ *sp.space-L1-norm-list* ∧ *ys* ∈ *sp.space-L1-norm-list* ∧ *sp.dist-L1-norm-list xs ys* ≤ *1*}
let *?R2* = {(*xs*, *ys*) |*xs ys*. *xs* ∈ *sp2.space-L1-norm-list* ∧ *ys* ∈ *sp2.space-L1-norm-list* ∧ *sp2.dist-L1-norm-list xs ys* ≤ *1*}
show *?case*
using *Suc* **proof**(*induction k*)
case *0*
from *0*(*2*,*3*,*4*) **have** *sp2.dist-L1-norm-list xs ys* = *0*
using *sp2.MetL1.mdist-pos-eq sp2.MetL1.mdist-zero* **by** *fastforce*
moreover have *sp2.dist-L1-norm-list xs ys* ≥ *0*
by(*rule sp2.MetL1.nonneg*)
ultimately have *sp2.dist-L1-norm-list xs ys* = *0*
by *auto*
hence *xs* = *ys*
using *sp2.MetL1.zero*[*of xs ys*] *0*(*2*,*3*) **by** *auto*
then show *?case* **by** *auto*
next
case (*Suc k*)
then obtain *x y xs2 ys2* **where** *xs*: *xs* = *x* # *xs2* **and** *ys*: *ys* = *y* # *ys2* **and** *xs2*: *xs2* ∈ *sp.space-L1-norm-list* **and** *ys2*: *ys2* ∈ *sp.space-L1-norm-list*
using *Suc.prems*(*1*) *Suc.prems*(*2*) *length-Suc-conv*[*of xs n*] *length-Suc-conv*[*of ys n*] *sp.space-L1-norm-list-def sp2.space-L1-norm-list-def* **by** *auto*
hence ∗: |*int x* − *int y*| + *sp.dist-L1-norm-list xs2 ys2* = *sp2.dist-L1-norm-list xs ys*
using *L1-norm-list.list-sum-dist-Cons*[*of xs2 n ys2* (λ*x y*. *real-of-int* |*int x* − *int y*|) *x y*] *xs2 ys2*
unfolding *sp.dist-L1-norm-list-def sp2.dist-L1-norm-list-def xs ys sp.space-L1-norm-list-def lists-UNIV* **by** *auto*
show *?case*
proof(*cases x* = *y*)
case *True*
hence *sp.dist-L1-norm-list xs2 ys2* ≤ (*Suc k*)
using *Suc*(*5*) ∗ **by** *auto*
with *Suc.prems*(*1*)[*of xs2 ys2 Suc k*] *xs2 ys2* **have** (*xs2*, *ys2*) ∈ *?R* ^^ *Suc k*
by *auto*
thus *?thesis*
unfolding *True xs ys*
using *L1-norm-list.L1-adj-iterate-Cons1*[*of xs2 n ys2 Suc k y*,*OF xs2 ys2*] **by** *blast*
next
case *False*
then obtain *k2*::*nat* **where** *k2*: |*int x* − *int y*| = *k2*
by (*meson abs-ge-zero nonneg-int-cases*)
moreover then have *en*: *sp.dist-L1-norm-list xs2 ys2* ≤ *Suc k* − *k2*
using ∗ *Suc.prems*(*4*) **by** *auto*
moreover have *0* ≤ *sp.dist-L1-norm-list xs2 ys2*

by(*rule sp.MetL1.nonneg*)
ultimately have *plus*: *Suc k* $\geq$ *k2*
using $*$ *Suc.prems*(*4*) **by** *linarith*
have *p*: *k2* $\geq$ *1*
using $*$ *False k2* **by** *auto*
then obtain *k3*::*nat* **where** *k3*: *k2* = *Suc k3*
using *not0-implies-Suc* **by** *force*
with *plus en* **have** *en*: *k* $\geq$ *k3* **and** *sp.dist-L1-norm-list xs2 ys2* $\leq$ *k* $-$ *k3*
by *auto*
hence (*xs2*, *ys2*) $\in$ *?R* ^^ (*k* $-$ *k3*)
using *Suc.prems*(*1*)[*of xs2 ys2 k* $-$ *k3*] *xs2 ys2* **by** *auto*
hence *A*: (*y* # *xs2*, *y* # *ys2*) $\in$ *?R2* ^^ (*k* $-$ *k3*)
using *L1-norm-list.L1-adj-iterate-Cons1*[*of xs2 n ys2* (*k* $-$ *k3*) *y*] *xs2 ys2* **by** *blast*

have |*int x* $-$ *int y*| = (*Suc k3*)
by(*auto simp*: *k2 k3*)
then have *B*: (*x* # *xs2*, *y* # *xs2*) $\in$ *?R2* ^^ *Suc k3*
by(*intro L1-norm-list.L1-adj-iterate-Cons2 xs2*,*auto*)

have (*x* # *xs2*, *y* # *ys2*) $\in$ *?R2* ^^ (*Suc k3* + (*k* $-$ *k3*))
by(*intro relpow-trans*[*of - y*#*xs2*] *A B*)
thus (*xs*, *ys*) $\in$ *?R2* ^^ (*Suc k*)
unfolding *xs ys* **using** *en k3* **by** *auto*
qed
qed
qed

end

hide-fact (**open**) *list-sum-dist-Cons*

end

theory *Laplace-Distribution*
imports *HOL*$-$*Probability.Probability*
Additional-Lemmas-for-Integrals
Additional-Lemmas-for-Calculation
begin

3 Laplace Distribution

3.1 Desity Function and Cumulative Distribution Function

We refer `HOL/Probability/Distributions.thy` in the standard library.

definition *laplace-density* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
  laplace-density l m x = (*if l* > *0 then exp*(−|*x* − *m*| / *l*) / (*2* ∗ *l*) *else 0*)

definition *laplace-CDF* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
  laplace-CDF l m x =
 (*if l* > *0 then*
 if x < *m then exp*((*x* − *m*) / *l*) / *2 else 1* − *exp*(−(*x* − *m*) / *l*) / *2 else 0*)

lemma *laplace-density-nonneg*[*simp*]:
  shows *0* ≤ *laplace-density l m x*
  unfolding *laplace-density-def* **by** *auto*

lemma *laplace-CDF-nonneg*[*simp*]:
  shows *0* ≤ *laplace-CDF l m x*
  unfolding *laplace-CDF-def*
proof−
  consider*0* ≥ *l*|*0* < *l* ∧ *x* < *m* | *0* < *l* ∧ *x* ≥ *m*
    by *linarith*
  thus *0* ≤ (*if 0* < *l then if x* < *m then exp* ((*x* − *m*) / *l*) / *2 else 1* − *exp* (− (*x* − *m*) / *l*) / *2 else 0*)
  proof(*cases*)
    assume *l* ≤ *0*
    thus *?thesis* **by** *auto*
  next assume *0* < *l* ∧ *x* < *m*
    thus *?thesis* **by** *auto*
  next assume *0* < *l* ∧ *x* ≥ *m*
    hence *exp* (− (*x* − *m*) / *l*) ≤ *1*
      by (*auto simp*: *divide-nonpos-pos*)
    hence *exp* (− (*x* − *m*) / *l*)/*2* ≤ *1*
      by *linarith*
    thus *?thesis* **by** *auto*
  qed
qed

lemma *borel-measurable-laplace-density*[*measurable*]:
  shows *laplace-density l m* ∈ *borel* $\rightarrow_M$ *borel*
  unfolding *laplace-density-def* **by** *auto*

lemma *borel-measurable-laplace-density2*[*measurable*]:
  shows (λ *m* :: *real*. (*laplace-density l m x*)) ∈ *borel* $\rightarrow_M$ *borel*

unfolding *laplace-density-def* **by** *auto*

lemma *laplace-density-mirror*:
shows *laplace-density l m* $(2 * m - x) =$ *laplace-density l m x*
unfolding *laplace-density-def* **by** *auto*

lemma *laplace-density-shift*:
shows *laplace-density l* $(m + c)$ $(x + c) =$ *laplace-density l m x*
unfolding *laplace-density-def* **by** *auto*

lemma *borel-measurable-laplace-CDF*[*measurable*]:
shows *laplace-CDF l m* $\in$ *borel* $\rightarrow_M$ *borel*
unfolding *laplace-CDF-def* **by** *auto*

lemma *borel-measurable-laplace-CDF2*[*measurable*]:
shows $(\lambda\ m :: real.$ *laplace-CDF l m x*$)\in$ *borel* $\rightarrow_M$ *borel*
unfolding *laplace-CDF-def* **by** *auto*

lemma *laplace-CDF-mirror*:
assumes $l > 0$
shows *laplace-CDF l m* $(2 * m - x) = 1 -$ *laplace-CDF l m x*
unfolding *laplace-CDF-def*
using *assms* **by** *auto*

lemma *laplace-CDF-shift*:
shows *laplace-CDF l* $(m + c)$ $(x + c) =$ *laplace-CDF l m x*
unfolding *laplace-CDF-def* **by** *auto*

lemma *nn-integral-laplace-density-pos*:
assumes *pos*[*arith*]: $0 < l$ **and** *1*: $a \geq m$
shows $(\int^+ x \in \{a..\}.$ *ennreal* (*laplace-density l m x*) ∂*lborel*) $= 1$
$-$ *laplace-CDF l m a*
proof$-$
from *1* **have** $(\int^+ x \in \{a..\}.$ *ennreal* (*laplace-density l m x*) ∂*lborel*)
$= (\int^+ x \in \{a..\}.\ (exp\ ((m - x) / l)/ (2 * l))\ \partial lborel)$
by(*intro nn-integral-cong*,*unfold laplace-density-def*)(*auto simp add*:
field-simps split: *split-indicator*)
also have ... $= 0 - (- exp\ ((m - a) / l) / 2)$
proof(*rule nn-integral-FTC-atLeast*)
show $((\lambda a. - exp\ ((m - a) / l) / 2) \longrightarrow 0)$ *at-top*
proof(*rule tendstoI*,*subst eventually-at-top-linorder*)
fix $e :: real$ **assume** *A0*: $0 < e$
show $\exists N. \forall n \geq N.\ dist\ (- exp\ ((m - n) / l) / 2)\ 0 < e$
proof(*intro exI allI impI*)
fix *n* **assume** *A1*: $m - ln\ (e * 2) * l + 1 \leq n$
have $(m - n) / l \leq (m - (m - ln\ (e * 2) * l + 1)) / l$
using *A1 pos* **by** (*auto simp*: *divide-le-cancel*)
also have ... $=(m - m + ln\ (e * 2) * l - 1) / l$
by *auto*

also have ... =(*ln* (*e* ∗ *2*) ∗ *l* − *1*) / *l*
by *auto*
also have ... <(*ln* (*e* ∗ *2*) ∗ *l*) / *l*
by (*auto simp*: *mult-imp-div-pos-less*)
also have ... ≤*ln* (*e* ∗ *2*)
using *pos* **by** *auto*
finally have (*m* − *n*) / *l* < *ln* (*e* ∗ *2*)**.**
hence *exp* ((*m* − *n*) / *l*) < *exp* (*ln* (*e* ∗ *2*))
by *auto*
also have ... = *e* ∗ *2*
using *A0* **by** *auto*
finally show *dist* (− *exp* ((*m* − *n*) / *l*) / *2*) *0* < *e***by** *auto*
qed
qed
fix *x* :: *real* **assume** *a* ≤ *x*
have *DERIV* (*exp* ∘ (λ*x*. *x* ∗ (*1* / *l*)) ∘ (λ *x*. (*m*−*x*))) *x* :> *exp* ((*m*−*x*) ∗ (*1* / *l*)) ∗ (*1* / *l*) ∗ (−*1*)
proof(*intro DERIV-chain*)
show (*exp has-real-derivative exp* ((*m* − *x*) ∗ (*1* / *l*))) (*at* ((*m* − *x*) ∗ (*1* / *l*)))
by *auto*
show ((λ*x*. *x* ∗ (*1* / *l*)) *has-real-derivative 1* / *l*) (*at* (*m* − *x*))
by (*metis DERIV-cmult-Id mult-commute-abs*)
have *P21*: ((−) *m*) = (λ *x*. *m* − *x*)
by *auto*
have ((λ *x*. *m* − *x*) *has-real-derivative 0* − *1*) (*at x*)
by (*rule DERIV-diff*,*auto intro*!:*DERIV-const DERIV-ident*)
thus((−) *m has-real-derivative* − *1*) (*at x*)
using *P21* **by** *auto*
qed
hence *P1*: *DERIV* (λ *y*. (−*1*/*2*) ∗ (*exp* ∘ (λ*x*. *x* ∗ (*1* / *l*)) ∘ (λ *x*. (*m*−*x*))) *y*) *x* :> (−*1*/*2*) ∗ (*exp* ((*m*−*x*) ∗ (*1* / *l*)) ∗ (*1* / *l*) ∗ (−*1*))
by(*rule DERIV-cmult*)
have *P2*: (λ *y*. (−*1*/*2*) ∗ (*exp* ∘ (λ*x*. *x* ∗ (*1* / *l*)) ∘ (λ *x*. (*m*−*x*))) *y*) = ((λ*a*. − (*exp* ((*m* − *a*) / *l*) / *2*)))
by *auto*
have *P3*: (−*1*/*2*) ∗ (*exp* ((*m*−*x*) ∗ (*1* / *l*)) ∗ (*1* / *l*) ∗ (−*1*)) = *exp* ((*m* − *x*) / *l*) / (*2* ∗ *l*)
by *auto*
from *P1 P2 P3* **show** ((λ*a*. − *exp* ((*m* − *a*) / *l*) / *2*) *has-real-derivative exp* ((*m* − *x*) / *l*) / (*2* ∗ *l*)) (*at x*)
by *auto*
qed(*auto*)
also have ... = *ennreal* (*exp* ((*m* − *a*) / *l*) / *2*)
by *auto*
also have ... = *ennreal* (*1* − *laplace-CDF l m a*)
using *laplace-CDF-def pos 1* **by** *auto*
finally show (∫⁺*x*∈{*a*..}. *ennreal* (*laplace-density l m x*)∂*lborel*) = *ennreal* (*1* − *laplace-CDF l m a*) **by** *auto*

qed

lemma *nn-integral-laplace-density-pos-center*:
assumes *pos*[*arith*]: $0 < l$
shows $(\int^{+} x \in \{m..\}.\ ennreal\ (laplace\text{-}density\ l\ m\ x)\ \partial lborel) = 1/2$
proof−
have $(\int^{+} x \in \{m..\}.\ ennreal\ (laplace\text{-}density\ l\ m\ x)\ \partial lborel) = ennreal\ (1 - laplace\text{-}CDF\ l\ m\ m)$
using *nn-integral-laplace-density-pos* **by** *auto*
also have ... $= 1/2$
by (*auto simp*: *laplace-CDF-def assms divide-ennreal-def*)
finally show *?thesis*.
qed

lemma *nn-integral-laplace-density-neg*:
assumes *pos*[*arith*]: $0 < l$ **and** *1*: $a \leq m$
shows $(\int^{+} x \in \{..a\}.\ ennreal\ (laplace\text{-}density\ l\ m\ x)\ \partial lborel) = laplace\text{-}CDF\ l\ m\ a$
proof−
from *1* **have** $(\int^{+} x \in \{..a\}.\ ennreal\ (laplace\text{-}density\ l\ m\ x)\ \partial lborel) = (\int^{+} x \in \{..a\}.\ (exp\ ((x - m)\ /\ l)/\ (2 * l))\ \partial lborel)$
by(*intro nn-integral-cong*,*unfold laplace-density-def*)(*auto simp add*: *field-simps split*: *split-indicator*)
also have ... $= (exp\ ((a - m)\ /\ l)\ /\ 2) - 0$
proof(*rule nn-integral-FTC-atGreatest*)
show $((\lambda a.\ exp\ ((a - m)\ /\ l)\ /\ 2) \longrightarrow 0)$ *at-bot*
proof(*rule tendstoI*,*subst eventually-at-bot-linorder*)
fix *e* :: *real* **assume** *A0*: $0 < e$
show $\exists N.\ \forall n \leq N.\ dist\ (exp\ ((n - m)\ /\ l)\ /\ 2)\ 0 < e$
proof(*intro exI allI impI*)
fix *n* **assume** $n \leq m + ln\ (e * 2) * l - 1$
hence $(n - m) \leq ((m + ln\ (e * 2) * l - 1) - m)$
by *auto*
hence $(n - m)/l \leq (ln\ (e * 2) * l - 1)\ /l$
by (*auto simp*: *pos divide-le-cancel*)
also have ... $= ln\ (e * 2) - 1/l$
by (*auto simp*: *diff-divide-distrib*)
also have ... $< ln\ (e * 2)$
by *auto*
finally have $(n - m)/l < ln\ (e * 2)$.
hence $exp\ ((n - m)\ /\ l) < exp(ln\ (e * 2))$
by *auto*
also have ... $= e * 2$
by (*auto simp*: *A0*)
finally show $dist\ (exp\ ((n - m)\ /\ l)\ /\ 2)\ 0 < e$**by** *auto*
qed
qed
fix *x* :: *real* **assume** *A1*: $x \leq a$

have *DERIV* (*exp* ∘ (λ*x*. *x* ∗ (*1* / *l*)) ∘ (λ *x*. (*x* − *m*))) *x* :> *exp* ((*x*−*m*) ∗ (*1* / *l*)) ∗ (*1* / *l*) ∗ *1*
proof(*rule DERIV-chain*)
show ((λ*x*. *x* − *m*) *has-real-derivative 1*) (*at x*)
using *DERIV-const DERIV-ident Deriv.field-differentiable-diff*
proof −
have ∃ *r ra*. *r* − *ra* = *1* ∧ ((λ*r*. *m*) *has-real-derivative ra*) (*at x*) ∧ ((λ*r*. *r*) *has-real-derivative r*) (*at x*)
using *DERIV-const DERIV-ident diff-zero* **by** *blast*
then show *?thesis*
using *Deriv.field-differentiable-diff* **by** *force*
qed
show (*exp* ∘ (λ*x*. *x* ∗ (*1* / *l*)) *has-real-derivative exp* ((*x* − *m*) ∗ (*1* / *l*)) ∗ (*1* / *l*)) (*at* (*x* − *m*))
proof(*rule DERIV-chain*)
show ((λ*x*. *x* ∗ (*1* / *l*)) *has-real-derivative 1* / *l*) (*at* (*x* − *m*))
by (*metis DERIV-cmult-Id mult-commute-abs*)
show (*exp has-real-derivative exp* ((*x* − *m*) ∗ (*1* / *l*))) (*at* ((*x* − *m*) ∗ (*1* / *l*)))
by *auto*
qed
qed
hence *P1*: *DERIV* (λ *y*. (*1*/*2*) ∗ (*exp* ∘ (λ*x*. *x* ∗ (*1* / *l*)) ∘ (λ *x*. (*x* − *m*))) *y*) *x* :> (*1*/*2*) ∗(*exp* ((*x*−*m*) ∗ (*1* / *l*)) ∗ (*1* / *l*) ∗ *1*)
by(*rule DERIV-cmult*)
have *P2*: (λ *y*. (*1*/*2*) ∗ (*exp* ∘ (λ*x*. *x* ∗ (*1* / *l*)) ∘ (λ *x*. (*x* − *m*))) *y*) = ((λ*a*. *exp* ((*a* − *m*) / *l*) / *2*))
by *auto*
have *P3*: (*1*/*2*) ∗(*exp* ((*x*−*m*) ∗ (*1* / *l*)) ∗ (*1* / *l*) ∗ *1*) = *exp* ((*x* − *m*) / *l*) / (*2* ∗ *l*)
by *auto*
from *P1 P2 P3*
show ((λ*a*. *exp* ((*a* − *m*) / *l*) / *2*) *has-real-derivative exp* ((*x* − *m*) / *l*) / (*2* ∗ *l*)) (*at x*)
by *auto*
qed(*auto*)
also have ... = *ennreal* (*exp* ((*a* − *m*) / *l*) / *2*)
by *auto*
also have ... = *ennreal* (*laplace-CDF l m a*)
using *laplace-CDF-def pos 1* **by** *auto*
finally show *?thesis*.
qed

lemma *nn-integral-laplace-density-neg-center*:
assumes *pos*[*arith*]: *0* < *l*
shows (∫⁺ *x* ∈ {..*m*}. *ennreal* (*laplace-density l m x*) ∂*lborel*) = *1*/*2*
proof−
have (∫⁺ *x* ∈ {..*m*}. *ennreal* (*laplace-density l m x*) ∂*lborel*) =

ennreal (*laplace-CDF l m m*)
using *nn-integral-laplace-density-neg* **by** *auto*
also have ... = *1/2*
by (*auto simp*: *laplace-CDF-def assms divide-ennreal-def*)
finally show *?thesis*.
qed

proposition *nn-integral-laplace-density-mass-1*:
assumes *pos*[*arith*]: *0* < *l*
shows ($\int^{+}$ *x*. *ennreal* (*laplace-density l m x*) ∂*lborel*) = *1*
proof−
have ($\int^{+}$ *x*. *ennreal* (*laplace-density l m x*) ∂*lborel*) = ($\int^{+}$ *x* $\in$ {..*m*}. *ennreal* (*laplace-density l m x*) ∂*lborel*) + ($\int^{+}$ *x* $\in$ {*m*..}. *ennreal* (*laplace-density l m x*) ∂*lborel*)
by(*rule nn-integral-lborel-split*,*auto*)
also have ... = *1/2* + *1/2*
by(*auto simp*:*nn-integral-laplace-density-neg-center*[*of l m*,*OF pos*] *nn-integral-laplace-density-pos-center*[*of l m*,*OF pos*])
also have ... = *2/2*
by (*auto simp*: *add-divide-distrib-ennreal*[*THEN sym*])
also have ... = *1*
by *auto*
finally show *?thesis*.
qed

lemma *emeasure-laplace-density-mass-1*:
0 < *l* $\Longrightarrow$ *emeasure* (*density lborel* (*laplace-density l m*)) *UNIV* = *1*
by(*auto simp*: *emeasure-density nn-integral-laplace-density-mass-1*)

proposition *nn-integral-laplace-density*:
assumes *pos*[*arith*]: *0* < *l*
shows ($\int^{+}$ *x* $\in$ {..*a*}. *ennreal* (*laplace-density l m x*) ∂*lborel*) = *laplace-CDF l m a*
proof(*casesa* $\leq$ *m*)
case *True*
thus *?thesis* **using** *nn-integral-laplace-density-neg pos* **by** *auto*
next
case *False*
have *eq0*: *1* − *laplace-CDF l m a* = ($\int^{+}$ *x* $\in$ {*a*..}. *ennreal* (*laplace-density l m x*) ∂*lborel*)
using *nn-integral-laplace-density-pos pos False* **by** *auto*
also have ... = ($\int^{+}$ *x* $\in$ {*a*<..}. *ennreal* (*laplace-density l m x*) ∂*lborel*)
by(*auto simp*: *nn-integral-interval-greaterThan-atLeast*)
finally have *eqA*: *1* − *laplace-CDF l m a* = ($\int^{+}$ *x* $\in$ {*a*<..}. *ennreal* (*laplace-density l m x*) ∂*lborel*).

hence *eqA2*: *1* = ($\int^{+}$ *x* $\in$ {*a*<..}. *ennreal* (*laplace-density l m x*) ∂*lborel*) + *laplace-CDF l m a*

proof(*intro ennreal-diff-add-transpose*)
show *ennreal* (*laplace-CDF l m a*) $\leq$ (*1* :: *ennreal*)
using *False laplace-CDF-def* **by** *auto*
show *1* $-$ *ennreal* (*laplace-CDF l m a*) = ($\int^+ x$ :: *real*$\in\{a<..\}$. *ennreal* (*laplace-density l m x*)∂*lborel*)
using *eqA ennreal-1 ennreal-minus laplace-CDF-nonneg* **by** *metis*
qed

have *1* = ($\int^+$ *x*. *ennreal* (*laplace-density l m x*) ∂*lborel*)
using *nn-integral-laplace-density-mass-1 pos* **by** *auto*
also have ... = ($\int^+$ *x*$\in\{..a\}$. *ennreal* (*laplace-density l m x*) ∂*lborel*) + ($\int^+$ *x*$\in$ $\{a<..\}$. *ennreal* (*laplace-density l m x*) ∂*lborel*)
using *nn-integral-lborel-split nn-integral-interval-greaterThan-atLeast* **by** *auto*
finally have *eqB*: *1* = ($\int^+$ *x*$\in\{..a\}$. *ennreal* (*laplace-density l m x*) ∂*lborel*) + ($\int^+$ *x*$\in$ $\{a<..\}$. *ennreal* (*laplace-density l m x*) ∂*lborel*).

from *eqA2 eqB*
have ($\int^+$ *x* $\in$ $\{a<..\}$. *ennreal* (*laplace-density l m x*) ∂*lborel*) + *laplace-CDF l m a* = ($\int^+$ *x*$\in\{..a\}$. *ennreal* (*laplace-density l m x*) ∂*lborel*) + ($\int^+$ *x*$\in$ $\{a<..\}$. *ennreal* (*laplace-density l m x*) ∂*lborel*)
by *auto*
thus *?thesis*
using *eqA* **by** *fastforce*
qed

lemma *emeasure-laplace-density*:
assumes *0* < *l*
shows *emeasure* (*density lborel* (*laplace-density l m*)) $\{..\ a\}$ = *laplace-CDF l m a*
by (*auto simp*: *emeasure-density nn-integral-laplace-density assms*)

3.2 Moments

lemma *laplace-moment-0-a*:
assumes *pos*[*arith*]: *0* < *l*
and *posa*[*arith*]: *0* $\leq$ *a*
shows *has-bochner-integral lborel* (λ *x*. (*indicator* $\{0..a\}$ *x* $*_R$ (*laplace-density l 0 x*)))(*1*/*2* $-$ *exp*($-a$ / *l*) /*2*)
proof(*rule has-bochner-integral-nn-integral*)
show (λ*x*. *indicat-real* $\{0..a\}$ *x* $*_R$ *laplace-density l 0 x*) $\in$ *borel-measurable lborel*
by *measurable*
show *AE x in lborel*. *0* $\leq$ *indicat-real* $\{0..a\}$ *x* $*_R$ *laplace-density l 0 x*
by *auto*
show *0* $\leq$*1* / *2* $-$ *exp* ($-$ *a* / *l*) / *2*
by(*auto simp add*: *pos posa*)
have ($\int^+$ *x* $\in$ $\{0..\}$. *ennreal* (*laplace-density l 0 x*) ∂*lborel*) = ($\int^+$

$x \in \{0..a\}$. *ennreal* (*laplace-density l 0 x*) ∂*lborel*) + ($\int^+$ $x \in \{a..\}$. *ennreal* (*laplace-density l 0 x*) ∂*lborel*)
by(*rule nn-set-integral-lborel-split-AtLeast*,*auto*)
hence *1/2* = ($\int^+$ $x \in \{0..a\}$. *ennreal* (*laplace-density l 0 x*) ∂*lborel*) + (*1* − *laplace-CDF l 0 a*)
by(*auto simp*: *nn-integral-laplace-density-pos-center*[*of l 0*,*OF pos*] *nn-integral-laplace-density-pos*[*of l 0 a*,*OF pos posa*])
also have ... = ($\int^+$ $x \in \{0..a\}$. *ennreal* (*laplace-density l 0 x*) ∂*lborel*) + (*1* − (*1* − *exp*(− *a* / *l*) / *2*))
unfolding *laplace-CDF-def* **by** *auto*
finally have *ennreal* (*1/2*) − *exp*(− *a* / *l*) / *2* = ($\int^+$ $x \in \{0..a\}$. *ennreal* (*laplace-density l 0 x*) ∂*lborel*) + (*1* − (*1* − *exp*(− *a* / *l*) / *2*)) − *exp*(− *a* / *l*) / *2*
by (*metis ennreal-divide-numeral ennreal-numeral numeral-One zero-le-numeral*)
hence *ennreal* (*1/2*) − *exp*(− *a* / *l*) / *2* = ($\int^+$ $x \in \{0..a\}$. *ennreal* (*laplace-density l 0 x*) ∂*lborel*)
by *auto*
hence *eq0a*: *1/2* − *exp*(− *a* / *l*) / *2* = ($\int^+$ $x \in \{0..a\}$. *ennreal* (*laplace-density l 0 x*) ∂*lborel*)
by (*metis* (*full-types*) *ennreal-minus exp-ge-zero zero-le-divide-iff zero-le-numeral*)
also have ... = ($\int^+$ *x*. *ennreal* (*indicat-real* {*0..a*} *x* $*_R$ *laplace-density l 0 x*) ∂*lborel*)
by (*auto simp*: *mult.commute nn-integral-set-ennreal*)
finally show ($\int^+$ *x*. *ennreal* (*indicat-real* {*0..a*} *x* $*_R$ *laplace-density l 0 x*) ∂*lborel*) = (*1/2* − *exp*(− *a* / *l*) / *2*)
by *presburger*
qed

lemma *laplace-moment-0-pos*:
assumes *pos*[*arith*]: *0* < *l*
shows *has-bochner-integral lborel* (λ *x*. (*indicator* {*0..*} *x* $*_R$ (*laplace-density l 0 x*)))(*1/2* :: *real*)
proof(*rule has-bochner-integral-nn-integral*)
have ($\int^+$ *x*. *ennreal* (*laplace-density l 0 x* ∗ *indicat-real* {*0..*} *x*) ∂*lborel*) = *1* / *2*
using *nn-integral-laplace-density-pos-center*
by (*auto simp*: *nn-integral-set-ennreal*)
thus($\int^+$ *x*. *ennreal* (*indicat-real* {*0..*} *x* $*_R$ *laplace-density l 0 x*) ∂*lborel*) = *ennreal* (*1* / *2*)
by (*auto simp*: *divide-ennreal-def mult.commute*)
show (λ*x*. *indicat-real* {*0..*} *x* $*_R$ *laplace-density l 0 x*) $\in$ *borel-measurable lborel*
by *measurable*
show *0* $\leq$ (*1/2* :: *real*)
by *auto*
show *AE x in lborel*. *0* $\leq$ *indicat-real* {*0..*} *x* $*_R$ *laplace-density l 0 x*

by *auto*
qed

lemma *laplace-moment-0*:
fixes k :: *nat*
assumes *pos*[*arith*]: $0 < l$
shows *has-bochner-integral lborel* (λ *x*. (*indicator* {*0*..} $x *_R$ ((*laplace-density* *l 0 x*) $*$ $x\hat{\ }k$)))(*fact* $k * l\hat{\ }k/2$)
and (λa. *LBINT x*. *indicator* {*0*..*a*} $x *_R$ ((*laplace-density l 0 x*) $*$ $x\hat{\ }k$)) $\longrightarrow$ (*LBINT y*. (*indicator* {*0*..} $y *_R$ ((*laplace-density l 0 y*) $*$ $y\hat{\ }k$)))
proof(*induction k*)
let *?f* =λ (*k* :: *nat*) (*x* :: *real*). ($1/(2 * l)$) $*$ (*exp*($- x / l$) $*$ $x\hat{\ }k$)
show *cond0*: *has-bochner-integral lborel* (λx. *indicat-real* {*0*..} $x *_R$ (*laplace-density l 0 x* $*$ $x \hat{\ } 0$)) (*fact* $0 * l \hat{\ } 0 / 2$)
using *laplace-moment-0-pos* **by** *auto*
show *cond1*: (λx. *LBINT xa*. *indicat-real* {*0*..*real x*} $xa *_R$ (*laplace-density l 0 xa* $*$ $xa \hat{\ } 0$)) $\longrightarrow$ (*LBINT y*. *indicat-real* {*0*..} $y *_R$ (*laplace-density l 0 y* $*$ $y \hat{\ } 0$))
proof(*rule tendstoI*, *subst eventually-at-top-linorder*, *intro exI allI impI*)
fix *e* :: *real* **and** *n* :: *nat*
assume *pose*: $0 < e$
assume *ineqn*: *nat*(*floor* ($l * ln$ (1 / ($2 * e$)))) + $1 \le n$
hence *posn*: $0 \le$ *real n***by** *auto*
from *ineqn* **have** *ineqn2*: $l * ln$ (1 / ($2 * e$)) $<$ *real n*
by *linarith*
show *dist* (*LBINT xa*. *indicat-real* {*0*..*real n*} $xa *_R$ (*laplace-density l 0 xa* $*$ $xa \hat{\ } 0$)) (*LBINT y*. *indicat-real* {*0*..} $y *_R$ (*laplace-density l 0 y* $*$ $y \hat{\ } 0$)) $< e$
proof(*unfold dist-real-def*)
have *eq1*: (*LBINT y*. *indicat-real* {*0*..} $y *$ *laplace-density l 0 y*) $= 1/2$
using *has-bochner-integral-integral-eq cond0* **by** *force*
hence *eq2*: (*LBINT x*. *indicat-real* {*0*..*real n*} $x *$ *laplace-density l 0 x*) $= 1/2 -$ *exp*($-$ *real n* / *l*) /2
using *has-bochner-integral-integral-eq*[*OF laplace-moment-0-a*[*of lreal n*, *OF pos posn*]] **by** *auto*
from *eq1 eq2 dist-real-def* **have** |(*LBINT x*. *indicat-real* {*0*..*real n*} $x *$ *laplace-density l 0 x*) $-$ (*LBINT y*. *indicat-real* {*0*..} $y *$ *laplace-density l 0 y*)| = *exp*($-$ *real n* / *l*) /2
proof $-$
have *f1*: (*LBINT r*. *indicat-real* {*0*..*real n*} $r *$ *laplace-density l 0 r*) $-$ (*LBINT r*. *indicat-real* {*0*..} $r *$ *laplace-density l 0 r*) < 0
by (*auto simp*: *eq1 eq2*)
have *f2*: *exp* ($-$ *real n* / *l*) / $2 = -$ ((*LBINT r*. *indicat-real* {*0*..*real n*} $r *$ *laplace-density l 0 r*) $-$ (*LBINT r*. *indicat-real* {*0*..} r $*$ *laplace-density l 0 r*))
using *eq1 eq2* **by** *linarith*

show *?thesis*
using *f2 f1* **by** *auto*
qed
also have ... $<$ *exp*(*ln* (*2* $*$ *e*)) /*2*
proof−
have *l* $*$ *ln* (*1* / (*2* $*$ *e*)) $<$ *real n*
using *ineqn2* **by** *auto*
hence *l* $*$ *ln* (*1* / (*2* $*$ *e*)) / *l* $<$ *real n* / *l*
using *pos divide-strict-right-mono* **by** *blast*
hence *ln* (*inverse*(*2* $*$ *e*)) $<$ *real n* / *l*
by (*auto simp*: *inverse-eq-divide*)
hence −*ln* (*2* $*$ *e*) $<$ *real n* / *l*
by (*metis ln-inverse mult-pos-pos pose zero-less-numeral*)
hence *ln* (*2* $*$ *e*) $>$ − *real n* / *l*
by *auto*
thus *?thesis* **by** *auto*
qed
also have ... = *e*
by (*auto simp*: *pose*)
finally show |(*LBINT xa. indicat-real* {*0*..*real n*} *xa* $*_R$ (*laplace-density l 0 xa* $*$ *xa* ^ *0*)) − (*LBINT y. indicat-real* {*0*..} *y* $*_R$ (*laplace-density l 0 y* $*$ *y* ^ *0*))| $<$ *e***by** *auto*
qed
qed

fix *k* :: *nat*
assume *khasBoh*: *has-bochner-integral lborel* (λ*x. indicat-real* {*0*..} *x* $*_R$ (*laplace-density l 0 x* $*$ *x* ^ *k*)) (*fact k* $*$ *l* ^ *k* / *2*) **and** *kconverge*: (λ*x. LBINT xa. indicat-real* {*0*..*real x*} *xa* $*_R$ (*laplace-density l 0 xa* $*$ *xa* ^ *k*)) ⟶ (*LBINT y. indicat-real* {*0*..} *y* $*_R$ (*laplace-density l 0 y* $*$ *y* ^ *k*))
have *kconverge2*: (λ*x. LBINT xa. 1* / (*2* $*$ *l*) $*$ (*exp* (− *xa* / *l*) $*$ *xa* ^ *k*) $*$ *indicat-real* {*0*..*real x*} *xa*) ⟶(*LBINT y. indicat-real* {*0*..} *y* $*$ (*laplace-density l 0 y* $*$ *y* ^ *k*))
proof(*subst tendsto-cong*)
show (λ*x. LBINT xa. indicat-real* {*0*..*real x*} *xa* $*$ (*laplace-density l 0 xa* $*$ *xa* ^ *k*)) ⟶ (*LBINT y. indicat-real* {*0*..} *y* $*$ (*laplace-density l 0 y* $*$ *y* ^ *k*))
using *kconverge* **by** *auto*
have *eq1*: ⋀*x.* (*LBINT xa. 1* / (*2* $*$ *l*) $*$ (*exp* (− *xa* / *l*) $*$ *xa* ^ *k*) $*$ *indicat-real* {*0*..*real x*} *xa*) = (*LBINT xa. indicat-real* {*0*..*real x*} *xa* $*$ (*laplace-density l 0 xa* $*$ *xa* ^ *k*))
unfolding *laplace-density-def* **by**(*rule Bochner-Integration.integral-cong*,*auto simp*: *split*: *split-indicator*)
thus $\forall_F$ *x in sequentially.* (*LBINT xa. 1* / (*2* $*$ *l*) $*$ (*exp* (− *xa* / *l*) $*$ *xa* ^ *k*) $*$ *indicat-real* {*0*..*real x*} *xa*) = (*LBINT xa. indicat-real* {*0*..*real x*} *xa* $*$ (*laplace-density l 0 xa* $*$ *xa* ^ *k*))
by(*rule eventually-sequentiallyI*)
qed

have (*LBINT y. indicat-real* {*0..*} *y* ∗ (*laplace-density l 0 y* ∗ *y* ^ *k*)) = (*LBINT x. 1* / (*2* ∗ *l*) ∗ (*exp* (− *x* / *l*) ∗ *x* ^ *k*) ∗ *indicat-real* {*0..*} *x*)
unfolding *laplace-density-def* **by**(*rule Bochner-Integration.integral-cong*,*auto simp*: *split*: *split-indicator*)
hence *kconverge3*: (λ*x. LBINT xa. 1* / (*2* ∗ *l*) ∗ (*exp* (− *xa* / *l*) ∗ *xa* ^ *k*) ∗ *indicat-real* {*0..real x*} *xa*) ⟶(*LBINT x. 1* / (*2* ∗ *l*) ∗ (*exp* (− *x* / *l*) ∗ *x* ^ *k*) ∗ *indicat-real* {*0..*} *x*)
using *kconverge2* **by** *presburger*

from *khasBoh* **have** *A0*: *has-bochner-integral lborel* (λ*x. indicat-real* {*0..*} *x* $*_R$ (*?f k x*)) (*fact k* ∗ *l* ^ *k* / *2*)
by(*subst has-bochner-integral-cong*, *auto intro*: *simp*: *assms laplace-density-def split*:*split-indicator*)
have *A01*: *integrable lborel* (λ*x. indicat-real* {*0..*} *x* $*_R$ (*?f k x*))
using *has-bochner-integral-iff A0* **by** *blast*
have *A02*: (*LBINT x. indicat-real* {*0..*} *x* $*_R$ (*?f k x*)) = (*fact k* ∗ *l* ^ *k* / *2*)
using *has-bochner-integral-iff A0* **by** *blast*

let *?AF* =λ (*x* :: *real*). *x*^(*Suc k*)
let *?Ag* =λ (*x* :: *real*). *exp*(− *x* / *l*)
let *?Af* =λ (*x* :: *real*). (*Suc k*) ∗ *x*^*k*
let *?AG* =λ (*x* :: *real*). (− *l*) ∗ *exp*(− *x* / *l*)

have *FgfSuc*: (λ *x* :: *real*. (*1*/(*2* ∗ *l*)) ∗ (*?AF x* ∗ *?Ag x*)) = *?f*(*Suc k*)
by (*auto simp*: *mult.commute*)
have *fgSucf*: (λ *x* :: *real*. (*1*/(*2* ∗ *l*)) ∗ (*?Af x* ∗ *?Ag x*)) = (λ *x* :: *real*. (*Suc k*) ∗ *?f k x*)
by *fastforce*
have *fglSucf*: (λ *x* :: *real*. (*1*/(*2* ∗ *l*)) ∗ (*?Af x* ∗ *?AG x*)) = (λ *x* :: *real*. (− *l*) ∗ (*Suc k*) ∗ *?f k x*)
by *fastforce*
have *cont-f*: ⋀*x. isCont ?Af x*
by *auto*
have *cont-g*: ⋀*x. isCont ?Ag x*
by *auto*
have *drv-F*: ⋀*x. DERIV ?AF x* :> *?Af x*
proof−
fix *x* :: *real*
show *DERIV ?AF x* :> *?Af x*
by (*metis DERIV-pow add-0 add-Suc add-diff-cancel-left'*)
qed
have *drv-G*: ⋀*x. DERIV ?AG x* :> *?Ag x*
proof−
fix *x* :: *real*
have (*exp o* (λ*x.* (− *x* / *l*)) *has-real-derivative exp* (− *x* / *l*) ∗ (− *1* / *l*)) (*at x*)

proof(*rule DERIV-chain*)
show (*exp has-real-derivative exp* (− *x* / *l*)) (*at* (− *x* / *l*))**by** *auto*
show ((λ*x*. − *x* / *l*) *has-real-derivative* − *1* / *l*) (*at x*)
using *DERIV-cdivide DERIV-ident DERIV-mirror* **by** *blast*
qed
hence *drv-G2*: ((λ*x*. − *l* ∗ (*exp o* (λ*x*. (− *x* / *l*)))*x*) *has-real-derivative* − *l* ∗ (*exp* (− *x* / *l*) ∗ (− *1* / *l*))) (*at x*)
by(*rule DERIV-cmult*)
thus*DERIV ?AG x* :> *?Ag x*
proof−
have (λ*x*. − *l* ∗ (*exp o* (λ*x*. (− *x* / *l*)))*x*) = *?AG*
unfolding *comp-def* **by** *auto*
moreover have − *l* ∗ (*exp* (− *x* / *l*) ∗ (− *1* / *l*)) = *?Ag x*
by *auto*
ultimately show *?thesis*
using *drv-G2* **by** *auto*
qed
qed

{
fix *a* :: *real* **assume** *posa*: *0* ≤ *a*
have *integrable lborel* (λ*x*.((*?AF x*) ∗ (*?Ag x*) + (*?Af x*) ∗ (*?AG x*)) ∗ *indicator* {*0* .. *a*} *x*)
by(*intro integral-by-parts-integrable cont-f cont-g drv-F drv-G posa*)
hence *A2*: *integrable lborel* (λ*x*. (*?f*(*Suc k*) *x* + (− *l*) ∗ (*Suc k*) ∗ (*?f k x*)) ∗ *indicator* {*0* .. *a*} *x*)
by(*auto simp*:*field-simps*)
hence *integrable lborel* (λ*x*. ((*?f*(*Suc k*) *x* + (− *l*) ∗ (*Suc k*) ∗ (*?f k x*)) ∗ *indicator* {*0* .. *a*} *x*) − ((− *l*) ∗ (*Suc k*) ∗ (*?f k x*) ∗ *indicator* {*0* .. *a*} *x*))
proof−
show *integrable lborel* (λ*x*. ((*?f*(*Suc k*) *x* + (− *l*) ∗ (*Suc k*) ∗ (*?f k x*)) ∗ *indicator* {*0* .. *a*} *x*) − ((− *l*) ∗ (*Suc k*) ∗ (*?f k x*) ∗ *indicator* {*0* .. *a*} *x*))
proof(*intro Bochner-Integration.integrable-diff A2 posa*)
have *A03*: *integrable lborel* (λ*x*. (*indicator* {*0* ..} *x*) ∗(*?f k x*) ∗ *indicator* {*0* .. *a*} *x*)
proof(*rule integrable-real-mult-indicator*)
show {*0*..*a*} ∈ *sets lborel***by** *auto*
show *integrable lborel* (λ*x*. *indicat-real* {*0*..} *x* ∗ (*?f k x*))
using *A01* **by** *auto*
qed
have *A04*: *integrable lborel* (λ*x*. (*?f k x*) ∗ *indicator* {*0* .. *a*} *x*)
proof−
have (λ*x*. (*indicator* {*0* ..} *x*) ∗(*?f k x*) ∗ *indicator* {*0* .. *a*} *x*) = (λ*x*. (*?f k x*) ∗ *indicator* {*0* .. *a*} *x*)
by(*auto simp*:*field-simps split*: *split-indicator*)

thus *?thesis*
using *A03* **by** *auto*
qed
hence *integrable lborel* ($\lambda x.$ ($-$ *l*) $*$ (*Suc k*) $*$((*?f k x*) $*$ *indicat-real* {*0..a*} *x*))
by(*intro integrable-mult-right A04*)
thus *integrable lborel* ($\lambda x.$ ($-$ *l*) $*$ (*Suc k*) $*$ (*?f k x*) $*$ *indicat-real* {*0..a*} *x*)
by(*auto simp*:*field-simps split*: *split-indicator*)
qed
qed
hence *integrable lborel* ($\lambda x.$ (*?f*(*Suc k*) *x* $*$ *indicator* {*0* .. *a*} *x*))
by (*auto simp*:*field-simps*)
} **note** *integrabledSuck* = *this*

{
fix *a* :: *real* **assume** *posa*: $0 \leq a$
have ($\int$ *x*. (*?AF x* $*$ *?Ag x*) $*$ *indicator* {*0* .. *a*} *x* ∂*lborel*) = *?AF a* $*$ *?AG a* $-$ *?AF 0* $*$ *?AG 0* $-$ $\int$ *x*. (*?Af x* $*$ *?AG x*) $*$ *indicator* {*0* .. *a*} *x* ∂*lborel*
by(*intro integral-by-parts cont-f cont-g drv-F drv-G posa*)
hence ($\int$ *x*. *?f*(*Suc k*) *x* $*$ *indicator* {*0* .. *a*} *x* ∂*lborel*) = (*1*/(*2* $*$ *l*)) $*$(*?AF a* $*$ *?AG a* $-$ *?AF 0* $*$ *?AG 0*) $-$ $\int$ *x*. ($-$ *l*) $*$ (*Suc k*) $*$ (*?f k x*) $*$ *indicator* {*0* .. *a*} *x* ∂*lborel*
by (*auto simp*:*field-simps*)
also have ... = (*1*/(*2* $*$ *l*)) $*$(*?AF a* $*$ *?AG a* $-$ *?AF 0* $*$ *?AG 0*) $-$ ($-$ *l*) $*$ $\int$ *x*. (*Suc k*) $*$ (*?f k x*) $*$ *indicator* {*0* .. *a*} *x* ∂*lborel*
using *integral-scaleR-right* **by** *auto*
also have ... = (*1*/(*2* $*$ *l*)) $*$(*?AF a* $*$ *?AG a* $-$ *?AF 0* $*$ *?AG 0*) + *l* $*$ $\int$ *x*. (*Suc k*) $*$ ((*?f k x*) $*$ *indicator* {*0* .. *a*} *x*) ∂*lborel*
by (*auto simp*:*field-simps*)
also have ... = (*1*/(*2* $*$ *l*)) $*$(*?AF a* $*$ *?AG a* $-$ *?AF 0* $*$ *?AG 0*) + *l* $*$ (*Suc k*) $*$ $\int$ *x*. (*?f k x*) $*$ *indicator* {*0* .. *a*} *x* ∂*lborel*
using *integral-scaleR-right* **by** *auto*
finally have ($\int$ *x*. *?f*(*Suc k*) *x* $*$ *indicator* {*0* .. *a*} *x* ∂*lborel*) = (*1*/(*2* $*$ *l*)) $*$(*?AF a* $*$ *?AG a* $-$ *?AF 0* $*$ *?AG 0*) + *l* $*$ (*Suc k*) $*$ $\int$ *x*. (*?f k x*) $*$ *indicator* {*0* .. *a*} *x* ∂*lborel*.
} **note** *integralfSuck* = *this*

let *?H* =λ (*i* :: *nat*). λ *x* :: *real*. *?f*(*Suc k*) *x* $*$ *indicator* {*0* .. *i*} *x*
have *Hi*: $\bigwedge$*i*. *integrable lborel* (*?H i*)
using *exp-ge-zero exp-total integrabledSuck* **by** *force*

have *Hmono*: *AE x in lborel*. *mono* (λn. *?H n x*)
by(*auto simp*: *mono-def split*: *split-indicator*)

have *Intervallim*: $\bigwedge$*x*. (λ*xa*. *indicat-real* {*0..real xa*} *x*) $\longrightarrow$ *indicat-real* {*0..*} *x*
proof$-$

fix *x::real* **show** $(\lambda xa.\ \mathit{indicat\text{-}real}\ \{0..\mathit{real}\ xa\}\ x) \longrightarrow \mathit{indicat\text{-}real}\ \{0..\}\ x$
proof(*unfold tendsto-def eventually-sequentially,intro allI impI*)
fix S :: *real set* **assume** *open S* **and** *inInt*: $\mathit{indicat\text{-}real}\ \{0..\}\ x \in S$
show $\exists N.\ \forall n \geq N.\ \mathit{indicat\text{-}real}\ \{0..\mathit{real}\ n\}\ x \in S$
proof(*casesx* < 0)
case *True*
thus *?thesis*
using *inInt* **by** *auto*
next
case *False*
hence $1 \in S$
using *inInt* **by** *auto*
hence $\forall n \geq \mathit{nat}(\mathit{floor}\ x + 1).\ \mathit{indicat\text{-}real}\ \{0..\mathit{real}\ n\}\ x \in S$
using *False*
by (*metis add1-zle-eq atLeastAtMost-iff atLeast-iff floor-less-cancel floor-of-nat inInt indicator-simps(1) indicator-simps(2) less-eq-real-def nat-le-iff*)
thus *?thesis* **by** *auto*
qed
qed
qed

have *Hlim*: *AE x in lborel*. $(\lambda i.\ ?H\ i\ x) \longrightarrow ?f(\mathit{Suc}\ k)\ x * \mathit{indicator}\ \{0\ ..\}\ x$
by(*intro AE-I2 tendsto-intros Intervallim*)

{
fix k :: *nat* **and** x :: *real*
assume *posx*: $0 \leq x$
have *1*: $\{0 :: \mathit{nat}..k\} = \{0..<(\mathit{Suc}\ k)\}$
by *auto*
have *ineq1*: $\mathit{inverse}(\mathit{fact}\ k) *_R ((x\ /\ l)\ \hat{}\ k) \leq (\sum n< (\mathit{Suc}\ k).\ \mathit{inverse}(\mathit{fact}\ n) *_R ((x\ /\ l)\ \hat{}\ n))$
by(*rule sum-nonneg-leq-bound[of{0..k}],auto simp: lessThan-atLeast0 posx 1*)
have *ineq2*: $0 \leq (\sum n.\ \mathit{inverse}(\mathit{fact}\ (n + (\mathit{Suc}\ k))) *_R ((x\ /\ l)\ \hat{}\ (n + (\mathit{Suc}\ k))))$
proof(*rule suminf-nonneg*)
show *summable* $(\lambda n.\ (x\ /\ l)\ \hat{}\ (n + (\mathit{Suc}\ k))\ /_R\ \mathit{fact}\ (n + (\mathit{Suc}\ k)))$
by(*subst summable-iff-shift, rule summable-exp-generic*)
show $\bigwedge n.\ 0 \leq (x\ /\ l)\ \hat{}\ (n + (\mathit{Suc}\ k))\ /_R\ \mathit{fact}\ (n + (\mathit{Suc}\ k))$
by (*auto simp*: *posx*)
qed
from *ineq1 ineq2* **have** $\mathit{inverse}(\mathit{fact}\ k) *_R ((x\ /\ l)\ \hat{}k) \leq (\sum n<(\mathit{Suc}\ k).\ \mathit{inverse}(\mathit{fact}\ n) *_R ((x\ /\ l)\ \hat{}\ n)) + (\sum n.\ \mathit{inverse}(\mathit{fact}\ (n +(\mathit{Suc}\ k))) *_R ((x\ /\ l)\ \hat{}(n + (\mathit{Suc}\ k))))$

by *linarith*
also have ... = *exp*(*x* / *l*)
using *exp-first-terms*[*ofx* / *l* (*Suc k*)] **by** *auto*
finally have *inverse*(*fact k*) ∗ ((*x* / *l*) ^ *k*) ≤ *exp*(*x* / *l*) **by** *auto*
hence (*x* / *l*) ^*k* / *fact k* ≤ *exp*(*x* / *l*)
by(*auto simp*: *field-simps*)
hence *ineq3*: (*x* / *l*) ^*k* ≤ *exp*(*x* / *l*) ∗ *fact k*
by(*auto simp*: *pos-divide-le-eq*)
have (*x* / *l*) ^*k* ∗ *exp*(−*x* / *l*) = (*x* / *l*) ^*k* / *exp*(*x* / *l*)
by (*auto simp*: *exp-minus field-class.field-divide-inverse*)
also have ... ≤ *exp*(*x* / *l*) ∗ *fact k* / *exp*(*x* / *l*)
using *ineq3* **by** (*metis divide-right-mono exp-ge-zero*)
finally have (*x* / *l*) ^ *k* ∗ *exp*(−*x* / *l*) ≤ *fact k***by** *auto*
} **note** *expineq* = *this*

have *Hilim01*: (λ*i* :: *nat*. (*1*/(*2* ∗ *l*)) ∗(*?AF i* ∗ *?AG i* − *?AF 0* ∗ *?AG 0*) + *l* ∗ (*Suc k*) ∗ ∫ *x*. (*?f k x*) ∗ *indicator* {*0* .. *i*} *x* ∂*lborel*) ⟶ (*1*/(*2* ∗ *l*)) ∗(*0* − *?AF 0* ∗ *?AG 0*) + *l* ∗ (*Suc k*) ∗ ∫ *x*. (*?f k x*) ∗ *indicator* {*0* ..} *x* ∂*lborel*
proof(*intro tendsto-intros integralfSuck*)
{
fix *k* :: *nat*
have (λ*x*. (*real x* / *l*) ^ *k* ∗ (*exp* (− *real x* / *l*))) ⟶ *0*
proof(*rule tendstoI*, *subst eventually-at-top-linorder*,*rule exI*)
fix *e* :: *real* **assume** *pose*: *0* < *e*
show ∀ *n* ≥ (λ *e* :: *real*. *nat* ⌊*l* ∗ *fact* (*Suc k*) / *e*⌋ + *1*) *e*. *dist* ((*real n* / *l*) ^ *k* ∗ *exp* (− *real n* / *l*)) *0* < *e*
proof(*intro allI impI*)
fix *n* :: *nat*
assume *assmsn*: *nat* ⌊*l* ∗ *fact* (*Suc k*) / *e*⌋ + *1* ≤ *n*
hence *npos*: *0* < *real n***by** *auto*
have ((*real n* / *l*) ∗ ((*real n* / *l*) ^ *k*) ∗ *exp*(− (*real n*) / *l*)) ≤ *fact* (*Suc k*)
using *expineq*[*ofreal n* (*Suc k*)] **by** *auto*
hence *ineq1*: ((*real n* / *l*) ^ *k* ∗ *exp*(− (*real n*) / *l*)) ≤ *fact* (*Suc k*) / (*real n* / *l*)
using *pos npos le-divide-eq mult.commute divide-pos-pos*
by (*metis vector-space-over-itself.scale-scale*)
also have ... ≤ *fact* (*Suc k*) / (*real* (*nat* ⌊*l* ∗ *fact* (*Suc k*) / *e*⌋ + *1*) / *l*)
using *ineq1 assmsn pos* **by** (*auto simp*: *frac-le*)
also have ... < *fact* (*Suc k*) / (*fact* (*Suc k*) / *e*)
proof(*rule divide-strict-left-mono*)
show (*0* :: *real*) < *fact* (*Suc k*)**by** *auto*
show (*0* :: *real*) < *real* (*nat* ⌊*l* ∗ *fact* (*Suc k*) / *e*⌋ + *1*) / *l* ∗ (*fact* (*Suc k*) / *e*)
by (*auto simp*: *pos pose*)
have *fact* (*Suc k*) / *e* ≤ (*l* ∗ *fact* (*Suc k*) / *e*) / *l*
by *auto*

also have ... $<$ *real* (*nat* $\lfloor l * fact\ (Suc\ k)\ /\ e\rfloor + 1)\ /\ l$
proof –
have $l * fact\ (Suc\ k)\ /\ e <$ *real* (*nat* $\lfloor l * fact\ (Suc\ k)\ /\ e\rfloor + 1)$
by *linarith*
thus *?thesis*
using *divide-strict-right-mono pos* **by** *blast*
qed
finally show *fact* (*Suc k*) / $e <$ *real* (*nat* $\lfloor l * fact\ (Suc\ k)\ /\ e\rfloor + 1)\ /\ l$**.**
qed
also have ... $= e$
by *auto*
finally have *ineq3*: $((real\ n\ /\ l)\ \hat{}\ k * exp(-\ real\ n\ /\ l)) < e$**.**
thus*dist* $((real\ n\ /\ l)\ \hat{}\ k * exp\ (-\ real\ n\ /\ l))\ 0 < e$
by *auto*
qed
qed
} **note** *limit1* = *this*

have *eqlim*: $(\lambda x.\ real\ x\ \hat{}\ Suc\ k * (-\ l * exp\ (-\ real\ x\ /\ l))) = (\lambda x.\ ((l\hat{}Suc\ k) * (-\ l)) * (((real\ x\ /\ l)\ \hat{}\ Suc\ k) * exp\ (-\ real\ x\ /\ l)))$
by (*auto simp add*: *power-divide*)
have $(\lambda x.\ ((l\hat{}Suc\ k) * (-\ l)) * (((real\ x\ /\ l)\ \hat{}\ Suc\ k) * exp\ (-\ real\ x\ /\ l))) \longrightarrow (((l\hat{}Suc\ k) * (-\ l)) * 0)$
proof(*intro tendsto-intros*)
show $(\lambda x.\ (real\ x\ /\ l)\ \hat{}\ Suc\ k * exp\ (-\ real\ x\ /\ l)) \longrightarrow 0$
using *limit1*[*ofSuc k*] **by** *auto*
qed
with *eqlim* **show** $(\lambda x.\ real\ x\ \hat{}\ Suc\ k * (-\ l * exp\ (-\ real\ x\ /\ l))) \longrightarrow 0$**by** *auto*

show $(\lambda x.\ LBINT\ xa.\ 1\ /\ (2 * l) * (exp\ (-\ xa\ /\ l) * xa\ \hat{}\ k) * indicat\text{-}real\ \{0..real\ x\}\ xa) \longrightarrow (LBINT\ x.\ 1\ /\ (2 * l) * (exp\ (-\ x\ /\ l) * x\ \hat{}\ k) * indicat\text{-}real\ \{0..\}\ x)$
using *kconverge3* **by** *auto*
qed
have $(1/(2 * l)) * (0 -\ ?AF\ 0 * ?AG\ 0) + l * (Suc\ k) * (\int x.\ (?f\ k\ x) * indicator\ \{0\ ..\}\ x\ \partial lborel) = (1/(2 * l)) * (0 -\ ?AF\ 0 * ?AG\ 0) + l * (Suc\ k) * (\int x.\ indicator\ \{0\ ..\}\ x *_R (?f\ k\ x)\ \partial lborel)$
proof –
have $\forall r.\ 1\ /\ (2 * l) * (exp\ (-\ r\ /\ l) * r\ \hat{}\ k) * indicat\text{-}real\ \{0..\}\ r = indicat\text{-}real\ \{0..\}\ r * (1\ /\ (2 * l) * (exp\ (-\ r\ /\ l) * r\ \hat{}\ k))$
by *auto*
thus *?thesis*
using *real-scaleR-def* **by** *presburger*
qed
also have ... $=(1/(2 * l)) * (0 -\ ?AF\ 0 * ?AG\ 0) + l * (Suc\ k) * (fact\ k * l\ \hat{}\ k\ /\ 2)$

by (*metis A02*)
also have ... = $l * (Suc\ k) * (fact\ k * l \,\hat{}\, k \,/\, 2)$
by *auto*
also have ... = $(Suc\ k) * (fact\ k) * l * l \,\hat{}\, k \,/\, 2$
by(*auto simp*: *field-simps*)
also have ... = $fact\ (Suc\ k) * l \,\hat{}\, Suc\ k \,/\, 2$
by(*auto simp*: *field-simps*)
finally have *Hilim02*: $(1/(2 * l)) * (0 - ?AF\ 0 * ?AG\ 0) + l * (Suc\ k) * (\int x.\ (?f\ k\ x) * indicator\ \{0\ ..\}\ x\ \partial lborel) = (fact\ (Suc\ k)) * l \,\hat{}\,(Suc\ k) \,/\, 2$.

