Negatively Associated Random Variables

Emin Karayel

January 20, 2025

Abstract

Negative Association is a generalization of independence for random variables, that retains some of the key properties of independent random variables. In particular closure properties, such as composition with monotone functions, as well as, the well-known Chernoff-Hoeffding bounds.

This entry introduces the concept and verifies the most important closure properties, as well as, the concentration inequalities. It also verifies the FKG inequality, which is a generalization of Chebyshev's sum inequality for distributive lattices and a key tool for establishing negative association, but has also many applications beyond the context of negative association, in particular, statistical physics and graph theory.

As an example, permutation distributions are shown to be negatively associated, from which many more sets of negatively random variables can be derived, such as, e.g., n-subsets, or the the balls-into-bins process.

Finally, the entry derives a correct false-positive rate for Bloom filters using the library.

Contents

1 Preliminary Definitions and Lemmas

theory *Negative-Association-Util*
  imports
    Concentration-Inequalities.Concentration-Inequalities-Preliminary
    Universal-Hash-Families.Universal-Hash-Families-More-Product-PMF
begin

abbreviation (*input*) *flip* :: ‹($'a \Rightarrow 'b \Rightarrow 'c$) $\Rightarrow 'b \Rightarrow 'a \Rightarrow 'c$› **where**
  ‹*flip f x y* $\equiv$ *f y x*›

Additional introduction rules for boundedness:

lemma *bounded-const-min*:
  fixes f :: $'a \Rightarrow$ *real*
  assumes *bdd-below* (f ‘ M)
  shows *bounded* (($\lambda x.$ *min c* (f x)) ‘ M)
proof –
 obtain h **where** $\bigwedge x.\ x \in M \Longrightarrow f\ x \geq h$ **using** *assms*(*1*) **unfolding** *bdd-below-def*
by *auto*
  thus *?thesis* **by** (*intro boundedI*[**where** B=*max* $|c|$ $|-h|$]) *force*
qed

lemma *bounded-prod*:
  fixes f :: $'i \Rightarrow 'a \Rightarrow$ *real*
  assumes *finite I*
  assumes $\bigwedge i.\ i \in I \Longrightarrow$ *bounded* (f i ‘ T)
  shows *bounded* (($\lambda x.$ ($\prod$ $i \in I.$ f i x)) ‘ T)
  using *assms* **by** (*induction I*) (*auto intro*:*bounded-mult-comp bounded-const*)

lemma *bounded-vec-mult-comp*:
  fixes f g :: $'a \Rightarrow$ *real*
  assumes *bounded* (f ‘ T) *bounded* (g ‘ T)
  shows *bounded* (($\lambda x.$ (f x) $*_R$ (g x)) ‘ T)
  using *bounded-mult-comp*[*OF assms*] **by** *simp*

lemma *bounded-max*:
  fixes f :: $'a \Rightarrow$ *real*
  assumes *bounded* (($\lambda x.$ f x) ‘ T)
  shows *bounded* (($\lambda x.$ *max c* (f x)) ‘ T)
proof –
  obtain m **where** *norm* (f x) $\leq m$ **if** $x \in T$ **for** x
    using *assms* **unfolding** *bounded-iff* **by** *auto*

  thus *?thesis* **by** (*intro boundedI*[**where** B=*max m c*]) *fastforce*
qed

lemma *bounded-of-bool*: *bounded* (*range of-bool*) **by** *auto*

lemma *bounded-range-imp*:
  assumes *bounded* (*range f*)
  shows *bounded* (($\lambda\omega$. *f* (*h* ω)) ‘ *S*)
  by (*intro bounded-subset*[*OF assms*]) *auto*

The following allows to state integrability and conditions about the integral simultaneously, e.g. *has-int-that M f* ($\lambda x.\ x \le c$) says f is integrable on M and the integral smaller or equal to c.

definition *has-int-that* **where**
  has-int-that M f P = (*integrable M f* $\wedge$ (*P* ($\int \omega.\ f\ \omega\ \partial M$)))

lemma *true-eq-iff*: $P \Longrightarrow True = P$ **by** *auto*
lemma *le-trans*: $y \le z \Longrightarrow x \le y \longrightarrow x \le (z :: 'a :: order)$ **by** *auto*

lemma *has-int-that-mono*:
  assumes $\bigwedge x.\ P\ x \longrightarrow Q\ x$
  shows *has-int-that M f P* $\le$ *has-int-that M f Q*
  using *assms* **unfolding** *has-int-that-def* **by** *auto*

lemma *has-int-thatD*:
  assumes *has-int-that M f P*
  shows *integrable M f P* (*integral*^{L} *M f*)
  using *assms has-int-that-def* **by** *auto*

This is useful to specify which components a functional depends on.

definition *depends-on* :: $(('a \Rightarrow 'b) \Rightarrow 'c) \Rightarrow 'a\ set \Rightarrow bool$
  where *depends-on f I* = ($\forall$ *x y*. *restrict x I* = *restrict y I* $\longrightarrow$ *f x* = *f y*)

lemma *depends-onI*:
  assumes $\bigwedge x.$ *f x* = *f* (λi. *if* $i \in I$ *then* (*x i*) *else undefined*)
  shows *depends-on f I*
proof –
  have *f x* = *f y* **if** *restrict x I* = *restrict y I* **for** *x y*
  proof –
    have *f x* = *f* (*restrict x I*) **using** *assms* **unfolding** *restrict-def* **by** *simp*
    also have ... = *f* (*restrict y I*) **using** *that* **by** *simp*
    also have ... = *f y* **using** *assms* **unfolding** *restrict-def* **by** *simp*
    finally show *?thesis* **by** *simp*
  qed
  thus *?thesis* **unfolding** *depends-on-def* **by** *blast*
qed

lemma *depends-on-comp*:
  assumes *depends-on f I*
  shows *depends-on* ($g \circ f$) *I*
  using *assms* **unfolding** *depends-on-def* **by** (*metis o-apply*)

lemma *depends-on-comp-2*:
  assumes *depends-on f I*

shows *depends-on* ($\lambda x.\ g\ (f\ x)$) *I*
using *assms* **unfolding** *depends-on-def* **by** *metis*

lemma *depends-onD*:
assumes *depends-on f I*
shows $f\ \omega = f\ (\lambda i{\in}I.\ (\omega\ i))$
using *assms* **unfolding** *depends-on-def* **by** (*metis extensional-restrict restrict-extensional*)

lemma *depends-onD2*:
assumes *depends-on f I restrict x I = restrict y I*
shows *f x = f y*
using *assms* **unfolding** *depends-on-def* **by** *metis*

lemma *depends-on-empty*:
assumes *depends-on f* {}
shows $f\ \omega = f$ *undefined*
by (*intro depends-onD2*[*OF assms*]) *auto*

lemma *depends-on-mono*:
assumes $I \subseteq J$ *depends-on f I*
shows *depends-on f J*
using *assms* **unfolding** *depends-on-def* **by** (*metis restrict-restrict Int-absorb1*)

abbreviation *square-integrable M f* $\equiv$ *integrable M* ((*power2* :: *real* $\Rightarrow$ *real*) $\circ$ *f*)

There are many results in the field of negative association, where a statement is true for simultaneously monotone or anti-monotone functions. With the below construction, we introduce a mechanism where we can parameterize on the direction of a relation:

datatype *RelDirection* = *Fwd* | *Rev*

definition *dir-le* :: *RelDirection* $\Rightarrow$ (('*d*::*order*) $\Rightarrow$ ('*d* :: *order*) $\Rightarrow$ *bool*) (**infixl** $\leq\geq_1$ *60*)
where *dir-le* η = (*if* η = *Fwd then* ($\leq$) *else* ($\geq$))

lemma *dir-le*[*simp*]:
$(\leq\geq_{Fwd}) = (\leq)$
$(\leq\geq_{Rev}) = (\geq)$
by (*auto simp*:*dir-le-def*)

definition *dir-sign* :: *RelDirection* $\Rightarrow$ '*a*::{*one*,*uminus*} ($\pm_1$)
where *dir-sign* η = (*if* η = *Fwd then 1 else* (−*1*))

lemma *dir-le-refl*: $x \leq\geq_\eta x$
by (*cases* η) *auto*

lemma *dir-sign*[*simp*]:
$(\pm_{Fwd}) = (1)$
$(\pm_{Rev}) = (-1)$

by (*auto simp*:*dir-sign-def*)

lemma *conv-rel-to-sign*:
fixes f :: $'a$::*order* $\Rightarrow$ *real*
shows *monotone* ($\leq$) ($\leq\geq_\eta$) f = *mono* (($*$)($\pm_\eta$) $\circ$ f)
by (*cases* η) (*simp-all add*:*monotone-def*)

instantiation *RelDirection* :: *times*
begin
definition *times-RelDirection* :: *RelDirection* $\Rightarrow$ *RelDirection* $\Rightarrow$ *RelDirection* **where**
times-RelDirection-def: *times-RelDirection* x y = (*if* x = y *then Fwd else Rev*)

instance by *standard*
end

lemmas *rel-dir-mult*[*simp*] = *times-RelDirection-def*

lemma *dir-mult-hom*: ($\pm_{\sigma\ *\ \tau}$) = ($\pm_\sigma$) $*$ (($\pm_\tau$)::*real*)
unfolding *dir-sign-def times-RelDirection-def* **by** (*cases* σ,*auto intro*:*RelDirection.exhaust*)

Additional lemmas about clamp for the specific case on reals.

lemma *clamp-eqI2*:
assumes $x \in \{a..b$::*real*$\}$
shows x = *clamp* a b x
using *assms* **unfolding** *clamp-def* **by** *simp*

lemma *clamp-eqI*:
assumes $|x| \leq$ (a::*real*)
shows x = *clamp* ($-a$) a x
using *assms* **by** (*intro clamp-eqI2*) *auto*

lemma *clamp-real-def*:
fixes x :: *real*
shows *clamp* a b x = *max* a (*min* x b)
proof −
consider (*i*) $x < a$ | (*ii*) $x \geq a$ $x \leq b$ | (*iii*) $x > b$ **by** *linarith*
thus *?thesis* **unfolding** *clamp-def* **by** (*cases*) *auto*
qed

lemma *clamp-range*:
assumes $a \leq b$
shows $\bigwedge x$. *clamp* a b $x \geq a$ $\bigwedge x$. *clamp* a b $x \leq b$ *range* (*clamp* a b) $\subseteq \{a..b$::*real*$\}$
using *assms* **by** (*auto simp*: *clamp-real-def*)

lemma *clamp-abs-le*:
assumes $a \geq$ (*0*::*real*)
shows $|$*clamp* ($-a$) a $x| \leq |x|$
using *assms* **unfolding** *clamp-real-def* **by** *simp*

lemma *bounded-clamp*:
  fixes *a b* :: *real*
  shows *bounded* ((*clamp a b* ∘ *f*) ‘ *S*)
proof (*cases a* ≤ *b*)
  case *True*
  show *?thesis* **using** *clamp-range*[*OF True*] *bounded-closed-interval bounded-subset*
    by (*metis image-comp image-mono subset-UNIV*)
next
  case *False*
  hence *clamp a b* (*f x*) = *a* **for** *x* **unfolding** *clamp-def* **by** (*simp add*: *max-def*)
  hence (*clamp a b* ∘ *f*) ‘ *S* ⊆ {*a..a*} **by** *auto*
  thus *?thesis* **using** *bounded-subset bounded-closed-interval* **by** *metis*
qed

lemma *bounded-clamp-alt*:
  fixes *a b* :: *real*
  shows *bounded* ((λ*x*. *clamp a b* (*f x*)) ‘ *S*)
  using *bounded-clamp* **by** (*auto simp*:*comp-def*)

lemma *clamp-borel*[*measurable*]:
  fixes *a b* :: ′*a*::{*euclidean-space*,*second-countable-topology*}
  shows *clamp a b* ∈ *borel-measurable borel*
  unfolding *clamp-def* **by** *measurable*

lemma *monotone-clamp*:
  assumes *monotone* (≤) (≤≥$_\eta$) *f*
  shows *monotone* (≤) (≤≥$_\eta$) (λω. *clamp a* (*b*::*real*) (*f* ω))
  using *assms* **unfolding** *monotone-def clamp-real-def* **by** (*cases* η) *force+*

This part introduces the term *KL-div* as the Kullback-Leibler divergence between a pair of Bernoulli random variables. The expression is useful to express some of the Chernoff bounds more concisely [12, Th. 1].

lemma *radon-nikodym-pmf*:
  assumes *set-pmf p* ⊆ *set-pmf q*
  defines *f* ≡ (λ*x*. *ennreal* (*pmf p x* / *pmf q x*))
  shows
    AE x in measure-pmf q. *RN-deriv q p x* = *f x* (**is** *?R1*)
    AE x in measure-pmf p. *RN-deriv q p x* = *f x* (**is** *?R2*)
proof −
  have *pmf p x* = *0* **if** *pmf q x* = *0* **for** *x*
    using *assms*(*1*) *that* **by** (*meson pmf-eq-0-set-pmf subset-iff*)
  hence *a*:(*pmf q x* ∗ (*pmf p x* / *pmf q x*)) = *pmf p x* **for** *x* **by** *simp*
  have *emeasure* (*density q f*) *A* = *emeasure p A* (**is** *?L* = *?R*) **for** *A*
  proof −
    have *?L* = *set-nn-integral* (*measure-pmf q*) *A f*
      by (*subst emeasure-density*) *auto*
    also have ... = (∫$^+$ *x*∈*A*. *ennreal* (*pmf q x*) ∗ *f x* ∂*count-space UNIV*)
      by (*simp add*: *ac-simps nn-integral-measure-pmf*)
    also have ... = (∫$^+$*x*∈*A*. *ennreal* (*pmf p x*) ∂*count-space UNIV*)

    using *a* **unfolding** *f-def* **by** (*subst ennreal-mult'*[*symmetric*]) *simp-all*
  also have ... = *emeasure* (*bind-pmf p return-pmf*) *A*
    unfolding *emeasure-bind-pmf nn-integral-measure-pmf* **by** *simp*
  also have ... = *?R* **by** *simp*
  finally show *?thesis* **by** *simp*
qed
hence *density* (*measure-pmf q*) *f* = *measure-pmf p* **by** (*intro measure-eqI*) *auto*
hence *AE x in measure-pmf q. f x* = *RN-deriv q p x* **by** (*intro measure-pmf.RN-deriv-unique*) *simp*
thus *?R1* **unfolding** *AE-measure-pmf-iff* **by** *auto*
thus *?R2* **using** *assms* **unfolding** *AE-measure-pmf-iff* **by** *auto*
qed

lemma *KL-divergence-pmf*:
  assumes *set-pmf q* ⊆ *set-pmf p*
  shows *KL-divergence b* (*measure-pmf p*) (*measure-pmf q*) = (∫ *x. log b* (*pmf q x* / *pmf p x*) ∂*q*)
  unfolding *KL-divergence-def entropy-density-def*
  by (*intro integral-cong-AE AE-mp*[*OF radon-nikodym-pmf*(*2*)[*OF assms*(*1*)] *AE-I2*]) *auto*

definition *KL-div* :: *real* ⇒ *real* ⇒ *real* **where**
  KL-div p q = *KL-divergence* (*exp 1*) (*bernoulli-pmf q*) (*bernoulli-pmf p*)

lemma *KL-div-eq*:
  assumes *q* ∈ {*0*<..<*1*} *p* ∈ {*0*..*1*}
  shows *KL-div p q* = *p* ∗ *ln* (*p*/*q*) + (*1*−*p*) ∗ *ln* ((*1*−*p*)/(*1*−*q*)) (**is** *?L* = *?R*)
proof −
  have *set-pmf* (*bernoulli-pmf p*) ⊆ *set-pmf* (*bernoulli-pmf q*)
    using *assms*(*1*) *set-pmf-bernoulli* **by** *auto*
  hence *?L* = (∫ *x. ln* (*pmf* (*bernoulli-pmf p*) *x* / *pmf* (*bernoulli-pmf q*) *x*) ∂*bernoulli-pmf p*)
    unfolding *KL-div-def* **by** (*subst KL-divergence-pmf*) (*simp-all add:log-ln*[*symmetric*])
  also have ... = *?R*
    using *assms*(*1*,*2*) **by** (*subst integral-bernoulli-pmf*) *auto*
  finally show *?thesis* **by** *simp*
qed

lemma *KL-div-swap*:
  assumes *q* ∈ {*0*<..<*1*} *p* ∈ {*0*..*1*}
  shows *KL-div p q* = *KL-div* (*1*−*p*) (*1*−*q*)
  using *assms* **by** (*subst* (*1 2*) *KL-div-eq*) *auto*

A few results about independent random variables:

lemma (**in** *prob-space*) *indep-vars-const*:
  assumes ⋀*i. i* ∈ *I* ⟹ *c i* ∈ *space* (*N i*)
  shows *indep-vars N* (λ*i -. c i*) *I*
proof −
  have *rv*: *random-variable* (*N i*) (λ-. *c i*) **if** *i* ∈ *I* **for** *i* **using** *assms*[*OF that*]

```
by simp
  have b:indep-sets (λi. {space M, {}}) I
  proof (intro indep-setsI, goal-cases)
    case (1 i) thus ?case by simp
  next
    case (2 A J)
    show ?case
    proof (cases ∀ j ∈ J. A j = space M)
      case True thus ?thesis using 2(1) by (simp add:prob-space)
    next
      case False
      then obtain i where i:A i = {} i ∈ J using 2 by auto
      hence prob (⋂ (A ‘ J)) = prob {} by (intro arg-cong[where f=prob]) auto
      also have ... = 0 by simp
      also have ... = (∏j∈J. prob (A j))
        using i by (intro prod-zero[symmetric] 2 bexI[where x=i]) auto
      finally show ?thesis by simp
    qed
  qed
  have {(λ-. c i) −‘ A ∩ space M |A. A ∈ sets (N i)} = {space M, {}} (is ?L =
?R) if i ∈ I for i
  proof
    show ?L ⊆ ?R by auto
  next
    have (λA. (λ-. c i) −‘ A ∩ space M) {} = {} {} ∈ N i by auto
    hence {} ∈ ?L unfolding image-Collect[symmetric] by blast
    moreover have (λA. (λ-. c i) −‘ A ∩ space M) (space (N i)) = space M space
(N i) ∈ N i
      using assms[OF that] by auto
    hence space M ∈ ?L unfolding image-Collect[symmetric] by blast
    ultimately show ?R ⊆ ?L by simp
  qed
  hence indep-sets (λi. {(λ-. c i) −‘ A ∩ space M |A. A ∈ sets (N i)}) I
    using iffD2[OF indep-sets-cong b] b by simp
  thus ?thesis unfolding indep-vars-def2 by (intro conjI rv ballI)
qed

lemma indep-vars-map-pmf:
  assumes prob-space.indep-vars (measure-pmf p) (λ-. discrete) (λi. X i ∘ f) I
  shows  prob-space.indep-vars (map-pmf f p) (λ-. discrete) X I
  using assms unfolding map-pmf-rep-eq by (intro measure-pmf.indep-vars-distr)
auto

lemma indep-var-pair-pmf:
  fixes x y :: 'a pmf
  shows prob-space.indep-var (pair-pmf x y) discrete fst discrete snd
proof −
  have split-bool-univ: UNIV = insert True {False} by auto
```

have *pair-prod*: *pair-pmf x y* = *map-pmf* (λω. (ω *True*, ω *False*)) (*prod-pmf UNIV* (*case-bool x y*))
  unfolding *split-bool-univ* **by** (*subst Pi-pmf-insert*)
    (*simp-all add*:*map-pmf-comp Pi-pmf-singleton pair-map-pmf2 case-prod-beta*)

have *case-bool-eq*: *case-bool discrete discrete* = (λ-. *discrete*)
  by (*intro ext*) (*simp add*: *bool.case-eq-if*)

have *prob-space.indep-vars* (*prod-pmf UNIV* (*case-bool x y*)) (λ-. *discrete*) (λ*i* ω. ω *i*) *UNIV*
  by (*intro indep-vars-Pi-pmf*) *auto*
moreover have (λ*i*. (*case-bool fst snd i*) ∘ (λω. ((ω *True*)::'*a*, ω *False*))) = (λ*i* ω. ω *i*)
  by (*auto intro*!:*ext split*:*bool.splits*)
ultimately show *?thesis*
  unfolding *prob-space.indep-var-def*[*OF prob-space-measure-pmf*] *pair-prod case-bool-eq*
  by (*intro indep-vars-map-pmf*) *simp*
qed

lemma *measure-pair-pmf*: *measure* (*pair-pmf p q*) (*A* × *B*) = *measure p A* ∗ *measure q B* (**is** *?L* = *?R*)
proof −
  have *?L* = *measure* (*pair-pmf p q*) ((*A* ∩ *set-pmf p*) × (*B* ∩ *set-pmf q*))
    by (*intro measure-eq-AE AE-pmfI*) *auto*
  also have ... = *measure p* (*A* ∩ *set-pmf p*) ∗ *measure q* (*B* ∩ *set-pmf q*)
    by (*intro measure-pmf-prob-product*) *auto*
  also have ... = *?R* **by** (*intro arg-cong2*[**where** *f*=(∗)] *measure-eq-AE AE-pmfI*) *auto*
  finally show *?thesis* **by** *simp*
qed

instance *bool* :: *second-countable-topology*
proof
  show ∃ *B*::*bool set set*. *countable B* ∧ *open* = *generate-topology B*
    by (*intro exI*[*of* - *range lessThan* ∪ *range greaterThan*]) (*auto simp*: *open-bool-def*)
qed

end

2 Definition

This section introduces the concept of negatively associated random variables (RVs). The definition follows, as closely as possible, the original description by Joag-Dev and Proschan [13].

However, the following modifications have been made:

Singleton and empty sets of random variables are considered negatively associated. This is useful because it simplifies many of the induction proofs. The second modification is that the RV's don't have to be real valued. Instead the range can be into any linearly ordered space with the borel σ-algebra. This is a major enhancement compared to the original work, as well as results by following authors [6, 7, 8, 14, 17].

```
theory Negative-Association-Definition
  imports
    Concentration-Inequalities.Bienaymes-Identity
    Negative-Association-Util
begin

context prob-space
begin

definition neg-assoc :: ('i ⇒ 'a ⇒ 'c :: {linorder-topology}) ⇒ 'i set ⇒ bool
  where neg-assoc X I = (
    (∀ i ∈ I. random-variable borel (X i)) ∧
    (∀ (f::nat ⇒ ('i ⇒ 'c) ⇒ real) J. J ⊆ I ∧
      (∀ ι<2. bounded (range (f ι)) ∧ mono(f ι) ∧ depends-on (f ι) ([J,I−J]!ι) ∧
      f ι ∈ PiM ([J,I−J]!ι) (λ-. borel) →M borel) ⟶
      covariance (f 0 ∘ flip X) (f 1 ∘ flip X) ≤ 0))

lemma neg-assocI:
  assumes ⋀i. i ∈ I ⟹ random-variable borel (X i)
  assumes ⋀f g J. J ⊆ I
    ⟹ depends-on f J ⟹ depends-on g (I−J)
    ⟹ mono f ⟹ mono g
    ⟹ bounded (range f::real set) ⟹ bounded (range g)
    ⟹ f ∈ PiM J (λ-. borel) →M borel ⟹ g ∈ PiM (I−J) (λ-. borel) →M borel
    ⟹ covariance (f ∘ flip X) (g ∘ flip X) ≤ 0
  shows neg-assoc X I
 using assms unfolding neg-assoc-def by (auto simp:numeral-eq-Suc All-less-Suc)

lemma neg-assocI2:
  assumes [measurable]: ⋀i. i ∈ I ⟹ random-variable borel (X i)
  assumes ⋀f g J. J ⊆ I
    ⟹ depends-on f J ⟹ depends-on g (I−J)
    ⟹ mono f ⟹ mono g
    ⟹ bounded (range f) ⟹ bounded (range g)
    ⟹ f ∈ PiM J (λ-. borel) →M (borel :: real measure)
    ⟹ g ∈ PiM (I−J) (λ-. borel) →M (borel :: real measure)
    ⟹ (∫ω. f(λi. X i ω) ∗ g(λi. X i ω) ∂M)≤(∫ω. f(λi. X i ω)∂M)∗(∫ω. g(λi.
X i ω) ∂M)
  shows neg-assoc X I
proof (rule neg-assocI,goal-cases)
  case (1 i) thus ?case using assms(1) by auto
next
  case (2 f g J)

  note [measurable] = 2(8,9)
```

```
  note bounded = integrable-bounded bounded-intros

  have [measurable]: random-variable borel (λω. f (λi. X i ω))
    using subsetD[OF 2(1)] by (subst depends-onD[OF 2(2)]) measurable
  moreover have [measurable]: random-variable borel (λω. g (λi. X i ω))
    by (subst depends-onD[OF 2(3)]) measurable
  moreover have integrable M (λω. ((f ∘ (λx y. X y x)) ω)^2)
    unfolding comp-def by (intro bounded bounded-subset[OF 2(6)]) auto
  moreover have integrable M (λω. ((g ∘ (λx y. X y x)) ω)^2)
    unfolding comp-def by (intro bounded bounded-subset[OF 2(7)]) auto
  ultimately show ?case using assms(2)[OF 2(1−9)]
    by (subst covariance-eq) (auto simp:comp-def)
qed

lemma neg-assoc-empty:
  neg-assoc X {}
proof (intro neg-assocI2, goal-cases)
  case (1 i)
  then show ?case by simp
next
  case (2 f g J)
  define fc gc where fc:fc = f undefined and gc:gc = g undefined

  have depends-on f {} depends-on g {} using 2 by auto
 hence fg-simps: f = (λx. fc) g = (λx. gc) unfolding fc gc using depends-on-empty
by auto
  then show ?case unfolding fg-simps by (simp add:prob-space)
qed

lemma neg-assoc-singleton:
  assumes random-variable borel (X i)
  shows neg-assoc X {i}
proof (rule neg-assocI2, goal-cases)
  case (1 i)
  then show ?case using assms by auto
next
  case (2 f g J)
  show ?case
  proof (cases J = {})
    case True
    define fc where fc = f undefined
    have f:f = (λ-. fc)
      unfolding fc-def by (intro ext depends-onD2[OF 2(2)]) (auto simp:True)
    then show ?thesis unfolding f by (simp add:prob-space)
  next
    case False
    hence J: J = {i} using 2(1) by auto
    define gc where gc = g undefined
    have g:g = (λ-. gc)
```

unfolding *gc-def* **by** (*intro ext depends-onD2*[*OF 2*(*3*)]) (*auto simp*:*J*)
then show *?thesis* **unfolding** *g* **by** (*simp add*:*prob-space*)
qed
qed

lemma *neg-assoc-imp-measurable*:
assumes *neg-assoc X I*
assumes *i* ∈ *I*
shows *random-variable borel* (*X i*)
using *assms* **unfolding** *neg-assoc-def* **by** *auto*

Even though the assumption was that defining property is true for pairs of monotone functions over the random variables, it is also true for pairs of anti-monotone functions.

lemma *neg-assoc-imp-mult-mono-bounded*:
fixes *f g* :: (*'i* ⇒ *'c*::*linorder-topology*) ⇒ *real*
assumes *neg-assoc X I*
assumes *J* ⊆ *I*
assumes *bounded* (*range f*) *bounded* (*range g*)
assumes *monotone* (≤) (≤≥$_\eta$) *f monotone* (≤) (≤≥$_\eta$) *g*
assumes *depends-on f J depends-on g* (*I*−*J*)
assumes [*measurable*]: *f* ∈ *borel-measurable* (*Pi$_M$ J* (λ-. *borel*))
assumes [*measurable*]: *g* ∈ *borel-measurable* (*Pi$_M$* (*I*−*J*) (λ-. *borel*))
shows
covariance (*f* ∘ *flip X*) (*g* ∘ *flip X*) ≤ *0*
(∫ *ω*. *f* (*λi*. *X i ω*) ∗ *g* (*λi*. *X i ω*) ∂*M*) ≤ *expectation* (*λx*. *f*(*λy*. *X y x*)) ∗ *expectation* (*λx*. *g*(*λy*. *X y x*))
(**is** *?L* ≤ *?R*)
proof −
define *q* **where** *q ι* = (*if ι* = *0 then f else g*) **for** *ι* :: *nat*
define *h* **where** *h ι* = ((∗) (±$_\eta$)) ∘ (*q ι*) **for** *ι* :: *nat*

note [*measurable*] = *neg-assoc-imp-measurable*[*OF assms*(*1*)]
note *bounded* = *integrable-bounded bounded-intros*

have *1*:*bounded* (*range* ((∗) (±$_\eta$) ∘ *q ι*)) *depends-on* (*q ι*) ([*J*,*I*−*J*]!*ι*)
q ι ∈ *PiM* ([*J*,*I*−*J*]!*ι*) (λ-. *borel*) →$_M$ *borel mono* ((∗) (±$_\eta$) ∘ *q ι*) **if** *ι* ∈ {*0*,*1*} **for** *ι*
using *that assms* **unfolding** *q-def conv-rel-to-sign* **by** (*auto intro*:*bounded-mult-comp*)

have *2*: ((∗) (±$_\eta$::*real*)) ∈ *borel* →$_M$ *borel* **by** *simp*

have *3*:∀*ι*<*Suc* (*Suc 0*). *bounded* (*range* (*h ι*))∧*mono*(*h ι*) ∧ *depends-on* (*h ι*) ([*J*,*I*−*J*]!*ι*) ∧
h ι ∈ *PiM* ([*J*,*I*−*J*]!*ι*) (λ-. *borel*) →$_M$ *borel* **unfolding** *All-less-Suc h-def*
by (*intro conjI 1 depends-on-comp measurable-comp*[*OF - 2*]) *auto*

have *covariance* (*f* ∘ *flip X*) (*g* ∘ *flip X*) = *covariance* (*q 0* ∘ *flip X*) (*q 1* ∘ *flip X*)

```
    unfolding q-def by simp
  also have ... = covariance (h 0 ∘ flip X) (h 1 ∘ flip X)
   unfolding h-def covariance-def comp-def by (cases η) (auto simp:algebra-simps)
  also have ... ≤ 0 using 3 assms(1,2) numeral-2-eq-2 unfolding neg-assoc-def
by metis
  finally show covariance (f ∘ flip X) (g ∘ flip X) ≤ 0 by simp

  moreover have m-f: random-variable borel (λx. f(λi. X i x))
    using subsetD[OF assms(2)] by (subst depends-onD[OF assms(7)]) measurable
  moreover have m-g: random-variable borel (λx. g(λi. X i x))
    by (subst depends-onD[OF assms(8)]) measurable
  moreover have integrable M (λω. ((f ∘ (λx y. X y x)) ω)^2) unfolding comp-def
    by (intro bounded bounded-subset[OF assms(3)] measurable-compose[OF m-f])
auto
  moreover have integrable M (λω. ((g ∘ (λx y. X y x)) ω)^2) unfolding comp-def
    by (intro bounded bounded-subset[OF assms(4)] measurable-compose[OF m-g])
auto

  ultimately show ?L ≤ ?R by (subst (asm) covariance-eq) (auto simp:comp-def)
qed

lemma lim-min-n: (λn. min (real n) x) ⟶ x
proof −
  define m where m = nat ⌈x⌉
  have min (real (n+m)) x = x for n unfolding m-def by (intro min-absorb2)
linarith
  hence (λn. min (real (n+m)) x) ⟶ x by simp
  thus ?thesis using LIMSEQ-offset[where k=m] by fast
qed

lemma lim-clamp-n: (λn. clamp (−real n) (real n) x) ⟶ x
proof −
  define m where m = nat ⌈|x|⌉
  have clamp (−real (n+m)) (real (n+m)) x = x for n unfolding m-def
    by (intro clamp-eqI[symmetric]) linarith
  hence (λn. clamp (−real (n+m)) (real (n+m)) x) ⟶ x by simp
  thus ?thesis using LIMSEQ-offset[where k=m] by fast
qed

lemma neg-assoc-imp-mult-mono:
  fixes f g :: ('i ⇒ 'c::linorder-topology) ⇒ real
  assumes neg-assoc X I
  assumes J ⊆ I
  assumes square-integrable M (f ∘ flip X) square-integrable M (g ∘ flip X)
  assumes monotone (≤) (≤≥η) f monotone (≤) (≤≥η) g
  assumes depends-on f J depends-on g (I−J)
  assumes [measurable]: f ∈ borel-measurable (Pi_M J (λ-. borel))
  assumes [measurable]: g ∈ borel-measurable (Pi_M (I−J) (λ-. borel))
   shows (∫ω. f (λi. X i ω) ∗ g (λi. X i ω) ∂M) ≤ (∫x. f(λy. X y x)∂M) ∗ (∫x.
```

g(λ*y*. *X y x*)∂*M*)
  (**is** *?L* ≤ *?R*)
proof −
  let *?cf* = λ*n x*. *clamp* (−*real n*) (*real n*) (*f x*)
  let *?cg* = λ*n x*. *clamp* (−*real n*) (*real n*) (*g x*)

  note [*measurable*] = *neg-assoc-imp-measurable*[*OF assms*(*1*)]

  have *m-f*: *random-variable borel* (λ*x*. *f*(λ*i*. *X i x*))
   using *subsetD*[*OF assms*(*2*)] **by** (*subst depends-onD*[*OF assms*(*7*)]) *measurable*

  have *m-g*: *random-variable borel* (λ*x*. *g*(λ*i*. *X i x*))
   by (*subst depends-onD*[*OF assms*(*8*)]) *measurable*

  note *intro-rules* = *borel-measurable-times measurable-compose*[*OF - clamp-borel*]
AE-I2
   measurable-compose[*OF - borel-measurable-norm*] *lim-clamp-n m-f m-g*

  have *a*: (λ*n*. (∫ω. *?cf n* (λ*i*. *X i* ω) ∗ *?cg n* (λ*i*. *X i* ω) ∂*M*)) ⟶ *?L* **using**
assms(*3*,*4*)
   by (*intro integral-dominated-convergence*[**where** *w*=λω. *norm* (*f*(λ*i*. *X i* ω))∗*norm*
(*g*(λ*i*. *X i* ω))]
     intro-rules tendsto-mult cauchy-schwartz(*1*)[**where** *M*=*M*])
    (*auto intro*!: *clamp-abs-le mult-mono simp add*:*comp-def abs-mult*)

  have (λ*n*. (∫*x*. *?cf n* (λ*y*. *X y x*)∂*M*)) ⟶ (∫*x*. *f*(λ*y*. *X y x*)∂*M*)
   using *square-integrable-imp-integrable*[*OF m-f*] *assms*(*3*) **unfolding** *comp-def*
   by (*intro integral-dominated-convergence*[**where** *w* = λω. *norm* (*f*(λ*i*. *X i* ω))]
intro-rules)
    (*simp-all add*:*clamp-abs-le*)

  moreover have (λ*n*. (∫*x*. *?cg n* (λ*y*. *X y x*)∂*M*)) ⟶ (∫*x*. *g*(λ*y*. *X y x*)∂*M*)
   using *square-integrable-imp-integrable*[*OF m-g*] *assms*(*4*) **unfolding** *comp-def*
   by (*intro integral-dominated-convergence*[**where** *w* = λω. *norm* (*g*(λ*i*. *X i* ω))]
intro-rules)
    (*simp-all add*:*clamp-abs-le*)

  ultimately have *b*: (λ*n*. (∫*x*. *?cf n* (λ*y*. *X y x*)∂*M*) ∗ (∫*x*. *?cg n* (λ*y*. *X y x*)
∂*M*)) ⟶ *?R*
   by (*rule tendsto-mult*)

  show *?thesis*
   by (*intro lim-mono*[*OF - a b*, **where** *N*=*0*] *bounded-clamp-alt assms*(*5*,*6*,*9*,*10*)
monotone-clamp
    neg-assoc-imp-mult-mono-bounded[*OF assms*(*1*,*2*), **where** η=η] *depends-on-comp-2*[*OF*
assms(*7*)]
     measurable-compose[*OF - clamp-borel*] *depends-on-comp-2*[*OF assms*(*8*)])
qed

Property P4 [13]

lemma *neg-assoc-subset*:
assumes $J \subseteq I$
assumes *neg-assoc X I*
shows *neg-assoc X J*
proof (*rule neg-assocI*,*goal-cases*)
case (*1 i*)
then show *?case* **using** *neg-assoc-imp-measurable*[*OF assms*(*2*)] *assms*(*1*) **by** *auto*
next
case (*2 f g K*)
have *a*:$K \subseteq I$ **using** *2 assms*(*1*) **by** *auto*

have $g = g \circ (\lambda m.\ restrict\ m\ (J-K))$
using *2 depends-onD* **unfolding** *comp-def* **by** (*intro ext*) *auto*
also have ... $\in$ *borel-measurable* (Pi_M $(I - K)$ (λ-. *borel*))
using *2 assms*(*1*) **by** (*intro measurable-comp*[*OF measurable-restrict-subset*]) *auto*
finally have $g \in$ *borel-measurable* (Pi_M $(I - K)$ (λ-. *borel*)) **by** *simp*
moreover have *depends-on g* $(I-K)$ **using** *depends-on-mono assms*(*1*) *2*
by (*metis Diff-mono dual-order.eq-iff*)
ultimately show *covariance* ($f \circ$ *flip X*) ($g \circ$ *flip X*) ≤ 0
using *2* **by** (*intro neg-assoc-imp-mult-mono-bounded*[*OF assms*(*2*) *a*, **where** η=*Fwd*]) *simp-all*
qed

lemma *neg-assoc-imp-mult-mono-nonneg*:
fixes *f g* :: ($'i \Rightarrow$ $'c$::*linorder-topology*) $\Rightarrow$ *real*
assumes *neg-assoc X I* $J \subseteq I$
assumes *range f* $\subseteq \{0..\}$ *range g* $\subseteq \{0..\}$
assumes *integrable M* ($f \circ$ *flip X*) *integrable M* ($g \circ$ *flip X*)
assumes *monotone* ($\leq$) ($\leq\geq_\eta$) *f monotone* ($\leq$) ($\leq\geq_\eta$) *g*
assumes *depends-on f J depends-on g* $(I-J)$
assumes $f \in$ *borel-measurable* (Pi_M J (λ-. *borel*)) $g \in$ *borel-measurable* (Pi_M $(I-J)$ (λ-. *borel*))
shows *has-int-that M* ($\lambda\omega$. *f* (*flip X* ω) $*$ *g* (*flip X* ω))
($\lambda r.\ r \leq$ *expectation* ($f \circ$ *flip X*) $*$ *expectation* ($g \circ$ *flip X*))
proof −
let *?cf* = ($\lambda n\ x.\ min$ (*real n*) (*f x*))
let *?cg* = ($\lambda n\ x.\ min$ (*real n*) (*g x*))

define *u* **where** $u = (\lambda\omega.\ f\ (\lambda i.\ X\ i\ \omega) * g\ (\lambda i.\ X\ i\ \omega))$
define *h* **where** $h\ n\ \omega$ = *?cf n* ($\lambda i.\ X\ i\ \omega$) $*$ *?cg n* ($\lambda i.\ X\ i\ \omega$) **for** $n\ \omega$
define *x* **where** x = (*SUP n. expectation* (*h n*))

note *borel-intros* = *borel-measurable-times borel-measurable-const borel-measurable-min borel-measurable-power*
note *bounded-intros'* = *integrable-bounded bounded-intros bounded-const-min*

have *f-meas*: *random-variable borel* ($\lambda x.\ f\ (\lambda i.\ X\ i\ x)$)

using *borel-measurable-integrable*[*OF assms*(*5*)] **by** (*simp add*:*comp-def*)
have *g-meas*: *random-variable borel* (λ*x*. *g* (λ*i*. *X i x*))
using *borel-measurable-integrable*[*OF assms*(*6*)] **by** (*simp add*:*comp-def*)

have *h-int*: *integrable M* (*h n*) **for** *n*
unfolding *h-def* **using** *assms*(*3*,*4*) **by** (*intro bounded-intros′ borel-intros f-meas g-meas*) *fast+*

have *exp-h-le-R*: *expectation* (*h n*) ≤ *expectation* (*f*∘*flip X*) ∗ *expectation* (*g*∘*flip X*) **for** *n*
proof −
have *square-integrable M* ((λ*a*. *min* (*real n*) (*f a*)) ∘ (λ*x y*. *X y x*))
using *assms*(*3*) **unfolding** *comp-def*
by (*intro bounded-intros′ bdd-belowI*[**where** *m*=*0*] *borel-intros f-meas*) *auto*
moreover have *square-integrable M* ((λ*a*. *min* (*real n*) (*g a*)) ∘ (λ*x y*. *X y x*))
using *assms*(*4*) **unfolding** *comp-def*
by (*intro bounded-intros′ bdd-belowI*[**where** *m*=*0*] *borel-intros g-meas*) *auto*
moreover have *monotone* (≤) (≤≥$_\eta$) ((λ*a*. *min* (*real n*) (*f a*)))
using *monotoneD*[*OF assms*(*7*)] **unfolding** *comp-def min-mult-distrib-left*
by (*intro monotoneI*) (*cases* η, *fastforce+*)
moreover have *monotone* (≤) (≤≥$_\eta$) ((λ*a*. *min* (*real n*) (*g a*)))
using *monotoneD*[*OF assms*(*8*)] **unfolding** *comp-def min-mult-distrib-left*
by (*intro monotoneI*) (*cases* η, *fastforce+*)
ultimately have *expectation* (*h n*) ≤ *expectation* (*?cf n*∘*flip X*) ∗ *expectation* (*?cg n*∘*flip X*)
unfolding *h-def comp-def*
by (*intro neg-assoc-imp-mult-mono*[*OF assms*(*1*−*2*)] *borel-intros assms*(*11*,*12*) *depends-on-comp-2*[*OF assms*(*10*)] *depends-on-comp-2*[*OF assms*(*9*)]) (*auto simp*:*comp-def*)
also have ... ≤ *expectation* (*f*∘*flip X*) ∗ *expectation* (*g*∘*flip X*)
using *assms*(*3*,*4*) **by** (*intro mult-mono integral-nonneg-AE AE-I2 integral-mono′ assms*(*5*,*6*)) *auto*
finally show *?thesis* **by** *simp*
qed

have *h-mono-ptw*: *AE* ω *in M*. *mono* (λ*n*. *h n* ω)
using *assms*(*3*,*4*) **unfolding** *h-def* **by** (*intro AE-I2 monoI mult-mono*) *auto*
have *h-mono*: *mono* (λ*n*. *expectation* (*h n*))
by (*intro monoI integral-mono-AE AE-mp*[*OF h-mono-ptw AE-I2*] *h-int*) (*simp add*:*mono-def*)

have *random-variable borel u* **using** *f-meas g-meas* **unfolding** *u-def* **by** (*intro borel-intros*)
moreover have *AE* ω *in M*. (λ*n*. *h n* ω) ⟶ *u* ω
unfolding *h-def u-def* **by** (*intro tendsto-mult lim-min-n AE-I2*)
moreover have *bdd-above* (*range* (λ*n*. *expectation* (*h n*)))
using *exp-h-le-R* **by** (*intro bdd-aboveI*) *auto*
hence (λ*n*. *expectation* (*h n*)) ⟶ *x*
using *LIMSEQ-incseq-SUP*[*OF - h-mono*] **unfolding** *x-def* **by** *simp*

ultimately have *has-bochner-integral M u x* **using** *h-int h-mono-ptw*
by (*intro has-bochner-integral-monotone-convergence*[**where** *f*=*h*])
moreover have $x \leq$ *expectation* (*f*∘*flip X*) ∗ *expectation* (*g*∘*flip X*)
unfolding *x-def* **by** (*intro cSUP-least exp-h-le-R*) *simp*
ultimately show *?thesis* **unfolding** *has-bochner-integral-iff u-def has-int-that-def* **by** *auto*
qed

Property P2 [13]

lemma *neg-assoc-imp-prod-mono*:
fixes *f* :: *'i* ⇒ (*'c*::*linorder-topology*) ⇒ *real*
assumes *finite I*
assumes *neg-assoc X I*
assumes ⋀*i*. *i* ∈ *I* ⟹ *integrable M* (λω. *f i* (*X i* ω))
assumes ⋀*i*. *i* ∈ *I* ⟹ *monotone* (≤) (≤≥$_\eta$) (*f i*)
assumes ⋀*i*. *i* ∈ *I* ⟹ *range* (*f i*)⊆{*0..*}
assumes ⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *borel-measurable borel*
shows *has-int-that M* (λω. (∏ *i*∈*I*. *f i* (*X i* ω))) (λ*r*. *r*≤(∏ *i*∈ *I*. *expectation* (λω. *f i* (*X i* ω))))
using *assms*
proof (*induction I rule*:*finite-induct*)
case *empty* **then show** *?case* **by** (*simp add*:*has-int-that-def*)
next
case (*insert x F*)

define *g* **where** *g v* = *f x* (*v x*) **for** *v*
define *h* **where** *h v* = (∏ *i*∈*F*. *f i* (*v i*)) **for** *v*

have *0*: {*x*} ⊆ *insert x F* **by** *auto*

have *ran-g*: *range g* ⊆ {*0..*} **using** *insert*(*7*) **unfolding** *g-def* **by** *auto*

have *True* = *has-int-that M* (λω. ∏ *i*∈*F*. *f i*(*X i* ω)) (λ*r*. *r*≤(∏ *i*∈*F*. *expectation*(λω. *f i*(*X i* ω))))
by (*intro true-eq-iff insert neg-assoc-subset*[*OF - insert*(*4*)]) *auto*
also have ... = *has-int-that M* (*h* ∘ *flip X*) (λ*r*. *r* ≤ (∏ *i*∈*F*. *expectation* (λω. *f i* (*X i* ω))))
unfolding *h-def* **by** (*intro arg-cong2*[**where** *f*=*has-int-that M*] *refl*)(*simp add*:*comp-def*)
finally have *2*: *has-int-that M* (*h* ∘ *flip X*) (λ*r*. *r* ≤ (∏ *i*∈*F*. *expectation* (λω. *f i* (*X i* ω))))
by *simp*

have (∏ *i*∈*F*. *f i* (*v i*)) ≥ *0* **for** *v* **using** *insert*(*7*) **by** (*intro prod-nonneg*) *auto*
hence *range h* ⊆ {*0..*} **unfolding** *h-def* **by** *auto*
moreover have *integrable M* (*g* ∘ *flip X*) **unfolding** *g-def* **using** *insert*(*5*) **by** (*auto simp*:*comp-def*)
moreover have *3*:*monotone* (≤) (≤≥$_\eta$) (*f x*) **using** *insert*(*6*) **by** *simp*
have *monotone* (≤) (≤≥$_\eta$) *g* **using** *monotoneD*[*OF 3*]

unfolding *g-def* **by** (*intro monotoneI*) (*auto simp*:*comp-def le-fun-def*)
moreover have *4*:*monotone* $(\leq)$ $(\leq\geq_\eta)$ $(f\ i)$ $\bigwedge x.\ f\ i\ x \geq 0$ **if** $i \in F$ **for** *i*
using *that insert*(*6*,*7*) **by** *force+*
hence *monotone* $(\leq)$ $(\leq\geq_\eta)$ *h* **using** *monotoneD*[*OF 4*(*1*)] **unfolding** *h-def*
by (*intro monotoneI*) (*cases* η, *auto intro*:*prod-mono simp*:*comp-def le-fun-def*)
moreover have *depends-on g* $\{x\}$ **unfolding** *g-def* **by** (*intro depends-onI*) *force*
moreover have *depends-on h F*
unfolding *h-def* **by** (*intro depends-onI prod.cong refl*) *force*
hence *depends-on h* $(F - \{x\})$ **using** *insert*(*2*) **by** *simp*
moreover have $g \in$ *borel-measurable* (Pi_M $\{x\}$ (λ-. *borel*)) **unfolding** *g-def*
by (*intro measurable-compose*[*OF - insert*(*8*)] *measurable-component-singleton*) *auto*
moreover have $h \in$ *borel-measurable* (Pi_M *F* (λ-. *borel*))
unfolding *h-def* **by** (*intro borel-measurable-prod measurable-compose*[*OF - insert*(*8*)]
measurable-component-singleton) *auto*
hence $h \in$ *borel-measurable* (Pi_M $(F - \{x\})$ (λ-. *borel*)) **using** *insert*(*2*) **by** *simp*
ultimately have *True* = *has-int-that M* ($\lambda\omega$. *g* (*flip X* ω) $*$ *h* (*flip X* ω))
(λr. $r \leq$ *expectation* ($g \circ$ *flip X*) $*$ *expectation* ($h \circ$ *flip X*))
by (*intro true-eq-iff neg-assoc-imp-mult-mono-nonneg*[*OF insert*(*4*) *0*, **where** η=η]
ran-g has-int-thatD[*OF 2*]) *simp-all*
also have ... = *has-int-that M* ($\lambda\omega$. ($\prod i{\in}$*insert x F*. *f i* (*X i* ω)))
(λr. $r \leq$ *expectation* ($g \circ$ *flip X*) $*$ *expectation* ($h \circ$ *flip X*))
unfolding *g-def h-def* **using** *insert*(*1*,*2*) **by** (*intro arg-cong2*[**where** *f*=*has-int-that M*] *refl*) *simp*
also have ... $\leq$ *has-int-that M* ($\lambda\omega$. ($\prod i{\in}$*insert x F*. *f i* (*X i* ω)))
(λr. $r \leq$ *expectation* ($g \circ$ *flip X*) $*$ ($\prod i \in F$. *expectation* ($f\ i \circ X\ i$)))
using *ran-g has-int-thatD*[*OF 2*] **by** (*intro has-int-that-mono le-trans mult-left-mono*
integral-nonneg-AE AE-I2) (*auto simp*: *comp-def*)
also have ... = *has-int-that M*
($\lambda\omega$. $\prod i{\in}$*insert x F*. *f i* (*X i* ω)) (λr. $r \leq$ ($\prod i{\in}$*insert x F*. *expectation* ($f\ i \circ X\ i$)))
using *insert*(*1*,*2*) **by** (*intro arg-cong2*[**where** *f*=*has-int-that M*] *refl*) (*simp add*:*g-def comp-def*)
finally show *?case* **using** *le-boolE* **by** (*simp add*:*comp-def*)
qed

Property P5 [13]

lemma *neg-assoc-compose*:
fixes *f* :: $'j \Rightarrow ('i \Rightarrow ('c$::*linorder-topology*$)) \Rightarrow ('d$ ::*linorder-topology*$)$
assumes *finite I*
assumes *neg-assoc X I*
assumes $\bigwedge j.\ j \in J \Longrightarrow$ *deps* $j \subseteq I$
assumes $\bigwedge j1\ j2.\ j1 \in J \Longrightarrow j2 \in J \Longrightarrow j1 \neq j2 \Longrightarrow$ *deps j1* $\cap$ *deps j2* $= \{\}$
assumes $\bigwedge j.\ j \in J \Longrightarrow$ *monotone* $(\leq)$ $(\leq\geq_\eta)$ $(f\ j)$
assumes $\bigwedge j.\ j \in J \Longrightarrow$ *depends-on* $(f\ j)$ (*deps j*)
assumes $\bigwedge j.\ j \in J \Longrightarrow f\ j \in$ *borel-measurable* (*PiM* (*deps j*) (λ-. *borel*))
shows *neg-assoc* ($\lambda j\ \omega$. *f j* (λi. *X i* ω)) *J*

proof (*rule neg-assocI*, *goal-cases*)
  case (*1 i*)
  note [*measurable*] = *neg-assoc-imp-measurable*[*OF assms*(*2*)] *assms*(*7*)[*OF 1*]
  note *3* = *assms*(*3*)[*OF 1*]
  have *2*: $f\ i\ (\lambda i.\ X\ i\ \omega) = f\ i\ (\lambda i{\in}deps\ i.\ X\ i\ \omega)$ **for** ω
    using *3* **by** (*intro depends-onD2*[*OF assms*(*6*)] *1*) *fastforce*
  show *?case* **unfolding** *2* **by** *measurable* (*rule subsetD*[*OF 3*])
next
  case (*2 g h K*)

  let $?g = (\lambda\omega.\ g\ (\lambda j.\ f\ j\ \omega))$
  let $?h = (\lambda\omega.\ h\ (\lambda j.\ f\ j\ \omega))$

  note *dep-f* = *depends-onD*[*OF depends-on-mono*[*OF - assms*(*6*)],*symmetric*]

  have *g-alt-1*: $?g = (\lambda\omega.\ g\ (\lambda j \in J.\ f\ j\ \omega))$
    using *2*(*1*) **by** (*intro ext depends-onD*[*OF depends-on-mono*[*OF - 2*(*2*)]]) *auto*
  have *g-alt-2*: $?g = (\lambda\omega.\ g\ (\lambda j \in K.\ f\ j\ \omega))$
    by (*intro ext depends-onD*[*OF 2*(*2*)])
  have *g-alt-3*: $?g = (\lambda\omega.\ g\ (\lambda j \in K.\ f\ j\ (restrict\ \omega\ (deps\ j))))$ **unfolding** *g-alt-2*
using *2*(*1*)
   by (*intro restrict-ext ext arg-cong*[**where** *f*=*g*] *depends-onD*[*OF assms*(*6*)]) *auto*

  have *h-alt-1*: $?h = (\lambda\omega.\ h\ (\lambda j \in J.\ f\ j\ \omega))$
    by (*intro ext depends-onD*[*OF depends-on-mono*[*OF - 2*(*3*)]]) *auto*
  have *h-alt-2*: $?h = (\lambda\omega.\ h\ (\lambda j \in J{-}K.\ f\ j\ \omega))$
    by (*intro ext depends-onD*[*OF 2*(*3*)])
  have *h-alt-3*: $?h = (\lambda\omega.\ h\ (\lambda j \in J{-}K.\ f\ j\ (restrict\ \omega\ (deps\ j))))$ **unfolding**
h-alt-2
   by (*intro restrict-ext ext arg-cong*[**where** *f*=*h*] *depends-onD*[*OF assms*(*6*)]) *auto*

  have *3*: $\bigcup\ (deps\ `\ (J{-}K)) \subseteq I - \bigcup\ (deps\ `\ K)$ **using** *assms*(*3*,*4*) *2*(*1*) **by** *blast*

  have $\bigcup\ (deps\ `\ K) \subseteq I$ **using** *2*(*1*) *assms*(*3*) **by** *auto*
  moreover have *bounded* (*range ?g*) *bounded* (*range ?h*)
    using *2*(*6*,*7*) **by** (*auto intro*: *bounded-subset*)
  moreover have *monotone* ($\le$) ($\le\ge_\eta$) *?g*
    unfolding *g-alt-1* **using** *monotoneD*[*OF assms*(*5*)]
    by (*intro monotoneI*) (*cases* η, *auto intro*!:*monoD*[*OF 2*(*4*)] *le-funI*)
  moreover have *monotone* ($\le$) ($\le\ge_\eta$) *?h*
    unfolding *h-alt-1* **using** *monotoneD*[*OF assms*(*5*)]
    by (*intro monotoneI*) (*cases* η, *auto intro*!:*monoD*[*OF 2*(*5*)] *le-funI*)
  moreover have *depends-on ?g* ($\bigcup\ (deps\ `\ K)$)
    using *2*(*1*) **unfolding** *g-alt-2*
      by (*intro depends-onI arg-cong*[**where** *f*=*g*] *restrict-ext depends-onD2*[*OF*
assms(*6*)]) *auto*
  moreover have *depends-on ?h* ($\bigcup\ (deps\ `\ (J{-}K))$)
    unfolding *h-alt-2*
      by (*intro depends-onI arg-cong*[**where** *f*=*h*] *restrict-ext depends-onD2*[*OF*

assms(*6*)]) *auto*
hence *depends-on ?h* $(I - \bigcup$ (*deps* ‘ *K*)) **using** *depends-on-mono*[*OF 3*] **by** *auto*
moreover have *?g* $\in$ *borel-measurable* (Pi_M ($\bigcup$ (*deps* ‘ *K*)) (λ-. *borel*))
unfolding *g-alt-3* **using** *2*(*1*)
by (*intro measurable-compose*[*OF - 2*(*8*)] *measurable-compose*[*OF - assms*(*7*)]
measurable-restrict measurable-component-singleton) *auto*
moreover have *?h* $\in$ *borel-measurable* (Pi_M ($I - \bigcup$ (*deps* ‘ *K*)) (λ-. *borel*))
unfolding *h-alt-3* **using** *3*
by (*intro measurable-compose*[*OF - 2*(*9*)] *measurable-compose*[*OF - assms*(*7*)]
measurable-restrict
measurable-component-singleton) *auto*
ultimately have *covariance* (*?g* $\circ$ *flip X*) (*?h* $\circ$ *flip X*) $\leq$ *0*
by (*rule neg-assoc-imp-mult-mono-bounded*[*OF assms*(*2*), **where** *J*=$\bigcup$(*deps* ‘
K) **and** η=η])
thus *covariance* ($g \circ (\lambda x\ y.\ f\ y\ (\lambda i.\ X\ i\ x))$) ($h \circ (\lambda x\ y.\ f\ y\ (\lambda i.\ X\ i\ x))$) $\leq$ *0*
by (*simp add*:*comp-def*)
qed

lemma *neg-assoc-compose-simple*:
fixes *f* :: $'i \Rightarrow$ ($'c$::*linorder-topology*) $\Rightarrow$ ($'d$ ::*linorder-topology*)
assumes *finite I*
assumes *neg-assoc X I*
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *monotone* ($\leq$) ($\leq\geq_\eta$) (*f i*)
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow f\ i \in$ *borel-measurable borel*
shows *neg-assoc* ($\lambda i\ \omega.\ f\ i\ (X\ i\ \omega)$) *I*
proof –
have *depends-on* ($\lambda\omega.\ f\ i\ (\omega\ i)$) {*i*} **if** $i \in I$ **for** *i*
by (*intro depends-onI*) *auto*
moreover have *monotone* ($\leq$) ($\leq\geq_\eta$) ($\lambda\omega.\ f\ i\ (\omega\ i)$) **if** $i \in I$ **for** *i*
using *monotoneD*[*OF assms*(*3*)[*OF that*]] **by** (*intro monotoneI*) (*cases* η, *auto*
dest:*le-funE*)
ultimately show *?thesis*
by (*intro neg-assoc-compose*[*OF assms*(*1*,*2*), **where** *deps*=$\lambda i.$ {*i*} **and** η=η])
simp-all
qed

lemma *covariance-distr*:
fixes *f g* :: $'b \Rightarrow$ *real*
assumes [*measurable*]: $\varphi \in M \rightarrow_M N$ $f \in$ *borel-measurable N* $g \in$ *borel-measurable*
N
shows *prob-space.covariance* (*distr M N* φ) *f g* = *covariance* ($f \circ \varphi$) ($g \circ \varphi$) (**is**
?L = *?R*)
proof –
let *?M'* = *distr M N* φ
have *ps-distr*: *prob-space ?M'* **by** (*intro prob-space-distr*) *measurable*
interpret *p2*: *prob-space ?M'*
using *ps-distr* **by** *auto*
have *?L* = *expectation* ($\lambda x.\ (f(\varphi\ x) -$ *expectation* $(\lambda x.\ f(\varphi\ x))) * (g(\varphi\ x) -$ *expectation*
$(\lambda x.\ g(\varphi\ x)))$)

```
    unfolding p2.covariance-def by (subst (1 2 3) integral-distr) measurable
  also have ... = ?R
    unfolding covariance-def comp-def by simp
  finally show ?thesis by simp
qed

lemma neg-assoc-iff-distr:
  assumes [measurable]: ⋀i. i ∈ I ⟹ X i ∈ borel-measurable M
  shows neg-assoc X I ⟷
    prob-space.neg-assoc (distr M (PiM I (λ-. borel)) (λω. λi∈I. X i ω)) (flip id) I
    (is ?L ⟷ ?R)
proof
  let ?M′ = distr M (Pi_M I (λ-. borel)) (λω. λi∈I. X i ω)
  have ps-distr: prob-space ?M′
    by (intro prob-space-distr) measurable

  interpret p2: prob-space ?M′
    using ps-distr by auto

  show ?R if ?L
  proof (rule p2.neg-assocI, goal-cases)
    case (1 i)
    thus ?case using assms that unfolding id-def by measurable
  next
    case (2 f g J)

    have dep-I: depends-on f I depends-on g I
       using depends-on-mono[OF Diff-subset[of I J]] depends-on-mono[OF 2(1)]
2(2−3) by auto

    have f-meas[measurable]: (λx. f x) ∈ borel-measurable (Pi_M I (λ-. borel))
       by (subst depends-onD[OF 2(2)]) (intro 2 measurable-compose[OF measur-
able-restrict-subset])

    have g-meas[measurable]: (λx. g x) ∈ borel-measurable (Pi_M I (λ-. borel))
      by (subst depends-onD[OF 2(3)])
        (intro 2 measurable-compose[OF measurable-restrict-subset], auto)

    have covariance (f ∘ id ∘ (λω. λi∈I. X i ω)) (g ∘ id ∘ (λω. λi∈I. X i ω)) =
      covariance (f ∘ flip X) (g ∘ flip X)
    using depends-onD[OF dep-I(2)] depends-onD[OF dep-I(1)] by (simp add:comp-def)
    also have ... ≤ 0
     using 2 by (intro neg-assoc-imp-mult-mono-bounded[OF that 2(1,6,7), where
η=Fwd]) simp-all
    finally have covariance (f ∘ id ∘ (λω. λi∈I. X i ω)) (g ∘ id ∘ (λω. λi∈I. X i
ω)) ≤ 0 by simp
    thus ?case by (subst covariance-distr) measurable
  qed
```

show *?L* **if** *?R*
proof (*rule neg-assocI*, *goal-cases*)
case (*1 i*)
then show *?case* **by** *measurable*
next
case (*2 f g J*)

have *dep-I*: *depends-on f I depends-on g I*
using *depends-on-mono*[*OF Diff-subset*[*of I J*]] *depends-on-mono*[*OF 2*(*1*)] *2*(*2−3*) **by** *auto*

have *f-meas*[*measurable*]: (λ*x*. *f x*) ∈ *borel-measurable* (*Pi*$_M$ *I* (λ-. *borel*))
by (*subst depends-onD*[*OF 2*(*2*)]) (*intro 2 measurable-compose*[*OF measurable-restrict-subset*])

have *g-meas*[*measurable*]: (λ*x*. *g x*) ∈ *borel-measurable* (*Pi*$_M$ *I* (λ-. *borel*))
by (*subst depends-onD*[*OF 2*(*3*)])
(*intro 2 measurable-compose*[*OF measurable-restrict-subset*], *auto*)