have *Hilim*: $(\lambda i :: nat.\ \int x.\ ?H\ i\ x\ \partial lborel) \longrightarrow (fact\ (Suc\ k)) * l \,\hat{}\,(Suc\ k) \,/\, 2$
proof−
have $(\lambda i :: nat.\ \int x.\ ?H\ i\ x\ \partial lborel) = (\lambda i :: nat.\ (1/(2 * l)) *(?AF\ i * ?AG\ i - ?AF\ 0 * ?AG\ 0) + l * (Suc\ k) * \int x.\ (?f\ k\ x) * indicator\ \{0\ ..\ i\}\ x\ \partial lborel)$
using *integralfSuck* **by** *auto*
thus *?thesis*
using *Hilim01 Hilim02* **by** *metis*
qed

have *Hu*: $(\lambda\ x.\ ?f(Suc\ k)\ x * indicator\ \{0\ ..\}\ x) \in borel\text{-}measurable\ lborel$
by *measurable*

thus *cond3*: *has-bochner-integral lborel* $(\lambda x.\ indicat\text{-}real\ \{0..\}\ x *_R (laplace\text{-}density\ l\ 0\ x * x \,\hat{}\, Suc\ k))\ (fact\ (Suc\ k) * l \,\hat{}\, Suc\ k \,/\, 2)$
proof(*subst has-bochner-integral-cong*)
show *boch2*: *has-bochner-integral lborel* $(\lambda x.\ ?f(Suc\ k)\ x * indicator\ \{0\ ..\}\ x)\ (fact\ (Suc\ k) * l \,\hat{}\, Suc\ k \,/\, 2)$
by(*rule has-bochner-integral-monotone-convergence*[*OF Hi Hmono Hlim Hilim Hu*])
show *lborel* = *lborel***by** *auto*
fix $x :: real$ **assume** $x \in space\ lborel$
show $indicat\text{-}real\ \{0..\}\ x *_R (laplace\text{-}density\ l\ 0\ x * x \,\hat{}\, Suc\ k) = 1 \,/\, (2 * l) * (exp\ (- x \,/\, l) * x \,\hat{}\, Suc\ k) * indicat\text{-}real\ \{0..\}\ x$
unfolding *laplace-density-def* **by**(*auto simp*: *field-simps split*: *split-indicator*)
next
show $fact\ (Suc\ k) * l \,\hat{}\, Suc\ k \,/\, 2 = fact\ (Suc\ k) * l \,\hat{}\, Suc\ k \,/\, 2$**by** *auto*
qed

show $(\lambda x.\ LBINT\ xa.\ indicat\text{-}real\ \{0..real\ x\}\ xa *_R (laplace\text{-}density\ l\ 0\ xa * xa \,\hat{}\, Suc\ k)) \longrightarrow (LBINT\ y.\ indicat\text{-}real\ \{0..\}\ y *_R (laplace\text{-}density\ l\ 0\ y * y \,\hat{}\, Suc\ k))$
proof−
have *eq1*: $(\lambda i :: nat.\ \int x.\ ?H\ i\ x\ \partial lborel) = (\lambda x.\ LBINT\ xa.$

indicat-real {*0..real x*} *xa* ∗ (*laplace-density l 0 xa* ∗ (*xa* ∗ *xa* ^ *k*)))
proof fix *i* :: *nat* **show** (*LBINT x. 1* / (*2* ∗ *l*) ∗ (*exp* (− *x* / *l*) ∗ *x* ^ *Suc k*) ∗ *indicat-real* {*0..real i*} *x*) = (*LBINT xa. indicat-real* {*0..real i*} *xa* ∗ (*laplace-density l 0 xa* ∗ (*xa* ∗ *xa* ^ *k*)))
by(*rule Bochner-Integration.integral-cong*,*unfold laplace-density-def*, *auto split*: *split-indicator*)
qed
have (*LBINT y. indicat-real* {*0..*} *y* ∗ (*laplace-density l 0 y* ∗ (*y* ^ (*Suc k*)))) = ((*1* + *real k*) ∗ *fact k* ∗ (*l* ∗ *l* ^ *k*) / *2*)
using *cond3 has-bochner-integral-integral-eq* **by** *force*
also have ... = (*fact* (*Suc k*)) ∗ *l* ^ (*Suc k*) / *2*
by *auto*
finally have *eq2*: (*LBINT y. indicat-real* {*0..*} *y* ∗ (*laplace-density l 0 y* ∗ (*y* ^ (*Suc k*)))) = (*fact* (*Suc k*)) ∗ *l* ^ (*Suc k*) / *2*.

from *Hilim eq2*[*THEN sym*] *eq1* **show** (λ*x. LBINT xa. indicat-real* {*0..real x*} *xa* $*_R$ (*laplace-density l 0 xa* ∗ *xa* ^ *Suc k*)) ⟶ (*LBINT y. indicat-real* {*0..*} *y* $*_R$ (*laplace-density l 0 y* ∗ *y* ^ *Suc k*))
by *auto*
qed
qed

lemma *laplace-moment-even*:
fixes *k* :: *nat*
assumes *pos*[*arith*]: *0* < *l*
shows *has-bochner-integral lborel* (λ *x.* ((*laplace-density l m x*) ∗ (*x* − *m*)^(*2* ∗ *k*)))(*fact* (*2* ∗ *k*) ∗ *l*^(*2* ∗ *k*))
proof−
have *has-bochner-integral lborel* (λ *x.* ((*laplace-density l 0 x*) ∗ *x*^(*2* ∗ *k*)))(*2* $*_R$ (*fact* (*2* ∗ *k*) ∗ *l*^(*2* ∗ *k*) /*2*))
proof(*rule has-bochner-integral-even-function*)
show *has-bochner-integral lborel* (λ*x. indicat-real* {*0..*} *x* $*_R$ (*laplace-density l 0 x* ∗ *x* ^ (*2* ∗ *k*))) (*fact* (*2* ∗ *k*) ∗ *l* ^ (*2* ∗ *k*) / *2*)
using *laplace-moment-0*(*1*)[*of l*(*2* ∗ *k*)] *pos* **by** *auto*
show ⋀*x. laplace-density l 0* (− *x*) ∗ (− *x*) ^ (*2* ∗ *k*) = *laplace-density l 0 x* ∗ *x* ^ (*2* ∗ *k*)
unfolding *laplace-density-def* **by** *auto*
qed
hence *has-bochner-integral lborel* (λ *x.* ((*laplace-density l 0* (*x* − *m*)) ∗ (*x* − *m*)^(*2* ∗ *k*)))(*2* $*_R$ (*fact* (*2* ∗ *k*) ∗ *l*^(*2* ∗ *k*) /*2*))
by(*subst lborel-has-bochner-integral-real-affine-iff*[**where** *c* =*1* :: *real* **and** *t* =*m*],*auto*)
thus *?thesis*
using *laplace-density-shift*[*of l m*−*m*,*simplified*] **by** *auto*
qed

lemma *laplace-moment-odd*:
fixes *k* :: *nat*
assumes *pos*[*arith*]: *0* < *l*

shows *has-bochner-integral lborel* (λ *x*. ((*laplace-density l m x*) $*$ (*x* $-$ *m*)^(*2* $*$ *k* + *1*)))(*0*)
proof−
have *has-bochner-integral lborel* (λ *x*. ((*laplace-density l 0 x*) $*$ *x*^(*2* $*$ *k* + *1*)))(*0*)
proof(*rule has-bochner-integral-odd-function*)
show *has-bochner-integral lborel* (λ*x*. *indicat-real* {*0*..} *x* $*_R$ (*laplace-density l 0 x* $*$ *x* ^ (*2* $*$ *k* + *1*))) (*fact* (*2* $*$ *k* + *1*) $*$ *l* ^ (*2* $*$ *k* + *1*) / *2*)
using *laplace-moment-0*(*1*)[*of l*(*2* $*$ *k* + *1*)] *pos* **by** *auto*
show $\bigwedge$*x*. *laplace-density l 0* ($-$ *x*) $*$ ($-$ *x*) ^ (*2* $*$ *k* + *1*) = $-$ (*laplace-density l 0 x* $*$ *x* ^ (*2* $*$ *k* + *1*))
unfolding *laplace-density-def* **by** *auto*
qed
hence *has-bochner-integral lborel* (λ *x*. ((*laplace-density l 0* (*x* $-$ *m*)) $*$ (*x* $-$ *m*)^(*2* $*$ *k* + *1*)))(*0*)
by(*subst lborel-has-bochner-integral-real-affine-iff* [**where** *c* =*1* :: *real* **and** *t* =*m*],*auto*)
thus *?thesis*
using *laplace-density-shift*[*of l m*−*m*,*simplified*] **by** *auto*
qed

lemma *laplace-moment-abs-odd*:
fixes *k* :: *nat*
assumes *pos*[*arith*]: *0* < *l*
shows *has-bochner-integral lborel* (λ *x*. ((*laplace-density l m x*) $*$ |*x* $-$ *m*|^(*2* $*$ *k* + *1*)))(*fact* (*2* $*$ *k* + *1*) $*$ *l* ^ (*2* $*$ *k* + *1*))
proof−
have *has-bochner-integral lborel* (λ *x*. ((*laplace-density l 0 x*) $*$ |*x*|^(*2* $*$ *k* + *1*)))(*2* $*_R$ (*fact* (*2* $*$ *k* + *1*) $*$ *l*^(*2* $*$ *k* + *1*) /*2*))
proof(*rule has-bochner-integral-even-function*)
have *has-bochner-integral lborel* (λ*x*. *indicat-real* {*0*..} *x* $*_R$ (*laplace-density l 0 x* $*$ *x* ^ (*2* $*$ *k* + *1*)))(*fact* (*2* $*$ *k* + *1*) $*$ *l* ^ (*2* $*$ *k* + *1*) / *2*)
using *laplace-moment-0*(*1*)[*of l*(*2* $*$ *k* + *1*)] *pos* **by** *auto*
thus*has-bochner-integral lborel* (λ*x*. *indicat-real* {*0*..} *x* $*_R$ (*laplace-density l 0 x* $*$ |*x*| ^ (*2* $*$ *k* + *1*)))(*fact* (*2* $*$ *k* + *1*) $*$ *l* ^ (*2* $*$ *k* + *1*) / *2*)
by(*subst has-bochner-integral-cong*,*auto*)
show $\bigwedge$*x*. *laplace-density l 0* ($-$ *x*) $*$ |$-$ *x*| ^ (*2* $*$ *k* + *1*) = *laplace-density l 0 x* $*$ |*x*| ^ (*2* $*$ *k* + *1*)
unfolding *laplace-density-def* **by** *auto*
qed
hence *has-bochner-integral lborel* (λ *x*. ((*laplace-density l 0* (*x* $-$ *m*)) $*$ |*x* $-$ *m*|^(*2* $*$ *k* + *1*)))(*2* $*_R$ (*fact* (*2* $*$ *k* + *1*) $*$ *l*^(*2* $*$ *k* + *1*) /*2*))
by(*subst lborel-has-bochner-integral-real-affine-iff* [**where** *c* =*1* :: *real* **and** *t* =*m*],*auto*)
thus *?thesis*
using *laplace-density-shift*[*of l m*−*m*,*simplified*] **by** *auto*
qed

lemma *integral-laplace-moment-even*:

assumes *pos*[*arith*]: $0 < l$
shows $integral^L$ *lborel* ($\lambda x.$(*laplace-density l m x*) $*$ $(x - m)$^($2 *$ k)) = *fact* ($2 * k$) $*$ l^($2 * k$)
using *laplace-moment-even*[*of l m k*] *pos* **by** (*auto simp*: *has-bochner-integral-integral-eq*)

lemma *integral-laplace-moment-odd*:
assumes *pos*[*arith*]: $0 < l$
shows $integral^L$ *lborel* ($\lambda x.$(*laplace-density l m x*) $*$ $(x - m)$^($2 * k$ $+ 1$)) = 0
using *laplace-moment-odd*[*of l m k*] *pos* **by** (*auto simp*: *has-bochner-integral-integral-eq*)

lemma *integral-laplace-moment-abs-odd*:
assumes *pos*[*arith*]: $0 < l$
shows $integral^L$ *lborel* ($\lambda x.$(*laplace-density l m x*) $*$ $|x - m|$^($2 * k$ $+ 1$)) =*fact* ($2 * k + 1$) $*$ l^($2 * k + 1$)
using *laplace-moment-abs-odd*[*of l m k*] *pos* **by** (*auto simp*: *has-bochner-integral-integral-eq*)

lemma *integrable-laplace-moment*:
fixes k :: *nat*
assumes *pos*[*arith*]: $0 < l$
shows *integrable lborel* (λ *x*. ((*laplace-density l m x*) $*$ $(x - m)$^(k)))
proof *cases*
assume *even k***thus** *?thesis*
using *integrable.intros*[*OF laplace-moment-even*] **by** (*auto elim*: *evenE*)
next
assume *odd k***thus** *?thesis*
using *integrable.intros*[*OF laplace-moment-odd*] **by** (*auto elim*: *oddE*)
qed

lemma *integrable-laplace-moment-abs*:
fixes k :: *nat*
assumes *pos*[*arith*]: $0 < l$
shows *integrable lborel* (λ *x*. ((*laplace-density l m x*) $*$ $| x - m |$^(k)))
proof *cases*
assume *even k***thus** *?thesis*
using *integrable.intros*[*OF laplace-moment-even*] **by** (*auto simp add*: *power-even-abs elim*: *evenE*)
next
assume *odd k***thus** *?thesis*
using *integrable.intros*[*OF laplace-moment-abs-odd*] **by** (*auto elim*: *oddE*)
qed

lemma *integrable-laplace-density*[*simp*, *intro*]:
assumes *pos*[*arith*]: $0 < l$
shows *integrable lborel* (*laplace-density l m*)

using *integrable-laplace-moment*[*of l m 0*,*OF pos*] **by** *auto*

lemma *integral-laplace-density*[*simp*, *intro*]:
assumes *pos*[*arith*]: $0 < l$
shows ($\int$ *x. laplace-density l m x* ∂*lborel*) = *1*
using *integral-laplace-moment-even*[*of l m 0*,*OF pos*] **by** *auto*

3.3 The Probability Space Distributed by Laplace Distribution

lemma *prob-space-laplacian-density*:
assumes *pos*[*arith*]: $0 < l$
shows *prob-space* (*density lborel* (*laplace-density l m*))
proof
show *emeasure* (*density lborel* (*laplace-density l m*)) (*space* (*density lborel* (*laplace-density l m*))) = *1*
by(*auto simp*:*emeasure-laplace-density-mass-1*[*OF pos*])
qed

lemma (**in** *prob-space*) *laplace-distributed-le*:
assumes *D*: *distributed M lborel X* (*laplace-density l m*)
and [*simp*, *arith*]: $0 < l$
shows $\mathcal{P}$(*x in M. X x* $\leq$ *a*) = *laplace-CDF l m a*
proof−
have *emeasure M* {*x* $\in$ *space M. X x* $\leq$ *a*} = *emeasure* (*distr M lborel X*) {.. *a*}
using *distributed-measurable*[*OF D*]
by (*subst emeasure-distr*) (*auto intro*!: *arg-cong2*[**where** *f*=*emeasure*])
also have ... = *emeasure* (*density lborel* (*laplace-density l m*)) {.. *a*}
unfolding *distributed-distr-eq-density*[*OF D*] **by** *auto*
also have ... = *laplace-CDF l m a*
using *emeasure-laplace-density assms* **by** *auto*
finally show *?thesis*
by (*auto simp*: *emeasure-eq-measure*)
qed

lemma (**in** *prob-space*) *laplace-distributed-gt*:
assumes *D*: *distributed M lborel X* (*laplace-density l m*)
and [*simp*, *arith*]: $0 < l$
shows $\mathcal{P}$(*x in M. X x* > *a*) = *1* − *laplace-CDF l m a*
proof−
have *1* − *laplace-CDF l m a* = *1*− $\mathcal{P}$(*x in M. X x* $\leq$ *a*)
using *laplace-distributed-le assms* **by** *auto*
also have ... = *prob* (*space M* − {*x* $\in$ *space M. X x* $\leq$ *a*})
using *distributed-measurable*[*OF D*]
by (*auto simp*: *prob-compl*)
also have ... = $\mathcal{P}$(*x in M. X x* > *a*)
by (*auto intro*!: *arg-cong*[**where** *f*=*prob*] *simp*: *not-le*)
finally show *?thesis*

```
  by auto
qed

end


theory Comparable-Probability-Measures
  importsHOL−Probability.Probability
    Additional-Lemmas-for-Integrals
begin
```

4 Comparable Pairs of Probability Measures

To compare two probability measures M and N on the same space, we introduce their Radon-Nikodym derivatives (i.e. density functions) with respect to the sum measure M + N. We could consider various base measures, but we choose the sum measure, because it is finite; it is easy to check the absolute continuity M, N « M + N; the Radon-Nikodym derivatives dM and dN are bounded and are essentially partition of unity(i.e. dM + dN = 1 a.e.).

4.1 Sum of two measures

```
definition sum-measure :: 'a measure ⇒ 'a measure ⇒ 'a measure
where
  sum-measure M N = measure-of (space M) (sets M) (λ A. emeasure
M A + emeasure N A)

lemma sum-measure-space[simp, measurable]:
  shows space (sum-measure M N) = space M
    and sets (sum-measure M N) = sets M
  by (auto simp: sum-measure-def)

lemma sum-measure-commutativitiy:
  assumes finite-measure M
    and finite-measure N
    and space M = space N
    and sets M = sets N
  shows sum-measure N M = sum-measure M N
  by (auto simp: sum-measure-def ac-simps realrel-def assms)

lemma sum-measure-space2:
  assumes space M = space N
    and sets M = sets N
```

shows *space* (*sum-measure M N*) = *space N*
and *sets* (*sum-measure M N*) = *sets N*
by (*auto simp*: *assms*)

This lemma is inspired from $[\![$*finite-measure ?M*; *finite-measure ?N*; *sets ?M* = *sets ?N*; $\bigwedge$*A*. *A* ∈ *sets ?M* ⟹ *emeasure ?N A* ≤ *emeasure ?M A*; *?A* ∈ *sets ?M*$]\!]$ ⟹ *emeasure* (*diff-measure ?M ?N*) *?A* = *emeasure ?M ?A* − *emeasure ?N ?A* in the standard library.

lemma *emeasure-sum-measure* [*simp*]:
assumes *fin*: *finite-measure M*
and *finite-measure N*
and *sets-eq*: *sets M* = *sets N*
and *A*: *A* ∈ *sets M*
shows *emeasure* (*sum-measure M N*) *A* = *emeasure M A* + *emeasure N A*
(**is**- = *?μ A*)
proof(*unfold sum-measure-def*, *rule emeasure-measure-of-sigma*)
show *sigma-algebra* (*space M*) (*sets M*)
by (*auto simp*: *sets.sigma-algebra-axioms*)
show *positive* (*sets M*) *?μ*
by (*auto simp*: *positive-def*)
show *countably-additive* (*sets M*) *?μ*
proof(*rule countably-additiveI*)
fix *A* :: *nat* ⇒ -
assume *A*: *range A* ⊆ *sets M* **and** *disjoint-family A*
hence *suminf*:
($\sum$*i*. *emeasure M* (*A i*)) = *emeasure M* ($\bigcup$*i*. *A i*)
($\sum$*i*. *emeasure N* (*A i*)) = *emeasure N* ($\bigcup$*i*. *A i*)
by (*simp-all add*: *suminf-emeasure sets-eq*)
with *A* **have** ($\sum$*i*. *emeasure M* (*A i*) + *emeasure N* (*A i*)) = ($\sum$*i*. *emeasure M* (*A i*)) + ($\sum$*i*. *emeasure N* (*A i*))
using *fin suminf-add summableI* **by**(*metis* (*full-types*))
thus($\sum$*i* :: *nat*. *emeasure M* (*A i*) + *emeasure N* (*A i*)) = *emeasure M* ($\bigcup$ (*range A*)) + *emeasure N* ($\bigcup$ (*range A*))
by (*auto simp*: *suminf*(*1*) *suminf*(*2*))
qed
show *A* ∈ *sets M* **by** (*auto simp*: *A*)
qed

lemma *sum-measure-finiteness* [*simp*]:
assumes *finite-measure M*
and *finite-measure N*
and *space M* = *space N*
and *sets M* = *sets N*
shows *finite-measure* (*sum-measure M N*)
proof(*rule finite-measureI*)
have *emeasure* (*sum-measure M N*) (*space* (*sum-measure M N*)) = *emeasure M* (*space M*) + *emeasure N* (*space N*)

by (*auto simp*: *assms*)
thus *emeasure* (*sum-measure M N*) (*space* (*sum-measure M N*)) $\neq$ ∞
using *assms*(*1*,*2*) *finite-measure.finite-emeasure-space* **by** *auto*
qed

lemma *absolute-continuity-sum-measure* [*simp*]:
assumes *finite-measure M*
and *finite-measure N*
and *space M* = *space N*
and *sets M* = *sets N*
shows *absolutely-continuous* (*sum-measure M N*) *M*
unfolding *absolutely-continuous-def null-sets-def*
proof(*intro subsetI allI impI CollectI*)
fix *x* **assume** $x \in \{x2 \in sets\ (sum\text{-}measure\ M\ N).\ emeasure\ (sum\text{-}measure\ M\ N)\ x2 = 0\}$
hence *A*: $x \in sets\ M \wedge emeasure\ (sum\text{-}measure\ M\ N)\ x = 0$ **by** *auto*
have *A1*: (*sum-measure M N*) *x* = *emeasure M x* + *emeasure N x*
using *A assms*(*1*) *assms*(*2*) *assms*(*4*) *emeasure-sum-measure* **by** *blast*
have *A2*: *emeasure M x* $\leq$ *emeasure M x* + *emeasure N x* **by** *auto*
with *A A1* **have** *A3*: *emeasure M x* + *emeasure N x* = *0* **by** *auto*
thus $x \in sets\ M \wedge emeasure\ M\ x = 0$ **using** *A* **by** *auto*
qed

lemma *absolute-continuity-sum-measure2*[*simp*]:
assumes *finite-measure M*
and *finite-measure N*
and *space M* = *space N*
and *sets M* = *sets N*
shows *absolutely-continuous* (*sum-measure N M*) *M*
by (*auto simp*: *assms sum-measure-commutativitiy*)

lemma *density-sum-measure*:
assumes *finite-measure M*
and *finite-measure N*
and *space M* = *space N*
and *sets M* = *sets N*
shows *density* (*sum-measure M N*) (*RN-deriv* (*sum-measure M N*) *M*) = *M*
proof−
have *P1*: *finite-measure* (*sum-measure M N*)
by (*auto simp*: *assms*)
have *absolutely-continuous* (*sum-measure M N*) *M*
by (*auto simp*: *assms*)
thus *?thesis*
by (*auto simp*: *P1 finite-measure.sigma-finite-measure sigma-finite-measure.density-RN-deriv sum-measure-space*(*2*)[*of M N*])
qed

lemma *density-sum-measure2*:
assumes *finite-measure M*
and *finite-measure N*
and *space M = space N*
and *sets M = sets N*
shows *density* (*sum-measure N M*) (*RN-deriv* (*sum-measure N M*) *M*) = *M*
by (*auto simp*: *density-sum-measure assms sum-measure-space*(*2*)[*of M N*] *sum-measure-commutativitiy*[*of M N*])

lemma *absolutely-continuous-emeasure-less*[*simp*]:
assumes *sets M = sets N*
and ∀ *A* ∈ *sets M*. *emeasure M A* ≤ *emeasure N A*
shows *absolutely-continuous N M*
unfolding *absolutely-continuous-def null-sets-def*
proof(*rule subsetI*)
fix *A* **assume** *A* ∈ {*A* ∈ *sets N*. *emeasure N A* = *0*}
hence *A* ∈ *sets N* **and** *emeasure N A* = *0* **and** *A* ∈ *sets M* **and** *emeasure M A* ≤ *emeasure N A*
using *assms* **by** *auto*
moreover hence *emeasure M A* = *0*
by *auto*
ultimately show *A* ∈ {*A* ∈ *sets M*. *emeasure M A* = *0*}
by *auto*
qed

lemma *emeasure-less-RN-deriv-bound-1*[*simp*]:
assumes *finite-measure M*
and *finite-measure N*
and *space M = space N*
and *sets M = sets N*
and ∀ *A* ∈ *sets M*. *emeasure M A* ≤ *emeasure N A*
shows *AE x in N* . *RN-deriv N M x* ≤ *1*
proof(*rule AE-upper-bound-inf-ennreal*)
fix *e* :: *real* **assume** *pose*: *0* < *e*
have *abs*: *absolutely-continuous N M*
using *absolutely-continuous-emeasure-less assms* **by** *blast*
show *AE x in N*. *RN-deriv N M x* ≤ *1* + *ennreal e*
proof(*rule AE-I*)
let *?C* = {*x* ∈ *space N*. ¬ *RN-deriv N M x* ≤ *1* + *ennreal e*}
show *measurableA*: *?C* ∈ *N*
by *measurable*
show *?C* ⊆ *?C*
by *auto*
have (*1* + *ennreal e*) ∗ *emeasure N ?C* = (*1* + *ennreal e*) ∗ ∫ + *x*. *indicator ?C x* ∂*N*
by *auto*
also have ... = ∫ + *x*. (*1* + *ennreal e*)∗ *indicator ?C x* ∂*N*

by (*metis* (*mono-tags*, *lifting*) *calculation measurableA nn-integral-cmult-indicator nn-integral-cong*)
also have ... $\leq \int^{+} x.\ RN\text{-}deriv\ N\ M\ x * indicator\ ?C\ x\ \partial N$
proof(*rule nn-integral-mono*,*cases*)
fix x **assume** $x \in space\ N$ **assume** A: $x \in ?C$
thus($1 + ennreal\ e) * indicator\ ?C\ x \leq RN\text{-}deriv\ N\ M\ x * indicator\ ?C\ x$
using A **by** *force*
next
fix x **assume** $x \in space\ N$ **assume** A: $x \notin ?C$
thus($1 + ennreal\ e) * indicator\ ?C\ x \leq RN\text{-}deriv\ N\ M\ x * indicator\ ?C\ x$
using A **by** *force*
qed
also have ... $= \int^{+} x.\ indicator\ ?C\ x\ \partial M$
proof(*rule sigma-finite-measure.RN-deriv-nn-integral*[*THEN sym*])
show *sigma-finite-measure N*
using *assms*(*2*) *finite-measure-def* **by** *auto*
show *absolutely-continuous N M*
by(*intro abs*)
show *sets M = sets N*
by(*intro assms*)
show *indicator* $\{x \in space\ N.\ \neg\ RN\text{-}deriv\ N\ M\ x \leq 1 + ennreal\ e\} \in borel\text{-}measurable\ N$
by *measurable*
qed
also have ... $= emeasure\ M\ ?C$
by (*auto simp*: *assms*(*4*))
also have ... $\leq emeasure\ N\ ?C$
by (*auto simp*: *assms*(*4*) *assms*(*5*))
finally have $(1 + ennreal\ e)*emeasure\ N\ ?C \leq emeasure\ N\ ?C$**.**
hence $emeasure\ N\ ?C + ennreal\ e * emeasure\ N\ ?C \leq emeasure\ N\ ?C$
by (*auto simp*: *comm-semiring-class.distrib*)
hence $emeasure\ N\ ?C - emeasure\ N\ ?C + ennreal\ e * emeasure\ N\ ?C \leq emeasure\ N\ ?C - emeasure\ N\ ?C$
by (*auto simp*: *ennreal-minus-mono*)
hence *ineq2*: $(emeasure\ N\ ?C - emeasure\ N\ ?C) + ennreal\ e * emeasure\ N\ ?C \leq (emeasure\ N\ ?C - emeasure\ N\ ?C)$
by *auto*
have $(emeasure\ N\ ?C - emeasure\ N\ ?C) = 0$
proof(*rule ennreal-diff-self*)
from *assms*(*2*) *finite-measure.emeasure-real*[*of N*]
obtain r :: *real* **where** $emeasure\ N\ ?C = ennreal\ r$**by** *auto*
also have ... $\neq \top$
using *top-neq-ennreal* **by** *auto*
finally show $emeasure\ N\ ?C \neq \top$**.**
qed
from *this ineq2* **have** $ennreal\ e * emeasure\ N\ ?C = 0$

by *auto*
thus*emeasure N ?C = 0*
using *pose* **by** *auto*
qed
qed

lemma *functions-less-upto-AE*[*simp*]:
assumes (*f* :: *-⇒ 'b* :: {*linorder-topology*, *second-countable-topology*}) ∈ *borel-measurable M*
and (*g* :: *-⇒'b*) ∈ *borel-measurable M*
and (*h* :: *-⇒ 'b*) ∈ *borel-measurable M*
and *fgequal*: *AE x in M. f x = g x*
and *AE x in M. g x ≤ h x*
shows *AE x in M. f x ≤ h x*
proof(*rule AE-mp*[*OF fgequal*],*rule AE-I'*)
let *?C* = {*x* ∈ *space M. g x > h x*}
let *?B* = {*x* ∈ *space M. f x ≠ g x*}
have *Anull*: *?C* ∈ *null-sets M*
using *assms* **by** (*subst AE-iff-null-sets*, *auto*)
have *Bnull*: *?B* ∈ *null-sets M*
using *assms* **by** (*subst AE-iff-null-sets*, *auto*)
show *?C* ∪ *?B* ∈ *null-sets M*
using *Anull Bnull* **by** *auto*
qed(*auto*)

lemma *density-sum-measure-bounded*[*simp*]:
assumes *finite-measure M*
and *finite-measure N*
and *space M = space N*
and *sets M = sets N*
shows *AE x in* (*sum-measure M N*) .(*RN-deriv* (*sum-measure M N*) *M*) *x ≤ 1*
by(*rule emeasure-less-RN-deriv-bound-1*,*simp-all add*: *assms*)

lemma *density-sum-measure-bounded2*[*simp*]:
assumes *finite-measure M*
and *finite-measure N*
and *space M = space N*
and *sets M = sets N*
shows *AE x in* (*sum-measure M N*) .(*RN-deriv* (*sum-measure M N*) *N*) *x ≤ 1*
by(*rule emeasure-less-RN-deriv-bound-1*,*simp-all add*: *assms*)

4.2 Miscellaneous additional lemmas on probability measures

lemma *measurable-bind-prob-space-simple*:
assumes[*measurable*]: $f \in L \rightarrow_M$ (*prob-algebra K*)
shows (λ*M.* (*M* ⤜ *f*)) ∈ (*prob-algebra L*)$\rightarrow_M$ (*prob-algebra K*)

by(*rule measurable-bind-prob-space*[**where** $N = L$],*auto*)

lemma
assumes $M \in$ *space* (*prob-algebra L*)
shows *actual-prob-space*: *prob-space M*
and *space-prob-algebra-sets*: *sets M = sets L*
and *space-prob-algebra-space*: *space M = space L*
proof−
show *prob-space M* **and** *1*: *sets M = sets L*
using *assms space-prob-algebra* **by** *auto*
show *space M = space L*
using *sets-eq-imp-space-eq*[*OF 1*] **by** *auto*
qed

lemma *evaluations-probabilistic-process* [*simp*,*intro*]:
assumes *f*: $f \in$ *measurable L* (*prob-algebra K*)
and $A \in$ *sets K*
shows (λ *x*. *measure* (*f x*) *A*) $\in$ *borel-measurable L*
and (λ *x*. *emeasure* (*f x*) *A*) $\in$ *borel-measurable L*
and $\forall$ $x \in$ *space L*. *measure* (*f x*) $A \leq 1$
and $\forall$ $x \in$ *space L*. *measure* (*f x*) $A \geq 0$
and $\forall$ $x \in$ *space L*. *norm*(*measure* (*f x*) *A*) ≤ 1
and $\forall$ $x \in$ *space L*. *emeasure* (*f x*) $A \leq 1$
and (*sets N = sets L*) $\Longrightarrow$ (*finite-measure N*)$\Longrightarrow$ *integrable N* (λ
x. *measure* (*f x*) *A*)
proof−
show *p0*: (λ*x*. *measure* (*f x*) *A*) $\in$ *borel-measurable L*
using *assms* **by** *measurable*
show (λ*x*. *emeasure* (*f x*) *A*) $\in$ *borel-measurable L*
by (*metis assms*(*1*) *assms*(*2*) *measurable-emeasure-kernel measur-able-prob-algebraD*)
show $\forall$ $x \in$ *space L*. *emeasure* (*f x*) $A \leq 1$
proof
fix *x* **assume** *xinL*: $x \in$ *space L*
show *emeasure* (*f x*) $A \leq 1$
using *assms subprob-space.subprob-emeasure-le-1 subprob-space-kernel*[*of f* ,*OF measurable-prob-algebraD*[*of f L K*]] *xinL*
by *blast*
qed
thus *p1*: $\forall$ $x \in$ *space L*. *measure* (*f x*) $A \leq 1$
by (*auto intro*: *f measurable-prob-algebraD subprob-space.subprob-measure-le-1 subprob-space-kernel*)
show *p2*: $\forall$ $x \in$ *space L*. *measure* (*f x*) $A \geq 0$
by *auto*
show *p3*: $\forall$ $x \in$ *space L*. *norm*(*measure* (*f x*) *A*) ≤ 1
using *p1 p2* **by** *auto*
show (*sets N = sets L*) $\Longrightarrow$ (*finite-measure N*)$\Longrightarrow$*integrable N* (λ
x. *measure* (*f x*) *A*)
proof−

assume *a2*: (*sets* N = *sets* L) **and** *a3*: (*finite-measure* N)
show *integrable* N (λx. *measure* (f x) A)
proof(*rule integrableI-bounded*)
show (λx. *measure* (f x) A) $\in$ *borel-measurable* N
using *p0 a2* **by** *measurable*
have ($\int^+ x$. *ennreal* (*norm* (*measure* (f x) A)) ∂N) $\leq$ ($\int^+ x$. *ennreal 1* ∂N)
proof(*intro nn-integral-mono*)
fix x **assume** $x \in$ *space* N**hence** $*$: $x \in$ *space* L
using *a2 sets-eq-imp-space-eq* **by** *auto*
thus*ennreal* (*norm* (*Sigma-Algebra.measure* (f x) A)) $\leq$ *ennreal 1*
using *p3* **by** *auto*
qed
also have ... = *emeasure* N (*space* N)
by *auto*
also have ... < ∞
using *a3 finite-measure.finite-emeasure-space top.not-eq-extremum* **by** *auto*
finally show ($\int^+ x$. *ennreal* (*norm* (*measure* (f x) A)) ∂N) < ∞.
qed
qed
qed

lemma *Giry-strength-bind-return*:
assumes $N \in$ *space* (*prob-algebra* L)
and $x \in$ *space* K
shows *return* K x $\bigotimes_M$ N = N $\gg=$ (λy. *return* (*return* K x $\bigotimes_M$ N) (x, y))
proof−
have *1*: *sigma-finite-measure* (*return* K x)
by (*auto simp*: *assms*(*2*) *prob-space-imp-sigma-finite prob-space-return*)
have *2*: *prob-space* N
using *actual-prob-space assms*(*1*) **by** *auto*
hence *3*: *sigma-finite-measure* N
by (*auto simp*: *prob-space-imp-sigma-finite*)
have *4*: *prob-space* (*return* K x)
by (*auto simp*: *assms*(*2*) *prob-space-imp-sigma-finite prob-space-return*)
have *return* K x $\bigotimes_M$ N = *return* K x $\gg=$ (λxa. N $\gg=$ (λy. *return* (*return* K x $\bigotimes_M$ N) (xa, y)))
by(*rule pair-prob-space.pair-measure-eq-bind*[*of*(*return* K x)N])(*auto simp*: *pair-prob-space-def pair-sigma-finite-def 1 2 3 4*)
also have ... = N $\gg=$ (λy. *return* K x $\gg=$ (λxa. *return* (*return* K x $\bigotimes_M$ N) (xa, y)))
by(*rule pair-prob-space.bind-rotate*[*of* - - - (*return* K x $\bigotimes_M$ N)])(*auto simp*: *pair-prob-space-def pair-sigma-finite-def 1 2 3 4*)
also have ... = N $\gg=$ (λy. *return* (*return* K x $\bigotimes_M$ N) (x, y))
by(*intro bind-cong-All ballI bind-return*[**where** $N = K \bigotimes_M N$],*subst*

return-sets-cong[**where** $N = K \bigotimes_M N$])(*auto simp*: *assms*(*2*))
finally show (*return K x*) $\bigotimes_M$ $N = (N \gg\!= (\lambda y.$ *return* ((*return K x*) $\bigotimes_M$ *N*) (*x*, *y*))).
qed

4.3 intgrability for pointwise multiplication of funcitons

definition *ess-bounded* :: *'a measure* ⇒ (*'a* ⇒ *'b* :: {*banach*, *second-countable-topology*}) ⇒ *bool* **where**
ess-bounded M f ≡ ((*f* ∈ *borel-measurable M*)∧(∃ *r* :: *real*. *AE x in M*. *norm*(*f x*) ≤ *r*))

lemma *integrable-mult-ess-bounded-integrable*[*simp*]:
fixes *f* :: *-*⇒ *'b* :: {*banach*, *second-countable-topology*,*real-normed-algebra*}
assumes *ess-bounded M f*
and *integrable M g*
shows *integrable M* (λ *x*. *g x* ∗ *f x*)
proof−
obtain *r* **where** *normed*: *AE x in M*. *norm*(*f x*) ≤ *r*
using *assms ess-bounded-def* **by** *auto*
have [*measurable*]: *f* ∈ *borel-measurable M*
using *assms*(*1*) *ess-bounded-def* **by** *auto*
have [*measurable*]: *g* ∈ *borel-measurable M*
using *assms*(*2*) **by** *auto*
show *integrable M* (λ *x*. *g x* ∗ *f x*)
proof(*rule Bochner-Integration.integrable-bound*[*of M*λ *x*. *r* $*_R$ *g x* (λ *x*. *g x* ∗ *f x*)])
show *integrable M* (λ*x*. *r* $*_R$ *g x*)
using *assms*(*2*) **by** *auto*
show (λ*x*. *g x* ∗ *f x*) ∈ *borel-measurable M*
using *borel-measurable-times* **by** *auto*
show *AE x in M*. *norm* (*g x* ∗ *f x*) ≤ *norm* (*r* $*_R$ *g x*)
proof(*rule AE-mp*[*OF normed*],*intro AE-I2 impI*)
fix *x* **assume** *x* ∈ *space M* **and** *normed2*: *norm* (*f x*) ≤ *r*
show *norm* (*g x* ∗ *f x*) ≤ *norm* (*r* $*_R$ *g x*)
by (*metis abs-of-nonneg dual-order.trans mult.commute mult-right-mono norm-ge-zero norm-mult-ineq norm-scaleR normed2*)
qed
qed
qed

lemma *integrable-mult-integrable-ess-bounded*[*simp*]:
fixes *f* :: *-*⇒ *'b* :: {*banach*, *second-countable-topology*,*real-normed-algebra*}
assumes *ess-bounded M f* **and** *integrable M g*
shows *integrable M* (λ *x*. *f x* ∗ *g x*)
proof−
obtain *r* **where** *normed*: *AE x in M*. *norm*(*f x*) ≤ *r*
using *assms ess-bounded-def* **by** *auto*

have [*measurable*]: $f \in$ *borel-measurable M*
using *assms*(*1*) *ess-bounded-def* **by** *auto*
have [*measurable*]: $g \in$ *borel-measurable M*
using *assms*(*2*) **by** *auto*
show *integrable M* (λ *x*. *f x* $*$ *g x*)
proof(*rule Bochner-Integration.integrable-bound*[*of M*λ *x*. *r* $*_R$ *g x* (λ *x*. (*f x*) $*$ (*g x*))])
show *integrable M* (λ*x*. *r* $*_R$ *g x*)
using *assms*(*2*) **by** *auto*
show (λ *x*. *f x* $*$ *g x*) $\in$ *borel-measurable M*
using *borel-measurable-times* **by** *auto*
show *AE x in M*. *norm* ((*f x*) $*$ (*g x*)) $\leq$ *norm* (*r* $*_R$ *g x*)
proof(*rule AE-mp*[*OF normed*],*intro AE-I2 impI*)
fix *x* **assume** $x \in$ *space M* **and** *normed2*: *norm* (*f x*) $\leq$ *r*
show *norm* (*f x* $*$ *g x*) $\leq$ *norm* (*r* $*_R$ *g x*)
by (*metis abs-of-nonneg dual-order.trans mult-right-mono norm-ge-zero norm-mult-ineq norm-scaleR normed2*)
qed
qed
qed

lemma *ess-bounded-const-real*[*simp*,*intro*]:
shows *ess-bounded M* (λ *x*. *r* :: *real*)
unfolding *ess-bounded-def* **by** *auto*

lemma
fixes *f g* :: - $\Rightarrow$ *'b* :: {*banach*, *second-countable-topology*, *linorder-topology*}
assumes *ess-bounded M f* **and** *ess-bounded M g*
shows *ess-bounded-max-real*[*simp*,*intro*]: *ess-bounded M* (λ *x*. (*max* (*f x*) (*g x*)))
and *ess-bounded-min-real*[*simp*,*intro*]: *ess-bounded M* (λ *x*. (*min* (*f x*) (*g x*)))
proof−
have *f*[*measurable*]: $f \in$ *borel-measurable M*
using *assms*(*1*) *ess-bounded-def* **by** *auto*
have *g*[*measurable*]: $g \in$ *borel-measurable M*
using *assms*(*2*) *ess-bounded-def* **by** *auto*
obtain *r1* :: *real* **where** *r1*: *AE x in M*. *norm* (*f x*) $\leq$ *r1*
using *assms*(*1*) *ess-bounded-def* **by** *blast*
obtain *r2* :: *real* **where** *r2*: *AE x in M*. *norm* (*g x*) $\leq$ *r2*
using *assms*(*2*) *ess-bounded-def* **by** *blast*
let *?A* = {$x \in$ *space M*. *norm* (*f x*) > *r1*}
let *?B* = {$x \in$ *space M*. *norm* (*g x*) > *r2*}
have *NA*: *?A* $\in$ *null-sets M*
proof(*subst AE-iff-null-sets*)
show *?A*$\in$ *sets M***using** *f* **by** *measurable*
show *AE x in M*. $x \notin$ *?A***using** *r1* **by** *auto*
qed

have *NB*: *?B* ∈ *null-sets M*
proof(*subst AE-iff-null-sets*)
show *?B*∈ *sets M*
by *measurable*
show *AE x in M*. *x* ∉ *?B***using** *r2*
by *auto*
qed
show *ess-bounded M* (λ*x*. *max* (*f x*) (*g x*))
proof(*unfold ess-bounded-def*,*intro conjI*)
show (λ*x*. *max* (*f x*) (*g x*)) ∈ *borel-measurable M*
by *measurable*
show ∃ *r*. *AE x in M*. *norm* (*max* (*f x*) (*g x*)) ≤ *r*
proof
show *AE x in M*. *norm* (*max* (*f x*) (*g x*)) ≤ (*max r1 r2*)
proof(*rule AE-I′*)
show *?A* ∪ *?B* ∈ *null-sets M*
by (*auto simp*: *NA NB null-sets.Un*)
show {*x* ∈ *space M*. ¬ *norm* (*max* (*f x*) (*g x*)) ≤ *max r1 r2*} ⊆
{*x* ∈ *space M*. *r1* < *norm* (*f x*)} ∪ {*x* ∈ *space M*. *r2* < *norm* (*g x*)}
by *auto*
qed
qed
qed

show *ess-bounded M* (λ*x*. *min* (*f x*) (*g x*))
proof(*unfold ess-bounded-def*,*intro conjI*)
show (λ*x*. *min* (*f x*) (*g x*)) ∈ *borel-measurable M*
by *measurable*
show ∃ *r*. *AE x in M*. *norm* (*min* (*f x*) (*g x*)) ≤ *r*
proof
show *AE x in M*. *norm* (*min* (*f x*) (*g x*)) ≤ (*max r1 r2*)
proof(*rule AE-I′*)
show *?A* ∪ *?B* ∈ *null-sets M*
by (*auto simp*: *NA NB null-sets.Un*)
show {*x* ∈ *space M*. ¬ *norm* (*min* (*f x*) (*g x*)) ≤ *max r1 r2*} ⊆
{*x* ∈ *space M*. *r1* < *norm* (*f x*)} ∪ {*x* ∈ *space M*. *r2* < *norm* (*g x*)}
by *auto*
qed
qed
qed
qed

lemma
fixes *f g* :: -⇒ ′*b* :: {*banach*, *second-countable-topology*,*real-normed-algebra*}
assumes *ess-bounded M f*
and *ess-bounded M g*
shows *ess-bounded-add*[*simp*,*intro*]: *ess-bounded M* (λ *x*. *f x* + *g x*)
and *ess-bounded-diff*[*simp*,*intro*]: *ess-bounded M* (λ *x*. *f x* − *g x*)
and *ess-bounded-mult*[*simp*,*intro*]: *ess-bounded M* (λ *x*. *f x* ∗ *g x*)

```
proof(unfold ess-bounded-def)
  have f[measurable]: f ∈ borel-measurable M
    using assms(1) ess-bounded-def by auto
  have g[measurable]: g ∈ borel-measurable M
    using assms(2) ess-bounded-def by auto
  obtain r1 :: real where r1: AE x in M. norm (f x) ≤ r1
    using assms(1) ess-bounded-def by auto
  obtain r2 :: real where r2: AE x in M. norm (g x) ≤ r2
    using assms(2) ess-bounded-def by auto
  let ?A = {x ∈ space M. norm (f x) > r1}
  let ?B = {x ∈ space M. norm (g x) > r2}
  have NA: ?A ∈ null-sets M
  proof(subst AE-iff-null-sets)
    show ?A∈ sets Musing f by measurable
    show AE x in M. x ∉ ?Ausing r1 by auto
  qed
  have NB: ?B ∈ null-sets M
  proof(subst AE-iff-null-sets)
    show ?B∈ sets Musing g by measurable
    show AE x in M. x ∉ ?Busing r2 by auto
  qed

 show (λx. f x + g x) ∈ borel-measurable M ∧ (∃ r. AE x in M. norm
(f x + g x) ≤ r)
  proof
    show (λx. f x + g x) ∈ borel-measurable M
      by measurable
    show ∃ r. AE x in M. norm (f x + g x) ≤ r
    proof(intro exI[of - r1 + r2] AE-I')
      show ?A ∪ ?B ∈ null-sets M
        by (auto simp: NA NB null-sets.Un)
     show {x ∈ space M. ¬ norm (f x + g x) ≤ r1 + r2} ⊆ ?A ∪ ?B
      proof(safe)
        fix x assume x ∈ space M assume ¬ norm (f x + g x) ≤ r1
+ r2
        hence *: ¬ (r2 ≥ norm (g x) ∧ r1 ≥ norm (f x))
       by (metis add-mono-thms-linordered-semiring(1) dual-order.trans
norm-triangle-ineq)
        assume ¬ r2 < norm (g x)
        thusr1 < norm (f x)using * by auto
      qed
    qed
  qed


 show (λx. f x − g x) ∈ borel-measurable M ∧ (∃ r. AE x in M. norm
(f x − g x) ≤ r)
  proof
    show (λx. f x − g x) ∈ borel-measurable M
```

by *measurable*
show $\exists r.$ *AE x in M. norm* $(f\ x - g\ x) \leq r$
proof(*intro exI*[*of - r1 + r2*] *AE-I'*)
show *?A* $\cup$ *?B* $\in$ *null-sets M*
by (*auto simp*: *NA NB null-sets.Un*)
show $\{x \in$ *space M*. $\neg$ *norm* $(f\ x - g\ x) \leq r1 + r2\} \subseteq$ *?A* $\cup$ *?B*
proof(*safe*)
fix *x* **assume** $x \in$ *space M* **assume** *A0*: $\neg$ *norm* $(f\ x - g\ x) \leq$ $r1 + r2$
hence $*$: $\neg$ ($r2 \geq$ *norm* $(g\ x) \wedge r1 \geq$ *norm* $(f\ x))$
proof $-$
have *norm* $(f\ x - g\ x) \leq$ *norm* $(f\ x)$ + *norm* $(g\ x)$
using *norm-triangle-ineq4* **by** *blast*
thus *?thesis*
using *A0* **by** *force*
qed
thus $\neg$ $r2 <$ *norm* $(g\ x) \Longrightarrow r1 <$ *norm* $(f\ x)$
using $*$ *A0* **by** *auto*
qed
qed
qed

show $(\lambda x.\ f\ x * g\ x) \in$ *borel-measurable M* $\wedge$ ($\exists r.$ *AE x in M. norm* $(f\ x * g\ x) \leq r)$
proof
show $(\lambda x.\ f\ x * g\ x) \in$ *borel-measurable M*
using *f g borel-measurable-times* **by** *blast*
show $\exists r.$ *AE x in M. norm* $(f\ x * g\ x) \leq r$
proof(*intro exI*[*of - r1 * r2*] *AE-I'*)
show *?A* $\cup$ *?B* $\in$ *null-sets M*
by (*auto simp*: *NA NB null-sets.Un*)
show $\{x \in$ *space M*. $\neg$ *norm* $(f\ x * g\ x) \leq r1 * r2\} \subseteq$ *?A* $\cup$ *?B*
proof(*safe*)
fix *x* **assume** *A1*: $\neg$ *norm* $(f\ x * g\ x) \leq r1 * r2$
hence *A11*: $\neg$ (*norm* $(f\ x) \leq r1 \wedge$ *norm* $(g\ x) \leq r2)$
using *norm-mult-ineq mult-mono'*
by (*metis dual-order.trans norm-ge-zero*)
assume *A2*: $\neg$ $r2 <$ *norm* $(g\ x)$**show** $r1 <$ *norm* $(f\ x)$
using *A11 A2* **by** *auto*
qed
qed
qed
qed

lemma *integrable-ess-bounded-finite-measure*[*simp*]:
assumes *finite-measure M*
and *ess-bounded M f*
shows *integrable M f*
proof(*subst integrable-iff-bounded*,*rule conjI*)

show $f \in$ *borel-measurable M*
  using *assms*(*2*) *ess-bounded-def* **by** *auto*
obtain r **where** *normed*: *AE x in M. norm*(*f x*) $\leq r$
  using *assms ess-bounded-def* **by** *auto*
have ($\int^+$ *x. ennreal* (*norm* (*f x*)) ∂M) $\leq$($\int^+$ *x. ennreal r* ∂M)
proof(*rule nn-integral-mono-AE*)
  show *AE x in M. ennreal* (*norm* (*f x*)) $\leq$ *ennreal r*
    using *normed ennreal-leI* **by** *force*
qed
also have ... = *ennreal r* $*$ *emeasure M* (*space M*)
  by *auto*
also have ... $< \infty$
  using *assms*
 by (*auto simp*: *ennreal-mult-less-top finite-measure.emeasure-eq-measure*)
finally show ($\int^+$ *x. ennreal* (*norm* (*f x*)) ∂M) $< \infty$**.**
qed

lemma *indicat-real-ess-bounded*[*simp*,*intro*]:
  assumes $A \in$ *sets M*
  shows *ess-bounded M* (*indicat-real A*)
  unfolding *ess-bounded-def*
proof
  show *indicat-real A* $\in$ *borel-measurable M*
    using *assms* **by** *measurable*
  have *AE x in M. norm* (*indicat-real A x*) $\leq$ *1*
  proof(*rule AE-I2*,*cases*)
    fix x **assume** $x \in$ *space M*
    assume $x \in A$
    hence *indicat-real A x* = *1***by** *auto*
    thus*norm* (*indicat-real A x*) $\leq$ *1* **by** *auto*
  next fix x **assume** $x \in$ *space M*
    assume $x \notin A$
    hence *indicat-real A x* = *0***by** *auto*
    thus*norm* (*indicat-real A x*) $\leq$ *1* **by** *auto*
  qed
  thus$\exists$ *r. AE x in M. norm* (*indicat-real A x*) $\leq$ *r***by** *auto*
qed

lemma *indicat-real-integrable-finite-measure*[*simp*,*intro*]:
  assumes *finite-measure M* **and** $A \in$ *sets M*
  shows *integrable M* (*indicat-real A*)
  by (*auto simp*: *assms*)

lemma *probability-kernel-evaluation-ess-bounded*:
  assumes $f \in$ *measurable L* (*prob-algebra M*) **and** $A \in$ *sets M*
  shows *ess-bounded L* (λ *x. measure* (*f x*) *A*)
  unfolding *ess-bounded-def*
proof
  show (λ*x. measure* (*f x*) *A*) $\in$ *borel-measurable L*

using *assms* **by** *measurable*
have $\forall\ x \in space\ L.\ (measure\ (f\,x)\ A) \leq 1$
proof
fix x **assume** P: $x \in space\ L$
hence $(emeasure\ (f\,x)\ A) \leq 1$
by (*meson actual-prob-space assms*(*1*) *measurable-space prob-space.measure-le-1*)
thus$(measure\ (f\,x)\ A) \leq 1$
by (*metis P assms*(*1*) *assms*(*2*) *evaluations-probabilistic-process*(*3*))
qed
thus $\exists\,r.\ AE\ x\ in\ L.\ norm\ (measure\ (f\,x)\ A) \leq r$**by** *auto*
qed

lemma *probability-kernel-evaluation-integrable*[*simp*,*intro*]:
assumes *finite-measure L*
and $f \in measurable\ L\ (prob\text{-}algebra\ M)$
and $A \in sets\ M$
shows *integrable* $L\ (\lambda\ x.\ measure\ (f\,x)\ A)$
using *assms probability-kernel-evaluation-ess-bounded integrable-ess-bounded-finite-measure*
by *auto*

4.4 real-valued bounded density functions of finite measures

definition *real-RN-deriv* :: $'a\ measure \Rightarrow\ 'a\ measure \Rightarrow\ 'a \Rightarrow real$
where
real-RN-deriv L K =
$(if\ \exists\,k.\ (k \in borel\text{-}measurable\ L)$
$\wedge\ (density\ L\ (ennreal\ o\ k) = K)$
$\wedge\ (AE\ x\ in\ K.\ 0 < k\ x) \wedge (\forall\ x.\ 0 \leq k\ x)$
$then\ SOME\ k.\ ((k \in borel\text{-}measurable\ L)$
$\wedge\ (density\ L\ (ennreal\ o\ k) = K)$
$\wedge\ (AE\ x\ in\ K.\ 0 < k\ x) \wedge (\forall\ x.\ 0 \leq k\ x))$
$else\ (\lambda\text{-}.\ 0))$

lemma *real-RN-deriv-I*[*intro*]:
assumes $k \in borel\text{-}measurable\ L \wedge density\ L\ (ennreal \circ k) = K \wedge (AE\ x\ in\ K.\ 0 < k\ x) \wedge (\forall x.\ 0 \leq k\ x)$
shows $density\ L\ (ennreal\ o\ (real\text{-}RN\text{-}deriv\ L\ K)) = K$
and $(AE\ x\ in\ K.\ 0 < real\text{-}RN\text{-}deriv\ L\ K\ x)$
proof−
have $*$: $\exists\,k.\ (k \in borel\text{-}measurable\ L) \wedge (density\ L\ (ennreal\ o\ k) = K) \wedge (AE\ x\ in\ K.\ 0 < k\ x) \wedge (\forall\ x.\ 0 \leq k\ x)$
using *assms* **by** *auto*
hence $(density\ L\ (ennreal\ o\ (SOME\ k.\ ((k \in borel\text{-}measurable\ L) \wedge (density\ L\ (ennreal\ o\ k) = K) \wedge (AE\ x\ in\ K.\ 0 < k\ x) \wedge (\forall\ x.\ 0 \leq k\ x)))))) = K$
by (*rule someI2-ex*, *blast*)
thus $density\ L\ (ennreal\ o\ real\text{-}RN\text{-}deriv\ L\ K) = K$
using $*$ **by** (*auto simp*: *real-RN-deriv-def*)

from ∗ **have** *AE x in K. 0 < (SOME k. ((k ∈ borel-measurable L) ∧ (density L (ennreal o k) = K) ∧ (AE x in K. 0 < k x) ∧ (∀ x. 0 ≤ k x))) x*
by (*rule someI2-ex*, *blast*)
thus*AE x in K. 0 < real-RN-deriv L K x*
using ∗ **by** (*auto simp*: *real-RN-deriv-def*)
qed

lemma *real-RN-deriv-density*[*simp*]:
assumes *sigma-finite-measure L*
and *absolutely-continuous L K*
and *finite-measure K*
and *sets K = sets L*
shows *density L (ennreal o real-RN-deriv L K) = K*
proof−
obtain *k* **where** *kborel*: *k ∈ borel-measurable L*
and *AEkdensity*: *AE x in L. RN-deriv L K x = ennreal (k x)*
and *AEkpos*: *AE x in K. 0 < k x*
and *kpos*: *∀ x. 0 ≤ k x*
by(*rule sigma-finite-measure.real-RN-deriv*[*of L K*],*simp-all add*: *assms*)

have *kdensity*: *density L (ennreal o k) = density L (RN-deriv L K)*
proof(*rule density-cong*,*unfold comp-def*)
show (*λx. ennreal (k x)*) ∈ *borel-measurable L*
using *kborel* **by** *measurable*
show *AE x in L. ennreal (k x) = RN-deriv L K x*
using *AEkdensity* **by** *auto*
show *RN-deriv L K ∈ borel-measurable L*
by *auto*
qed
also have ... = *K*
using *sigma-finite-measure.density-RN-deriv assms*(*1*,*2*,*4*) **by** *auto*

hence *P*: (*k ∈ borel-measurable L*) ∧ (*density L (ennreal o k) = K*) ∧ (*AE x in K. 0 < k x*) ∧ (*∀ x. 0 ≤ k x*)
by (*auto simp*: *AEkpos kborel kdensity kpos*)
show *density L (ennreal o real-RN-deriv L K) = K*
by(*rule real-RN-deriv-I*[*OF P*])
qed

lemma *real-RN-deriv-positive-AE*[*simp*,*intro*]:
assumes *sigma-finite-measure L*
and *absolutely-continuous L K*
and *finite-measure K*
and *sets K = sets L*
shows *AE x in K. 0 < real-RN-deriv L K x*
proof−
obtain *k* **where** *kborel*[*measurable*]: *k ∈ borel-measurable L*

and *AEkdensity*: *AE x in L. RN-deriv L K x* = *ennreal* (*k x*)
and *AEkpos*: *AE x in K.* $0 < k\ x$
and *kpos*: $\forall\ x.\ 0 \leq k\ x$
by(*rule sigma-finite-measure.real-RN-deriv*[*of L K*],*simp-all add*: *assms*)
hence *density L* (*ennreal o k*) = *density L* (*RN-deriv L K*)
by(*intro density-cong*,*unfold comp-def*,*auto*)
also have ... = *K*
using *sigma-finite-measure.density-RN-deriv assms*(*1*) *assms*(*2*) *assms*(*4*) **by** *auto*
hence *P*: ($k \in$ *borel-measurable L*) $\wedge$ (*density L* (*ennreal o k*) = *K*) $\wedge$ (*AE x in K.* $0 < k\ x$) $\wedge$ ($\forall\ x.\ 0 \leq k\ x$)
by (*auto simp*: *calculation AEkpos kborel kpos*)
show *AE x in K.* $0 <$ *real-RN-deriv L K x*
by(*rule real-RN-deriv-I*[*OF P*])
qed

lemma *borel-measurable-real-RN-deriv*[*measurable*]:
shows *real-RN-deriv L K* $\in$ *borel-measurable L*
proof−
{**assume** $\exists\,k.$ (($k \in$ *borel-measurable L*) $\wedge$ (*density L* (*ennreal o k*) = *K*) $\wedge$ (*AE x in K.* $0 < k\ x$) $\wedge$ ($\forall\ x.\ 0 \leq k\ x$))
hence (*SOME k.* ($k \in$ *borel-measurable L*) $\wedge$ (*density L* (*ennreal o k*) = *K*) $\wedge$ (*AE x in K.* $0 < k\ x$) $\wedge$ ($\forall\ x.\ 0 \leq k\ x$)) $\in$ *borel-measurable L*
by (*rule someI2-ex*) *auto*}
thus *?thesis* **by** (*auto simp*: *real-RN-deriv-def*)
qed

lemma *real-RN-deriv-nonegative*[*simp*,*intro*]:
shows $\forall\,x.\ 0 \leq$ *real-RN-deriv L K x*
proof−
{
assume $\exists\,k.\ k \in$ *borel-measurable L* $\wedge$ *density L* (*ennreal* $\circ$ *k*) = *K* $\wedge$ (*AE x in K.* $0 < k\ x$) $\wedge$ ($\forall\,x.\ 0 \leq k\ x$)
hence $\forall\ x.\ 0 \leq$ (*SOME k.* ($k \in$ *borel-measurable L*) $\wedge$ (*density L* (*ennreal o k*) = *K*) $\wedge$ (*AE x in K.* $0 < k\ x$) $\wedge$ ($\forall\ x.\ 0 \leq k\ x$)) *x*
by (*rule someI2-ex*) *auto*
}
thus *?thesis*
by (*auto simp*: *real-RN-deriv-def*)
qed

lemma *real-RN-deriv-equals-RN-deriv-AE*:
assumes *sigma-finite-measure L*
and *absolutely-continuous L K*
and *finite-measure K*
and *sets K* = *sets L*
shows *AE x in L.* (*ennreal o real-RN-deriv L K*) *x* = (*RN-deriv L*

K) *x*
proof−
  have *P*: *density L* (*ennreal o real-RN-deriv L K*) = *K*
    using *assms real-RN-deriv-density* **by** *auto*
  show *AE x in L*. (*ennreal o real-RN-deriv L K*) *x* = (*RN-deriv L K x*)
  proof(*rule sigma-finite-measure.RN-deriv-unique*)
    show *density L* (*ennreal* ∘ *real-RN-deriv L K*) = *K*
      using *real-RN-deriv-density*[*OF assms*] **by** *auto*
    show *ennreal* ∘ *real-RN-deriv L K* ∈ *borel-measurable L*
      by *auto*
    show *sigma-finite-measure L*
      using *assms* **by** *auto*
  qed
qed

lemma *real-RN-deriv-equals-RN-deriv-AE2*:
  assumes *sigma-finite-measure L*
    and *absolutely-continuous L K*
    and *finite-measure K*
    and *sets K* = *sets L*
  shows *AE x in L*. *real-RN-deriv L K x* = *enn2real*((*RN-deriv L K*) *x*)
proof−
  have *AE x in L*. (*ennreal*((*real-RN-deriv L K*) *x*)) = (*RN-deriv L K*) *x*
    using *real-RN-deriv-equals-RN-deriv-AE*[*OF assms*] **by** *auto*
  hence *AE x in L*. *enn2real* (*ennreal*((*real-RN-deriv L K*) *x*)) = *enn2real* ((*RN-deriv L K*) *x*)
    by *fastforce*
  thus*AE x in L*. *real-RN-deriv L K x* = *enn2real* (*RN-deriv L K x*)
    by *auto*
qed

lemma *real-RN-deriv-bind*[*simp*]:
  assumes *sigma-finite-measure L*
    and *absolutely-continuous L K*
    and *K* ∈ *space* (*prob-algebra L*)
    and *f* ∈ *measurable L* (*prob-algebra M*)
    and *A* ∈ (*sets M*)
  shows *measure* (*K* ≫= *f*) *A* = ∫ *x*. (*real-RN-deriv L K x*) ∗ (*measure* (*f x*) *A*) ∂*L*
proof−
  have *finK*: *finite-measure K*
    using *space-prob-algebra-sets assms prob-space.finite-measure*[*of K*]
    by (*metis in-space-prob-algebra prob-spaceI sets-eq-imp-space-eq*)
  have *setsK*: *sets K* = *sets L*
    using *space-prob-algebra-sets assms* **by** *auto*

have *measure* $(K \gg\!= f)$ $A = \int x.($*measure* $(f\ x)$ $A)$ ∂K
proof(*rule subprob-space.measure-bind*[**where** $N = M$])
show *subprob-space K*
using *assms*(*3*) *prob-space-imp-subprob-space*[*of K*]
by (*metis in-space-prob-algebra prob-spaceI setsK sets-eq-imp-space-eq*)
show $f \in K \to_M$ *subprob-algebra M*
using *assms*(*4*) *assms*(*3*)
by (*metis measurable-cong-sets measurable-prob-algebraD space-prob-algebra-sets*)
show $A \in$ *sets M*
by (*intro assms*(*5*))
qed
also have ... $= \int x.$ (*measure* $(f\ x)$ $A)$ $\partial($*density L* (*ennreal o* (*real-RN-deriv L K*)))
using *real-RN-deriv-density*[*OF assms*(*1*) *assms*(*2*) *finK setsK*,*THEN sym*]
by *auto*
also have ... $= \int x.$ (*measure* $(f\ x)$ $A)$ $\partial($*density L* (*real-RN-deriv L K*))
by (*unfold comp-def*,*auto*)
also have ... $= \int x.$ (*real-RN-deriv L K x*) $*$ (*measure* $(f\ x)$ $A)$ ∂L
using *integral-density*[*of* λ *x.* (*measure* $(f\ x)$ $A)$*Lreal-RN-deriv L K*] *real-RN-deriv-nonegative assms*(*4*) *assms*(*5*)
by *force*
finally show *measure* $(K \gg\!= f)$ $A = \int x.$ (*real-RN-deriv L K x*) $*$ (*measure* $(f\ x)$ $A)$ ∂L.
qed

4.5 Locale for pairs of probability measures to compare.

locale *comparable-probability-measures* =
fixes *L M N* :: *'a measure*
assumes *M*: $M \in$ *space* (*prob-algebra L*)
and *N*: $N \in$ *space* (*prob-algebra L*)
begin

lemma *prob-spaceM*[*simp*,*intro*]: *prob-space M*
using *M space-prob-algebra* **by** *auto*
lemma *prob-spaceN*[*simp*,*intro*]: *prob-space N*
using *N space-prob-algebra* **by** *auto*
lemma *spaceM*[*simp*]: *sets M* = *sets L*
using *M space-prob-algebra* **by** *auto*
lemma *spaceN*[*simp*]: *sets N* = *sets L*
using *N space-prob-algebra* **by** *auto*
lemma *finM*[*simp*,*intro*]: *finite-measure M*
by(*rule prob-space.finite-measure*,*auto*)
lemma *finN*[*simp*,*intro*]: *finite-measure N*

by(*rule prob-space.finite-measure*,*auto*)
lemma *spaceML*[*simp*]: *space M* = *space L*
using *spaceM sets-eq-imp-space-eq* **by** *blast*
lemma *spaceNL*[*simp*]: *space N* = *space L*
using *spaceN sets-eq-imp-space-eq* **by** *blast*
lemma *McontMN*[*simp*,*intro*]: *absolutely-continuous* (*sum-measure M N*) *M*
by *auto*
lemma *NcontMN*[*simp*,*intro*]: *absolutely-continuous* (*sum-measure M N*) *N*
by *auto*
lemma *MNfinite*[*simp*,*intro*]: *finite-measure* (*sum-measure M N*)
by *auto*
lemma *MNsfinite*[*simp*,*intro*]: *sigma-finite-measure* (*sum-measure M N*)
using *finite-measure.sigma-finite-measure MNfinite* **by** *blast*
lemma *setMN-L*[*simp*]: *sets* (*sum-measure M N*) = *sets L*
by *auto*
lemma *spaceMN-L*[*simp*]: *space* (*sum-measure M N*) = *space L*
by *auto*
lemma *spaceMN-M*: *space* (*sum-measure M N*) = *space M*
by *auto*
lemma *setMN-M*: *sets* (*sum-measure M N*) = *sets M*
by *auto*
lemma *spaceMN-N*: *space* (*sum-measure M N*) = *space N*
by *auto*
lemma *setMN-N*: *sets* (*sum-measure M N*) = *sets N*
by *auto*
lemma *space-sumM*[*simp*]: *M* ∈ *space* (*prob-algebra* (*sum-measure M N*))
using *subprob-algebra-cong space-prob-algebra* **by** *fastforce*
lemma *space-sumN*[*simp*]: *N* ∈ *space* (*prob-algebra* (*sum-measure M N*))
using *subprob-algebra-cong space-prob-algebra* **by** *fastforce*

definition *dM* = *real-RN-deriv* (*sum-measure M N*) *M*

lemma
shows *borel-measurable-dM*[*measurable*]: *dM* ∈ *borel-measurable* (*sum-measure M N*)
and *dM-positive-AE*[*simp*]: *AE x in M*. *0* < *dM x*
and *dM-density* [*simp*]: *density* (*sum-measure M N*) (*ennreal o dM*) = *M*
and *dM-nonnegative*[*simp*]: (∀ *x*. *0* ≤ *dM x*)
by (*simp-all add*: *dM-def*)

lemma *dM-RN-deriv-AE*:

shows *AE x in sum-measure M N. dM x = enn2real* (*RN-deriv* (*sum-measure M N*) *M x*)
proof−
have *AE x in* (*sum-measure M N*). (*ennreal* ∘ *real-RN-deriv* (*sum-measure M N*) *M*) *x* = *RN-deriv* (*sum-measure M N*) *M x*
by(*rule real-RN-deriv-equals-RN-deriv-AE*,*simp-all*)
hence *AE x in* (*sum-measure M N*). *RN-deriv* (*sum-measure M N*) *M x* = *dM x*
using *dM-def* **by** *fastforce*
thus*AE x in sum-measure M N. dM x* = *enn2real* (*RN-deriv* (*sum-measure M N*) *M x*)
by *auto*
qed

lemma *dM-less-1-AE*:
shows *AE x in* (*sum-measure M N*). *dM x* ≤ *1*
proof(*rule functions-less-upto-AE*[**where** *g* = *real-of-ereal o enn2ereal o* (*RN-deriv* (*sum-measure M N*) *M*)])
show *AE x in sum-measure M N. dM x* = (*real-of-ereal* ∘ *enn2ereal* ∘ *RN-deriv* (*sum-measure M N*) *M*) *x*
using *real-RN-deriv-equals-RN-deriv-AE2* **by** (*auto simp*: *dM-RN-deriv-AE*)
have *AE x in sum-measure M N.* (*RN-deriv* (*sum-measure M N*) *M x*) ≤ *1*
by *auto*
thus*AE x in sum-measure M N.* (*real-of-ereal* ∘ *enn2ereal* ∘ *RN-deriv* (*sum-measure M N*) *M*) *x* ≤ *1*
using *enn2real-leI* **by** *fastforce*
qed(*auto*)

lemma *dM-bounded*:
shows *AE x in* (*sum-measure M N*). |*dM x*| ≤ *1*
using *dM-less-1-AE dM-nonnegative* **by** *auto*

lemma *dM-boundes2*[*simp*,*intro*]:
shows *ess-bounded* (*sum-measure M N*) *dM*
unfolding *ess-bounded-def* **using** *dM-bounded borel-measurable-dM* **by** *fastforce*

lemma *dM-integrable*[*simp*,*intro*]:
shows *integrable*(*sum-measure M N*) *dM*
using *dM-boundes2* **by** *auto*

lemma *dM-bind-integral*[*simp*]:
assumes *f* ∈ *measurable* (*sum-measure M N*) (*prob-algebra K*) *A* ∈ (*sets K*)
shows *measure* (*M* ⋙= *f*) *A* = ∫ *x.* (*dM x*) ∗ (*measure* (*f x*) *A*) ∂(*sum-measure M N*)
using *assms*(*1*) *assms*(*2*) *dM-def* **by** *auto*

definition *dN* = *real-RN-deriv* (*sum-measure M N*) *N*

lemma
shows *borel-measurable-dN*[*measurable*]: *dN* ∈ *borel-measurable* (*sum-measure M N*)
and *dN-positive-AE*[*simp*]: *AE x in N*. *0* < *dN x*
and *dN-density*[*simp*]: *density* (*sum-measure M N*) (*ennreal o dN*) = *N*
and *dN-nonnegative*[*simp*]: (∀ *x*. *0* ≤ *dN x*)
by (*simp-all add*: *dN-def*)

lemma *dN-less-1-AE*:
shows *AE x in* (*sum-measure M N*). *dN x* ≤ *1*
proof(*rule functions-less-upto-AE*[**where** *g* = *real-of-ereal o enn2ereal o* (*RN-deriv* (*sum-measure M N*) *N*)])
have *AE x in* (*sum-measure M N*). (*ennreal* ∘ *real-RN-deriv* (*sum-measure M N*) *N*) *x* = *RN-deriv* (*sum-measure M N*) *N x*
by(*rule real-RN-deriv-equals-RN-deriv-AE*,*simp-all*)
thus*AE x in sum-measure M N*. *dN x* = (*real-of-ereal* ∘ *enn2ereal* ∘ *RN-deriv* (*sum-measure M N*) *N*) *x*
by (*auto simp*: *dN-def real-RN-deriv-equals-RN-deriv-AE2*)
have *AE x in sum-measure M N*. (*RN-deriv* (*sum-measure M N*) *N x*) ≤ *1*
using *spaceMN-N* **by** *auto*
thus*AE x in sum-measure M N*. (*real-of-ereal* ∘ *enn2ereal* ∘ *RN-deriv* (*sum-measure M N*) *N*) *x* ≤ *1*
using *enn2real-leI* **by** *fastforce*
qed(*auto*)

lemma *dN-bounded*:
shows *AE x in* (*sum-measure M N*). |*dN x*| ≤ *1*
using *dN-less-1-AE dN-nonnegative* **by** *auto*

lemma *dN-boundes2*[*simp*,*intro*]:
shows *ess-bounded* (*sum-measure M N*) *dN*
unfolding *ess-bounded-def* **using** *dN-bounded borel-measurable-dN* **by** *fastforce*

lemma *dN-integrable*[*simp*,*intro*]:
shows *integrable*(*sum-measure M N*) *dN*
using *dN-boundes2* **by** *auto*

lemma *dN-bind-integral*[*simp*]:
assumes *f* ∈ *measurable* (*sum-measure M N*) (*prob-algebra K*) **and** *A* ∈ (*sets K*)
shows *measure* (*N* ⋙= *f*) *A* = ∫ *x*. (*dN x*) ∗ (*measure* (*f x*) *A*) ∂(*sum-measure M N*)
using *assms*(*1*) *assms*(*2*) *dN-def* **by** *auto*

lemma *dM-dN-partition-1-AE*:
shows *AE x in sum-measure M N. dM x + dN x = 1*
proof−
have *F*: ∀ *A* ∈ *sets*(*sum-measure M N*). *emeasure* (*density* (*sum-measure M N*) (λ *x. ennreal* (*dM x* + *dN x*))) *A* = *emeasure* (*sum-measure M N*) *A*
proof
fix *A* **assume** *A*: *A* ∈ *sets* (*sum-measure M N*)
hence *emeasure* (*density* (*sum-measure M N*) (λ*x. ennreal* (*dM x* + *dN x*))) *A* = *emeasure* (*density* (*sum-measure M N*) (λ*x. ennreal* (*dM x*))) *A* + *emeasure* (*density* (*sum-measure M N*) (λ*x. ennreal* (*dN x*))) *A*
by(*auto simp* : *A intro*!: *Nonnegative-Lebesgue-Integration.emeasure-density-add*[*symmetric*])
also have ... = *emeasure M A* + *emeasure N A*
by(*metis dN-density dM-density comp-def*[*THEN sym*])
also have ... = *emeasure* (*sum-measure M N*) *A*
using *A* **by** *auto*
finally show *emeasure* (*density* (*sum-measure M N*) (λ*x. ennreal* (*dM x* + *dN x*))) *A* = *emeasure* (*sum-measure M N*) *A*.
qed
hence (*density* (*sum-measure M N*) (λ *x. ennreal* (*dM x* + *dN x*))) = (*sum-measure M N*)
by(*auto intro*!: *measure-eqI*)
hence *P1*: *AE x in sum-measure M N. ennreal*(*dM x* + *dN x*) = (*RN-deriv* (*sum-measure M N*) (*sum-measure M N*) *x*)
by(*auto intro*!: *sigma-finite-measure.RN-deriv-unique*)
have *P3*: (*density* (*sum-measure M N*) (λ *x. 1* :: *ennreal*)) = (*sum-measure M N*)
by (*auto simp*: *density-1*)
have *P2*: *AE x in sum-measure M N.* (λ *y. 1* :: *ennreal*) *x* = (*RN-deriv* (*sum-measure M N*) (*sum-measure M N*) *x*)
by(*rule sigma-finite-measure.RN-deriv-unique*,*auto simp add*: *P3*)
have *AE x in sum-measure M N. ennreal*(*dM x* + *dN x*) = (*1* :: *ennreal*)
using *P1 P2* **by** *auto*
thus*AE x in sum-measure M N.* (*dM x* + *dN x*) = (*1* :: *real*)
proof
show *AE x in sum-measure M N. ennreal* (*dM x* + *dN x*) = *1* ⟶ *dM x* + *dN x* = *1*
by (*metis* (*mono-tags*, *lifting*) *AE-I2 ennreal-eq-1*)
qed
qed

end

end

theory *Differential-Privacy-Divergence*

imports *Comparable-Probability-Measures*
begin

5 Divergence for Differential Privacy

First, we introduce the divergence for differential privacy.

definition *DP-divergence* :: $'a$ *measure* $\Rightarrow$ $'a$ *measure* $\Rightarrow$ *real* $\Rightarrow$ *ereal*
where
DP-divergence $M\ N\ \varepsilon = Sup\ \{ereal(measure\ M\ A - (exp\ \varepsilon) * measure\ N\ A) \mid A{::}'a\ set.\ A \in (sets\ M)\}$

lemma *DP-divergence-SUP*:
shows *DP-divergence* $M\ N\ \varepsilon = (\bigsqcup\ (A{::}'a\ set) \in (sets\ M).\ ereal(measure\ M\ A - (exp\ \varepsilon) * measure\ N\ A))$
by (*auto simp*: *setcompr-eq-image DP-divergence-def*)

5.1 Basic Properties

5.1.1 Non-negativity

lemma *DP-divergence-nonnegativity*:
shows $0 \leq$ *DP-divergence* $M\ N\ \varepsilon$
proof(*unfold DP-divergence-SUP*, *rule Sup-upper2*[*of 0*])
have $\{\} \in sets\ M\ (\lambda A :: 'a\ set.\ ereal\ (measure\ M\ A - exp\ \varepsilon * measure\ N\ A))\ \{\} = 0$
by *auto*
thus $0 \in (\lambda A.\ ereal\ (measure\ M\ A - exp\ \varepsilon * measure\ N\ A))$ ‘ $sets\ M$
by *force*
qed(*auto*)

5.1.2 Graded predicate

lemma *DP-divergence-forall*:
shows $(\forall\ A \in (sets\ M).\ (measure\ M\ A - (exp\ \varepsilon) * measure\ N\ A \leq (\delta :: real)))$
$\longleftrightarrow$ *DP-divergence* $M\ N\ \varepsilon \leq (\delta :: real)$
unfolding *DP-divergence-SUP* **by** (*auto simp*: *SUP-le-iff*)

lemma *DP-divergence-leE*:
assumes *DP-divergence* $M\ N\ \varepsilon \leq (\delta :: real)$
shows $measure\ M\ A \leq (exp\ \varepsilon) * measure\ N\ A + (\delta :: real)$
proof(*cases* $A \in (sets\ M)$)
case *True*
then show *?thesis* **using** *DP-divergence-forall assms*(*1*) **by** *force*
next
case *False*
then have $measure\ M\ A = 0\ \ 0 \leq\ measure\ N\ A\ \ 0 \leq (exp\ \varepsilon)$
using *measure-notin-sets* **by** *auto*

moreover have $0 \leq$ *ereal* δ
using *assms DP-divergence-nonnegativity*[*of M N* ε] *dual-order.trans* **by** *blast*
ultimately show *?thesis* **by** *auto*
qed

lemma *DP-divergence-leI*:
assumes $\bigwedge A.\ A \in (sets\ M) \Longrightarrow measure\ M\ A \leq (exp\ \varepsilon) * measure\ N\ A + (\delta :: real)$
shows *DP-divergence M N* $\varepsilon \leq (\delta :: real)$
using *assms* **unfolding** *DP-divergence-def* **by**(*intro cSup-least*, *fastforce+*)

5.1.3 (*0*,*0*)-DP means the equality of distributions

lemma *prob-measure-eqI-le*:
assumes *M*: $M \in space\ (prob\text{-}algebra\ L)$
and *N*: $N \in space\ (prob\text{-}algebra\ L)$
and *le*:$\forall\ A \in sets\ M.\ emeasure\ M\ A \leq emeasure\ N\ A$
shows $M = N$
proof(*rule measure-eqI*)
interpret *pM*: *prob-space M*
using *actual-prob-space M* **by** *auto*
interpret *pN*: *prob-space N*
using *actual-prob-space N* **by** *auto*
show $*$: $sets\ M = sets\ N$
using *M N space-prob-algebra-sets* **by** *blast*
show $\bigwedge A.\ A \in sets\ M \Longrightarrow emeasure\ M\ A = emeasure\ N\ A$
proof (*rule ccontr*)
fix *A* **assume** *A*: $A \in sets\ M$ **and** *neq*:$emeasure\ M\ A \neq emeasure\ N\ A$
then have *A2*: $A \in sets\ N$ **and** *A3*: $A \in sets\ L$
using *M N space-prob-algebra-sets* **by** *blast+*
from *neq* **consider**
$emeasure\ M\ A < emeasure\ N\ A \mid emeasure\ M\ A > emeasure\ N\ A$
by *fastforce*
then show *False*
proof(*cases*)
case *1*
have *Ms*: $emeasure\ M\ (space\ M - A) = emeasure\ M\ (space\ M) - emeasure\ M\ A$
using *A sets.sets-into-space* **by**(*subst emeasure-Diff*,*auto*)
have *Ns*: $emeasure\ N\ (space\ M - A) = emeasure\ N\ (space\ N) - emeasure\ N\ A$
using *A2 sets.sets-into-space sets-eq-imp-space-eq*[*OF* $*$] **by**(*subst emeasure-Diff*,*auto*)
have $emeasure\ M\ (space\ M - A) > emeasure\ N\ (space\ M - A)$
unfolding *Ms Ns pM.emeasure-space-1 pN.emeasure-space-1*
by (*meson 1 ennreal-mono-minus-cancel ennreal-one-less-top leI*

less-le-not-le pM.measure-le-1 pN.measure-le-1)
    moreover have *emeasure M* (*space M* $-$ *A*)$\leq$ *emeasure N* (*space M* $-$ *A*)
      using *le A* **by** *auto*
     ultimately show *?thesis*
      by *auto*
   next
    case *2*
    from *le A* **have** *emeasure M A* $\leq$ *emeasure N A*
     by *auto*
    with *2* **show** *?thesis*
     by *auto*
   qed
  qed
qed

lemma *DP-divergence-zero*:
  assumes *M*: *M* $\in$ *space* (*prob-algebra L*)
   and *N*: *N* $\in$ *space* (*prob-algebra L*)
   and *DP0*:*DP-divergence M N* (*0*::*real*) $\leq$ (*0*::*real*)
  shows *M* = *N*
proof(*intro prob-measure-eqI-le*[*of M L N*] *M N ballI*)
  interpret *comparable-probability-measures L M N*
   by(*unfold-locales*,*auto simp*: *M N*)
  interpret *pM*: *prob-space M*
   by *auto*
  interpret *pN*: *prob-space N*
   by *auto*
  fix *A* **assume** *A* $\in$ *sets M*
  hence *measure M A* $-$ *measure N A* $\leq$ *0*
   using *DP0*[*simplified zero-ereal-def*] **unfolding** *DP-divergence-forall*[*symmetric*] **by** *auto*
  hence *measure M A* $\leq$ *measure N A*
   by *auto*
  thus *emeasure M A* $\leq$ *emeasure N A*
   by (*simp add*: *pM.emeasure-eq-measure pN.emeasure-eq-measure*)
qed

5.1.4 Conversion from pointwise DP [4]

lemma *DP-divergence-pointwise*:
  assumes *M*: *M* $\in$ *space* (*prob-algebra L*)
   and *N*: *N* $\in$ *space* (*prob-algebra L*)
   and *C*: *C* $\in$ *sets M*
   and $\forall$ *A*$\in$*sets M*. *A* $\subseteq$ *C* $\longrightarrow$ *measure M A* $\leq$ *exp* ε $*$ *measure N A*
   and *1* $-$ δ $\leq$ *measure M C*
  shows *DP-divergence M N* (ε::*real*) $\leq$ δ
proof(*rule DP-divergence-leI*)

interpret *comparable-probability-measures L M N*
by(*unfold-locales*, *auto simp*: *assms*)
fix *A* **assume** *A*:$A \in$ *sets M*
define *A1* **and** *A2* **where** $A1 = A \cap C$ **and** $A2 = A - C$
hence *A1s*[*measurable*]:$A1 \in$ *sets M* **and** *A2s*[*measurable*]:$A2 \in$ *sets M* **and** *A1C*:$A1 \subseteq C$ **and** *A12*: $A1 \cup A2 = A$
unfolding *A1-def A2-def* **using** *C A* **by** *auto*
hence *measure M A1* $\leq$ (*exp* ε) $*$ *measure N A1*
using *assms*(*4*) **by** *blast*
also have ... $\leq$ (*exp* ε) $*$ *measure N A*
using *A1s A2s* **unfolding** *A12*[*symmetric*]
by(*subst measure-Union*[*of N A1 A2*]) (*auto simp add*: *finite-measure.emeasure-finite A1-def A2-def*)
finally have *1*: *measure M A1* $\leq$ *exp* ε $*$ *measure N A*.

have *A2Cc*:$A2 \subseteq$ *space M* $-$ *C*
by (*metis A A2-def Diff-mono sets.sets-into-space subset-refl*)
have *measure M A2* $\leq$ *measure M* (*space M* $-$ *C*)
by(*intro finite-measure.finite-measure-mono A2Cc Sigma-Algebra.sets.compl-sets C*,*auto*)
also have ... $=$ *measure M* (*space M*) $-$ *measure M C*
using *assms*(*3*) *finM finite-measure.finite-measure-compl* **by** *blast*
also have ...$\leq$ *1* $-$ (*1* $-$ δ)
using *M actual-prob-space prob-space.axioms assms*(*5*) *prob-space.prob-space* **by** *force*
also have ... $= \delta$
by *auto*
finally have *2*: *measure M A2* $\leq \delta$.

have *measure M A* $=$ *measure M A1* $+$ *measure M A2*
using *A1s A2s* **unfolding** *A12*[*symmetric*]
by(*subst measure-Union*[*of M A1 A2*]) (*auto simp add*: *finite-measure.emeasure-finite A1-def A2-def*)
also have ... $\leq$ *exp* ε $*$ *measure N A* $+ \delta$
using *1 2* **by** *auto*
finally show *measure M A* $\leq$ *exp* ε $*$ *measure N A* $+ \delta$ **.**
qed

5.1.5 (Reverse-) Monotonicity for ε

lemma *DP-divergence-monotonicity*:
assumes $\varepsilon 1 \leq \varepsilon 2$
shows *DP-divergence M N* $\varepsilon 2$ $\leq$ *DP-divergence M N* $\varepsilon 1$
proof(*unfold DP-divergence-SUP*, *rule SUP-mono*)
fix *A* **assume** $*$: $A \in$ *sets M*
have *ereal* (*measure M A* $-$ *exp* $\varepsilon 2$ $*$ *measure N A*) $\leq$ *ereal* (*measure M A* $-$ *exp* $\varepsilon 1$ $*$ *measure N A*)
by (*auto simp*: *assms mult-mono'*)
with $*$ **show** $\exists\, m \in$*sets M*. *ereal* (*measure M A* $-$ *exp* $\varepsilon 2$ $*$ *measure*

N A) ≤ *ereal* (*measure M m* − *exp ε1* ∗ *measure N m*)
  by *blast*
qed

corollary *DP-divergence-monotonicity′*:
 assumes *M*: *M* ∈ *space* (*prob-algebra L*)
  and *N*: *N* ∈ *space* (*prob-algebra L*)
  and *ε1* ≤ *ε2* **and** *δ1* ≤ *δ2*
 shows *DP-divergence M N ε1* ≤ *δ1* ⟹ *DP-divergence M N ε2* ≤ *δ2*
 by (*meson DP-divergence-monotonicity M N assms gfp.leq-trans*)

5.1.6 Reflexivity

lemma *DP-divergence-reflexivity*:
 shows *DP-divergence M M 0* = *0*
proof(*unfold DP-divergence-SUP*)
 have ∀ *A* ∈*sets M*. *ereal* (*measure M A* − *exp 0* ∗ *measure M A*) = (*0* :: *ereal*)
  by *auto*
 thus (⨆*A* ∈*sets M*. *ereal* (*measure M A* − *exp 0* ∗ *measure M A*)) = (*0* :: *ereal*)
   by (*metis* (*no-types*, *lifting*) *SUP-ereal-eq-0-iff-nonneg empty-iff order-refl sets.top*)
qed

corollary *DP-divergence-reflexivity′*:
 assumes *0* ≤ *ε*
 shows *DP-divergence M M ε* = *0*
proof−
 have *DP-divergence M M ε* ≤ *DP-divergence M M 0*
  by(*rule DP-divergence-monotonicity*, *auto simp*: *assms*)
 also have ... = *0*
  by(*auto simp*: *DP-divergence-reflexivity*)
 finally have *1*: *DP-divergence M M ε* ≤ *0*.
 have *2*: *0* ≤ *DP-divergence M M ε*
  by(*auto simp*: *DP-divergence-nonnegativity*[*of M M ε*])
 from *1 2* **show** *?thesis* **by** *auto*
qed

5.1.7 Transitivity

lemma *DP-divergence-transitivity*:
 assumes *DP1*: *DP-divergence M1 M2 ε1* ≤ *0*
  and *DP2*: *DP-divergence M2 M3 ε2* ≤ *0*
 shows *DP-divergence M1 M3* (*ε1*+*ε2*) ≤ *0*
 unfolding *zero-ereal-def*
proof(*subst DP-divergence-forall*[*symmetric*],*intro ballI*)
 fix *A* **assume** *AM1*: *A* ∈ *sets M1*
 have *measure M1 A* ≤ *exp ε1* ∗ *measure M2 A*

using *DP-divergence-leE*[*OF DP1*[*simplified zero-ereal-def*]] **by** *auto*
also have ... $\leq$ *exp* ε*1* $*$ *exp* ε*2* $*$ *measure M3 A*
using *DP-divergence-leE*[*OF DP2*[*simplified zero-ereal-def*]] **by** *auto*
finally have *measure M1 A* $\leq$ *exp* (ε*1* + ε*2*) $*$ *measure M3 A*
unfolding *exp-add*[*of* ε*1* ε*2*, *symmetric*] **by** *auto*
thus *measure M1 A* $-$ *exp* (ε*1* + ε*2*) $*$ *measure M3 A* $\leq$ *0* **by** *auto*
qed

5.1.8 Composability

proposition *DP-divergence-composability*:
assumes *M*: *M* $\in$ *space* (*prob-algebra L*)
and *N*: *N* $\in$ *space* (*prob-algebra L*)
and *f*: *f* $\in$ *L* $\rightarrow_M$ *prob-algebra K*
and *g*: *g* $\in$ *L* $\rightarrow_M$ *prob-algebra K*
and *div1*: *DP-divergence M N* ε*1* $\leq$ (δ*1* :: *real*)
and *div2*: $\forall$ *x* $\in$ (*space L*). *DP-divergence* (*f x*) (*g x*) ε*2* $\leq$ (δ*2* :: *real*)
and *0* $\leq$ ε*1* **and** *0* $\leq$ ε*2*
shows *DP-divergence* (*M* $\gg$= *f*) (*N* $\gg$= *g*) (ε*1* + ε*2*) $\leq$ δ*1* + δ*2*
proof(*subst DP-divergence-forall*[*THEN sym*])
note [*measurable*] = *f g*
interpret *comparable-probability-measures L M N*
by(*unfold-locales*, *auto simp*: *assms*)

show $\forall$ *A*$\in$*sets* (*M* $\gg$= *f*). *measure* (*M* $\gg$= *f*) *A* $-$ *exp* (ε*1* + ε*2*) $*$ *measure* (*N* $\gg$= *g*) *A* $\leq$ δ*1* + δ*2*
proof
fix *A* **assume** *A* $\in$ *sets* (*M* $\gg$= *f*)
hence *AinSetsK*: *A* $\in$ *sets K*
using *M f sets-bind'* **by** *blast*

have *f2*[*measurable*]: *f* $\in$ *sum-measure M N* $\rightarrow_M$ *prob-algebra K*
using *setMN-L* **by** *measurable*
have *g2*[*measurable*]: *g* $\in$ *sum-measure M N* $\rightarrow_M$ *prob-algebra K*
using *setMN-L* **by** *measurable*
have *ess-boundedf*[*simp*]: *ess-bounded* (*sum-measure M N*) (λ*x*. *measure* (*f x*) *A*)
by (*intro probability-kernel-evaluation-ess-bounded*[**where** *M* = *K*] *AinSetsK f2*)
have *ess-boundedg*[*simp*]: *ess-bounded* (*sum-measure M N*) (λ*x*. *measure* (*g x*) *A*)
by (*intro probability-kernel-evaluation-ess-bounded*[**where** *M* = *K*] *AinSetsK g2*)
have *integrablef*[*simp*]: *integrable* (*sum-measure M N*) (λ*x*. *measure* (*f x*) *A*)
by (*auto simp*:*AinSetsK*)

have *integrableg*[*simp*]: *integrable* (*sum-measure M N*) (λx. *measure* (*g x*) *A*)
by (*auto simp*: *AinSetsK*)
have *meas1*[*measurable-cong*]: *measurable L* (*prob-algebra K*) = *measurable* (*sum-measure M N*) (*prob-algebra K*)
by(*rule measurable-cong-sets*, *simp-all*)
have *f2*[*measurable*]: $f \in$ *sum-measure M N* $\rightarrow_M$ *prob-algebra K*
by *auto*
have *g2*[*measurable*]: $g \in$ *sum-measure M N* $\rightarrow_M$ *prob-algebra K*
by *auto*

have $\forall x\in$ *space* (*sum-measure M N*). *measure* (*f x*) *A* − (*exp* $\varepsilon 2$) ∗ *measure* (*g x*) *A* ≤ $\delta 2$
proof
fix *x* **assume** *A*: $x \in$ *space* (*sum-measure M N*)
hence *A2*: $A \in$ *sets* (*f x*)
by (*metis AinSetsK f2 measurable-prob-algebraD subprob-measurableD*(*2*))
from *A* **have** *xxx*: *DP-divergence* (*f x*) (*g x*) $\varepsilon 2 \leq \delta 2$
using *div2 spaceMN-L* **by** *auto*
show *measure* (*f x*) *A* − *exp* $\varepsilon 2$∗ *measure* (*g x*) *A* ≤ $\delta 2$
using *xxx*[*simplified DP-divergence-forall*[*THEN sym*,*rule-format*]] *A2* **by** *blast*
qed
hence *ineq7*: $\forall x\in$ *space* (*sum-measure M N*). *measure* (*f x*) *A* − $\delta 2$ ≤ *exp* $\varepsilon 2$ ∗ *measure* (*g x*) *A*
by *auto*

have $\forall x\in$ *space* (*sum-measure M N*). *measure* (*f x*) *A*− $\delta 2$ ≤ *min 1* (*exp* $\varepsilon 2$ ∗ *measure* (*g x*) *A*)
proof
fix *x* **assume** *Ass0*: $x \in$ *space* (*sum-measure M N*)
hence *0* ≤ *DP-divergence* (*f x*) (*g x*) $\varepsilon 2$
using *DP-divergence-nonnegativity* **by** *auto*
also have *DP-divergence* (*f x*) (*g x*) $\varepsilon 2 \leq \delta 2$
using *div2 Ass0 spaceMN-L* **by** *auto*
finally have *posdelta2*: *0* ≤ $\delta 2$**by** *auto*
hence *minA*: *measure* (*f x*) *A* − $\delta 2$ ≤ *1*
using *evaluations-probabilistic-process*(*3*)[*OF f AinSetsK*] *assms Ass0 spaceMN-L* **by** *auto*
have *minB*: *measure* (*f x*) *A* − $\delta 2$ ≤ *exp* $\varepsilon 2$ ∗ *measure* (*g x*) *A*
using *Ass0 ineq7* **by** *auto*
from *minA minB* **show** *measure* (*f x*) *A* − $\delta 2$ ≤ *min 1* (*exp* $\varepsilon 2$ ∗ *measure* (*g x*) *A*)
by *auto*
qed
hence *inequalityB*: $\forall x\in$ *space* (*sum-measure M N*). *max 0* (*measure* (*f x*) *A* − $\delta 2$) ≤ *min 1* (*exp* $\varepsilon 2$ ∗ *measure* (*g x*) *A*)
by *auto*

have *inequalityC*: $\forall x \in$ *space* (*sum-measure M N*). *dM x* $*$ *max 0* (*measure* (*f x*) *A* $-$ $\delta 2$) $\leq$ *dM x* $*$ *min 1* (*exp* $\varepsilon 2$ $*$ *measure* (*g x*) *A*)
proof
fix *x* **assume** *A0*: $x \in$ *space* (*sum-measure M N*)
with *inequalityB* **have** *P1*: (*max 0* (*measure* (*f x*) *A* $-$ $\delta 2$)) $\leq$ *min 1* ((*exp* $\varepsilon 2$) $*$ *measure* (*g x*) *A*)
by *auto*
thus *dM x* $*$ *max 0* (*measure* (*f x*) *A* $-$ $\delta 2$) $\leq$ *dM x* $*$ *min 1* (*exp* $\varepsilon 2$ $*$ *measure* (*g x*) *A*)
by(*intro mult-left-mono*, *auto*)
qed

have (*measure* (*M* $\ggg$= *f*) *A*) $-$ *exp* ($\varepsilon 1$ + $\varepsilon 2$) $*$ (*measure* (*N* $\ggg$= *g*) *A*) = ($\int$ *x*. (*dM x*) $*$ (*measure* (*f x*) *A*) ∂(*sum-measure M N*)) $-$ (*exp* ($\varepsilon 1$ + $\varepsilon 2$)) $*$ ($\int$ *x*. (*dN x*) $*$ (*measure* (*g x*) *A*) ∂(*sum-measure M N*))
using *AinSetsK dM-bind-integral*[*OF f2*] *dN-bind-integral*[*OF g2*] **by** *auto*
also have ... = ($\int$ *x*. (*dM x*) $*$ (*measure* (*f x*) *A*) ∂(*sum-measure M N*)) $-$ ($\int$ *x*. (*exp* ($\varepsilon 1$ + $\varepsilon 2$)) $*$ (*dN x*) $*$ (*measure* (*g x*) *A*) ∂(*sum-measure M N*))
by (*metis* (*mono-tags*, *lifting*) *Bochner-Integration.integral-cong integral-mult-right-zero mult.assoc*)
also have ... = ($\int$ *x*. (*dM x*) $*$ (*measure* (*f x*) *A*) $-$ (*exp* ($\varepsilon 1$ + $\varepsilon 2$)) $*$ (*dN x*) $*$ (*measure* (*g x*) *A*) ∂(*sum-measure M N*))
by(*rule Bochner-Integration.integral-diff*[*THEN sym*],*auto*)
also have ... = ($\int$ *x*. (*dM x*) $*$ (*measure* (*f x*) *A*) $-$ (*exp* $\varepsilon 1$) $*$ (*exp* $\varepsilon 2$) $*$ (*dN x*) $*$ (*measure* (*g x*) *A*) ∂(*sum-measure M N*))
by (*auto simp*: *exp-add*)
also have ... $\leq$ ($\int$ *x*. (*dM x*) $*$ (*max 0* (*measure* (*f x*) *A* $-$ $\delta 2$) + $\delta 2$) $-$ (*exp* $\varepsilon 1$) $*$ (*dN x*) $*$ *min 1* ((*exp* $\varepsilon 2$) $*$ (*measure* (*g x*) *A*)) ∂(*sum-measure M N*))
proof(*rule integral-mono*)
fix *x* **assume** *A0*: $x \in$ *space* (*sum-measure M N*)
have *dM x* $*$ *measure* (*f x*) *A* $-$ *exp* $\varepsilon 1$ $*$ *exp* $\varepsilon 2$ $*$ *dN x* $*$ *measure* (*g x*) *A* = *dM x* $*$ ((*measure* (*f x*) *A* $-$ $\delta 2$) + $\delta 2$)$-$ *exp* $\varepsilon 1$ $*$ *dN x* $*$(*exp* $\varepsilon 2$ $*$ *measure* (*g x*) *A*)
by *auto*
also have ... $\leq$ *dM x* $*$ ((*max 0* (*measure* (*f x*) *A* $-$ $\delta 2$)) + $\delta 2$)$-$ *exp* $\varepsilon 1$ $*$ *dN x* $*$(*exp* $\varepsilon 2$ $*$ *measure* (*g x*) *A*)
by(*auto simp*: *mult-left-mono*)
also have ... $\leq$ *dM x* $*$ ((*max 0* (*measure* (*f x*) *A* $-$ $\delta 2$)) + $\delta 2$)$-$ *exp* $\varepsilon 1$ $*$ *dN x* $*$ (*min 1* (*exp* $\varepsilon 2$ $*$ *measure* (*g x*) *A*))
by(*auto simp*: *mult-left-mono*)
finally show *dM x* $*$ *measure* (*f x*) *A* $-$ *exp* $\varepsilon 1$ $*$ *exp* $\varepsilon 2$ $*$ *dN x* $*$ *measure* (*g x*) *A* $\leq$ *dM x* $*$ (*max 0* (*measure* (*f x*) *A* $-$ $\delta 2$) + $\delta 2$) $-$ *exp* $\varepsilon 1$ $*$ *dN x* $*$ *min 1* (*exp* $\varepsilon 2$ $*$ *measure* (*g x*) *A*)**.**
qed(*auto*)
also have ... = ($\int$ *x*. (*dM x*) $*$ (*max 0* (*measure* (*f x*) *A* $-$ $\delta 2$))

− (*exp ε1*) ∗ (*dN x*) ∗ *min 1* ((*exp ε2*)∗ (*measure* (*g x*) *A*)) + (*dM x*) ∗*δ2* ∂(*sum-measure M N*))
by (*auto simp*: *diff-add-eq ring-class.ring-distribs*(*1*))
also have ... ≤ (∫ *x*. (*dM x*) ∗ (*min 1* ((*exp ε2*)∗ (*measure* (*g x*) *A*))) − (*exp ε1*) ∗ (*dN x*) ∗ *min 1* ((*exp ε2*)∗ (*measure* (*g x*) *A*)) + *dM x* ∗*δ2* ∂(*sum-measure M N*))
proof(*rule integral-mono*)
show *integrable* (*sum-measure M N*) (λ*x*. *dM x* ∗ (*min 1* ((*exp ε2*) ∗ *measure* (*g x*) *A*)) − *exp ε1* ∗ *dN x* ∗ *min 1* (*exp ε2* ∗ *measure* (*g x*) *A*) + *dM x* ∗ *δ2*) **by** *auto*
fix *x* **assume** *x* ∈ *space* (*sum-measure M N*)
with *inequalityC* **have** *dM x* ∗ *max 0* (*measure* (*f x*) *A* − *δ2*) ≤ *dM x* ∗ (*min 1* ((*exp ε2*) ∗ *measure* (*g x*) *A*))
by *blast*
thus *dM x* ∗ *max 0* (*measure* (*f x*) *A* − *δ2*) − *exp ε1* ∗ *dN x* ∗ *min 1* (*exp ε2* ∗ *measure* (*g x*) *A*) + *dM x* ∗ *δ2* ≤ *dM x* ∗ *min 1* (*exp ε2* ∗ *measure* (*g x*) *A*) − *exp ε1* ∗ *dN x* ∗ *min 1* (*exp ε2* ∗ *measure* (*g x*) *A*) + *dM x* ∗ *δ2*
by *auto*
qed(*auto*)
also have ... =(∫ *x*. ((*dM x*) − (*exp ε1*) ∗ (*dN x*)) ∗ *min 1* ((*exp ε2*) ∗ (*measure* (*g x*) *A*)) + *dM x* ∗*δ2* ∂(*sum-measure M N*))
by (*auto simp*: *vector-space-over-itself.scale-left-diff-distrib*)
also have ... =(∫ *x*. ((*dM x*) − (*exp ε1*) ∗ (*dN x*)) ∗ *min 1* ((*exp ε2*) ∗ (*measure* (*g x*) *A*))∂(*sum-measure M N*)) +(∫ *x*. *dM x* ∗*δ2* ∂(*sum-measure M N*))
by(*rule Bochner-Integration.integral-add*,*auto*)
finally have ∗: (*measure* (*M* ⋙ *f*) *A*) − *exp* (*ε1* + *ε2*) ∗ (*measure* (*N* ⋙ *g*) *A*)≤(∫ *x*. ((*dM x*) − (*exp ε1*) ∗ (*dN x*)) ∗ *min 1* ((*exp ε2*)∗ (*measure* (*g x*) *A*))∂(*sum-measure M N*)) +(∫ *x*. *dM x* ∗*δ2* ∂(*sum-measure M N*))**.**

have (∫ *x*. *dM x* ∗*δ2* ∂(*sum-measure M N*))=(∫ *x*. *δ2* ∂(*density* (*sum-measure M N*) *dM*))
by(*rule integral-real-density*[*THEN sym*],*auto*)
also have ... =(∫ *x*. *δ2* ∂(*density* (*sum-measure M N*)(*ennreal o dM*)))
by (*meson comp-def*[*THEN sym*])
also have ... =(∫ *x*. *δ2* ∂*M*)
using *dM-density* **by** *auto*
also have ... = *δ2* ∗ *measure M* (*space M*)
by *auto*
also have ... ≤ *δ2*
using *prob-space.prob-space prob-spaceM* **by** *fastforce*
finally have ∗∗: (∫ *x*. *dM x* ∗*δ2* ∂(*sum-measure M N*)) ≤ *δ2***.**

let *?B* = {*x*∈ *space* (*sum-measure M N*). *0* ≤ ((*dM x*) − (*exp ε1*) ∗ (*dN x*))}
have *mble10*: *?B* ∈ *sets* (*sum-measure M N*)

by *measurable*

have ($\int$ x. ((*dM* x) $-$ (*exp* ε*1*) $*$ (*dN* x)) $*$ *min 1* ((*exp* ε*2*)$*$ (*measure* (g x) A))∂(*sum-measure M N*)) $\leq$ ($\int$ $x$$\in$ *?B*. (((*dM* x) $-$ (*exp* ε*1*) $*$ (*dN* x)) $*$ *min 1* ((*exp* ε*2*)$*$ (*measure* (g x) A)))∂(*sum-measure M N*))
proof(*rule integral-drop-negative-part2*)
show (λx. (*dM* x $-$ *exp* ε*1* $*$ *dN* x) $*$ *min 1* (*exp* ε*2* $*$ *measure* (g x) A)) $\in$ *borel-measurable* (*sum-measure M N*)
using *integrablef integrableg* **by** *measurable*
show *integrable* (*sum-measure M N*) (λx. (*dM* x $-$ *exp* ε*1* $*$ *dN* x) $*$ *min 1* (*exp* ε*2* $*$ *measure* (g x) A))
by *auto*
show $\{x \in$ *space* (*sum-measure M N*). *0* $\leq$ *dM* x $-$ *exp* ε*1* $*$ *dN* $x\}$ $\in$ *sets* (*sum-measure M N*)
by *measurable*
show $\{x \in$ *space* (*sum-measure M N*). *0* $<$ (*dM* x $-$ *exp* ε*1* $*$ *dN* x) $*$ *min 1* (*exp* ε*2* $*$ *measure* (g x) A)$\}$ $\subseteq$ $\{x \in$ *space* (*sum-measure M N*). *0* $\leq$ *dM* x $-$ *exp* ε*1* $*$ *dN* $x\}$
proof(*rule subsetI*, *safe*)
fix x **assume** $x \in$ *space* (*sum-measure M N*) **and** *0* $<$ (*dM* x $-$ *exp* ε*1* $*$ *dN* x) $*$ *min 1* (*exp* ε*2* $*$ *measure* (g x) A)
hence *0* $<$ (*dM* x $-$ *exp* ε*1* $*$ *dN* x) $\wedge$ *0* $<$ *min 1* (*exp* ε*2* $*$ *measure* (g x) A)
by (*metis exp-gt-zero lambda-one min.absorb4 min.order-iff mult-pos-pos vector-space-over-itself.scale-zero-right verit-comp-simplify1*(*3*) *zero-less-measure-iff zero-less-mult-pos2*)
thus *0* $\leq$ *dM* x $-$ *exp* ε*1* $*$ *dN* x
by *auto*
qed
show $\{x \in$ *space* (*sum-measure M N*). *0* $\leq$ *dM* x $-$ *exp* ε*1* $*$ *dN* $x\}$ $\subseteq$ $\{x \in$ *space* (*sum-measure M N*). *0* $\leq$ (*dM* x $-$ *exp* ε*1* $*$ *dN* x) $*$ *min 1* (*exp* ε*2* $*$ *measure* (g x) A)$\}$
by *auto*
qed
also have ... $\leq$ ($\int$ $x$$\in$ *?B*. ((*dM* x) $-$ (*exp* ε*1*) $*$ (*dN* x)) ∂(*sum-measure M N*))
using *mble10* **by**(*intro set-integral-mono*,*unfold set-integrable-def*,*auto simp*: *mult-left-le*)
also have ... = ($\int$ $x$$\in$ *?B*. (*dM* x) ∂(*sum-measure M N*)) $-$ ($\int$ $x$$\in$ *?B*. ((*exp* ε*1*) $*$ (*dN* x)) ∂(*sum-measure M N*))
using *mble10* **by**(*intro set-integral-diff*, *auto simp*:*set-integrable-def*)
also have ... = ($\int$ $x$$\in$ *?B*. (*dM* x) ∂(*sum-measure M N*)) $-$ (*exp* ε*1*) $*$ ($\int$ $x$$\in$ *?B*. (*dN* x) ∂(*sum-measure M N*))
by *auto*
also have ... = *measure M ?B* $-$ (*exp* ε*1*) $*$ (*measure N ?B*)
proof$-$
have ($\int$ $x$$\in$ *?B*. *dM* x ∂(*sum-measure M N*)) = ($\int$ x. (*dM* x $*$ *indicator ?B* x) ∂ (*sum-measure M N*))

by (*auto simp*: *mult.commute set-lebesgue-integral-def*)
also have ... = ($\int$ *x. indicator ?B x* ∂(*density* (*sum-measure M N*) *dM*))
by(*rule integral-real-density*[*THEN sym*],*measurable*)
also have ... = ($\int$ *x. indicator ?B x* ∂(*density* (*sum-measure M N*) (*ennreal o dM*)))
by (*metis comp-apply*)
also have ... = ($\int$ *x. indicator ?B x* $*_R$ *1* ∂(*density* (*sum-measure M N*) (*ennreal o dM*)))
by *auto*
also have ... = ($\int$ *x*$\in$ *?B. 1* ∂(*density* (*sum-measure M N*) (*ennreal o dM*)))
by(*auto simp*:*set-lebesgue-integral-def*)
also have ... = *1* $*$ *measure* (*density* (*sum-measure M N*) (*ennreal o dM*)) *?B*
proof(*rule set-integral-indicator*)
show *emeasure* (*density* (*sum-measure M N*) (*ennreal* $\circ$ *dM*)) *?B* $<$ $\top$
using *finite-measure.emeasure-finite*[*of M ?B*,*OF finM*] *diff-gt-0-iff-gt-ennreal* **by** *force*
qed(*measurable*)
also have ... = *1* $*$ *measure M ?B*
using *dM-density* **by** *auto*
finally have *eq11*: ($\int$ *x*$\in$ *?B. dM x* ∂(*sum-measure M N*)) = *measure M ?B*
by *auto*

have ($\int$ *x*$\in$ *?B. dN x* ∂(*sum-measure M N*)) = ($\int$ *x.* (*dN x* $*$ *indicator ?B x*) ∂ (*sum-measure M N*))
by (*auto simp*: *mult.commute set-lebesgue-integral-def*)
also have ... = ($\int$ *x. indicator ?B x* ∂(*density* (*sum-measure M N*) *dN*))
by(*rule integral-real-density*[*THEN sym*],*measurable*)
also have ... = ($\int$ *x. indicator ?B x* ∂(*density* (*sum-measure M N*) (*ennreal o dN*)))
by (*metis comp-apply*)
also have ... = ($\int$ *x*$\in$ *?B. 1* ∂(*density* (*sum-measure M N*) (*ennreal o dN*)))
by(*auto simp*:*set-lebesgue-integral-def*)
also have ... = *1* $*$ *measure* (*density* (*sum-measure M N*) (*ennreal o dN*)) *?B*
proof(*rule set-integral-indicator*,*measurable*)
show *emeasure* (*density* (*sum-measure M N*) (*ennreal* $\circ$ *dN*)) *?B* $<$ $\top$
using *finite-measure.emeasure-finite*[*of N ?B*,*OF finN*] *diff-gt-0-iff-gt-ennreal* **by** *force*
qed
also have ... = *1* $*$ *measure N ?B*
using *dN-density* **by** *auto*

finally have *eq21*: $(\int x \in ?B.\ dN\ x\ \partial(sum\text{-}measure\ M\ N)) = measure\ N\ ?B$
by *auto*

show *?thesis*
using *eq11 eq21* **by** *auto*
qed
also have ... $\leq \delta 1$
using *div1 DP-divergence-forall mble10 setMN-M* **by** *blast*
finally have $***$: $(\int x.\ ((dM\ x) - (exp\ \varepsilon 1) * (dN\ x)) * min\ 1\ ((exp\ \varepsilon 2) * (measure\ (g\ x)\ A))\partial(sum\text{-}measure\ M\ N)) \leq \delta 1$.

show $measure\ (M \gg= f)\ A - exp\ (\varepsilon 1 + \varepsilon 2) * measure\ (N \gg= g)\ A \leq \delta 1 + \delta 2$
using $*$ $**$ $***$ **by** *auto*
qed
qed

5.1.9 Post-processing inequality

corollary *DP-divergence-postprocessing*:
assumes *M*: $M \in space\ (prob\text{-}algebra\ L)$
and *N*: $N \in space\ (prob\text{-}algebra\ L)$
and *f*: $f \in L \rightarrow_M (prob\text{-}algebra\ K)$
and *div1*: $DP\text{-}divergence\ M\ N\ \varepsilon 1 \leq (\delta 1 :: real)$
and $0 \leq \varepsilon 1$
shows $DP\text{-}divergence\ (bind\ M\ f)\ (bind\ N\ f)\ \varepsilon 1 \leq \delta 1$
proof−
have *div2*: $\forall x \in (space\ L).\ DP\text{-}divergence\ (f\ x)\ (f\ x)\ 0 \leq ereal\ 0$
proof
fix *x* **assume** *A0*: $x \in space\ L$
have $DP\text{-}divergence\ (f\ x)\ (f\ x)\ 0 = 0$
by(*rule DP-divergence-reflexivity*)
thus $DP\text{-}divergence\ (f\ x)\ (f\ x)\ 0 \leq ereal\ 0$
by *auto*
qed
have $DP\text{-}divergence\ (M \gg= f)\ (N \gg= f)\ (\varepsilon 1 + 0) \leq ereal\ (\delta 1 + 0)$
by(*rule DP-divergence-composability*[*of M L N f K f* $\varepsilon 1$ $\delta 1$ *0 0*], *auto simp*: *assms div2*)
thus $DP\text{-}divergence\ (M \gg= f)\ (N \gg= f)\ \varepsilon 1 \leq ereal\ \delta 1$ **by** *auto*
qed