note [*measurable*] = *2*(*8*,*9*)
have *covariance* (*f* ∘ (λ*x y*. *X y x*)) (*g* ∘ (λ*x y*. *X y x*)) =
covariance (*f* ∘ (λω. λ*i*∈*I*. *X i* ω)) (*g* ∘ (λω. λ*i*∈*I*. *X i* ω))
using *depends-onD*[*OF dep-I*(*2*)] *depends-onD*[*OF dep-I*(*1*)] **by** (*simp add*:*comp-def*)
also have ... = *p2.covariance* (*f* ∘ *id*) (*g* ∘ *id*) **by** (*subst covariance-distr*) *measurable*
also have ... ≤ *0*
using *2* **by** (*intro p2.neg-assoc-imp-mult-mono-bounded*[*OF that 2*(*1*), **where** η=*Fwd*])
(*simp-all add*:*comp-def*)
finally show *?case* **by** *simp*
qed
qed

lemma *neg-assoc-cong*:
assumes *finite I*
assumes [*measurable*]: ⋀*i*. *i* ∈ *I* ⟹ *Y i* ∈ *borel-measurable M*
assumes *neg-assoc X I* ⋀*i*. *i* ∈ *I* ⟹ *AE* ω *in M*. *X i* ω = *Y i* ω
shows *neg-assoc Y I*
proof –
have [*measurable*]: ⋀*i*. *i* ∈ *I* ⟹ *X i* ∈ *borel-measurable M*
using *neg-assoc-imp-measurable*[*OF assms*(*3*)] **by** *auto*

let *?B* = (λ-. *borel*)
have *a*:*AE x in M*. (∀ *i*∈*I*. (*X i x* = *Y i x*)) **by** (*intro AE-finite-allI assms*)
have *AE x in M*. (λ*i*∈*I*. *X i x*) = (λ*i*∈*I*. *Y i x*) **by** (*intro AE-mp*[*OF a AE-I2*]) *auto*
hence *b*:*distr M* (*PiM I ?B*) (λω. λ*i*∈*I*. *X i* ω) = *distr M* (*PiM I ?B*) (λω. λ*i*∈*I*. *Y i* ω)
by (*intro distr-cong-AE refl*) *measurable*

have *prob-space.neg-assoc* (*distr M* (*PiM I* (λ-. *borel*)) (λω. λi∈I. *X i* ω)) (*flip id*) *I*
using *assms*(2,3) **by** (*intro iffD1*[*OF neg-assoc-iff-distr*]) *measurable*
thus *?thesis* **unfolding** *b* **using** *assms*(2)
by (*intro iffD2*[*OF neg-assoc-iff-distr*[**where** *I*=*I*]]) *auto*
qed

lemma *neg-assoc-reindex-aux*:
assumes *inj-on h I*
assumes *neg-assoc X* (*h* ‘ *I*)
shows *neg-assoc* (λk. *X* (*h k*)) *I*
proof (*rule neg-assocI*, *goal-cases*)
case (*1 i*) **thus** *?case* **using** *neg-assoc-imp-measurable*[*OF assms*(2)] **by** *simp*
next
case (*2 f g J*)
let *?f* = (λω. *f* (*compose J* ω *h*))
let *?g* = (λω. *g* (*compose* (*I*−*J*) ω *h*))

note *neg-assoc-imp-mult-mono-intros* =
neg-assoc-imp-mult-mono-bounded(1)[*OF assms*(2), **where** *J*=*h*‘*J* **and** η=*Fwd*]
measurable-compose[*OF* - *2*(8)] *measurable-compose*[*OF* - *2*(9)]
measurable-compose[*OF* - *measurable-finmap-compose*]
bounded-range-imp[*OF 2*(6)] *bounded-range-imp*[*OF 2*(7)]

have [*simp*]:*h* ‘ *I* − *h* ‘ *J* = *h* ‘ (*I*−*J*)
using *assms*(1) *2*(1) **by** (*simp add*: *inj-on-image-set-diff*)

have *covariance* (*f*∘(λx y. *X*(*h y*)*x*)) (*g*∘(λx y. *X*(*h y*)*x*)) = *covariance* (*?f* ∘ *flip X*) (*?g* ∘ *flip X*)
unfolding *comp-def*
by (*intro arg-cong2*[**where** *f*=*covariance*] *ext depends-onD2*[*OF 2*(2)] *depends-onD2*[*OF 2*(3)])
(*auto simp*:*compose-def*)
also have . . . ≤ *0* **using** *2*(1)
by (*intro neg-assoc-imp-mult-mono-intros monotoneI depends-onI*) (*auto intro*!:
monoD[*OF 2*(4)] *monoD*[*OF 2*(5)] *simp*:*le-fun-def compose-def restrict-def cong*:*if-cong*)
finally show *?case* **by** *simp*
qed

lemma *neg-assoc-reindex*:
assumes *inj-on h I finite I*
shows *neg-assoc X* (*h* ‘ *I*) ⟷ *neg-assoc* (λk. *X* (*h k*)) *I* (**is** *?L* ⟷ *?R*)
proof
assume *?L*
thus *?R* **using** *neg-assoc-reindex-aux*[*OF assms*(1)] **by** *blast*
next
note *inv-h-inj* = *inj-on-the-inv-into*[*OF assms*(1)]
assume *a*:*?R*

hence *b*:*neg-assoc* (λ*k*. *X* (*h* (*the-inv-into I h k*))) (*h* ' *I*)
using *the-inv-into-onto*[*OF assms*(*1*)] **by** (*intro neg-assoc-reindex-aux*[*OF inv-h-inj*]) *auto*
show *?L*
using *f-the-inv-into-f*[*OF assms*(*1*)] *neg-assoc-imp-measurable*[*OF a*] *assms*(*2*)
by (*intro neg-assoc-cong*[*OF - - b*]) *auto*
qed

lemma *measurable-compose-merge-1*:
assumes *depends-on h K*
assumes $h \in PiM\ K\ M' \to_M N$ $K \subseteq I \cup J$
assumes (λ*x*. *restrict* (*fst* (*f x*)) ($K \cap I$)) $\in A \to_M PiM\ (K \cap I)\ M'$
assumes (λ*x*. *restrict* (*snd* (*f x*)) ($K \cap J$)) $\in A \to_M PiM\ (K \cap J)\ M'$
shows (λ*x*. *h*(*merge I J* (*f x*))) $\in A \to_M N$
proof −
let *?f1* = λ*x*. *fst* (*f x*)
let *?f2* = λ*x*. *snd* (*f x*)
let *?g1* = λ*x*. *restrict* (*fst* (*f x*)) ($K \cap I$)
let *?g2* = λ*x*. *restrict* (*snd* (*f x*)) ($K \cap J$)

have *a1*:(λ*x*. *merge I J* (*?g1 x*, *?g2 x*) *i*) $\in A \to_M M'\ i$ **if** $i \in K \cap I$ **for** *i*
using *that measurable-compose*[*OF assms*(*4*) *measurable-component-singleton*[*OF that*]]
by (*simp add*:*merge-def*)

have *a2*:(λ*x*. *merge I J* (*?g1 x*, *?g2 x*) *i*) $\in A \to_M M'\ i$ **if** $i \in K \cap J$ $i \notin I$ **for** *i*
using *that measurable-compose*[*OF assms*(*5*) *measurable-component-singleton*[*OF that*(*1*)]]
by (*simp add*:*merge-def*)

have *a*:(λ*x*. *merge I J* (*?g1 x*, *?g2 x*) *i*) $\in A \to_M M'\ i$ **if** $i \in K$ **for** *i*
using *assms*(*3*) *a1 a2 that* **by** *auto*

have (λ*x*. *h*(*merge I J* (*f x*))) = (λ*x*. *h*(*merge I J* (*?f1 x*, *?f2 x*))) **by** *simp*
also have ... = (λ*x*. *h*(λ$i \in K$. *merge I J* (*?f1 x*, *?f2 x*) *i*))
using *depends-onD*[*OF assms*(*1*)] **by** *simp*
also have ... = (λ*x*. *h*(λ$i \in K$. *merge I J* (*?g1 x*, *?g2 x*) *i*))
by (*intro ext arg-cong*[**where** *f*=*h*]) (*auto simp*:*comp-def restrict-def merge-def case-prod-beta*)
also have ... $\in A \to_M N$
by (*intro measurable-compose*[*OF - assms*(*2*)] *measurable-restrict a*)
finally show *?thesis* **by** *simp*
qed

lemma *measurable-compose-merge-2*:
assumes *depends-on h K* $h \in PiM\ K\ M' \to_M N$ $K \subseteq I \cup J$
assumes (λ*x*. *restrict* (*f x*) ($K \cap I$)) $\in A \to_M PiM\ (K \cap I)\ M'$
assumes (λ*x*. *restrict* (*g x*) ($K \cap J$)) $\in A \to_M PiM\ (K \cap J)\ M'$
shows (λ*x*. *h*(*merge I J* (*f x*, *g x*))) $\in A \to_M N$

using *assms* **by** (*intro measurable-compose-merge-1*[*OF assms*(*1–3*)]) *simp-all*

lemma *neg-assoc-combine*:
fixes *I I1 I2* :: *'i set*
fixes *X* :: *'i* ⇒ *'a* ⇒ (*'b*::*linorder-topology*)
assumes *finite I I1* ∪ *I2* = *I I1* ∩ *I2* = {}
assumes *indep-var* (*PiM I1* (λ-. *borel*)) (λω. λ*i*∈*I1*. *X i* ω) (*PiM I2* (λ-. *borel*)) (λω. λ*i*∈*I2*. *X i* ω)
assumes *neg-assoc X I1*
assumes *neg-assoc X I2*
shows *neg-assoc X I*
proof –
define *X'* **where** *X' i* = (*if i* ∈ *I then X i else* (λ-. *undefined*)) **for** *i*

have *X-measurable*: *random-variable borel* (*X i*) **if** *i* ∈ *I* **for** *i*
using *that assms*(*2*) *neg-assoc-imp-measurable*[*OF assms*(*5*)]
neg-assoc-imp-measurable[*OF assms*(*6*)] **by** *auto*

have *rv*[*measurable*]: *random-variable borel* (*X' i*) **for** *i*
unfolding *X'-def* **using** *X-measurable* **by** *auto*

have *na-I1*: *neg-assoc X' I1* **using** *neg-assoc-cong*
unfolding *X'-def* **using** *assms*(*1*,*2*) *neg-assoc-imp-measurable*[*OF assms*(*5*)]
by (*intro neg-assoc-cong*[*OF - - assms*(*5*)] *AE-I2*) *auto*

have *na-I2*: *neg-assoc X' I2* **using** *neg-assoc-cong*
unfolding *X'-def* **using** *assms*(*1*,*2*) *neg-assoc-imp-measurable*[*OF assms*(*6*)]
by (*intro neg-assoc-cong*[*OF - - assms*(*6*)] *AE-I2*) *auto*

have *iv*:*indep-var*(*PiM I1* (λ-. *borel*))(λω. λ*i*∈*I1*. *X' i* ω)(*PiM I2* (λ-. *borel*))(λω. λ*i*∈*I2*. *X' i* ω)
using *assms*(*2*,*4*) **unfolding** *indep-var-def X'-def* **by** (*auto simp add*:*restrict-def cong*:*if-cong*)

let *?N* = Pi_M *I1* (λ-. *borel*) $\bigotimes_M$ Pi_M *I2* (λ-. *borel*)
let *?A* = *distr M* (Pi_M *I1* (λ-. *borel*)) (λω. λ*i*∈*I1*. *X' i* ω)
let *?B* = *distr M* (Pi_M *I2* (λ-. *borel*)) (λω. λ*i*∈*I2*. *X' i* ω)
let *?H* = *distr M ?N* (λω. (λ*i*∈*I1*. *X' i* ω, λ*i*∈*I2*. *X' i* ω))

have *indep*: *?H* = (*?A* $\bigotimes_M$ *?B*)
and *rvs*: *random-variable* (Pi_M *I1* (λ-. *borel*)) (λω. λ*i*∈*I1*. *X' i* ω)
random-variable (Pi_M *I2* (λ-. *borel*)) (λω. λ*i*∈*I2*. *X' i* ω)
using *iffD1*[*OF indep-var-distribution-eq iv*] **by** *auto*

interpret *pa*: *prob-space ?A* **by** (*intro prob-space-distr rvs*)
interpret *pb*: *prob-space ?B* **by** (*intro prob-space-distr rvs*)
interpret *pair-sigma-finite ?A ?B*
using *pa.sigma-finite-measure pb.sigma-finite-measure* **by** (*intro pair-sigma-finite.intro*)

interpret *pab*: *prob-space* (*?A* $\bigotimes_M$ *?B*)
by (*intro prob-space-pair pa.prob-space-axioms pb.prob-space-axioms*)

have *pa-na*: *pa.neg-assoc* (*λx y. y x*) *I1*
using *assms*(*2*) *iffD1*[*OF neg-assoc-iff-distr na-I1*] **by** *fastforce*

have *pb-na*: *pb.neg-assoc* (*λx y. y x*) *I2*
using *assms*(*2*) *iffD1*[*OF neg-assoc-iff-distr na-I2*] **by** *fastforce*

have *na-X′*: *neg-assoc X′ I*
proof (*rule neg-assocI2*, *goal-cases*)
case (*1 i*) **thus** *?case* **by** *measurable*
next
case (*2 f g K*)

note *bounded-intros* =
bounded-range-imp[*OF 2*(*6*)] *bounded-range-imp*[*OF 2*(*7*)] *pa.integrable-bounded*
pb.integrable-bounded pab.integrable-bounded bounded-intros pb.finite-measure-axioms

have [*measurable*]:
restrict x I ∈ *space* (*Pi*$_M$ *I* (*λ-. borel*)) **for** *x* :: (*′i* ⇒ *′b*) **and** *I* **by** (*simp add*:*space-PiM*)

have *a*: *K* ⊆ *I1* ∪ *I2* **using** *2 assms*(*2*) **by** *auto*
have *b*: *I*−*K* ⊆ *I1* ∪ *I2* **using** *assms*(*2*) **by** *auto*

note *merge-1* = *measurable-compose-merge-2*[*OF 2*(*2*,*8*) *a*] *measurable-compose-merge-2*[*OF 2*(*3*,*9*) *b*]
note *merge-2* = *measurable-compose-merge-1*[*OF 2*(*2*,*8*) *a*] *measurable-compose-merge-1*[*OF 2*(*3*,*9*) *b*]

have *merge-mono*:
merge I1 I2 (*w*, *y*) ≤ *merge I1 I2* (*x*, *z*) **if** *w* ≤ *x y* ≤ *z* **for** *w x y z* :: *′i* ⇒ *′b*
using *le-funD*[*OF that*(*1*)] *le-funD*[*OF that*(*2*)] **unfolding** *merge-def* **by** (*intro le-funI*) *auto*

have *split-h*: *h* ∘ *flip X′* = (*λω. h* (*merge I1 I2* (*λi*∈*I1. X′ i ω*, *λi*∈*I2. X′ i ω*)))
if *depends-on h I* **for** *h* :: *-* ⇒ *real*
using *assms*(*2*) **unfolding** *comp-def*
by (*intro ext depends-onD2*[*OF that*]) (*auto simp*:*restrict-def merge-def*)

have *depends-on f I depends-on g I*
using *2*(*1*) **by** (*auto intro*:*depends-on-mono*[*OF - 2*(*2*)] *depends-on-mono*[*OF - 2*(*3*)])
note *split* = *split-h*[*OF this*(*1*)] *split-h*[*OF this*(*2*)]

have *step-1*: (∫ *y. f* (*merge I1 I2* (*x*, *y*)) ∗ *g* (*merge I1 I2* (*x*, *y*)) ∂*?B*) ≤
(∫ *y. f* (*merge I1 I2* (*x*, *y*))∂ *?B*) ∗ (∫ *y. g* (*merge I1 I2* (*x*, *y*)) ∂*?B*) (**is** *?L1* ≤ *?R1*)

for x
proof –
have *step1-1*: *monotone* $(\leq)$ $(\leq\geq_{Fwd})$ $(\lambda a.\ f\ (merge\ I1\ I2\ (x,\ a)))$
unfolding *dir-le* **by** (*intro monoI monoD*[*OF 2*(*4*)] *merge-mono*) *simp*
have *step1-2*: *monotone* $(\leq)$ $(\leq\geq_{Fwd})$ $(\lambda a.\ g\ (merge\ I1\ I2\ (x,\ a)))$
unfolding *dir-le* **by** (*intro monoI monoD*[*OF 2*(*5*)] *merge-mono*) *simp*
have *step1-3*: *depends-on* $(\lambda a.\ f\ (merge\ I1\ I2\ (x,\ a)))$ $(K \cap I2)$
by (*subst depends-onD*[*OF 2*(*2*)])
(*auto intro*:*depends-onI simp*:*merge-def restrict-def cong*:*if-cong*)
have *step1-4*: *depends-on* $(\lambda a.\ g\ (merge\ I1\ I2\ (x,\ a)))$ $(I2 - K \cap I2)$
by (*subst depends-onD*[*OF 2*(*3*)])
(*auto intro*:*depends-onI simp*:*merge-def restrict-def cong*:*if-cong*)
show *?thesis*
by (*intro pb.neg-assoc-imp-mult-mono-bounded*(*2*)[*OF pb-na*, **where** η=*Fwd* **and** J=$K \cap I2$]
bounded-intros merge-1 step1-1 step1-2 step1-3 step1-4) *measurable*
qed

have *step2-1*: *monotone* $(\leq)$ $(\leq\geq_{Fwd})$ $(\lambda a.\ pb.expectation\ (\lambda y.\ f\ (merge\ I1\ I2\ (a,y))))$
unfolding *dir-le*
by (*intro monoI integral-mono bounded-intros merge-1 monoD*[*OF 2*(*4*)] *merge-mono*) *measurable*

have *step2-2*: *monotone* $(\leq)$ $(\leq\geq_{Fwd})$ $(\lambda a.\ pb.expectation\ (\lambda y.\ g\ (merge\ I1\ I2\ (a,y))))$
unfolding *dir-le*
by (*intro monoI integral-mono bounded-intros merge-1 monoD*[*OF 2*(*5*)] *merge-mono*) *measurable*

have *step2-3*: *depends-on* $(\lambda a.\ pb.expectation\ (\lambda y.\ f\ (merge\ I1\ I2\ (a,\ y))))$ $(K \cap I1)$
by (*subst depends-onD*[*OF 2*(*2*)])
(*auto intro*:*depends-onI simp*:*merge-def restrict-def cong*:*if-cong*)

have *step2-4*: *depends-on* $(\lambda a.\ pb.expectation\ (\lambda y.\ g\ (merge\ I1\ I2\ (a,\ y))))$ $(I1 - K \cap I1)$
by (*subst depends-onD*[*OF 2*(*3*)])
(*auto intro*:*depends-onI simp*:*merge-def restrict-def cong*:*if-cong*)

have $(\int \omega.\ (f \circ flip\ X')\ \omega * (g \circ flip\ X')\ \omega\ \partial M) = (\int \omega.\ f\ (merge\ I1\ I2\ \omega) * g(merge\ I1\ I2\ \omega)\ \partial ?H)$
unfolding *split* **by** (*intro integral-distr*[*symmetric*] *merge-2 borel-measurable-times*) *measurable*
also have ... $= (\int \omega.\ f(merge\ I1\ I2\ \omega) * g(merge\ I1\ I2\ \omega)\ \partial\ (?A \bigotimes_M\ ?B))$
unfolding *indep* **by** *simp*
also have ... $= (\int x.\ (\int y.\ f(merge\ I1\ I2\ (x,y)) * g(merge\ I1\ I2\ (x,y))\ \partial ?B)\ \partial ?A)$
by (*intro integral-fst'*[*symmetric*] *bounded-intros merge-2 borel-measurable-times*)

measurable
also have ... $\leq$ (∫ *x*. (∫ *y*. *f*(*merge I1 I2* (*x*,*y*)) ∂*?B*) ∗ (∫ *y*. *g*(*merge I1 I2* (*x*,*y*)) ∂*?B*) ∂*?A*)
by (*intro integral-mono-AE bounded-intros step-1 AE-I2 pb.borel-measurable-lebesgue-integral borel-measurable-times iffD2*[*OF measurable-split-conv*] *merge-2*) *measurable*
also have ... ≤(∫ *x*.(∫ *y*. *f*(*merge I1 I2* (*x*,*y*))∂*?B*)∂*?A*)∗(∫ *x*.(∫ *y*. *g*(*merge I1 I2*(*x*,*y*))∂*?B*)∂*?A*)
by (*intro pa.neg-assoc-imp-mult-mono-bounded*[*OF pa-na*, **where** *η*=*Fwd* **and** *J*=*K* ∩ *I1*]
bounded-intros pb.borel-measurable-lebesgue-integral iffD2[*OF measurable-split-conv*]
merge-2 step2-1 step2-2 step2-3 step2-4) *measurable*
also have ... = (∫ *ω*. *f*(*merge I1 I2 ω*) ∂(*?A* ⨂$_M$ *?B*)) ∗ (∫ *ω*. *g*(*merge I1 I2 ω*) ∂(*?A* ⨂$_M$ *?B*))
by (*intro arg-cong2*[**where** *f*=(∗)] *integral-fst′ merge-2 bounded-intros*) *measurable*
also have ... = (∫ *ω*. *f*(*merge I1 I2 ω*) ∂*?H*) ∗ (∫ *ω*. *g*(*merge I1 I2 ω*) ∂*?H*)
unfolding *indep* **by** *simp*
also have ... = (∫ *ω*. (*f*∘*flip X′*) *ω* ∂*M*) ∗ (∫ *ω*. (*g*∘*flip X′*) *ω* ∂*M*)
unfolding *split* **by** (*intro arg-cong2*[**where** *f*=(∗)] *integral-distr merge-2*) *measurable*
finally show *?case* **by** (*simp add*:*comp-def*)
qed
show *?thesis* **by** (*intro neg-assoc-cong*[*OF assms*(*1*) *X-measurable na-X′*]) (*simp-all add*:*X′-def*)
qed

Property P7 [13]

lemma *neg-assoc-union*:
fixes *I* :: *′i set*
fixes *p* :: *′j* ⇒ *′i set*
fixes *X* :: *′i* ⇒ *′a* ⇒ (*′b*::*linorder-topology*)
assumes *finite I* ⋃(*p* ‘ *J*) = *I*
assumes *indep-vars* (λ*j*. *PiM* (*p j*) (λ-. *borel*)) (λ*j ω*. λ*i* ∈ *p j*. *X i ω*) *J*
assumes ⋀*j*. *j* ∈ *J* ⟹ *neg-assoc X* (*p j*)
assumes *disjoint-family-on p J*
shows *neg-assoc X I*
proof −
let *?B* = (λ-. *borel*)
define *T* **where** *T* = {*j* ∈ *J*. *p j* ≠ {}}

define *g* **where** *g i* = (*THE j*. *j* ∈ *J* ∧ *i* ∈ *p j*) **for** *i*
have *g*: *g i* = *j* **if** *i* ∈ *p j j* ∈ *J* **for** *i j* **unfolding** *g-def*
proof (*rule the1-equality*)
show ∃!*j*. *j* ∈ *J* ∧ *i* ∈ *p j*
using *assms*(*5*) *that* **unfolding** *bex1-def disjoint-family-on-def* **by** *auto*
show *j* ∈ *J* ∧ *i* ∈ *p j* **using** *that* **by** *auto*
qed

have *ran-T*: $T \subseteq J$ **unfolding** *T-def* **by** *simp*
hence *disjoint-family-on p T* **using** *assms*(*5*) *disjoint-family-on-mono* **by** *metis*
moreover have *finite* $(\bigcup (p \text{ ` } T))$ **using** *ran-T assms*(*1*,*2*)
by (*meson Union-mono finite-subset image-mono*)
moreover have $\bigwedge i.\ i \in T \Longrightarrow p\ i \neq \{\}$ **unfolding** *T-def* **by** *auto*
ultimately have *fin-T*: *finite T* **using** *infinite-disjoint-family-imp-infinite-UNION*
by *auto*

have *neg-assoc X* $(\bigcup (p \text{ ` } T))$
using *fin-T ran-T*
proof (*induction T rule*:*finite-induct*)
case *empty* **thus** *?case* **using** *neg-assoc-empty* **by** *simp*
next
case (*insert x F*)

note r = *indep-var-compose*[*OF indep-var-restrict*[*OF assms*(*3*), **where** $A{=}F$
and $B{=}\{x\}$] -]

have a: $(\lambda\omega.\ \lambda i{\in}\bigcup (p\text{`}F).\ X\ i\ \omega) = (\lambda\omega.\ \lambda i{\in}\bigcup (p\text{`}F).\ \omega\ (g\ i)\ i) \circ (\lambda\omega.\ \lambda i{\in}F.\ \lambda i{\in}p\ i.\ X\ i\ \omega)$
using *insert*(*4*) *g* **by** (*intro restrict-ext ext*) *auto*
have b: $(\lambda\omega.\ \lambda i{\in}p\ x.\ X\ i\ \omega) = (\lambda\omega\ i.\ \omega\ x\ i) \circ (\lambda\omega.\ \lambda i{\in}\{x\}.\ \lambda i{\in}p\ i.\ X\ i\ \omega)$
by (*simp add*:*comp-def restrict-def*)

have c:$(\lambda x.\ x\ (g\ i)\ i) \in$ *borel-measurable* $(Pi_M\ F\ (\lambda j.\ Pi_M\ (p\ j)\ ?B))$ **if** $i \in (\bigcup (p\text{`}F))$ **for** i
proof −
have h: $i \in p\ (g\ i)$ **and** q: $g\ i \in F$ **using** *g that insert*(*4*) **by** *auto*
thus *?thesis*
by (*intro measurable-compose*[*OF measurable-component-singleton*[*OF q*]]) *measurable*
qed

have *finite* $(\bigcup\ (p \text{ ` } insert\ x\ F))$ **using** *assms*(*1*,*2*) *insert*(*4*)
by (*meson Sup-subset-mono image-mono infinite-super*)
moreover have $\bigcup\ (p \text{ ` } F) \cup p\ x = \bigcup\ (p \text{ ` } insert\ x\ F)$ **by** *auto*
moreover have $\bigcup\ (p \text{ ` } F) \cap p\ x = \{\}$
using *assms*(*5*) *insert*(*2*,*4*) **unfolding** *disjoint-family-on-def* **by** *fast*
moreover have
indep-var $(PiM\ (\bigcup (p\text{`}F))\ ?B)\ (\lambda\omega.\ \lambda i{\in}\bigcup (p\text{`}F).\ X\ i\ \omega)\ (PiM\ (p\ x)\ ?B)\ (\lambda\omega.\ \lambda i{\in}p\ x.\ X\ i\ \omega)$
unfolding *a b* **using** *insert*(*1*,*2*,*4*) **by** (*intro r measurable-restrict c*) *simp-all*
moreover have *neg-assoc X* $(\bigcup\ (p \text{ ` } F))$ **using** *insert*(*4*) **by** (*intro insert*(*3*)) *auto*
moreover have *neg-assoc X* $(p\ x)$ **using** *insert*(*4*) **by** (*intro assms*(*4*)) *auto*
ultimately show *?case* **by** (*rule neg-assoc-combine*)
qed
moreover have $(\bigcup (p \text{ ` } T)) = I$ **using** *assms*(*2*) **unfolding** *T-def* **by** *auto*
ultimately show *?thesis* **by** *auto*

qed

Property P5 [13]

```
lemma indep-imp-neg-assoc:
  assumes finite I
  assumes indep-vars (λ-. borel) X I
  shows neg-assoc X I
proof –
  have a:neg-assoc X {i} if i ∈ I for i
    using that assms(2) unfolding indep-vars-def
    by (intro neg-assoc-singleton) auto
  have b: (⋃j∈I. {j}) = I by auto
  have c: indep-vars (λj. Pi_M {j} (λ-. borel)) (λj ω. λi∈{j}. X j ω) I
    by (intro indep-vars-compose2[OF assms(2)]) measurable
  have d: indep-vars (λj. Pi_M {j} (λ-. borel)) (λj ω. λi∈{j}. X i ω) I
    by (intro iffD2[OF indep-vars-cong c] restrict-ext ext) auto
 show ?thesis by (intro neg-assoc-union[OF assms(1) b d a]) (auto simp:disjoint-family-on-def)
qed

end

lemma neg-assoc-map-pmf:
 shows measure-pmf.neg-assoc (map-pmf f p) X I = measure-pmf.neg-assoc p (λi
ω. X i (f ω)) I
    (is ?L ⟷ ?R)
proof –
  let ?d1 = distr (measure-pmf (map-pmf f p)) (Pi_M I (λ-. borel)) (λω. λi∈I. X
i ω)
  let ?d2 = distr (measure-pmf p) (Pi_M I (λ-. borel)) (λω. λi∈I. X i (f ω))

  have emeasure ?d1 A = emeasure ?d2 A if A ∈ sets (Pi_M I (λ-. borel)) for A
  proof –
    have emeasure ?d1 A = emeasure (measure-pmf p) {x. (λi∈I. X i (f x)) ∈ A}
      using that by (subst emeasure-distr) (simp-all add:vimage-def space-PiM)
    also have ... = emeasure ?d2 A
      using that by (subst emeasure-distr) (simp-all add:space-PiM vimage-def)
    finally show ?thesis by simp
  qed

  hence a:?d1 = ?d2 by (intro measure-eqI) auto

  have ?L ⟷ prob-space.neg-assoc ?d1 (λx y. y x) I
    by (subst measure-pmf.neg-assoc-iff-distr) auto
  also have ... ⟷ prob-space.neg-assoc ?d2 (λx y. y x) I
    unfolding a by simp
  also have ... ⟷ ?R
    by (subst measure-pmf.neg-assoc-iff-distr) auto
  finally show ?thesis by simp
qed
```

end

3 Chernoff-Hoeffding Bounds

This section shows that all the well-known Chernoff-Hoeffding bounds hold also for negatively associated random variables. The proofs follow the derivations by Hoeffding [11], as well as, Motwani and Raghavan [16, Ch. 4], with the modification that the crucial steps, where the classic proofs use independence, are replaced with the application of Property P2 for negatively associated RV's.