5.1.10 Law for the strength of the Giry monad

lemma *DP-divergence-strength*:
assumes $M \in space\ (prob\text{-}algebra\ L)$
and $N \in space\ (prob\text{-}algebra\ L)$
and $DP\text{-}divergence\ M\ N\ \varepsilon 1 \leq (\delta 1 :: real)$
and $0 \leq \varepsilon 1$

and x:$x \in$ *space K*
shows *DP-divergence* ((*return K x*) $\bigotimes_M$ *M*) ((*return K x*) $\bigotimes_M$ *N*) $\varepsilon 1 \leq \delta 1$
proof−
interpret *comparable-probability-measures L M N*
by(*unfold-locales*, *auto simp*: *assms*)
define *f* **where** $f = (\lambda\ y.\ (return\ (K \bigotimes_M L)\ (x,\ y)))$
have *f*: $f \in L \rightarrow_M$ (*prob-algebra* ($K \bigotimes_M L$))
using *assms f-def* **by** *force*
have *1*:*DP-divergence* (*bind M f*) (*bind N f*) $\varepsilon 1 \leq \delta 1$
using *DP-divergence-postprocessing assms f* **by** *blast*
moreover have *2*:(*return K x*) $\bigotimes_M$ $M = (M \gg= f)$
by(*subst Giry-strength-bind-return*[*OF M x*])(*metis f-def return-cong sets-pair-measure-cong sets-return spaceM*)
moreover have *3*:(*return K x*) $\bigotimes_M$ $N = (N \gg= f)$
by(*subst Giry-strength-bind-return*[*OF N x*])(*metis f-def return-cong sets-pair-measure-cong sets-return spaceN*)
ultimately show *DP-divergence* ((*return K x*) $\bigotimes_M$ *M*) ((*return K x*) $\bigotimes_M$ *N*) $\varepsilon 1 \leq \delta 1$
by *auto*
qed

5.1.11 Additivity: law for the double-strength of the Giry monad

lemma *DP-divergence-additivity*:
assumes *M*: $M \in$ *space* (*prob-algebra L*)
and *N*: $N \in$ *space* (*prob-algebra L*)
and *M2*: $M2 \in$ *space* (*prob-algebra K*)
and *N2*: $N2 \in$ *space* (*prob-algebra K*)
and *div1*: *DP-divergence M N* $\varepsilon 1 \leq (\delta 1 :: real)$
and *div2'*: *DP-divergence M2 N2* $\varepsilon 2 \leq (\delta 2 :: real)$
and $0 \leq \varepsilon 1$ **and** $0 \leq \varepsilon 2$
shows *DP-divergence* ($M \bigotimes_M M2$) ($N \bigotimes_M N2$) ($\varepsilon 1 + \varepsilon 2$) $\leq \delta 1 + \delta 2$
proof−
interpret *comparable-probability-measures L M N*
by(*unfold-locales*, *auto simp*: *assms*)
interpret *c2*: *comparable-probability-measures K M2 N2*
by(*unfold-locales*, *auto simp*: *assms*)
have *1*: $M \bigotimes_M M2 = M \gg= (\lambda x.\ M2 \gg= (\lambda y.\ return\ (L \bigotimes_M K)\ (x,\ y)))$
by(*subst return-cong*[*of* $M \bigotimes_M M2$,*symmetric*])(*auto simp*: *pair-prob-space-def pair-sigma-finite-def assms finite-measure-def prob-space-imp-sigma-finite sets-pair-measure intro*!: *pair-prob-space.pair-measure-eq-bind*)
have *2*: $N \bigotimes_M N2 = N \gg= (\lambda x.\ N2 \gg= (\lambda y.\ return\ (L \bigotimes_M K)\ (x,\ y)))$
by(*subst return-cong*[*of* $N \bigotimes_M N2$,*symmetric*])(*auto simp*: *pair-prob-space-def pair-sigma-finite-def assms finite-measure-def prob-space-imp-sigma-finite*

sets-pair-measure intro!: *pair-prob-space.pair-measure-eq-bind*)
have *DP-divergence* $(M \bigotimes_M M2)$ $(N \bigotimes_M N2)$ $(\varepsilon 1 + \varepsilon 2)$ = *DP-divergence* $(M \gg= (\lambda x.\ M2 \gg= (\lambda y.\ return\ (L \bigotimes_M K)\ (x, y))))$ $(N \gg= (\lambda x.\ N2 \gg= (\lambda y.\ return\ (L \bigotimes_M K)\ (x, y))))$ $(\varepsilon 1 + \varepsilon 2)$
using *1 2* **by** *auto*
also have ... $\leq \delta 1 + \delta 2$
proof(*rule DP-divergence-composability*[**where** $L = L$ **and** $K = (L \bigotimes_M K)$])
show $(\lambda x.\ M2 \gg= (\lambda y.\ return\ (L \bigotimes_M K)\ (x, y))) \in L \to_M$ *prob-algebra* $(L \bigotimes_M K)$
and $(\lambda x.\ N2 \gg= (\lambda y.\ return\ (L \bigotimes_M K)\ (x, y))) \in L \to_M$ *prob-algebra* $(L \bigotimes_M K)$
by(*rule measurable-bind-prob-space2*[**where** $N = K$],*auto simp*: *assms*)+
qed(*auto intro*!: *DP-divergence-postprocessing*[**where** $K = L \bigotimes_M K$] *assms measurable-bind-prob-space2*[**where** $N = K$])
finally show *?thesis***.**
qed

5.2 Hypotehsis testing interpretation

5.2.1 Privacy region

definition *DP-region-one-side* :: *real* ⇒ *real* ⇒ (*real* × *real*) *set* **where**
DP-region-one-side ε δ = $\{(x::real, y::real).\ (x - exp(\varepsilon) * y \leq \delta) \wedge 0 \leq x \wedge x \leq 1 \wedge 0 \leq y \wedge y \leq 1\}$

lemma *DP-divergence-region*:
assumes *M*: $M \in$ *space* (*prob-algebra L*)
and *N*: $N \in$ *space* (*prob-algebra L*)
shows ($\forall$ $A \in$ (*sets M*). ((*measure M A*, *measure N A*) $\in$ *DP-region-one-side* ε δ)) $\longleftrightarrow$ *DP-divergence M N* $\varepsilon \leq (\delta :: real)$
proof−
interpret *comparable-probability-measures L M N*
by(*unfold-locales*,*auto simp*: *assms*)
show *?thesis*
by(*subst DP-divergence-forall*[*symmetric*],*unfold DP-region-one-side-def*,*auto intro*!: *prob-space.prob-le-1*)
qed

5.2.2 2-generatedness of *DP-divergence* [1]

lemma *DP-divergence-2-generated-deterministic*:
assumes *M*: $M \in$ *space* (*prob-algebra L*)
and *N*: $N \in$ *space* (*prob-algebra L*)
and $0 \leq \varepsilon$
shows *DP-divergence M N* ε = ($\bigsqcup$ $f \in L \to_M$ (*count-space* (*UNIV* :: *bool set*)). *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) ε)
proof−

interpret *comparable-probability-measures L M N*
by(*unfold-locales*,*auto simp*: *assms*)
show *?thesis*
proof(*intro SUP-eqI*[*THEN sym*])
fix *f* :: *-⇒bool* **assume** *A0*: *Measurable.pred L f*
hence *eq1*: *distr M* (*count-space UNIV*) *f* = *M* ≫= *return* (*count-space UNIV*) ∘ *f*
by (*metis bind-return-distr measurable-cong-sets prob-space.not-empty prob-spaceM spaceM*)
have *eq2*: *distr N* (*count-space UNIV*) *f* = *N* ≫= *return* (*count-space UNIV*) ∘ *f*
by (*metis A0 bind-return-distr measurable-cong-sets prob-space.not-empty prob-spaceN spaceN*)
have *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) *ε* = *DP-divergence* (*M* ≫= *return* (*count-space UNIV*) ∘ *f*) (*N* ≫= *return* (*count-space UNIV*) ∘ *f*) *ε*
using *eq1 eq2* **by** *auto*
also have ... ≤ *DP-divergence M N ε*
by(*rule ereal-le-real*,*auto intro*!: *DP-divergence-postprocessing measurable-comp*[**where** *N* = *count-space UNIV*] *A0 assms measurable-return-prob-space*)
finally show *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) *ε* ≤ *DP-divergence M N ε*.
next
fix *y* :: *ereal* **assume** *A0*: (⋀*f*. *Measurable.pred L f* ⟹ *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) *ε* ≤ *y*)
show *DP-divergence M N ε* ≤ *y*
unfolding *DP-divergence-def*
proof(*subst Sup-le-iff*,*safe*)
fix *i* **assume** *A1*: *i* ∈ *sets M*
hence [*measurable*]: *Measurable.pred L* (λ *x*.(*x* ∈ *i*))
by *auto*
have [*simp*]:(λ*x*. *x* ∈ *i*) −‘ {*True*} = *i*
by *auto*
from *A1 A0* **have** *DP-divergence* (*distr M* (*count-space UNIV*) (λ *x*.(*x* ∈ *i*))) (*distr N* (*count-space UNIV*) (λ *x*.(*x* ∈ *i*))) *ε* ≤ *y*
by *auto*
hence *ineq1*: *measure* (*distr M* (*count-space UNIV*) (λ *x*.(*x* ∈ *i*))) {*True*} − *exp ε* ∗ *measure* (*distr N* (*count-space UNIV*) (λ *x*.(*x* ∈ *i*))) {*True*} ≤ *y*
using *DP-divergence-def*[*of* (*distr M* (*count-space UNIV*) (λ *x*.(*x* ∈ *i*))) (*distr N* (*count-space UNIV*) (λ *x*.(*x* ∈ *i*)))*ε*] **by** (*auto simp*: *Sup-le-iff*)
have *eq1*: *measure* (*distr M* (*count-space UNIV*) (λ *x*.(*x* ∈ *i*))) {*True*} = *measure M i*
using *measure-distr*[*of* (λ *x*.(*x* ∈ *i*))*M* (*count-space UNIV*) {*True*}] *spaceM*(*1*) *A1* **by** *auto*
have *eq2*: *measure* (*distr N* (*count-space UNIV*) (λ *x*.(*x* ∈ *i*)))

$\{True\}$ = *measure N i*
using *measure-distr*[*of* (λ *x*.(*x* $\in$ *i*))*N* (*count-space UNIV*) $\{True\}$] *spaceN*(*1*) *A1* **by** *auto*
from *ineq1* **show** *ereal* (*measure M i* $-$ *exp* ε $*$ *measure N i*) $\leq$ *y*
unfolding *eq1 eq2* **by** *auto*
qed
qed
qed

lemma *DP-divergence-2-generated*:
assumes *M*: *M* $\in$ *space* (*prob-algebra L*)
and *N*: *N* $\in$ *space* (*prob-algebra L*)
assumes $0 \leq \varepsilon$
shows *DP-divergence M N* ε = ($\bigsqcup$ *f* $\in$ *L* $\to_M$ (*prob-algebra* (*count-space* (*UNIV* :: *bool set*))). *DP-divergence* (*M* $\gg=$ *f*) (*N* $\gg=$ *f*) ε)
proof−
interpret *comparable-probability-measures L M N*
by(*unfold-locales*,*auto simp*: *assms*)
show *?thesis*
proof(*subst DP-divergence-2-generated-deterministic*[*OF assms*],*intro order-antisym*)
show ($\bigsqcup$*f*$\in$*L* $\to_M$ *prob-algebra* (*count-space* (*UNIV* :: *bool set*)). *DP-divergence* (*M* $\gg=$ *f*) (*N* $\gg=$ *f*) ε) $\leq$ ($\bigsqcup$*f*$\in$*L* $\to_M$ *count-space* (*UNIV* :: *bool set*). *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) ε)
proof(*subst SUP-le-iff*)
show $\forall$ *i*$\in$*L* $\to_M$ *prob-algebra* (*count-space* (*UNIV* :: *bool set*)). *DP-divergence* (*M* $\gg=$ *i*) (*N* $\gg=$ *i*) ε $\leq$ ($\bigsqcup$*f*$\in$*L* $\to_M$ *count-space* (*UNIV* :: *bool set*). *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) ε)
proof
fix *i* :: - $\Rightarrow$ *bool measure* **assume** *A0*: *i* $\in$ *L* $\to_M$ *prob-algebra* (*count-space UNIV*)
have *DP-divergence* (*M* $\gg=$ *i*) (*N* $\gg=$ *i*) ε $\leq$ *DP-divergence M N* ε
using *DP-divergence-postprocessing A0 assms ereal-le-real* **by** *blast*
also have ... = ($\bigsqcup$*f* :: *'a* $\Rightarrow$ *bool*$\in$*L* $\to_M$ *count-space UNIV*. *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) ε)
using *DP-divergence-2-generated-deterministic assms* **by** *blast*
finally show *DP-divergence* (*M* $\gg=$ *i*) (*N* $\gg=$ *i*) ε $\leq$ ($\bigsqcup$*f* :: *'a* $\Rightarrow$ *bool*$\in$*L* $\to_M$ *count-space UNIV*. *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) ε)**.**
qed
qed
show ($\bigsqcup$*f* :: *'a* $\Rightarrow$ *bool*$\in$*L* $\to_M$ *count-space UNIV*. *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) ε) $\leq$

$(\bigsqcup f :: {'a} \Rightarrow$ *bool measure*$\in L \rightarrow_M$ *prob-algebra* (*count-space UNIV*). *DP-divergence* $(M \gg= f)$ $(N \gg= f)$ $\varepsilon)$
proof(*rule SUP-mono*)
fix f :: -$\Rightarrow$*bool* **assume** *A0*: $f \in L \rightarrow_M$ *count-space UNIV*
show $\exists\, m :: {'a} \Rightarrow$ *bool measure*$\in L \rightarrow_M$ *prob-algebra* (*count-space UNIV*). *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) $\varepsilon \le$ *DP-divergence* $(M \gg= m)$ $(N \gg= m)$ ε
proof(*unfold Bex-def*)
have (((*return* (*count-space UNIV*)) *o f*) $\in L \rightarrow_M$ *prob-algebra* (*count-space UNIV*)) $\wedge$ (*DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) ε = *DP-divergence* ($M \gg=$((*return* (*count-space* (*UNIV* :: *bool set*))) *o f*)) ($N \gg=$((*return* (*count-space* (*UNIV* :: *bool set*))) *o f*)) ε)
proof(*safe*)
show *return* (*count-space UNIV*) $\circ f \in L \rightarrow_M$ *prob-algebra* (*count-space UNIV*)**using** *A0* **by** *auto*
from *A0* **have** *eq1*: (*distr M* (*count-space UNIV*) *f*) = $M \gg=$((*return* (*count-space* (*UNIV* :: *bool set*))) *o f*)
by (*metis bind-return-distr measurable-return1 prob-space.not-empty return-sets-cong spaceM prob-spaceM*)
from *A0* **have** *eq2*: (*distr N* (*count-space UNIV*) *f*) = $N \gg=$((*return* (*count-space* (*UNIV* :: *bool set*))) *o f*)
by (*metis bind-return-distr measurable-cong-sets prob-space.not-empty spaceMN-M spaceMN-N spaceN prob-spaceN*)
from *eq1 eq2* **show** *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) ε = *DP-divergence* ($M \gg=$ *return* (*count-space UNIV*) $\circ f$) ($N \gg=$ *return* (*count-space UNIV*) $\circ f$) ε
by *auto*
qed
thus $\exists\, x :: {'a} \Rightarrow$ *bool measure*. $x \in L \rightarrow_M$ *prob-algebra* (*count-space UNIV*) $\wedge$ *DP-divergence* (*distr M* (*count-space UNIV*) *f*) (*distr N* (*count-space UNIV*) *f*) $\varepsilon \le$ *DP-divergence* $(M \gg= x)$ $(N \gg= x)$ ε
by *auto*
qed
qed
qed
qed

5.3 real version of *DP-divergence*

5.3.1 finiteness

definition *DP-divergence-real* :: *'a measure* $\Rightarrow$ *'a measure* $\Rightarrow$ *real* $\Rightarrow$ *real* **where**
DP-divergence-real M N ε = *Sup* {(*measure M A* − (*exp* ε) ∗ *measure N A*) | *A*::*'a set*. $A \in$ (*sets M*)}

lemma *DP-divergence-real-SUP*:

shows *DP-divergence-real M N* ε = ($\bigsqcup$ (*A*::$'a$ *set*) $\in$ (*sets M*). (*measure M A* $-$ (*exp* ε) $*$ *measure N A*))
by (*auto simp*: *setcompr-eq-image DP-divergence-real-def*)

lemma *DP-divergence-real-leI*:
assumes $\bigwedge$ *A*. *A* $\in$ (*sets M*) $\Longrightarrow$ *measure M A* $\leq$ (*exp* ε) $*$ *measure N A* + (δ :: *real*)
shows *DP-divergence-real M N* $\varepsilon \leq$ (δ :: *real*)
using *assms* **unfolding** *DP-divergence-real-def* **by**(*intro cSup-least*, *fastforce*+)

5.3.2 real version of *DP-divergence*

If M and N are finite then *DP-divergence* and the following version *DP-divergence-real* are the same. However, if M and N are not finite, *DP-divergence-real M N* ε is not well defined. Hence, the latter needs extra assumptions for the finiteness of measures M and N. For example the following lemmas need a kind of finiteness of$M$ and N.

lemma *DP-divergence-is-real*:
assumes *M*: *M* $\in$ *space* (*prob-algebra L*)
and *N*: *N* $\in$ *space* (*prob-algebra L*)
shows *DP-divergence M N* ε = *DP-divergence-real M N* ε
unfolding *DP-divergence-def DP-divergence-real-def*
proof(*subst ereal-Sup*)
interpret *pM*: *prob-space M*
using *M actual-prob-space* **by** *auto*
interpret *pN*: *prob-space N*
using *N actual-prob-space* **by** *auto*
{
fix *A* **assume** *A* $\in$ (*sets M*)
have *pM.prob A* $-$ *exp* ε $*$ *pN.prob A* $\leq$ *pM.prob A*
by *auto*
also have ... $\leq$ *1*
by *auto*
finally have *1*: *pM.prob A* $-$ *exp* ε $*$ *pN.prob A* $\leq$ *1*.

have $-$ *exp* $\varepsilon \leq -$ *exp* ε $*$ *pN.prob A*
by *auto*
also have ... $\leq$ *pM.prob A* $-$ *exp* ε $*$ *pN.prob A*
by *auto*
finally have *2*: $-$ *exp* $\varepsilon \leq$ *pM.prob A* $-$ *exp* ε $*$ *pN.prob A*.
note *1 2*
}note $*$ = *this*

hence $\bigsqcup$ (*ereal* ‘ {*pM.prob A* $-$ *exp* ε $*$ *pN.prob A* |*A*. *A* $\in$ *pM.events*}) $\leq$ *ereal 1*
unfolding *SUP-le-iff* **by** *auto*

moreover from $*$ **have** *ereal* $(-\ exp\ \varepsilon) \leq \bigsqcup$ (*ereal* ‘ $\{pM.prob\ A - exp\ \varepsilon * pN.prob\ A\ |A.\ A \in pM.events\}$)
by(*intro SUP-upper2*, *auto*)
ultimately show $|\bigsqcup$ (*ereal* ‘ $\{pM.prob\ A - exp\ \varepsilon * pN.prob\ A\ |A.\ A \in pM.events\})| \neq \infty$
by *auto*
have $\{ereal\ (pM.prob\ A - exp\ \varepsilon * pN.prob\ A)\ |A.\ A \in pM.events\}$ $=$ *ereal* ‘ $\{pM.prob\ A - exp\ \varepsilon * pN.prob\ A\ |A.\ A \in pM.events\}$
by *blast*
thus $\bigsqcup \{ereal\ (pM.prob\ A - exp\ \varepsilon * pN.prob\ A)\ |A.\ A \in pM.events\}$ $= \bigsqcup$ (*ereal* ‘ $\{pM.prob\ A - exp\ \varepsilon * pN.prob\ A\ |A.\ A \in pM.events\}$)
by *auto*
qed

corollary *DP-divergence-real-forall*:
assumes M: $M \in$ *space* (*prob-algebra* L)
and N: $N \in$ *space* (*prob-algebra* L)
shows ($\forall\ A \in$ (*sets* M). (*measure* $M\ A - (exp\ \varepsilon) *$ *measure* $N\ A \leq (\delta :: real))) \longleftrightarrow$ *DP-divergence-real* $M\ N\ \varepsilon \leq (\delta :: real)$
unfolding *DP-divergence-forall DP-divergence-is-real*[*OF M N*] **by** *auto*

corollary *DP-divergence-real-inequality1*:
assumes M: $M \in$ *space* (*prob-algebra* L)
and N: $N \in$ *space* (*prob-algebra* L)
and $A \in$ (*sets* M) **and** *DP-divergence-real* $M\ N\ \varepsilon \leq (\delta :: real)$
shows *measure* $M\ A \leq (exp\ \varepsilon) *$ *measure* $N\ A + (\delta :: real)$
using *DP-divergence-real-forall*[*OF M N*] *assms*(*3*) *assms*(*4*) **by** *force*

end

theory *Differential-Privacy-Randomized-Response*
imports*Differential-Privacy-Divergence*
begin

6 Randomized Response Mechanism

definition *RR-mechanism* :: *real* $\Rightarrow$ *bool* $\Rightarrow$ *bool pmf*
where *RR-mechanism* $\varepsilon\ x =$
(*if x then* (*bernoulli-pmf* ((*exp* ε) / (*1* + *exp* ε)))
else (*bernoulli-pmf* (*1* / (*1* + *exp* ε))))

lemma *meausrable-RR-mechanism*[*measurable*]:
shows *measure-pmf o* (*RR-mechanism* ε) $\in$ *measurable* (*count-space UNIV*) (*prob-algebra* (*count-space UNIV*))
proof(*rule measurable-prob-algebraI*)
fix x :: *bool*

show *prob-space* ((*measure-pmf* ∘ *RR-mechanism* ε) *x*)
unfolding *RR-mechanism-def* **using** *prob-space-measure-pmf if-split-mem1*
by *auto*
thus *measure-pmf* ∘ *RR-mechanism* ε ∈ *count-space UNIV* $\rightarrow_M$ *subprob-algebra* (*count-space UNIV*)
unfolding *RR-mechanism-def space-measure-pmf*
by (*auto simp*: *measure-pmf-in-subprob-space*)
qed

lemma *RR-probability-flip1*:
fixes ε :: *real*
assumes ε ≥ *0*
shows *1* − *1* / (*1* + *exp* ε) = *exp* ε / (*1* + *exp* ε)
using *add-diff-cancel-left′ add-divide-distrib div-self not-exp-less-zero square-bound-lemma vector-space-over-itself.scale-zero-left*
by *metis*

lemma *RR-probability-flip2*:
fixes ε :: *real*
assumes ε ≥ *0*
shows *1*− *exp* ε / (*1* + *exp* ε) = *1* / (*1* + *exp* ε)
using *RR-probability-flip1 assms*
by *fastforce*

lemma *RR-mechanism-flip*:
assumes ε ≥ *0*
shows *bind-pmf* (*RR-mechanism* ε *x*) (λ *b* :: *bool*. *return-pmf* (¬ *b*)) = (*RR-mechanism* ε (¬ *x*))
unfolding *RR-mechanism-def*
proof−
have *eq1*: *bernoulli-pmf* (*exp* ε / (*1* + *exp* ε)) ⤜ (λ*b*. *return-pmf* (¬ *b*)) = *bernoulli-pmf* (*1* / (*1* + *exp* ε))
proof(*rule pmf-eqI*)
fix *i* :: *bool*
have *pmf* (*bernoulli-pmf* (*exp* ε / ((*1* :: *real*) + *exp* ε)) ⤜ (λ*b* :: *bool*. *return-pmf* (¬ *b*))) *i* = (∫ ⁺*x*. (*pmf* ((λ*b* :: *bool*. *return-pmf* (¬ *b*)) *x*) *i*) ∂(*measure-pmf* (*bernoulli-pmf* (*exp* ε / ((*1* :: *real*) + *exp* ε)))))
using *ennreal-pmf-bind* **by** *metis*
also have ... = *ennreal* (*pmf* (*return-pmf* (¬ *True*)) *i*) ∗ *ennreal* (*exp* ε / (*1* + *exp* ε)) + *ennreal* (*pmf* (*return-pmf* (¬ *False*)) *i*) ∗ *ennreal* (*1* − *exp* ε / (*1* + *exp* ε))
by(*rule nn-integral-bernoulli-pmf* ; *simp-all add*: *add-pos-nonneg*)
also have ... = *ennreal* (*pmf* (*return-pmf True*) *i*) ∗ *ennreal* (*1* − *exp* ε / (*1* + *exp* ε)) + *ennreal* (*pmf* (*return-pmf False*) *i*) ∗ *ennreal* (*1*− (*1* − *exp* ε / (*1* + *exp* ε)))
by *auto*
also have ... = (∫ ⁺*x*. (*pmf* (*return-pmf x*) *i*) ∂*bernoulli-pmf* (*1* − *exp* ε / (*1* + *exp* ε)))

by(*rule nn-integral-bernoulli-pmf*[**where** p =(*1* − *exp* ε / (*1* + *exp* ε)),*THEN sym*] ; *simp-all add*: *add-pos-nonneg*)
also have ... = *pmf* (*bind-pmf* (*bernoulli-pmf* (*1* − *exp* ε / (*1* + *exp* ε))) *return-pmf*) *i*
using *ennreal-pmf-bind* **by** *metis*
also have ... = (*pmf* (*bernoulli-pmf* (*1* − *exp* ε / (*1* + *exp* ε))) *i*)
by (*auto simp*: *bind-return-pmf'*)
also have ... = (*pmf* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *i*)
by (*metis add.commute add-diff-cancel-left' add-divide-distrib div-self not-exp-less-zero square-bound-lemma vector-space-over-itself.scale-zero-left*)
finally show *pmf* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε)) ≫= (λb. *return-pmf* (¬ *b*))) *i* = *pmf* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *i***by** *auto*
qed
have *eq2*: *bernoulli-pmf* (*1* / (*1* + *exp* ε)) ≫= (λb. *return-pmf* (¬ *b*)) = *bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))
proof(*rule pmf-eqI*)
fix *i* :: *bool*
have *pmf* (*bernoulli-pmf* (*1*/ ((*1* :: *real*) + *exp* ε)) ≫= (λb :: *bool*. *return-pmf* (¬ *b*))) *i* = ($\int^+ x$. (*pmf* ((λb :: *bool*. *return-pmf* (¬ *b*)) *x*) *i*) ∂(*measure-pmf* (*bernoulli-pmf* (*1* / ((*1* :: *real*) + *exp* ε)))))
using *ennreal-pmf-bind* **by** *metis*
also have ... = *ennreal* (*pmf* (*return-pmf* (¬ *True*)) *i*) ∗ *ennreal* (*1* / (*1* + *exp* ε)) + *ennreal* (*pmf* (*return-pmf* (¬ *False*)) *i*) ∗ *ennreal* (*1* − *1*/ (*1* + *exp* ε))
by(*rule nn-integral-bernoulli-pmf* ; *simp-all add*: *add-pos-nonneg*)
also have ... = *ennreal* (*pmf* (*return-pmf True*) *i*) ∗ *ennreal* (*exp* ε / (*1* + *exp* ε)) + *ennreal* (*pmf* (*return-pmf False*) *i*) ∗ *ennreal* (*1* − *exp* ε / (*1* + *exp* ε))
by(*auto simp*:*RR-probability-flip1 RR-probability-flip2 assms*)
also have ... = ($\int^+ x$. (*pmf* (*return-pmf x*) *i*) ∂*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε)))
by(*rule nn-integral-bernoulli-pmf*[**where** p =(*exp* ε / (*1* + *exp* ε)),*THEN sym*] ; *simp-all add*: *add-pos-nonneg*)
also have ... = *pmf* (*bind-pmf* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *return-pmf*) *i*
using *ennreal-pmf-bind* **by** *metis*
also have ... = (*pmf* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *i*)
by (*auto simp*: *bind-return-pmf'*)
finally show *pmf* (*bernoulli-pmf* (*1*/ ((*1* :: *real*) + *exp* ε)) ≫= (λb :: *bool*. *return-pmf* (¬ *b*))) *i* = *pmf* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *i***by** *auto*
qed
show (*if x then bernoulli-pmf* (*exp* ε / ((*1* :: *real*) + *exp* ε)) *else bernoulli-pmf* ((*1* :: *real*) / ((*1* :: *real*) + *exp* ε))) ≫= (λb :: *bool*. *return-pmf* (¬ *b*)) =
(*if* ¬ *x then bernoulli-pmf* (*exp* ε / ((*1* :: *real*) + *exp* ε)) *else bernoulli-pmf* ((*1* :: *real*) / ((*1* :: *real*) + *exp* ε)))
using *eq1 eq2* **by** *auto*

qed

proposition *DP-RR-mechanism*:
fixes ε :: *real* **and** *x y* :: *bool*
assumes $\varepsilon > 0$
shows *DP-divergence* (*RR-mechanism* ε *x*) (*RR-mechanism* ε *y*) ε
$\leq$ (*0* :: *real*)
proof(*subst DP-divergence-forall*[*THEN sym*],*unfold RR-mechanism-def*)
{
fix *A* :: *bool set* **assume** $A \in$ *measure-pmf.events* (*if x then bernoulli-pmf* (*exp* ε / ((*1* :: *real*) + *exp* ε)) *else bernoulli-pmf* ((*1* :: *real*) / ((*1* :: *real*) + *exp* ε)))
have *ineq1*: *measure-pmf.prob* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *A* $\leq$ *exp* ε $*$ *measure-pmf.prob* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *A*
by (*auto simp*: *assms leI mult-le-cancel-right1 order-less-imp-not-less*)
have *ineq2*: *measure-pmf.prob* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *A* $\leq$ *exp* ε $*$ *measure-pmf.prob* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *A*
by (*meson assms linorder-not-less measure-nonneg′ mult-le-cancel-right1 nle-le one-le-exp-iff*)
consider $A = \{\}|A = \{True\}|A = \{False\}|A = \{True,False\}$
by (*metis* (*full-types*) *Set.set-insert all-not-in-conv insertI2*)
note *split-A* = *this*
hence *ineq3*: *measure-pmf.prob* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *A* $\leq$ *exp* ε $*$ *measure-pmf.prob* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *A*
proof(*cases*)
case *1*
thus *?thesis* **by** *auto*
next
case *2*
hence *measure-pmf.prob* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *A* = (*exp* ε / (*1* + *exp* ε))
using *pmf-bernoulli-True*
by (*auto simp*: *pmf.rep-eq pos-add-strict*)
also have ... = *exp* ε $*$ (*1* / (*1* + *exp* ε))
by *auto*
also have ... = *exp* ε $*$ *measure-pmf.prob* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *A*
using *pmf-bernoulli-True 2*
by (*auto simp*: *pmf.rep-eq pos-add-strict*)
finally show *?thesis* **by** *auto*
next
case *3*
hence *measure-pmf.prob* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *A* = *1* $-$ (*exp* ε / (*1* + *exp* ε))
using *pmf-bernoulli-False*
by (*auto simp*: *pmf.rep-eq pos-add-strict*)
also have ... = (*1* / (*1* + *exp* ε))

using *RR-probability-flip2 assms* **by** *auto*
also have ... $\leq$ *exp* ε $*$ (*exp* ε / (*1* + *exp* ε))
by (*metis* (*no-types*, *opaque-lifting*) *add-le-same-cancel1 assms divide-right-mono dual-order.trans less-add-one linorder-not-less mult-exp-exp nle-le one-le-exp-iff times-divide-eq-right*)
also have ... = *exp* ε $*$ (*measure-pmf.prob* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *A*)
proof−
have (*measure-pmf.prob* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *A*) = *1* − (*1* / (*1* + *exp* ε))
using *pmf-bernoulli-False 3*
by (*auto simp*: *pmf.rep-eq pos-add-strict*)
also have ... = *exp* ε / (*1* + *exp* ε)
using *RR-probability-flip1 assms* **by** *auto*
finally show *?thesis* **by** *auto*
qed
thus *?thesis*
using *calculation* **by** *auto*
next
case *4*
hence *A* = *UNIV*
by *auto*
thus *?thesis*
using *assms* **by** *fastforce*
qed
from *split-A*
have *ineq4*: *measure-pmf.prob* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *A* $\leq$ *exp* ε $*$ *measure-pmf.prob* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *A*
proof(*cases*)
case *1*
thus *?thesis*
by *auto*
next
case *2*
hence *measure-pmf.prob* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *A* = (*1* / (*1* + *exp* ε))
using *pmf-bernoulli-True*
by (*auto simp*: *pmf.rep-eq pos-add-strict*)
also have ... $\leq$ *exp* ε $*$ (*exp* ε / (*1* + *exp* ε))
by (*metis* (*no-types*, *opaque-lifting*) *add-le-same-cancel1 assms divide-right-mono dual-order.trans less-add-one linorder-not-less mult-exp-exp nle-le one-le-exp-iff times-divide-eq-right*)
also have ... = *exp* ε $*$ *measure-pmf.prob* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *A*
using *pmf-bernoulli-True 2*
by (*auto simp*: *pmf.rep-eq pos-add-strict*)
finally show *?thesis*
by *auto*

next
case *3*
hence *measure-pmf.prob* (*bernoulli-pmf* (*1* / (*1* + *exp* ε))) *A* = *1* − (*1* / (*1* + *exp* ε))
using *pmf-bernoulli-False*
by (*auto simp*: *pmf.rep-eq pos-add-strict*)
also have ... = (*exp* ε / (*1* + *exp* ε))
using *RR-probability-flip1 assms* **by** *auto*
also have ... = *exp* ε ∗ (*1* / (*1* + *exp* ε))
by *auto*
also have ... = *exp* ε ∗ (*measure-pmf.prob* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *A*)
proof−
have (*measure-pmf.prob* (*bernoulli-pmf* (*exp* ε / (*1* + *exp* ε))) *A*) = *1* − (*exp* ε / (*1* + *exp* ε))
using *pmf-bernoulli-False 3*
by (*auto simp*: *pmf.rep-eq pos-add-strict*)
also have ... = *1* / (*1* + *exp* ε)
using *RR-probability-flip2 assms* **by** *auto*
finally show *?thesis* **by** *auto*
qed
thus *?thesis*
using *calculation* **by** *auto*
next
case *4*
hence *A* = *UNIV*
by *auto*
thus *?thesis*
using *assms* **by** *fastforce*
qed
note *ineq1 ineq2 ineq3 ineq4*
}
thus∀ *A* :: *bool set*∈*measure-pmf.events* (*if x then bernoulli-pmf* (*exp* ε / ((*1* :: *real*) + *exp* ε)) *else bernoulli-pmf* ((*1* :: *real*) / ((*1* :: *real*) + *exp* ε))). *measure-pmf.prob* (*if x then bernoulli-pmf* (*exp* ε / ((*1* :: *real*) + *exp* ε)) *else bernoulli-pmf* ((*1* :: *real*) / ((*1* :: *real*) + *exp* ε))) *A* − *exp* ε ∗ *measure-pmf.prob* (*if y then bernoulli-pmf* (*exp* ε / ((*1* :: *real*) + *exp* ε)) *else bernoulli-pmf* ((*1* :: *real*) / ((*1* :: *real*) + *exp* ε))) *A* ≤ (*0* :: *real*)
by *auto*
qed

end

theory *Differential-Privacy-Laplace-Mechanism*
imports*Differential-Privacy-Divergence*
Laplace-Distribution
begin

7 Laplace mechanism

7.1 Laplace mechanism as a noise-adding mechanism

7.1.1 Laplace distribution with scale b and average μ

definition *Lap-dist* :: *real* ⇒ *real* ⇒ *real measure* **where**
Lap-dist b μ = (*if b* ≤ *0 then return borel μ else density lborel* (*laplace-density b μ*))

lemma shows *prob-space-Lap-dist*[*simp*,*intro*]: *prob-space* (*Lap-dist b x*)
and *subprob-space-Lap-dist*[*simp*,*intro*]: *subprob-space* (*Lap-dist b x*)
and *sets-Lap-dist*[*measurable-cong*]: *sets* (*Lap-dist b x*) = *sets borel*
and *space-Lap-dist*: *space* (*Lap-dist ε x*) = *UNIV*
using *prob-space-laplacian-density prob-space-return*[*of x borel*] *prob-space-imp-subprob-space sets-eq-imp-space-eq Lap-dist-def*
by(*cases b* ≤ *0*,*auto*)

lemma *measurable-Lap-dist*[*measurable*]:
shows *Lap-dist b* ∈ *borel* $\rightarrow_M$ *prob-algebra borel*
proof(*casesb* ≤ *0*)
case *True*
then show *?thesis* **unfolding** *Lap-dist-def* **by** *auto*
next
case *False*
thus *?thesis*
proof(*intro measurable-prob-algebraI*)
show ⋀*x*. *x* ∈ *space borel* ⟹ *prob-space* (*Lap-dist b x*)
by *auto*
show *Lap-dist b* ∈ *borel* $\rightarrow_M$ *subprob-algebra borel*
proof(*rule measurable-subprob-algebra-generated*[**where** Ω=*UNIV* **and** *G*=(*range* (λ *a* :: *real*. {..*a*}))])
show *pow1*: (*range* (λ *a* :: *real*. {..*a*})) ⊆ *Pow UNIV*
by *auto*
show *sets borel* = *sigma-sets UNIV* (*range* (λ *a* :: *real*. {..*a*}))
using *borel-eq-atMost* **by** (*metis pow1 sets-measure-of*)
show *Int-stable* (*range* (λ *a* :: *real*. {..*a*}))
by(*subst Int-stable-def*,*auto*)
show ⋀*a*. *a* ∈ *space borel* ⟹ *subprob-space* (*Lap-dist b a*)
by (*auto simp*: *prob-space-imp-subprob-space*)
show ⋀*a*. *a* ∈ *space borel* ⟹ *sets* (*Lap-dist b a*) = *sets borel*
by (*auto simp*: *sets-Lap-dist*)
show ⋀*A*. *A* ∈ *range atMost* ⟹ (λ*a*. *emeasure* (*Lap-dist b a*) *A*) ∈ *borel* $\rightarrow_M$ *borel*
proof(*safe*,*unfold Lap-dist-def*)
fix *x* :: *real* **assume** *x* ∈ *UNIV*

have (λa :: *real. emeasure* (*density lborel* (λx :: *real. ennreal* (*laplace-density b a x*))) {..*x*}) = (λa. *laplace-CDF b a x*)
using *emeasure-laplace-density False* **by** *auto*
also have ... $\in$ *borel* $\rightarrow_M$ *borel*
using *borel-measurable-laplace-CDF2* **by** *auto*
finally show (λa :: *real. emeasure* (*if b* $\leq$ (*0* :: *real*) *then return borel a else density lborel* (λx :: *real. ennreal* (*laplace-density b a x*))) {..*x*}) $\in$ *borel* $\rightarrow_M$ *borel*
using *False* **by** *auto*
qed
have (λa. *emeasure* (*Lap-dist b a*) *UNIV*) = (λa. *1*)
using *Lap-dist-def emeasure-laplace-density-mass-1* **by** *auto*
thus (λa. *emeasure* (*Lap-dist b a*) *UNIV*) $\in$ *borel* $\rightarrow_M$ *borel*
by *auto*
qed
qed
qed

lemma *Lap-dist-space-prob-algebra*[*simp*,*measurable*]:
shows (*Lap-dist b x*) $\in$ *space* (*prob-algebra borel*)
by (*metis iso-tuple-UNIV-I measurable-Lap-dist measurable-space space-borel*)

lemma *Lap-dist-space-subprob-algebra*[*simp*,*measurable*]:
shows (*Lap-dist b x*) $\in$ *space* (*subprob-algebra borel*)
by (*metis UNIV-I measurable-Lap-dist measurable-prob-algebraD measurable-space space-borel*)

7.1.2 The Laplace distribution with scale *b* and average *0*

definition *Lap-dist0 b* $\equiv$ *Lap-dist b 0*

Actually *Lap-dist b* is a noise-addition of Laplace distribution with scale *b* and average *0*.

lemma shows *prob-space-Lap-dist0*[*simp*,*intro*,*measurable*]: *prob-space* (*Lap-dist0 b*)
and *subprob-space-Lap-dist0*[*simp*,*intro*,*measurable*]: *subprob-space* (*Lap-dist0 b*)
and *sets-Lap-dist0*[*measurable-cong*]: *sets* (*Lap-dist0 b*) = *sets borel*
and *space-Lap-dist0*: *space* (*Lap-dist0 b*) = *UNIV*
and *Lap-dist0-space-prob-algebra*[*simp*,*measurable*]: (*Lap-dist0 b*) $\in$ *space* (*prob-algebra borel*)
and *Lap-dist0-space-subprob-algebra*[*simp*,*measurable*]: (*Lap-dist0 b*) $\in$ *space* (*subprob-algebra borel*)
unfolding *Lap-dist0-def* **by**(*auto simp*:*sets-Lap-dist space-Lap-dist*)

lemma *Lap-dist-def2*:

shows (*Lap-dist b x*) = *do*{*r* ← *Lap-dist0 b*; *return borel* (*x* + *r*)}
proof−
show *Lap-dist b x* = *Lap-dist0 b* ⤜ (*λr*. *return borel* (*x* + *r*))
proof(*casesb* ≤ *0*)
case *True*
hence *Lap-dist b x* = *return borel x*
by(*auto simp*: *Lap-dist-def*)
also have ... = *return borel 0* ⤜ (*λr*. *return borel* (*x* + *r*))
by(*subst bind-return*, *measurable*)
also have ... = *do*{*r* ← *Lap-dist0 b*; *return borel* (*x* + *r*)}
by(*auto simp*: *Lap-dist0-def Lap-dist-def True*)
finally show *?thesis*.
next
case *False*
hence (*Lap-dist b x*) = *density lborel* (*laplace-density b x*)
by(*auto simp*: *Lap-dist-def*)
also have ... = *density* (*distr lborel borel* ((+) *x*)) (*laplace-density b x*)
by(*auto simp*: *Lebesgue-Measure.lborel-distr-plus*[*of x*])
also have ... = (*density lborel* (*laplace-density b 0*)) ⤜ (*λr*. *return borel* (*x* + *r*))
proof(*subst bind-return-distr'*)
show *density* (*distr lborel borel* ((+) *x*)) (*λxa*. *ennreal* (*laplace-density b x xa*)) =
distr (*density lborel* (*λx*. *ennreal* (*laplace-density b 0 x*))) *borel* ((+) *x*)
proof(*subst density-distr*)
show *distr* (*density lborel* (*λxa*. *ennreal* (*laplace-density b x* (*x* + *xa*)))) *borel* ((+) *x*) =
distr (*density lborel* (*λx*. *ennreal* (*laplace-density b 0 x*))) *borel* ((+) *x*)
using *laplace-density-shift*[*of b 0*::*real x*] **by** (*auto simp*: *add.commute*)
qed(*auto*)
qed(*auto*)
also have ... = *do*{*r* ← *Lap-dist0 b*; *return borel* (*x* + *r*)}
by(*auto simp*: *Lap-dist-def Lap-dist0-def False*)
finally show *?thesis*.
qed
qed

corollary *Lap-dist-shift*:
shows (*Lap-dist b* (*x* + *y*)) = *do*{*r* ← *Lap-dist b x*; *return borel* (*y* + *r*)}
unfolding *Lap-dist-def2*
by(*subst bind-assoc*[**where** *N* = *borel* **and** *R* = *borel*],*auto simp*: *bind-return*[**where** *N* = *borel*] *Lap-dist0-def ac-simps*)

7.1.3 Differential Privacy of Laplace noise addition

proposition *DP-divergence-Lap-dist'*:
assumes $b > 0$
and $|\ x - y\ | \le r$
shows *DP-divergence* (*Lap-dist b x*) (*Lap-dist b y*) $(r\ /\ b) \le (0 ::$ *real*)
proof(*casesr* = *0*)
case *True*
hence *0*:$x = y$
using *assms*(*2*) **by** *auto*
have *DP-divergence* (*Lap-dist b x*) (*Lap-dist b y*) $(r\ /\ b) \le (0 ::$ *real*)
by (*auto simp*: *DP-divergence-reflexivity True 0*)
thus *?thesis* **by** *argo*
next
case *False*
hence *posr*: $r > 0$**using** *assms*(*2*) **by** *auto*
show *DP-divergence* (*Lap-dist b x*) (*Lap-dist b y*) $(r\ /\ b) \le$ *ereal 0*
proof(*intro DP-divergence-leI*,*unfold Lap-dist-def*)
fix *A* :: *real set* **assume** $A \in$ *sets* (*if* $b \le 0$ *then return borel x else density lborel* ($\lambda s.$ *ennreal* (*laplace-density b x s*)))
hence *A*[*measurable*]: $A \in$ *sets borel*
using *assms* **by** (*metis* (*mono-tags*, *lifting*) *sets-density sets-lborel sets-return*)
have *pos*[*simp*]: $b > 0$
using *posr assms* **by** *auto*
hence *nonneg*[*simp*]: $b \ge 0$
by *argo*
have *Sigma-Algebra.measure* (*if* $b \le 0$ *then return borel x else density lborel* ($\lambda xa.$ *ennreal* (*laplace-density b x xa*))) $A =$
(*Sigma-Algebra.emeasure* (*density lborel* ($\lambda xa.$ *ennreal* (*laplace-density b x xa*))) *A*)
using *pos* **by**(*split if-split*, *intro conjI impI*, *linarith*) (*metis emeasure-eq-ennreal-measure emeasure-laplace-density-mass-1 emeasure-mono ennreal-one-less-top leD pos sets-density sets-lborel space-in-borel top-greatest*)
also have ... $= (\int^{+} z.$ *ennreal* (*laplace-density b x z*) $*$ *indicator A z* ∂*lborel*)
by(*rule emeasure-density*,*auto intro*: *A*)
also have ... $\le (\int^{+} z.$ (*exp* $(r\ /\ b)$) $*$ *ennreal* (*laplace-density b y z*) $*$ *indicator A z* ∂*lborel*)
proof(*rule nn-integral-mono*)
fix *z* :: *real* **assume** *z*: $z \in$*space lborel*
have *0*: $-|z - x| \le r - |z - y|$
using *assms*(*2*) *posr* **by** *auto*
hence *1*: *exp* $(-\ |z - x|\ /\ b\ ) \le$*exp* $((\ r - |z - y\ |)/\ b)$
by(*subst exp-le-cancel-iff*, *intro divide-right-mono*, *auto*)
have *ennreal* (*laplace-density b x z*) $=$ *exp* $(-\ |z - x|\ /\ b\ )\ /\ (2 * b)$
by(*auto simp*: *laplace-density-def z*)

also have ... $\leq$ *exp* ((*r* − |*z* − *y* |) / *b*) / (*2* ∗ *b*)
proof(*subst ennreal-le-iff*)
show *0* $\leq$ *exp* ((*r* − |*z* − *y*|) / *b*) / (*2* ∗ *b*)
by *auto*
show *exp* (− |*z* − *x*| / *b*) / (*2* ∗ *b*) $\leq$ *exp* ((*r* − |*z* − *y*|) / *b*) / (*2* ∗ *b*)
by(*subst divide-right-mono*, *rule 1*,*auto*)
qed
also have ... = *exp* (− |*z* − *y* |/ *b* + *r* / *b*) / (*2* ∗ *b*)
by *argo*
also have ... = (*exp* (− |*z* − *y* |/ *b*) ∗ *exp* (*r* / *b*)) / (*2* ∗ *b*)
using *exp-add*[*of* − |*z* − *y* |/ *b* *r* / *b*] **by** *auto*
also have ... = *exp* (*r* / *b*) ∗ (*exp* (− |*z* − *y* |/ *b*)) / (*2* ∗ *b*)
by *argo*
also have ... = *exp* (*r* / *b*) ∗ (*laplace-density b y z*)
by(*auto simp*: *laplace-density-def*)
finally have *ennreal* (*laplace-density b x z*) $\leq$ *ennreal* (*exp* (*r* / *b*) ∗ *laplace-density b y z*) **.**

thus *ennreal* (*laplace-density b x z*) ∗ *indicator A z* $\leq$ *ennreal* (*exp* (*r* / *b*)) ∗ *ennreal* (*laplace-density b y z*) ∗ *indicator A z*
by (*simp add*: *ennreal-mult′ mult-right-mono*)
qed
also have ... = ($\int^+$ *z*. (*exp* (*r* / *b*)) ∗ (*ennreal* (*laplace-density b y z*) ∗ *indicator A z*) ∂*lborel*)
by (*auto simp*: *mult.assoc*)
also have ... = (*exp* (*r* / *b*)) ∗ ($\int^+$ *z*. *ennreal* (*laplace-density b y z*) ∗ *indicator A z* ∂*lborel*)
by(*rule nn-integral-cmult*, *auto*)
also have ... = (*exp* (*r* / *b*)) ∗ (*Sigma-Algebra.emeasure* (*density lborel* (λ*xa*. *ennreal* (*laplace-density b y xa*))) *A*)
by(*subst emeasure-density*,*auto intro*: *A*)
also have ... = *ennreal* (*exp* (*r* / *b*)) ∗ *ennreal* (*Sigma-Algebra.measure* (*if b* $\leq$ *0 then return borel y else density lborel* (λ*xa*. *ennreal* (*laplace-density b y xa*))) *A*)
using *pos* **by**(*split if-split*, *intro conjI impI*, *linarith*)(*metis emeasure-eq-ennreal-measure emeasure-laplace-density-mass-1 emeasure-mono ennreal-one-less-top leD pos sets-density sets-lborel space-in-borel top-greatest*)
also have ... = (*exp* (*r* / *b*)) ∗ (*Sigma-Algebra.measure* (*if b* $\leq$ *0 then return borel y else density lborel* (λ*xa*. *ennreal* (*laplace-density b y xa*))) *A*)
by (*auto simp*: *ennreal-mult″*)
finally show *measure* (*if b* $\leq$ *0 then return borel x else density lborel* (λ*xa*. *ennreal* (*laplace-density b x xa*))) *A*
$\leq$ *exp* (*r* / *b*) ∗ *measure* (*if b* $\leq$ *0 then return borel y else density lborel* (λ*x*. *ennreal* (*laplace-density b y x*))) *A* + *0*
by *auto*
qed
qed

corollary *DP-divergence-Lap-dist′-eps*:
  assumes $\varepsilon > 0$
    and $| x - y | \leq r$
  shows *DP-divergence* (*Lap-dist* (r / ε) x) (*Lap-dist* (r / ε) y) $\varepsilon \leq$ (*0* :: *real*)
proof(*cases* $r = 0$)
  case *True*
  with *assms* **have** *1*: $x = y$ **and** *2*: (r / ε) $= 0$
    by *auto*
  thus *?thesis*
    unfolding *1 2* **using** *DP-divergence-reflexivity′*[*of* ε *Lap-dist 0 y*] *assms* **by** *auto*
next
  case *False*
  with *assms* **have** *0*: $0 < r$ **and** *1*: $0 < (r / \varepsilon)$
    by *auto*
  show *?thesis*
    using *0 DP-divergence-Lap-dist′*[*OF 1 assms*(*2*)] **by** *auto*
qed

corollary *DP-divergence-Lap-dist-eps*:
  assumes $\varepsilon > 0$
    and $| x - y | \leq 1$
  shows *DP-divergence* (*Lap-dist* (*1* / ε) x) (*Lap-dist* (*1* / ε) y) $\varepsilon \leq$ (*0* :: *real*)
  using *DP-divergence-Lap-dist′-eps*[*of* ε x y *1*] *assms* **by** *auto*

end

theory *Differential-Privacy-Laplace-Mechanism-Multi*
  imports *List-Space*/*List-Space*
    Differential-Privacy-Laplace-Mechanism
begin

7.2 Bundled Laplace Noise

lemma *space-listM-borel-UNIV*[*measurable*,*simp*]:
  shows *space* (*listM borel*) = *UNIV*
  unfolding *space-listM* **by** *auto*

context
  fixes b::*real*
begin

The list of Laplace distribution with scale b and average *0* **primrec** *Lap-dist0-list* :: *nat* ⇒ (*real list*) *measure* **where**
  Lap-dist0-list 0 = *return* (*listM borel*) [] |

Lap-dist0-list (*Suc n*) = *do*{*x1* ← (*Lap-dist0 b*); *x2* ← (*Lap-dist0-list n*); *return* (*listM borel*) (*x1* # *x2*)}

A function adding Laplace noise to each element of a real list **primrec** *Lap-dist-list* :: *real list* ⇒ (*real list*) *measure* **where**
Lap-dist-list [] = *return* (*listM borel*) []|
Lap-dist-list (*x* # *xs*) = *do*{*x1* ← (*Lap-dist b x*); *x2* ← (*Lap-dist-list xs*); *return* (*listM borel*) (*x1* # *x2*)}

lemma *measurable-Lap-dist-list*[*measurable*]:
shows *Lap-dist-list* ∈ *listM borel* $\rightarrow_M$ *prob-algebra* (*listM borel*)
proof−
have *1*: (*return* (*listM borel*) []) ∈ *space* (*prob-algebra* (*listM borel*))
by (*metis UNIV-I lists-UNIV measurable-return-prob-space measurable-space space-borel space-listM*)
show *?thesis*
unfolding *Lap-dist-list-def* **using** *1 measurable-rec-list'''* **by** *measurable*
qed

lemma *prob-space-Lap-dist-list*[*measurable*,*simp*]:
shows *prob-space* (*Lap-dist-list xs*)
using *space-listM-borel-UNIV measurable-Lap-dist-list measurable-space*[*of Lap-dist-list*] *actual-prob-space* **by** *blast*

lemma *Laprepsp'*[*measurable*,*simp*]:
shows *Lap-dist-list xs* ∈ *space* (*prob-algebra* (*listM borel*))
by (*metis UNIV-I measurable-Lap-dist-list measurable-space space-listM-borel-UNIV*)

lemma *Laprepsp*[*measurable*,*simp*]:
shows *Lap-dist-list xs* ∈ *space* (*subprob-algebra* (*listM borel*))
by (*simp add*: *prob-space-imp-subprob-space space-prob-algebra-sets space-subprob-algebra*)

lemma *sets-Lap-dist-list*[*measurable-cong*]:
shows ⋀*zs*. *sets*(*Lap-dist-list zs*) = *sets*(*listM borel*)
by (*simp add*: *space-prob-algebra-sets*)

lemma *space-Lap-dist-list*:
shows ⋀*zs*. *space*(*Lap-dist-list zs*) = *space* (*listM borel*)
by (*simp add*: *sets-Lap-dist-list sets-eq-imp-space-eq*)

lemma *emeasure-Lap-dist-list-length*:
shows *emeasure* (*Lap-dist-list ys*) {*xs*. *length xs* = *length ys*} = *1*
proof(*induction ys*)
case *Nil*
have *1*: {*xs*. *length xs* = *length* []} = {[]}
by *auto*
have *2*: {[]} ∈ *sets* (*listM borel*)

using *sets-listM-length*[*of borel0*] **unfolding** *space-listM* **by** *auto*
thus *?case* **by**(*auto simp*: *2 1*)
next
case (*Cons a ys*)
note [*simp*] = *sets-Lap-dist-list sets-Lap-dist space-pair-measure space-Lap-dist*
have [*measurable*]: *return* (*Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys*) $\in$ *borel* $\bigotimes_M$ *listM borel* $\rightarrow_M$ *subprob-algebra* (*borel* $\bigotimes_M$ *listM borel*)
by (*metis* (*mono-tags*, *opaque-lifting*) *sets-Lap-dist-list return-measurable return-sets-cong sets-Lap-dist sets-pair-measure-cong*)
have *Lap-dist-list* (*a* # *ys*) = *Lap-dist b a* $\gg=$ (λ*x1*. *Lap-dist-list ys* $\gg=$ (λ*x2*. *return* (*listM borel*) (*x1* # *x2*)))
unfolding *Lap-dist-list.simps*(*2*) **by** *auto*
also have ... = ((*Lap-dist b a*) $\bigotimes_M$ (*Lap-dist-list ys*)) $\gg=$ (λ(*x1*,*x2*). *return* (*listM borel*) (*x1* # *x2*))
proof(*subst pair-prob-space.pair-measure-eq-bind*)
show *Lap-dist b a* $\gg=$ (λ*x1*. *Lap-dist-list ys* $\gg=$ (λ*x2*. *return* (*listM borel*) (*x1* # *x2*))) = *Lap-dist b a* $\gg=$ (λ*x*. *Lap-dist-list ys* $\gg=$ (λ*y*. *return* (*Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys*) (*x*, *y*))) $\gg=$ (λ(*x1*, *x2*). *return* (*listM borel*) (*x1* # *x2*))
proof(*subst bind-assoc*)
show *Lap-dist b a* $\gg=$ (λ*x1*. *Lap-dist-list ys* $\gg=$ (λ*x2*. *return* (*listM borel*) (*x1* # *x2*))) = *Lap-dist b a* $\gg=$ (λ*x*. *Lap-dist-list ys* $\gg=$ (λ*y*. *return* (*Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys*) (*x*, *y*)) $\gg=$ (λ(*x1*, *x2*). *return* (*listM borel*) (*x1* # *x2*)))
by(*subst bind-assoc*,*measurable*)(*subst bind-return*[**where** *N* =(*listM borel*)],*auto simp*: *sets-eq-imp-space-eq*)
show (λ(*x1*, *x2*). *return* (*listM borel*) (*x1* # *x2*)) $\in$ *borel* $\bigotimes_M$ *listM borel* $\rightarrow_M$ *subprob-algebra* (*listM borel*)
by *measurable*
show (λ*x*. *Lap-dist-list ys* $\gg=$ (λ*y*. *return* (*Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys*) (*x*, *y*))) $\in$ *Lap-dist b a* $\rightarrow_M$ *subprob-algebra* (*borel* $\bigotimes_M$ *listM borel*)
proof(*subst measurable-bind*)
show (λ*x*. *return* (*Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys*) (*fst x*, *snd x*)) $\in$ *Lap-dist b a* $\bigotimes_M$ (*Lap-dist-list ys*) $\rightarrow_M$ *subprob-algebra* (*borel* $\bigotimes_M$ *listM borel*)
by(*rule measurable-compose*,*measurable*)
show (λ*x*. *Lap-dist-list ys*) $\in$ *Lap-dist b a* $\rightarrow_M$ *subprob-algebra* (*Lap-dist-list ys*)
by (*auto simp*: *prob-space.M-in-subprob*)
qed(*auto*)
qed
show *pair-prob-space* (*Lap-dist b a*) (*Lap-dist-list ys*)
unfolding *pair-prob-space-def pair-sigma-finite-def* **using** *prob-space-Lap-dist prob-space-Lap-dist-list*
by (*auto simp*: *prob-space-imp-sigma-finite*)
qed
finally have *eq1*: *Lap-dist-list* (*a* # *ys*) = *Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys* $\gg=$ (λ(*x1*, *x2*). *return* (*listM borel*) (*x1* # *x2*)).