theory *Negative-Association-Chernoff-Bounds*
  imports
    Negative-Association-Definition
    Concentration-Inequalities.McDiarmid-Inequality
    Weighted-Arithmetic-Geometric-Mean.Weighted-Arithmetic-Geometric-Mean
begin

context *prob-space*
begin

context
  fixes I :: *'i set*
  fixes X :: $'i \Rightarrow 'a \Rightarrow$ *real*
  assumes *na-X*: *neg-assoc X I*
  assumes *fin-I*: *finite I*
begin

private lemma *transfer-to-clamped-vars*:
  assumes $(\forall i{\in}I.\ AE\ \omega\ in\ M.\ X\ i\ \omega \in \{a\ i..b\ i\} \wedge a\ i \leq b\ i)$
  assumes $\mathcal{X}$*-def*: $\mathcal{X} = (\lambda i.\ clamp\ (a\ i)\ (b\ i) \circ X\ i)$
  shows *neg-assoc* $\mathcal{X}$ *I* (**is** *?A*)
    and $\bigwedge i.\ i \in I \Longrightarrow expectation\ (\mathcal{X}\ i) = expectation\ (X\ i)$
    and $\mathcal{P}(\omega\ in\ M.\ (\sum i \in I.\ X\ i\ \omega) \leq\geq_\eta\ c) = \mathcal{P}(\omega\ in\ M.\ (\sum i \in I.\ \mathcal{X}\ i\ \omega) \leq\geq_\eta$
$c)$ (**is** *?C*)
    and $\bigwedge i\ \omega.\ i \in I \Longrightarrow \mathcal{X}\ i\ \omega \in \{a\ i..b\ i\}$
    and $\bigwedge i\ S.\ i \in I \Longrightarrow bounded\ (\mathcal{X}\ i\ `\ S)$
    and $\bigwedge i.\ i \in I \Longrightarrow expectation\ (\mathcal{X}\ i) \in \{a\ i..b\ i\}$
proof −
  note [*measurable*] = *clamp-borel*
  note *rv-X* = *neg-assoc-imp-measurable*[*OF na-X*]

  hence *rv-*$\mathcal{X}$: *random-variable borel* $(\mathcal{X}\ i)$ **if** $i \in I$ **for** i
    unfolding $\mathcal{X}$*-def* **using** *rv-X*[*OF that*] **by** *measurable*

  have *a*:$AE\ x\ in\ M.\ \mathcal{X}\ i\ x = X\ i\ x$ **if** $i \in I$ **for** i
    unfolding $\mathcal{X}$*-def* **using** *clamp-eqI2* **by** (*intro AE-mp*[*OF bspec*[*OF assms*(*1*)

that] *AE-I2*]) *auto*

hence *b*:*AE x in M*. (∀ *i* ∈ *I*. 𝒳 *i x* = *X i x*)
by (*intro AE-finite-allI*[*OF fin-I*]) *simp*

show *?A*
using *a* **by** (*intro neg-assoc-cong*[*OF fin-I rv-*𝒳 *na-X*]) *force+*

show *expectation* (𝒳 *i*) = *expectation* (*X i*) **if** *i* ∈ *I* **for** *i*
by (*intro integral-cong-AE a rv-X rv-*𝒳 *that*)

have {ω ∈ *space M*. (∑ *i*∈*I*. *X i* ω) ≤≥$_\eta$ *c*} ∈ *events* **using** *rv-X* **by** (*cases* η) *simp-all*
moreover have {ω ∈ *space M*. (∑ *i*∈*I*. 𝒳 *i* ω) ≤≥$_\eta$ *c*} ∈ *events* **using** *rv-*𝒳 **by** (*cases* η) *simp-all*
ultimately show *?C* **by** (*intro measure-eq-AE AE-mp*[*OF b AE-I2*]) *auto*

show *c*:𝒳 *i* ω ∈ {*a i*..*b i*} **if** *i* ∈ *I* **for** ω *i*
unfolding 𝒳*-def comp-def* **using** *assms*(*1*) *clamp-range that* **by** *simp*

show *d*:*bounded* (𝒳 *i* ‘ *S*) **if** *i* ∈ *I* **for** *S i*
using *c*[*OF that*] *assms*(*2*) *bounded-clamp* **by** *blast*

show *expectation* (𝒳 *i*) ∈ {*a i*..*b i*} **if** *i* ∈ *I* **for** *i*
unfolding *atLeastAtMost-iff* **using** *c*[*OF that*] *rv-*𝒳[*OF that*]
by (*intro conjI integral-ge-const integral-le-const AE-I2 integrable-bounded d*[*OF that*]) *auto*
qed

lemma *ln-one-plus-x-lower-bound*:
assumes *x* ≥ (*0*::*real*)
shows *2*∗*x*/(*2*+*x*) ≤ *ln* (*1* + *x*)
proof −
define *v* **where** *v x* = *ln*(*1*+*x*) − *2* ∗ *x*/ (*2*+*x*) **for** *x* :: *real*
define *v*′ **where** *v*′ *x* = *1*/(*1*+*x*) − *4*/(*2*+*x*)^*2* **for** *x* :: *real*

have *v-deriv*: (*v has-real-derivative* (*v*′ *x*)) (*at x*) **if** *x* ≥ *0* **for** *x*
using *that* **unfolding** *v-def v*′*-def power2-eq-square* **by** (*auto intro*!:*derivative-eq-intros*)
have *v-deriv-nonneg*: *v*′ *x* ≥ *0* **if** *x* ≥ *0* **for** *x*
using *that* **unfolding** *v*′*-def*
by (*simp add*:*divide-simps power2-eq-square*) (*simp add*:*algebra-simps*)

have *v-mono*: *v x* ≤ *v y* **if** *x* ≤ *y x* ≥ *0* **for** *x y*
using *v-deriv v-deriv-nonneg that order-trans*
by (*intro DERIV-nonneg-imp-nondecreasing*[*OF that*(*1*)]) *blast*

have *0* = *v 0* **unfolding** *v-def* **by** *simp*
also have . . . ≤ *v x* **using** *v-mono assms* **by** *auto*
finally have *v x* ≥ *0* **by** *simp*

thus *?thesis* **unfolding** *v-def* **by** *simp*
qed

Based on Theorem 4.1 by Motwani and Raghavan [16].

theorem *multiplicative-chernoff-bound-upper*:
assumes $\delta > 0$
assumes $\bigwedge i.\ i \in I \Longrightarrow AE\ \omega\ in\ M.\ X\ i\ \omega \in \{0..1\}$
defines $\mu \equiv (\sum i \in I.\ expectation\ (X\ i))$
shows $\mathcal{P}(\omega\ in\ M.\ (\sum i \in I.\ X\ i\ \omega) \geq (1{+}\delta) * \mu) \leq (exp\ \delta/((1{+}\delta)\ powr\ (1{+}\delta)))\ powr\ \mu$ (**is** $?L \leq ?R$)
and $\mathcal{P}(\omega\ in\ M.\ (\sum i \in I.\ X\ i\ \omega) \geq (1{+}\delta) * \mu) \leq exp\ (-(\delta\hat{\ }2) * \mu\ /\ (2{+}\delta))$ (**is** - $\leq$ *?R1*)
proof −
define $\mathcal{X}$ **where** $\mathcal{X} = (\lambda i.\ clamp\ 0\ 1 \circ X\ i)$
have *transfer-to-clamped-vars-assms*: $(\forall i{\in}I.\ AE\ \omega\ in\ M.\ X\ i\ \omega \in \{0\ ..\ 1\} \wedge 0 \leq (1{::}real))$
using *assms*(2) **by** *auto*
note *ttcv* = *transfer-to-clamped-vars*[*OF transfer-to-clamped-vars-assms* $\mathcal{X}$*-def*]
note [*measurable*] = *neg-assoc-imp-measurable*[*OF ttcv*(1)]

define t **where** $t = ln\ (1{+}\delta)$
have *t-gt-0*: $t > 0$ **using** *assms*(1) **unfolding** *t-def* **by** *simp*

let $?h = (\lambda x.\ 1 + (exp\ t - 1) * x)$

note *bounded′* = *integrable-bounded bounded-prod bounded-vec-mult-comp bounded-intros ttcv*(5)

have *int*: *integrable M* ($\mathcal{X}$ *i*) **if** $i \in I$ **for** *i*
using *that* **by** (*intro bounded′*) *simp-all*

have $2{*}\delta \leq (2{+}\delta){*}\ ln\ (1 + \delta)$
using *assms*(1) *ln-one-plus-x-lower-bound*[*OF less-imp-le*[*OF assms*(1)]] **by** (*simp add:field-simps*)
hence $(1{+}\delta){*}(2{*}\delta) \leq (1 + \delta)\ {*}(2{+}\delta){*}\ ln\ (1 + \delta)$ **using** *assms*(1) **by** *simp*
hence $a{:}(\delta - (1 + \delta) * ln\ (1 + \delta)) \leq -\ (\delta\hat{\ }2)/(2{+}\delta)$
using *assms*(1) **by** (*simp add:field-simps power2-eq-square*)

have *μ-ge-0*: $\mu \geq 0$ **unfolding** *μ-def* **using** *ttcv*(2,6) **by** (*intro sum-nonneg*) *auto*

note $\mathcal{X}$*-prod-mono* = *has-int-thatD*(2)[*OF neg-assoc-imp-prod-mono*[*OF fin-I ttcv*(1), **where** η=*Fwd*]]

have $?L = \mathcal{P}(\omega\ in\ M.\ (\sum i \in I.\ \mathcal{X}\ i\ \omega) \geq (1{+}\delta) * \mu)$ **using** *ttcv*(3)[**where** η=*Rev*] **by** *simp*
also have $\ldots = \mathcal{P}(\omega\ in\ M.\ (\prod i \in I.\ exp\ (t * \mathcal{X}\ i\ \omega)) \geq exp\ (t * (1{+}\delta) * \mu))$
using *t-gt-0* **by** (*simp add*: *sum-distrib-left*[*symmetric*] *exp-sum*[*OF fin-I*,*symmetric*])
also have $\ldots \leq expectation\ (\lambda\omega.\ (\prod i \in I.\ exp\ (t * \mathcal{X}\ i\ \omega)))\ /\ exp\ (t{*}(1{+}\delta){*}\mu)$

```
    by (intro integral-Markov-inequality-measure[where A={}] bounded' AE-I2
prod-nonneg fin-I)
      simp-all
  also have ... ≤ (∏ i ∈ I. expectation (λω. exp (t*𝒳 i ω))) / exp (t*(1+δ)*μ)
    using t-gt-0 by (intro divide-right-mono 𝒳-prod-mono bounded' image-subsetI
monotoneI) simp-all
  also have ... = (∏ i ∈ I. expectation (λω. exp ((1−𝒳 i ω) *R 0+ 𝒳 i ω *R
t))) / exp (t*(1+δ)*μ)
    by (simp add:ac-simps)
  also have ... ≤ (∏ i ∈ I. expectation (λω. (1−𝒳 i ω) * exp 0 + 𝒳 i ω * exp
t)) / exp (t*(1+δ)*μ)
    using ttcv(4)
      by (intro divide-right-mono prod-mono integral-mono conjI bounded' con-
vex-onD[OF exp-convex])
      simp-all
  also have ... = (∏ i ∈ I. ?h (expectation (𝒳 i))) / exp (t*(1+δ)*μ)
    using int by (simp add:algebra-simps prob-space cong:prod.cong)
  also have ... ≤ (∏ i ∈ I. exp((exp t−1)* expectation (𝒳 i))) / exp (t*(1+δ)*μ)
    using t-gt-0 ttcv(4)
   by (intro divide-right-mono prod-mono exp-ge-add-one-self conjI add-nonneg-nonneg
        mult-nonneg-nonneg) simp-all
  also have ... = exp ((exp t−1)* μ) / exp (t*(1+δ)*μ)
   unfolding exp-sum[OF fin-I, symmetric] μ-def by (simp add:ttcv(2) sum-distrib-left)
  also have ... = exp (δ * μ) / exp ( ln (1+δ)*(1+δ) * μ)
     using assms(1) unfolding μ-def t-def by (simp add:sum-distrib-left)
  also have ... = exp δ powr μ / exp ( ln(1+δ)*(1+δ)) powr μ
    unfolding powr-def by (simp add:ac-simps)
 also have ... = ?R using assms(1) by (subst powr-divide) (simp-all add:powr-def)
  finally show ?L ≤ ?R by simp
  also have ... =  exp (μ * ln (exp δ / exp ((1 + δ) * ln (1 + δ))))
    using assms unfolding powr-def by simp
  also have ... = exp (μ * (δ − (1 + δ) * ln (1 + δ))) by (subst ln-div) simp-all
  also have ... ≤ exp (μ * (−(δ^2)/(2+δ)))
    by (intro iffD2[OF exp-le-cancel-iff] mult-left-mono a μ-ge-0)
  also have ... = ?R1 by simp
  finally show ?L ≤ ?R1 by simp
qed

lemma ln-one-minus-x-lower-bound:
  assumes x ∈ {(0::real)..<1}
  shows (x^2/2−x)/(1−x) ≤ ln (1 − x)
proof −
  define v where v x = ln(1−x) − (x^2/2−x) / (1−x)  for x :: real
  define v' where v' x = −1/(1−x) − (−(x^2)/2+x−1)/((1−x)^2) for x :: real

  have v-deriv: (v has-real-derivative (v' x)) (at x) if x ∈ {0..<1} for x
    using that unfolding v-def v'-def power2-eq-square
    by (auto intro!:derivative-eq-intros simp:algebra-simps)
  have v-deriv-nonneg: v' x ≥ 0 if x ≥ 0 for x
```

using *that* **unfolding** *v′-def* **by** (*simp add*:*divide-simps power2-eq-square*)

have *v-mono*: $v\ x \le v\ y$ **if** $x \le y$ $x \ge 0$ $y < 1$ **for** *x y*
using *v-deriv v-deriv-nonneg that* **unfolding** *atLeastLessThan-iff*
by (*intro DERIV-nonneg-imp-nondecreasing*[*OF that*(*1*)])
(*metis* (*mono-tags*, *opaque-lifting*) *Ico-eq-Ico ivl-subset linorder-not-le order-less-irrefl*)

have $0 = v\ 0$ **unfolding** *v-def* **by** *simp*
also have $\ldots \le v\ x$ **using** *v-mono assms* **by** *auto*
finally have $v\ x \ge 0$ **by** *simp*
thus *?thesis* **unfolding** *v-def* **by** *simp*
qed

Based on Theorem 4.2 by Motwani and Raghavan [16].

theorem *multiplicative-chernoff-bound-lower*:
assumes $\delta \in \{0<..<1\}$
assumes $\bigwedge i.\ i \in I \Longrightarrow AE\ \omega\ in\ M.\ X\ i\ \omega \in \{0..1\}$
defines $\mu \equiv (\sum i \in I.\ expectation\ (X\ i))$
shows $\mathcal{P}(\omega\ in\ M.\ (\sum i \in I.\ X\ i\ \omega) \le (1{-}\delta){*}\mu) \le (exp\ ({-}\delta)/(1{-}\delta)\ powr\ (1{-}\delta))\ powr\ \mu$ (**is** $?L \le ?R$)
and $\mathcal{P}(\omega\ in\ M.\ (\sum i \in I.\ X\ i\ \omega) \le (1{-}\delta){*}\mu) \le (exp\ ({-}(\delta\hat{\ }2){*}\mu/2))$ (**is** - $\le$ $?R1$)
proof −
define $\mathcal{X}$ **where** $\mathcal{X} = (\lambda i.\ clamp\ 0\ 1 \circ X\ i)$
have *transfer-to-clamped-vars-assms*: $(\forall i{\in}I.\ AE\ \omega\ in\ M.\ X\ i\ \omega \in \{0\ ..\ 1\} \wedge 0 \le (1{::}real))$
using *assms*(*2*) **by** *auto*
note *ttcv* = *transfer-to-clamped-vars*[*OF transfer-to-clamped-vars-assms* $\mathcal{X}$*-def*]
note [*measurable*] = *neg-assoc-imp-measurable*[*OF ttcv*(*1*)]

define *t* **where** $t = ln\ (1{-}\delta)$
have *t-lt-0*: $t < 0$ **using** *assms*(*1*) **unfolding** *t-def* **by** *simp*

let $?h = (\lambda x.\ 1 + (exp\ t - 1) * x)$

note *bounded′* = *integrable-bounded bounded-prod bounded-vec-mult-comp bounded-intros ttcv*(*5*)

have *μ-ge-0*: $\mu \ge 0$ **unfolding** *μ-def* **using** *ttcv*(*2*,*6*) **by** (*intro sum-nonneg*) *auto*

have *int*: *integrable M* ($\mathcal{X}$ *i*) **if** $i \in I$ **for** *i*
using *that* **by** (*intro bounded′*) *simp-all*

note $\mathcal{X}$*-prod-mono* = *has-int-thatD*(*2*)[*OF neg-assoc-imp-prod-mono*[*OF fin-I ttcv*(*1*), **where** *η*=*Rev*]]

have *0*: $0 \le 1 + (exp\ t - 1) * expectation\ (\mathcal{X}\ i)$ **if** $i \in I$ **for** *i*

proof −
have *0* ≤ *1* + (*exp t* − *1*) ∗ *1* **by** *simp*
also have . . . ≤ *1* + (*exp t* − *1*) ∗ *expectation* (𝒳 *i*)
using *t-lt-0 ttcv*(*6*)[*OF that*] **by** (*intro add-mono mult-left-mono-neg*) *auto*
finally show *?thesis* **by** *simp*
qed

have δ ∈ {*0*..<*1*} **using** *assms*(*1*) **by** *simp*
from *ln-one-minus-x-lower-bound*[*OF this*]
have δ² / *2* − δ ≤ (*1* − δ) ∗ *ln* (*1* − δ) **using** *assms*(*1*) **by** (*simp add*:*field-simps*)
hence *1*: − δ − (*1* − δ) ∗ *ln* (*1* − δ) ≤ − δ² / *2* **by** (*simp add*:*algebra-simps*)

have *?L* = 𝒫(ω *in M*. (∑*i* ∈ *I*. 𝒳 *i* ω) ≤ (*1*−δ) ∗ μ) **using** *ttcv*(*3*)[**where** η=*Fwd*] **by** *simp*
also have . . . = 𝒫(ω *in M*. (∏*i* ∈ *I*. *exp* (*t* ∗ 𝒳 *i* ω)) ≥ *exp* (*t* ∗ (*1*−δ) ∗ μ))
using *t-lt-0* **by** (*simp add*: *sum-distrib-left*[*symmetric*] *exp-sum*[*OF fin-I*,*symmetric*])
also have . . . ≤ *expectation* (λω. (∏*i* ∈ *I*. *exp* (*t* ∗ 𝒳 *i* ω))) / *exp* (*t*∗(*1*−δ)∗μ)
by (*intro integral-Markov-inequality-measure*[**where** *A*={}] *bounded′ AE-I2 prod-nonneg fin-I*)
simp-all
also have . . . ≤ (∏*i* ∈ *I*. *expectation* (λω. *exp* (*t*∗𝒳 *i* ω))) / *exp* (*t*∗(*1*−δ)∗μ)
using *t-lt-0* **by** (*intro divide-right-mono* 𝒳*-prod-mono bounded′ image-subsetI monotoneI*) *simp-all*
also have . . . = (∏*i* ∈ *I*. *expectation* (λω. *exp* ((*1*−𝒳 *i* ω) ∗$_R$ *0*+ 𝒳 *i* ω ∗$_R$ *t*))) / *exp* (*t*∗(*1*−δ)∗μ)
by (*simp add*:*ac-simps*)
also have . . . ≤ (∏*i* ∈ *I*. *expectation* (λω. (*1*−𝒳 *i* ω) ∗ *exp 0* + 𝒳 *i* ω ∗ *exp t*)) / *exp* (*t*∗(*1*−δ)∗μ)
using *ttcv*(*4*)
by (*intro divide-right-mono prod-mono integral-mono conjI bounded′ convex-onD*[*OF exp-convex*])
simp-all
also have . . . = (∏*i* ∈ *I*. *?h* (*expectation* (𝒳 *i*))) / *exp* (*t*∗(*1*−δ)∗μ)
using *int* **by** (*simp add*:*algebra-simps prob-space cong*:*prod.cong*)
also have . . . ≤ (∏*i* ∈ *I*. *exp*((*exp t*−*1*)∗ *expectation* (𝒳 *i*))) / *exp* (*t*∗(*1*−δ)∗μ)
using *0* **by** (*intro divide-right-mono prod-mono exp-ge-add-one-self conjI*) *simp-all*
also have . . . = *exp* ((*exp t*−*1*)∗ μ) / *exp* (*t*∗(*1*−δ)∗μ)
unfolding *exp-sum*[*OF fin-I*, *symmetric*] μ*-def* **by** (*simp add*:*ttcv*(*2*) *sum-distrib-left*)
also have . . . = *exp* ((−δ) ∗ μ) / *exp* (*ln* (*1*−δ)∗(*1*−δ) ∗ μ)
using *assms*(*1*) **unfolding** μ*-def t-def* **by** (*simp add*:*sum-distrib-left*)
also have . . . = *exp* (−δ) *powr* μ / *exp* (*ln*(*1*−δ)∗(*1*−δ)) *powr* μ
unfolding *powr-def* **by** (*simp add*:*ac-simps*)
also have . . . = *?R* **using** *assms*(*1*) **by** (*subst powr-divide*) (*simp-all add*:*powr-def*)
finally show *?L* ≤ *?R* **by** *simp*
also have . . . = *exp* (μ ∗ (− δ − (*1* − δ) ∗ *ln* (*1* − δ)))
using *assms*(*1*) **unfolding** *powr-def* **by** (*simp add*:*ln-div*)
also have . . . ≤ *exp* (μ ∗ (−(δ^*2*) / *2*))
by (*intro iffD2*[*OF exp-le-cancel-iff*] *mult-left-mono* μ*-ge-0 1*)

finally show *?L* ≤ *?R1* **by** (*simp add*:*ac-simps*)
qed

theorem *multiplicative-chernoff-bound-two-sided*:
assumes *δ* ∈ {*0*<..<*1*}
assumes ⋀*i*. *i* ∈ *I* ⟹ *AE ω in M*. *X i ω* ∈ {*0*..*1*}
defines *μ* ≡ (∑ *i* ∈ *I*. *expectation* (*X i*))
shows 𝒫(*ω in M*. |(∑ *i* ∈ *I*. *X i ω*) − *μ*| ≥ *δ*∗*μ*) ≤ *2*∗(*exp* (−(*δ*^*2*)∗*μ*/*3*)) (**is** *?L* ≤ *?R*)
proof −
define 𝒳 **where** 𝒳 = (*λi*. *clamp 0 1* ∘ *X i*)
have *transfer-to-clamped-vars-assms*: (∀ *i*∈*I*. *AE ω in M*. *X i ω* ∈ {*0* .. *1*} ∧ *0* ≤ (*1*::*real*))
using *assms*(*2*) **by** *auto*
note *ttcv* = *transfer-to-clamped-vars*[*OF transfer-to-clamped-vars-assms* 𝒳*-def*]

have *μ-ge-0*: *μ* ≥ *0* **unfolding** *μ-def* **using** *ttcv*(*2*,*6*) **by** (*intro sum-nonneg*) *auto*

note [*measurable*] = *neg-assoc-imp-measurable*[*OF na-X*]

have *?L* = 𝒫(*ω in M*. (∑ *i*∈*I*. *X i ω*) ≥ (*1*+*δ*)∗*μ* ∨ (∑ *i*∈*I*. *X i ω*) ≤ (*1*−*δ*)∗*μ*) **unfolding** *abs-real-def*
by (*intro arg-cong*[**where** *f*=*prob*] *Collect-cong*) (*auto simp*:*algebra-simps*)
also have ... =*measure M*({*ω*∈*space M*.(∑ *i*∈*I*. *X i ω*)≥(*1*+*δ*)∗*μ*}∪{*ω*∈*space M*. (∑ *i*∈*I*. *X i ω*)≤(*1*−*δ*)∗*μ*})
by (*intro arg-cong*[**where** *f*=*prob*]) *auto*
also have ... ≤ 𝒫(*ω in M*. (∑ *i*∈*I*. *X i ω*) ≥ (*1*+*δ*)∗*μ*) + 𝒫(*ω in M*.(∑ *i*∈*I*. *X i ω*) ≤ (*1*−*δ*)∗*μ*)
by (*intro measure-Un-le*) *measurable*
also have ... ≤ *exp* (−(*δ*^*2*)∗*μ*/(*2*+*δ*)) + *exp* (−(*δ*^*2*)∗*μ*/*2*)
unfolding *μ-def* **using** *assms*(*1*,*2*)
by (*intro multiplicative-chernoff-bound-lower multiplicative-chernoff-bound-upper add-mono*) *auto*
also have ... ≤ *exp* (−(*δ*^*2*)∗*μ*/*3*) + *exp* (−(*δ*^*2*)∗*μ*/*3*)
using *assms*(*1*) *μ-ge-0* **by** (*intro iffD2*[*OF exp-le-cancel-iff*] *add-mono divide-left-mono-neg*) *auto*
also have ... = *?R* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *additive-chernoff-bound-upper-aux*:
assumes ⋀*i*. *i*∈*I* ⟹ *AE ω in M*. *X i ω* ∈ {*0*..*1*} *I* ≠ {}
defines *μ* ≡ (∑ *i*∈*I*. *expectation* (*X i*)) / *real* (*card I*)
assumes *δ* ∈ {*0*<..<*1*−*μ*} *μ* ∈ {*0*<..<*1*}
shows 𝒫(*ω in M*. (∑ *i*∈*I*. *X i ω*) ≥ (*μ*+*δ*)∗*real* (*card I*)) ≤ *exp* (−*real* (*card I*) ∗ *KL-div* (*μ*+*δ*) *μ*)
(**is** *?L* ≤ *?R*)
proof −

define $\mathcal{X}$ **where** $\mathcal{X} = (\lambda i.\ clamp\ 0\ 1 \circ X\ i)$
have *transfer-to-clamped-vars-assms*: $(\forall\ i{\in}I.\ AE\ \omega\ in\ M.\ X\ i\ \omega \in \{0..1\} \wedge 0 \leq (1{::}real))$
using *assms*(*1*) **by** *auto*
note *ttcv* = *transfer-to-clamped-vars*[*OF transfer-to-clamped-vars-assms* $\mathcal{X}$*-def*]
note [*measurable*] = *neg-assoc-imp-measurable*[*OF ttcv*(*1*)]

define t :: *real* **where** $t = ln\ ((\mu{+}\delta)/\mu) - ln\ ((1{-}\mu{-}\delta)/(1{-}\mu))$
let $?h = \lambda x.\ 1 + (exp\ t - 1) * x$
let $?n = real\ (card\ I)$

have *n-gt-0*: $?n > 0$ **using** *assms*(*2*) *fin-I* **by** *auto*

have *a*: $(1 - \mu - \delta) > 0\ \mu > 0\ 1 - \mu > 0\ \mu + \delta > 0$
using *assms*(*4*,*5*) **by** *auto*

have $ln\ ((1 - \mu - \delta)\ /\ (1 - \mu)) < 0$ **using** *a assms*(*4*) **by** (*intro ln-less-zero*) *auto*
moreover have $ln\ ((\mu + \delta)\ /\ \mu) > 0$ **using** *a assms*(*4*) **by** (*intro ln-gt-zero*) *auto*
ultimately have *t-gt-0*: $t > 0$ **unfolding** *t-def* **by** *simp*

note *bounded′* = *integrable-bounded bounded-prod bounded-vec-mult-comp bounded-intros ttcv*(*5*)

note $\mathcal{X}$*-prod-mono* = *has-int-thatD*(*2*)[*OF neg-assoc-imp-prod-mono*[*OF fin-I ttcv*(*1*), **where** η=*Fwd*]]

have *int*: *integrable M* ($\mathcal{X}$ *i*) **if** $i \in I$ **for** *i*
using *that* **by** (*intro bounded′*) *simp-all*

have *0*: $0 \leq 1 + (exp\ t - 1) * expectation\ (\mathcal{X}\ i)$ **if** $i \in I$ **for** *i*
using *t-gt-0 ttcv*(*6*)[*OF that*] **by** (*intro add-nonneg-nonneg mult-nonneg-nonneg*) *auto*

have $1 + (exp\ t - 1) * \mu = 1 + ((\mu + \delta) * (1 - \mu)\ /\ (\mu * (1 - \mu - \delta)) - 1) * \mu$
using *a* **unfolding** *t-def exp-diff* **by** *simp*
also have $\ldots = 1 + (\delta\ /\ (\mu * (1 - \mu - \delta))) * \mu$
using *a* **by** (*subst divide-diff-eq-iff*) (*simp*, *simp add*:*algebra-simps*)
also have $\ldots = (1{-}\mu{-}\delta)/(1{-}\mu{-}\delta) + (\delta\ /\ (1{-}\mu{-}\delta))$ **using** *a* **by** *simp*
also have $\ldots = (1{-}\mu)\ /\ (1{-}\mu{-}\delta)$
unfolding *add-divide-distrib*[*symmetric*] **by** (*simp add*:*algebra-simps*)
also have $\ldots = inverse\ ((1{-}\mu{-}\delta)\ /\ (1{-}\mu))$ **using** *a* **by** *simp*
also have $\ldots = exp\ (ln\ (inverse\ ((1{-}\mu{-}\delta)\ /\ (1{-}\mu))))$ **using** *a* **by** *simp*
also have $\ldots = exp\ (-\ ln((1{-}\mu{-}\delta)\ /\ (1{-}\mu)))$ **using** *a* **by** (*subst ln-inverse*) (*simp-all*)
finally have *1*: $1 + (exp\ t - 1) * \mu = exp\ (-\ ln((1{-}\mu{-}\delta)\ /\ (1{-}\mu)))$ **by** *simp*

have *?L* = $\mathcal{P}$(*ω in M*. ($\sum$ *i* ∈ *I*. $\mathcal{X}$ *i ω*) ≥ (*μ*+*δ*) ∗ *?n*) **using** *ttcv*(*3*)[**where** *η*=*Rev*] **by** *simp*
also have … = $\mathcal{P}$(*ω in M*. ($\prod$ *i* ∈ *I*. *exp* (*t* ∗ $\mathcal{X}$ *i ω*)) ≥ *exp* (*t* ∗ (*μ*+*δ*) ∗ *?n*))
using *t-gt-0* **by** (*simp add*: *sum-distrib-left*[*symmetric*] *exp-sum*[*OF fin-I*,*symmetric*])
also have … ≤ *expectation* (*λω*. ($\prod$ *i* ∈ *I*. *exp* (*t* ∗ $\mathcal{X}$ *i ω*))) / *exp* (*t* ∗ (*μ*+*δ*) ∗ *?n*)
by (*intro integral-Markov-inequality-measure*[**where** *A*={}] *bounded′ AE-I2 prod-nonneg fin-I*)
simp-all
also have … ≤ ($\prod$ *i* ∈ *I*. *expectation* (*λω*. *exp* (*t*∗$\mathcal{X}$ *i ω*))) / *exp* (*t* ∗ (*μ*+*δ*) ∗ *?n*)
using *t-gt-0* **by** (*intro divide-right-mono* $\mathcal{X}$*-prod-mono bounded′ image-subsetI monotoneI*) *simp-all*
also have … = ($\prod$ *i* ∈ *I*. *expectation* (*λω*. *exp* ((*1*−$\mathcal{X}$ *i ω*) $*_R$ *0* + $\mathcal{X}$ *i ω* $*_R$ *t*))) / *exp* (*t*∗(*μ*+*δ*)∗*?n*)
by (*simp add*:*ac-simps*)
also have … ≤ ($\prod$ *i*∈*I*. *expectation* (*λω*. (*1*−$\mathcal{X}$ *i ω*)∗*exp 0* + $\mathcal{X}$ *i ω* ∗ *exp t*)) / *exp* (*t* ∗ (*μ*+*δ*) ∗ *?n*)
using *ttcv*(*4*)
by (*intro divide-right-mono prod-mono integral-mono conjI bounded′ convex-onD*[*OF exp-convex*])
simp-all
also have … = ($\prod$ *i* ∈ *I*. *?h* (*expectation* ($\mathcal{X}$ *i*))) / *exp* (*t* ∗ (*μ*+*δ*) ∗ *?n*)
using *int* **by** (*simp add*:*algebra-simps prob-space cong*:*prod.cong*)
also have … = (*root* (*card I*) ($\prod$ *i*∈*I*. *1*+(*exp t*−*1*)∗*expectation* ($\mathcal{X}$ *i*)))^(*card I*) / *exp* (*t*∗(*μ*+*δ*)∗*?n*)
using *n-gt-0*
by (*intro arg-cong2*[**where** *f*=(/)] *real-root-pow-pos2*[*symmetric*] *prod-nonneg refl 0*) *auto*
also have … ≤ (($\sum$ *i* ∈ *I*. *1* + (*exp t* − *1*) ∗ *expectation* ($\mathcal{X}$ *i*)) / *?n*)^(*card I*) / *exp* (*t*∗(*μ*+*δ*)∗*?n*)
by (*intro divide-right-mono power-mono arithmetic-geometric-mean*[*OF fin-I*] *real-root-ge-zero*
prod-nonneg 0) *simp-all*
also have … ≤ (($\sum$ *i* ∈ *I*. *1* + (*exp t* − *1*) ∗ *expectation* ($\mathcal{X}$ *i*)) / *?n*) *powr ?n* / *exp* (*t*∗(*μ*+*δ*)∗*?n*)
using *n-gt-0 0* **by** (*subst powr-realpow′*) (*auto intro*!:*sum-nonneg divide-nonneg-pos 0*)
also have … ≤ (($\sum$ *i* ∈ *I*. *1* + (*exp t* − *1*) ∗ *expectation* (*X i*)) / *?n*) *powr ?n* / *exp* (*t*∗(*μ*+*δ*)∗*?n*)
using *ttcv*(*2*) **by** (*simp cong*:*sum.cong*)
also have … = (*1* + (*exp t* − *1*) ∗ *μ*) *powr ?n* / *exp* (*t*∗(*μ*+*δ*)∗*?n*)
using *n-gt-0* **unfolding** *μ-def sum.distrib sum-distrib-left*[*symmetric*] **by** (*simp add*:*divide-simps*)
also have … = (*1* + (*exp t* − *1*) ∗ *μ*) *powr ?n* / *exp* (*t*∗(*μ*+*δ*)) *powr ?n*
unfolding *powr-def* **by** *simp*
also have … = ((*1* + (*exp t* − *1*) ∗ *μ*)/*exp*(*t*∗(*μ*+*δ*))) *powr ?n*
using *a t-gt-0* **by** (*auto intro*: *powr-divide*[*symmetric*] *add-nonneg-nonneg mult-nonneg-nonneg*)