show *emeasure* (*Lap-dist-list* (*a* # *ys*)) {*xs. length xs* = *length* (*a* # *ys*)} = (*1* :: *ennreal*)**unfolding** *eq1*
proof(*subst Giry-Monad.emeasure-bind emeasure-return*)
show ($\int^+$ *x. emeasure* (*case x of* (*x1*, *x2*) ⇒ (*return* (*listM borel*) (*x1* # *x2*))) {*xs.* (*length xs*) = (*length* (*a* # *ys*))} ∂((*Lap-dist b a*) $\bigotimes_M$ (*Lap-dist-list ys*))) = *1*
proof(*subst pair-sigma-finite.nn-integral-snd*[*THEN sym*])
show ($\int^+$ *y.* $\int^+$ *x. emeasure* (*case* (*x*, *y*) *of* (*x1*, *x2*) ⇒ *return* (*listM borel*) (*x1* # *x2*)) {*xs. length xs* = *length* (*a* # *ys*)} ∂*Lap-dist b a* ∂*Lap-dist-list ys*) = *1*
proof−
{
fix *y* :: *real list*
have ($\int^+$ *x. emeasure* (*case* (*x*, *y*) *of* (*x1*, *x2*) ⇒ *return* (*listM borel*) (*x1* # *x2*)) {*xs. length xs* = *length* (*a* # *ys*)} ∂*Lap-dist b a*) = ($\int^+$ *x. emeasure* (*return* (*listM borel*) (*x* # *y*)) {*xs. length xs* = *length* (*a* # *ys*)} ∂*Lap-dist b a*)**by** *auto*
also have ... = ($\int^+$ *x. indicator* {*x. length* (*x* # *y*) = *length* (*a* # *ys*)} *x* ∂*Lap-dist b a*)
proof(*subst emeasure-return*)
show ⋀*x.* {*xs. length xs* = *length* (*a* # *ys*)} ∈ *sets* (*listM borel*)
by (*metis* (*no-types*, *lifting*) *Collect-cong iso-tuple-UNIV-I sets-listM-length space-listM-borel-UNIV*)
show ($\int^+$ *x. indicator* {*xs. length xs* = *length* (*a* # *ys*)} (*x* # *y*) ∂*Lap-dist b a*) = ($\int^+$ *x. indicator* {*x. length* (*x* # *y*) = *length* (*a* # *ys*)} *x* ∂*Lap-dist b a*)
by(*rule nn-integral-cong*,*auto*)
qed
also have ... = ($\int^+$ *x. indicator* {*x. length y* = *length ys*} *x* ∂*Lap-dist b a*)
by *auto*
also have ... = *indicator* {*z. length z* = *length ys*} *y*
proof(*split split-indicator*,*intro conjI impI*)
show *y* ∉ {*z. length z* = *length ys*} ⟹ *integral*^{N} (*Lap-dist b a*) (*indicator* {*x. length y* = *length ys*}) = *0*
by *auto*
show *y* ∈ {*z. length z* = *length ys*} ⟹ *integral*^{N} (*Lap-dist b a*) (*indicator* {*x. length y* = *length ys*}) = *1*
using *prob-space-Lap-dist prob-space.emeasure-space-1* **by** *fastforce*
qed
finally have ($\int^+$ *x. emeasure* (*case* (*x*, *y*) *of* (*x1*, *x2*) ⇒ *return* (*listM borel*) (*x1* # *x2*)) {*xs. length xs* = *length* (*a* # *ys*)} ∂*Lap-dist b a*) = *indicator* {*z. length z* = *length ys*} *y*.
}**note** ∗ = *this*

hence ($\int^+$ *y.* $\int^+$ *x. emeasure* (*case* (*x*, *y*) *of* (*x1*, *x2*) ⇒ *return*

(*listM borel*) (*x1* # *x2*)) {*xs*. *length xs* = *length* (*a* # *ys*)} ∂*Lap-dist* *b a* ∂*Lap-dist-list ys*) = ($\int^{+}$ *y*. *indicator* {*z*. *length z* = *length ys*} *y* ∂*Lap-dist-list ys*)
by *auto*
also have ... = *emeasure* (*Lap-dist-list ys*) {*z*. *length z* = *length ys*}
by(*subst nn-integral-indicator*)(*use sets-listM-length*[*of borel-length ys*] **in** *auto*)
also have ... = *1*
by(*auto simp*:*Cons.IH*)
finally show *?thesis*.
qed
show *pair-sigma-finite* (*Lap-dist b a*) (*Lap-dist-list ys*)
by (*auto simp*: *pair-sigma-finite-def prob-space-imp-subprob-space subprob-space-imp-sigma-finite*)

have (λ*x*. *emeasure* (*case x of* (*x1*, *x2*) ⇒ *return* (*listM borel*) (*x1* # *x2*)) {*xs*. *length xs* = *length* (*a* # *ys*)}) = (λ*x*. *emeasure x* {*xs*. *length xs* = *length* (*a* # *ys*)}) *o* (λ*x*. (*case x of* (*x1*, *x2*) ⇒ *return* (*listM borel*) (*x1* # *x2*)))
by *auto*
also have ... ∈ *borel-measurable* (*Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys*)
proof(*intro measurable-comp*)
show (λ*x*. *case x of* (*x1*, *x2*) ⇒ *return* (*listM borel*) (*x1* # *x2*)) ∈ *Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys* $\rightarrow_M$ *subprob-algebra*(*listM borel*)
by *measurable*
show (λ*x*. *emeasure x* {*xs*. *length xs* = *length* (*a* # *ys*)}) ∈ *borel-measurable* (*subprob-algebra* (*listM borel*))
by(*rule measurable-emeasure-subprob-algebra*)(*use sets-listM-length*[*of borellength* (*a*#*ys*)] **in** *auto*)
qed
finally show (λ*x*. *emeasure* (*case x of* (*x1*, *x2*) ⇒ *return* (*listM borel*) (*x1* # *x2*)) {*xs*. *length xs* = *length* (*a* # *ys*)}) ∈ *borel-measurable* (*Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys*).
qed
show *space* (*Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys*) ≠ {}
by (*auto simp*: *prob-space.not-empty prob-space-pair*)
show {*xs*. *length xs* = *length* (*a* # *ys*)} ∈ *sets* (*listM borel*)
by (*use sets-listM-length*[*of borellength* (*a*#*ys*)] **in** *auto*)
show (λ(*x1*, *x2*). *return* (*listM borel*) (*x1* # *x2*)) ∈ *Lap-dist b a* $\bigotimes_M$ *Lap-dist-list ys* $\rightarrow_M$ *subprob-algebra* (*listM borel*)
by *measurable*
qed
qed

lemma *Lap-dist0-list-Lap-dist-list*:
shows *Lap-dist-list* (*replicate n 0*) = *Lap-dist0-list n*
by(*induction n*,*auto simp*: *Lap-dist0-def*)

corollary
shows *prob-space-Lap-dist0-list*[*measurable*,*simp*]: *prob-space* (*Lap-dist0-list* *n*)
and *prob-algebra-Lap-dist0-list*[*measurable*,*simp*]: *Lap-dist0-list n* $\in$ *space* (*prob-algebra* (*listM borel*))
and *subprob-algebra-Lap-dist0-list*[*measurable*,*simp*]: *Lap-dist0-list* *n* $\in$ *space* (*subprob-algebra* (*listM borel*))
and *sets-Lap-dist0-list*[*measurable-cong*]:*sets*(*Lap-dist0-list n*) = *sets*(*listM borel*)
by(*auto simp add*: *Lap-dist0-list-Lap-dist-list*[*symmetric*] *sets-Lap-dist-list*)

corollary *emeasure-Lap-dist0-list-length*:
shows *emeasure* (*Lap-dist0-list n*) {*xs. length xs* = *n*} = *1*
using *emeasure-Lap-dist-list-length*[*of* (*replicate n 0*)]
by(*auto simp*: *Lap-dist0-list-Lap-dist-list*[*symmetric*])

lemma *Lap-dist-list-def2*:
shows *Lap-dist-list xs* = *do*{*ys* ← (*Lap-dist0-list* (*length xs*)); *return* (*listM borel*) (*map2* (+) *xs ys*)}
proof(*induction xs*)
case *Nil*
thus *?case* **unfolding** *Lap-dist-list.simps*(*1*) *list.map*(*1*) *zip-Nil* **by**(*subst Giry-Monad.bind-const′*,*auto intro*!: *prob-space-return subprob-space-return simp*: *space-listM*)
next
case (*Cons a xs*)
note [*measurable del*] = *borel-measurable-count-space*

have [*simp*]: (*replicate* (*length xs*) *0*) $\in$ *space*(*listM borel*)
unfolding *space-listM* **by** *auto*
have *4*[*measurable*]: *sets* (*Lap-dist b 0* $\gg=$ (λx. *return borel* (*a* + *x*))) = *sets borel*
unfolding *Lap-dist-shift*[*symmetric*] *sets-Lap-dist* **by** *auto*
have *5*[*measurable*]: *sets* (*Lap-dist-list* (*replicate* (*length xs*) *0*)) = *sets* (*listM borel*)
by(*simp add*:*sets-Lap-dist-list*)
hence [*measurable*]: (*map2* (+) (*a#xs*)) $\in$ *listM borel* $\rightarrow_M$ *listM borel*
by *measurable*
have [*simp*]: *prob-space* (*Lap-dist b 0* $\gg=$ (λx. *return borel* (*a* + *x*)))
by(*rule prob-space-bind′*[*of - borel - borel*],*auto*)

have *Lap-dist-list* (*a* # *xs*) = *Lap-dist b a* $\gg=$ ($\lambda x1$. *Lap-dist0-list* (*length xs*) $\gg=$ (λys. *return* (*listM borel*) (*map2* (+) *xs ys*)) $\gg=$ ($\lambda x2$. *return* (*listM borel*) (*x1* # *x2*)))
unfolding *Lap-dist-list.simps*(*2*) *Cons* **by** *auto*
also have ... = *Lap-dist b 0* $\gg=$ (λx. *return borel* (*a* + *x*)) $\gg=$ ($\lambda x1$. *Lap-dist0-list* (*length xs*) $\gg=$ (λys. *return* (*listM borel*) (*map2* (+) *xs*

ys)) $\gg=$ ($\lambda x2$. *return* (*listM borel*) (*x1* # *x2*)))
  unfolding *Lap-dist-shift*[*of b 0 a*,*simplified*] **by** *auto*
 also have ... = *Lap-dist b 0* $\gg=$ (λx. *return borel* (*a* + *x*)) $\gg=$ ($\lambda x1$. *Lap-dist0-list* (*length xs*) $\gg=$ (λys. *return* (*listM borel*) (*map2* (+) *xs ys*) $\gg=$ ($\lambda x2$. *return* (*listM borel*) (*x1* # *x2*))))
 proof(*subst bind-assoc*[*THEN sym*])
  show $\bigwedge x1$. ($\lambda x2$. *return* (*listM borel*) (*x1* # *x2*)) $\in$(*listM borel*) $\to_M$ *subprob-algebra* (*listM borel*)
   by *measurable*
  have *sets* (*Lap-dist0-list* (*length xs*)) = *sets* (*listM borel*)
   by(*auto simp*: *sets-Lap-dist0-list*)
  thus(λys. *return* (*listM borel*) (*map2* (+) *xs ys*)) $\in$ *Lap-dist0-list* (*length xs*) $\to_M$ *subprob-algebra* (*listM borel*)
   by *measurable*
 qed(*auto*)
 also have ... = *Lap-dist b 0* $\gg=$ (λx. *return borel* (*a* + *x*)) $\gg=$ ($\lambda x1$. *Lap-dist0-list* (*length xs*) $\gg=$ (λys. *return* (*listM borel*) (*x1* # (*map2* (+) *xs ys*))))
  by(*subst bind-return*[**where** *N* =*listM borel*],*auto*)
 also have ... = *Lap-dist0-list* (*length xs*) $\gg=$ (λy. *Lap-dist b 0* $\gg=$ (λx. *return borel* (*a* + *x*)) $\gg=$ (λx. *return* (*listM borel*) (*x* # *map2* (+) *xs y*)))
 proof(*subst pair-prob-space.bind-rotate*[**where** *N* =*listM borel*])
  show *pair-prob-space* (*Lap-dist b 0* $\gg=$ (λx. *return borel* (*a* + *x*))) (*Lap-dist0-list*(*length xs*))
    by (*auto simp*: *prob-space-imp-sigma-finite pair-prob-space-def pair-sigma-finite-def*)
  have *6*: *sets* ((*Lap-dist b 0* $\gg=$ (λx. *return borel* (*a* + *x*))) $\bigotimes_M$ *Lap-dist0-list* (*length xs*)) = *sets*(*borel* $\bigotimes_M$ *listM borel*)
   using *4 5* **by**(*auto simp*: *Lap-dist0-list-Lap-dist-list*)
   show (λ(*x*, *y*). *return* (*listM borel*) (*x* # *map2* (+) *xs y*)) $\in$ (*Lap-dist b 0* $\gg=$ (λx. *return borel* (*a* + *x*))) $\bigotimes_M$ *Lap-dist0-list* (*length xs*) $\to_M$ *subprob-algebra* (*listM borel*)
  by(*subst measurable-cong-sets*[*OF 6*],*auto simp*: *Lap-dist0-list-Lap-dist-list*)
 qed(*auto*)
 also have ... = *Lap-dist0-list* (*length xs*) $\gg=$ (λy. *Lap-dist b 0* $\gg=$ (λx. (*return borel* (*a* + *x*)) $\gg=$ (λx. *return* (*listM borel*) (*x* # *map2* (+) *xs y*))))
  by(*subst bind-assoc*,*measurable*)
 also have ... = *Lap-dist0-list* (*length xs*) $\gg=$ (λy. *Lap-dist b 0* $\gg=$ (λx. (*return* (*listM borel*) ((*a* + *x*) # *map2* (+) *xs y*))))
  by(*subst bind-return*,*measurable*)
 also have ... = *Lap-dist b 0* $\gg=$ (λx. *Lap-dist0-list* (*length xs*) $\gg=$ (λys. (*return* (*listM borel*) (*map2* (+) (*a*#*xs*) (*x*#*ys*)))))
  by(*subst pair-prob-space.bind-rotate*[**where** *N* =*listM borel*],*auto simp*: *prob-space-imp-sigma-finite pair-prob-space-def pair-sigma-finite-def*)
 also have ... = *Lap-dist b 0* $\gg=$ (λx. *Lap-dist0-list* (*length xs*) $\gg=$ (λzs. (*return* (*listM borel*)(*x* # *zs*)) $\gg=$ (λys. (*return* (*listM borel*) (*map2* (+) (*a*#*xs*) (*ys*))))))

by(*subst bind-return*[**where** *N* =*listM borel*],*auto*)
also have ... = (*Lap-dist b 0* $\gg=$ (λx. *Lap-dist0-list* (*length xs*) $\gg=$ (λzs. (*return* (*listM borel*)(*x* # *zs*))) $\gg=$ (λys. (*return* (*listM borel*) (*map2* (+) (*a*#*xs*) (*ys*))))))
by(*subst bind-assoc*[*THEN sym*],*measurable*)
also have ... = *Lap-dist0-list* (*length* (*a* # *xs*)) $\gg=$ (λys. *return* (*listM borel*) (*map2* (+) (*a* # *xs*) *ys*))
unfolding *Lap-dist-list.simps*(*2*) *length-Cons Lap-dist0-list.simps*(*2*) *Lap-dist0-def*
by(*subst bind-assoc*[*THEN sym*],*measurable*)
finally show *?case* **by** *auto*
qed

end

lemma *sum-list-cons*:
fixes *xs ys* :: *'a list* **and** *n* :: *nat*
assumes *length xs* = *n* **and** *length ys* = *n*
shows ($\sum$ *i*$\in${*1*..*Suc n*}. *d* (*nth* (*x* # *xs*) (*i*−*1*)) (*nth* (*y* # *ys*) (*i*−*1*))) = *d x y* + ($\sum$ *i*$\in${*1*..*n*}. *d* (*nth xs* (*i*−*1*)) (*nth ys* (*i*−*1*)))
proof−
have *0*: {*0*..*n*} = {*0*..*0*} $\cup$ {*1*..*n*}
by *auto*
show ($\sum$ *i* = *1*..*Suc n*. *d* (*nth* (*x* # *xs*) (*i*−*1*)) (*nth* (*y* # *ys*) (*i*−*1*))) = *d x y* + ($\sum$(*i* :: *nat*)$\in${*1*..*n*}. *d* (*nth* (*xs*) (*i*−*1*)) (*nth* (*ys*) (*i*−*1*)))
by(*subst sum.reindex-bij-witness*[*of* {*1*..*Suc n*} *Suc* λi. *i* − *1* {*0*..*n*}],*auto simp*: *0*)
qed

lemma *adj-Cons-partition*:
fixes *xs ys* :: *real list* **and** *n* :: *nat* **and** *r* :: *real*
assumes *length xs* = *n* **and** *length ys* = *n*
and *adj*: ($\sum$ *i*$\in${*1*..*Suc n*}. $|$ *nth* (*x* # *xs*) (*i*−*1*) − *nth* (*y* # *ys*) (*i*−*1*) $|$) $\leq$ *r*
and *posr*: *r* $\geq$ *0*
obtains *r1 r2*::*real* **where** *0* $\leq$ *r1* **and** *0* $\leq$ *r2* **and** $|$*x* − *y*$|$ $\leq$ *r1* **and** ($\sum$ *i* = *1*..*n*. $|$*xs* ! (*i* − *1*) − *ys* ! (*i* − *1*)$|$) $\leq$ *r2* **and** *r1* + *r2* $\leq$ *r*
proof−
have ($\sum$ *i* = *1*..*Suc n*. $|$(*x* # *xs*) ! (*i* − *1*) − (*y* # *ys*) ! (*i* − *1*)$|$) = $|$ *x* − *y*$|$ + ($\sum$(*i* :: *nat*)$\in${*1*..*n*}. $|$*xs* ! (*i*−*1*) − *ys* ! (*i*−*1*)$|$)
using *sum-list-cons*[*OF assms*(*1*,*2*)] **by** *auto*
with *adj* **have** $|$*x* − *y*$|$ + ($\sum$ *i* = *1*..*n*. $|$*xs* ! (*i* − *1*) − *ys* ! (*i* − *1*)$|$) $\leq$ *r* **and** *0* $\leq$ $|$*x* − *y*$|$ **and** *0* $\leq$ ($\sum$ *i* = *1*..*n*. $|$*xs* ! (*i* − *1*) − *ys* ! (*i* − *1*)$|$)
by *auto*
thus ($\bigwedge$*r1 r2*. *0* $\leq$ *r1* $\Longrightarrow$ *0* $\leq$ *r2* $\Longrightarrow$ $|$*x* − *y*$|$ $\leq$ *r1* $\Longrightarrow$ ($\sum$ *i* = *1*..*n*. $|$*xs* ! (*i* − *1*) − *ys* ! (*i* − *1*)$|$) $\leq$ *r2* $\Longrightarrow$ *r1* + *r2* $\leq$ *r* $\Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*

by *blast*
qed

lemma *adj-list-0*:
fixes *xs ys* :: *real list* **and** *n* :: *nat*
shows *length xs* = *n* $\Longrightarrow$ *length ys* = *n* $\Longrightarrow$ ($\sum$ *i*$\in${*1..n*}. | *nth xs* (*i*−*1*) − *nth ys* (*i*−*1*) |) $\leq$ *0* $\Longrightarrow$ *xs* = *ys*
proof(*induction xs arbitrary*: *ys n*)
case *Nil*
thus *?case* **by** *auto*
next
case (*Cons x xs2*)
then obtain *m y ys2* **where** *n*: *n* = *Suc m* **and** *ys*: *ys* = (*y* # *ys2*) **and** *xs2*: *length xs2* = *m* **and** *ys2*: *length ys2* = *m*
by (*metis length-Suc-conv*)
hence ∗: ($\sum$ *i*$\in${*1..Suc m*}. | *nth* (*x* # *xs2*) (*i*−*1*) − *nth* (*y* # *ys2*) (*i*−*1*) |) $\leq$ *0*
using *Cons.prems*(*3*) **unfolding** *ys n* **by** *auto*
then obtain *r1 r2*::*real* **where** *r1*: *0* $\leq$ *r1* **and** *r2*: *0* $\leq$ *r2* **and** *r1a*: |*x* − *y*| $\leq$ *r1* **and** *r2a*: ($\sum$*i* = *1..m*. |*xs2* ! (*i* − *1*) − *ys2* ! (*i* − *1*)|) $\leq$ *r2* **and** *req*: *r1* + *r2* $\leq$ *0*
using *adj-Cons-partition*[*of xs2 m ys2 x y 0*, *OF xs2 ys2* ∗] **by** *auto*
hence *r1* = *0* **and** *r2* = *0* **and** *x* = *y* **and** *xs2* = *ys2*
using *Cons.IH*[*of m ys2*, *OF xs2 ys2*] *r2a r1a* **by** *auto*
thus *x* # *xs2* = *ys* **unfolding** *ys* **by** *auto*
qed

lemma *DP-Lap-dist-list*:
fixes *xs ys* :: *real list* **and** *n* :: *nat* **and** *r* :: *real* **and** *b*::*real*
assumes *posb*: *b* > (*0* :: *real*)
and *length xs* = *n* **and** *length ys* = *n*
and *adj*: ($\sum$ *i*$\in${*1..n*}. | *nth xs* (*i*−*1*) − *nth ys* (*i*−*1*) |) $\leq$ *r*
and *posr*: *r* $\geq$ *0*
shows *DP-divergence* (*Lap-dist-list b xs*) (*Lap-dist-list b ys*) (*r* / *b*) $\leq$ *0*
using *assms*
proof(*induction xs arbitrary*: *ys n r b*)
case *Nil*
hence *n* = *0* **and** *ys* = []
by *auto*
hence *1*: *Lap-dist-list b ys* = (*Lap-dist-list b* [])
by *auto*
have *2*: *0* $\leq$ *r* / *b*
using *Nil* **by** *auto*
have *DP-divergence* (*Lap-dist-list b* []) (*Lap-dist-list b ys*) (*r*/*b*) $\leq$ *DP-divergence* (*Lap-dist-list b* []) (*Lap-dist-list b* []) *0*
by(*unfold 1*, *rule DP-divergence-monotonicity*)
(*auto simp*: *prob-space-return space-prob-algebra 2*)

then show *?case*
by (*simp add*: *DP-divergence-reflexivity*)
next
case (*Cons x xs2*)
then obtain *y ys2 m* **where** *n*: *n* = *Suc m* **and** *ys*: *ys* = *y* # *ys2*
and *xs2*: *length xs2* = *m* **and** *ys2*: *length ys2* = *m*
by (*auto simp*: *length-Suc-conv*)
then obtain *r1 r2*::*real*
where *r1*: $0 \le r1$
and *r2*: $0 \le r2$
and *r1a*: $|x - y| \le r1$
and *r2a*: $(\sum i = 1..m.\ |xs2\ !\ (i - 1) - ys2\ !\ (i - 1)|) \le r2$
and *req*: $r1 + r2 \le r$
using *adj-Cons-partition*[*of xs2 m ys2 x y r*] *Cons* **by** *blast*
have *DPr2*: *DP-divergence* (*Lap-dist-list b xs2*) (*Lap-dist-list b ys2*)
(*r2* / *b*) ≤ *0*
by (*auto simp*: *Cons.IH*[*of b m ys2 r2*, *OF Cons.prems*(*1*) *xs2 ys2*
r2a r2])
have *DPr1*: *DP-divergence* (*Lap-dist b x*) (*Lap-dist b y*) (*r1* / *b*) ≤
0
by (*auto simp*: *DP-divergence-Lap-dist′*[*of b x y r1* , *OF Cons.prems*(*1*)
r1a] *zero-ereal-def*)
have *r1* / *b* + *r2* / *b* ≤ *r* / *b*
by (*metis Cons.prems*(*1*) *add-divide-distrib divide-right-mono leI*
order-trans-rules(*20*) *req*)
hence *DP-divergence* (*Lap-dist-list b* (*x* # *xs2*)) (*Lap-dist-list b ys*)
(*r* / *b*) ≤ *DP-divergence* (*Lap-dist-list b* (*x* # *xs2*)) (*Lap-dist-list b ys*)
(*r1* / *b* + *r2* / *b*)
by(*intro DP-divergence-monotonicity Laprepsp′*) *auto*
also have ... ≤ *ereal* (*0* + *0*)
unfolding *ys Lap-dist-list.simps*(*2*)
proof(*rule DP-divergence-composability*[*of Lap-dist b x borel Lap-dist*
b y - - - *r1* / *b 0 r2* / *b 0*])
show *Lap-dist b x* ∈ *space* (*prob-algebra borel*)
and *Lap-dist b y* ∈ *space* (*prob-algebra borel*)
and (*λx1*. *Lap-dist-list b xs2* ≫= (*λx2*. *return* (*listM borel*) (*x1*
x2))) ∈ *borel* $\rightarrow_M$ *prob-algebra* (*listM borel*)
and (*λx1*. *Lap-dist-list b ys2* ≫= (*λx2*. *return* (*listM borel*) (*x1*
x2))) ∈ *borel* $\rightarrow_M$ *prob-algebra* (*listM borel*)
by *auto*
show ∀ *x*∈*space borel*.
DP-divergence (*Lap-dist-list b xs2* ≫= (*λx2*. *return* (*listM borel*) (*x*
x2))) (*Lap-dist-list b ys2* ≫= (*λx2*. *return* (*listM borel*) (*x* # *x2*)))
(*r2* / *b*) ≤ *ereal 0*
proof
fix *z*::*real* **assume** *z* ∈ *space borel*
have *DP-divergence* (*Lap-dist-list b xs2* ≫= (*λx2*. *return* (*listM*
borel) (*z* # *x2*))) (*Lap-dist-list b ys2* ≫= (*λx2*. *return* (*listM borel*) (*z*
x2))) (*r2* / *b* + *0*) ≤ *ereal* (*0* + *0*)

proof(*rule DP-divergence-composability*[*of Lap-dist-list b xs2 listM borel Lap-dist-list b ys2 - - - r2 / b 0*])
show *Lap-dist-list b xs2* ∈ *space* (*prob-algebra* (*listM borel*))
and *Lap-dist-list b ys2* ∈ *space* (*prob-algebra* (*listM borel*))
and (*λx2. return* (*listM borel*) (*z # x2*)) ∈ *listM borel* $\rightarrow_M$ *prob-algebra* (*listM borel*)
and (*λx2. return* (*listM borel*) (*z # x2*)) ∈ *listM borel* $\rightarrow_M$ *prob-algebra* (*listM borel*)
and (*0*::*real*) ≤ *0*
by *auto*
show *DP-divergence* (*Lap-dist-list b xs2*) (*Lap-dist-list b ys2*) (*r2 / b*) ≤ *ereal 0*
using *DPr2* **by**(*auto simp*: *zero-ereal-def*)
show ∀ *x*∈*space* (*listM borel*). *DP-divergence* (*return* (*listM borel*) (*z # x*)) (*return* (*listM borel*) (*z # x*)) *0* ≤ *ereal 0*
by (*auto simp*: *DP-divergence-reflexivity order.refl zero-ereal-def*)
show *0* ≤ *r2 / b*
by (*auto simp*: *Cons.prems*(*1*) *r2 divide-nonneg-pos*)
qed
thus *DP-divergence* (*Lap-dist-list b xs2* ⤜ (*λx2. return* (*listM borel*) (*z # x2*))) (*Lap-dist-list b ys2* ⤜ (*λx2. return* (*listM borel*) (*z # x2*))) (*r2 / b*) ≤ *ereal 0*
by *auto*
qed
show *DP-divergence* (*Lap-dist b x*) (*Lap-dist b y*) (*r1 / b*) ≤ *ereal 0*
using *DPr1* **by**(*auto simp*: *zero-ereal-def*)
show *0* ≤ *r1 / b*
and *0* ≤ *r2 / b*
by (*auto simp*: *Cons.prems*(*1*) *r1 r2 divide-nonneg-pos*)
qed
also have ... = *0*
by *auto*
finally show *DP-divergence* (*Lap-dist-list b* (*x # xs2*)) (*Lap-dist-list b ys*) (*r / b*) ≤ *0* **.**
qed

corollary *DP-Lap-dist-list-eps*:
fixes *xs ys* :: *real list* **and** *n* :: *nat* **and** *r* :: *real*
assumes *pose*: *ε* > (*0* :: *real*)
and *length xs* = *n* **and** *length ys* = *n*
and *adj*: ($\sum$ *i*∈{*1..n*}. | *nth xs* (*i*−*1*) − *nth ys* (*i*−*1*) |) ≤ *r*
and *posr*: *r* ≥ *0*
shows *DP-divergence* (*Lap-dist-list* (*r / ε*) *xs*) (*Lap-dist-list* (*r / ε*) *ys*) *ε* ≤ *0*
proof(*cases r* = *0*)
case *True*
with *assms adj-list-0* **have** ∗: *xs* = *ys*
by *blast*

have *nne*: $0 \leq \varepsilon$
using *pose* **by** *auto*
then show *?thesis*
unfolding $*$ **using** *DP-divergence-reflexivity'*[*of* ε *Lap-dist-list* (r / ε) *ys*] **by** *auto*
next
case *False*
with *posr* **have** *posr'*: $0 < r$
by *auto*
note $* =$ *DP-Lap-dist-list*[*of* (r / ε) *xs n ys r* , *OF* - *assms*(2,3,4) *posr*]
from *posr' pose* **have** *1*: $r / (r / \varepsilon) = \varepsilon$ **and** *2*: $0 < r / \varepsilon$
by *auto*
then show *?thesis* **using** $*$[*OF 2*] **unfolding** *1* **by** *blast*
qed

hide-fact(**open**) *adj-Cons-partition sum-list-cons adj-list-0*

end

theory *Differential-Privacy-Standard*
imports
Differential-Privacy-Laplace-Mechanism-Multi
L1-norm-list
HOL.Transitive-Closure
begin

8 Formalization of Differential privacy

AFDP in this section means the textbook "The Algorithmic Foundations of Differential Privacy"written by Cynthia Dwork and Aaron Roth.

8.1 Predicate of Differential privacy

The inequality for DP (cf. [Def 2.4, AFDP]) **definition** *DP-inequality*:: *'a measure* $\Rightarrow$ *'a measure* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *bool***where**
DP-inequality M N ε δ $\equiv$ ($\forall$ *A* $\in$ *sets M*. *measure M A* $\leq$ (*exp* ε) $*$ *measure N A* + δ)

The divergence for DP (cf. [Barthe & Olmedo, ICALP 2013]) **proposition** *DP-inequality-cong-DP-divergence*:
shows *DP-inequality M N* ε δ $\longleftrightarrow$ *DP-divergence M N* ε $\leq \delta$
by(*auto simp*: *DP-divergence-forall*[*THEN sym*] *DP-inequality-def*)

corollary *DP-inequality-zero*:
assumes *M* $\in$ *space* (*prob-algebra L*)

and $N \in$ *space* (*prob-algebra* L)
and *DP-inequality* M N 0 0
shows $M = N$
using *assms DP-divergence-zero* **unfolding** *DP-inequality-cong-DP-divergence*
by *auto*

Definition of the standard differential privacy with adjacency, and its basic properties We first we abstract the domain of database and the adjacency relations. later we instantiate them according to the textbook.

definition *differential-privacy* :: $('a \Rightarrow 'b\ measure) \Rightarrow ('a\ rel) \Rightarrow real \Rightarrow real \Rightarrow bool$ **where**
differential-privacy M *adj* ε $\delta \equiv \forall (d1,d2) \in adj.$ *DP-inequality* (M $d1$) (M $d2$) ε δ $\wedge$ *DP-inequality* (M $d2$) (M $d1$) ε δ

lemma *differential-privacy-adj-sym*:
assumes *sym adj*
shows *differential-privacy* M *adj* ε $\delta \longleftrightarrow (\forall\ (d1,d2) \in adj.$ *DP-inequality* (M $d1$) (M $d2$) ε δ)
using *assms* **by**(*auto simp*: *differential-privacy-def sym-def*)

lemma *differential-privacy-symmetrize*:
assumes *differential-privacy* M *adj* ε δ
shows *differential-privacy* M $(adj \cup adj^{-1})$ ε δ
using *assms* **unfolding** *differential-privacy-def* **by** *blast*

lemma *differential-privacy-restrict*:
assumes *differential-privacy* M *adj* ε δ
and $adj' \subseteq adj$
shows *differential-privacy* M adj' ε δ
using *assms* **unfolding** *differential-privacy-def* **by** *blast*

Lemmas for group privacy (cf. [Theorem 2.2, AFDP])
lemma *pure-differential-privacy-comp*:
assumes $adj1 \subseteq$ (*space* X) $\times$ (*space* X)
and $adj2 \subseteq$ (*space* X) $\times$ (*space* X)
and *differential-privacy* M *adj1* $\varepsilon 1$ 0
and *differential-privacy* M *adj2* $\varepsilon 2$ 0
and $M{:}M \in X \rightarrow_M$ (*prob-algebra* R)
shows *differential-privacy* M (*adj1 O adj2*) $(\varepsilon 1 + \varepsilon 2)$ 0
using *assms* **unfolding** *differential-privacy-def*
proof(*intro ballI*)
note [*measurable*] = M
fix x **assume** *a1*: $\forall (d1,\ d2) \in adj1.$ *DP-inequality* (M $d1$) (M $d2$) $\varepsilon 1$ 0 $\wedge$ *DP-inequality* (M $d2$) (M $d1$) $\varepsilon 1$ 0

and *a2*: ∀(*d2*, *d3*)∈ *adj2*. *DP-inequality* (*M d2*) (*M d3*) *ε2 0* ∧ *DP-inequality* (*M d3*) (*M d2*) *ε2 0*
and *x* ∈ (*adj1 O adj2*)
then obtain *d1 d2 d3*
where *x*: *x* = (*d1*,*d3*)**and** *a*: (*d1*,*d2*) ∈*Restr adj1* (*space X*) **and** *b*: (*d2*,*d3*) ∈*Restr adj2* (*space X*)
using *assms* **by** *blast*
then have *DP-inequality* (*M d1*) (*M d2*) *ε1 0 DP-inequality* (*M d2*) (*M d1*) *ε1 0 DP-inequality* (*M d2*) (*M d3*) *ε2 0 DP-inequality* (*M d3*) (*M d2*) *ε2 0*
using *a1 a2* **by** *auto*
moreover have *M d1* ∈ *space* (*prob-algebra R*) *M d2* ∈ *space* (*prob-algebra R*) *M d3* ∈ *space* (*prob-algebra R*)
using *a b* **by**(*auto intro*!: *measurable-space*[*OF M*])
ultimately have *1*: *DP-inequality* (*M d1*) (*M d3*) (*ε1* + *ε2*) *0* **and** *DP-inequality* (*M d3*) (*M d1*) (*ε2* + *ε1*) *0*
unfolding *DP-inequality-cong-DP-divergence zero-ereal-def*[*symmetric*] **by**(*intro DP-divergence-transitivity*[*of* - (*M d2*)],*auto*)+
hence *DP-inequality* (*M d3*) (*M d1*) (*ε1* + *ε2*) *0*
by *argo*
with *1* **show** *case x of* (*d1*, *d2*) ⇒ *DP-inequality* (*M d1*) (*M d2*) (*ε1* + *ε2*) *0* ∧ *DP-inequality* (*M d2*) (*M d1*) (*ε1* + *ε2*) *0*
unfolding *x case-prod-beta* **by** *auto*
qed

lemma *adj-trans-k*:
assumes *adj* ⊆ *A* × *A*
and *0* < *k*
shows (*adj* ^^ *k*) ⊆ *A* × *A*
using *assms*(*2*) **by** (*induction k*)(*use assms*(*1*) **in** *fastforce*)+

lemma *pure-differential-privacy-trans-k*:
assumes *adj* ⊆ (*space X*) × (*space X*)
and *differential-privacy M adj ε 0*
and *M*[*measurable*]:*M* ∈ *X* $\rightarrow_M$ (*prob-algebra R*)
shows *differential-privacy M* (*adj* ^^ *k*) (*k* ∗ *ε*) *0*
proof(*induction k*)
case *0*
then show *?case*
by(*auto simp*: *differential-privacy-adj-sym sym-on-def DP-divergence-reflexivity DP-inequality-cong-DP-divergence*)
next
case (*Suc k*)
then show *?case*
proof(*cases k*)
case *0*
then show *?thesis*
using *assms*(*1*) *assms*(*2*) *inf*.*orderE mult-eq-1-iff* **by** *fastforce*
next

case (*Suc l*)
then have *1*: *0* < *k*
by *auto*
have *2*: ((*Suc k*) ∗ ε) = (*k* ∗ ε) + ε
by (*simp add*: *mult.commute nat-distrib*(*2*))
show *?thesis*
unfolding *relpow.simps 2*
proof(*subst pure-differential-privacy-comp*[**where** *X* = *X* **and** *R* = *R*])
show *adj* ^^ *k* ⊆ *space X* × *space X*
by(*unfold Suc*, *rule adj-trans-k*, *auto simp*: *assms*)
qed(*auto simp*: *assms Suc.IH*)
qed
qed

The relaxation of parameters (obvious in the pencil and paper manner). **lemma** *DP-inequality-relax*:
assumes *1*: ε ≤ ε′ **and** *2*: δ ≤ δ′
and *DP*: *DP-inequality M N* ε δ
shows *DP-inequality M N* ε′ δ′
unfolding *DP-inequality-def*
proof
fix *A* **assume** *A*: *A* ∈ *sets M*
hence *measure M A* ≤ *exp* ε ∗ *measure N A* + δ
using *DP* **by**(*auto simp*: *DP-inequality-def*)
also have ... ≤ *exp* ε′ ∗ *measure N A* + δ′
by (*auto simp*: *add-mono mult-right-mono 1 2*)
finally show *measure M A* ≤ *exp* ε′ ∗ *measure N A* + δ′**.**
qed

proposition *differential-privacy-relax*:
assumes *DP*:*differential-privacy M adj* ε δ
and *1*: ε ≤ ε′**and** *2*: δ ≤ δ′
shows *differential-privacy M adj* ε′ δ′
using *DP DP-inequality-relax* [*OF 1 2*] **unfolding** *differential-privacy-def*
by *blast*

Stability for post-processing (cf. [Prop 2.1, AFDP])
proposition *differential-privacy-postprocessing*:
assumes ε ≥ *0*
and *differential-privacy M adj* ε δ
and *M*: *M* ∈ *X* $\rightarrow_M$ (*prob-algebra R*)
and *f*: *f* ∈ *R* $\rightarrow_M$ (*prob-algebra R*′)
and *adj* ⊆ (*space X*) × (*space X*)
shows *differential-privacy* (λ*x*. *do*{*y* ← *M x*; *f y*}) *adj* ε δ
proof(*subst differential-privacy-def*)
note [*measurable*] = *M f*
note [*arith*] = *assms*(*1*)
show ∀(*d1*, *d2*)∈ *adj*. *DP-inequality* (*M d1* ≫= *f*) (*M d2* ≫= *f*) ε

$\delta \wedge$ *DP-inequality* $(M\ d2 \gg= f)\ (M\ d1 \gg= f)\ \varepsilon\ \delta$
proof
fix $x :: 'a \times 'a$**assume** x:$x \in adj$
then obtain $d1\ d2 :: 'a$
where x: $x = (d1,d2)$
and $d1$: $d1 \in space\ X$
and $d2$: $d2 \in space\ X$
and d: $(d1,d2) \in Restr\ adj\ (space\ X)$
and *div1*[*simp*]: *DP-divergence* $(M\ d1)\ (M\ d2)\ \varepsilon \leq \delta$
and *div2*[*simp*]: *DP-divergence* $(M\ d2)\ (M\ d1)\ \varepsilon \leq \delta$
using *DP-inequality-cong-DP-divergence assms differential-privacy-def*[*of* $M\ adj\ \varepsilon\ \delta$] **by** *fastforce+*
hence *Md1*[*simp*]: $M\ d1 \in space\ (prob\text{-}algebra\ R)$**and** *Md2*[*simp*]: $M\ d2 \in space\ (prob\text{-}algebra\ R)$
by(*metis M measurable-space*)+
have *DP-divergence* $(M\ d1 \gg= f)\ (M\ d2 \gg= f)\ \varepsilon \leq \delta$
by(*auto simp*: *DP-divergence-postprocessing*[**where** $L = R$ **and** $K = R'$])
moreover have *DP-divergence* $(M\ d2 \gg= f)\ (M\ d1 \gg= f)\ \varepsilon \leq \delta$
by(*auto simp*: *DP-divergence-postprocessing*[**where** $L = R$ **and** $K = R'$])
ultimately show *case* x *of* $(d1,d2) \Rightarrow$ *DP-inequality* $(M\ d1 \gg= f)\ (M\ d2 \gg= f)\ \varepsilon\ \delta \wedge$ *DP-inequality* $(M\ d2 \gg= f)\ (M\ d1 \gg= f)\ \varepsilon\ \delta$
unfolding *DP-inequality-cong-DP-divergence* **using** x **by** *auto*
qed
qed

corollary *differential-privacy-postprocessing-deterministic*:
assumes $\varepsilon \geq 0$
and *differential-privacy* $M\ adj\ \varepsilon\ \delta$
and M[*measurable*]: $M \in X \to_M (prob\text{-}algebra\ R)$
and f[*measurable*]: $f \in R \to_M R'$
and $adj \subseteq (space\ X) \times (space\ X)$
shows *differential-privacy* $(\lambda x.\ do\{y \leftarrow M\ x;\ return\ R'\ (f\ y)\})\ adj\ \varepsilon\ \delta$
by(*rule differential-privacy-postprocessing*[**where** $f = (\lambda\ y.\ return\ R'\ (f\ y))$**and** $X = X$ **and** $R = R$ **and** $R' = R'$],*auto simp*: *assms*)

To handle the sensitivity, we prepare the conversions of adjacency relations by pre-processing.

lemma *differential-privacy-preprocessing*:
assumes $\varepsilon \geq 0$
and *differential-privacy* $M\ adj\ \varepsilon\ \delta$
and f: $f \in X' \to_M X$
and *ftr*: $\forall (x,y) \in adj'.\ (f\ x,\ f\ y) \in adj$
and $adj \subseteq (space\ X) \times (space\ X)$
and $adj' \subseteq (space\ X') \times (space\ X')$
shows *differential-privacy* $(M\ o\ f)\ adj'\ \varepsilon\ \delta$
proof(*subst differential-privacy-def*)

note [*measurable*] = *f*
show $\forall$(*d1*, *d2*)$\in$ *adj'*. *DP-inequality* ((*M* $\circ$ *f*) *d1*) ((*M* $\circ$ *f*) *d2*) ε δ $\wedge$ *DP-inequality* ((*M* $\circ$ *f*) *d2*) ((*M* $\circ$ *f*) *d1*) ε δ
proof
fix *x* ::'*c* $\times$ '*c***assume** *x*:*x* $\in$ *adj'*
then obtain *d1 d2* ::'*c* **where** *x*: *x* = (*d1*,*d2*)**and** *d1*: *d1* $\in$ *space X'***and** *d2*: *d2* $\in$ *space X'***and** *d*: (*d1*,*d2*) $\in$ *Restr adj'* (*space X'*)
using *assms* **by** *auto*
hence *fd*: (*f d1*, *f d2*) $\in$ *adj*
using *ftr* **by** *auto*
hence *DP-inequality* (*M* (*f d1*)) (*M* (*f d2*)) ε δ $\wedge$ *DP-inequality* (*M* (*f d2*)) (*M* (*f d1*)) ε δ
using *differential-privacy-def*[*of M adj* ε δ] *assms* **by** *blast*
thus *case x of* (*d1*,*d2*) $\Rightarrow$ *DP-inequality* ((*M o f*) *d1*) ((*M o f*) *d2*) ε δ $\wedge$ *DP-inequality* ((*M o f*) *d2*) ((*M o f*) *d1*) ε δ
using *x* **by** *auto*
qed
qed

"Adaptive"composition (cf. [Theorem B.1, AFDP]) **proposition** *differential-privacy-composition-adaptive*:
assumes $\varepsilon \geq 0$
and $\varepsilon' \geq 0$
and *M*: $M \in X \rightarrow_M$ (*prob-algebra Y*)
and *DPM*: *differential-privacy M adj* ε δ
and *N*: $N \in (X \bigotimes_M Y) \rightarrow_M$ (*prob-algebra Z*)
and *DPN*: $\forall$ *y* $\in$ *space Y*. *differential-privacy* (λ *x*. *N* (*x*,*y*)) *adj* ε' δ'
and *adj* $\subseteq$ (*space X*) $\times$ (*space X*)
shows *differential-privacy* (λ*x*. *do*{*y* $\leftarrow$ *M x*; *N* (*x*, *y*)}) *adj* ($\varepsilon + \varepsilon'$) ($\delta + \delta'$)
proof(*subst differential-privacy-def*)
note [*measurable*] = *M N*
note [*simp*] = *space-pair-measure*
show $\forall$(*d1*, *d2*)$\in$*adj*.
DP-inequality (*M d1* $\gg=$ (λ*y*. *N* (*d1*, *y*))) (*M d2* $\gg=$ (λ*y*. *N* (*d2*, *y*))) ($\varepsilon + \varepsilon'$) ($\delta + \delta'$) $\wedge$
DP-inequality (*M d2* $\gg=$ (λ*y*. *N* (*d2*, *y*))) (*M d1* $\gg=$ (λ*y*. *N* (*d1*, *y*))) ($\varepsilon + \varepsilon'$) ($\delta + \delta'$)
proof
fix *x* ::'*a* $\times$ '*a*
assume *x*:*x* $\in$ *adj*
then obtain *x1 x2* ::'*a*
where *x*: *x* = (*x1*,*x2*)
and *x1*: *x1* $\in$ *space X*
and *x2*: *x2* $\in$ *space X*
and *x'*: (*x1*,*x2*) $\in$ *Restr adj* (*space X*)
and *div1*: *DP-divergence* (*M x1*) (*M x2*) $\varepsilon \leq \delta$
and *div2*: *DP-divergence* (*M x2*) (*M x1*) $\varepsilon \leq \delta$

using *DP-inequality-cong-DP-divergence assms differential-privacy-def*[*of M adj* ε δ] **by** *fastforce*+
hence *N1*: (λy. *N* (*x1*, *y*)) $\in$ *Y* $\rightarrow_M$ *prob-algebra Z***and** *N2*: (λy. *N* (*x2*, *y*)) $\in$ *Y* $\rightarrow_M$ *prob-algebra Z*
by *auto*
have *Mx1*: (*M x1*)$\in$ *space*(*prob-algebra Y*)**and** *Mx2*: (*M x2*)$\in$ *space*(*prob-algebra Y*)
by (*meson measurable-space M x1 x2*)+
{
fix *y*::$'b$ **assume** *y*: $y \in$ *space Y*
have *DP-divergence* (*N* (*x1*, *y*)) (*N* (*x2*, *y*)) $\varepsilon' \leq \delta' \wedge$ *DP-divergence* (*N* (*x2*, *y*)) (*N* (*x1*, *y*)) $\varepsilon' \leq \delta'$
using *DPN DP-inequality-cong-DP-divergence differential-privacy-def*[*of* (λ *x*. *N* (*x*,*y*))*adj* ε' δ'] *x1 x2 x' y* **by** *fastforce*+
}
hence *p1*: $\forall y \in$ *space Y*. *DP-divergence* (*N* (*x1*, *y*)) (*N* (*x2*, *y*)) $\varepsilon' \leq \delta'$**and** *p2*: $\forall y \in$ *space Y*. *DP-divergence* (*N* (*x2*, *y*)) (*N* (*x1*, *y*)) $\varepsilon' \leq \delta'$
by *auto*
hence *DP-divergence* (*M x1* $\gg=$ (λy. *N* (*x1*, *y*))) (*M x2* $\gg=$ (λy. *N* (*x2*, *y*))) ($\varepsilon + \varepsilon'$) $\leq$ ($\delta + \delta'$) $\wedge$ *DP-divergence* (*M x2* $\gg=$ (λy. *N* (*x2*, *y*))) (*M x1* $\gg=$ (λy. *N* (*x1*, *y*))) ($\varepsilon + \varepsilon'$) $\leq$ ($\delta + \delta'$)
using *DP-divergence-composability*[*of M x1 Y M x2* (λy. *N* (*x1*, *y*))*Z* (λy. *N* (*x2*, *y*))ε δ ε' δ' ,*OF Mx1 Mx2 N1 N2 div1 - assms*(*1*,*2*)]
DP-divergence-composability[*of M x2 Y M x1* (λy. *N* (*x2*, *y*))*Z* (λy. *N* (*x1*, *y*))ε δ ε' δ' ,*OF Mx2 Mx1 N2 N1 div2 - assms*(*1*,*2*)] **by** *auto*
thus *case x of* (*x1*,*x2*) $\Rightarrow$ *DP-inequality* (*M x1* $\gg=$ (λy. *N* (*x1*, *y*))) (*M x2* $\gg=$ (λy. *N* (*x2*, *y*))) ($\varepsilon + \varepsilon'$) ($\delta + \delta'$) $\wedge$ *DP-inequality* (*M x2* $\gg=$ (λy. *N* (*x2*, *y*))) (*M x1* $\gg=$ (λy. *N* (*x1*, *y*))) ($\varepsilon + \varepsilon'$) ($\delta + \delta'$)
by (*auto simp*: *x DP-inequality-cong-DP-divergence*)
qed
qed

"Sequential"composition [Theorem 3.14, AFDP, generalized] **proposition** *differential-privacy-composition-pair*:
assumes $\varepsilon \geq 0$
and $\varepsilon' \geq 0$
and *DPM*: *differential-privacy M adj* ε δ
and *M*[*measurable*]: $M \in X \rightarrow_M$ (*prob-algebra Y*)
and *DPN*: *differential-privacy N adj* ε' δ'
and *N*[*measurable*]: $N \in X \rightarrow_M$ (*prob-algebra Z*)
and *adj* $\subseteq$ (*space X*) $\times$ (*space X*)
shows *differential-privacy* (λx. *do*{$y \leftarrow M\ x$; $z \leftarrow N\ x$; *return* ($Y \bigotimes_M Z$) (*y*,*z*)}) *adj* ($\varepsilon + \varepsilon'$) ($\delta + \delta'$)
proof−
have *p*: (λx. *do*{$y \leftarrow M\ x$; $z \leftarrow N\ x$; *return* ($Y \bigotimes_M Z$) (*y*,*z*)}) = (λx. *M x* $\gg=$ (λy. *case* (*x*, *y*) *of* (*x*, *y*) $\Rightarrow$ *N x* $\gg=$ (λz. *return* (*Y*

$\bigotimes_M$ Z) (y, z))))
by *auto*
have $\bigwedge y$. $y \in$ *space* $Y \Longrightarrow$ *differential-privacy* (λx. N $x \gg=$ (λz. *return* ($Y \bigotimes_M Z$) (y, z))) *adj* ε' δ'
proof−
fix y **assume** [*measurable*]: $y \in$ *space* Y
hence m: (λx. N $x \gg=$ (λz. *return* ($Y \bigotimes_M Z$) (y, z))) $\in X \to_M$ *prob-algebra* ($Y \bigotimes_M Z$)
by *auto*
show *differential-privacy* (λx. N $x \gg=$ (λz. *return* ($Y \bigotimes_M Z$) (y, z))) *adj* ε' δ'
by(*rule differential-privacy-postprocessing*[**where** f = (λz. *return* ($Y \bigotimes_M Z$) (y, z)) **and** R' = ($Y \bigotimes_M Z$) **and** $R = Z$ **and** $X = X$], *auto simp*: *assms*)
qed
thus *?thesis* **unfolding** p **using** *assms*
by(*subst differential-privacy-composition-adaptive*[**where** N =(λ(x,y). N $x \gg=$ (λz. *return* ($Y \bigotimes_M Z$) (y, z)))**and** Z = ($Y \bigotimes_M Z$)], *auto*)
qed

Laplace mechanism (1-dimensional version) (cf. [Def. 3.3 and Thm 3.6, AFDP]) **locale** *Lap-Mechanism-1dim* =
fixes X::$'a$ *measure*
and f::$'a \Rightarrow$ *real*
and *adj*::($'a$ *rel*)
assumes [*measurable*]: $f \in X \to_M$ *borel*
and *adj*: *adj* $\subseteq$ (*space* X) $\times$ (*space* X)
begin

definition *sensitivity*:: *ereal* **where**
sensitivity = *Sup*{ *ereal* $|(f\,x) - (f\,y)|$ | x y::$'a$. $x \in$ *space* $X \wedge y \in$ *space* $X \wedge (x,y) \in$ *adj* }

definition *LapMech-1dim*::*real* $\Rightarrow$ $'a \Rightarrow$ *real measure***where**
LapMech-1dim ε x = *Lap-dist* ((*real-of-ereal sensitivity*) / ε) (f x)

lemma *measurable-LapMech-1dim*[*measurable*]:
shows *LapMech-1dim* $\varepsilon \in X \to_M$ *prob-algebra borel*
unfolding *LapMech-1dim-def* **by** *auto*

lemma *LapMech-1dim-def2*:
shows *LapMech-1dim* ε x = *do*{$y \leftarrow$ *Lap-dist0* ((*real-of-ereal sensitivity*) / ε); *return borel* (f x + y) }
using *Lap-dist-def2 LapMech-1dim-def* **by** *auto*

proposition *differential-privacy-LapMech-1dim*:
assumes *pose*: $0 < \varepsilon$ **and** *sensitivity* > 0**and** *sensitivity* $< \infty$
shows *differential-privacy* (*LapMech-1dim* ε) *adj* ε 0

proof(*subst differential-privacy-def*)
show $\forall$ (*d1*::'*a*, *d2*::'*a*)$\in$*adj*.
DP-inequality (*LapMech-1dim* ε *d1*) (*LapMech-1dim* ε *d2*) ε (*0*::*real*)
$\wedge$
DP-inequality (*LapMech-1dim* ε *d2*) (*LapMech-1dim* ε *d1*) ε (*0*::*real*)
proof
fix *x*::'*a* $\times$ '*a***assume** *x* $\in$ *adj*
then obtain *d1 d2* **where** *x*: *x* = (*d1*,*d2*)**and** (*d1*,*d2*) $\in$ *adj* **and**
d1 $\in$ (*space X*)**and** *d2* $\in$ (*space X*)
using *adj* **by** *blast*
with *assms* **have** |(*f d1*) $-$ (*f d2*)| $\leq$ *sensitivity*
unfolding *sensitivity-def* **using** *le-Sup-iff* **by** *blast*
with *assms* **have** *q*: |(*f d1*) $-$ (*f d2*)| $\leq$ *real-of-ereal sensitivity*
by (*auto simp*: *ereal-le-real-iff*)
hence *q2*: |(*f d2*) $-$ (*f d1*)| $\leq$ *real-of-ereal sensitivity*
by *auto*
from *assms* **have** *pos*: *0* < (*real-of-ereal sensitivity*)/ε
by (*auto simp*: *zero-less-real-of-ereal*)
have *a1*: *DP-inequality* (*LapMech-1dim* (ε::*real*) *d1*) (*LapMech-1dim*
ε *d2*) ε (*0*::*real*)
by (*auto simp*: *DP-divergence-Lap-dist'-eps pose q DP-inequality-cong-DP-divergence*
LapMech-1dim-def)
have *a2*: *DP-inequality* (*LapMech-1dim* (ε::*real*) *d2*) (*LapMech-1dim*
ε *d1*) ε (*0*::*real*)
by (*auto simp*: *DP-divergence-Lap-dist'-eps pose q2 DP-inequality-cong-DP-divergence*
LapMech-1dim-def)
from *a1 a2* **show** *case x of*
(*d1*::'*a*, *d2*::'*a*) $\Rightarrow$
DP-inequality (*LapMech-1dim* ε *d1*) (*LapMech-1dim* ε *d2*) ε (*0*::*real*)
$\wedge$
DP-inequality (*LapMech-1dim* ε *d2*) (*LapMech-1dim* ε *d1*) ε (*0*::*real*)
by(*auto simp*:*x*)
qed
qed

end

Laplace mechanism (generalized version) (cf. [Def. 3.3 and Thm 3.6, AFDP]) **locale** *Lap-Mechanism-list* =
fixes *X*::'*a measure*
and *f*::'*a* $\Rightarrow$ *real list*
and *adj*::('*a rel*)
and *m*::*nat*
assumes [*measurable*]: *f* $\in$ *X* $\rightarrow_M$ (*listM borel*)
and *len*: $\bigwedge$ *x*. *x* $\in$ *space X* $\Longrightarrow$ *length* (*f x*) = *m*
and *adj*: *adj* $\subseteq$ (*space X*) $\times$ (*space X*)
begin

definition *sensitivity*:: *ereal* **where**
sensitivity = *Sup*{ *ereal* ($\sum$ *i*∈{*1..m*}. | *nth* (*f x*) (*i−1*) − *nth* (*f y*) (*i−1*) |) | *x y*::′*a*. *x* ∈ *space X* ∧ *y* ∈ *space X* ∧ (*x*,*y*) ∈ *adj*}

definition *LapMech-list*::*real* ⇒ ′*a* ⇒ (*real list*) *measure***where**
LapMech-list ε *x* = *Lap-dist-list* ((*real-of-ereal sensitivity*) / ε) (*f x*)

lemma *LapMech-list-def2*:
assumes *x* ∈ *space X*
shows *LapMech-list* ε *x* = *do*{ *xs* ← *Lap-dist0-list* (*real-of-ereal sensitivity* / ε) *m*; *return* (*listM borel*) (*map2* (+) (*f x*) *xs*)}
using *Lap-dist-list-def2 len*[*OF assms*] *LapMech-list-def* **by** *auto*

proposition *differential-privacy-LapMech-list*:
assumes *pose*: ε > *0*
and *sensitivity* > *0*
and *sensitivity* < ∞
shows *differential-privacy* (*LapMech-list* ε) *adj* ε *0*
proof(*subst differential-privacy-def*)
show ∀(*d1*, *d2*)∈ *adj*. *DP-inequality* (*LapMech-list* ε *d1*) (*LapMech-list* ε *d2*) ε *0*
∧ *DP-inequality* (*LapMech-list* ε *d2*) (*LapMech-list* ε *d1*) ε *0*
proof(*rule ballI*)
fix *x*::′*a* × ′*a***assume** *x* ∈ *adj*
then obtain *d1 d2*::′*a*
where *x*: *x* = (*d1*,*d2*)
and (*d1*,*d2*) ∈ *adj*
and *d1*: *d1* ∈ (*space X*)
and *d2*: *d2* ∈ (*space X*)
using *adj* **by** *blast*
with *assms* **have** ($\sum$ *i*∈{*1..m*}. | *nth* (*f d1*) (*i−1*) − *nth* (*f d2*) (*i−1*) |) ≤ *sensitivity*
unfolding *sensitivity-def* **using** *le-Sup-iff* **by** *blast*
with *assms* **have** *q*: ($\sum$ *i*∈{*1..m*}. | *nth* (*f d1*) (*i−1*) − *nth* (*f d2*) (*i−1*) |) ≤ *real-of-ereal sensitivity*
by (*auto simp*: *ereal-le-real-iff*)
hence *q2*: ($\sum$ *i*∈{*1..m*}. | *nth* (*f d2*) (*i−1*) − *nth* (*f d1*) (*i−1*) |) ≤ *real-of-ereal sensitivity*
by (*simp add*: *abs-minus-commute*)
from *assms* **have** *pos*: *0* ≤ (*real-of-ereal sensitivity*)
by (*simp add*: *real-of-ereal-pos*)
have *fd1*: *length* (*f d1*) = *m***and** *fd2*: *length* (*f d2*) = *m*
using *len d1 d2* **by** *auto*
have *a1*: *DP-inequality* (*LapMech-list* (ε::*real*) *d1*) (*LapMech-list* ε *d2*) ε *0*
using *DP-Lap-dist-list-eps*[*OF pose fd1 fd2 q pos*] **by** (*simp add*:

LapMech-list-def DP-inequality-cong-DP-divergence zero-ereal-def)
have *a2*: *DP-inequality* (*LapMech-list* (*ε*::*real*) *d2*) (*LapMech-list ε d1*) *ε 0*
using *DP-Lap-dist-list-eps*[*OF pose fd2 fd1 q2 pos*] **by** (*simp add*: *LapMech-list-def DP-inequality-cong-DP-divergence zero-ereal-def*)
show *case x of* (*d1*, *d2*) ⇒ *DP-inequality* (*local.LapMech-list ε d1*) (*local.LapMech-list ε d2*) *ε 0* ∧ *DP-inequality* (*local.LapMech-list ε d2*) (*local.LapMech-list ε d1*) *ε 0*
by(*auto simp*:*x a1 a2*)
qed

qed

end

8.2 Formalization of Results in [AFDP]

We finally instantiate *X* and *adj* according to the textbook [AFDP]

locale *results-AFDP* =
fixes *n* ::*nat*
begin

interpretation *L1-norm-list* (*UNIV*::*nat set*) (λ *x y*. |*int x* − *int y*|) *n*
by(*unfold-locales*,*auto*)

definition *sp-Dataset* :: *nat list measure***where**
sp-Dataset ≡ *restrict-space* (*listM* (*count-space UNIV*)) *space-L1-norm-list*

definition *adj-L1-norm* :: (*nat list* × *nat list*) *set***where**
adj-L1-norm ≡ {(*xs*,*ys*)| *xs ys*. *xs* ∈ *space sp-Dataset* ∧ *ys* ∈ *space sp-Dataset* ∧ *dist-L1-norm-list xs ys* ≤ *1*}

definition *dist-L1-norm* :: *nat* ⇒ (*nat list* × *nat list*) *set***where**
dist-L1-norm k ≡ {(*xs*,*ys*)| *xs ys*. *xs* ∈ *space sp-Dataset* ∧ *ys* ∈ *space sp-Dataset* ∧ *dist-L1-norm-list xs ys* ≤ *k*}

abbreviation
differential-privacy-AFDP M ε δ ≡ *differential-privacy M adj-L1-norm ε δ*

lemma *adj-sub*: *adj-L1-norm* ⊆ *space sp-Dataset* × *space sp-Dataset*
unfolding *sp-Dataset-def adj-L1-norm-def* **by** *blast*

lemmas *differential-privacy-relax-AFDP′* =
differential-privacy-relax[*of - adj-L1-norm*]

lemmas *differential-privacy-postprocessing-AFDP* =
differential-privacy-postprocessing[*of - - adj-L1-norm*, *OF - - - -*

adj-sub]

lemmas *differential-privacy-composition-adaptive-AFDP* =
differential-privacy-composition-adaptive[*of* - - - - - *adj-L1-norm*, *OF* - - - - - - *adj-sub*]

lemmas *differential-privacy-composition-pair-AFDP* =
differential-privacy-composition-pair[*of* - - - *adj-L1-norm*, *OF* - - - - - - *adj-sub*]

Group privacy [Theorem 2.2, AFDP].

lemma *group-privacy-AFDP*:
assumes *M*: $M \in$ *sp-Dataset* $\rightarrow_M$ *prob-algebra Y*
and *DP*: *differential-privacy-AFDP M* ε *0*
shows *differential-privacy M* (*dist-L1-norm k*) (*real k* $*$ ε) *0*
proof(*rule differential-privacy-restrict*[**where** *adj* = *adj-L1-norm* ^^ *k***and** *adj'* = *dist-L1-norm k*])
show *differential-privacy M* (*adj-L1-norm* ^^ *k*) (*real k* $*$ ε) *0*
by(*rule pure-differential-privacy-trans-k*[*OF adj-sub DP M*])
show *dist-L1-norm k* $\subseteq$ *adj-L1-norm* ^^ *k*
using *L1-adj-iterate*[*of* - *n* - *k*] **unfolding** *space-restrict-space space-listM space-count-space lists-UNIV adj-L1-norm-def dist-L1-norm-def sp-Dataset-def* **by** *auto*
qed

context
fixes *f*::*nat list* $\Rightarrow$ *real*
assumes [*measurable*]: $f \in$ *sp-Dataset* $\rightarrow_M$ *borel*
begin

interpretation *Lap-Mechanism-1dim sp-Datasetf adj-L1-norm*
by(*unfold-locales*,*auto simp*: *adj-sub*)

thm *LapMech-1dim-def*

definition *LapMech-1dim-AFDP* :: *real* $\Rightarrow$ *nat list* $\Rightarrow$ *real measure***where**
LapMech-1dim-AFDP ε *x* = *do*{*y* $\leftarrow$ *Lap-dist0* ((*real-of-ereal sensitivity*) / ε); *return borel* (*f x* + *y*) }

lemma *LapMech-1dim-AFDP'*:
shows *LapMech-1dim-AFDP* = *LapMech-1dim*
unfolding *LapMech-1dim-AFDP-def LapMech-1dim-def2* **by** *auto*

lemmas *differential-privacy-Lap-Mechanism-1dim-AFDP* =
differential-privacy-LapMech-1dim[*of* - , *simplified LapMech-1dim-AFDP'*[*symmetric*]]

end

context
  fixes *f*::*nat list* ⇒ *real list*
    and *m*::*nat*
  assumes [*measurable*]: $f \in$ *sp-Dataset* $\rightarrow_M$ (*listM borel*)
    and *len*: $\bigwedge$ *x*. $x \in$ *space X* $\Longrightarrow$ *length* (*f x*) = *m*
begin

interpretation *Lap-Mechanism-list sp-Datasetf adj-L1-norm m*
  by(*unfold-locales*,*auto simp*: *len adj-sub*)

definition *LapMech-list-AFDP* :: *real* ⇒ *nat list* ⇒ *real list measure***where**
  LapMech-list-AFDP ε *x* = *do*{ *ys* ← (*Lap-dist0-list*(*real-of-ereal sensitivity* / ε) *m*); *return* (*listM borel*) (*map2* (+) (*f x*) *ys*) }

thm *LapMech-list-def*

lemma *LapMech-list-AFDP′*:
  assumes $x \in$ *space sp-Dataset*
  shows *LapMech-list-AFDP* ε *x* = *LapMech-list* ε *x*
  unfolding *LapMech-list-AFDP-def LapMech-list-def2*[*OF assms*] **by** *auto*

lemma *differential-privacy-Lap-Mechanism-list-AFDP*:
  assumes *0* < ε
    and *0* < *sensitivity*
    and *sensitivity* < ∞
  shows *differential-privacy-AFDP* (*LapMech-list-AFDP* ε) ε *0*
proof(*subst differential-privacy-def*)
  {
    fix *d1 d2* **assume** *d*: (*d1*, *d2*)∈ *adj-L1-norm*
   then have *1*: *d1* ∈ (*space sp-Dataset*)**and** *2*: *d2* ∈ (*space sp-Dataset*)
      using *adj-sub* **by** *auto*
   have *DP-inequality* (*LapMech-list-AFDP* ε *d1*) (*LapMech-list-AFDP* ε *d2*) ε *0* ∧ *DP-inequality* (*LapMech-list-AFDP* ε *d2*) (*LapMech-list-AFDP* ε *d1*) ε *0*
     using *differential-privacy-LapMech-list*[*OF assms*(*1*)] *assms*(*2*,*3*) *d*
      unfolding *LapMech-list-AFDP′*[*OF 1*] *LapMech-list-AFDP′*[*OF 2*] *differential-privacy-def* **by** *blast*
  }
 thus ∀ (*d1*, *d2*)∈ *adj-L1-norm*. *DP-inequality* (*LapMech-list-AFDP* ε *d1*) (*LapMech-list-AFDP* ε *d2*) ε *0* ∧ *DP-inequality* (*LapMech-list-AFDP* ε *d2*) (*LapMech-list-AFDP* ε *d1*) ε *0*
    by *blast*

qed

end

end

end

theory *Differential-Privacy-Example-Report-Noisy-Max*
  imports *Differential-Privacy-Standard*
begin

9 Report Noisy Max mechanism for counting query with Laplace noise

lemma *measurable-let*[*measurable*]:
  assumes [*measurable*]: $g \in M \rightarrow_M L$
    and [*measurable*]: $(\lambda(x,y).\ f\ x\ y) \in M \bigotimes_M L \rightarrow_M N$
  shows $(\lambda x.\ (Let\ (g\ x)\ (f\ x))) \in M \rightarrow_M N$
  unfolding *Let-def* **by** *auto*

9.1 Formalization of argmax procedure

primrec *max-argmax* :: *real list* ⇒ (*ereal* × *nat*) **where**
  max-argmax [] = (−∞,*0*)|
  max-argmax (*x*#*xs*) = (*let* (*m*, *i*) = *max-argmax xs in if x* > *m then* (*x*,*0*) *else* (*m*, *Suc i*))

value *max-argmax* []
value *max-argmax* [*1*]
value *max-argmax* [*2*,*1*]
value *max-argmax* [*1*,*2*,*3*,*4*,*5*]
value *max-argmax* [*1*,*5*,*2*,*3*,*4*]
value ([*1*,*2*,*3*,*4*,*5*]::*int list*) ! *4*

primrec *max-argmax′* :: *real list* ⇒ (*ereal* × *nat*) **where**
  max-argmax′ [] = (−∞,*0*)|
  max-argmax′ (*x*#*xs*) = (*if x* > *fst* (*max-argmax′ xs*) *then* (*x*,*0*) *else* (*fst* (*max-argmax′ xs*),*Suc* (*snd* (*max-argmax′ xs*))))

lemma *max-argmax-is-max-argmax′*:
  shows *max-argmax xs* = *max-argmax′ xs*
  by(*induction xs*,*auto simp*: *case-prod-beta*)

lemma *measurable-max-argmax*[*measurable*]:

shows *max-argmax* $\in$ (*listM* (*borel* :: *real measure*)) $\rightarrow_M$ (*borel* :: (*ereal measure*)) $\bigotimes_M$ *count-space UNIV*
proof−
have *1*: $(-\infty, 0) \in$ *space* (*borel* $\bigotimes_M$ *count-space UNIV*)
by (*metis iso-tuple-UNIV-I measurable-Pair1′ measurable-space space-borel space-count-space*)
thus*max-argmax* $\in$ *listM borel* $\rightarrow_M$ (*borel* :: (*ereal measure*)) $\bigotimes_M$ *count-space UNIV*
unfolding *max-argmax-def* **using** *measurable-rec-list′′′ 1* **by** *measurable*
qed

definition *argmax-list* :: *real list* $\Rightarrow$ *nat* **where**
argmax-list = *snd o max-argmax*

lemma *measurable-argmax-list*[*measurable*]:
shows *argmax-list* $\in$ (*listM* (*borel* :: *real measure*)) $\rightarrow_M$ *count-space UNIV*
by(*unfold argmax-list-def*, *rule measurable-comp*[**where** *N* = (*borel* :: (*ereal measure*)) $\bigotimes_M$ *count-space UNIV*],*auto*)

lemma *argmax-list-le-length*:
shows *length xs* = *Suc k* $\Longrightarrow$ (*argmax-list xs*) $\leq$ *k*
proof(*induction k arbitrary*:*xs*)
case *0*
then obtain *x* **where** *xs*: *xs* = [*x*]
by (*metis length-0-conv length-Suc-conv*)
then show *?case* **unfolding** *xs argmax-list-def comp-def max-argmax.simps* **by** *auto*
next
case (*Suc k*)
then obtain *y ys* **where** *xs*: *xs* = *y* # *ys* **and** *len*:*length ys* = *Suc k*
by (*meson length-Suc-conv*)
from *Suc.IH*[*of ys*, *OF len*] **obtain** *k′ z* **where** *1*: *max-argmax ys* = (*z*,*k′*) **and** *2*: *k′* $\leq$ *k*
unfolding *argmax-list-def comp-def* **by** (*metis prod.collapse*)
then show *?case* **unfolding** *xs argmax-list-def comp-def max-argmax.simps 1* **using** *2* **by** *auto*
qed

lemma *fst-max-argmax-append*:
shows (*fst* (*max-argmax* (*xs* @ *ys*))) = *max* (*fst* (*max-argmax xs*)) (*fst* (*max-argmax ys*))
unfolding *max-argmax-is-max-argmax′* **by**(*induction xs arbitrary*: *ys*,*auto*)

lemma *fst-max-argmax-is-max*:
shows *fst* (*max-argmax xs*) = *Max* (*set xs* $\cup$ $\{-\infty\}$)

unfolding *max-argmax-is-max-argmax'*
proof(*induction xs*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a xs*)
have *1*: $Max\ (set\ (map\ ereal\ (a\ \#\ xs)) \cup \{-\ \infty\}) = max\ a\ (Max\ (set\ (map\ ereal\ xs) \cup \{-\ \infty\}))$
by (*simp add*: *Un-ac*(*3*))
have $fst\ (max\text{-}argmax'\ (a\ \#\ xs)) = (\ if\ Max\ (set\ (map\ ereal\ xs) \cup \{-\ \infty\}) < ereal\ a\ then\ (ereal\ a)\ else\ (Max\ (set\ (map\ ereal\ xs) \cup \{-\ \infty\})))$
unfolding *max-argmax'.simps Cons.IH* **by** *auto*
also have ... $= max\ a\ (Max\ (set\ (map\ ereal\ xs) \cup \{-\ \infty\}))$
unfolding *max-def*
by (*meson Metric-Arith.nnf-simps*(*8*))
finally show *?case* **using** *1* **by** *auto*
qed

lemma *argmax-list-max-argmax*:
assumes $xs \neq []$
shows $(argmax\text{-}list\ xs) = m \longleftrightarrow max\text{-}argmax\ xs = (ereal\ (xs\ !\ m),\ m)$
using *assms* **proof**(*induction xs arbitrary*: *m*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a xs*)
then show *?case*
proof(*cases* $xs = []$)
case *True*
hence *1*: $a\ \#\ xs = [a]$
by *auto*
then show *?thesis* **unfolding** *1 argmax-list-def comp-def max-argmax.simps* **by** *auto*
next
case *False*
then show *?thesis*
unfolding *argmax-list-def comp-def max-argmax-is-max-argmax' max-argmax'.simps* **using** *less-ereal.simps*(*5*) *max-argmax'.simps*(*1*) *nth-Cons-0 nth-Cons-Suc split-pairs local.Cons*(*1*) *max-argmax-is-max-argmax'* **by** *fastforce*
qed
qed

lemma *argmax-list-gives-max*:
assumes $xs \neq []$
shows $(argmax\text{-}list\ xs) = m \Longrightarrow \forall\, i \in \{0..<length\ xs\}.\ xs\ !\ i \leq xs\ !\ m$

proof
  fix *i* **assume** *1*: (*argmax-list xs*) = *m* **and** *i*: $i \in \{0..<length\ xs\}$
  hence *max-argmax xs* = (*ereal* (*xs* ! *m*), *m*)
    using *argmax-list-max-argmax*[*OF assms*] **by** *blast*
  with *fst-max-argmax-is-max 1* **have** *0*: *xs* ! *m* = *Max* (*set xs* ∪ {−∞})
    by (*metis fst-eqD*)
  have *xs* ! *i* ≤ *Max* (*set xs* ∪ {−∞})
    using *i* **by** *auto*
  thus *xs* ! *i* ≤ *xs* ! *m*
    using *0* **by** (*metis ereal-less-eq*(*3*))
qed

lemma *argmax-list-gives-max2*:
  assumes *xs* ≠ []
  shows (*x* ≤ (*xs* ! *m*) ∧ *argmax-list xs* = *m*) = (*max-argmax* (*x*#*xs*) = ((*xs* ! *m*), (*Suc m*)))
proof(*induction m*)
  case *0*
  then show *?case*
   unfolding *argmax-list-max-argmax*[*of xs 0*, *OF assms*] *max-argmax-is-max-argmax′ max-argmax′.simps*
   by (*metis* (*mono-tags*, *lifting*) *Suc-n-not-n diff-Suc-1′ ereal-less-eq*(*3*) *not-le split-conv split-pairs*)
next
  case (*Suc m*)
  then show *?case*
   unfolding *argmax-list-max-argmax*[*of xs m*, *OF assms*] *max-argmax-is-max-argmax′ max-argmax′.simps*
   by(*split if-split*) (*metis Suc-inject Zero-neq-Suc argmax-list-max-argmax assms basic-trans-rules*(*20*) *ereal-less-eq*(*3*) *fst-conv le-cases less-eq-ereal-def max-argmax-is-max-argmax′ old.prod.case snd-conv*)
qed

lemma *fst-max-argmax-adj*:
  fixes *xs ys rs* :: *real list* **and** *n* :: *nat*
  assumes *length xs* = *n*
    and *length ys* = *n*
    and *length rs* = *n*
    and *adj*: *list-all2* (λ *x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) *xs ys*
  shows (*fst* (*max-argmax* (*map2* (+) *xs rs*))) ≥ (*fst* (*max-argmax* (*map2* (+) *ys rs*))) ∧ (*fst* (*max-argmax* (*map2* (+) *xs rs*))) ≤ (*fst* (*max-argmax* (*map2* (+) *ys rs*))) + *1*
  using *assms*
proof(*induction n arbitrary*: *xs ys rs*)
  case *0*
  hence *q* : *xs* = [] *ys* = [] *rs* = []
    by *auto*
  thus *?case* **unfolding** *q* **by** *auto*

next
 case (*Suc n*)
 then obtain *x2 y2 r2 ys2 xs2 rs2* **where** *q*: *xs* = *x2* # *xs2 ys* = *y2* # *ys2 rs* = *r2* # *rs2*
 using *length-Suc-conv* **by** *metis*
 hence *q2*: *length xs2* = *n length ys2* = *n length rs2* = *n*
 using *Suc.prems*(*1*,*2*,*3*) **by** *auto*
 from *Suc.prems*(*4*) **have** *q3*: *list-all2* ($\lambda x\ y.\ y \le x \wedge x \le y + 1$) *xs2 ys2* **and** *q4*: $y2 \le x2 \wedge x2 \le y2 + 1$
 unfolding *q* **by** *auto*
 hence *q5*: *fst* (*max-argmax* (*map2* (+) *ys2 rs2*)) ≤ *fst* (*max-argmax* (*map2* (+) *xs2 rs2*)) ∧
 fst (*max-argmax* (*map2* (+) *xs2 rs2*)) ≤ *fst* (*max-argmax* (*map2* (+) *ys2 rs2*)) + *1*
 using *Suc*(*1*) *q2* **by** *auto*
 from *q4* **have** *q6*: *ereal* (*y2* + *r2*) ≤ *ereal* (*x2* + *r2*) ∧ *ereal* (*x2* + *r2*) ≤ *ereal* (*y2* + *r2*) + *1*
 by *auto*
 have *indx*: *fst* (*max-argmax* (*map2* (+) *xs rs*)) = *max* (*x2* + *r2*) (*fst* (*max-argmax* (*map2* (+) *xs2 rs2*)))
 unfolding *q max-argmax-is-max-argmax′* **by** *auto*
 have *indy*: *fst* (*max-argmax* (*map2* (+) *ys rs*)) = *max* (*y2* + *r2*) (*fst* (*max-argmax* (*map2* (+) *ys2 rs2*)))
 unfolding *q max-argmax-is-max-argmax′* **by** *auto*
 show *?case*
 unfolding *indx indy* **proof**(*intro conjI*)
 show *max* (*ereal* (*y2* + *r2*)) (*fst* (*max-argmax* (*map2* (+) *ys2 rs2*))) ≤ *max* (*ereal* (*x2* + *r2*)) (*fst* (*max-argmax* (*map2* (+) *xs2 rs2*)))
 using *q5 q6* **using** *ereal-less-eq*(*3*) *max.mono* **by** *blast*
 have *max* (*ereal* (*x2* + *r2*)) (*fst* (*max-argmax* (*map2* (+) *xs2 rs2*))) ≤ *max* (*ereal* (*y2* + *r2*) + *1*) (*fst* (*max-argmax* (*map2* (+) *ys2 rs2*)) + *1*)
 using *q5 q6* **using** *ereal-less-eq*(*3*) *max.mono* **by** *auto*
 also have ... ≤ (*max* (*ereal* (*y2* + *r2*)) (*fst* (*max-argmax* (*map2* (+) *ys2 rs2*)))) + *1*
 by (*meson add-right-mono max.bounded-iff max.cobounded1 max.cobounded2*)
 finally show *max* (*ereal* (*x2* + *r2*)) (*fst* (*max-argmax* (*map2* (+) *xs2 rs2*))) ≤ *max* (*ereal* (*y2* + *r2*)) (*fst* (*max-argmax* (*map2* (+) *ys2 rs2*))) + *1*.
 qed
qed

9.2 An auxiliary function to calculate the argmax of a list where an element has been inserted.

definition *argmax-insert* :: *real* ⇒ *real list* ⇒ *nat* ⇒ *nat* **where**
 argmax-insert k ks i = *argmax-list* (*list-insert k ks i*)

lemma *argmax-insert-i-i-0*:
 shows (*argmax-insert k xs 0* = *0*) ⟷ (*k* > (*fst* (*max-argmax xs*)))
 unfolding *argmax-insert-def argmax-list-def* **by** (*auto simp*: *max-argmax-is-max-argmax′ list-insert-def*)

lemma *argmax-insert-i-i-expand*:
 assumes *xs* ≠ []
  and *m* ≤ *n*
  and *length xs* = *n*
 shows (*argmax-insert k xs m* = *m*) ⟷ (*max-argmax* ((*take m xs*) @ [*k*] @ (*drop m xs*))) = (*k*,*m*)
 unfolding *argmax-insert-def* **by**(*subst argmax-list-max-argmax*) (*auto simp*:*assms nth-append list-insert-def*)

lemma *argmax-insert-i-i-iterate*:
 assumes *m* ≤ *n*
  and *length xs* = *n*
 shows (*argmax-insert k* (*x* # *xs*) (*Suc m*) = (*Suc m*)) ⟷ ((*x* ≤ *k*) ∧ (*argmax-insert k xs m* = *m*))
 unfolding *argmax-insert-def list-insert-def*
proof(*subst argmax-list-max-argmax*)
 show *0*: *take* (*Suc m*) (*x* # *xs*) @ [*k*] @ *drop* (*Suc m*) (*x* # *xs*) ≠ []
  by *auto*
 have *m* ≤ *length xs*
  using *assms* **by** *auto*
 hence *1*: (*take m xs* @ [*k*] @ *drop m xs*) ! *m* = *k*
  by(*subst nth-append*, *auto*)
 have *2*: *take m xs* @ [*k*] @ *drop m xs* ≠ []
  by *auto*
 show (*max-argmax* (*take* (*Suc m*) (*x* # *xs*) @ [*k*] @ *drop* (*Suc m*) (*x* # *xs*)) = (*ereal* ((*take* (*Suc m*) (*x* # *xs*) @ [*k*] @ *drop* (*Suc m*) (*x* # *xs*)) ! *Suc m*), *Suc m*)) =
 (*x* ≤ *k* ∧ *argmax-list* (*take m xs* @ [*k*] @ *drop m xs*) = *m*)
  using *argmax-list-gives-max2*[*of take m xs* @ [*k*] @ *drop m xs x m*, *OF 2*] **unfolding** *1 case-prod-beta*
  by (*metis 1 Cons-eq-appendI drop-Suc-Cons nth-Cons-Suc prod.sel*(*1*) *prod.sel*(*2*) *take-Suc-Cons*)
qed

lemma *argmax-insert-i-i*:
 assumes *m* ≤ *n*
  and *length xs* = *n*
 shows (*argmax-insert k xs m* = *m*) ⟷ (*ereal k* > (*fst* (*max-argmax* (*drop m xs*))) ∧ (*ereal k* ≥ (*fst* (*max-argmax* (*take m xs*)))))
 using *assms* **unfolding** *max-argmax-is-max-argmax′*
proof(*induction xs arbitrary*: *n k m*)
 case *Nil*
 thus *?case*
  by (*metis drop-Nil drop-eq-Nil2 argmax-insert-i-i-0 less-eq-ereal-def*

list.size(*3*) *order-antisym-conv take-Nil max-argmax-is-max-argmax′*)
next
case (*Cons a zs*)
thus *?case*
proof(*induction m arbitrary*: *n zs*)
case *0*
hence *p*: $\bigwedge$ *xs*. (*take 0 xs*) = []
by *auto*
show *?case*
unfolding *p max-argmax′.simps*(*1*) **by** (*metis drop0 ereal-less-eq*(*2*) *fst-conv argmax-insert-i-i-0 max-argmax-is-max-argmax′*)
next
case (*Suc m*)
have *a1*: (*argmax-insert k* (*a* # *zs*) (*Suc m*) = *Suc m*) ⟷ ((*a* ≤ *k*) ∧ (*argmax-insert k zs m* = *m*))
using *argmax-insert-i-i-iterate Suc.prems*(*2*) *Suc.prems*(*3*) **by** *auto*
have *a2*: (*argmax-insert k zs m* = *m*) ⟷ (*fst* (*max-argmax′* (*drop m zs*)) < *ereal k* ∧ *fst* (*max-argmax′* (*take m zs*)) ≤ *ereal k*)
using *Suc.prems*(*1*) *Suc.prems*(*2*) *Suc.prems*(*3*) **by** *auto*
consider*fst* (*max-argmax′* (*take m zs*)) < *ereal a*|*fst* (*max-argmax′* (*take m zs*)) > *ereal a*| *fst* (*max-argmax′* (*take m zs*)) = *ereal a*
using *linorder-less-linear* **by** *auto*
hence *a4*: ((*a* ≤ *k*) ∧ *fst* (*max-argmax′* (*take m zs*)) ≤ *ereal k*) ⟷ *fst* (*max-argmax′* (*take* (*Suc m*) (*a* # *zs*))) ≤ *ereal k*
proof(*cases*)
case *1*
thus *?thesis*
unfolding *max-argmax′.simps*(*2*) **using** *less-eq-ereal-def order-less-le-trans* **by** *fastforce*
next
case *2*
hence (*a* ≤ *k* ∧ *fst* (*max-argmax′* (*take m zs*)) ≤ *ereal k*) ⟷ (*a* < *k* ∧ *fst* (*max-argmax′* (*take m zs*)) ≤ *ereal k*)
using *leD nless-le* **by** *auto*
thus *?thesis*
unfolding *max-argmax′.simps*(*2*) **using** *2 le-ereal-less* **by** *auto*
next
case *3*
thus *?thesis*
unfolding *max-argmax′.simps*(*2*) **by** *auto*
qed
from *a1 a2 a4* **show** (*argmax-insert k* (*a* # *zs*) (*Suc m*) = *Suc m*) ⟷ ((*fst* (*max-argmax′* (*drop* (*Suc m*) (*a* # *zs*))) < *ereal k* ∧ *fst* (*max-argmax′* (*take* (*Suc m*) (*a* # *zs*))) ≤ *ereal k*))
by *auto*
qed
qed

lemma *argmax-insert-i-i'*:
assumes $m \leq n$
and *length xs* $= n$
shows (*argmax-insert k xs m* $=$ *m*) $\longleftrightarrow$ (*ereal k* $\geq$ (*fst* (*max-argmax xs*)) $\wedge$ (*ereal k* $\neq$ (*fst* (*max-argmax* (*drop m xs*)))))
proof−
have (*argmax-insert k xs m* $=$ *m*) $\longleftrightarrow$ (*ereal k* $>$ (*fst* (*max-argmax* (*drop m xs*))) $\wedge$ (*ereal k* $\geq$ (*fst* (*max-argmax* (*take m xs*)))))
by (*rule argmax-insert-i-i*[*OF assms*])
also have ... $\longleftrightarrow$ (*ereal k* $\geq$ (*fst* (*max-argmax xs*)) $\wedge$ (*ereal k* $\neq$ (*fst* (*max-argmax* (*drop m xs*)))))
proof−
have *fst* (*max-argmax xs*) $=$ *fst* (*max-argmax* ((*take m xs*) @ (*drop m xs*)))
by *auto*
also have ... $=$ *max* (*fst* (*max-argmax* (*take m xs*))) (*fst* (*max-argmax* (*drop m xs*)))
using *fst-max-argmax-append*[*symmetric*] **by** *auto*
finally show *?thesis* **by** *auto*
qed
finally show *?thesis*.
qed

lemma *argmax-insert-i-i'-region*:
assumes *length xs* $= n$ **and** *length ys* $= n$ **and** $i \leq n$
shows $\{r \mid r ::$ *real*. *argmax-insert* ($r + d$) ((*map2* (+) *xs ys*)) $i = i\} = \{r \mid (r ::$ *real*). (*ereal* ($r + d$) $\geq$ (*fst* (*max-argmax* (*map2* (+) *xs ys*))) $\wedge$ (*ereal* ($r + d$) $\neq$ (*fst* (*max-argmax* (*drop i* (*map2* (+) *xs ys*))))))$\}$
using *assms argmax-insert-i-i'* **by** *auto*

10 Formal proof of DP of RNM

10.1 A formal proof of the main part of differential privacy of report noisy max [Claim 3.9, AFDP]

locale *Lap-Mechanism-RNM-mainpart* =
fixes *M*::*'a measure*
and *adj*::*'a rel*
and *c*::*'a* $\Rightarrow$ *real list*
assumes *c*: $c \in M \rightarrow_M$ (*listM borel*)
and *cond*: $\forall$ (*x*,*y*) $\in$ *adj*. *list-all2* ($\lambda x\ y.\ y \leq x \wedge x \leq y + 1$) (*c x*) (*c y*) $\vee$ *list-all2* ($\lambda x\ y.\ y \leq x \wedge x \leq y + 1$) (*c y*) (*c x*)
and *adj*: *adj* $\subseteq$ (*space M*) $\times$ (*space M*)
begin

We define an abstracted version of report noisy max mechanism. We later instantiate c with the counting query.

definition *LapMech-RNM* :: *real* $\Rightarrow$ *'a* $\Rightarrow$ *nat measure* **where**

LapMech-RNM ε x = do {y ← Lap-dist-list (1 / ε) (c x); return (count-space UNIV) (argmax-list y)}

lemma *measurable-LapMech-RNM*[*measurable*]:
shows *LapMech-RNM ε ∈ M* $\rightarrow_M$ *prob-algebra*(*count-space UNIV*)
unfolding *LapMech-RNM-def argmax-list-def* **using** *c* **by** *auto*

Calculating the density of the probability distributions sampled from the main part of *LapMech-RNM* **context**
fixes *ε*::*real*
assumes *pose*:*0 < ε*
begin

The main part of *LapMech-RNM* **definition** *RNM′* :: *real list ⇒ nat measure* **where**
RNM′ zs = do {y ← Lap-dist-list (1 / ε) (zs); return (count-space UNIV) (argmax-list y)}

lemma *RNM′-def2*:
shows *RNM′ zs = do {rs ← Lap-dist0-list (1 / ε) (length zs); y ← return (listM borel) (map2 (+) zs rs); return (count-space UNIV) (argmax-list y)}*
by (*unfold RNM′-def Lap-dist-list-def2*,*subst bind-assoc*[**where** *N = listM borel* **and** *R = count-space UNIV*],*auto*)

lemma *measurable-RNM′*[*measurable*]:
shows *RNM′ ∈ listM borel* $\rightarrow_M$ *prob-algebra*(*count-space UNIV*)
unfolding *RNM′-def argmax-list-def* **by** *auto*

lemma *LapMech-RNM-RNM′-c*:
shows *LapMech-RNM ε = RNM′ o c*
unfolding *RNM′-def LapMech-RNM-def* **by** *auto*

lemma *RNM′-Nil*:
shows *RNM′* [] *= return* (*count-space* (*UNIV* :: *nat set*)) *0*
by(*unfold RNM′-def Lap-dist-list.simps*(*1*), *subst bind-return*[**where** *N =* (*count-space UNIV*)],*auto simp*: *argmax-list-def*)

lemma *RNM′-singleton*:
shows *RNM′* [*x*] *= return* (*count-space* (*UNIV* :: *nat set*)) *0*
proof−
have *RNM′* [*x*] = (*Lap-dist* (*1/ε*) *x*) ⪢= (*λx1. return* (*listM borel*) [] ⪢= (*λx2. return* (*listM borel*) (*x1 # x2*))) ⪢= (*λy. return* (*count-space UNIV*) (*argmax-list y*))
unfolding *RNM′-def* **by** *auto*
also have ... = (*Lap-dist* (*1/ε*) *x*) ⪢= (*λx1. return* (*listM borel*) (*x1 #* [])) ⪢= (*λy. return* (*count-space UNIV*) (*argmax-list y*))
by(*subst bind-return*, *measurable*)
also have ... = (*Lap-dist* (*1/ε*) *x*) ⪢= (*λx1. return* (*count-space*

UNIV) (*argmax-list* [*x1*]))
  proof(*subst bind-assoc*)
    show *Lap-dist* (*1* / ε) *x* $\gg=$ (λ*x*. *return* (*listM borel*) [*x*] $\gg=$ (λ*y*. *return* (*count-space UNIV*) (*argmax-list y*))) = *Lap-dist* (*1* / ε) *x* $\gg=$ (λ*x1*. *return* (*count-space UNIV*) (*argmax-list* [*x1*]))
      by(*subst bind-return*, *measurable*)
  qed *measurable*
  also have ... = *return* (*count-space* (*UNIV* :: *nat set*)) *0*
    by(*auto simp*: *bind-const′ subprob-space-return argmax-list-def*)
  finally show *?thesis***.**
qed

lemma *RNM′-support*:
  assumes $i \geq$ *length xs*
    and *xs* $\neq$ []
  shows *emeasure* (*RNM′ xs*){*i*} = *0*
  unfolding *RNM′-def*
proof−
  define *n* :: *nat* **where** *n*: *n* = *length xs*
  have *1*: *emeasure* (*Lap-dist-list* (*1*/ε) *xs*) {*zs*. *length zs* = *n*} = *1*
    using *emeasure-Lap-dist-list-length*[*of - xs*] *n* **by** *auto*
  show *emeasure* (*Lap-dist-list* (*1* / ε) *xs* $\gg=$ (λ*y*. *return* (*count-space UNIV*) (*argmax-list y*))) {*i*} = *0*
  proof(*subst Giry-Monad.emeasure-bind*)
    show *space* (*Lap-dist-list* (*1* / ε) *xs*) $\neq$ {}
      by(*auto simp*: *space-Lap-dist-list*)
    show {*i*} $\in$ *sets* (*count-space UNIV*)
      by *auto*
    show (λ*y*. *return* (*count-space UNIV*) (*argmax-list y*)) $\in$ *Lap-dist-list* (*1* / ε) *xs* $\rightarrow_M$ *subprob-algebra* (*count-space UNIV*)
      by *auto*
    show ($\int^+$ *x*. *emeasure* (*return* (*count-space UNIV*) (*argmax-list x*)) {*i*} ∂*Lap-dist-list* (*1* / ε) *xs*) = *0*
    proof(*subst Giry-Monad.emeasure-return*)
      show ($\int^+$ *x*. *indicator* {*i*} (*argmax-list x*) ∂*Lap-dist-list* (*1* / ε) *xs*) = *0*
      proof(*rule Distributions.nn-integral-zero′*,*rule AE-mp*)
        show *almost-everywhere* (*Lap-dist-list* (*1* / ε) *xs*) (λ*zs*. *zs* $\in${*zs*. *length zs* = *n*})
          by(*rule prob-space.AE-prob-1*,*auto simp*: *measure-def 1*)
        show *AE x*$\in${*zs*. *length zs* = *n*} *in Lap-dist-list* (*1* / ε) *xs*. *indicator* {*i*} (*argmax-list x*) = *0*
        proof(*rule AE-I2*,*rule impI*)
          fix *x* :: *real list* **assume** *x* $\in$ {*zs*. *length zs* = *n*}
          with *n assms* **show** *indicator* {*i*} (*argmax-list x*) = *0*
            by (*metis* (*mono-tags*, *lifting*) *One-nat-def Suc-diff-le argmax-list-le-length diff-Suc-1′ diff-is-0-eq indicator-simps*(*2*) *length-0-conv mem-Collect-eq not-less-eq-eq singletonD*)
        qed

qed
qed *fact*
qed
qed

The conditional distribution of RNM' **when** $n - 1$ **values of noise are fixed.** **definition** *RNM-M* :: *real list* ⇒ *real list* ⇒ *real* ⇒ *nat* ⇒ *nat measure* **where**
RNM-M cs rs d i = *do*{*r* ← (*Lap-dist0* (*1*/ε)); *return* (*count-space UNIV*) (*argmax-insert* (*r*+*d*) ((λ (*xs*,*ys*). (*map2* (+) *xs ys*))(*cs*, *rs*)) *i*)}

lemma *space-RNM-M*:
shows *space* (*RNM-M cs rs d i*) = (*UNIV* :: *nat set*)
unfolding *RNM-M-def Lap-dist0-def* **by** (*metis empty-not-UNIV sets-return space-count-space space-bind space-Lap-dist*)

lemma *sets-RNM-M*[*measurable-cong*]:
shows *sets* (*RNM-M cs rs d i*) = *sets* (*count-space* (*UNIV* :: *nat set*))
unfolding *RNM-M-def Lap-dist0-def* **by**(*metis sets-bind sets-return space-Lap-dist empty-not-UNIV*)

lemma *RNM-M-Nil*:
shows *emeasure* (*RNM-M* [] [] *d 0*) {*j* ∈ *UNIV*. *j* = *0*} = *1*
and *k* ≠ *0* ⟹ *emeasure* (*RNM-M* [] [] *d 0*) {*j* ∈ *UNIV*. *j* = *k*} = *0*
proof−
have *b*: *Lap-dist0* (*1*/ε) ⤜ (λ*r*. *return* (*count-space UNIV*) (*0* :: *nat*)) = *return* (*count-space UNIV*) (*0* :: *nat*)
proof(*rule measure-eqI*)
show *sets* (*Lap-dist0* (*1*/ε) ⤜ (λ*r*. *return* (*count-space UNIV*) *0*)) = *sets* (*return* (*count-space UNIV*) *0*)
unfolding *Lap-dist0-def* **by** (*metis countable-empty sets-bind space-Lap-dist uncountable-UNIV-real*)
show ⋀*A*. *A* ∈ *sets* (*Lap-dist0* (*1*/ε) ⤜ (λ*r*. *return* (*count-space UNIV*) *0*)) ⟹
emeasure (*Lap-dist0* (*1*/ε) ⤜ (λ*r*. *return* (*count-space UNIV*) *0*)) *A* = *emeasure* (*return* (*count-space UNIV*) *0*) *A*
unfolding *Lap-dist0-def* **by**(*subst emeasure-bind-const-prob-space*,*auto simp*: *subprob-space-return-ne*)
qed
{
fix *k* :: *nat*
have *measure* (*Lap-dist0* (*1*/ε) ⤜ (λ*r*. *return* (*count-space UNIV*) *0*)) {*j* :: *nat*. *j* = *k* ∧ *j* ∈ *space* (*Lap-dist0* (*1*/ε) ⤜ (λ*r*. *return* (*count-space UNIV*) *0*))}
= *measure* (*return* (*count-space UNIV*) *0*) {*j* :: *nat*. *j* = *k* ∧ *j* ∈ *space* (*return* (*count-space UNIV*) *0*)}

unfolding *b* **by** *standard*
also have ... = *measure* (*return* (*count-space UNIV*) *0*){*k* :: *nat*}
by *auto*
}**note** ∗ = *this*
show *emeasure* (*RNM-M* [] [] *d 0*) {*j* ∈ *UNIV*. *j* = *0*} = *1*
by(*auto simp*: *RNM-M-def argmax-insert-def argmax-list-def b list-insert-def*)
show *k* ≠ *0* ⟹ *emeasure* (*RNM-M* [] [] *d 0*) {*j* ∈ *UNIV*. *j* = *k*} = *0*
by(*auto simp*: *RNM-M-def argmax-insert-def argmax-list-def b list-insert-def*)
qed

lemma *RNM-M-Nil-indicator*:
shows *emeasure* (*RNM-M* [] [] *d 0*) = *indicator* {*A* :: *nat set*. *A* ∈ *UNIV* ∧ *0* ∈ *A*}
proof
fix *B* :: *nat set*
have *1*: ⋀ *k* :: *nat*. (*if* (*k* ∈ *B* ∧ *k* ≠ *0*) *then* {*k*} *else* {}) ∈ *null-sets* (*RNM-M* [] [] *d 0*)
proof(*split if-split*, *intro conjI impI*)
show ⋀*k*. *k* ∈ *B* ∧ *k* ≠ *0* ⟹ {*k*} ∈ *null-sets* (*RNM-M* [] [] *d 0*)
using *RNM-M-Nil*(*2*)[*of - d*] **by** *auto*
show ⋀*k*. ¬ (*k* ∈ *B* ∧ *k* ≠ *0*) ⟹ {} ∈ *null-sets* (*RNM-M* [] [] *d 0*)
by *auto*
qed
have *2*: ⋃ (*range* (λ *k*. *if* (*k* ∈ *B* ∧ *k* ≠ *0*) *then* {*k*} *else* {})) = {*k* :: *nat*. *k* ∈ *B* ∧ *k* ≠ *0*}
by *auto*
have {*k* :: *nat*. *k* ∈ *B* ∧ *k* ≠ *0*} ∈ *null-sets* (*RNM-M* [] [] *d 0*)
using *null-sets-UN'*[*ofUNIV* :: *nat set* (λ *k*. *if* (*k* ∈ *B* ∧ *k* ≠ *0*) *then* {*k*} *else* {}) (*RNM-M* [] [] *d 0*),*OF - 1*]
unfolding *2*[*THEN sym*] **by** *auto*
hence *b*: *emeasure* (*RNM-M* [] [] *d 0*) {*k* ∈ *B*. *k* ≠ *0*} = *0*
by *blast*
have *a*: *emeasure* (*RNM-M* [] [] *d 0*) {*0*} = *1*
using *RNM-M-Nil*(*1*)[*of d*] **by** *auto*
show *emeasure* (*RNM-M* [] [] *d 0*) *B* = *indicator* {*A* ∈ *UNIV*. *0* ∈ *A*} *B*
proof(*cases0* ∈ *B*)
case *True*
hence *B*: *B* = {*0*} ∪ {*k*. *k* ∈ *B* ∧ *k* ≠ *0*}**by** *auto*
hence *emeasure* (*RNM-M* [] [] *d 0*) *B* = *emeasure* (*RNM-M* [] [] *d 0*) ({*0*} ∪ {*k*. *k* ∈ *B* ∧ *k* ≠ *0*})
by *auto*
also have ... = *emeasure* (*RNM-M* [] [] *d 0*) {*0*} + *emeasure* (*RNM-M* [] [] *d 0*) {*k*. *k* ∈ *B* ∧ *k* ≠ *0*}
by(*rule plus-emeasure*[*THEN sym*],*auto simp*: *sets-RNM-M*)

```
    also have ... = 1
      using a b by auto
    finally show ?thesis using True by auto
  next
    case False
    hence B: B = {k. k ∈ B ∧ k ≠ 0}
      using Collect-mem-eq by fastforce
    hence emeasure (RNM-M [] [] d 0) B = emeasure (RNM-M [] []
d 0) ({k. k ∈ B ∧ k ≠ 0})
      by auto
    also have ... = 0
      using b by auto
    finally show ?thesis using False by auto
  qed
qed

lemma RNM-M-Nil-is-return-0:
  shows (RNM-M [] [] d 0) = return (count-space UNIV) 0
proof(rule measure-eqI)
  show sets (RNM-M [] [] d 0) = sets (return (count-space UNIV) 0)
    by (auto simp: sets-RNM-M)
  show ⋀A. A ∈ sets (RNM-M [] [] d 0) ⟹ emeasure (RNM-M [] []
d 0) A = emeasure (return (count-space UNIV) 0) A
    unfolding RNM-M-Nil-indicator sets-RNM-M emeasure-return
    by(split split-indicator,auto)
qed

lemma RNM-M-probability:
  fixes cs rs :: real list
  assumes length cs = n and length rs = n
    and i ≤ n
    and pose: ε > 0
  shows 𝒫(j in (RNM-M cs rs d i). j = i) = 𝒫(r in (Lap-dist0 (1/ε)).
ereal(r + d) ≥ (fst (max-argmax (map2 (+) cs rs))))
  using assms
proof−
  have 𝒫(j in (RNM-M cs rs d i). j = i) = measure (RNM-M cs rs d
i) {j :: nat | j. j∈ UNIV ∧ j = i}
    using space-RNM-M by auto
  also have ... = measure (Lap-dist0 (1/ε) ⤜ (λr. return (count-space
(UNIV :: nat set)) (argmax-insert (r + d) (map (λ(x, y). x + y) (zip
cs rs)) i))) {j :: nat | j. j∈ UNIV ∧ j = i}
    unfolding RNM-M-def by auto
  also have ... = LINT r|(Lap-dist0 (1/ε)). measure (return (count-space
(UNIV :: nat set)) (argmax-insert (r + d) (map (λ(x, y). x + y) (zip
cs rs)) i)) {j :: nat | j. j∈ UNIV ∧ j = i}
  proof(intro subprob-space.measure-bind subprob-space-Lap-dist0)
    show {j |j. j ∈ ⊤ ∧ j = i} ∈ sets (count-space (UNIV :: nat set))
      by auto
```

have *1*: *take i* (*map* (λ(*x*, *y*). *x* + *y*) (*zip cs rs*)) $\in$ *space* (*listM borel*)
unfolding *space-listM* **by** *auto*
have *2*: *drop i* (*map* (λ(*x*, *y*). *x* + *y*) (*zip cs rs*)) $\in$ *space* (*listM borel*)
unfolding *space-listM* **by** *auto*
hence *3*: (λ*x*. *drop i* (*map* (λ(*x*, *y*). *x* + *y*) (*zip cs rs*))) $\in$ *Lap-dist0* (*1*/ε) $\rightarrow_M$ *listM borel*
by *measurable*
have (λ*x*. [*x* + *d*] @ *drop i* (*map* (λ(*x*, *y*). *x* + *y*) (*zip cs rs*))) $\in$ *Lap-dist0* (*1*/ε) $\rightarrow_M$ *listM borel*
using *listM-Nil 2* **by** *measurable*
show (λ*r*. *return* (*count-space* $\top$) (*argmax-insert* (*r* + *d*) (*map* (λ(*x*, *y*). *x* + *y*) (*zip cs rs*)) *i*)) $\in$ *Lap-dist0* (*1*/ε) $\rightarrow_M$ *subprob-algebra* (*count-space* $\top$)
unfolding *argmax-insert-def* **using** *space-listM sets-Lap-dist measurable-cong-sets 1 listM-Nil 2* **by** *measurable*
qed
also have ... = *LINT r*|(*Lap-dist0* (*1*/ε)). (λ *r*. *indicator* {*j* :: *nat* | *j*. *j*$\in$ *UNIV* $\wedge$ *j* = *i*} (*argmax-insert* (*r* + *d*) (*map* (λ(*x*, *y*). *x* + *y*) (*zip cs rs*)) *i*)) *r*
by(*subst measure-return*,*auto*)
also have ... = *LINT r*|(*Lap-dist0* (*1*/ε)). *indicator* {*r* | *r* :: *real*. *argmax-insert* (*r* + *d*) ((*map2* (+) *cs rs*)) *i* = *i*} *r*
proof(*rule Bochner-Integration.integral-cong*)
show *Lap-dist0* (*1*/ε) = *Lap-dist0* (*1*/ε)
by *auto*
fix *x* :: *real* **assume** *x* $\in$ *space* (*Lap-dist0* (*1*/ε))
hence *x*: *x* $\in$ *UNIV*
by(*auto simp*: *space-Lap-dist*)
thus *indicator* {*j* |*j*. *j* $\in$ $\top$ $\wedge$ *j* = *i*} (*argmax-insert* (*x* + *d*) (*map* (λ(*x*, *y*). *x* + *y*) (*zip cs rs*)) *i*) = *indicator* {*r* |*r*. *argmax-insert* (*r* + *d*) (*map* (λ(*x*, *y*). *x* + *y*) (*zip cs rs*)) *i* = *i*} *x*
by(*split split-indicator*,*auto*)
qed
also have ... = *measure* (*Lap-dist0* (*1*/ε)) ({*r* | *r* :: *real*. *argmax-insert* (*r* + *d*) ((*map2* (+) *cs rs*)) *i* = *i*} $\cap$ *space* (*Lap-dist0* (*1*/ε)))
by *auto*
also have ... = *measure* (*Lap-dist0* (*1*/ε)) ({*r* | *r* :: *real*. *argmax-insert* (*r* + *d*) ((*map2* (+) *cs rs*)) *i* = *i*})
unfolding *space-Lap-dist0* **by** *auto*
also have ... = *measure* (*Lap-dist0* (*1*/ε)) {*r* | (*r* :: *real*). (*ereal* (*r* + *d*) $\geq$ (*fst* (*max-argmax* (*map2* (+) *cs rs*)))) $\wedge$ ((*ereal* (*r* + *d*) $\neq$ (*fst* (*max-argmax* (*drop i* (*map2* (+) *cs rs*))))))}
by(*subst argmax-insert-i-i′ assms* ,*auto simp*: *assms*)
also have ... = *measure* (*Lap-dist0* (*1*/ε)) ({*r* | (*r* :: *real*). (*ereal* (*r* + *d*) $\geq$ (*fst* (*max-argmax* (*map2* (+) *cs rs*))))}
− {*r* | (*r* :: *real*). (*ereal* (*r* + *d*) $\geq$ (*fst* (*max-argmax* (*map2* (+) *cs rs*)))) $\wedge$ (*ereal* (*r* + *d*) = (*fst* (*max-argmax* (*drop i* (*map2* (+) *cs*

rs)))))})
proof−
have {*r* | (*r* :: *real*). (*ereal* (*r* + *d*) ≥ (*fst* (*max-argmax* (*map2* (+) *cs rs*)))) ∧ ((*ereal* (*r* + *d*) ≠ (*fst* (*max-argmax* (*drop i* (*map2* (+) *cs rs*))))))}
= ({*r* | (*r* :: *real*). (*ereal* (*r* + *d*) ≥ (*fst* (*max-argmax* (*map2* (+) *cs rs*))))}
− {*r* | (*r* :: *real*). (*ereal* (*r* + *d*) ≥ (*fst* (*max-argmax* (*map2* (+) *cs rs*)))) ∧ (*ereal* (*r* + *d*) = (*fst* (*max-argmax* (*drop i* (*map2* (+) *cs rs*)))))})
by *auto*
thus *?thesis*
by *auto*
qed
also have ... = *measure* (*Lap-dist0* (*1*/*ε*)) {*r* | (*r* :: *real*). (*ereal* (*r* + *d*) ≥ (*fst* (*max-argmax* (*map2* (+) *cs rs*))))}
− *measure* (*Lap-dist0* (*1*/*ε*)) {*r* | (*r* :: *real*). (*ereal* (*r* + *d*) ≥ (*fst* (*max-argmax* (*map2* (+) *cs rs*)))) ∧ (*ereal* (*r* + *d*) = (*fst* (*max-argmax* (*drop i* (*map2* (+) *cs rs*)))))}
using *subprob-space-Lap-dist* **by**(*intro finite-measure.finite-measure-Diff*,*auto simp*: *subprob-space.axioms*(*1*))
also have ... = *measure* (*Lap-dist0* (*1*/*ε*)) {*r* | (*r* :: *real*). (*ereal* (*r* + *d*) ≥ (*fst* (*max-argmax* (*map2* (+) *cs rs*))))}
(**is***measure* (*Lap-dist0* (*1*/*ε*)) *?A* − *measure* (*Lap-dist0* (*1*/*ε*)) *?B* = *measure* (*Lap-dist0* (*1*/*ε*)) *?A*)
proof−
have *measure* (*Lap-dist0* (*1*/*ε*)) *?B* = *0*
proof−
from *pose* **have** (*Lap-dist0* (*1*/*ε*)) = (*density lborel* (*λx*. *ennreal* (*laplace-density* (*1 div ε*) *0 x*)))
unfolding *Lap-dist-def Lap-dist0-def* **by** *auto*
hence *2*: *absolutely-continuous lborel* (*Lap-dist0* (*1*/*ε*))
by(*auto intro*: *absolutely-continuousI-density*)
have *3*: *?B* = (*UNIV* :: *real set*) ∩ (*λr*. *ereal* (*r* + *d*)) −‘ {*r* . *r* ≥ (*fst* (*max-argmax* (*map2* (+) *cs rs*))) ∧ *r* = (*fst* (*max-argmax* (*drop i* (*map2* (+) *cs rs*))))}
(**is***?B* = *?C*)
by *auto*
have *?C* ∈ *null-sets lborel*
proof(*intro countable-imp-null-set-lborel countable-image-inj-Int-vimage*)
show *inj-on* (*λr* :: *real*. *ereal* (*r* + *d*)) ⊤
unfolding *inj-on-def* **by** (*metis add.commute add-left-cancel ereal.inject*)
show *countable* {*r*. *fst* (*max-argmax* (*map* (*λ*(*x*, *y*). *x* + *y*) (*zip cs rs*))) ≤ *r* ∧ *r* = *fst* (*max-argmax* (*drop i* (*map* (*λ*(*x*, *y*). *x* + *y*) (*zip cs rs*))))}
by(*rule countable-subset*,*auto*)
qed
hence *4*: *?B* ∈ *null-sets* (*Lap-dist0* (*1*/*ε*))

using *3 2* **unfolding** *absolutely-continuous-def* **by** *auto*
thus *measure* (*Lap-dist0* (*1*/ε)) *?B* = *0*
unfolding *null-sets-def* **using** *measure-eq-emeasure-eq-ennreal* [*of0* :: *real* (*Lap-dist0* (*1*/ε)) *?B*] **by** *auto*
qed
thus *?thesis*
by *auto*
qed
also have ... = $\mathcal{P}$(*r in* (*Lap-dist0* (*1*/ε)). *ereal*(*r* + *d*) $\geq$ (*fst* (*max-argmax* (*map2* (+) *cs rs*))))
by (*metis UNIV-I space-Lap-dist0*)
finally show *?thesis* **.**
qed

differential privacy inequality on *RNM-M* **lemma** *DP-RNM-M-i*:
fixes *xs ys rs* :: *real list* **and** *x y* :: *real* **and** *i n* :: *nat*
assumes *length xs* = *n* **and** *length ys* = *n* **and** *length rs* = *n*
and *adj'*: *x* $\geq$ *y* $\wedge$ *x* $\leq$ *y* + *1*
and *adj*: *list-all2* (λ *x y*. *x* $\geq$ *y* $\wedge$ *x* $\leq$ *y* + *1*) *xs ys*
and *i* $\leq$ *n*
shows $\mathcal{P}$(*j in* (*RNM-M xs rs x i*). *j* = *i*) $\leq$ (*exp* ε) $*$ $\mathcal{P}$(*j in* (*RNM-M ys rs y i*). *j* = *i*) $\wedge$ $\mathcal{P}$(*j in* (*RNM-M ys rs y i*). *j* = *i*) $\leq$ (*exp* ε) $*$ $\mathcal{P}$(*j in* (*RNM-M xs rs x i*). *j* = *i*)
proof(*casesn* = *0*)
case *True*
hence *xs*: *xs* = [] **and** *ys*: *ys* = [] **and** *rs*: *rs* = [] **and** *i*: *i* = *0*
using *assms* **by** *blast+*
from *pose* **have** *e*: *exp* ε $\geq$ *1*
by *auto*
thus *?thesis* **unfolding** *xs ys rs i RNM-M-Nil-is-return-0*
by (*auto simp*: *measure-return*)
next
case *False*
have *eq1*: $\bigwedge$*i* :: *nat*. *i* $\leq$ *n* $\Longrightarrow$ $\mathcal{P}$(*j in* (*RNM-M xs rs x i*). *j* = *i*) = $\mathcal{P}$(*r in* (*Lap-dist0* (*1*/ε)). *ereal*(*r* + *x*) $\geq$ (*fst* (*max-argmax* (*map2* (+) *xs rs*))))
using *RNM-M-probability assms pose* **by** *auto*
have *eq2*: $\bigwedge$*i* :: *nat*. *i* $\leq$ *n* $\Longrightarrow$ $\mathcal{P}$(*j in* (*RNM-M ys rs y i*). *j* = *i*) = $\mathcal{P}$(*r in* (*Lap-dist0* (*1*/ε)). *ereal*(*r* + *y*) $\geq$ (*fst* (*max-argmax* (*map2* (+) *ys rs*))))
using *RNM-M-probability assms pose* **by** *auto*
have *fin*: *finite-measure* (*Lap-dist0* (*1*/ε))
using *prob-space-Lap-dist0* **by** (*metis prob-space.finite-measure*)

show *?thesis*
using *False assms*
proof(*intro conjI*)
show $\mathcal{P}$(*j in RNM-M xs rs x i*. *j* = *i*) $\leq$ *exp* ε $*$ $\mathcal{P}$(*j in RNM-M ys rs y i*. *j* = *i*)

unfolding *eq1*[*OF assms*(*6*)] *eq2*[*OF assms*(*6*)]
proof−
have $\{r :: real.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \le ereal\ (r + x)\} \subseteq \{r :: real.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (r + x)\}$
using *fst-max-argmax-adj*[*ofxs-ys rs*] *assms* **by**(*intro subsetI*, *unfold mem-Collect-eq*,*auto*)
also have ... $\subseteq \{r :: real.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (r + y + 1)\}$
using *adj′ Collect-mono-iff le-ereal-le* **by** *force*
also have ... $= \{r \mid v\ r :: real.\ v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\}$
by (*auto simp*: *add.commute add.left-commute*)
finally have *ineq1*: $\{r.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \le ereal\ (r + x)\} \subseteq \{r \mid v\ r :: real.\ v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\}$.

have *meas1*: $\{z :: real.\ \exists\, v\ r.\ z = r \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\} \in sets\ (Lap\text{-}dist0\ (1/\varepsilon))$
by *auto*

have $measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{r.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \le ereal\ (r + x)\} \le measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{r \mid v\ r :: real.\ v = r + 1 \wedge fst\ (max\text{-}argmax(map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\}$
proof(*intro measure-mono-fmeasurable ineq1*)
show $\{z.\ \exists\, v\ r.\ z = r \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\} \in fmeasurable\ (Lap\text{-}dist0\ (1/\varepsilon))$
using *prob-space-Lap-dist finite-measure.fmeasurable-eq-sets fin meas1* **by** *blast*
show $\{r.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \le ereal\ (r + x)\} \in sets\ (Lap\text{-}dist0\ (1/\varepsilon))$
by *auto*
qed
also have ... $\le (exp\ \varepsilon) * measure\ (Lap\text{-}dist\ (1/\varepsilon)\ (-1))\ \{r \mid v\ r :: real.\ v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\} + (0 :: real)$
using *meas1* **unfolding** *Lap-dist0-def* **by**(*intro DP-divergence-leE meas1 DP-divergence-Lap-dist′-eps pose*, *auto*)
also have ... $= (exp\ \varepsilon) * measure\ (Lap\text{-}dist\ (1/\varepsilon)\ (-1))\ \{r \mid v\ r :: real.\ v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\}$
by *auto*
also have ... $= (exp\ \varepsilon) * measure\ ((Lap\text{-}dist0\ (1/\varepsilon)) \gg= (\lambda\ r.\ return\ borel\ (r + (-1))))\ \{r \mid v\ r :: real.\ v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\}$
unfolding *Lap-dist-def2*[*of* (*1* / ε) −*1*] *Lap-dist0-def* **by** *auto*
also have ... $= (exp\ \varepsilon) * measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{(r + 1) \mid v\ r :: real.\ v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\}$
proof(*subst subprob-space.measure-bind*)

show *subprob-space* $(Lap\text{-}dist0\ (1/\varepsilon))$
by *auto*
show $(\lambda r.\ return\ borel\ (r + (-1))) \in Lap\text{-}dist0\ (1/\varepsilon) \to_M$ *subprob-algebra borel*
by *auto*
show $exp\ \varepsilon * (LINT\ x|Lap\text{-}dist0\ (1/\varepsilon).\ measure\ (return\ borel\ (x + (-1)))\ \{z.\ \exists\, v\ r.\ z = r \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (v + y)\})$
$= exp\ \varepsilon * measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{z.\ \exists\, v\ r.\ z = r + 1 \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (v + y)\}$
using *meas1* **proof**(*subst measure-return*)
show $\{z.\ \exists\, v\ r.\ z = r \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (v + y)\} \in sets\ borel$
by *auto*
have $exp\ \varepsilon * (LINT\ x|Lap\text{-}dist0\ (1/\varepsilon).\ indicat\text{-}real\ \{z.\ \exists\, v\ r.\ z = r \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (v + y)\}\ (x + (-1))) = exp\ \varepsilon * (LINT\ x|Lap\text{-}dist0\ (1/\varepsilon).\ indicat\text{-}real\ \{z + 1 \mid z.\ \exists\, v\ r.\ z = r \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (v + y)\}\ x)$
proof(*subst Bochner-Integration.integral-cong*)
have *1*: $\bigwedge x.\ \forall z.\ x = z + 1 \longrightarrow \neg\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (z + 1 + y) \Longrightarrow indicat\text{-}real\ \{z.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (z + 1 + y)\}\ (x - 1) = 0$
proof−
fix $x :: real$ **assume** $\forall z :: real.\ x = z + 1 \longrightarrow \neg\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (z + 1 + y)$
hence $*$: $\bigwedge z.\ x = z + 1 \Longrightarrow \neg\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (x + y)$**by** *auto*
hence $\neg\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (x + y)$**using** $*[of x - 1]$ **by** *auto*
thus $indicat\text{-}real\ \{z.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (z + 1 + y)\}\ (x - 1) = 0$**by** *auto*
qed
show $\bigwedge x.\ indicat\text{-}real\ \{z.\ \exists\, v\ r.\ z = r \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (v + y)\}\ (x + - 1) = indicat\text{-}real\ \{z + 1 \mid z.\ \exists\, v\ r.\ z = r \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (v + y)\}\ x$
by(*split split-indicator*, *intro conjI impI allI*,*auto simp*: *1*)
qed(*auto*)
also have ... $= exp\ \varepsilon * measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{z + 1 \mid z.\ \exists\, v\ r.\ z = r \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (v + y)\}$
using *Bochner-Integration.integral-indicator space-Lap-dist0* **by** *auto*
also have ... $= exp\ \varepsilon * measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{z.\ \exists\, v\ r.\ z = r + 1 \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (v + y)\}$
by (*metis* (*no-types*, *lifting*) *add-cancel-right-right is-num-normalize*(*1*) *real-add-minus-iff*)

finally show $exp\ \varepsilon * (LINT\ x|Lap\text{-}dist0\ (1/\varepsilon).\ indicat\text{-}real\ \{z.\ \exists\, v\ r.\ z = r \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\}\ (x + - 1)) = exp\ \varepsilon * Sigma\text{-}Algebra.measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{z.\ \exists\, v\ r.\ z = r + 1 \wedge v = r + 1 \wedge fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\}$**.**

qed

qed(*auto*)

also have ... $= exp\ \varepsilon * Sigma\text{-}Algebra.measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (v + y)\}$

by (*metis* (*no-types*, *lifting*) *add-cancel-right-right is-num-normalize*(*1*) *real-add-minus-iff*)

finally have $Sigma\text{-}Algebra.measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{r.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \le ereal\ (r + x)\} \le exp\ \varepsilon * Sigma\text{-}Algebra.measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{r.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (r + y)\}$**by** *auto*

thus$\mathcal{P}(r\ in\ Lap\text{-}dist0\ (1/\varepsilon).\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \le ereal\ (r + x)) \le exp\ \varepsilon * \mathcal{P}(r\ in\ Lap\text{-}dist0\ (1/\varepsilon).\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (r + y))$

using *space-Lap-dist0* **by** *auto*

qed

next

show $\mathcal{P}(j\ in\ RNM\text{-}M\ ys\ rs\ y\ i.\ j = i) \le exp\ \varepsilon * \mathcal{P}(j\ in\ RNM\text{-}M\ xs\ rs\ x\ i.\ j = i)$

unfolding *eq1*[*OF assms*(*6*)] *eq2*[*OF assms*(*6*)]

proof−

have $\{r :: real.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (r + y)\} \subseteq \{r :: real.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \le ereal\ (r + x)\}$

using *assms adj′ Collect-mono-iff le-ereal-le* **by** *force*

also have ... $\subseteq \{r :: real.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) - 1 \le ereal\ (r + x)\}$

using *ereal-diff-add-transpose fst-max-argmax-adj*[*ofxs-ys rs*] *assms* **by** *fastforce*

also have ... $\subseteq \{v - (1 :: real)\ |v :: real.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \le ereal\ (v + x)\}$

proof(*rule subsetI*)

fix $z :: real$ **assume** $z \in \{r.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) - 1 \le ereal\ (r + x)\}$

hence $fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) - 1 \le ereal\ (z + x)$

by *blast*

hence $fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) - 1 + 1 \le ereal\ (z + x) + 1$

by (*metis add.commute add-left-mono*)

hence $fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) - 1 + 1 \le ereal\ ((z + 1) + x)$

by (*auto simp*: *add.commute add.left-commute*)

hence $fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \le ereal\ ((z + 1) + x)$

by (*metis add.commute add-0 add-diff-eq-ereal cancel-comm-monoid-add-class.diff-cancel ereal-minus*(*1*) *one-ereal-def zero-ereal-def*)

hence $z + (1 :: real) \in \{v :: real.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\}$
by *auto*
thus$z \in \{v - (1 :: real)\ |v :: real.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\}$
by *force*
qed
finally have *ineq2*: $\{r.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (r + y)\} \subseteq \{v - 1\ |v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\}$.

have $\{v - 1\ |v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\} = (\lambda r.\ r + 1)-`\ \{v\ |v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\}$
by *force*
also have ... $\in$ *sets* $(Lap\text{-}dist0\ (1/\varepsilon))$
by *auto*
finally have *meas2*[*measurable*]: $\{v - 1\ |v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\} \in sets\ (Lap\text{-}dist0\ (1/\varepsilon))$.

hence $measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{r.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (r + y)\} \leq measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{v - 1\ |v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\}$
proof(*intro ineq2 measure-mono-fmeasurable*)
show $\{r.\ fst\ (max\text{-}argmax\ (map2\ (+)\ ys\ rs)) \leq ereal\ (r + y)\} \in sets\ (Lap\text{-}dist0\ (1/\varepsilon))$
by *auto*
thus $\{v - 1\ |v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\} \in fmeasurable\ (Lap\text{-}dist0\ (1/\varepsilon))$
using *prob-space-Lap-dist finite-measure.fmeasurable-eq-sets fin meas2* **by** *blast*
qed
also have ... $\leq (exp\ \varepsilon) * measure\ (Lap\text{-}dist\ (1/\varepsilon)\ (-1))\ \{v - 1\ |v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\} + 0$
using *meas2* **unfolding** *Lap-dist0-def* **by** (*intro DP-divergence-leE DP-divergence-Lap-dist′-eps pose*, *auto*)
also have ... $= (exp\ \varepsilon) * measure\ ((Lap\text{-}dist0\ (1/\varepsilon)) \gg= (\lambda\ r.\ return\ borel\ (r + (-1))))\ \{v - 1\ |v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\}$
unfolding *Lap-dist-def2*[*of* $(1\ /\ \varepsilon)\ -1$] *Lap-dist0-def* **by** *auto*
also have ... $= (exp\ \varepsilon) * measure\ (Lap\text{-}dist0\ (1/\varepsilon))\ \{v\ |v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs)) \leq ereal\ (v + x)\}$
proof(*subst subprob-space.measure-bind*)
show *subprob-space* $(Lap\text{-}dist0\ (1/\varepsilon))$
by *auto*
show $(\lambda r.\ return\ borel\ (r + -1)) \in Lap\text{-}dist0\ (1/\varepsilon) \to_M subprob\text{-}algebra\ borel$
by *auto*
show *meas22*: $\{v - 1\ |v.\ fst\ (max\text{-}argmax\ (map2\ (+)\ xs\ rs))$

$\leq$ *ereal* $(v + x)\}$ $\in$ *sets borel*
using *meas2 sets-Lap-dist0* **by** *blast*
show *exp* ε $*$ (*LINT z*|*Lap-dist0* (*1*/ε). *measure* (*return borel* (*z* + $-$ *1*)) $\{v - 1 \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$) =
exp ε $*$ *measure* (*Lap-dist0* (*1*/ε)) $\{v \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$
proof(*subst measure-return*,*intro meas22*)
have *exp* ε $*$ (*LINT z*|*Lap-dist0* (*1*/ε). *indicat-real* $\{v - 1 \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$ (*z* + $-$ *1*)) =
exp ε $*$ (*LINT z*|*Lap-dist0* (*1*/ε). *indicat-real* $\{v \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$ *z*)
proof(*subst Bochner-Integration.integral-cong*)
fix *z* :: *real*
show *indicat-real* $\{v - 1 \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$ (*z* + $-$ *1*) = *indicat-real* $\{v \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$ *z*
unfolding *indicator-def* **by** *auto*
show *exp* ε $*$ *integral*^{L} (*Lap-dist0* (*1*/ε)) (*indicat-real* $\{v \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$) =
exp ε $*$ *integral*^{L} (*Lap-dist0* (*1*/ε)) (*indicat-real* $\{v \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$)
by *auto*
qed(*auto*)
also have ... = *exp* ε $*$ *measure* (*Lap-dist0* (*1*/ε)) $\{v \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$
using *Bochner-Integration.integral-indicator space-Lap-dist* **by** *auto*

finally show *exp* ε $*$ (*LINT z*|*Lap-dist0* (*1*/ε). *indicat-real* $\{v - 1 \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$ (*z* + $-$ *1*)) =
exp ε $*$ *Sigma-Algebra.measure* (*Lap-dist0* (*1*/ε)) $\{v \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$.
qed
qed

finally have *Sigma-Algebra.measure* (*Lap-dist0* (*1*/ε)) $\{r.$ *fst* (*max-argmax* (*map2* (+) *ys rs*)) $\leq$ *ereal* $(r + y)\}$
$\leq$ *exp* ε $*$ *Sigma-Algebra.measure* (*Lap-dist0* (*1*/ε)) $\{v \mid v.$ *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(v + x)\}$.
thus$\mathcal{P}$(*r in Lap-dist0* (*1*/ε). *fst* (*max-argmax* (*map2* (+) *ys rs*)) $\leq$ *ereal* $(r + y)$) $\leq$ *exp* ε $*$ $\mathcal{P}$(*r in Lap-dist0* (*1*/ε). *fst* (*max-argmax* (*map2* (+) *xs rs*)) $\leq$ *ereal* $(r + x)$)
using *space-Lap-dist0* **by** *auto*
qed
qed
qed

Aggregating the differential privacy inequalities on *RNM-M* to those of *RNM′* **lemma** *RNM′-expand*:
fixes *n* :: *nat*
assumes *length xs* = *n* **and** *i* ≤ *n*
shows (*RNM′* (*list-insert x xs i*)) = *do*{*rs* ← (*Lap-dist0-list* (*1* / *ε*) (*length xs*)); (*RNM-M xs rs x i*)}
using *assms*
proof(*induction n arbitrary*: *xs i*)
note [*measurable del*] = *borel-measurable-count-space*
case *0*
hence *xs*: *xs* = [] **and** *i*: *i* = *0*
by *auto*
have *Lap-dist0-list* (*1* / *ε*) *0* ⤜ (*λrs*. *RNM-M* [] *rs x i*) = (*return* (*listM borel*) []) ⤜ (*λrs*. *RNM-M* [] *rs x i*)
by *auto*
also have ... = *RNM-M* [] [] *x i*
by(*subst bind-return*[**where** *N* = (*count-space UNIV*)],*auto simp*: *space-listM RNM-M-def argmax-insert-def argmax-list-def comp-def case-prod-beta*)
also have ... = *return* (*count-space UNIV*) *0*
using *RNM-M-Nil-is-return-0 RNM-M-def argmax-insert-def* **unfolding** *list-insert-def* **by** *force*
also have ... = *RNM′* (*list-insert x* [] *i*)
unfolding *list-insert-def take.simps drop.simps append.simps RNM′-singleton* **by** *auto*
finally show *?case* **unfolding** *xs i* **by** *auto*
next
note *borel-measurable-count-space*[*measurable del*]
case (*Suc n*)
then obtain *a ys* **where** *xs*: *xs* = *a* # *ys* **and** *n*: *length ys* = *n*
by (*metis Suc-length-conv*)
have *xxsl*: ⋀*x*. *length* (*x* # *xs*) = *Suc* (*Suc n*)
using *n xs* **by** *auto*

have *Lap-mechanism-multi-replicate-insert*: ⋀*n k* :: *nat*. *k* ≤ *n* ⟹(*Lap-dist0-list* (*1* / *ε*) (*Suc n*)) = *Lap-dist0* (*1*/*ε*) ⤜ (*λz*. *Lap-dist0-list* (*1* / *ε*) *n* ⤜ (*λzs*. *return* (*listM borel*) (*list-insert z zs k*)))
proof−
fix *n k* :: *nat* **assume** *k* ≤ *n*
thus*Lap-dist0-list* (*1* / *ε*) (*Suc n*) = *Lap-dist0* (*1*/*ε*) ⤜ (*λz*. *Lap-dist0-list* (*1* / *ε*) *n* ⤜ (*λzs*. *return* (*listM borel*) (*list-insert z zs k*)))
proof(*induction n arbitrary*: *k*)
case *0*
hence *k*: *k* = *0*
by *auto*
thus *?case* **unfolding** *k list-insert-def* **unfolding** *Lap-dist0-def Lap-dist0-list.simps*(*2*) *append-Cons append-Nil drop-0 take-0* **by** *auto*
next
case (*Suc n*)

thus *?case* **proof**(*induction k*)
case *0*
thus *?case* **unfolding** *list-insert-def* **unfolding** *Lap-dist0-def Lap-dist0-list.simps*(*2*) *append-Cons append-Nil drop-0 take-0* **by** *auto*
next
case (*Suc k*)
hence $k \leq n$ **by** *auto*
hence *1*: *Lap-dist0-list* (*1* / ε) (*Suc n*) = *Lap-dist0* (*1*/ε) $\gg=$ (λz. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λzs. *return* (*listM borel*) (*list-insert z zs k*)))
using *Suc.prems*(*1*) **by** *auto*
have *Lap-dist0-list* (*1* / ε) (*Suc* (*Suc n*)) = *Lap-dist0* (*1*/ε) $\gg=$ (λy. *Lap-dist0-list* (*1* / ε) (*Suc n*) $\gg=$ (λys. *return* (*listM borel*) (*y* # *ys*)))
unfolding *Lap-dist0-list.simps*(*2*) *Lap-dist0-def* **by** *auto*
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λy. (*Lap-dist0* (*1*/ε) $\gg=$ (λz. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λzs. *return* (*listM borel*) (*list-insert z zs k*)))) $\gg=$ (λys. *return* (*listM borel*) (*y* # *ys*)))
using *1* **by** *auto*
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λy. (*Lap-dist0* (*1*/ε) $\gg=$ (λz. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λzs. (*return* (*listM borel*) (*list-insert z zs k*)) $\gg=$ (λys. *return* (*listM borel*) (*y* # *ys*))))))
proof(*subst bind-assoc*)
show $\bigwedge y$. (λz. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λzs. *return* (*listM borel*) (*list-insert z zs k*))) $\in$ *Lap-dist0* (*1*/ε) $\rightarrow_M$ *subprob-algebra* (*listM borel*)
unfolding *list-insert-def* **by** *measurable*
show $\bigwedge y$. (λys. *return* (*listM borel*) (*y* # *ys*)) $\in$ *listM borel* $\rightarrow_M$ *subprob-algebra* (*listM borel*)
by *measurable*
show *Lap-dist0* (*1*/ε) $\gg=$ (λy. *Lap-dist0* (*1*/ε) $\gg=$ (λx. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λzs. *return* (*listM borel*) (*list-insert x zs k*)) $\gg=$ (λys. *return* (*listM borel*) (*y* # *ys*)))) = *Lap-dist0* (*1*/ε) $\gg=$ (λy. *Lap-dist0* (*1*/ε) $\gg=$ (λz. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λzs. *return* (*listM borel*) (*list-insert z zs k*) $\gg=$ (λys. *return* (*listM borel*) (*y* # *ys*)))))
by(*subst bind-assoc*,*auto simp*:*list-insert-def*) *measurable*
qed
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λy. (*Lap-dist0* (*1*/ε) $\gg=$ (λz. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λzs. *return* (*listM borel*) (*y* # (*list-insert z zs k*))))))
by(*subst bind-return*,*auto simp*:*list-insert-def space-listM*) *measurable*
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λz. (*Lap-dist0* (*1*/ε) $\gg=$ (λy. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λzs. *return* (*listM borel*) ((*list-insert y* (*z* # *zs*) (*Suc k*)))))))
unfolding *list-insert-def* **by** *auto*
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λy. (*Lap-dist0* (*1*/ε) $\gg=$ (λz. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λzs. *return* (*listM borel*) ((*list-insert*

y (*z* # *zs*) (*Suc k*)))))))
proof(*subst Giry-Monad.pair-prob-space.bind-rotate*)
show *pair-prob-space* (*Lap-dist0* (*1*/ε)) (*Lap-dist0* (*1*/ε))
unfolding *pair-prob-space-def pair-sigma-finite-def prob-space-Lap-dist* **using** *prob-space-Lap-dist prob-space-imp-sigma-finite* **by** *auto*
show (λ(*x*, *y*). *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λ*zs*. *return* (*listM borel*) (*list-insert y* (*x* # *zs*) (*Suc k*)))) $\in$ *Lap-dist0* (*1*/ε) $\bigotimes_M$ *Lap-dist0* (*1*/ε) $\rightarrow_M$ *subprob-algebra* (*listM borel*)
unfolding *list-insert-def* **by** *auto*
qed(*auto*)
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λ*y*. (*Lap-dist0* (*1*/ε) $\gg=$ (λ*z*. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λ*zs*. (*return* (*listM borel*) (*z* # *zs*)) $\gg=$ (λ*zs2*. *return* (*listM borel*) ((*list-insert y zs2* (*Suc k*))))))))
by(*subst bind-return*,*auto simp*:*space-listM list-insert-def*) *measurable*
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λ*y*. (*Lap-dist0* (*1*/ε) $\gg=$ (λ*z*. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λ*zs*. (*return* (*listM borel*) (*z* # *zs*))) $\gg=$ (λ*zs2*. *return* (*listM borel*) ((*list-insert y zs2* (*Suc k*)))))))
by(*subst bind-assoc*,*auto simp*:*space-listM list-insert-def*) *measurable*
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λ*y*. *Lap-dist0-list* (*1* / ε) (*Suc n*) $\gg=$ (λ*zs2*. *return* (*listM borel*) ((*list-insert y zs2* (*Suc k*)))))
unfolding *Lap-dist0-list.simps*(*2*) **by**(*subst bind-assoc*[**where** *N* = *listM borel* **and** *R* = *listM borel*,*symmetric*],*auto simp*:*space-listM list-insert-def Lap-dist0-def*)
finally show *?case***.**
qed
qed
qed

{
fix *x* :: *real* **and** *xs* :: *real list* **assume** *lxs*: *length xs* = *Suc n*
have *xxsl*: $\bigwedge$*x*. *length* (*x* # *xs*) = *Suc* (*Suc n*)
using *lxs* **by** *auto*

have *RNM′* (*x* # *xs*) = *Lap-dist0-list* (*1* / ε) (*Suc* (*Suc n*)) $\gg=$ (λ*ys*. *return* (*listM borel*) (*map2* (+) (*x* # *xs*) *ys*)) $\gg=$ (λ*ys*. *return* (*count-space UNIV*) (*snd* (*max-argmax ys*)))
unfolding *RNM′-def Lap-dist-list-def2 argmax-list-def lxs xxsl* **by** *auto*
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λ*x1*. *Lap-dist0-list* (*1* / ε) (*Suc n*) $\gg=$ (λ*x2*. *return* (*listM borel*) (*x1* # *x2*))) $\gg=$ (λ*ys*. *return* (*listM borel*) (*map2* (+) (*x* # *xs*) *ys*)) $\gg=$ (λ*ys*. *return* (*count-space UNIV*) (*snd* (*max-argmax ys*)))
unfolding *Lap-dist0-list.simps*(*2*) *Lap-dist0-def* **by** *auto*
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λ*x1*. *Lap-dist0-list* (*1* / ε) (*Suc n*) $\gg=$ (λ*x2*. (*return* (*listM borel*) (*x1* # *x2*)) $\gg=$ (λ*ys*. *return* (*listM borel*) (*map2* (+) (*x* # *xs*) *ys*)))) $\gg=$ (λ*ys*. *return* (*count-space UNIV*) (*snd* (*max-argmax ys*)))

by(*subst bind-assoc*[**where** N =(*listM borel*) **and** R =(*listM borel*)],*measurable*)+
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λ*x1*. *Lap-dist0-list* (*1* / ε) (*Suc n*) $\gg=$ (λ*x2*. *return* (*listM borel*) (*map2* (+) (*x* # *xs*) (*x1* # *x2*)))) $\gg=$ (λ*ys*. *return* (*count-space UNIV*) (*snd* (*max-argmax ys*)))
by(*subst bind-return*,*measurable*)
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λ*x1*. *Lap-dist0-list* (*1* / ε) (*Suc n*) $\gg=$ (λ*x2*. *return* (*listM borel*) ((*x* + *x1*) # (*map2* (+) *xs* *x2*)))) $\gg=$ (λ*ys*. *return* (*count-space UNIV*) (*snd* (*max-argmax ys*)))
unfolding *list.map* **by** *auto*
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λ*x1*. *Lap-dist0-list* (*1* / ε) (*Suc n*) $\gg=$ (λ*x2*. (*return* (*listM borel*) ((*x* + *x1*) # (*map2* (+) *xs* *x2*))) $\gg=$ (λ*ys*. *return* (*count-space UNIV*) (*snd* (*max-argmax ys*)))))
by(*subst bind-assoc*[**where** N =(*listM borel*) **and** R =(*count-space UNIV*)],*measurable*)+
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λ*x1*. *Lap-dist0-list* (*1* / ε) (*Suc n*) $\gg=$ (λ*x2*. *return* (*count-space UNIV*) (*snd* (*max-argmax* ((*x* + *x1*) # (*map2* (+) *xs* *x2*))))))
by(*subst bind-return*,*measurable*)
also have ... = *Lap-dist0-list* (*1* / ε) (*Suc n*) $\gg=$ (λ*x2*. *Lap-dist0* (*1*/ε) $\gg=$ (λ*x1*. *return* (*count-space UNIV*) (*snd* (*max-argmax* ((*x* + *x1*) # (*map2* (+) *xs* *x2*))))))
proof(*subst Giry-Monad.pair-prob-space.bind-rotate*)
show *pair-prob-space* (*Lap-dist0* (*1*/ε)) (*Lap-dist0-list* (*1* / ε) (*Suc n*))
unfolding *pair-prob-space-def pair-sigma-finite-def*
by (*meson prob-space-Lap-dist0 prob-space-Lap-dist0-list prob-space-imp-sigma-finite*)
have (λ(*xa*, *y*). *return* (*count-space UNIV*) (*snd* (*max-argmax* ((*x* + *xa*) # *map2* (+) *xs* *y*)))) = (*return* (*count-space UNIV*)) *o* (λ *zs*. *snd* (*max-argmax zs*)) *o* (λ(*xa*, *y*). ((*x* + *xa*) # *map2* (+) *xs* *y*))
by *auto*
also have ... $\in$ *Lap-dist0* (*1*/ε) $\bigotimes_M$ *Lap-dist0-list* (*1* / ε) (*Suc n*) $\rightarrow_M$ *subprob-algebra* (*count-space UNIV*)
by(*intro measurable-comp*[**where** N = *listM borel*] *measurable-comp*[**where** N = (*count-space UNIV*)], *measurable*)
finally show (λ(*xa*, *y*). *return* (*count-space UNIV*) (*snd* (*max-argmax* ((*x* + *xa*) # *map2* (+) *xs* *y*)))) $\in$ *Lap-dist0* (*1*/ε) $\bigotimes_M$ *Lap-dist0-list* (*1* / ε) (*Suc n*) $\rightarrow_M$ *subprob-algebra* (*count-space UNIV*).
qed(*auto*)
finally have *RNM′* (*x* # *xs*) = *Lap-dist0-list* (*1* / ε) (*Suc n*) $\gg=$ (λ*x2*. *Lap-dist0* (*1*/ε) $\gg=$ (λ*x1*. *return* (*count-space UNIV*) (*snd* (*max-argmax* ((*x* + *x1*) # *map2* (+) *xs* *x2*))))).
}**note** $*$ = *this*

show *?case* **proof**(*casesi* = *0*)
case *True*
hence *xxs*: (*list-insert x xs i*) = *x* # *xs*
unfolding *list-insert-def* **by** *auto*
have *RNM′* (*list-insert x xs i*) = *Lap-dist0-list* (*1* / ε) (*Suc n*)

$\gg=$ ($\lambda x2$. *Lap-dist0* (*1*/ε) $\gg=$ ($\lambda x1$. *return* (*count-space UNIV*) (*snd* (*max-argmax* ((*x* + *x1*) # (*map2* (+) *xs x2*))))))
unfolding *xxs* **using** ∗[*of xs x*, *OF Suc*(*2*)] **by** *auto*
also have ... = *Lap-dist0-list* (*1* / ε) (*length xs*) $\gg=$ (λrs. *RNM-M xs rs x i*)
unfolding *RNM-M-def argmax-insert-def True drop-0 take-0 Suc.prems argmax-list-def comp-def list-insert-def*
by (*metis* (*no-types*, *lifting*) *Cons-eq-appendI add.commute case-prod-conv ext self-append-conv2*)
finally show *?thesis*.
next
case *False*
then obtain *k* :: *nat* **where** *i*: *i* = *Suc k*
by (*metis list-decode.cases*)
hence *insxxsi*: (*list-insert x xs i*) = *a* # (*list-insert x ys k*)
unfolding *list-insert-def* **using** *xs* **by** *auto*
have *klessn*: *k* ≤ *n*
using *n i* **using** *Suc.prems*(*2*) **by** *auto*
have *leninsxxsi*: *length* (*list-insert x ys k*) = *Suc n*
unfolding *list-insert-def length-append length-drop length-take n* **using** *klessn* **by** *auto*

have *RNM′* (*list-insert x xs i*) = *Lap-dist0-list* (*1* / ε) (*Suc n*)$\gg=$ ($\lambda x2$. *Lap-dist0* (*1*/ε) $\gg=$ ($\lambda x1$. *return* (*count-space UNIV*) (*snd* (*max-argmax* ((*a* + *x1*) # *map2* (+) (*list-insert x ys k*) *x2*)))))
unfolding *insxxsi* ∗[*of*(*list-insert x ys k*)*a*,*OF leninsxxsi*] **by** *auto*
also have ... = (*Lap-dist0* (*1*/ε) $\gg=$ (λp. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λps. *return* (*listM borel*) (*list-insert p ps k*)))) $\gg=$ ($\lambda x2$. *Lap-dist0* (*1*/ε) $\gg=$ ($\lambda x1$. *return* (*count-space UNIV*) (*snd* (*max-argmax* ((*a* + *x1*) # *map2* (+) (*list-insert x ys k*) *x2*)))))
unfolding *Lap-mechanism-multi-replicate-insert*[*of k n*, *OF klessn*] **by** *auto*
also have ... = *Lap-dist0* (*1*/ε) $\gg=$ (λp. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λps. (*return* (*listM borel*) (*list-insert p ps k*)) $\gg=$ ($\lambda x2$. *Lap-dist0* (*1*/ε) $\gg=$ ($\lambda x1$. *return* (*count-space UNIV*) (*snd* (*max-argmax* ((*a* + *x1*) # *map2* (+) (*list-insert x ys k*) *x2*)))))))
proof(*subst bind-assoc*[**where** *N* =(*listM borel*) **and** *R* =*count-space UNIV*])
show (λp. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λps. *return* (*listM borel*) (*list-insert p ps k*))) ∈ *Lap-dist0* (*1*/ε) $\rightarrow_M$ *subprob-algebra* (*listM borel*)
unfolding *list-insert-def* **by** *measurable*
show *1*: ($\lambda x2$. *Lap-dist0* (*1*/ε) $\gg=$ ($\lambda x1$. *return* (*count-space UNIV*) (*snd* (*max-argmax* ((*a* + *x1*) # *map2* (+) (*list-insert x ys k*) *x2*))))) ∈ *listM borel* $\rightarrow_M$ *subprob-algebra* (*count-space UNIV*)
by(*rule measurable-bind*[**where** *N* = *borel*],*auto*)
thus *Lap-dist0* (*1*/ε) $\gg=$
(λxa. *Lap-dist0-list* (*1* / ε) *n* $\gg=$ (λps. *return* (*listM borel*) (*list-insert xa ps k*)) $\gg=$

(λx2. Lap-dist0 (1/ε) ≫= (λx1. return (count-space UNIV) (snd (max-argmax ((a + x1) # map2 (+) (list-insert x ys k) x2)))))) =
Lap-dist0 (1/ε) ≫=
(λp. Lap-dist0-list (1 / ε) n ≫=
(λps. return (listM borel) (list-insert p ps k) ≫= (λx2. Lap-dist0 (1/ε) ≫= (λx1. return (count-space UNIV) (snd (max-argmax ((a + x1) # map2 (+) (list-insert x ys k) x2)))))))
unfolding *list-insert-def*
by(*subst bind-assoc*[**where** *N* =*listM borel* **and** *R* =*count-space UNIV*],*auto*)
qed
also have ... = *Lap-dist0* (*1/ε*) ≫= (λ*p. Lap-dist0-list* (*1* / ε) *n* ≫= (λ*ps. Lap-dist0* (*1/ε*) ≫= (λ*x1. return* (*count-space UNIV*) (*snd* (*max-argmax* ((*a* + *x1*) # *map2* (+) (*list-insert x ys k*) (*list-insert p ps k*)))))))
by(*subst bind-return*[**where** *N* = *count-space UNIV*],*auto*)
also have ... = *Lap-dist0* (*1/ε*) ≫= (λ*x1. Lap-dist0-list* (*1* / ε) *n* ≫= (λ*ps. Lap-dist0* (*1/ε*) ≫= (λ*p. return* (*count-space UNIV*) (*snd* (*max-argmax* (*map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*))))))))
unfolding *list-insert-def* **by** *auto*
also have ... = *Lap-dist0* (*1/ε*) ≫= (λ*x1. Lap-dist0-list* (*1* / ε) *n* ≫= (λ*ps. Lap-dist0* (*1/ε*) ≫= (λ*p. return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*)))))))
proof−
have *AE ps in Lap-dist0-list* (*1* / ε) *n*. ∀ *x1 p*. (*snd* (*max-argmax* (*map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*))))) = (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*))))
proof(*rule AE-mp*)
show ⋀*x1 p. almost-everywhere* (*Lap-dist0-list* (*1* / ε) *n*) (λ*ps. ps* ∈ {*ps. length ps* = *n*})
proof(*rule prob-space.AE-prob-1*)
show *prob-space* (*Lap-dist0-list* (*1* / ε) *n*)
by *auto*
show *Sigma-Algebra.measure* (*Lap-dist0-list* (*1* / ε) *n*) {*ps. length ps* = *n*} = *1*
unfolding *Lap-dist0-list-Lap-dist-list*[*symmetric*] **using** *emeasure-Lap-dist-list-length* [*of* (*1* / ε) (*replicate n 0*)] **unfolding** *measure-def length-replicate* **by** *auto*
qed
show *x*: *AE ps*∈{*ps. length ps* = *n*} *in Lap-dist0-list* (*1* / ε) *n*. ∀ *x1 p*.
snd (*max-argmax* (*map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*)))) =
snd (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*)))
proof(*rule AE-I2*,*rule impI*,*intro allI*)

fix *x1 p ps* **assume** *ps* ∈ *space* (*Lap-dist0-list* (*1* / *ε*) *n*) **and** *ps* ∈ {*ps*. *length ps* = *n*}
hence *length ps* = *n***by** *auto*
hence *l*: *length ps* = *length ys***using** *n* **by** *auto*
hence (*map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*))) = *map* (λ (*x*,*y*). *x* + *y*) (*zip* (*take* (*Suc k*) (*a* # *ys*) @ [*x*] @ *drop* (*Suc k*) (*a* # *ys*)) (*take* (*Suc k*) (*p* # *ps*) @ [*x1*] @ *drop* (*Suc k*) (*p* # *ps*)))
unfolding *list-insert-def* **by** *auto*
also have ... = *map* (λ (*x*,*y*). *x* + *y*) ((*zip* (*take* (*Suc k*) (*a* # *ys*))(*take* (*Suc k*) (*p* # *ps*))) @ [(*x*,*x1*)] @ (*zip* (*drop* (*Suc k*) (*a* # *ys*)))(*drop* (*Suc k*) (*p* # *ps*)))
using *l zip-append* **by** *auto*
also have ... = *map* (λ (*x*,*y*). *x* + *y*) ((*take* (*Suc k*) (*zip* (*a* # *ys*) (*p* # *ps*))) @ [(*x*,*x1*)] @ (*drop* (*Suc k*) (*zip* (*a* # *ys*)(*p* # *ps*))))
using *l take-zip drop-zip* **by** *metis*
also have ... = *list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*)
unfolding *list-insert-def map-append drop-map take-map* **by** *auto*
finally have *map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*)) = *list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*)**.**
thus*snd* (*max-argmax* (*map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*)))) = *snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*)))**by** *auto*
qed
qed
hence *x1*: ⋀ *x1*. *AE ps in Lap-dist0-list* (*1* / *ε*) *n*. (*Lap-dist0* (*1*/*ε*) ⤜ (λ*p*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*))))))) = (*Lap-dist0* (*1*/*ε*) ⤜ (λ*p*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*))))))
using *AE-mp* **by** *auto*
hence ∀ *x1*. *Lap-dist0-list* (*1* / *ε*) *n* ⤜
(λ*ps*. *Lap-dist0* (*1*/*ε*) ⤜ (λ*p*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*))))))) =
Lap-dist0-list (*1* / *ε*) *n* ⤜
(λ*ps*. *Lap-dist0* (*1*/*ε*) ⤜ (λ*p*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*))))))(**is**∀ *x1*. *?P x1*)
proof(*intro allI*)
fix *x1* **show** *?P x1*
proof(*rule bind-cong-AE′*[*of Lap-dist0-list* (*1* / *ε*) *n listM borel* (λ*ps*. *Lap-dist0* (*1*/*ε*) ⤜ (λ*p*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*))))))) (*count-space UNIV*) :: *nat measure* (λ*ps*.

Lap-dist0 (*1*/*ε*) ⤜ (*λp*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*)))))),*OF* - - - *x1*])
show *Lap-dist0-list* (*1* / *ε*) *n* ∈ *space* (*prob-algebra* (*listM borel*))
by *auto*
have *1*: *take* (*Suc k*) (*a* # *ys*) @ [*x*] @ *drop* (*Suc k*) (*a* # *ys*) ∈ *space* (*listM borel*)
using *space-listM-borel-UNIV* **by** *auto*
have *2*: [*x1*] ∈ *space* (*listM borel*)
by *auto*
show (*λps*. *Lap-dist0* (*1*/*ε*) ⤜ (*λp*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*))))))) ∈ *listM borel* $\rightarrow_M$ *prob-algebra* (*count-space UNIV*) **unfolding** *list-insert-def*
using *1 2* **by** *measurable*
have *3*: [*x* + *x1*] ∈ *space* (*listM borel*)
by *auto*
show (*λps*. *Lap-dist0* (*1*/*ε*) ⤜ (*λp*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*)))))) ∈ *listM borel* $\rightarrow_M$ *prob-algebra* (*count-space UNIV*)
unfolding *list-insert-def* **using** *1 3* **by** *measurable*
qed
qed
thus*Lap-dist0* (*1*/*ε*) ⤜
(*λx1*. *Lap-dist0-list* (*1* / *ε*) *n* ⤜
(*λps*. *Lap-dist0* (*1*/*ε*) ⤜ (*λp*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*map2* (+) (*list-insert x* (*a* # *ys*) (*Suc k*)) (*list-insert x1* (*p* # *ps*) (*Suc k*)))))))) =
Lap-dist0 (*1*/*ε*) ⤜
(*λx1*. *Lap-dist0-list* (*1* / *ε*) *n* ⤜
(*λps*. *Lap-dist0* (*1*/*ε*) ⤜ (*λp*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) (*p* # *ps*)) (*Suc k*)))))))
by(*auto elim*!: *bind-cong-AE′*)
qed

also have ... = *Lap-dist0* (*1*/*ε*) ⤜ (*λx1*. *Lap-dist0-list* (*1* / *ε*) *n* ⤜ (*λps*. *Lap-dist0* (*1*/*ε*) ⤜ (*λp*. (*return* (*listM borel*)(*p* # *ps*)) ⤜ (*λrs*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) *rs*) (*Suc k*))))))))
unfolding *list-insert-def* **by**(*subst bind-return*,*measurable*)
also have ... = *Lap-dist0* (*1*/*ε*) ⤜ (*λx1*. ((*Lap-dist0-list* (*1* / *ε*) *n* ⤜ (*λps*. *Lap-dist0* (*1*/*ε*) ⤜ (*λp*. (*return* (*listM borel*)(*p* # *ps*))) ⤜ (*λrs*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) *rs*) (*Suc k*)))))))))
unfolding *list-insert-def* **by**(*subst bind-assoc*,*measurable*)
also have ... = *Lap-dist0* (*1*/*ε*) ⤜ (*λx1*. ((*Lap-dist0-list* (*1* / *ε*) *n* ⤜ (*λps*. *Lap-dist0* (*1*/*ε*) ⤜ (*λp*. (*return* (*listM borel*)(*p* # *ps*))))

$\gg=$ (*λrs. return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) *rs*) (*Suc k*))))))))
unfolding *list-insert-def*
by(*subst bind-assoc*[**where** *N* = *listM borel* **and** *R* = *count-space UNIV*, *THEN sym*],*auto*)
also have ... = *Lap-dist0* (*1*/*ε*) $\gg=$ (*λx1*. (*Lap-dist0-list* (*1* / *ε*) (*Suc n*) $\gg=$ (*λrs. return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) *rs*) (*Suc k*)))))))
unfolding *Lap-dist0-list.simps Lap-dist0-def* **proof**(*subst pair-prob-space.bind-rotate* [**where** *N* = *listM borel*])
show *pair-prob-space* (*Lap-dist0-list* (*1* / *ε*) *n*) (*Lap-dist* (*1* / *ε*) *0*)
unfolding *pair-prob-space-def pair-sigma-finite-def Lap-dist-list-def2*[*symmetric*] **using** *prob-space-Lap-dist prob-space-Lap-dist*
by (*auto simp*: *prob-space-imp-sigma-finite*)
qed *auto*
also have ... = *Lap-dist0-list* (*1* / *ε*) (*Suc n*) $\gg=$ (*λrs. Lap-dist0* (*1*/*ε*) $\gg=$ (*λx1*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) *rs*) (*Suc k*))))))
proof(*subst pair-prob-space.bind-rotate*[**where** *N* = *count-space UNIV*])
show *pair-prob-space* (*Lap-dist0* (*1* / *ε*)) (*Lap-dist0-list* (*1* / *ε*) (*Suc n*))
unfolding *pair-prob-space-def pair-sigma-finite-def* **using** *prob-space-Lap-dist0 prob-space-Lap-dist0-list prob-space-imp-sigma-finite* **by** *metis*
show (*λ*(*xa*, *y*). *return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *xa*) (*map2* (+) (*a* # *ys*) *y*) (*Suc k*))))) ∈ *Lap-dist0* (*1*/*ε*) $\bigotimes_M$ *Lap-dist0-list* (*1* / *ε*) (*Suc n*) $\rightarrow_M$ *subprob-algebra* (*count-space UNIV*)
unfolding *list-insert-def listM-Nil* **by** *measurable*
qed(*auto*)
finally have ∗: *RNM′* (*list-insert x xs i*) = *Lap-dist0-list* (*1* / *ε*) (*Suc n*) $\gg=$ (*λrs. Lap-dist0* (*1* / *ε*) $\gg=$ (*λx1*. *return* (*count-space UNIV*) (*snd* (*max-argmax* (*list-insert* (*x* + *x1*) (*map2* (+) (*a* # *ys*) *rs*) (*Suc k*)))))) **.**

show *?thesis* **unfolding** *RNM-M-def* ∗ *argmax-insert-def case-prod-beta argmax-list-def comp-def*
by (*simp add*: *i n xs Suc.prems list-insert-def Groups.add-ac*(*2*))
qed
qed

lemma *DP-RNM′-M-i*:
fixes *xs ys* :: *real list* **and** *i n* :: *nat*
assumes *lxs*: *length xs* = *n*
and *lys*: *length ys* = *n*
and *adj*: *list-all2* (*λ x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) *xs ys*
shows $\mathcal{P}$(*j in* (*RNM′ xs*). *j* = *i*) ≤ (*exp ε*) ∗ $\mathcal{P}$(*j in* (*RNM′ ys*). *j* = *i*) ∧ $\mathcal{P}$(*j in* (*RNM′ ys*). *j* = *i*) ≤ (*exp ε*) ∗ $\mathcal{P}$(*j in* (*RNM′ xs*). *j* = *i*)

using *assms* **proof**(*casesn* = *0*)
case *True*
hence *xs*: *xs* = [] **and** *ys*: *ys* = []
using *lxs lys* **by** *blast+*
have *0*: *RNM′* [] = *return* (*count-space UNIV*) *0***using** *RNM′-Nil*
by *auto*
thus *?thesis* **proof**(*casesi* = *0*)
case *True*
hence *02*: $\mathcal{P}$(*j in local.RNM′* []. *j* = *0*) = *1***unfolding** *0 measure-def* **by** *auto*
show *?thesis* **unfolding** *xs ys True 02* **using** *pose* **by** *auto*
next
case *False*
hence *02*: $\mathcal{P}$(*j in local.RNM′* []. *j* = *i*) = *0***unfolding** *0 measure-def*
by *auto*
show *?thesis* **unfolding** *xs ys 02* **by** *auto*
qed
next
case *False*
have *space-Lap-dist-multi* [*simp*]: ⋀ *xs b*. *space* (*Lap-dist-list b xs*)
= *UNIV*
by (*auto simp*: *sets-Lap-dist-list sets-eq-imp-space-eq*)
have *space-RNM′-UNIV*[*simp*]: ⋀*xs*. *space* (*local.RNM′ xs*) = *UNIV*
unfolding *RNM′-def* **by** (*metis empty-not-UNIV sets-return space-Lap-dist-multi space-bind space-count-space*)
hence *space-RNM′-UNIV2*[*simp*]: ⋀*xs i*. {*j*. *j*∈ *space* (*RNM′ xs*) ∧
j = *i*} = {*i*}
by *auto*

thus *?thesis* **proof**(*casesi* < *n*)
case *True*
define *x* **where** *x*: *x* = *nth xs i*
define *y* **where** *y*: *y* = *nth ys i*
define *xs2* **where** *xs2*: *xs2* = (*list-exclude i xs*)
define *ys2* **where** *ys2*: *ys2* = (*list-exclude i ys*)
have *xs*: *xs* = (*list-insert x xs2 i*)
by (*metis length-0-conv length-greater-0-conv list-insert′-is-list-insert lxs cong-nth-total-nth False True insert-exclude x xs2*)
have *ys*: *ys* = (*list-insert y ys2 i*)
by (*metis length-0-conv length-greater-0-conv list-insert′-is-list-insert lys cong-nth-total-nth False True insert-exclude y ys2*)
obtain *m* :: *nat* **where** *n*: *n* = *Suc m***using** *True False not0-implies-Suc*
by *blast*
have *lxs2*: *length xs2* = *m*
by (*metis One-nat-def True add.assoc add.commute add-left-imp-eq id-take-nth-drop length-append list.size*(*4*) *list-exclude-def lxs n plus-1-eq-Suc xs2*)
have *lys2*: *length ys2* = *m*
by (*metis One-nat-def True add.assoc add.commute add-left-imp-eq*

id-take-nth-drop length-append list.size(*4*) *list-exclude-def lys n plus-1-eq-Suc ys2*)
  have *ilessm*: $i \leq m$
   using *True n* **by** *auto*
  have *1*: *take i xs2* = *take i xs*
   unfolding *xs2 list-exclude-def* **by** *auto* **have** *2*: *take i ys2* = *take i ys*
   unfolding *ys2 list-exclude-def* **by** *auto*
  have *3*: *drop i xs2* = *drop* (*Suc i*) *xs*
  by (*metis1append-eq-append-conv append-take-drop-id list-exclude-def xs2*)
  have *4*: *drop i ys2* = *drop* (*Suc i*) *ys*
  by (*metis2append-eq-append-conv append-take-drop-id list-exclude-def ys2*)
  from *adj* **have** *5*: *list-all2* (λ *x y*. $x \geq y \wedge x \leq y + 1$) (*take i xs2*)(*take i ys2*) **and** *6*: *list-all2* (λ *x y*. $x \geq y \wedge x \leq y + 1$) (*drop i xs2*)(*drop i ys2*)
   unfolding *1 2 3 4* **using** *list-all2-takeI list-all2-dropI* **by** *blast*+
  have *adj'*: $x \geq y \wedge x \leq y + 1$
   unfolding *x y* **using** *list-all2-conv-all-nth True False lxs lys adj* **by** (*metis* (*mono-tags*, *lifting*))
  have *adj2*: *list-all2* (λ *x y*. $x \geq y \wedge x \leq y + 1$) *xs2 ys2*
   using *5 6 lxs2 lys2 append-take-drop-id*[*of i xs2*,*THEN sym*] *append-take-drop-id*[*of i ys2*,*THEN sym*] *list-all2-appendI* **by** *fastforce*
  note $*$ = *DP-RNM-M-i*[*of xs2 m ys2 - y x*, *OF lxs2 lys2 - adj' adj2 ilessm*]
   have *7*: *local.RNM' xs* = (*Lap-dist0-list* (*1* /ε) *m*) $\gg=$ (λ*rs*. *RNM-M xs2 rs x i*)
   using *RNM'-expand*[*of xs2 m i x*,*OF lxs2 ilessm*] **unfolding** *xs lxs2* **by** *auto*
  have *8*: *local.RNM' ys* = (*Lap-dist0-list* (*1* /ε) *m*) $\gg=$ (λ*rs*. *local.RNM-M ys2 rs y i*)
   using *RNM'-expand*[*of ys2 m i y*,*OF lys2 ilessm*] **unfolding** *ys lys2* **by** *auto*

  have $\mathcal{P}$(*j in local.RNM' xs*. *j* = *i*) = *measure* (*Lap-dist0-list* (*1* /ε) *m* $\gg=$ (λ*rs*. *local.RNM-M xs2 rs x i*)) {*i*}
   by (*metis7space-RNM'-UNIV2*)
 also have ... = (*LINT rs*|*Lap-dist0-list* (*1* /ε) *m*. *measure* (*local.RNM-M xs2 rs x i*) {*i*})
  proof(*rule subprob-space.measure-bind*)
   show *subprob-space* (*Lap-dist0-list* (*1* /ε) *m*)
   by (*metis prob-space-Lap-dist0-list prob-space-imp-subprob-space*)
   thus (λ*rs*. *local.RNM-M xs2 rs x i*) $\in$ (*Lap-dist0-list* (*1* /ε) *m*) $\rightarrow_M$ *subprob-algebra* (*count-space UNIV*)
   unfolding *RNM-M-def argmax-insert-def argmax-list-def* **using** *list-insert-def* **by** *auto*
   show {*i*} $\in$ *sets* (*count-space UNIV*)
    by *auto*

qed
also have ... = ($\int$ *rs* ∈ *space* (*Lap-dist0-list* (*1* /ε) *m*). *measure* (*local.RNM-M xs2 rs x i*) {*i*} ∂(*Lap-dist0-list* (*1* /ε) *m*))
unfolding *set-lebesgue-integral-def*
by (*metis* (*no-types*, *lifting*) *Bochner-Integration.integral-cong indicator-eq-1-iff scaleR-simps*(*12*))
also have ... = ($\int$ *rs* ∈ {*xs. length xs* = *m*} . *measure* (*local.RNM-M xs2 rs x i*) {*i*} ∂(*Lap-dist0-list* (*1* /ε) *m*))
proof(*subst set-integral-null-delta*)
show *integrable* (*Lap-dist0-list* (*1* /ε) *m*) (λ*rs. Sigma-Algebra.measure* (*local.RNM-M xs2 rs x i*) {*i*})
by(*rule probability-kernel-evaluation-integrable*[**where** *M* =*count-space UNIV*],*auto simp*: *RNM-M-def argmax-insert-def prob-space.finite-measure*)
show *1*: {*xs. length xs* = *m*} ∈ *sets* (*Lap-dist0-list* (*1* /ε) *m*)
unfolding *sets-Lap-dist0-list* **using** *sets-listM-length*[*of borel m*] **by** *auto*
show *space* (*Lap-dist0-list* (*1* /ε) *m*) ∈ *sets* (*Lap-dist0-list* (*1* /ε) *m*)
by *blast*
have *emeasure* (*Lap-dist0-list* (*1* /ε) *m*) (*space* (*Lap-dist0-list* (*1* /ε) *m*)) = *1*
by(*rule prob-space.emeasure-space-1*,*auto*)
hence *emeasure* (*Lap-dist0-list* (*1* /ε) *m*) (*space* (*Lap-dist0-list* (*1* /ε) *m*) − {*xs. length xs* = *m*}) = *0*
using *emeasure-Lap-dist-list-length*[*of* (*1* /ε) *replicate m 0*] **unfolding** *length-replicate length-map Lap-dist0-list-Lap-dist-list*[*symmetric*] *space-Lap-dist-multi*
proof(*subst emeasure-Diff*)
have *space* (*Lap-dist-list* (*1* / ε) (*replicate m 0*)) ∈ *sets* (*Lap-dist-list* (*1* / ε) (*replicate m 0*))
by *blast*
thus *UNIV* ∈ *sets* (*Lap-dist-list* (*1* / ε) (*replicate m 0*))
unfolding *sets-Lap-dist-list* **by** *auto*
show {*xs. length xs* = *m*} ∈ *sets* (*Lap-dist-list* (*1* / ε) (*replicate m 0*))
unfolding *sets-Lap-dist-list* **using** *sets-listM-length*[*of borel m*] *space-borel lists-UNIV* **by** *auto*
qed(*auto*)
thus *sym-diff* (*space* (*Lap-dist0-list* (*1* / ε) *m*)) {*xs. length xs* = *m*} ∈ *null-sets* (*Lap-dist0-list* (*1* / ε) *m*)
by (*metis1Diff-mono null-setsI sets.compl-sets sets.sets-into-space sup.orderE*)
qed(*auto*)
finally have *9*: $\mathcal{P}$(*j in RNM′ xs. j* = *i*) = ($\int$ *rs*∈{*xs. length xs* = *m*}. *Sigma-Algebra.measure* (*RNM-M xs2 rs x i*) {*i*}∂*Lap-dist0-list* (*1* / ε) *m*).

have $\mathcal{P}$(*j in local.RNM′ ys. j* = *i*) = *measure* (*Lap-dist0-list* (*1*

$/\varepsilon)$ $m \gg= (\lambda rs.\ local.RNM\text{-}M\ ys2\ rs\ y\ i))\ \{i\}$
by (*metis8space-RNM′-UNIV2*)
also have ... = *LINT rs*| *Lap-dist0-list* ($1\ /\varepsilon$) *m*. *measure* (*local.RNM-M ys2 rs y i*) $\{i\}$
proof(*rule subprob-space.measure-bind*)
show *subprob-space* (*Lap-dist0-list* ($1\ /\varepsilon$) *m*)
by (*metis prob-space-Lap-dist0-list prob-space-imp-subprob-space*)
show ($\lambda rs.$ *local.RNM-M ys2 rs y i*) $\in$ *Lap-dist0-list* ($1\ /\varepsilon$) *m* $\rightarrow_M$ *subprob-algebra* (*count-space UNIV*)
unfolding *RNM-M-def argmax-insert-def* **by**(*auto simp*: *list-insert-def*)
show $\{i\} \in$ *sets* (*count-space UNIV*)**by** *auto*
qed
also have ... = ($\int$ *rs* $\in$ *space* (*Lap-dist0-list* ($1\ /\varepsilon$) *m*) . *measure* (*local.RNM-M ys2 rs y i*) $\{i\}$ ∂(*Lap-dist0-list* ($1\ /\varepsilon$) *m*))
unfolding *set-lebesgue-integral-def*
by (*metis* (*no-types*, *lifting*) *Bochner-Integration.integral-cong indicator-eq-1-iff scaleR-simps*(*12*))
also have ... = ($\int$ *rs* $\in$ $\{$*ys. length ys* = *m*$\}$. *measure* (*local.RNM-M ys2 rs y i*) $\{i\}$ ∂(*Lap-dist0-list* ($1\ /\varepsilon$) *m*))
proof(*subst set-integral-null-delta*)
show *integrable* (*Lap-dist0-list* ($1\ /\varepsilon$) *m*) ($\lambda rs.$ *Sigma-Algebra.measure* (*local.RNM-M ys2 rs y i*) $\{i\}$)
by(*rule probability-kernel-evaluation-integrable*[**where** *M* =*count-space UNIV*],*auto simp*: *RNM-M-def argmax-insert-def prob-space.finite-measure*)
show *1*: $\{$*ys. length ys* = *m*$\}$ $\in$ *sets* (*Lap-dist0-list* ($1\ /\varepsilon$) *m*)
unfolding *sets-Lap-dist0-list* **using** *sets-listM-length*[*of borel m*] **by** *auto*
show *space* (*Lap-dist0-list* ($1\ /\varepsilon$) *m*) $\in$ *sets* (*Lap-dist0-list* ($1\ /\varepsilon$) *m*)
by *blast*
have *emeasure* (*Lap-dist0-list* ($1\ /\varepsilon$) *m*) (*space* (*Lap-dist0-list* ($1\ /\varepsilon$) *m*)) = *1*
by(*rule prob-space.emeasure-space-1*,*auto*)
hence *emeasure* (*Lap-dist0-list* ($1\ /\varepsilon$) *m*) (*space* (*Lap-dist0-list* ($1\ /\varepsilon$) *m*) $-$ $\{$*ys. length ys* = *m*$\}$) = *0*
using *emeasure-Lap-dist-list-length*[*of* ($1\ /\varepsilon$) *replicate m 0*]
unfolding *length-replicate length-map Lap-dist0-list-Lap-dist-list*[*symmetric*] *space-Lap-dist-multi*
proof(*subst emeasure-Diff*)
have *space* (*Lap-dist-list* ($1\ /\ \varepsilon$) (*replicate m 0*)) $\in$ *sets* (*Lap-dist-list* ($1\ /\ \varepsilon$) (*replicate m 0*))
by *blast*
thus *UNIV* $\in$ *sets* (*Lap-dist-list* ($1\ /\ \varepsilon$) (*replicate m 0*))
unfolding *sets-Lap-dist-list*[*of* ($1\ /\ \varepsilon$) (*replicate m 0*)] *space-Lap-dist-list*[*of* ($1\ /\ \varepsilon$) (*replicate m 0*)] *space-listM space-borel lists-UNIV* **by** *auto*
show $\{$*xs. length xs* = *m*$\}$ $\in$ *sets* (*Lap-dist-list* ($1\ /\ \varepsilon$) (*replicate m 0*))
unfolding *sets-Lap-dist-list*[*of* ($1\ /\ \varepsilon$) (*replicate m 0*)]

using *sets-listM-length*[*of borel m*] *space-borel lists-UNIV* **by** *auto*
qed(*auto*)
thus *sym-diff* (*space* (*Lap-dist0-list* (*1* / ε) *m*)) {*xs. length xs* = *m*} $\in$ *null-sets* (*Lap-dist0-list* (*1* / ε) *m*)
by (*metis1Diff-mono null-setsI sets.compl-sets sets.sets-into-space sup.orderE*)
qed(*auto*)
finally have *10*: $\mathcal{P}$(*j in local.RNM′ ys. j* = *i*) = ($\int$ *rs*$\in${*ys. length ys* = *m*}. *Sigma-Algebra.measure* (*local.RNM-M ys2 rs y i*) {*i*}∂*Lap-dist0-list* (*1* / ε) *m*).