also have ... = $(exp\ (-\ ln((1{-}\mu{-}\delta)\ /\ (1{-}\mu)))\ *\ exp(\ -(t\ *\ (\mu{+}\delta))))\ powr\ ?n$
unfolding *1 exp-minus inverse-eq-divide* **by** *simp*
also have ... = $exp\ (\ -ln((1-\mu{-}\delta)/(1-\mu)){-}\ t\ *\ (\mu{+}\delta))\ powr\ ?n$
unfolding *exp-add*[*symmetric*] **by** *simp*
also have ... = $exp\ (\ -ln((1-\mu{-}\delta)/(1-\mu)){-}\ (ln\ ((\mu{+}\delta)/\mu)\ -\ ln\ ((1{-}\mu{-}\delta)/(1{-}\mu))){*}(\mu{+}\delta))$
$powr\ ?n$
using *a* **unfolding** *t-def* **by** (*simp add*:*divide-simps*)
also have ... = $exp(\ -KL\text{-}div\ (\mu{+}\delta)\ \mu)\ powr\ ?n$
using *a* **by** (*subst KL-div-eq*) (*simp-all add*:*field-simps*)
also have ... = *?R* **unfolding** *powr-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *additive-chernoff-bound-upper-aux-2*:
assumes $\bigwedge i.\ i{\in}I \Longrightarrow AE\ \omega\ in\ M.\ X\ i\ \omega \in \{0..1\}\ I \neq \{\}$
defines $\mu \equiv (\sum i{\in}I.\ expectation\ (X\ i))\ /\ real\ (card\ I)$
assumes $\mu \in \{0{<}..{<}1\}$
shows $\mathcal{P}(\omega\ in\ M.\ (\sum i{\in}I.\ X\ i\ \omega) \geq real\ (card\ I)) \leq exp\ (-real\ (card\ I)\ *\ KL\text{-}div$
$1\ \mu)$
(**is** $?L \leq ?R$)
proof −
define $\mathcal{X}$ **where** $\mathcal{X} = (\lambda i.\ clamp\ 0\ 1 \circ X\ i)$
have *transfer-to-clamped-vars-assms*: $(\forall i{\in}I.\ AE\ \omega\ in\ M.\ X\ i\ \omega \in \{0..1\} \wedge 0 \leq$
$(1{::}real))$
using *assms*(*1*) **by** *auto*
note *ttcv* = *transfer-to-clamped-vars*[*OF transfer-to-clamped-vars-assms* $\mathcal{X}$*-def*]
note [*measurable*] = *neg-assoc-imp-measurable*[*OF ttcv*(*1*)]

let $?n = real\ (card\ I)$

have *n-gt-0*: $?n > 0$ **using** *assms*(*2*) *fin-I* **by** *auto*

note *bounded'* = *integrable-bounded bounded-prod bounded-vec-mult-comp bounded-intros*
ttcv(*5*)
bounded-max

note $\mathcal{X}$*-prod-mono* = *has-int-thatD*(*2*)[*OF neg-assoc-imp-prod-mono*[*OF fin-I*
ttcv(*1*), **where** η=*Fwd*]]

have *a2*:$(\prod i \in I.\ max\ 0\ (\mathcal{X}\ i\ \omega)) \geq 1$ **if** $(\sum i \in I.\ \mathcal{X}\ i\ \omega) \geq ?n$ **for** ω
proof −
have $(\sum i \in I.\ 1 - \mathcal{X}\ i\ \omega) \leq 0$ **using** *that* **by** (*simp add*:*sum-subtractf*)
moreover have $(\sum i \in I.\ 1 - \mathcal{X}\ i\ \omega) \geq 0$ **using** *ttcv*(*4*) **by** (*intro sum-nonneg*)
simp
ultimately have $(\sum i \in I.\ 1 - \mathcal{X}\ i\ \omega) = 0$ **by** *simp*
with *iffD1*[*OF sum-nonneg-eq-0-iff*[*OF fin-I*] *this*]
have $\forall i \in I.\ 1 - \mathcal{X}\ i\ \omega = 0$ **using** *ttcv*(*4*) **by** *simp*
hence $\mathcal{X}\ i\ \omega = 1$ **if** $i \in I$ **for** *i* **using** *that* **by** *auto*
thus *?thesis* **by** (*intro prod-ge-1*) *fastforce*

qed

have *?L* = $\mathcal{P}$(*ω in M*. ($\sum$ *i* ∈ *I*. $\mathcal{X}$ *i ω*) ≥ *?n*) **using** *ttcv*(*3*)[**where** *η*=*Rev*] **by** *simp*
also have ... ≤ $\mathcal{P}$(*ω in M*. ($\prod$ *i* ∈ *I*. *max 0* ($\mathcal{X}$ *i ω*)) ≥ *1*)
using *a2* **by** (*intro finite-measure-mono*) *auto*
also have ... ≤ *expectation* (*λω*. ($\prod$ *i* ∈ *I*. *max 0* ($\mathcal{X}$ *i ω*))) / *1*
by (*intro integral-Markov-inequality-measure*[**where** *A*={}] *bounded′ AE-I2 prod-nonneg fin-I*)
auto
also have ... ≤ ($\prod$ *i* ∈ *I*. *expectation* (*λω*. *max 0* ($\mathcal{X}$ *i ω*))) / *1*
by (*intro divide-right-mono* $\mathcal{X}$*-prod-mono bounded′ image-subsetI monotoneI*) *simp-all*
also have ... ≤ ($\prod$ *i* ∈ *I*. *expectation* ($\mathcal{X}$ *i*)) **using** *ttcv*(*4*) **by** *simp*
also have ... = (*root* (*card I*) ($\prod$ *i*∈*I*. *expectation* ($\mathcal{X}$ *i*)))^(*card I*)
using *n-gt-0 ttcv*(*6*) **by** (*intro real-root-pow-pos2*[*symmetric*] *prod-nonneg refl*) *auto*
also have ... ≤ (($\sum$ *i* ∈ *I*. *expectation* ($\mathcal{X}$ *i*)) / *?n*)^(*card I*)
using *ttcv*(*6*) **by** (*intro power-mono arithmetic-geometric-mean*[*OF fin-I*] *real-root-ge-zero*
prod-nonneg) *auto*
also have ... ≤ (($\sum$ *i* ∈ *I*. *expectation* ($\mathcal{X}$ *i*)) / *?n*) *powr ?n*
using *n-gt-0 ttcv*(*6*) **by** (*subst powr-realpow′*) (*auto intro*!:*sum-nonneg divide-nonneg-pos*)
also have ... ≤ *μ powr ?n* **using** *ttcv*(*2*) **unfolding** *μ-def* **by** *simp*
also have ... = *?R* **using** *assms*(*4*) **unfolding** *powr-def* **by** (*subst KL-div-eq*) (*auto simp*:*ln-div*)
finally show *?thesis* **by** *simp*
qed

Based on Theorem 1 by Hoeffding [11].

lemma *additive-chernoff-bound-upper*:
assumes $\bigwedge$*i*. *i*∈*I* ⟹ *AE ω in M*. *X i ω* ∈ {*0..1*} *I* ≠ {}
defines *μ* ≡ ($\sum$ *i*∈*I*. *expectation* (*X i*)) / *real* (*card I*)
assumes *δ* ∈ {*0..1*−*μ*} *μ* ∈ {*0*<..<*1*}
shows $\mathcal{P}$(*ω in M*. ($\sum$ *i*∈*I*. *X i ω*) ≥ (*μ*+*δ*)∗*real* (*card I*)) ≤ *exp* (−*real* (*card I*) ∗ *KL-div* (*μ*+*δ*) *μ*)
(**is** *?L* ≤ *?R*)
proof −
note [*measurable*] = *neg-assoc-imp-measurable*[*OF na-X*]

let *?n* = *real* (*card I*)
have *n-gt-0*: *?n* > *0* **using** *assms fin-I* **by** *auto*

note *X-prod-mono* = *has-int-thatD*(*2*)[*OF neg-assoc-imp-prod-mono*[*OF fin-I na-X*, **where** *η*=*Fwd*]]

have *b*:*AE x in M*. (∀ *i* ∈ *I*. *X i x* ∈ {*0..1*})
using *assms*(*1*) **by** (*intro AE-finite-allI*[*OF fin-I*]) *simp*

```
  hence c:AE x in M. (∑ i∈I. 1 − X i x) ≥ 0
    by (intro AE-mp[OF b AE-I2] impI sum-nonneg) auto

   consider (i) δ=0 | (ii) δ ∈ {0<..<1−μ} | (iii) 1−μ=δ using assms(4) by
fastforce
  thus ?thesis
  proof (cases)
    case i
    hence KL-div (μ+δ) μ = 0 using assms(4,5) by (subst KL-div-eq) auto
    thus ?thesis by simp
  next
    case ii
   thus ?thesis unfolding μ-def using assms by (intro additive-chernoff-bound-upper-aux)
auto
  next
    case iii
    hence a:μ+δ=1 by simp
    thus ?thesis unfolding a mult-1 unfolding μ-def using assms
      by (intro additive-chernoff-bound-upper-aux-2) auto
  qed
qed
```

Based on Theorem 2 by Hoeffding [11].

```
lemma hoeffding-bound-upper:
  assumes ⋀i. i∈I ⟹ a i ≤ b i
  assumes ⋀i. i∈I ⟹ AE ω in M. X i ω ∈ {a i..b i}
  defines n ≡ real (card I)
  defines μ ≡ (∑ i∈I. expectation (X i))
  assumes δ ≥ 0 (∑ i∈I. (b i − a i)^2) > 0
  shows 𝒫(ω in M. (∑ i∈I. X i ω) ≥ μ + δ ∗ n) ≤ exp (−2∗(n∗δ)^2 / (∑ i∈I.
(b i − a i)^2))
    (is ?L ≤ ?R)
proof (cases δ=0)
  case True thus ?thesis by simp
next
  case False
  define 𝒳 where 𝒳 = (λi. clamp (a i) (b i) ∘ X i)
  have transfer-to-clamped-vars-assms: (∀ i∈I. AE ω in M. X i ω ∈ {a i.. b i} ∧
a i ≤ b i)
    using assms(1,2) by auto
  note ttcv = transfer-to-clamped-vars[OF transfer-to-clamped-vars-assms 𝒳-def]
  note [measurable] = neg-assoc-imp-measurable[OF ttcv(1)]

  define s where s = (∑ i∈I. (b i − a i)^2)
  have s-gt-0: s > 0 using assms unfolding s-def by auto

  have I-ne: I ≠ {} using assms(6) by auto

  have n-gt-0: n > 0 using I-ne fin-I unfolding n-def by auto
```

define t **where** $t = 4 * \delta * n / s$

have *t-gt-0*: $t > 0$ **unfolding** *t-def* **using** *False n-gt-0 s-gt-0 assms* **by** *auto*

note *bounded′* = *integrable-bounded bounded-prod bounded-vec-mult-comp bounded-intros ttcv*(*5*)
note $\mathcal{X}$*-prod-mono* = *has-int-thatD*(*2*)[*OF neg-assoc-imp-prod-mono*[*OF fin-I ttcv*(*1*), **where** η=*Fwd*]]

have *int*: *integrable M* ($\mathcal{X}$ *i*) **if** $i \in I$ **for** *i*
using *that* **by** (*intro bounded′*) *simp-all*

define ν **where** $\nu\ i$ = *expectation* (*X i*) **for** *i*
have *1*: *expectation* ($\lambda x.\ \mathcal{X}\ i\ x - \nu\ i$) = *0* **if** $i \in I$ **for** *i*
unfolding ν*-def* **using** *int*[*OF that*] *ttcv*(*2*)[*OF that*] **by** (*simp add*:*prob-space*)

have *?L* = $\mathcal{P}(\omega\ in\ M.\ (\sum i \in I.\ \mathcal{X}\ i\ \omega) \geq \mu + \delta * n)$ **using** *ttcv*(*3*)[**where** η=*Rev*] **by** *simp*
also have ... = $\mathcal{P}(\omega\ in\ M.\ (\sum i \in I.\ \mathcal{X}\ i\ \omega - \nu\ i) \geq \delta * n)$
using *n-gt-0* **unfolding** μ*-def* ν*-def* **by** (*simp add*:*algebra-simps sum-subtractf*)
also have ... = $\mathcal{P}(\omega\ in\ M.\ (\prod i \in I.\ exp\ (t * (\mathcal{X}\ i\ \omega - \nu\ i))) \geq exp\ (t * \delta * n))$
using *t-gt-0* **by** (*simp add*: *sum-distrib-left*[*symmetric*] *exp-sum*[*OF fin-I*,*symmetric*])
also have ... $\leq$ *expectation* ($\lambda\omega.\ (\prod i \in I.\ exp\ (t * (\mathcal{X}\ i\ \omega - \nu\ i)))) / exp\ (t * \delta * n)$
by (*intro integral-Markov-inequality-measure*[**where** *A*={}] *bounded′ AE-I2 prod-nonneg fin-I*)
simp-all
also have ... $\leq (\prod i \in I.\ expectation\ (\lambda\omega.\ exp\ (t*(\mathcal{X}\ i\ \omega - \nu\ i)))) / exp\ (t * \delta * n)$
using *t-gt-0* **by** (*intro divide-right-mono* $\mathcal{X}$*-prod-mono bounded′ image-subsetI monotoneI*) *simp-all*
also have ... $\leq (\prod i \in I.\ exp\ (t\hat{}2 * ((b\ i - \nu\ i) - (a\ i - \nu\ i))^2 / 8)) / exp\ (t*\delta*n)$
using *ttcv*(*4*) *1*
by (*intro divide-right-mono prod-mono conjI Hoeffdings-lemma-bochner t-gt-0 AE-I2*) *simp-all*
also have ... = $(\prod i \in I.\ exp\ (t\hat{}2 * (b\ i - a\ i)^2 / 8)) / exp\ (t*\delta*n)$ **by** *simp*
also have ... = $exp(\ (t\hat{}2/8) * (\sum i \in I.\ (b\ i - a\ i)\hat{}2)) / exp\ (t*\delta*n)$
unfolding *exp-sum*[*OF fin-I*, *symmetric*] **by** (*simp add*:*algebra-simps sum-distrib-left*)
also have ... = $exp(\ (t\hat{}2/8) * s - t*\delta*n)$
unfolding *exp-diff s-def* **by** *simp*
also have ... = $exp(-2 * (n*\delta)\hat{}2 / s)$
using *s-gt-0* **unfolding** *t-def* **by** (*simp add*:*divide-simps power2-eq-square*)
also have ... = *?R* **unfolding** *s-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

end

Dual and two-sided versions of Theorem 1 and 2 by Hoeffding [11].

```
lemma additive-chernoff-bound-lower:
  assumes neg-assoc X I finite I
  assumes ⋀i. i∈I ⟹ AE ω in M. X i ω ∈ {0..1} I ≠ {}
  defines μ ≡ (∑ i∈I. expectation (X i)) / real (card I)
  assumes δ ∈ {0..μ} μ ∈ {0<..<1}
  shows 𝒫(ω in M. (∑ i∈I. X i ω) ≤ (μ−δ)∗real (card I)) ≤ exp (−real (card I)
∗ KL-div (μ−δ) μ)
    (is ?L ≤ ?R)
proof −
  note [measurable] = neg-assoc-imp-measurable[OF assms(1)]

  have int[simp]: integrable M (X i) if i ∈ I for i
   using that by (intro integrable-const-bound[where B=1] AE-mp[OF assms(3)[OF
that] AE-I2]) auto
  have n-gt-0: real (card I) > 0 using assms by auto

  hence 0: (1−μ) = (∑ i∈I. expectation (λω. 1 − X i ω)) / real (card I)
    unfolding μ-def by (simp add:prob-space sum-subtractf divide-simps)
  have 1:  neg-assoc (λi ω. 1 − X i ω) I
     by (intro neg-assoc-compose-simple[OF assms(2,1), where η=Rev]) (auto in-
tro:antimonoI)

  have 2: δ ≤ (1 − (1 − μ)) δ ≥ 0 using assms by auto
  have 3: 1−μ ∈ {0<..<1} using assms by auto
  have ?L = 𝒫(ω in M. (∑ i∈I. 1 − X i ω) ≥ ((1−μ)+δ)∗real (card I))
    by (simp add:sum-subtractf algebra-simps)
  also have ... ≤ exp (−real (card I) ∗ KL-div ((1−μ)+δ) (1−μ))
      using assms(3) 1 2 3 unfolding 0 by (intro additive-chernoff-bound-upper
assms(2,4)) auto
   also have ... = exp (−real (card I) ∗ KL-div (1−(μ−δ)) (1−μ)) by (simp
add:algebra-simps)
  also have ... = ?R using assms(6,7) by (subst KL-div-swap) (simp-all add:algebra-simps)
  finally show ?thesis by simp
qed

lemma hoeffding-bound-lower:
  assumes neg-assoc X I finite I
  assumes ⋀i. i∈I ⟹ a i ≤ b i
  assumes ⋀i. i∈I ⟹ AE ω in M. X i ω ∈ {a i..b i}
  defines n ≡ real (card I)
  defines μ ≡ (∑ i∈I. expectation (X i))
  assumes δ ≥ 0 (∑ i∈I. (b i − a i)^2) > 0
  shows 𝒫(ω in M. (∑ i∈I. X i ω) ≤ μ−δ∗n) ≤ exp (−2∗(n∗δ)^2 / (∑ i∈I. (b i
− a i)^2))
    (is ?L ≤ ?R)
proof −
  have 0: −μ = (∑ i∈I. expectation (λω. − X i ω)) unfolding μ-def by (simp
add:sum-negf)
```

have *1*: *neg-assoc* $(\lambda i\ \omega.\ -\ X\ i\ \omega)\ I$
by (*intro neg-assoc-compose-simple*[*OF assms*(*2*,*1*), **where** η=*Rev*]) (*auto intro*:*antimonoI*)

have *?L* = $\mathcal{P}(\omega\ in\ M.\ (\sum i{\in}I.\ -X\ i\ \omega) \geq (-\mu){+}\delta{*}n)$ **by** (*simp add*:*algebra-simps sum-negf*)
also have ... $\leq$ *exp* $(-2{*}(n{*}\delta)\hat{}2\ /\ (\sum i{\in}I.\ ((-a\ i) - (-b\ i))\hat{}2))$
using *assms*(*3*,*4*,*8*) **unfolding** *0 n-def* **by** (*intro hoeffding-bound-upper*[*OF 1*] *assms*(*2*,*4*,*7*)) *auto*
also have ... = *?R* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *hoeffding-bound-two-sided*:
assumes *neg-assoc X I finite I*
assumes $\bigwedge i.\ i{\in}I \Longrightarrow a\ i \leq b\ i$
assumes $\bigwedge i.\ i{\in}I \Longrightarrow AE\ \omega\ in\ M.\ X\ i\ \omega \in \{a\ i..b\ i\}\ I \neq \{\}$
defines $n \equiv real\ (card\ I)$
defines $\mu \equiv (\sum i{\in}I.\ expectation\ (X\ i))$
assumes $\delta \geq 0\ (\sum i{\in}I.\ (b\ i\ -\ a\ i)\hat{}2) > 0$
shows $\mathcal{P}(\omega\ in\ M.\ |(\sum i{\in}I.\ X\ i\ \omega){-}\mu| \geq \delta{*}n) \leq 2{*}exp\ (-2{*}(n{*}\delta)\hat{}2\ /\ (\sum i{\in}I.\ (b\ i\ -\ a\ i)\hat{}2))$
(**is** *?L* $\leq$ *?R*)
proof –
note [*measurable*] = *neg-assoc-imp-measurable*[*OF assms*(*1*)]

have *?L* = $\mathcal{P}(\omega\ in\ M.\ (\sum i{\in}I.\ X\ i\ \omega) \geq \mu{+}\delta{*}n \lor (\sum i{\in}I.\ X\ i\ \omega) \leq \mu{-}\delta{*}n\ )$
unfolding *abs-real-def* **by** (*intro arg-cong*[**where** *f*=*prob*] *Collect-cong*) *auto*
also have ... = *measure M* $(\{\omega{\in}space\ M.\ (\sum i{\in}I.\ X\ i\ \omega){\geq}\mu{+}\delta{*}n\}\cup\{\omega{\in}space\ M.\ (\sum i{\in}I.\ X\ i\ \omega){\leq}\mu{-}\delta{*}n\})$
by (*intro arg-cong*[**where** *f*=*prob*]) *auto*
also have ... $\leq \mathcal{P}(\omega\ in\ M.\ (\sum i{\in}I.\ X\ i\ \omega) \geq \mu{+}\delta{*}n) + \mathcal{P}(\omega\ in\ M.(\sum i{\in}I.\ X\ i\ \omega) \leq \mu{-}\delta{*}n\ )$
by (*intro measure-Un-le*) *measurable*
also have ... $\leq$ *exp* $(-2{*}(n{*}\delta)\hat{}2\ /\ (\sum i{\in}I.\ (b\ i{-}a\ i)\hat{}2)) + exp\ (-2{*}(n{*}\delta)\hat{}2\ /\ (\sum i{\in}I.\ (b\ i{-}a\ i)\hat{}2))$
unfolding *n-def* μ*-def* **by** (*intro hoeffding-bound-lower hoeffding-bound-upper add-mono assms*)
also have ... = *?R* **by** *simp*
finally show *?thesis* **by** *simp*
qed

end

end

4 The FKG inequality

The FKG inequality [9] is a generalization of Chebyshev's less known other inequality. It is sometimes referred to as Chebyshev's sum inequality. Although there is a also a continuous version, which can be stated as:

$$E[fg] \geq E[f]E[g]$$

where f, g are continuous simultaneously monotone or simultaneously antimonotone functions on the Lebesgue probability space $[a,b] \subseteq \mathbb{R}$. ($Ef$ denotes the expectation of the function.)

Note that the inequality is also true for totally ordered discrete probability spaces, for example: $\{1,\ldots,n\}$ with uniform probabilities.

The FKG inequality is essentially a generalization of the above to not necessarily totally ordered spaces, but finite distributive lattices.

The proof follows the derivation in the book by Alon and Spencer [2, Ch. 6].

theory *Negative-Association-FKG-Inequality*
  imports
    Negative-Association-Util
    Birkhoff-Finite-Distributive-Lattices.Birkhoff-Finite-Distributive-Lattices
begin

theorem *four-functions-helper*:
  fixes φ :: *nat* ⇒ *'a set* ⇒ *real*
  assumes *finite I*
  assumes $\bigwedge i.\ i \in \{0..3\} \Longrightarrow \varphi\ i \in Pow\ I \to \{0..\}$
  assumes $\bigwedge A\ B.\ A \subseteq I \Longrightarrow B \subseteq I \Longrightarrow \varphi\ 0\ A * \varphi\ 1\ B \leq \varphi\ 2\ (A \cup B) * \varphi\ 3\ (A \cap B)$
  shows $(\sum A{\in}Pow\ I.\ \varphi\ 0\ A)*(\sum B{\in}Pow\ I.\ \varphi\ 1\ B) \leq (\sum C{\in}Pow\ I.\ \varphi\ 2\ C)*(\sum D{\in}Pow\ I.\ \varphi\ 3\ D)$
  using *assms*
proof (*induction I arbitrary*:φ *rule*:*finite-induct*)
  case *empty*
  then show *?case* **using** *empty* **by** *auto*
next
  case (*insert x I*)
  define φ' **where** $\varphi'\ i\ A = \varphi\ i\ A + \varphi\ i\ (A \cup \{x\})$ **for** *i A*

  have *a*:$(\sum A{\in}Pow\ (insert\ x\ I).\ \varphi\ i\ A) = (\sum A{\in}Pow\ I.\ \varphi'\ i\ A)$ (**is** *?L1* = *?R1*) **for** *i*
  proof –
    have $?L1 = (\sum A{\in}Pow\ I.\ \varphi\ i\ A) + (\sum A{\in}insert\ x\ \text{‘}\ Pow\ I.\ \varphi\ i\ A)$
      using *insert*(*1*,*2*) **unfolding** *Pow-insert* **by** (*intro sum.union-disjoint*) *auto*
    also have $\ldots = (\sum A{\in}Pow\ I.\ \varphi\ i\ A) + (\sum A{\in}Pow\ I.\ \varphi\ i\ (insert\ x\ A))$
      using *insert*(*2*) **by** (*subst sum.reindex*) (*auto intro*!:*inj-onI*)
    also have ... = *?R1* **using** *insert*(*1*) **unfolding** φ'*-def sum.distrib* **by** *simp*

finally show *?thesis* **by** *simp*
qed

have *φ-int*: $\varphi\ 0\ A * \varphi\ 1\ B \leq \varphi\ 2\ C * \varphi\ 3\ D$
if $C = A \cup B$ $D = A \cap B$ $A \subseteq$ *insert x I* $B \subseteq$ *insert x I* **for** *A B C D*
using *that insert*(*5*) **by** *auto*

have *φ-nonneg*: $\varphi\ i\ A \geq 0$ **if** $A \subseteq$ *insert x I* $i \in \{0..3\}$ **for** *i A*
using *that insert*(*4*) **by** *auto*

have $\varphi'\ 0\ A * \varphi'\ 1\ B \leq \varphi'\ 2\ (A \cup B) * \varphi'\ 3\ (A \cap B)$ **if** $A \subseteq I$ $B \subseteq I$ **for** *A B*
proof −
define *a0 a1* **where** *a*: $a0 = \varphi\ 0\ A$ $a1 = \varphi\ 0$ (*insert x A*)
define *b0 b1* **where** *b*: $b0 = \varphi\ 1\ B$ $b1 = \varphi\ 1$ (*insert x B*)
define *c0 c1* **where** *c*: $c0 = \varphi\ 2\ (A \cup B)$ $c1 = \varphi\ 2$ (*insert x* $(A \cup B)$)
define *d0 d1* **where** *d*: $d0 = \varphi\ 3\ (A \cap B)$ $d1 = \varphi\ 3$ (*insert x* $(A \cap B)$)

have *0*:$a0 * b0 \leq c0 * d0$ **using** *that* **unfolding** *a b c d* **by** (*intro φ-int*) *auto*
have *1*:$a0 * b1 \leq c1 * d0$ **using** *that insert*(*2*) **unfolding** *a b c d* **by** (*intro φ-int*) *auto*
have *2*:$a1 * b0 \leq c1 * d0$ **using** *that insert*(*2*) **unfolding** *a b c d* **by** (*intro φ-int*) *auto*
have *3*:$a1 * b1 \leq c1 * d1$ **using** *that insert*(*2*) **unfolding** *a b c d* **by** (*intro φ-int*) *auto*
have *4*:$a0 \geq 0$ $a1 \geq 0$ $b0 \geq 0$ $b1 \geq 0$ **using** *that* **unfolding** *a b* **by** (*auto intro*!: *φ-nonneg*)
have *5*:$c0 \geq 0$ $c1 \geq 0$ $d0 \geq 0$ $d1 \geq 0$ **using** *that* **unfolding** *c d* **by** (*auto intro*!: *φ-nonneg*)

consider (*a*) $c1 = 0$ | (*b*) $d0 = 0$ | (*c*) $c1 > 0$ $d0 > 0$ **using** *4 5* **by** *argo*

then have $(a0 + a1) * (b0 + b1) \leq (c0 + c1) * (d0 + d1)$
proof (*cases*)
case *a*
hence $a0 * b1 = 0$ $a1 * b0 = 0$ $a1 * b1 = 0$
using *1 2 3* **by** (*intro order-antisym mult-nonneg-nonneg 4 5*;*simp-all*)+
then show *?thesis* **unfolding** *distrib-left distrib-right*
using *0 4 5* **by** (*metis add-mono mult-nonneg-nonneg*)
next
case *b*
hence $a0 * b0 = 0$ $a0 * b1 = 0$ $a1 * b0 = 0$
using *0 1 2* **by** (*intro order-antisym mult-nonneg-nonneg 4 5*;*simp-all*)+
then show *?thesis* **unfolding** *distrib-left distrib-right*
using *3 4 5* **by** (*metis add-mono mult-nonneg-nonneg*)
next
case *c*
have $0 \leq (c1*d0-a0*b1) * (c1*d0 - a1*b0)$
using *1 2* **by** (*intro mult-nonneg-nonneg*) *auto*
hence $(a0 + a1) * (b0 + b1)*d0*c1 \leq (a0*b0 + c1*d0) * (c1*d0 + a1*b1)$

```
      by (simp add:algebra-simps)
    hence (a0 + a1) ∗ (b0 + b1) ≤ ((a0∗b0)/d0 + c1) ∗ (d0 + (a1∗b1)/c1)
      using c 4 5 by (simp add:field-simps)
    also have ... ≤ (c0 + c1) ∗ (d0 + d1)
    using 0 3 c 4 5 by (intro mult-mono add-mono order.refl) (simp add:field-simps)+
    finally show ?thesis by simp
  qed

  thus ?thesis unfolding φ′-def a b c d by auto
 qed

 moreover have φ′ i ∈ Pow I → {0..} if i ∈ {0..3} for i
  using insert(4)[OF that] unfolding φ′-def by (auto intro!:add-nonneg-nonneg)
 ultimately show ?case unfolding a by (intro insert(3)) auto
qed
```

The following is the Ahlswede-Daykin inequality [1] also referred to by Alon and Spencer as the four functions theorem [2, Th. 6.1.1].

```
theorem four-functions:
 fixes α β γ δ :: ′a set ⇒ real
 assumes finite I
 assumes α ∈ Pow I → {0..} β ∈ Pow I → {0..} γ ∈ Pow I → {0..} δ ∈ Pow I
→ {0..}
 assumes ⋀A B. A ⊆ I ⟹ B ⊆ I ⟹ α A ∗ β B ≤ γ (A ∪ B) ∗ δ (A ∩ B)
 assumes M ⊆ Pow I N ⊆ Pow I
 shows (∑A∈M. α A)∗(∑B∈N. β B) ≤ (∑C| ∃A∈M. ∃B∈N. C=A∪B. γ
C)∗(∑D| ∃A∈M. ∃B∈N. D=A∩B. δ D)
  (is ?L ≤ ?R)
proof −
 define α′ where α′ A = (if A ∈ M then α A else 0) for A
 define β′ where β′ B = (if B ∈ N then β B else 0) for B
 define γ′ where γ′ C = (if ∃A∈M. ∃B∈N. C=A∪B then γ C else 0) for C
 define δ′ where δ′ D = (if ∃A∈M. ∃B∈N. D=A∩B then δ D else 0) for D

 define φ where φ i = [α′,β′,γ′,δ′] ! i for i

 have list-all (λi. φ i ∈ Pow I → {0..}) [0..<4]
  unfolding φ-def α′-def β′-def γ′-def δ′-def using assms(2−5)
  by (auto simp add:numeral-eq-Suc)
 hence φ-nonneg: φ i ∈ Pow I → {0..} if i ∈ {0..3} for i
  unfolding list.pred-set using that by auto

 have 0: φ 0 A ∗ φ 1 B ≤ φ 2 (A ∪ B) ∗ φ 3 (A ∩ B) (is ?L1 ≤ ?R1) if A ⊆ I
B ⊆ I for A B
 proof (cases A ∈ M ∧ B ∈ N)
  case True
  have ?L1 = α A ∗ β B using True unfolding φ-def α′-def β′-def by auto
  also have ... ≤ γ (A ∪ B) ∗ δ (A ∩ B) by(intro that assms(6))
  also have ... = ?R1 using True unfolding γ′-def δ′-def φ-def by auto
```

finally show *?thesis* **by** *simp*
next
case *False*
hence *?L1 = 0* **unfolding** *α′-def β′-def φ-def* **by** *auto*
also have ... ≤ *?R1* **using** *φ-nonneg*[*of 2*] *φ-nonneg*[*of 3*] *that*
by (*intro mult-nonneg-nonneg*) *auto*
finally show *?thesis* **by** *simp*
qed

have *fin-pow*: *finite* (*Pow I*) **using** *assms*(*1*) **by** *simp*

have *?L* = (∑ *A* ∈ *Pow I*. *α′ A*) ∗ (∑ *B* ∈ *Pow I*. *β′ B*)
unfolding *α′-def β′-def* **using** *assms*(*1*,*7*,*8*) **by** (*simp add*: *sum.If-cases Int-absorb1*)
also have ... = (∑ *A* ∈ *Pow I*. *φ 0 A*) ∗ (∑ *A* ∈ *Pow I*. *φ 1 A*) **unfolding** *φ-def* **by** *simp*
also have ... ≤ (∑ *A* ∈ *Pow I*. *φ 2 A*) ∗ (∑ *A* ∈ *Pow I*. *φ 3 A*)
by (*intro four-functions-helper assms*(*1*) *φ-nonneg 0*) *auto*
also have ... = (∑ *A* ∈ *Pow I*. *γ′ A*) ∗ (∑ *B* ∈ *Pow I*. *δ′ B*) **unfolding** *φ-def* **by** *simp*
also have ... = *?R*
unfolding *γ′-def δ′-def sum.If-cases*[*OF fin-pow*] *sum.neutral-const add-0-right* **using** *assms*(*7*,*8*)
by (*intro arg-cong2*[**where** *f*=(∗)] *sum.cong refl*) *auto*
finally show *?thesis* **by** *simp*
qed

Using Birkhoff's Representation Theorem [3, 5] it is possible to generalize the previous to finite distributive lattices [2, Cor. 6.1.2].

lemma *four-functions-in-lattice*:
fixes *α β γ δ* :: *′a* :: *finite-distrib-lattice* ⇒ *real*
assumes *range α* ⊆ {*0*..} *range β* ⊆ {*0*..} *range γ* ⊆ {*0*..} *range δ* ⊆ {*0*..}
assumes ⋀*x y*. *α x* ∗ *β y* ≤ *γ* (*x* ⊔ *y*) ∗ *δ* (*x* ⊓ *y*)
shows (∑ *x*∈*M*. *α x*)∗(∑ *y*∈*N*. *β y*) ≤ (∑ *c*| ∃ *x*∈*M*. ∃ *y*∈*N*. *c*=*x*⊔*y*. *γ c*)∗(∑ *d*| ∃ *x*∈*M*. ∃ *y*∈*N*. *d*=*x*⊓*y*. *δ d*)
(**is** *?L* ≤ *?R*)
proof −
let *?e* = *λx*::*′a*. ⦃ *x* ⦄
let *?f* = *the-inv ?e*

have *ran-e*: *range ?e* = 𝒪𝒥 **by** (*rule bij-betw-imp-surj-on*[*OF birkhoffs-theorem*])
have *inj-e*: *inj ?e* **by** (*rule bij-betw-imp-inj-on*[*OF birkhoffs-theorem*])

define *conv* :: (*′a* ⇒ *real*) ⇒ *′a set* ⇒ *real*
where *conv φ I* = (*if I* ∈ 𝒪𝒥 *then φ*(*?f I*) *else 0*) **for** *φ I*

define *α′ β′ γ′ δ′* **where** *prime-def*:*α′* = *conv α β′* = *conv β γ′* = *conv γ δ′* = *conv δ*

have *1*:*conv* $\varphi \in$ *Pow* $\mathcal{J} \rightarrow \{0..\}$ **if** *range* $\varphi \subseteq \{(0::real)..\}$ **for** φ
using *that* **unfolding** *conv-def* **by** (*intro Pi-I*) *auto*

have *0*:α' $A * \beta'$ $B \leq \gamma'$ $(A \cup B) * \delta'$ $(A \cap B)$ **if** $A \subseteq \mathcal{J}$ $B \subseteq \mathcal{J}$ **for** A B
proof (*cases* $A \in \mathcal{OJ} \wedge B \in \mathcal{OJ}$)
case *True*
define *x y* **where** *xy*: $x = ?f\ A$ $y = ?f\ B$

have *p0*:$?e\ (x \sqcup y) = A \cup B$
using *True ran-e* **unfolding** *join-irreducibles-join-homomorphism xy*
by (*subst* (*1 2*) *f-the-inv-into-f*[*OF inj-e*]) *auto*
hence *p1*:$A \cup B \in \mathcal{OJ}$ **using** *ran-e* **by** *auto*

have *p2*:$?e\ (x \sqcap y) = A \cap B$
using *True ran-e* **unfolding** *join-irreducibles-meet-homomorphism xy*
by (*subst* (*1 2*) *f-the-inv-into-f*[*OF inj-e*]) *auto*
hence *p3*:$A \cap B \in \mathcal{OJ}$ **using** *ran-e* **by** *auto*

have $\alpha'\ A * \beta'\ B = \alpha\ (?f\ A) * \beta\ (?f\ B)$ **using** *True* **unfolding** *prime-def conv-def* **by** *simp*
also have $\ldots \leq \gamma\ (?f\ A \sqcup ?f\ B) * \delta\ (?f\ A \sqcap ?f\ B)$ **by** (*intro assms*(*5*))
also have $\ldots = \gamma\ (x \sqcup y) * \delta\ (x \sqcap y)$ **unfolding** *xy* **by** *simp*
also have $\ldots = \gamma\ (?f\ (?e\ (x \sqcup y))) * \delta\ (?f\ (?e\ (x \sqcap y)))$ **by** (*simp add*: *the-inv-f-f*[*OF inj-e*])
also have $\ldots = \gamma\ (?f\ (A \cup B)) * \delta\ (?f\ (A \cap B))$ **unfolding** *p0 p2* **by** *auto*
also have $\ldots = \gamma'\ (A \cup B) * \delta'\ (A \cap B)$ **using** *p1 p3* **unfolding** *prime-def conv-def* **by** *auto*
finally show *?thesis* **by** *simp*
next
case *False*
hence $\alpha'\ A * \beta'\ B = 0$ **unfolding** *prime-def conv-def* **by** *simp*
also have $\ldots \leq \gamma'\ (A \cup B) * \delta'\ (A \cap B)$ **unfolding** *prime-def*
using *1 that assms*(*3*,*4*) **by** (*intro mult-nonneg-nonneg*) *auto*
finally show *?thesis* **by** *simp*
qed

define M' **where** $M' = (\lambda x.\ \{\!\!\{ x \}\!\!\})$ ‘ M
define N' **where** $N' = (\lambda x.\ \{\!\!\{ x \}\!\!\})$ ‘ N

have *ran1*: $M' \subseteq \mathcal{OJ}$ $N' \subseteq \mathcal{OJ}$ **unfolding** *M'-def N'-def* **using** *ran-e* **by** *auto*
hence *ran2*: $M' \subseteq$ *Pow* $\mathcal{J}$ $N' \subseteq$ *Pow* $\mathcal{J}$ **unfolding** *down-irreducibles-def* **by** *auto*

have $?f \in ?e$ ‘ $S \rightarrow S$ **for** S **using** *inj-e* **by** (*simp add*: *Pi-iff the-inv-f-f*)
hence *bij-betw*: *bij-betw* $?f$ ($?e$ ‘ S) S **for** S :: *'a set*
by (*intro bij-betwI*[**where** *g*=*?e*] *the-inv-f-f f-the-inv-into-f inj-e*) *auto*

have *a*: $\{C.\ \exists A{\in}M'.\ \exists B{\in}N'.\ C = A \cup B\} = ?e$ ‘ $\{c.\ \exists x{\in}M.\ \exists y{\in}N.\ c{=}x{\sqcup}y\}$
unfolding *M'-def N'-def Set.bex-simps join-irreducibles-join-homomorphism*[*symmetric*]

by *auto*
have *b*: {*D*. ∃ *A*∈*M'*. ∃ *B*∈*N'*. *D* = *A* ∩ *B*} = *?e* ` {*c*. ∃ *x*∈*M*. ∃ *y*∈*N*. *c*=*x*⊓*y*}
unfolding *M'-def N'-def Set.bex-simps join-irreducibles-meet-homomorphism*[*symmetric*]
by *auto*

have *M'-N'-un-ran*: {*C*. ∃ *A*∈*M'*. ∃ *B*∈*N'*. *C* = *A* ∪ *B*} ⊆ 𝒪𝒥
unfolding *a* **using** *ran-e* **by** *auto*
have *M'-N'-int-ran*: {*C*. ∃ *A*∈*M'*. ∃ *B*∈*N'*. *C* = *A* ∩ *B*} ⊆ 𝒪𝒥
unfolding *b* **using** *ran-e* **by** *auto*

have *?L* =(∑ *A*∈*M'*. *α* (*?f A*)) ∗ (∑ *A*∈*N'*. *β* (*?f A*))
unfolding *M'-def N'-def*
by (*intro arg-cong2*[**where** *f*=(∗)] *sum.reindex-bij-betw*[*symmetric*] *bij-betw*)
also have ... = (∑ *A*∈*M'*. *α' A*)∗(∑ *A*∈*N'*. *β' A*)
unfolding *prime-def conv-def* **using** *ran1* **by** (*intro arg-cong2*[**where** *f*=(∗)] *sum.cong refl*) *auto*
also have ... ≤ (∑ *C* | ∃ *A*∈*M'*. ∃ *B*∈*N'*. *C* = *A* ∪ *B*. *γ' C*) ∗ (∑ *D* | ∃ *A*∈*M'*. ∃ *B*∈*N'*. *D* = *A* ∩ *B*. *δ' D*)
using *ran2* **by** (*intro four-functions*[**where** *I*=𝒥] *0*) (*auto intro*!:*1 assms simp*:*prime-def*)
also have ... = (∑ *C*|∃ *A*∈*M'*. ∃ *B*∈*N'*. *C*=*A*∪*B*. *γ*(*?f C*))∗(∑ *D*|∃ *A*∈*M'*.∃ *B*∈*N'*. *D*=*A*∩*B*. *δ*(*?f D*))
using *M'-N'-un-ran M'-N'-int-ran* **unfolding** *prime-def conv-def*
by (*intro arg-cong2*[**where** *f*=(∗)] *sum.cong refl*) *auto*
also have ... = *?R*
unfolding *a b* **by** (*intro arg-cong2*[**where** *f*=(∗)] *sum.reindex-bij-betw bij-betw*)
finally show *?thesis* **by** *simp*
qed

theorem *fkg-inequality*:
fixes *μ* :: *'a* :: *finite-distrib-lattice* ⇒ *real*
assumes *range μ* ⊆ {*0*..} *range f* ⊆ {*0*..} *range g* ⊆ {*0*..}
assumes ⋀*x y*. *μ x* ∗ *μ y* ≤ *μ* (*x* ⊔ *y*) ∗ *μ* (*x* ⊓ *y*)
assumes *mono f mono g*
shows (∑ *x*∈*UNIV*. *μ x*∗*f x*) ∗ (∑ *x*∈*UNIV*. *μ x*∗*g x*) ≤ (∑ *x*∈*UNIV*. *μ x*∗*f x*∗*g x*) ∗ *sum μ UNIV*
(**is** *?L* ≤ *?R*)
proof −
define *α* **where** *α x* = *μ x* ∗ *f x* **for** *x*
define *β* **where** *β x* = *μ x* ∗ *g x* **for** *x*
define *γ* **where** *γ x* = *μ x* ∗ *f x* ∗ *g x* **for** *x*
define *δ* **where** *δ x* = *μ x* **for** *x*

have *0*:*f x* ≥ *0* **if** *range f* ⊆ {*0*..} **for** *f* :: *'a* ⇒ *real* **and** *x*
using *that* **by** *auto*

note *μfg-nonneg* = *0*[*OF assms*(*1*)] *0*[*OF assms*(*2*)] *0*[*OF assms*(*3*)]

have *1*:*α x* ∗ *β y* ≤ *γ* (*x* ⊔ *y*) ∗ *δ* (*x* ⊓ *y*) (**is** *?L1* ≤ *?R1*) **for** *x y*

proof −
have *?L1* = (*μ x* ∗ *μ y*) ∗ (*f x* ∗ *g y*) **unfolding** *α-def β-def* **by** (*simp add*:*ac-simps*)
also have ... ≤ (*μ* (*x* ⊔ *y*) ∗ *μ* (*x* ⊓ *y*)) ∗ (*f x* ∗ *g y*)
using *assms*(*2*,*3*) **by** (*intro mult-right-mono assms*(*4*) *mult-nonneg-nonneg*) *auto*
also have ... ≤ (*μ* (*x* ⊔ *y*) ∗ *μ* (*x* ⊓ *y*)) ∗ (*f* (*x* ⊔ *y*) ∗ *g* (*x* ⊔ *y*))
using *μfg-nonneg*
by (*intro mult-left-mono mult-mono monoD*[*OF assms*(*5*)] *monoD*[*OF assms*(*6*)] *mult-nonneg-nonneg*)
simp-all
also have ... = *?R1* **unfolding** *γ-def δ-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

have *?L* = (∑ *x*∈*UNIV*. *α x*) ∗ (∑ *y*∈*UNIV*. *β y*) **unfolding** *α-def β-def* **by** *simp*
also have ... ≤ (∑ *c*| ∃ *x*∈*UNIV*. ∃ *y*∈*UNIV*. *c*=*x*⊔*y*. *γ c*)∗(∑ *d*| ∃ *x*∈*UNIV*. ∃ *y*∈*UNIV*. *d*=*x*⊓*y*. *δ d*)
using *μfg-nonneg* **by** (*intro four-functions-in-lattice 1*) (*auto simp*:*α-def β-def γ-def δ-def*)
also have ... = (∑ *x*∈*UNIV*. *γ x*) ∗ (∑ *x*∈*UNIV*. *δ x*)
using *sup.idem*[**where** *'a*=*'a*] *inf.idem*[**where** *'a*=*'a*]
by (*intro arg-cong2*[**where** *f*=(∗)] *sum.cong refl UNIV-eq-I*[*symmetric*] *CollectI*) (*metis UNIV-I*)+
also have ... = *?R* **unfolding** *γ-def δ-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

theorem *fkg-inequality-gen*:
fixes *μ* :: *'a* :: *finite-distrib-lattice* ⇒ *real*
assumes *range μ* ⊆ {*0*..}
assumes ⋀*x y*. *μ x* ∗ *μ y* ≤ *μ* (*x* ⊔ *y*) ∗ *μ* (*x* ⊓ *y*)
assumes *monotone* (≤) ($\leq\geq_\tau$) *f monotone* (≤) ($\leq\geq_\sigma$) *g*
shows (∑ *x*∈*UNIV*. *μ x*∗*f x*) ∗ (∑ *x*∈*UNIV*. *μ x*∗*g x*) $\leq\geq_{\tau*\sigma}$ (∑ *x*∈*UNIV*. *μ x*∗*f x*∗*g x*) ∗ *sum μ UNIV*
(**is** *?L* $\leq\geq_{?x}$ *?R*)
proof −
define *a* **where** *a* = *max* (*MAX x*. −*f x*∗($\pm_\tau$)) (*MAX x*. −*g x*∗($\pm_\sigma$))

define *f'* **where** *f' x* = *a* + *f x*∗($\pm_\tau$) **for** *x*
define *g'* **where** *g' x* = *a* + *g x*∗($\pm_\sigma$) **for** *x*

have *f'-mono*: *mono f'* **unfolding** *f'-def* **using** *monotoneD*[*OF assms*(*3*)]
by (*intro monoI add-mono order.refl*) (*cases τ*, *auto simp*:*comp-def ac-simps*)

have *g'-mono*: *mono g'* **unfolding** *g'-def* **using** *monotoneD*[*OF assms*(*4*)]
by (*intro monoI add-mono order.refl*) (*cases σ*, *auto simp*:*comp-def ac-simps*)

have *f′-nonneg*: $f'\ x \geq 0$ **for** *x*
unfolding *f′-def a-def max-add-distrib-left*
by (*intro max.coboundedI1*) (*auto intro*!:*Max.coboundedI simp*: *algebra-simps real-0-le-add-iff*)

have *g′-nonneg*: $g'\ x \geq 0$ **for** *x*
unfolding *g′-def a-def max-add-distrib-left*
by (*intro max.coboundedI2*) (*auto intro*!:*Max.coboundedI simp*: *algebra-simps real-0-le-add-iff*)

let $?M = (\sum x \in UNIV.\ \mu\ x)$
let $?sum = (\lambda f.\ (\sum x{\in}UNIV.\ \mu\ x * f\ x))$

have $(\pm_{\tau*\sigma}) * ?L = ?sum\ (\lambda x.\ f\ x*(\pm_\tau)) * ?sum\ (\lambda x.\ g\ x*(\pm_\sigma))$
by (*simp add*:*ac-simps sum-distrib-left*[*symmetric*] *dir-mult-hom del*:*rel-dir-mult*)
also have $\ldots = (?sum\ (\lambda x.\ (f\ x*(\pm_\tau)+a))-?M*a) * (?sum\ (\lambda x.\ (g\ x*(\pm_\sigma)+a))-?M*a)$
by (*simp add*:*algebra-simps sum.distrib sum-distrib-left*)
also have $\ldots = (?sum\ f')*(?sum\ g') - ?M*a*?sum\ f' - ?M*a*?sum\ g' + ?M*?M*a*a$
unfolding *f′-def g′-def* **by** (*simp add*:*algebra-simps*)
also have $\ldots \leq ((\sum x{\in}UNIV.\ \mu\ x*f'\ x*g'\ x)*?M) - ?M*a*?sum\ f' - ?M*a*?sum\ g' + ?M*?M*a*a$
using *f′-nonneg g′-nonneg*
by (*intro diff-mono add-mono order.refl fkg-inequality assms*(*1,2*) *f′-mono g′-mono*) *auto*
also have $\ldots = ?sum\ (\lambda x.\ (f\ x*(\pm_\tau))*(g\ x*(\pm_\sigma)))*?M$
unfolding *f′-def g′-def* **by** (*simp add*:*algebra-simps sum.distrib sum-distrib-left*[*symmetric*])
also have $\ldots = (\pm_{\tau*\sigma}) * ?R$
by (*simp add*:*ac-simps sum.distrib sum-distrib-left*[*symmetric*] *dir-mult-hom del*:*rel-dir-mult*)
finally have $(\pm_{\tau*\sigma}) * ?L \leq (\pm_{\tau*\sigma}) * ?R$ **by** *simp*
thus *?thesis* **by** (*cases* $\tau*\sigma$, *auto*)
qed

theorem *fkg-inequality-pmf*:
fixes *M* :: (*′a* :: *finite-distrib-lattice*) *pmf*
fixes *f g* :: *′a* ⇒ *real*
assumes $\bigwedge x\ y.\ pmf\ M\ x * pmf\ M\ y \leq pmf\ M\ (x \sqcup y) * pmf\ M\ (x \sqcap y)$
assumes *monotone* $(\leq)$ $(\leq\geq_\tau)$ *f monotone* $(\leq)$ $(\leq\geq_\sigma)$ *g*
shows $(\int x.\ f\ x\ \partial M) * (\int x.\ g\ x\ \partial M) \leq\geq_{\tau\ *\ \sigma} (\int x.\ f\ x * g\ x\ \partial M)$
(**is** $?L \leq\geq_{-} ?R$)
proof −
have $0{:}?L = (\sum a{\in}UNIV.\ pmf\ M\ a * f\ a) * (\sum a{\in}UNIV.\ pmf\ M\ a * g\ a)$
by (*subst* (*1 2*) *integral-measure-pmf-real*[**where** *A*=*UNIV*]) (*auto simp*:*ac-simps*)
have $?R = ?R * (\int x.\ 1\ \partial M)$ **by** *simp*
also have $\ldots = (\sum a{\in}UNIV.\ pmf\ M\ a*f\ a*g\ a) * sum\ (pmf\ M)\ UNIV$
by (*subst* (*1 2*) *integral-measure-pmf-real*[**where** *A*=*UNIV*]) (*auto simp*:*ac-simps*)
finally have $1{:}\ ?R = (\sum a{\in}UNIV.\ pmf\ M\ a*f\ a*g\ a) * sum\ (pmf\ M)\ UNIV$ **by** *simp*

thus *?thesis* **unfolding** *0 1*
by (*intro fkg-inequality-gen assms*) *auto*
qed

end

5 Preliminary Results on Lattices

This entry establishes a few missing lemmas for the set-based theory of lattices from "HOL-Algebra". In particular, it introduces the sublocale for distributive lattices.

More crucially, a transfer theorem which can be used in conjunction with the Types-To-Sets mechanism to be able to work with locally defined finite distributive lattices.

This is being needed for the verification of the negative association of permutation distributions in Section 6.

theory *Negative-Association-More-Lattices*
imports *HOL−Algebra.Lattice*
begin

Lemma 1 Birkhoff Lattice Theory, p.8, L3

lemma (**in** *lattice*) *meet-assoc-law*:
assumes $x \in carrier\ L$ $y \in carrier\ L$ $z \in carrier\ L$
shows $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$
using *assms* **by** (*metis* (*full-types*) *eq-is-equal weak-meet-assoc*)

Lemma 1 Birkhoff Lattice Theory, p.8, L3

lemma (**in** *lattice*) *join-assoc-law*:
assumes $x \in carrier\ L$ $y \in carrier\ L$ $z \in carrier\ L$
shows $x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$
using *assms* **by** (*metis* (*full-types*) *eq-is-equal weak-join-assoc*)

Lemma 1 Birkhoff Lattice Theory, p.8, L4

lemma (**in** *lattice*) *absorbtion-law*:
assumes $x \in carrier\ L$ $y \in carrier\ L$
shows $x \sqcap (x \sqcup y) = x$ $x \sqcup (x \sqcap y) = x$
proof −
have $x \sqsubseteq x \sqcup y$ **using** *assms join-left* **by** *auto*
hence $x = x \sqcap (x \sqcup y)$ **using** *assms* **by** (*intro iffD1*[*OF le-iff-join*]) *auto*
thus $x \sqcap (x \sqcup y) = x$ **by** *simp*

have $x \sqcap y \sqsubseteq x$ **using** *assms meet-left* **by** *auto*
hence $(x \sqcap y) \sqcup x = x$ **using** *assms le-iff-meet* **by** (*intro iffD1*[*OF le-iff-meet*]) *auto*
thus $x \sqcup (x \sqcap y) = x$ **using** *join-comm* **by** *metis*
qed

Theorem 9 Birkhoff Lattice Theory, p.11

lemma (**in** *lattice*) *distrib-laws-equiv*:
defines *meet-distrib* ≡ (∀ *x y z*. {*x*,*y*,*z*}⊆*carrier L* ⟶ (*x* ⊓ (*y* ⊔ *z*)) = (*x* ⊓ *y*) ⊔ (*x* ⊓ *z*))
defines *join-distrib* ≡ (∀ *x y z*. {*x*,*y*,*z*}⊆*carrier L* ⟶ (*x* ⊔ (*y* ⊓ *z*)) = (*x* ⊔ *y*) ⊓ (*x* ⊔ *z*))
shows *meet-distrib* ⟷ *join-distrib*
proof
assume *a*:*meet-distrib*
have (*x* ⊔ *y*) ⊓ (*x* ⊔ *z*)= *x* ⊔ (*y* ⊓ *z*) (**is** *?L* = *?R*) **if** {*x*,*y*,*z*} ⊆ *carrier L* **for** *x y z*
proof −
have *?L* = ((*x* ⊔ *y*) ⊓ *x*) ⊔ ((*x* ⊔ *y*) ⊓ *z*) **using** *a that* **unfolding** *meet-distrib-def* **by** *simp*
also have ... = *x* ⊔ (*z* ⊓ (*x* ⊔ *y*)) **using** *that absorbtion-law meet-comm* **by** (*metis insert-subset*)
also have ... = *x* ⊔ ((*z* ⊓ *x*) ⊔ (*z* ⊓ *y*)) **using** *a that* **unfolding** *meet-distrib-def* **by** *simp*
also have ... = (*x* ⊔ (*z* ⊓ *x*)) ⊔ (*z* ⊓ *y*) **using** *that meet-assoc-law join-assoc-law* **by** *simp*
also have ... = *x* ⊔ (*z* ⊓ *y*) **using** *that absorbtion-law meet-comm* **by** (*metis insert-subset*)
also have ... = *?R* **by** (*metis meet-comm*)
finally show *?thesis* **by** *simp*
qed
thus *join-distrib* **unfolding** *join-distrib-def* **by** *auto*
next
assume *a*:*join-distrib*
have (*x* ⊓ *y*) ⊔ (*x* ⊓ *z*)= *x* ⊓ (*y* ⊔ *z*) (**is** *?L* = *?R*) **if** {*x*,*y*,*z*} ⊆ *carrier L* **for** *x y z*
proof −
have *?L* = ((*x* ⊓ *y*) ⊔ *x*) ⊓ ((*x* ⊓ *y*) ⊔ *z*) **using** *a that* **unfolding** *join-distrib-def* **by** *simp*
also have ... = *x* ⊓ (*z* ⊔ (*x* ⊓ *y*)) **using** *that absorbtion-law join-comm* **by** (*metis insert-subset*)
also have ... = *x* ⊓ ((*z* ⊔ *x*) ⊓ (*z* ⊔ *y*)) **using** *a that* **unfolding** *join-distrib-def* **by** *simp*
also have ... = (*x* ⊓ (*z* ⊔ *x*)) ⊓ (*z* ⊔ *y*) **using** *that meet-assoc-law join-assoc-law* **by** *simp*
also have ... = *x* ⊓ (*z* ⊔ *y*) **using** *that absorbtion-law join-comm* **by** (*metis insert-subset*)
also have ... = *?R* **by** (*metis join-comm*)
finally show *?thesis* **by** *simp*
qed
thus *meet-distrib* **unfolding** *meet-distrib-def* **by** *auto*
qed

lemma (**in** *lattice*) *lub-unique-set*:
assumes *is-lub L z S*

shows $z = \bigsqcup S$
proof –
have *a*:*is-lub L z′ S* $\Longrightarrow$ $z = z'$ **for** z'
using *least-unique assms* **by** *simp*
show *?thesis*
unfolding *sup-def*
by (*rule someI2*[**where** *a*=*z*], *rule assms*(*1*), *rule a*)
qed

lemma (**in** *lattice*) *lub-unique*:
assumes *is-lub L z* {*x*,*y*}
shows $z = x \sqcup y$
using *lub-unique-set*[*OF assms*] **unfolding** *join-def* **by** *auto*

lemma (**in** *lattice*) *glb-unique-set*:
assumes *is-glb L z S*
shows $z = \bigsqcap S$
proof –
have *a*:*is-glb L z′ S* $\Longrightarrow$ $z = z'$ **for** z'
using *greatest-unique assms*(*1*) **by** *simp*
show *?thesis*
unfolding *meet-def inf-def*
by (*rule someI2*[**where** *a*=*z*], *rule assms*(*1*), *rule a*)
qed

lemma (**in** *lattice*) *glb-unique*:
assumes *is-glb L z* {*x*,*y*}
shows $z = x \sqcap y$
using *glb-unique-set*[*OF assms*] **unfolding** *meet-def* **by** *auto*

lemma (**in** *lattice*) *inf-lower*:
assumes $S \subseteq$ *carrier L* $s \in S$ *finite S*
shows $\bigsqcap S \sqsubseteq s$
proof –
have *is-glb L* ($\bigsqcap S$) *S* **using** *assms*(*2*) **by** (*intro finite-inf-greatest assms*(*1*,*3*)) *auto*
hence ($\bigsqcap S$) $\in$ *Lower L S* **using** *greatest-mem* **by** *metis*
thus *?thesis* **using** *assms*(*1*,*2*) **by** *auto*
qed

lemma (**in** *lattice*) *sup-upper*:
assumes $S \subseteq$ *carrier L* $s \in S$ *finite S*
shows $s \sqsubseteq \bigsqcup S$
proof –
have *is-lub L* ($\bigsqcup S$) *S* **using** *assms*(*2*) **by** (*intro finite-sup-least assms*(*1*,*3*)) *auto*
hence ($\bigsqcup S$) $\in$ *Upper L S* **using** *least-mem* **by** *metis*
thus *?thesis* **using** *assms*(*1*,*2*) **by** *auto*
**qed*

locale *distrib-lattice* = *lattice* +
assumes *max-distrib*:
$x \in carrier\ L \Longrightarrow y \in carrier\ L \Longrightarrow z \in carrier\ L \Longrightarrow (x \sqcap (y \sqcup z)) = (x \sqcap y) \sqcup (x \sqcap z)$
begin

lemma *min-distrib*:
assumes $x \in carrier\ L\ y \in carrier\ L\ z \in carrier\ L$
shows $(x \sqcup (y \sqcap z)) = (x \sqcup y) \sqcap (x \sqcup z)$
proof −
have *a*:$\forall x\ y\ z.\ \{x,\ y,\ z\} \subseteq carrier\ L \longrightarrow x \sqcap (y \sqcup z) = x \sqcap y \sqcup x \sqcap z$ **using** *max-distrib* **by** *auto*
show *?thesis* **using** *iffD1*[*OF distrib-laws-equiv a*] *assms* **by** *simp*
qed

end

locale *finite-ne-distrib-lattice* = *distrib-lattice* +
assumes *non-empty-carrier*: *carrier L* ≠ {}
assumes *finite-carrier*: *finite* (*carrier L*)
begin