have *c*: *finite-measure* (*Lap-dist0-list* (*1* / ε) *m*)
by (*auto simp*: *prob-space.finite-measure*)

show *?thesis*
proof(*rule conjI*)
show $\mathcal{P}$(*j in local.RNM′ xs. j* = *i*) $\leq$ *exp* ε $*$ $\mathcal{P}$(*j in local.RNM′ ys. j* = *i*)
unfolding *9 10*
proof(*subst set-integral-mult-right*[*THEN sym*],*intro set-integral-mono*)
show $\bigwedge$*rs. rs* $\in$ {*xs. length xs* = *m*} $\Longrightarrow$ *Sigma-Algebra.measure* (*local.RNM-M xs2 rs x i*) {*i*} $\leq$ *exp* ε $*$ *Sigma-Algebra.measure* (*local.RNM-M ys2 rs y i*) {*i*}
using *conjunct1*[*OF* $*$] *space-RNM-M* **by** *auto*

show *set-integrable* (*Lap-dist0-list* (*1* / ε) *m*) {*xs. length xs* = *m*} (λ*xa. Sigma-Algebra.measure* (*local.RNM-M xs2 xa x i*) {*i*})
unfolding *set-integrable-def real-scaleR-def*
proof(*intro integrable-ess-bounded-finite-measure c ess-bounded-mult indicat-real-ess-bounded probability-kernel-evaluation-ess-bounded*)
show {*xs. length xs* = *m*} $\in$ *sets* (*Lap-dist0-list* (*1* / ε) *m*)
unfolding *Lap-dist0-list-Lap-dist-list*[*symmetric*] *sets-Lap-dist-list* **using** *sets-listM-length*[*of borel m*] **by** *auto*
show (λ*xa. local.RNM-M xs2 xa x i*) $\in$ *Lap-dist0-list* (*1* / ε) *m* $\rightarrow_M$ *prob-algebra* (*count-space UNIV*)
unfolding *RNM-M-def argmax-insert-def* **by**(*auto simp*: *list-insert-def*)
qed(*auto*)
show *set-integrable* (*Lap-dist0-list* (*1* / ε) *m*) {*xs. length xs* = *m*} (λ*x. exp* ε $*$ *Sigma-Algebra.measure* (*local.RNM-M ys2 x y i*) {*i*})
unfolding *set-integrable-def real-scaleR-def*
proof(*intro integrable-ess-bounded-finite-measure c ess-bounded-mult indicat-real-ess-bounded probability-kernel-evaluation-ess-bounded*)
show {*xs. length xs* = *m*} $\in$ *sets* (*Lap-dist0-list* (*1* / ε) *m*)
unfolding *Lap-dist0-list-Lap-dist-list*[*symmetric*] *sets-Lap-dist-list* **using** *sets-listM-length*[*of borel m*] **by** *auto*
show (λ*x. local.RNM-M ys2 x y i*) $\in$ *Lap-dist0-list* (*1* / ε) *m* $\rightarrow_M$ *prob-algebra* (*count-space UNIV*)

unfolding *RNM-M-def argmax-insert-def* **by**(*auto simp*: *list-insert-def*)
qed(*auto*)
qed
show $\mathcal{P}$(*j in local.RNM′ ys. j* = *i*) ≤ *exp* ε ∗ $\mathcal{P}$(*j in local.RNM′ xs. j* = *i*)
unfolding *9 10*
proof(*subst set-integral-mult-right*[*THEN sym*],*intro set-integral-mono*)
show ⋀*rs. rs* ∈ {*ys. length ys* = *m*} ⟹ *Sigma-Algebra.measure* (*local.RNM-M ys2 rs y i*) {*i*} ≤ *exp* ε ∗ *Sigma-Algebra.measure* (*local.RNM-M xs2 rs x i*) {*i*}
using *conjunct2*[*OF* ∗] *space-RNM-M* **by** *auto*
show *set-integrable* (*Lap-dist0-list* (*1* / ε) *m*) {*ys. length ys* = *m*} (λ*x. Sigma-Algebra.measure* (*local.RNM-M ys2 x y i*) {*i*})
unfolding *set-integrable-def real-scaleR-def*
proof(*intro integrable-ess-bounded-finite-measure c ess-bounded-mult indicat-real-ess-bounded probability-kernel-evaluation-ess-bounded*)
show {*xs. length xs* = *m*} ∈ *sets* (*Lap-dist0-list* (*1* / ε) *m*)
unfolding *Lap-dist0-list-Lap-dist-list*[*symmetric*] *sets-Lap-dist-list* **using** *sets-listM-length*[*of borel m*] **by** *auto*
show (λ*x. local.RNM-M ys2 x y i*) ∈ *Lap-dist0-list* (*1* / ε) *m* $\rightarrow_M$ *prob-algebra* (*count-space UNIV*)
unfolding *RNM-M-def argmax-insert-def* **by**(*auto simp*: *list-insert-def*)
qed(*auto*)
show *set-integrable* (*Lap-dist0-list* (*1* / ε) *m*) {*ys. length ys* = *m*} (λ*xa. exp* ε ∗ *Sigma-Algebra.measure* (*local.RNM-M xs2 xa x i*) {*i*})
unfolding *set-integrable-def real-scaleR-def*
proof(*intro integrable-ess-bounded-finite-measure c ess-bounded-mult indicat-real-ess-bounded probability-kernel-evaluation-ess-bounded*)
show {*xs. length xs* = *m*} ∈ *sets* (*Lap-dist0-list* (*1* / ε) *m*)
unfolding *Lap-dist0-list-Lap-dist-list*[*symmetric*] *sets-Lap-dist-list* **using** *sets-listM-length*[*of borel m*] **by** *auto*
show (λ*xa. local.RNM-M xs2 xa x i*) ∈ *Lap-dist0-list* (*1* / ε) *m* $\rightarrow_M$ *prob-algebra* (*count-space UNIV*)
unfolding *RNM-M-def argmax-insert-def* **by**(*auto simp*: *list-insert-def*)
qed(*auto*)
qed
qed
next
case *False*
hence *xsl*: *i* ≥ *length xs* **and** *ysl*: *i* ≥ *length ys*
using *lxs lys* **by** *auto*
have *xs′*: *xs* ≠ [] **and** *ys′*: *ys* ≠ []**using** ‹*n* ≠ *0*› *assms* **by** *blast*+
hence *1*: $\mathcal{P}$(*j in local.RNM′ xs. j* = *i*) = *0***using** *RNM′-support*[*of xs i*, *OF xsl xs′*] **unfolding** *measure-def* **by** *auto*
hence *2*: $\mathcal{P}$(*j in local.RNM′ ys. j* = *i*) = *0***using** *RNM′-support*[*of*

ys i, *OF ysl ys'*] **unfolding** *measure-def* **by** *auto*
  show *?thesis* **unfolding** *1 2* **by** *auto*
 qed
qed

lemma *DP-divergence-RNM'*:
 fixes *xs ys* :: *real list*
 assumes *adj*: *list-all2* (λ *x y*. $x \geq y \wedge x \leq y + 1$) *xs ys*
 shows *DP-divergence* (*RNM' xs*) (*RNM' ys*) $\varepsilon \leq 0 \wedge$ *DP-divergence* (*RNM' ys*) (*RNM' xs*) $\varepsilon \leq 0$
proof−
 have *sets-RNM'*: $\bigwedge$*xs*. *sets* (*local.RNM' xs*) = *sets* (*count-space UNIV*)
  unfolding *RNM'-def*
  by (*metis emeasure-empty indicator-eq-0-iff indicator-eq-1-iff prob-space.emeasure-space-1 prob-space-Lap-dist-list sets-bind sets-return*)
 hence *space-RNM'*: $\bigwedge$*xs*. *space* (*local.RNM' xs*) = *UNIV*
  by (*metis sets-eq-imp-space-eq space-count-space*)

 have *0*: $\bigwedge$ *xs A*. *emeasure* (*local.RNM' xs*) *A* = *ennreal* (*Sigma-Algebra.measure* (*local.RNM' xs*) *A*)
 proof(*intro finite-measure.emeasure-eq-measure*)
  fix *xs* :: *real list* **and** *A* :: *nat set***have** *xs* $\in$ *space*(*listM borel*)**by** *auto*
  show *finite-measure* (*local.RNM' xs*)
   by (*meson* ‹*xs* $\in$ *space* (*listM borel*)› *measurable-RNM' measurable-prob-algebraD subprob-space.axioms*(*1*) *subprob-space-kernel*)
 qed

 have *length xs* = *length ys*
  using *adj* **by**(*auto simp*: *list-all2-lengthD*)
 then obtain *n* :: *nat* **where** *lxs*: *length xs* = *n* **and** *lys*: *length ys* = *n*
  by *auto*
 have *1*: $\bigwedge$*i* :: *nat*. $\mathcal{P}$(*j in* (*RNM' xs*). *j* = *i*) $\leq$ (*exp* ε) $*$ $\mathcal{P}$(*j in* (*RNM' ys*). *j* = *i*)
 $\wedge$ $\mathcal{P}$(*j in* (*RNM' ys*). *j* = *i*) $\leq$ (*exp* ε) $*$ $\mathcal{P}$(*j in* (*RNM' xs*). *j* = *i*)
  using *DP-RNM'-M-i*[*OF lxs lys adj*] **by** *auto*
 hence $\bigwedge$*i* :: *nat*. $\mathcal{P}$(*j in* (*RNM' xs*). *j* = *i*) $\leq$ (*exp* ε) $*$ $\mathcal{P}$(*j in* (*RNM' ys*). *j* = *i*)
  by *auto*
 hence $\bigwedge$*n* :: *nat*. *enn2real*(*measure* (*local.RNM' xs*) {*n*}) $\leq$ *enn2real*((*exp* ε) $*$ *measure* (*local.RNM' ys*){*n*})
  unfolding *measure-def* **by** (*auto simp*: *space-RNM'*)
 hence *ineq1*: $\bigwedge$*n* :: *nat*. *emeasure* (*local.RNM' xs*) {*n*} $\leq$ (*exp* ε) $*$ *emeasure* (*local.RNM' ys*){*n*}
  unfolding *0*
   by (*metis* (*no-types*, *opaque-lifting*)*0Sigma-Algebra.measure-def dual-order.trans enn2real-nonneg ennreal-enn2real ennreal-leI ennreal-mult''*

linorder-not-le top-greatest)
from *1* **have** $\bigwedge i :: nat.\ \mathcal{P}(j\ in\ (RNM'\ ys).\ j = i) \le (exp\ \varepsilon) * \mathcal{P}(j\ in\ (RNM'\ xs).\ j = i)$
by *auto*
hence $\bigwedge n :: nat.\ enn2real(measure\ (local.RNM'\ ys)\ \{n\}) \le enn2real((exp\ \varepsilon) * measure\ (local.RNM'\ xs)\{n\})$
unfolding *measure-def* **by** (*auto simp*: *space-RNM'*)
hence *ineq2*: $\bigwedge n :: nat.\ emeasure\ (local.RNM'\ ys)\ \{n\} \le (exp\ \varepsilon) * emeasure\ (local.RNM'\ xs)\{n\}$
unfolding *0*
by (*metis* (*no-types*, *opaque-lifting*) *0Sigma-Algebra.measure-def dual-order.trans enn2real-nonneg ennreal-enn2real ennreal-leI ennreal-mult'' linorder-not-le top-greatest*)

{
fix *A* :: *nat set* **and** *xs* :: *real list*
have $emeasure\ (local.RNM'\ xs)\ A = (\sum i.\ emeasure\ (local.RNM'\ xs)\ ((\lambda j :: nat.\ if\ j \in A\ then\ \{j\}\ else\ \{\})\ i))$
proof(*subst suminf-emeasure*)
show $range\ (\lambda i :: nat.\ if\ i \in A\ then\ \{i\}\ else\ \{\}) \subseteq sets\ (local.RNM'\ xs)$
using *sets-RNM'* **by** *auto*
show $disjoint\text{-}family\ (\lambda i.\ if\ i \in A\ then\ \{i\}\ else\ \{\})$
unfolding *disjoint-family-on-def* **by** *auto*
show $emeasure\ (local.RNM'\ xs)\ A = emeasure\ (local.RNM'\ xs)\ (\bigcup i.\ if\ i \in A\ then\ \{i\}\ else\ \{\})$**by** *auto*
qed
also have $... = (\sum i.\ emeasure\ (local.RNM'\ xs)\{i\} * indicator\ A\ i)$
by(*rule suminf-cong*,*auto*)
finally have $emeasure\ (local.RNM'\ xs)\ A = (\sum i.\ emeasure\ (local.RNM'\ xs)\ \{i\} * indicator\ A\ i)$**.**
}note $*$ = *this*

show *?thesis* **proof**(*rule conjI*)
have $DP\text{-}divergence\ (local.RNM'\ xs)\ (local.RNM'\ ys)\ \varepsilon \le (0 :: real)$
proof(*rule DP-divergence-leI*)
fix *A* **assume** $A \in sets\ (local.RNM'\ xs)$
have $emeasure\ (local.RNM'\ xs)\ A = (\sum i.\ emeasure\ (local.RNM'\ xs)\{i\} * indicator\ A\ i)$
using $*$[*of xs A*] **by** *auto*
also have $... \le (\sum i.\ (exp\ \varepsilon) * (emeasure\ (local.RNM'\ ys)\{i\} * indicator\ A\ i))$
proof(*rule suminf-le*)
show $summable\ (\lambda i.\ emeasure\ (local.RNM'\ xs)\ \{i\} * indicator\ A\ i)$**by** *auto*
show $summable\ (\lambda i.\ ennreal\ (exp\ \varepsilon) * (emeasure\ (local.RNM'\ ys)\{i\} * indicator\ A\ i))$**by** *auto*

show $\bigwedge n.$ *emeasure* (*local.RNM′ xs*) $\{n\}$ $*$ *indicator A n* $\leq$ *ennreal* (*exp* ε) $*$ (*emeasure* (*local.RNM′ ys*) $\{n\}$ $*$ *indicator A n*)
using *ineq1*
by (*metis ab-semigroup-mult-class.mult-ac*(*1*) *antisym-conv1 mult-right-mono nless-le zero-less-iff-neq-zero*)
qed
also have ... = *ennreal* (*exp* ε) $*$ ($\sum$ *i.* (*emeasure* (*local.RNM′ ys*) $\{i\}$ $*$ *indicator A i*))
by *auto*
also have ... = *ennreal* (*exp* ε) $*$ *emeasure* (*local.RNM′ ys*) *A*
using $*$[*of ys A*] **by** *auto*
finally have *emeasure* (*local.RNM′ xs*) *A* $\leq$ *ennreal* (*exp* ε) $*$ *emeasure* (*local.RNM′ ys*) *A*.
thus*Sigma-Algebra.measure* (*local.RNM′ xs*) *A* $\leq$ *exp* ε $*$ *Sigma-Algebra.measure* (*local.RNM′ ys*) *A* + *0*
by (*metis0add-cancel-right-right enn2real-eq-posreal-iff enn2real-leI ennreal-mult′′ exp-ge-zero measure-nonneg split-mult-pos-le zero-less-measure-iff*)
qed
thus*DP-divergence* (*local.RNM′ xs*) (*local.RNM′ ys*) ε $\leq$ *0***using** *zero-ereal-def* **by** *auto*
next
have *DP-divergence* (*local.RNM′ ys*) (*local.RNM′ xs*) ε $\leq$ (*0* :: *real*)
proof(*rule DP-divergence-leI*)
fix *A* **assume** $A \in$ *sets* (*local.RNM′ ys*)
have *emeasure* (*local.RNM′ ys*) *A* = ($\sum$ *i. emeasure* (*local.RNM′ ys*)$\{i\}$ $*$ *indicator A i*)
using $*$[*of ys A*] **by** *auto*
also have ... $\leq$ ($\sum$ *i.* (*exp* ε) $*$ (*emeasure* (*local.RNM′ xs*)$\{i\}$ $*$ *indicator A i*))
proof(*rule suminf-le*)
show *summable* ($\lambda i.$ *emeasure* (*local.RNM′ ys*) $\{i\}$ $*$ *indicator A i*)**by** *auto*
show *summable* ($\lambda i.$ *ennreal* (*exp* ε) $*$ (*emeasure* (*local.RNM′ xs*)$\{i\}$ $*$ *indicator A i*))**by** *auto*
show $\bigwedge n.$ *emeasure* (*local.RNM′ ys*) $\{n\}$ $*$ *indicator A n* $\leq$ *ennreal* (*exp* ε) $*$ (*emeasure* (*local.RNM′ xs*) $\{n\}$ $*$ *indicator A n*)
using *ineq2*
by (*metis ab-semigroup-mult-class.mult-ac*(*1*) *antisym-conv1 mult-right-mono nless-le zero-less-iff-neq-zero*)
qed
also have ... = *ennreal* (*exp* ε) $*$ ($\sum$ *i.* (*emeasure* (*local.RNM′ xs*) $\{i\}$ $*$ *indicator A i*))
by *auto*
also have ... = *ennreal* (*exp* ε) $*$ *emeasure* (*local.RNM′ xs*) *A*
using $*$[*of xs A*] **by** *auto*
finally have *emeasure* (*local.RNM′ ys*) *A* $\leq$ *ennreal* (*exp* ε) $*$ *emeasure* (*local.RNM′ xs*) *A*.
thus*Sigma-Algebra.measure* (*local.RNM′ ys*) *A* $\leq$ *exp* ε $*$ *Sigma-Algebra.measure*

(*local.RNM′ xs*) *A* + *0*
by (*metis0add-cancel-right-right enn2real-eq-posreal-iff enn2real-leI ennreal-mult′′ exp-ge-zero measure-nonneg split-mult-pos-le zero-less-measure-iff*)
qed
thus*DP-divergence* (*local.RNM′ ys*) (*local.RNM′ xs*) *ε* ≤ *0***using** *zero-ereal-def* **by** *auto*
qed
qed

lemma *differential-privacy-LapMech-RNM′*:
shows *differential-privacy RNM′* {(*xs*, *ys*) | *xs ys*. *list-all2* (*λ x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) *xs ys* } *ε 0*
unfolding *differential-privacy-def DP-inequality-cong-DP-divergence*
proof(*rule ballI*)
fix *x*::*real list* × *real list* **assume** *x* ∈ {(*xs*, *ys*) |*xs ys*. *list-all2* (*λx y*. *y* ≤ *x* ∧ *x* ≤ *y* + *1*) *xs ys*}
then obtain *xs ys* **where** *x*: *x* = (*xs*,*ys*) **and** *1*: *list-all2* (*λx y*. *y* ≤ *x* ∧ *x* ≤ *y* + *1*) *xs ys*
by *blast*
thus *case x of* (*d1*, *d2*) ⇒ *DP-divergence* (*RNM′ d1*) (*RNM′ d2*) *ε* ≤ *ereal 0* ∧ *DP-divergence* (*RNM′ d2*) (*RNM′ d1*) *ε* ≤ *ereal 0*
using *DP-divergence-RNM′*[*of xs ys*]
by (*simp add*: *zero-ereal-def*)
qed

corollary *differential-privacy-LapMech-RNM′-sym*:
shows *differential-privacy RNM′* {(*xs*, *ys*) | *xs ys*. *list-all2* (*λ x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) *xs ys* ∨ *list-all2* (*λ x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) *ys xs* } *ε 0*
proof−
have *1*: {(*xs*, *ys*) | *xs ys*. *list-all2* (*λ x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) *xs ys* ∨ *list-all2* (*λ x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) *ys xs* }
= {(*xs*, *ys*) | *xs ys*. *list-all2* (*λ x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) *xs ys* } ∪ *converse* {(*xs*, *ys*) | *xs ys*. *list-all2* (*λ x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) *xs ys* }
by *blast*
show *?thesis* **unfolding** *1* **proof**(*rule differential-privacy-symmetrize*)
show *differential-privacy RNM′* {(*xs*, *ys*) |*xs ys*. *list-all2* (*λx y*. *y* ≤ *x* ∧ *x* ≤ *y* + *1*) *xs ys*} *ε 0*
by (*rule differential-privacy-LapMech-RNM′*)
qed
qed

theorem *differential-privacy-LapMech-RNM*:
shows *differential-privacy* (*LapMech-RNM ε*) *adj ε 0*
unfolding *LapMech-RNM-RNM′-c*
proof(*rule differential-privacy-preprocessing*)
show (*0*::*real*) ≤ *ε*
using *pose* **by** *auto*
show *differential-privacy RNM′* {(*xs*, *ys*) | *xs ys*. *list-all2* (*λ x y*. *x*

$\geq y \wedge x \leq y + 1$) *xs ys* $\vee$ *list-all2* (λ *x y*. $x \geq y \wedge x \leq y + 1$) *ys xs*
} ε *0*
by (*rule differential-privacy-LapMech-RNM′-sym*)
show $c \in M \rightarrow_M$ *listM borel*
using *c* **by** *auto*
show $\forall (x, y) \in adj.$ $(c\ x, c\ y) \in \{(xs, ys)$ |*xs ys*. *list-all2* (λx *y*. $y \leq x \wedge x \leq y + 1$) *xs ys* $\vee$ *list-all2* (λx *y*. $y \leq x \wedge x \leq y + 1$) *ys xs*}
using *cond adj* **by** *auto*
show $adj \subseteq$ *space M* $\times$ *space M*
using *adj*.
show {(*xs*, *ys*) |*xs ys*. *list-all2* (λx *y*. $y \leq x \wedge x \leq y + 1$) *xs ys* $\vee$ *list-all2* (λx *y*. $y \leq x \wedge x \leq y + 1$) *ys xs*} $\subseteq$ *space* (*listM borel*) $\times$ *space* (*listM borel*)
unfolding *space-listM space-borel lists-UNIV* **by** *auto*
qed

end

end

10.2 Formal Proof of Exact [Claim 3.9,AFDP]

In the above theorem ⟦*Lap-Mechanism-RNM-mainpart ?M ?adj ?c*; $0 < ?\varepsilon$⟧ $\Longrightarrow$ *differential-privacy* (*Lap-Mechanism-RNM-mainpart.LapMech-RNM ?c ?ε*) *?adj ?ε 0*, the query *c* to add noise is abstracted. We here instantiate it with the counting query. Thus we here formalize and show the monotonicity and Lipschitz property of it.

locale *Lap-Mechanism-RNM-counting* =
fixes *n*::*nat*
and *m*::*nat*
and *Q* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
assumes $\bigwedge i.\ i \in \{0..<m\} \Longrightarrow (Q\ i) \in UNIV \rightarrow UNIV$
begin

interpretation *results-AFDP n*
by(*unfold-locales*)

interpretation *L1-norm-list* (*UNIV*::*nat set*) (λ *x y*. |*int x* $-$ *int y*|) *n*
by(*unfold-locales*,*auto*)

lemma *adjacency-1-int-list*:
assumes (*xs*,*ys*) $\in$ *adj-L1-norm*
shows (*xs* = *ys*) $\vee$ ($\exists$ *as a b bs*. *xs* = *as* @ (*a* # *bs*) $\wedge$ *ys* = *as* @ (*b* # *bs*) $\wedge$ |*int a* $-$ *int b*| = *1*)
using *assms* **unfolding** *adj-L1-norm-def sp-Dataset-def space-restrict-space dist-L1-norm-list-def space-L1-norm-list-def*
proof(*induction n arbitrary*: *xs ys*)

case *0*
hence $xs = []$ **and** $ys = []$ **by** *auto*
then show *?case* **by** *auto*
next
case (*Suc n*)
hence *length xs = Suc n*
and *length ys = Suc n*
and *sum*:($\sum i = 1..Suc\ n.\ real\text{-}of\text{-}int\ |int\ (xs\ !\ (i - 1)) - int\ (ys\ !\ (i - 1))|) \leq 1$
by *blast+*
then obtain *x y ::nat* **and** *xs2 ys2 ::nat list*
where *xs*: $xs = x\ \#\ xs2$
and *ys*: $ys = y\ \#\ ys2$
and *xs2*: *length xs2 = n*
and *ys2*:*length ys2 = n*
by (*meson length-Suc-conv*)
hence *1*: $|int\ x - int\ y| + (\sum i = 1..n.\ real\text{-}of\text{-}int\ |int\ (xs2\ !\ (i - 1)) - int\ (ys2\ !\ (i - 1))|) = (\sum i = 1..Suc\ n.\ real\text{-}of\text{-}int\ |int\ (xs\ !\ (i - 1)) - int(ys\ !\ (i - 1))|)$
using *L1-norm-list.list-sum-dist-Cons*[*symmetric*, *OF xs2 ys2*] **by** *auto*
then show *?case*
proof(*cases x = y*)
case *True*
hence $|x - y| = 0$
by *auto*
with *1 sum* **have** *2*: $(\sum i = 1..n.\ real\text{-}of\text{-}int\ |int(xs2\ !\ (i - 1)) - int(ys2\ !\ (i - 1))|) \leq 1$
by *auto*
have $*$: $(xs2, ys2) \in \{(xs, ys)\ |xs\ ys.\ xs \in \{xs \in lists\ UNIV.\ length\ xs = n\} \wedge ys \in \{xs \in lists\ UNIV.\ length\ xs = n\} \wedge (\sum i = 1..n.\ real\text{-}of\text{-}int\ |int(xs\ !\ (i - 1)) - int(ys\ !\ (i - 1))|) \leq 1\}$
using *xs2 ys2 2* **by** *auto*
from *Suc.IH*[*of xs2 ys2*] $*$ **have** $(xs2 = ys2) \vee (\exists\ as\ a\ b\ bs.\ xs2 = as\ @\ (a\ \#\ bs) \wedge ys2 = as\ @\ (b\ \#\ bs) \wedge |int\ a - int\ b| = 1)$
unfolding *space-listM space-count-space* **by** *auto*
with *True xs ys* **show** *?thesis*
by (*meson Cons-eq-appendI*)
next
case *False*
hence *0*: $1 \leq |int\ x - int\ y|$
by *auto*
with *1 sum* **have** *2*: $(\sum i = 1..n.\ real\text{-}of\text{-}int\ |int(xs2\ !\ (i - 1)) - int(ys2\ !\ (i - 1))|) \leq 0$
by *auto*
have $(\sum i = 1..n.\ real\text{-}of\text{-}int\ |int(xs2\ !\ (i - 1)) - int(ys2\ !\ (i - 1))|) \geq 0$
by *auto*
with *2* **have** *3*:$(\sum i = 1..n.\ real\text{-}of\text{-}int\ |int(xs2\ !\ (i - 1)) - int(ys2$

! $(i - 1))|) = 0$
      by *argo*
    with *sum 1* **have** $|int\ x - int\ y| \leq 1$
      by *auto*
    with *0* **have** *4*: $|int\ x - int\ y| = 1$
      by *auto*
    hence $xs2 = ys2$ **using** *xs2 ys2 3*
     proof(*subst L1-norm-list.dist-L1-norm-list-zero*[*of UNIV* λ *x y. real-of-int* $|int\ x - int\ y|$ *xs2 n ys2* ,*symmetric*])
      show *1*: *L1-norm-list UNIV* ($\lambda x\ y.$ *real-of-int* $|int\ x - int\ y|$)
        by(*unfold-locales*)
      show ($\sum i = 1..n.$ *real-of-int* $|int(xs2\ !\ (i - 1)) - int(ys2\ !\ (i - 1))|) = 0 \Longrightarrow$ *L1-norm-list.dist-L1-norm-list* ($\lambda x\ y.$ *real-of-int* $|int\ x - int\ y|$) *n xs2 ys2* $= 0$
        unfolding *L1-norm-list.dist-L1-norm-list-def*[*OF 1*] **by** *simp*
    qed(*auto*)
    with *False xs ys 4* **show** *?thesis* **by** *blast*
  qed
qed

formalization of a list of counting query **primrec** *counting'*::*nat* ⇒ *nat* ⇒ *nat list* ⇒ *nat* **where**
  counting' i 0 - = 0 |
  counting' i (*Suc k*) *xs* = (*if Q i k then* (*nth-total 0 xs k*) *else 0*) + *counting' i k xs*

lemma *measurable-counting'*[*measurable*]:
  shows (λ *xs. counting' i k xs*) ∈ (*listM* (*count-space* (*UNIV*::*nat set*))) $\rightarrow_M$ (*count-space* (*UNIV*::*nat set*))
  by(*induction k*,*auto*)

lemma *counting'-sum*:
  assumes $k \leq$ *length xs*
  shows *counting' i k xs* = ($\sum j \in \{0..<k\}$. (*if Q i j then* (*xs* ! *j*) *else 0*))
proof−
  have *counting' i k xs* = ($\sum j \in \{0..<k\}$. (*if Q i j then* (*nth-total 0 xs j*) *else 0*))
    by(*induction k*,*auto*)
  also have ... = ($\sum j \in \{0..<k\}$. (*if Q i j then* (*xs* ! *j*) *else 0*))
  proof(*subst sum.cong*)
    show $\{0..<k\} = \{0..<k\}$ **by** *auto*
    show $\bigwedge x.\ x \in \{0..<k\} \Longrightarrow$ (*if Q i x then* (*nth-total 0 xs x*) *else 0*) = (*if Q i x then xs* ! *x else 0*)
      using *assms* **unfolding** *nth-total-def2* **by** *auto*
  qed(*auto*)
  finally show *?thesis* **by** *auto*
qed

lemma *sensitvity-counting′*:
assumes $(xs,ys) \in$ *adj-L1-norm*
and *len*: $k \leq n$
shows $|int\ (counting'\ i\ k\ xs) - int\ (counting'\ i\ k\ ys)| \leq 1$
proof−
from *assms* **have** *xsl*: *length xs* = *n* **and** *ysl*: *length ys* = *n* **and** *le*: *dist-L1-norm-list xs ys* ≤ *1* **and** *i*: *k* ≤ *length xs* **and** *i2*: *k* ≤ *length ys*
by (*auto simp*:*space-L1-norm-list-def adj-L1-norm-def sp-Dataset-def space-restrict-space dist-L1-norm-list-def space-listM*)
from *adjacency-1-int-list*[*OF assms*(*1*)]
consider *xs* = *ys* | (∃ *as a b bs*. *xs* = *as* @ *a* # *bs* ∧ *ys* = *as* @ *b* # *bs* ∧ |*int a* − *int b*| = *1*)
by *auto*
then show *?thesis*
proof(*cases*)
case *1*
then show *?thesis* **by** *auto*
next
case *2*
then obtain *as a b bs*
where *xs*: *xs* = *as* @ *a* # *bs*
and *ys*: *ys* = *as* @ *b* # *bs*
and *a*: |*int a* − *int b*| = *1*
by *blast*
then show *?thesis* **unfolding** *counting′-sum*[*OF i*] *counting′-sum*[*OF i2*]
proof−
have $|int\ (\sum j = 0..<k.\ if\ Q\ i\ j\ then\ xs\ !\ j\ else\ 0) - int\ (\sum j = 0..<k.\ if\ Q\ i\ j\ then\ ys\ !\ j\ else\ 0)|$
$= |(\sum j = 0..<k.\ (if\ Q\ i\ j\ then\ int\ (xs\ !\ j)\ else\ int\ 0)) - (\sum j = 0..<k.\ if\ Q\ i\ j\ then\ int\ (ys\ !\ j)\ else\ int\ 0)|$
unfolding *of-nat-sum if-distrib* **by** *auto*
also have ... $= |(\sum j = 0..<k.\ (if\ Q\ i\ j\ then\ int\ (xs\ !\ j)\ else\ 0)) - (\sum j = 0..<k.\ if\ Q\ i\ j\ then\ int\ (ys\ !\ j)\ else\ 0)|$
unfolding *int-ops*(*1*) **by** *auto*
also have ... $= |(\sum j = 0..<k.\ (if\ Q\ i\ j\ then\ int\ (xs\ !\ j)\ else\ 0) - (if\ Q\ i\ j\ then\ int\ (ys\ !\ j)\ else\ 0))|$
by (*auto simp*: *sum-subtractf*)
also have ... $= |(\sum j = 0..<k.\ (if\ Q\ i\ j\ then\ int\ (xs\ !\ j) - int\ (ys\ !\ j)\ else\ 0))|$
by (*metis* (*full-types*, *opaque-lifting*) *cancel-comm-monoid-add-class.diff-cancel*)
also have ... $\leq (\sum j = 0..<k.\ |(if\ Q\ i\ j\ then\ int\ (xs\ !\ j) - int\ (ys\ !\ j)\ else\ 0)|)$
by(*rule sum-abs*)
also have ... $\leq (\sum j = 0..<k.\ (if\ Q\ i\ j\ then\ |\ int\ (xs\ !\ j) - int\ (ys\ !\ j)|\ else\ 0))$
by(*auto simp*: *if-distrib*)
also have ... $\leq (\sum j = 0..<k.\ |\ int\ (xs\ !\ j) - int\ (ys\ !\ j)|)$

by (*auto simp*: *sum-mono*)
also have ... $\leq$ ($\sum j = 0..<n.$ | *int* (*xs* ! *j*) $-$ *int* (*ys* ! *j*)|)
by(*rule sum-mono2*, *auto simp*:*len*)
also have ... = ($\sum j = 1..n.$ | *int* (*xs* ! (*j* $-$ *1*)) $-$ *int* (*ys* ! (*j* $-$ *1*))|)
proof(*subst sum.reindex-cong*[**where** $l = \lambda\ i.\ i - 1$ **and** $B = \{1..n\}$ **and** $h = \lambda\ x.$ |*int* (*xs* ! (*x* $-$ *1*)) $-$ *int* (*ys* ! (*x* $-$ *1*))|])
show *inj-on* ($\lambda i.\ i - 1$) $\{1..n\}$
unfolding *inj-on-def* **by** *auto*
show $\{0..<n\} = (\lambda i.\ i - 1)$ ' $\{1..n\}$
unfolding *image-def* **by** *force*
qed(*auto*)
also have ... $\leq$ *1*
using *le of-int-le-1-iff* **unfolding** *dist-L1-norm-list-def* **by** *fastforce*
finally show |*int* ($\sum j = 0..<k.$ *if Q i j then xs* ! *j else 0*) $-$ *int* ($\sum j = 0..<k.$ *if Q i j then ys* ! *j else 0*)| $\leq$ *1* **.**
qed
qed
qed

A component of a tuple of counting queries **definition** *counting*::*nat* $\Rightarrow$ *nat list* $\Rightarrow$ *nat* **where**
counting i xs = *counting′ i* (*length xs*) *xs*

lemma *measurable-counting*[*measurable*]:
shows (*counting i*) $\in$ (*listM* (*count-space* (*UNIV*::*nat set*))) $\rightarrow_M$ (*count-space* (*UNIV*::*nat set*))
unfolding *counting-def* **by** *auto*

lemma *counting-sum*:
shows *counting i xs* = ($\sum j \in \{0..<length\ xs\}$. (*if Q i j then* (*xs* ! *j*) *else 0*))
unfolding *counting-def* **by**(*rule counting′-sum*,*auto*)

lemma *sensitvity-counting*:
assumes (*xs*,*ys*) $\in$ *adj-L1-norm*
shows |*int* (*counting k xs*) $-$ *int* (*counting k ys*)| $\leq$ *1*
proof−
from *assms* **have** *xsl*: *length xs* = *n* **and** *ysl*: *length ys* = *n* **and** *le*: *dist-L1-norm-list xs ys* $\leq$ *1*
by (*auto simp*:*space-L1-norm-list-def adj-L1-norm-def sp-Dataset-def space-restrict-space dist-L1-norm-list-def space-listM*)
thus *?thesis* **unfolding** *counting-def* **using** *sensitvity-counting′ assms* **by** *auto*
qed

A list(tuple) of counting queries **definition** *counting-query*::*nat list* $\Rightarrow$ *nat list* **where**

counting-query xs = *map* (λ *k. counting k xs*) [*0..<m*]

lemma *measurable-counting-query*[*measurable*]:
shows *counting-query* $\in$ *listM* (*count-space UNIV*) $\rightarrow_M$ *listM* (*count-space UNIV*)
unfolding *counting-query-def*
proof(*induction m*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc m*)
have (λ*xs. map* (λ*k. counting k xs*) [*0..<Suc m*]) = (λ*xs.* (λ*xs. map* (λ*k. counting k xs*) [*0..<m*]) *xs* @ [*counting m xs*])
by *auto*
also have ... $\in$ *listM* (*count-space UNIV*) $\rightarrow_M$ *listM* (*count-space UNIV*)
using *Suc* **by** *auto*
finally show *?case* **.**
qed

lemma *length-counting-query*:
shows *length*(*counting-query xs*) = *m*
unfolding *counting-query-def* **by** *auto*

lemma *counting-query-nth*:
fixes *k*::*nat* **assumes** *k* < *m*
shows *counting-query xs* ! *k* = *counting k xs*
unfolding *counting-query-def* **by**(*subst nth-map-upt*[*of k m 0* (λ *k. counting k xs*)],*auto simp*: *assms*)

corollary *counting-query-nth′*:
fixes *k*::*nat* **assumes** *k* < *m*
shows *map real* (*counting-query xs*) ! *k* = *real* (*counting k xs*)
unfolding *counting-query-def List.map.compositionality comp-def*
by(*subst nth-map-upt*[*of k m 0* (λ *k. real* (*counting k xs*))],*auto simp*: *assms*)

end

A formalization of the report noisy max of noisy counting

context
Lap-Mechanism-RNM-counting
begin

interpretation *L1-norm-list* (*UNIV*::*nat set*) (λ *x y.* |*int x* − *int y*|) *n*
by(*unfold-locales*,*auto*)

interpretation *results-AFDP n*

by(*unfold-locales*)

definition *RNM-counting* :: *real* ⇒ *nat list* ⇒ *nat measure* **where**
RNM-counting ε *x* = *do* {
y ← *Lap-dist-list* (*1* / ε) (*counting-query x*);
return (*count-space UNIV*) (*argmax-list y*)
}

Naive evaluation of differential privacy of *RNM-counting*
The naive proof is as follows: We first show that the counting query has the sensitivity *m*. *RNM-counting* adds the Laplace noise with scale *1* / ε to each element given by *counting-query*. Thanks to the postprocessing inequality, the final process *argmax-list* does not change the differential privacy. Therefore the differential privacy of *RNM-counting* is $m * \varepsilon$

interpretation *Lap-Mechanism-list listM* (*count-space UNIV*) *counting-query adj-L1-norm m*
unfolding *counting-query-def*
by(*unfold-locales*,*induction m*, *auto simp*: *space-listM*)

lemma *sensitvity-counting-query-part*:
fixes *k*::*nat* **assumes** $k < m$
and (*xs*,*ys*) ∈ *adj-L1-norm*
shows |*map real* (*counting-query xs*) ! *k* − *map real* (*counting-query ys*) ! *k* | ≤ *1*
unfolding *counting-query-nth′*[*OF assms*(*1*)] **using** *sensitvity-counting*[*OF assms*(*2*)]
by (*metis* (*mono-tags*, *opaque-lifting*) *of-int-abs of-int-diff of-int-le-1-iff of-int-of-nat-eq*)

lemma *sensitvity-counting-query*:
shows *sensitivity* ≤ *m*
proof(*unfold sensitivity-def*, *rule Sup-least*, *elim CollectE exE*)
fix *r*::*ereal* **and** *xs ys* ::*nat list* **assume** *r*: *r* = *ereal* ($\sum$ *i* = *1*..*m*. |*map real* (*counting-query xs*) ! (*i* − *1*) − *map real* (*counting-query ys*) ! (*i* − *1*)|) ∧
xs ∈ *space* (*listM* (*count-space UNIV*)) ∧ *ys* ∈ *space* (*listM* (*count-space UNIV*)) ∧ (*xs*, *ys*) ∈ *adj-L1-norm*
hence *r* = *ereal* ($\sum$ *i* = *1*..*m*. |*map real* (*counting-query xs*) ! (*i* − *1*) − *map real* (*counting-query ys*) ! (*i* − *1*)|)
by *auto*
also have ... = *ereal* ($\sum$ *i* = *0*..<*m*. |*map real* (*counting-query xs*) ! *i*− *map real* (*counting-query ys*) ! *i*|)
by(*subst sum.reindex-cong*[**where** *B* = {*0*..<*m*} **and** *l* = *Suc* **and** *h* = λ*i*. |*map real* (*counting-query xs*) ! *i* − *map real* (*counting-query ys*) ! *i*|],*auto*)
also have ... ≤ *ereal* ($\sum$ *i* = *0*..<*m*. (*1*::*real*))
by(*subst ereal-less-eq*(*3*),*subst sum-mono*, *auto intro* !: *sensitvity-counting-query-part*

simp: *r*)
also have ... $\leq$ *ereal m*
by *auto*
finally show $r \leq$ *ereal* (*real m*)**.**
qed

corollary *sensitvity-counting-query-finite*:
shows *sensitivity* $< \infty$
using *sensitvity-counting-query* **by** *auto*

theorem *Naive-differential-privacy-LapMech-RNM-AFDP*:
assumes *pose*: (ε::*real*) $>$ *0*
shows *differential-privacy-AFDP* (*RNM-counting* ε) (*real* ($m * \varepsilon$)) *0*
unfolding *RNM-counting-def*
proof(*rule differential-privacy-postprocessing*[**where** R' = *count-space UNIV* **and** R = *listM borel*])

interpret *Output*: *L1-norm-list UNIV*::*real set* λ *m n*. $|m - n|$ *m*
by(*unfold-locales*,*auto*)

show *0* $\leq$ *real* ($m * \varepsilon$)
using *pose* **by** *auto*
show (λx. *Lap-dist-list* (*1* / *real* ε) (*map real* (*counting-query x*))) $\in$ (*listM* (*count-space UNIV*)) $\rightarrow_M$ *prob-algebra* (*listM borel*)
by *auto*
show (λy. *return* (*count-space UNIV*) (*argmax-list y*)) $\in$ *listM borel* $\rightarrow_M$ *prob-algebra* (*count-space UNIV*)
by *auto*
show *adj*: *adj-L1-norm* $\subseteq$ *space* (*listM* (*count-space UNIV*)) $\times$ *space* (*listM* (*count-space UNIV*))
unfolding *space-listM space-count-space* **by** *auto*
have *differential-privacy* ((*Lap-dist-list* (*1* / *real* ε)) *o* (λx. (*map real* (*counting-query x*))))
adj-L1-norm ($m * \varepsilon$) *0*
proof(*intro differential-privacy-preprocessing*)
show *0* $\leq$ *real* ($m * \varepsilon$)
by *fact*
show (λx. *map real* (*counting-query x*)) $\in$ *listM* (*count-space UNIV*) $\rightarrow_M$ *listM borel*
by *auto*
show $\forall$(*x*, *y*)$\in$*adj-L1-norm*. (*map real* (*counting-query x*), *map real* (*counting-query y*))
$\in$ {(*xs*, *ys*) |*xs ys*. *length xs* = *m* $\wedge$ *length ys* = *m* $\wedge$ *Output.dist-L1-norm-list xs ys* $\leq$ *real m*}
unfolding *adj-L1-norm-def Output.dist-L1-norm-list-def Int-def prod.case case-prod-beta*
proof(*intro ballI*, *elim CollectE conjE exE*)
fix *p xs ys* **assume** *p*:*p* = (*xs*, *ys*) **and** *xs* $\in$ *space sp-Dataset*

and *ys* $\in$ *space sp-Dataset* **and** *adj*:*dist-L1-norm-list xs ys* $\leq$ *1*
hence *length xs* = *n* **and** *length ys* = *n* **and** *c*: (*xs*, *ys*) $\in$ *adj-L1-norm*
by (*auto simp*:*space-L1-norm-list-def adj-L1-norm-def sp-Dataset-def space-restrict-space dist-L1-norm-list-def space-listM*)
have *ereal* ($\sum$ *i* = *1*..*m*. |*map real* (*counting-query xs*) ! (*i* $-$ *1*) $-$ *map real* (*counting-query ys*) ! (*i* $-$ *1*)|) $\leq$ $\bigsqcup$ {*ereal* ($\sum$ *i* = *1*..*m*. |*map real* (*counting-query x*) ! (*i* $-$ *1*) $-$ *map real* (*counting-query y*) ! (*i* $-$ *1*)|) |*x y*. *x* $\in$ *UNIV* $\wedge$ *y* $\in$ *UNIV* $\wedge$ (*x*, *y*) $\in$ *adj-L1-norm*}
by(*auto intro*!: *Sup-upper c*)
also have ... $\leq$ *ereal*(*real m*)
using *sensitvity-counting-query* **unfolding** *sensitivity-def space-listM space-count-space* **by** *auto*
finally show (*map real* (*counting-query* (*fst p*)), *map real* (*counting-query* (*snd p*)))
$\in$ {(*xs*, *ys*) |*xs ys*. *length xs* = *m* $\wedge$ *length ys* = *m* $\wedge$ ($\sum$ *i* = *1*..*m*. |*xs* ! (*i* $-$ *1*) $-$ *ys* ! (*i* $-$ *1*)|) $\leq$ *real m*}
unfolding *p* **using** *length-counting-query* **by** *auto*
qed
show *differential-privacy* (*Lap-dist-list* (*1* / *real* ε)) {(*xs*, *ys*) |*xs ys*. *length xs* = *m* $\wedge$ *length ys* = *m* $\wedge$ *Output*.*dist-L1-norm-list xs ys* $\leq$ *real m*} (*real* (*m* $*$ ε)) *0*
proof(*subst differential-privacy-adj-sym*)
show *sym* {(*xs*, *ys*) |*xs ys*. *length xs* = *m* $\wedge$ *length ys* = *m* $\wedge$ *Output*.*dist-L1-norm-list xs ys* $\leq$ *real m*}
by(*rule symI*,*auto simp*:*Output*.*MetL1*.*commute*)
show $\forall$(*d1*, *d2*)$\in$ {(*xs*, *ys*) |*xs ys*. *length xs* = *m* $\wedge$ *length ys* = *m* $\wedge$ *Output*.*dist-L1-norm-list xs ys* $\leq$ *real m*}.
DP-inequality (*Lap-dist-list* (*1* / *real* ε) *d1*) (*Lap-dist-list* (*1* / *real* ε) *d2*) (*real* (*m* $*$ ε)) *0*
unfolding *Int-def prod*.*case case-prod-beta*
proof(*intro ballI*, *elim CollectE exE conjE*)
fix *p xs ys* **assume** *p*: *p* = (*xs*, *ys*) **and** *xs*: *length xs* = *m* **and** *ys*: *length ys* = *m* **and** *adj*: *Output*.*dist-L1-norm-list xs ys* $\leq$ *real m*
have *0*: (*real m* / (*1* / *real* ε)) = (*real* (*m* $*$ ε))
by *auto*
from *xs ys adj p DP-Lap-dist-list*[*of* (*1* / *real* ε) *xs m ys real m*,*simplified 0*]
show *DP-inequality* (*Lap-dist-list* (*1* / *real* ε) (*fst p*)) (*Lap-dist-list* (*1* / *real* ε) (*snd p*)) (*real* (*m* $*$ ε)) *0*
unfolding *DP-inequality-cong-DP-divergence Output*.*dist-L1-norm-list-def p fst-def snd-def*
by (*metis assms*(*1*) *case-prod-conv ereal-eq-0*(*2*) *of-nat-0-le-iff zero-less-divide-1-iff*)
qed
qed
qed(*auto simp*: *space-listM*)
thus *differential-privacy* (λ*x*. *Lap-dist-list* (*1* / *real* ε) (*map real* (*counting-query x*))) *adj-L1-norm* (*real* (*m* $*$ ε)) *0*

unfolding *comp-def* **by** *auto*
qed

True evaluation of differential privacy of *RNM-counting* in contrast to the naive proof, *counting-query* and *argmax-list* are essential.

lemma *finer-sensitivity-counting-query*:
assumes (*xs*,*ys*) ∈ *adj-L1-norm*
shows *list-all2* (λ *x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) (*counting-query xs*) (*counting-query ys*) ∨ *list-all2* (λ *x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) (*counting-query ys*) (*counting-query xs*)
proof−
note *adjacency-1-int-list*[*OF assms*]
then consider *xs* = *ys* | (∃ *as a b bs*. *xs* = *as* @ *a* # *bs* ∧ *ys* = *as* @ *b* # *bs* ∧ |*int a* − *int b*| = *1*) ∧ *xs* ≠ *ys*
by *auto*
then show *?thesis*
proof(*cases*)
case *1*
define *zs* **where** *zs*: *zs* = (*counting-query ys*)
have ∗: *list-all2* (λ *x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) *zs zs*
by(*induction zs*,*auto*)
thus *?thesis*
unfolding *1 zs* **by** *auto*
next
case *2*
then obtain *as a b bs*
where *xs*:*xs* = *as* @ *a* # *bs*
and *ys*: *ys* = *as* @ *b* # *bs*
and *d*:*a* < *b* ∨ *b* < *a*
and *diff*: |*int a* − *int b*| = *1*
using *linorder-less-linear* **by** *blast*
define *m1*::*nat* **where** *m1*: *m1* = *length as*
{
fix *k* **assume** *k*: *k*∈{*0*..<*m*}
have *1*: ({*0*..<*m1*}∪{*Suc m1*..<*length xs*})∪{*m1*} = {*0*..<*length xs*}
unfolding *m1 xs* **by** *auto*
have *a*: *xs* ! *m1* = *a* **and** *b*: *ys* ! *m1* = *b*
unfolding *m1 xs ys* **by** *auto*

have ∗∗∗: ⋀*j*. *j* ∈ ({*0*..<*m1*}∪{*Suc m1*..<*length xs*}) ⟹ *xs* ! *j* = *ys* ! *j*
proof−
fix *j* **assume** *j* ∈ ({*0*..<*m1*}∪{*Suc m1*..<*length xs*})
hence *2*: *j* ∈ {*0*..<*length xs*} **and** *3*: *j* ≠ *m1*
using *1* **by** *auto*
have ⋀*j*. *j* ∈ {*0*..<*length xs*} ∧ *j* ≠ *m1* ⟹ *xs* ! *j* = *ys* ! *j*

using *m1* **unfolding** *xs ys* **by** (*simp add*: *nth-Cons′ nth-append*)
thus *xs* ! *j* = *ys* ! *j* **using** *2 3* **by** *auto*
qed

have ($\sum$*j* = *0*..<*length xs*. *if Q k j then xs* ! *j else 0*) = ($\sum$*j*∈({*0*..<*m1*}∪{*Suc m1*..<*length xs*})∪{*m1*}. *if Q k j then xs* ! *j else 0*)
using *1* **by** *auto*
also have ... = ($\sum$*j*∈{*0*..<*m1*}∪{*Suc m1*..<*length xs*}. *if Q k j then xs* ! *j else 0*) + (*if Q k m1 then a else 0*)
by(*subst sum-Un-nat*,*auto simp*: *a*)
finally have *A*: ($\sum$*j* = *0*..<*length xs*. *if Q k j then xs* ! *j else 0*) = ($\sum$*j*∈{*0*..<*m1*}∪{*Suc m1*..<*length xs*}. *if Q k j then xs* ! *j else 0*) + (*if Q k m1 then a else 0*) **.**

have ($\sum$*j* = *0*..<*length ys*. *if Q k j then ys* ! *j else 0*) = ($\sum$*j*∈({*0*..<*m1*}∪{*Suc m1*..<*length xs*})∪{*m1*}. *if Q k j then ys* ! *j else 0*)
using *1 assms* **by** (*auto simp*:*space-L1-norm-list-def adj-L1-norm-def sp-Dataset-def space-restrict-space dist-L1-norm-list-def space-listM*)
also have ... = ($\sum$*j*∈{*0*..<*m1*}∪{*Suc m1*..<*length xs*}. *if Q k j then ys* ! *j else 0*) + (*if Q k m1 then b else 0*)
by(*subst sum-Un-nat*,*auto simp*: *b*)
also have ... = ($\sum$*j*∈{*0*..<*m1*}∪{*Suc m1*..<*length xs*}. *if Q k j then xs* ! *j else 0*) + (*if Q k m1 then b else 0*)
using ∗∗∗ **by**(*subst sum.cong*[**where** *B* = {*0*..<*m1*} ∪ {*Suc m1*..<*length xs*} **and** *h* = λ *x*. (*if Q k x then xs* ! *x else 0*)],*auto*)
finally have *B*: ($\sum$*j* = *0*..<*length ys*. *if Q k j then ys* ! *j else 0*) = ($\sum$*j*∈{*0*..<*m1*} ∪ {*Suc m1*..<*length xs*}. *if Q k j then xs* ! *j else 0*) + (*if Q k m1 then b else 0*) **.**

note *A B*
}note ∗∗∗ = *this*

from *d diff* **consider**
b = *a* + *1* | *a* = *b* + *1*
by *auto*
thus *?thesis* **proof**(*cases*)
case *1*
have *list-all2* (λ *x y*. *x* ≥ *y* ∧ *x* ≤ *y* + *1*) (*counting-query ys*) (*counting-query xs*)
proof(*subst list-all2-conv-all-nth*,*rule conjI*)
show *length* (*counting-query ys*) = *length* (*counting-query xs*)
using *length-counting-query* **by** *auto*
show ∀ *i*<*length* (*counting-query ys*). *counting-query xs* ! *i* ≤ *counting-query ys* ! *i* ∧ *counting-query ys* ! *i* ≤ *counting-query xs* ! *i* + *1*
proof(*intro allI impI*)
fix *i* **assume** *i* < *length* (*counting-query ys*)

hence *i*: $i < length\ [0..<m]$ **and** *i2*: $i \in \{0..<m\}$ **and** *i3*: $[0..<m]\ !\ i = i$
unfolding *length-counting-query* **by** *auto*
thus *counting-query xs* ! $i \leq$ *counting-query ys* ! $i \wedge$ *counting-query ys* ! $i \leq$ *counting-query xs* ! $i + 1$
unfolding *counting-query-def counting-sum list-all2-conv-all-nth length-map length-upt*
nth-map[*of i* [*0*..<*m*] $\lambda k. \sum j = 0..<length\ xs.\ if\ Q\ k\ j\ then\ xs\ !\ j\ else\ 0$,*OF i*]
nth-map[*of i* [*0*..<*m*] $\lambda k. \sum j = 0..<length\ ys.\ if\ Q\ k\ j\ then\ ys\ !\ j\ else\ 0$,*OF i*]
1 i3 ∗∗∗(*1*)[*OF i2*] ∗∗∗(*2*)[*OF i2*]
by *auto*
qed
qed
then show *?thesis* **by** *auto*
next
case *2*
have *list-all2* (λ *x y*. $x \geq y \wedge x \leq y + 1$) (*counting-query xs*) (*counting-query ys*)
proof(*subst list-all2-conv-all-nth*,*rule conjI*)
show *length* (*counting-query xs*) = *length* (*counting-query ys*)
using *length-counting-query* **by** *auto*
show $\forall$ *i*<*length* (*counting-query xs*). *counting-query ys* ! $i \leq$ *counting-query xs* ! $i \wedge$ *counting-query xs* ! $i \leq$ *counting-query ys* ! $i + 1$
proof(*intro allI impI*)
fix *i* **assume** $i < length$ (*counting-query xs*)
hence *i*: $i < length\ [0..<m]$ **and** *i2*: $i \in \{0..<m\}$ **and** *i3*: $[0..<m]\ !\ i = i$
unfolding *length-counting-query* **by** *auto*
thus *counting-query ys* ! $i \leq$ *counting-query xs* ! $i \wedge$ *counting-query xs* ! $i \leq$ *counting-query ys* ! $i + 1$
unfolding *counting-query-def counting-sum list-all2-conv-all-nth length-map length-upt*
nth-map[*of i* [*0*..<*m*] $\lambda k. \sum j = 0..<length\ xs.\ if\ Q\ k\ j\ then\ xs\ !\ j\ else\ 0$,*OF i*]
nth-map[*of i* [*0*..<*m*] $\lambda k. \sum j = 0..<length\ ys.\ if\ Q\ k\ j\ then\ ys\ !\ j\ else\ 0$,*OF i*]
2 i3 ∗∗∗(*1*)[*OF i2*] ∗∗∗(*2*)[*OF i2*]
by *auto*
qed
qed
then show *?thesis* **by** *auto*
qed
qed
qed

lemma *list-all2-map-real-adj*:

assumes *list-all2* $(\lambda x\ y.\ y \leq x \wedge x \leq y + 1)$ *xs ys*
shows *list-all2* $(\lambda x\ y.\ y \leq x \wedge x \leq y + 1)$ (*map real xs*) (*map real ys*)
using *assms nth-map*[*of - xs real*] *nth-map*[*of - ys real*] **unfolding** *list-all2-conv-all-nth length-map*
by *force*

lemma *finer-sensitivity-counting-query'*:
assumes (*xs*,*ys*) $\in$ *adj-L1-norm*
shows *list-all2* $(\lambda\ x\ y.\ x \geq y \wedge x \leq y + 1)$ (*map real* (*counting-query xs*))(*map real* (*counting-query ys*)) $\vee$ *list-all2* $(\lambda\ x\ y.\ x \geq y \wedge x \leq y + 1)$ (*map real* (*counting-query ys*))(*map real* (*counting-query xs*))
using *finer-sensitivity-counting-query*[*OF assms*] *list-all2-map-real-adj*
by *auto*

interpretation *Lap-Mechanism-RNM-mainpart* (*listM* (*count-space UNIV*)) *adj-L1-norm counting-query*
proof(*unfold-locales*)
show $(\lambda x.$ *map real* (*counting-query x*)$)$ $\in$ (*listM* (*count-space UNIV*)) $\rightarrow_M$ *listM borel*
by *auto*
show $\forall (x,\ y)\in$ *adj-L1-norm*.
list-all2 $(\lambda x\ y.\ y \leq x \wedge x \leq y + 1)$ (*map real* (*counting-query x*)) (*map real* (*counting-query y*)) $\vee$
list-all2 $(\lambda x\ y.\ y \leq x \wedge x \leq y + 1)$ (*map real* (*counting-query y*)) (*map real* (*counting-query x*))
using *finer-sensitivity-counting-query'* **by** *auto*
show *adj-L1-norm* $\subseteq$ *space* (*listM* (*count-space UNIV*)) $\times$ *space* (*listM* (*count-space UNIV*))
unfolding *space-listM space-count-space* **by** *auto*
qed

theorem *differential-privacy-LapMech-RNM-AFDP*:
assumes *pose*: $(\varepsilon::real) > 0$
shows *differential-privacy-AFDP* (*RNM-counting* ε) ε *0*
using *differential-privacy-LapMech-RNM*[*OF assms*]
unfolding *RNM-counting-def LapMech-RNM-def* **by** *auto*

end

end

References

[1] B. Balle, G. Barthe, M. Gaboardi, J. Hsu, and T. Sato. Hypothesis testing interpretations and renyi differential privacy. In S. Chi-

appa and R. Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (AISTATS 2020)*, volume 108 of *Proceedings of Machine Learning Research*, pages 2496–2506, Online, 26–28 Aug 2020. PMLR.

[2] G. Barthe and F. Olmedo. Beyond differential privacy: Composition theorems and relational logic for f-divergences between probabilistic programs. In *Automata, Languages, and Programming: 40th International Colloquium, ICALP 2013, Proceedings, Part II*, pages 49–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[3] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3-4):211–407, 2013.

[4] S. P. Kasiviswanathan and A. Smith. A note on differential privacy: Defining resistance to arbitrary side information. *Journal of Privacy and Confidentiality*, 6(1), 2014.

[5] T. Sato and S. Katsumata. Divergences on monads for relational program logics. *Mathematical Structures in Computer Science*, 33(45):427485, 2023.