lemma *bounded-lattice-axioms-1*: ∃ *x*. *least L x* (*carrier L*)
proof −
have ⨅ *carrier L* ∈ *Lower L* (*carrier L*)
by (*intro greatest-mem*[**where** *L*=*L*] *finite-inf-greatest*[*OF finite-carrier - non-empty-carrier*]) *auto*
hence ∀ *x* ∈ *carrier L*. (⨅ *carrier L*)⊑*x* **unfolding** *Lower-def* **by** *auto*
moreover have ⨅ *carrier L* ∈ *carrier L*
using *finite-inf-closed*[*OF finite-carrier - non-empty-carrier*] **by** *auto*
ultimately have *least L* (⨅ *carrier L*) (*carrier L*)
unfolding *least-def* **by** *auto*

thus *?thesis* **by** *auto*
qed

lemma *bounded-lattice-axioms-2*: ∃ *x*. *greatest L x* (*carrier L*)
proof −
have ⨆ *carrier L* ∈ *Upper L* (*carrier L*)
by (*intro least-mem*[**where** *L*=*L*] *finite-sup-least*[*OF finite-carrier - non-empty-carrier*]) *auto*
hence ∀ *x* ∈ *carrier L*. *x* ⊑ (⨆ *carrier L*) **unfolding** *Upper-def* **by** *auto*
moreover have ⨆ *carrier L* ∈ *carrier L*
using *finite-sup-closed*[*OF finite-carrier - non-empty-carrier*] **by** *auto*
ultimately have *greatest L* (⨆ *carrier L*) (*carrier L*)
unfolding *greatest-def* **by** *auto*

thus *?thesis* **by** *auto*
qed

```
sublocale bounded-lattice
  using bounded-lattice-axioms-1 bounded-lattice-axioms-2
  by (unfold-locales) auto

lemma inf-empty: ⨅ {} = ⊤
proof −
  have is-glb L ⊤ {} using top-greatest by simp
  thus ?thesis using glb-unique-set by auto
qed

lemma inf-closed: S ⊆ carrier L ⟹⨅S ∈ carrier L
  using finite-carrier inf-empty top-closed finite-inf-closed
  by (metis finite-subset)

lemma inf-insert:
  assumes x ∈ carrier L S ⊆ carrier L
  shows ⨅ (insert x S) = x ⊓ (⨅S)
proof −
  have fin-S: finite S using finite-carrier assms(2) finite-subset by metis
  have inf-S-carr: ⨅S ∈ carrier L using inf-closed[OF assms(2)] by force

  have x ⊓ (⨅S) ⊑ s if s ∈ S for s
  proof −
    have ⨅S ⊑ s using that fin-S assms(2)
      by (metis empty-iff finite-inf-greatest greatest-Lower-below)
     moreover have x ⊓ (⨅S) ⊑ ⨅S using inf-S-carr assms(1) meet-right by
metis
    ultimately show ?thesis using inf-S-carr meet-closed
      by (meson assms le-trans subsetD that)
  qed
  moreover have x ⊓ (⨅S) ⊑ x using inf-S-carr assms(1) meet-left by metis
  ultimately have x ⊓ (⨅S) ∈ Lower L (insert x S)
    using assms(1) meet-closed inf-S-carr unfolding Lower-def by auto
  moreover have y ⊑ (x ⊓ (⨅S)) if y ∈ Lower L (insert x S) for y
  proof−
    have y-carr: y ∈ carrier L using that assms unfolding Lower-def by auto
    have y-lb: y ⊑ x using that assms unfolding Lower-def by auto

    moreover have y ∈ Lower L S using that unfolding Lower-def by auto
    hence y ⊑ ⨅S using finite-inf-greatest[OF fin-S assms(2)]
      by (metis greatest-le inf-empty top-higher y-carr)
    ultimately show ?thesis
      using y-carr inf-S-carr assms(1) meet-le by simp
  qed
  ultimately have is-glb L (x ⊓ (⨅S)) (insert x S) by (simp add: greatest-def)
  thus ?thesis by (intro glb-unique-set[symmetric])
qed
```

lemma *sup-empty*: $\bigsqcup$ {} = ⊥
proof −
  have *is-lub L* ⊥ {} **using** *bottom-least* **by** *simp*
  thus *?thesis* **using** *lub-unique-set* **by** *auto*
qed

lemma *sup-closed*: *S* ⊆ *carrier L* ⟹$\bigsqcup$ *S* ∈ *carrier L*
  using *finite-carrier sup-empty bottom-closed finite-sup-closed*
  by (*metis finite-subset*)

lemma *sup-insert*:
  assumes *x* ∈ *carrier L S* ⊆ *carrier L*
  shows $\bigsqcup$ (*insert x S*) = *x* ⊔ ($\bigsqcup$*S*)
proof −
  have *fin-S*: *finite S* **using** *finite-carrier assms*(*2*) *finite-subset* **by** *metis*
  have *sup-S-carr*: $\bigsqcup$*S* ∈ *carrier L* **using** *sup-closed*[*OF assms*(*2*)] **by** *force*

  have *s* ⊑ *x* ⊔ ($\bigsqcup$*S*) **if** *s* ∈ *S* **for** *s*
  proof −
    have *s* ⊑ $\bigsqcup$*S* **using** *that fin-S assms*(*2*)
      by (*metis empty-iff finite-sup-least least-Upper-above*)
    moreover have $\bigsqcup$*S* ⊑ *x* ⊔ ($\bigsqcup$*S*) **using** *sup-S-carr assms*(*1*) *join-right* **by**
metis
    ultimately show *?thesis* **using** *sup-S-carr join-closed assms*
      by (*meson le-trans subsetD that*)
  qed
  moreover have *x* ⊑ *x* ⊔ ($\bigsqcup$*S*) **using** *sup-S-carr assms*(*1*) *join-left* **by** *metis*
  ultimately have *x* ⊔ ($\bigsqcup$*S*) ∈ *Upper L* (*insert x S*)
    using *assms*(*1*) *sup-S-carr* **unfolding** *Upper-def* **by** *auto*
  moreover have *x* ⊔ ($\bigsqcup$*S*) ⊑ *y* **if** *y* ∈ *Upper L* (*insert x S*) **for** *y*
  proof−
    have *y-carr*: *y* ∈ *carrier L* **using** *that assms* **unfolding** *Lower-def* **by** *auto*
    have *y-lb*: *x* ⊑ *y* **using** *that assms* **by** *auto*

    moreover have *y* ∈ *Upper L S* **using** *that* **unfolding** *Upper-def* **by** *auto*
    hence $\bigsqcup$*S* ⊑ *y* **using** *finite-sup-least*[*OF fin-S assms*(*2*)]
      using *least-le sup-empty bottom-lower y-carr* **by** *metis*
    ultimately show *?thesis*
      using *y-carr sup-S-carr assms*(*1*) *join-le* **by** *simp*
  qed
  ultimately have *is-lub L* (*x* ⊔ ($\bigsqcup$*S*)) (*insert x S*) **by** (*simp add*: *least-def*)
  thus *?thesis* **by** (*intro lub-unique-set*[*symmetric*])
qed

lemma *inf-carrier*: $\bigsqcap$ (*carrier L*) = ⊥
proof −
  have $\bigsqcap$*carrier L* ∈ *Lower L* (*carrier L*)
   by (*intro greatest-mem*[**where** *L*=*L*] *finite-inf-greatest*[*OF finite-carrier - non-empty-carrier*])
      auto

hence ∀ *x* ∈ *carrier L*. (⨅ *carrier L*)⊑*x* **unfolding** *Lower-def* **by** *auto*
moreover have ⨅ *carrier L* ∈ *carrier L*
using *finite-inf-closed*[*OF finite-carrier - non-empty-carrier*] **by** *auto*
ultimately show *?thesis* **by** (*intro bottom-eq*) *auto*
qed

lemma *sup-carrier*: ⨆ (*carrier L*) = ⊤
proof −
have ⨆ *carrier L* ∈ *Upper L* (*carrier L*)
by (*intro least-mem*[**where** *L*=*L*] *finite-sup-least*[*OF finite-carrier - non-empty-carrier*])
auto
hence ∀ *x* ∈ *carrier L*. *x*⊑ (⨆ *carrier L*) **unfolding** *Upper-def* **by** *auto*
moreover have ⨆ *carrier L* ∈ *carrier L*
using *finite-sup-closed*[*OF finite-carrier - non-empty-carrier*] **by** *auto*
ultimately show *?thesis* **by** (*intro top-eq*) *auto*
qed

lemma *transfer-to-type*:
assumes *finite* (*carrier L*) *type-definition Rep Abs* (*carrier L*)
defines *inf'* ≡ (λ*M*. *Abs* (⨅ *Rep* ‘ *M*))
defines *sup'* ≡ (λ*M*. *Abs* (⨆ *Rep* ‘ *M*))
defines *join'* ≡ (λ*x y*. *Abs* (*Rep x* ⊓ *Rep y*))
defines *le'* ≡ (λ*x y*. (*Rep x* ⊑ *Rep y*))
defines *less'* ≡ (λ*x y*. (*Rep x* ⊏ *Rep y*))
defines *meet'* ≡ (λ*x y*. (*Abs* (*Rep x* ⊔ *Rep y*)))
defines *bot'*≡ (*Abs* ⊥ :: '*c*)
defines *top'* ≡ *Abs* ⊤
shows *class.finite-distrib-lattice inf' sup' join' le' less' meet' bot' top'*
proof −
interpret *type-definition Rep Abs* (*carrier L*)
using *assms*(*2*) **by** *auto*

note *defs* = *inf'-def sup'-def join'-def le'-def less'-def meet'-def bot'-def bot'-def top'-def*
note *td* = *Rep Rep-inverse Abs-inverse inf-closed sup-closed meet-closed join-closed Rep-range*

have *class-lattice*: *class.lattice join' le' less' meet'*
unfolding *defs* **using** *td*
proof (*unfold-locales*, *goal-cases*)
case *1* **thus** *?case* **unfolding** *lless-eq* **by** *auto*
next
case *2* **thus** *?case* **by** (*metis le-refl*)
next
case *3* **thus** *?case* **by** (*metis le-trans*)
next
case *4* **thus** *?case* **by** (*meson Rep-inject local.le-antisym*)
next

case *5* **thus** *?case* **by** (*metis meet-left*)
next
case *6* **thus** *?case* **by** (*metis meet-right*)
next
case *7* **thus** *?case* **by** (*metis meet-le*)
next
case *8* **thus** *?case* **by** (*metis join-left*)
next
case *9* **thus** *?case* **by** (*metis join-right*)
next
case *10* **thus** *?case* **by** (*metis join-le*)
qed

have *class-distrib-lattice*: *class.distrib-lattice join′ le′ less′ meet′*
unfolding *class.distrib-lattice-def eqTrueI*[*OF class-lattice*]
unfolding *defs class.distrib-lattice-axioms-def* **using** *td*
using *min-distrib* **by** *auto*

have *class-finite*: *class.finite TYPE*(*′c*)
by (*unfold-locales*) (*metis assms*(*1*) *Abs-image finite-imageI*)

have *class-finite-lattice*: *class.finite-lattice inf′ sup′ join′ le′ less′ meet′ bot′ top′*
unfolding *class.finite-lattice-def eqTrueI*[*OF class-lattice*] *eqTrueI*[*OF class-finite*]
unfolding *defs class.distrib-lattice-axioms-def class.finite-lattice-axioms-def* **using** *td*
proof (*intro conjI TrueI*, *goal-cases*)
case *1* **thus** *?case* **using** *sup-carrier inf-empty* **by** *simp*
next
case *2* **thus** *?case* **unfolding** *image-insert* **by** (*metis inf-insert image-subsetI*)
next
case *3* **thus** *?case* **using** *inf-carrier sup-empty* **by** *simp*
next
case *4* **thus** *?case* **unfolding** *image-insert* **by** (*metis sup-insert image-subsetI*)
next
case *5* **thus** *?case* **using** *inf-carrier* **by** *simp*
next
case *6* **thus** *?case* **using** *sup-carrier* **by** *simp*
qed

show *?thesis*
using *class-finite-lattice class-distrib-lattice*
unfolding *class.finite-distrib-lattice-def* **by** *auto*
qed

end

end

6 Permutation Distributions

One of the fundamental examples for negatively associated random variables are permutation distributions.

Let $x_1, \ldots, x_n$ be n (not-necessarily) distinct values from a totally ordered set, then we choose a permutation $\sigma : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ uniformly at random Then the random variables defined by $X_i(\sigma) = x_{\sigma(i)}$ are negatively associated.

An important special case is the case where x consists of 1 one and $(n-1)$ zeros, modelling randomly putting a ball into one of n bins. Of course the process can be repeated independently, the resulting distribution is also referred to as the balls into bins process. Because of the closure properties established before, it is possible to conclude that the number of hits of each bin in such a process are also negatively associated random variables.

In this section, we will derive that permutation distributions are negatively associated. The proof follows Dubashi [8, Th. 10] closely. A very short proof was presented in the work by Joag-Dev [13], however after close inspection that proof seemed to missing a lot of details. In fact, I don't think it is correct.

theory *Negative-Association-Permutation-Distributions*
  imports
    Negative-Association-Definition
    Negative-Association-FKG-Inequality
    Negative-Association-More-Lattices
    Finite-Fields.Finite-Fields-More-PMF
    HOL−Types-To-Sets.Types-To-Sets
    Executable-Randomized-Algorithms.Randomized-Algorithm
    Twelvefold-Way.Card-Bijections
begin

The following introduces a lattice for n-element subsets of a finite set (with size larger or equal to n.) A subset x is smaller or equal to y, if the smallest element of x is smaller or equal to the smallest element of y, the second smallest element of x is smaller or equal to the second smallest element of y, etc.)

The lattice is introduced without name by Dubashi [**?**, Example 7].

definition *le-ordered-set-lattice* :: $('a::linorder)$ *set* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *bool*
 where *le-ordered-set-lattice S T = list-all2* $(\le)$ (*sorted-list-of-set S*) (*sorted-list-of-set T*)

definition *ordered-set-lattice* :: $('a :: linorder)$ *set* $\Rightarrow$ *nat* $\Rightarrow$ $'a$ *set gorder*
  where *ordered-set-lattice S n* =
    ⦇ *carrier* = $\{T.\ T \subseteq S \wedge$ *finite* $T \wedge$ *card* $T = n\}$,
      eq = (=),
      le = *le-ordered-set-lattice* ⦈

definition *osl-repr* :: (*'a* :: *linorder*) *set* ⇒ *nat* ⇒ *'a set* ⇒ *nat* ⇒ *'a*
  where *osl-repr S n e* = (λ*i* ∈ {..<*n*}. *sorted-list-of-set e* ! *i*)

lemma *osl-carr-sorted-list-of-set*:
  assumes *finite S n* ≤ *card S*
  assumes *s* ∈ *carrier* (*ordered-set-lattice S n*)
  defines *t* ≡ *sorted-list-of-set s*
  shows *finite s card s* = *n s* ⊆ *S length t* = *n set t* = *s sorted-wrt* (<) *t*
  using *assms* **unfolding** *ordered-set-lattice-def* **by** *auto*

lemma *ordered-set-lattice-carrier-intro*:
  assumes *finite S n* ≤ *card S*
  assumes *set s* ⊆ *S distinct s length s* = *n*
  shows *set s* ∈ *carrier* (*ordered-set-lattice S n*)
  using *assms distinct-card* **unfolding** *ordered-set-lattice-def* **by** *auto*

lemma *osl-list-repr-inj*:
  assumes *finite S n* ≤ *card S*
  assumes *s* ∈ *carrier* (*ordered-set-lattice S n*)
  assumes *t* ∈ *carrier* (*ordered-set-lattice S n*)
  assumes ⋀*i*. *osl-repr S n s i* = *osl-repr S n t i*
  shows *s* = *t*
proof −
  note *c1* = *osl-carr-sorted-list-of-set*[*OF assms*(*1*,*2*,*3*)]
  note *c2* = *osl-carr-sorted-list-of-set*[*OF assms*(*1*,*2*,*4*)]

  have *sorted-list-of-set s* ! *i* = *sorted-list-of-set t* ! *i* **if** *i* < *n* **for** *i*
    using *assms*(*5*) *that* **unfolding** *osl-repr-def lessThan-iff restrict-def* **by** *metis*
  hence *sorted-list-of-set s* = *sorted-list-of-set t*
    using *c1*(*4*) *c2*(*4*) **by** (*intro nth-equalityI*) *auto*
  thus *s* = *t*
    using *c1*(*1*) *c2*(*1*) *sorted-list-of-set-inject* **by** *auto*
qed

lemma *osl-leD*:
  assumes *finite S n* ≤ *card S*
  assumes *e* ∈ *carrier* (*ordered-set-lattice S n*)
  assumes *f* ∈ *carrier* (*ordered-set-lattice S n*)
  shows $e \sqsubseteq_{ordered\text{-}set\text{-}lattice\ S\ n} f$ ⟷ (∀ *i*. *osl-repr S n e i* ≤ *osl-repr S n f i*) (**is**
?L = *?R*)
proof −
  note *c1* = *osl-carr-sorted-list-of-set*[*OF assms*(*1*,*2*,*3*)]
  note *c2* = *osl-carr-sorted-list-of-set*[*OF assms*(*1*,*2*,*4*)]

  have *?L* = *list-all2* (≤) (*sorted-list-of-set e*) (*sorted-list-of-set f*)
    unfolding *ordered-set-lattice-def le-ordered-set-lattice-def* **by** *simp*
  also have . . . = *?R* **using** *c1*(*4*) *c2*(*4*) **unfolding** *list-all2-conv-all-nth osl-repr-def*
by *simp*

```
  finally show ?thesis by simp
qed

lemma ordered-set-lattice-partial-order:
  fixes S :: ('a :: linorder) set
  assumes finite S n ≤ card S
  shows partial-order (ordered-set-lattice S n)
proof −
  let ?L = ordered-set-lattice S n

  note osl-list-repr-inj = osl-list-repr-inj[OF assms]
  note osl-leD = osl-leD[OF assms]

  have ref:x ⊑?L x if x ∈ carrier ?L for x
    using osl-leD that by auto

  have antisym:x = y if x ⊑?L y y ⊑?L x x ∈ carrier ?L y ∈ carrier ?L for x y
    using osl-leD osl-list-repr-inj that by (metis order-antisym)

  have trans:x ⊑?L z
    if x ⊑?L y y ⊑?L z x ∈ carrier ?L y ∈ carrier ?L z ∈ carrier ?L  for x y z
    using osl-leD that by (meson order-trans)

  have eq-eq: (.= ?L) = (=) unfolding ordered-set-lattice-def by simp

  show partial-order ?L
    using ref antisym trans eq-eq by (unfold-locales) presburger+
qed

lemma map2-max-mono:
  fixes xs :: ('a :: linorder) list
  assumes length xs = length ys
  assumes sorted-wrt (<) xs sorted-wrt (<) ys
  shows sorted-wrt (<) (map2 max xs ys)
  using assms
proof (induction xs ys rule:list-induct2)
  case Nil
  then show ?case by simp
next
  case (Cons x xs y ys)
  have max x y < max a b if (a,b) ∈ set (zip xs ys) for a b
  proof −
    have x < a using set-zip-leftD[OF that] Cons(3) by auto
    moreover have y < b using set-zip-rightD[OF that] Cons(4) by auto
   ultimately show ?thesis by (auto intro: max.strict-coboundedI1 max.strict-coboundedI2)
  qed
  moreover have sorted-wrt (<) (map2 max xs ys)
    using Cons(3,4) by (intro Cons(2)) auto
  ultimately show ?case by auto
```

```
qed

lemma map2-min-mono:
  fixes xs :: ('a :: linorder) list
  assumes length xs = length ys
  assumes sorted-wrt (<) xs sorted-wrt (<) ys
  shows sorted-wrt (<) (map2 min xs ys)
  using assms
proof (induction xs ys rule:list-induct2)
  case Nil
  then show ?case by simp
next
  case (Cons x xs y ys)
  have min x y < min a b if (a,b) ∈ set (zip xs ys) for a b
  proof −
    have x < a using set-zip-leftD[OF that] Cons(3) by auto
    moreover have y < b using set-zip-rightD[OF that] Cons(4) by auto
   ultimately show ?thesis by (auto intro: min.strict-coboundedI1 min.strict-coboundedI2)
  qed
  moreover have sorted-wrt (<) (map2 min xs ys)
    using Cons(3,4) by (intro Cons(2)) auto
  ultimately show ?case by auto
qed

lemma ordered-set-lattice-carrier-finite-ne:
  assumes finite S n ≤ card S
  shows carrier (ordered-set-lattice S n) ≠ {} finite (carrier (ordered-set-lattice S
n))
proof −
  let ?C = carrier (ordered-set-lattice S n)

  have 0 < (card S choose n) by (intro zero-less-binomial assms(2))
   also have ... = card {T. T ⊆ S ∧ card T = n} unfolding n-subsets[OF
assms(1)] by simp
  also have ... = card {T. T ⊆ S ∧ finite T ∧ card T = n}
     using assms(1) finite-subset by (intro arg-cong[where f=card] Collect-cong)
auto
  also have ... = card ?C unfolding ordered-set-lattice-def by simp
  finally have card ?C > 0 by simp
  thus ?C ≠ {} finite ?C unfolding card-gt-0-iff by auto
qed

lemma ordered-set-lattice-lattice:
  fixes S :: ('a :: linorder) set
  assumes finite S n ≤ card S
  shows finite-ne-distrib-lattice (ordered-set-lattice S n)
proof −
  let ?L = ordered-set-lattice S n
```

note *osl-leD* = *osl-leD*[*OF assms*]
note *osl-list-repr-inj* = *osl-list-repr-inj*[*OF assms*]

interpret *partial-order ?L* **by** (*intro ordered-set-lattice-partial-order assms*)

define *lmax* **where** *lmax x y* = *set* (*map2 max* (*sorted-list-of-set x*) (*sorted-list-of-set y*))
for *x y* :: *'a set*

define *lmin* **where** *lmin x y* = *set* (*map2 min* (*sorted-list-of-set x*) (*sorted-list-of-set y*))
for *x y* :: *'a set*

have *lmax-1*:
osl-repr S n (*lmax s t*) *i* = *max* (*osl-repr S n s i*) (*osl-repr S n t i*) (**is** *?L1* = *?R1*)
lmax s t ∈ *carrier ?L*
if *s* ∈ *carrier ?L t* ∈ *carrier ?L* **for** *s t i*
proof −
note *s-carr* = *osl-carr-sorted-list-of-set*[*OF assms that*(*1*)]
note *t-carr* = *osl-carr-sorted-list-of-set*[*OF assms that*(*2*)]

have *s*:*sorted-wrt* (<) (*map2 max* (*sorted-list-of-set s*) (*sorted-list-of-set t*))
using *s-carr t-carr* **by** (*intro map2-max-mono*) *auto*
hence *?L1* = (λ*i* ∈ {..<*n*}. (*map2 max* (*sorted-list-of-set s*) (*sorted-list-of-set t*)) ! *i*) *i*
unfolding *lmax-def osl-repr-def strict-sorted-iff*
by (*subst linorder-class.sorted-list-of-set.idem-if-sorted-distinct*) *auto*
also have ... = (λ*i* ∈ {..<*n*}. *max* (*sorted-list-of-set s* ! *i*) (*sorted-list-of-set t* ! *i*)) *i*
using *s-carr t-carr* **by** *simp*
also have ... = *?R1* **unfolding** *osl-repr-def* **by** *auto*
finally show *?L1* = *?R1* **by** *simp*

have *set* (*zip* (*sorted-list-of-set s*) (*sorted-list-of-set t*)) ⊆ *S* × *S*
using *s-carr*(*3*,*5*) *t-carr*(*3*,*5*) **by** (*auto intro*:*set-zip-leftD set-zip-rightD*)
hence *set* (*map2 max* (*sorted-list-of-set s*) (*sorted-list-of-set t*)) ⊆ *S*
by (*auto simp*:*max-def*)
thus *lmax s t* ∈ *carrier ?L*
using *s-carr t-carr s* **unfolding** *lmax-def strict-sorted-iff*
by (*intro ordered-set-lattice-carrier-intro*[*OF assms*]) *auto*
qed

have *lmin-1*:
osl-repr S n (*lmin s t*) *i* = *min* (*osl-repr S n s i*) (*osl-repr S n t i*) (**is** *?L1* = *?R1*)
lmin s t ∈ *carrier ?L*
if *s* ∈ *carrier ?L t* ∈ *carrier ?L* **for** *s t i*
proof −

note *s-carr* = *osl-carr-sorted-list-of-set*[*OF assms that*(*1*)]
note *t-carr* = *osl-carr-sorted-list-of-set*[*OF assms that*(*2*)]

have *s*:*sorted-wrt* (<) (*map2 min* (*sorted-list-of-set s*) (*sorted-list-of-set t*))
using *s-carr t-carr* **by** (*intro map2-min-mono*) *auto*
hence *?L1* = ($\lambda i \in \{..<n\}$. (*map2 min* (*sorted-list-of-set s*) (*sorted-list-of-set t*)) ! *i*) *i*
unfolding *lmin-def osl-repr-def strict-sorted-iff*
by (*subst linorder-class.sorted-list-of-set.idem-if-sorted-distinct*) *auto*
also have ... = ($\lambda i \in \{..<n\}$. *min* (*sorted-list-of-set s* ! *i*) (*sorted-list-of-set t* ! *i*)) *i*
using *s-carr t-carr* **by** *simp*
also have ... = *?R1* **unfolding** *osl-repr-def* **by** *auto*
finally show *?L1* = *?R1* **by** *simp*

have *set* (*zip* (*sorted-list-of-set s*) (*sorted-list-of-set t*)) $\subseteq S \times S$
using *s-carr*(*3*,*5*) *t-carr*(*3*,*5*) **by** (*auto intro*:*set-zip-leftD set-zip-rightD*)
hence *set* (*map2 min* (*sorted-list-of-set s*) (*sorted-list-of-set t*)) $\subseteq S$
by (*auto simp*:*min-def*)
thus *lmin s t* ∈ *carrier ?L*
using *s-carr t-carr s* **unfolding** *lmin-def strict-sorted-iff*
by (*intro ordered-set-lattice-carrier-intro*[*OF assms*]) *auto*
qed

have *lmax*: *is-lub ?L* (*lmax x y*) {*x*,*y*} **if** *x* ∈ *carrier ?L y* ∈ *carrier ?L* **for** *x y*
using *that lmax-1 osl-leD* **by** (*intro least-UpperI*) (*auto simp*:*Upper-def*)
hence ∃ *s*. *is-lub ?L s* {*x*, *y*} **if** *x* ∈ *carrier ?L y* ∈ *carrier ?L* **for** *x y*
using *that* **by** *auto*
hence *1*: *upper-semilattice ?L* **by** (*unfold-locales*) *auto*

have *lmin*: *is-glb ?L* (*lmin x y*) {*x*,*y*} **if** *x* ∈ *carrier ?L y* ∈ *carrier ?L* **for** *x y*
using *that lmin-1 osl-leD* **by** (*intro greatest-LowerI*) (*auto simp*:*Lower-def*)
hence ∃ *s*. *is-glb ?L s* {*x*, *y*} **if** *x* ∈ *carrier ?L y* ∈ *carrier ?L* **for** *x y*
using *that* **by** *auto*
hence *2*: *lower-semilattice ?L* **by** (*unfold-locales*) *auto*

have *4*:*lattice ?L* **using** *1 2* **unfolding** *lattice-def* **by** *auto*
interpret *lattice ?L* **using** *4* **by** *simp*

have *join-eq*: $x \sqcap_{?L} y$ = *lmin x y* **if** *x* ∈ *carrier ?L y* ∈ *carrier ?L* **for** *x y*
by (*intro glb-unique*[*symmetric*] *that lmin*)

have *meet-eq*: $x \sqcup_{?L} y$ = *lmax x y* **if** *x* ∈ *carrier ?L y* ∈ *carrier ?L* **for** *x y*
by (*intro lub-unique*[*symmetric*] *that lmax*)

have $(x \sqcap_{?L} (y \sqcup_{?L} z)) = (x \sqcap_{?L} y) \sqcup_{?L} (x \sqcap_{?L} z)$
if *x* ∈ *carrier ?L y* ∈ *carrier ?L z* ∈ *carrier ?L* **for** *x y z*
proof −
have *osl-repr S n* (*lmin x* (*lmax y z*)) *i* = *osl-repr S n* (*lmax* (*lmin x y*) (*lmin*

x z)) *i* **for** *i*
using *lmax-1 that lmin-1* **by** (*simp add*:*min-max-distrib2*)
hence *lmin x* (*lmax y z*) = *lmax* (*lmin x y*) (*lmin x z*)
by (*intro osl-list-repr-inj lmax-1 lmin-1 that allI*)
thus *?thesis* **using** *that* **by** (*simp add*: *meet-eq join-eq lmax-1 lmin-1*)
qed
thus *?thesis* **using** *4 ordered-set-lattice-carrier-finite-ne*[*OF assms*(*1*,*2*)] **by** (*unfold-locales*) *auto*
qed

lemma *insort-eq*:
fixes *xs* :: (*'a* :: *linorder*) *list*
assumes *sorted xs*
shows ∃ *ys zs. insort e xs* = *ys*@*e*#*zs* ∧ *ys*@*zs*=*xs* ∧ *set ys* ⊆ {..<*e*} ∧ *set zs* ⊆ {*e*..}
proof −
let *?ys* = *takeWhile* (λ*x. x* < *e*) *xs*
let *?zs* = *dropWhile* (λ*x. x* < *e*) *xs*

have *a*:*insort e xs* = *?ys*@*e*#*?zs* **by** (*induction xs*) *auto*

have *sorted* (*?ys*@*e*#*?zs*) **unfolding** *a*[*symmetric*] **using** *assms sorted-insort* **by** *auto*
hence *sorted* ([*e*]@*?zs*) **by** (*simp add*: *sorted-append*)
hence *set ?zs* ⊆ {*e*..} **unfolding** *sorted-append* **by** *auto*
moreover have *set ?ys* ⊆ {..<*e*} **by** (*metis lessThan-iff set-takeWhileD subset-eq*)
moreover have *?ys* @ *?zs* = *xs* **by** *simp*
ultimately show *?thesis* **using** *a* **by** *blast*
qed

lemma *list-all2-insort*:
fixes *xs ys* :: (*'a* :: *linorder*) *list*
assumes *length xs* = *length ys sorted xs sorted ys*
shows *list-all2* (≤) *xs ys* ⟷ *list-all2* (≤) (*insort e xs*) (*insort e ys*)
proof −
obtain *x1 x3* **where** *xs*:
xs = *x1*@*x3 insort e xs* = *x1*@*e*#*x3 set x1* ⊆ {..<*e*} *set x3* ⊆ {*e*..}
using *insort-eq*[*OF assms*(*2*)] **by** *blast*
obtain *y1 y3* **where** *ys*: *ys* = *y1*@*y3*
insort e ys = *y1*@*e*#*y3 set y1* ⊆ {..<*e*} *set y3* ⊆ {*e*..}
using *insort-eq*[*OF assms*(*3*)] **by** *blast*

have *l*: *length y1* + *length y3* = *length x1* + *length x3* **using** *assms*(*1*) *xs*(*1*) *ys*(*1*) **by** *simp*

have *list-all2* (≤) *xs ys* ⟷ *list-all2* (≤) (*x1*@*x3*) (*y1*@*y3*) **by** (*simp add*: *xs ys*)
also have ... ⟷ *list-all2* (≤) (*x1*@*e*#*x3*) (*y1*@*e*#*y3*) (**is** *?L* ⟷ *?R*)
proof (*cases length x1* < *length y1*)
case *True*

```
    have length x3 > 0 using l True by linarith

    hence (x1@x3) ! length x1 ≥ e
      using xs(4) nth-mem in-mono unfolding nth-append by fastforce
    moreover have (y1@y3) ! length x1 < e
      using True ys(3) nth-mem unfolding nth-append by auto
    moreover have length x1 < length (x1@x3) using l True by auto
    ultimately have 1:?L = False
      unfolding xs ys list-all2-conv-all-nth by (meson leD order.trans)

    have (y1@e#y3) ! length x1 < e
      using True ys(3) nth-mem unfolding nth-append by auto
    moreover have (x1@e#x3) ! length x1 = e by simp
    moreover have length x1 < length (x1@e#x3) using l True by auto
    ultimately have ?R = False
      unfolding xs(2) ys(2) list-all2-conv-all-nth by (metis leD)

    thus  ?thesis using 1 by auto
  next
    case False
    let ?x1 = take (length y1) x1
    define x2 where [simp]: x2 = drop (length y1) x1

    define y2 where [simp]: y2 = take (length x1−length y1) y3
    let ?y3 = drop (length x1−length y1) y3

    have l2: length x2 = length y2 using False l by simp
    have set-x2: set x2 ⊆ {..<e}
      unfolding x2-def using xs(3) set-drop-subset subset-trans by metis
    have set-y2: set y2 ⊆ {e..}
      unfolding y2-def using ys(4) set-take-subset subset-trans by metis

    have set (x2@[e]) ⊆ {..e} set (e#y2) ⊆ {e..}
      using set-x2 set-y2 by auto
    hence a′:list-all2 (λx y. x ≤ e ∧ e ≤ y) (x2@[e]) (e#y2)
     using l2  set-zip-leftD set-zip-rightD by (intro list-all2I conjI ballI case-prodI2)
fastforce+
    have a:list-all2 (≤) (x2@[e]) (e#y2) by (intro list-all2-mono[OF a′]) auto

    have b′:list-all2 (λx y. x ≤ e ∧ e ≤ y) x2 y2
       using l2 set-x2 set-y2  set-zip-leftD set-zip-rightD by (intro list-all2I conjI
ballI case-prodI2) fastforce+
    have b:list-all2 (≤) x2 y2 by (intro list-all2-mono[OF b′]) auto

    have ?L ⟷ list-all2 (≤) ((?x1@x2)@x3) (y1@y2@?y3) by simp
   also have . . . ⟷ list-all2 (≤) (?x1@x2@x3) (y1@y2@?y3) using append-assoc
by metis
    also have . . . ⟷ list-all2 (≤) ?x1 y1 ∧ list-all2 (≤) (x2@x3) (y2@?y3)
      using False by (intro list-all2-append) auto
```

also have ... ⟷ *list-all2* (≤) *?x1 y1* ∧ (*list-all2* (≤) *x2 y2* ∧ *list-all2* (≤) *x3 ?y3*)
using *l False* **by** (*intro arg-cong2*[**where** *f*=(∧)] *refl list-all2-append*) *simp*
also have ... ⟷ *list-all2* (≤) *?x1 y1*∧(*list-all2* (≤) (*x2*@[*e*]) (*e*#*y2*)∧*list-all2* (≤) *x3 ?y3*)
using *a b* **by** *simp*
also have ... ⟷ *list-all2* (≤) *?x1 y1* ∧ (*list-all2* (≤) ((*x2*@[*e*])@*x3*) ((*e*#*y2*)@*?y3*))
using *l False* **by** (*intro arg-cong2*[**where** *f*=(∧)] *refl list-all2-append*[*symmetric*]) *simp*
also have ... ⟷ *list-all2* (≤) (*?x1*@((*x2*@[*e*])@*x3*)) (*y1*@((*e*#*y2*)@*?y3*))
using *False* **by** (*intro list-all2-append*[*symmetric*]) *auto*
also have ... ⟷ *list-all2* (≤) ((*?x1*@*x2*)@(*e*#*x3*)) (*y1*@*e*#(*y2*@*?y3*))
using *append-assoc* **by** (*intro arg-cong2*[**where** *f*=*list-all2* (≤)]) *simp-all*
also have ... ⟷ *?R* **by** *simp*
finally show *?thesis* **by** *simp*
qed
also have ... ⟷ *list-all2* (≤) (*insort e xs*) (*insort e ys*) **using** *xs ys* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *le-ordered-set-lattice-diff*:
fixes *x y* :: (′*a* :: *linorder*) *set*
assumes *finite x finite y card x* = *card y*
shows *le-ordered-set-lattice x y* ⟷ *le-ordered-set-lattice* (*x* − *y*) (*y* − *x*)
proof −
let *?le* = *le-ordered-set-lattice*
define *u v S* **where** *vars*:*u* = *x* − *y v* = *y* − *x S* = *x* ∩ *y*

have *fins*: *finite S finite u finite v* **unfolding** *vars* **using** *assms* **by** *auto*

have *disj*: *S* ∩ *u* = {} *S* ∩ *v* = {} **unfolding** *vars* **by** *auto*

have *cards*: *card u* = *card v* **unfolding** *vars* **using** *assms*
by (*simp add*: *card-le-sym-Diff order-antisym*)

have *?le x y* = *?le* (*u* ∪ *S*) (*v* ∪ *S*) **unfolding** *vars* **by** (*intro arg-cong2*[**where** *f*=*?le*]) *auto*
also have ... = *?le u v* **using** *fins*(*1*) *disj*
proof (*induction S rule*:*finite-induct*)
case *empty* **thus** *?case* **by** *simp*
next
case (*insert x F*)
define *us* **where** *us* = *sorted-list-of-set* (*u* ∪ *F*)
define *vs* **where** *vs* = *sorted-list-of-set* (*v* ∪ *F*)
have *card* (*u* ∪ *F*) = *card u* + *card F* **using** *insert fins* **by** (*intro card-Un-disjoint*) *auto*
also have ... = *card v* + *card F* **using** *cards* **by** *auto*
also have ... = *card* (*v* ∪ *F*) **using** *insert fins* **by** (*intro card-Un-disjoint*[*symmetric*]) *auto*

finally have *cards'*: *card* (*u* ∪ *F*) = *card* (*v* ∪ *F*) **by** *simp*

have *?le* (*u* ∪ *insert x F*) (*v* ∪ *insert x F*) = *?le* (*insert x* (*u* ∪ *F*)) (*insert x* (*v* ∪ *F*))
by *simp*
also have ... = *list-all2* (≤) (*insort x us*) (*insort x vs*)
unfolding *le-ordered-set-lattice-def us-def vs-def* **using** *insert fins*(*2*,*3*)
by (*intro arg-cong2*[**where** *f*=*list-all2* (≤)] *sorted-list-of-set-insert*) *auto*
also have ... = *list-all2* (≤) *us vs*
using *cards'* **by** (*intro list-all2-insort*[*symmetric*]) (*simp-all add*:*us-def vs-def*)
also have ... = *?le* (*u* ∪ *F*) (*v* ∪ *F*)
unfolding *le-ordered-set-lattice-def us-def vs-def* **by** *simp*
also have ... = *?le u v* **using** *insert* **by** (*intro insert*) *auto*
finally show *?case* **by** *simp*
qed
also have ... = *?le* (*x*− *y*) (*y*−*x*) **unfolding** *vars* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *ordered-set-lattice-carrier*:
assumes *T* ∈ *carrier* (*ordered-set-lattice S n*)
shows *finite T card T* = *n T* ⊆ *S*
using *assms* **unfolding** *ordered-set-lattice-def* **by** *auto*

lemma *ordered-set-lattice-dual*:
assumes *finite S n* ≤ *card S*
defines *L* ≡ *ordered-set-lattice S n*
defines *M* ≡ *ordered-set-lattice S* (*card S* − *n*)
shows
⋀*x*. *x* ∈ *carrier L* ⟹ (*S*−*x*) ∈ *carrier M*
⋀*x*. *x* ∈ *carrier M* ⟹ (*S*−*x*) ∈ *carrier L*
⋀*x y*. *x* ∈ *carrier L* ∧ *y* ∈ *carrier L* ⟹ $x \sqsubseteq_L y$ ⟷ $(S-y) \sqsubseteq_M (S-x)$
proof (*goal-cases*)
case (*1 x*)
thus *?case* **using** *assms*(*1*,*2*) **unfolding** *ordered-set-lattice-def M-def L-def*
by (*auto intro*:*card-Diff-subset*)
next
case (*2 x*)
thus *?case* **using** *assms*(*1*,*2*) **unfolding** *ordered-set-lattice-def M-def L-def*
by (*auto simp*:*card-Diff-subset-Int Int-absorb1*)
next
case (*3 x y*)
hence *a*:*finite x finite y card x* = *card y x* ⊆ *S y* ⊆ *S*
unfolding *ordered-set-lattice-def M-def L-def* **by** *auto*

have *b*:*card* (*S* − *m*) = *card S* − *card m* **if** *m* ⊆ *S* **for** *m*
using *that assms*(*1*) *card-Diff-subset finite-subset*[*OF - assms*(*1*)] **by** *auto*

have *le-ordered-set-lattice x y* = *le-ordered-set-lattice* (*x*−*y*) (*y*−*x*)

by (*intro le-ordered-set-lattice-diff a*)
also have ... = *le-ordered-set-lattice* ((*S*−*y*)−(*S*−*x*)) ((*S*−*x*)−(*S*−*y*))
using *a* **by** (*intro arg-cong2*[**where** *f*=*le-ordered-set-lattice*]) *auto*
also have ... = *le-ordered-set-lattice* (*S* − *y*) (*S* − *x*)
using *a b assms*(*1*) **by** (*intro le-ordered-set-lattice-diff*[*symmetric*]) *auto*
finally have *le-ordered-set-lattice x y* = *le-ordered-set-lattice* (*S* − *y*) (*S* − *x*)
by *simp*
thus *?case* **unfolding** *ordered-set-lattice-def M-def L-def* **by** *simp*
qed

lemma *bij-betw-ord-set-lattice-pairs*:
assumes *finite S n* ≤ *card S*
defines *L* ≡ *ordered-set-lattice S n*
assumes *x* ∈ *carrier L y* ∈ *carrier L x* $\sqsubseteq_L$ *y*
shows ∃ *φ*. *bij-betw φ x y* ∧ *strict-mono-on x φ* ∧ (∀ *e*. *φ e* ≥ *e*)
proof −
let *?xs* = *sorted-list-of-set x*
let *?ys* = *sorted-list-of-set y*

let *?p1* = *the-inv-into* {..<*n*} (λ*i*. *?xs* ! *i*)
let *?p2* = (λ*i*. *?ys* ! *i*)

have *x*: *card x* = *n finite x* **using** *assms*(*4*) **unfolding** *L-def ordered-set-lattice-def*
by *auto*
have *y*: *card y* = *n finite y* **using** *assms*(*5*) **unfolding** *L-def ordered-set-lattice-def*
by *auto*
have *l-xs*: *length ?xs* = *n* **using** *length-sorted-list-of-set x* **by** *simp*
have *l-ys*: *length ?ys* = *n* **using** *length-sorted-list-of-set y* **by** *simp*

have *le*: *?xs* ! *i* ≤ *?ys* ! *i* **if** *i* ∈ {..<*n*} **for** *i*
using *assms*(*6*) *l-xs l-ys that* **unfolding** *L-def ordered-set-lattice-def le-ordered-set-lattice-def*
by (*auto simp add*:*list-all2-conv-all-nth*)

have *xs-strict-mono*: *strict-mono-on* {..<*n*} ((!) *?xs*)
using *strict-sorted-list-of-set*
by (*metis l-xs lessThan-iff sorted-wrt-iff-nth-less strict-mono-onI*)

hence *inj-xs*: *inj-on* ((!) *?xs*) {..<*n*} **using** *strict-mono-on-imp-inj-on* **by** *auto*
have *set ?xs* = *x* **using** *set-sorted-list-of-set x* **by** *simp*
hence *ran-xs*: ((!) *?xs*) ‘ {..<*n*} = *x* **using** *set-conv-nth* **unfolding** *l-xs*[*symmetric*]
by *fast*

have *set ?ys* = *y* **using** *set-sorted-list-of-set y* **by** *simp*
hence *ran-ys*: ((!) *?ys*) ‘ {..<*n*} = *y* **using** *set-conv-nth* **unfolding** *l-ys*[*symmetric*]
by *fast*

have *p1-strict-mono*: *strict-mono-on x ?p1*
proof (*rule strict-mono-onI*)
fix *r s* **assume** *a*: *r* ∈ *x s* ∈ *x r* < *s*

```
    have ?p1 r ∈ {..<n} using a ran-xs by (intro the-inv-into-into[OF inj-xs])
auto
   moreover have ?p1 s ∈ {..<n} using a ran-xs by (intro the-inv-into-into[OF
inj-xs]) auto
   moreover have ?xs ! (?p1 r) = r using a ran-xs by (intro f-the-inv-into-f[OF
inj-xs]) auto
   moreover have ?xs ! (?p1 s) = s using a ran-xs by (intro f-the-inv-into-f[OF
inj-xs]) auto
  ultimately show ?p1 r < ?p1 s using a(3) strict-mono-on-leD[OF xs-strict-mono]
by fastforce
  qed

  have ran-p1: ?p1 ` x = {..<n} using ran-xs the-inv-into-onto[OF inj-xs] by simp

  have p2-strict-mono: strict-mono-on {..<n} ?p2
    using strict-sorted-list-of-set
    by (metis l-ys lessThan-iff sorted-wrt-iff-nth-less strict-mono-onI)

  define φ where φ = (λe. if e ∈ x then (?p2 (?p1 e)) else e)

  have strict-mono-on x (?p2 ∘ ?p1)
  proof (rule strict-mono-onI)
    fix r s assume a: r ∈ x s ∈ x r < s
    have ?p1 r < ?p1 s using a strict-mono-onD[OF p1-strict-mono] by auto
    moreover have ?p1 r ∈ {..<n} ?p1 s ∈ {..<n} using a ran-p1 by auto
    ultimately show (?p2 ∘ ?p1) r < (?p2 ∘ ?p1) s
      using strict-mono-onD[OF p2-strict-mono] by simp
  qed

  hence φ-strict-mono: strict-mono-on x φ unfolding φ-def strict-mono-on-def
by simp
  hence φ-inj: inj-on φ x using strict-mono-on-imp-inj-on by auto

  have φ ` x ⊆ y using ran-p1 ran-ys unfolding φ-def by auto
  hence φ ` x = y using card-image[OF φ-inj] x y by (intro card-seteq) auto
  hence bij-betw φ x y using φ-inj unfolding bij-betw-def by auto

  moreover have φ e ≥ e for e
  proof (cases e ∈ x)
    case True
    have e = ?xs ! (?p1 e)
      using True ran-xs by (intro f-the-inv-into-f[symmetric] inj-xs) auto
    also have ... ≤ ?p2 (?p1 e) using ran-p1 True by (intro le) auto
    also have ... = φ e using True by (simp add:φ-def)
    finally show ?thesis by simp
  next
    case False
    then show ?thesis unfolding φ-def by simp
  qed
```

ultimately show *?thesis* **using** *φ-strict-mono* **by** *auto*
qed

definition *bij-pmf I F = pmf-of-set {f. bij-betw f I F ∧ f ∈ extensional I}*

lemma *card-bijections′*:
assumes *finite A finite B card A = card B*
shows *card {f. bij-betw f A B ∧ f ∈ extensional A} = fact (card A)* (**is** *?L = ?R*)
proof −
have *?L = card {f ∈ A →$_E$ B. bij-betw f A B}*
using *bij-betw-imp-surj-on*[**where** *A=A* **and** *B=B*]
by (*intro arg-cong*[**where** *f=card*] *Collect-cong*) (*auto simp*:*PiE-def Pi-def*)
also have ... = *fact* (*card A*) **using** *card-bijections*[*OF assms*] *assms*(*3*) **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *bij-betw-non-empty-finite*:
assumes *finite I finite F card I = card F*
shows
finite {f. bij-betw f I F ∧ f ∈ extensional I} (**is** *?T1*)
{f. bij-betw f I F ∧ f ∈ extensional I} ≠ {} (**is** *?T2*)
proof −
have *fact* (*card I*) > (*0::nat*) **using** *fact-gt-zero* **by** *simp*
thus *?T1 ?T2*
using *card-bijections′*[*OF assms*] *card-gt-0-iff* **by** *force+*
qed

lemma *bij-pmf*:
assumes *finite I finite F card I = card F*
shows
set-pmf (*bij-pmf I F*) = *{f. bij-betw f I F ∧ f ∈ extensional I}*
finite (*set-pmf* (*bij-pmf I F*))
using *bij-betw-non-empty-finite*[*OF assms*] **unfolding** *bij-pmf-def* **by** *auto*

lemma *expectation-ge-eval-at-point*:
assumes ⋀*y. y ∈ set-pmf p ⟹ f y ≥* (*0::real*)
assumes *integrable p f*
shows *pmf p x ∗ f x ≤* (∫ *x. f x ∂p*) (**is** *?L ≤ ?R*)
proof −
have *?L =* (∑ *a∈{x}. f a ∗ of-bool*(*a=x*) ∗ *pmf p a*) **by** *simp*
also have ... = (∫ *a. f a ∗ of-bool* (*a = x*) *∂p*)
by (*intro integral-measure-pmf-real*[*symmetric*]) *auto*
also have ... ≤ *?R*
using *assms* **by** (*intro integral-mono-AE′ AE-pmfI*) *auto*
finally show *?thesis* **by** *simp*
qed

lemma *split-bij-pmf*:
  assumes *finite I finite F card I = card F J ⊆ I*
  shows *bij-pmf I F =*
    do {
      S ← pmf-of-set {*S. card S = card J ∧ S ⊆ F*};
      φ ← bij-pmf J S;
      ψ ← bij-pmf (*I−J*) (*F−S*);
      return-pmf (*merge J* (*I−J*) (*φ*, *ψ*))
    } (**is** *?L = ?R*)
proof (*rule pmf-eq-iff-le*)
  fix *x*

  let *?p1 = pmf-of-set* {*S. card S = card J ∧ S ⊆ F*}
  let *?p2 = bij-pmf J*
  let *?p3* = (*λS. bij-pmf* (*I−J*) (*F−S*))

  have *f0*: *finite J* **using** *finite-subset assms*(*1,4*) **by** *metis*
  have *f1*: *finite* (*I−J*) **using** *finite-subset assms*(*1,4*) **by** *force*

  note *pos1 = pmf-of-set*[*OF bij-betw-non-empty-finite*(*2,1*)[*OF assms*(*1−3*)]]

  show *pmf* (*bij-pmf I F*) *x ≤ pmf ?R x*
  proof (*cases x ∈ set-pmf ?L*)
    case *True*
    hence *a*:*bij-betw x I F x ∈ extensional I*
      using *bij-pmf*[*OF assms*(*1−3*)] **by** *auto*

    define *T* **where** *T = x ‘ J*
    define *y* **where** *y = restrict x J*
    define *z* **where** *z = restrict x* (*I−J*)

    have *x-on-compl*: *x ‘* (*I−J*) = (*F−T*) **using** *a assms*(*4*) **unfolding** *T-def bij-betw-def*
      by (*subst inj-on-image-set-diff*[**where** *C=I*]) *auto*

    have *T-F*: *T ⊆ F* **using** *bij-betw-imp-surj-on*[*OF a*(*1*)] *assms*(*4*) **unfolding** *T-def* **by** *auto*

    have *f2*: *finite T* **using** *assms*(*2*) *T-F finite-subset* **by** *auto*
    have *f3*: *finite* (*F − T*) **using** *assms*(*2*) *T-F finite-subset* **by** *auto*
    have *c1*: *card J = card T*
      unfolding *T-def* **using** *assms*(*4*) *inj-on-subset bij-betw-imp-inj-on*[*OF a*(*1*)]
      by (*intro card-image*[*symmetric*]) *auto*
    have *c2*: *card* (*I−J*) = *card* (*F−T*)
     unfolding *x-on-compl*[*symmetric*] **using** *inj-on-subset bij-betw-imp-inj-on*[*OF a*(*1*)]
      by (*intro card-image*[*symmetric*]) *force*

    have *restrict x* (*J ∪* (*I − J*)) = *restrict x I* **using** *assms*(*4*) **by** *force*

also have ... = x **using** *a extensional-restrict* **by** *auto*
finally have *b*:*restrict x* ($J \cup (I - J)$) = x **by** *simp*

have *y*: $y \in$ *extensional J bij-betw y J T*
using *assms*(*4*) *inj-on-subset a y-def* **unfolding** *bij-betw-def T-def* **by** *auto*

have *z* ' ($I-J$) = ($F-T$) **using** *x-on-compl* **unfolding** *z-def* **by** *auto*
hence *z*: $z \in$ *extensional* ($I-J$) *bij-betw z* ($I-J$) ($F-T$)
using *a z-def* **unfolding** *bij-betw-def T-def* **by** (*auto intro*:*inj-on-diff*)

have *pos-assms2*: {*S. card S* = *card J* $\wedge$ $S \subseteq F$} $\neq$ {} *finite* {*S. card S* = *card J* $\wedge$ $S \subseteq F$}
using *T-F c1* **by** (*auto intro*!: *finite-subset*[*OF - iffD2*[*OF finite-Pow-iff assms*(*2*)]]])

note *pos3* =
pmf-of-set[*OF bij-betw-non-empty-finite*(*2*,*1*)[*OF f0 f2 c1*]]
pmf-of-set[*OF bij-betw-non-empty-finite*(*2*,*1*)[*OF f1 f3 c2*]]

have *fin-pmf1*: *finite* (*set-pmf ?p1*) **using** *pos-assms2 set-pmf-of-set* **by** *simp*
note [*simp*] = *integrable-measure-pmf-finite*[*OF fin-pmf1*, **where** '*b*=*real*]

have *fin-pmf2*: *finite* (*set-pmf* (*?p2 T*)) **by** (*intro bij-pmf*[*OF f0 f2 c1*])
note [*simp*] = *integrable-measure-pmf-finite*[*OF fin-pmf2*, **where** '*b*=*real*]

have *fin-pmf3*: *finite* (*set-pmf* (*?p3 T*)) **by** (*intro bij-pmf*[*OF f1 f3 c2*])
note [*simp*] = *integrable-measure-pmf-finite*[*OF fin-pmf3*, **where** '*b*=*real*]

have *pmf ?L x* = *1* / *real* (*card* {*f. bij-betw f I F* $\wedge$ $f \in$ *extensional I*})
using *a pos1* **unfolding** *bij-pmf-def* **by** *simp*
also have ... = *1* / *real* (*fact* (*card I*)) **using** *assms* **by** (*simp add*: *card-bijections*')
also have ... = *1* / *real* (*fact* (*card J*) $*$ *fact* (*card I*$-$*card J*) $*$ (*card I choose card J*))
using *assms*(*1*,*4*) *card-mono* **by** (*subst binomial-fact-lemma*) *auto*
also have ... = *1* / *real* ((*card F choose card J*) $*$ *fact* (*card J*) $*$ *fact* (*card* ($I-J$)))
using *assms*(*3*) *card-Diff-subset*[*OF f0 assms*(*4*)] **by** *simp*
also have ... = *1*/*real*(*card* {*S*. $S \subseteq F \wedge$ *card S*=*card J*} $*$ *card* {*f. bij-betw f J T*$\wedge f \in$*extensional J*} $*$
card {*f. bij-betw f* ($I-J$) ($F-T$)$\wedge f \in$*extensional* ($I-J$)})
using *f0 f1 f2 f3 assms*(*2*) *c1 c2* **by** (*simp add*:*card-bijections*' *n-subsets*)
also have ... = *pmf ?p1 T* $*$ *pmf* (*?p2 T*) *y* $*$ *pmf* (*?p3 T*) *z*
using *y z c1 T-F* **unfolding** *bij-pmf-def pos3 pmf-of-set*[*OF pos-assms2*]
by (*simp add*:*conj-commute*)
also have ... = *pmf ?p1 T* $*$ (*pmf* (*?p2 T*) *y* $*$ (*pmf* (*?p3 T*) *z* $*$ *of-bool*(*merge J* ($I-J$) (*y*, *z*) = *x*)))
unfolding *y-def z-def merge-restrict merge-x-x-eq-restrict b* **by** *simp*
also have ... $\leq$ *pmf ?p1 T* $*$ (*pmf* (*?p2 T*) *y* $*$ ($\int \psi$. *of-bool*(*merge J* ($I-J$) (*y*, ψ) = *x*) ∂*?p3 T*))

by (*intro mult-left-mono expectation-ge-eval-at-point integral-nonneg-AE AE-pmfI*) *simp-all*
also have ... ≤ *pmf ?p1 T* ∗ (∫ φ. (∫ ψ. *of-bool*(*merge J* (*I*−*J*) (φ, ψ) = *x*) ∂*?p3 T*) ∂*?p2 T*)
by (*intro mult-left-mono expectation-ge-eval-at-point integral-nonneg-AE AE-pmfI*) *simp-all*
also have ... ≤ (∫ *S*. (∫ φ. (∫ ψ. *of-bool*(*merge J* (*I*−*J*) (φ, ψ) = *x*) ∂*?p3 S*) ∂*?p2 S*) ∂*?p1*)
by (*intro expectation-ge-eval-at-point integral-nonneg-AE AE-pmfI*) *simp-all*
also have ... = *pmf ?R x* **unfolding** *pmf-bind* **by** (*simp add*:*indicator-def*)
finally show *?thesis* **by** *simp*
next
case *False*
hence *pmf ?L x* = *0* **by** (*simp add*: *set-pmf-iff*)
also have ... ≤ *pmf ?R x* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma *map-bij-pmf*:
assumes *finite I finite F card I* = *card F inj-on* φ *F*
shows *map-pmf* (λ*f*. (λ*x*∈*I*. φ(*f x*))) (*bij-pmf I F*) = *bij-pmf I* (φ ‘ *F*)
proof−
let *?h* = *the-inv-into F* φ

have *h-bij*: *bij-betw ?h* (φ ‘ *F*) *F*
using *assms*(*4*) **by** (*simp add*: *bij-betw-the-inv-into inj-on-imp-bij-betw*)

have *bij-betw* (λ*f*. (λ*x*∈*I*. φ(*f x*)))
{*f*. *bij-betw f I F* ∧ *f* ∈ *extensional I*} {*f*. *bij-betw f I* (φ ‘ *F*) ∧ *f* ∈ *extensional I*}
proof (*intro bij-betwI*[**where** *g*=(λ*f*. (λ*x*∈*I*. *?h*(*f x*)))], *goal-cases*)
case *1* **thus** *?case*
using *bij-betw-trans*[*OF* - *inj-on-imp-bij-betw*[*OF assms*(*4*)], **where** *A*=*I*]
by (*auto simp*:*comp-def*)
next
case *2* **thus** *?case*
using *bij-betw-trans*[*OF* - *h-bij*, **where** *A*=*I*] **by** (*auto simp*:*comp-def*)
next
case (*3 x*)
hence *x* ∈ *I* → *F x* ∈ *extensional I* **using** *bij-betw-imp-surj-on* **by** *auto*
hence (λω∈*I*. *?h* ((λ*y*∈*I*. φ (*x y*)) ω)) ω = *x* ω **for** ω
by (*auto intro*!:*the-inv-into-f-f*[*OF assms*(*4*)] *simp*: *restrict-def extensional-def*)
thus *?case* **by** *auto*
next
case (*4 y*)
hence *y* ∈ *I* → (φ ‘ *F*) *y* ∈ *extensional I* **using** *bij-betw-imp-surj-on* **by** *blast*+
hence (λ*x*∈*I*. φ ((λ*x*∈*I*. *the-inv-into F* φ (*y x*)) *x*)) ω = *y* ω **for** ω
by (*auto intro*!:*f-the-inv-into-f*[*OF assms*(*4*)] *simp*: *restrict-def extensional-def*)

thus *?case* **by** *auto*
qed
thus *?thesis*
unfolding *bij-pmf-def* **by** (*intro map-pmf-of-set-bij-betw bij-betw-non-empty-finite assms*)
qed

lemma *pmf-of-multiset-eq-pmf-of-setI*:
assumes $c > 0$ $x \neq \{\#\}$
assumes $\bigwedge i.\ i \in y \Longrightarrow count\ x\ i = c$
assumes $\bigwedge i.\ i \in\#\ x \Longrightarrow i \in y$
shows *pmf-of-multiset x = pmf-of-set y*
proof (*rule pmf-eqI*)
fix *i*

have *a*:*set-mset x = y* **using** *assms*(*1*,*3*,*4*) *count-eq-zero-iff* **by** *force*
hence *y-ne*: $y \neq \{\}$ *finite y* **using** *assms*(*2*) **by** *auto*

have *size x = sum* (*count x*) *y* **unfolding** *size-multiset-overloaded-eq a* **by** *simp*
also have ... = *sum* (λ-. *c*) *y* **by** (*intro sum.cong refl assms*(*3*)) *auto*
also have ... = *c* ∗*card y* **using** *y-ne* **by** *simp*
finally have *c* ∗ *card y = size x* **by** *simp*
hence *rel*: *real* (*size x*)/*real c = real* (*card y*)
using *assms*(*1*) **by** (*simp add*:*field-simps flip*:*of-nat-mult*)

have *pmf* (*pmf-of-multiset x*) *i = real* (*count x i*) / *real* (*size x*)
using *assms*(*2*) **by** *simp*
also have ... = *real c* ∗ *of-bool*($i \in y$) / *real* (*size x*)
using *assms* **by** (*auto simp*:*of-bool-def count-eq-zero-iff*)
also have ... = *of-bool*($i \in y$) / *real* (*card y*)
unfolding *rel*[*symmetric*] **by** *simp*
also have ... = *pmf* (*pmf-of-set y*) *i*
using *y-ne* **by** *simp*
finally show *pmf* (*pmf-of-multiset x*) *i = pmf* (*pmf-of-set y*) *i* **by** *simp*
qed

lemma *card-multi-bij*:
assumes *finite J*
assumes $I = \bigcup (A\ `\ J)$ *disjoint-family-on A J*
assumes $\bigwedge j.\ j \in J \Longrightarrow$ *finite* (*A j*) ∧ *finite* (*B j*) ∧ *card* (*A j*) = *card* (*B j*)
shows *card* {*f*. (∀ *j*∈*J*. *bij-betw f* (*A j*) (*B j*)) ∧ *f*∈*extensional I*} = ($\prod$ *i*∈*J*. *fact* (*card* (*A i*)))
(**is** *card ?L = ?R*)
proof −
define *g* **where** *g i* = (*THE j*. $j \in J \wedge i \in A\ j$) **for** *i*
have *g*: *g i = j* **if** $i \in A\ j$ $j \in J$ **for** *i j* **unfolding** *g-def*
proof (*rule the1-equality*)
show $\exists !j.\ j \in J \wedge i \in A\ j$
using *assms*(*3*) *that* **unfolding** *bex1-def disjoint-family-on-def* **by** *auto*

show *j* ∈ *J* ∧ *i* ∈ *A j* **using** *that* **by** *auto*
qed

have *bij-betw* (*λφ*. (*λi*∈*I*. *φ* (*g i*) *i*))
(*PiE J* (*λj*. {*f*. *bij-betw f* (*A j*) (*B j*) ∧ *f*∈*extensional* (*A j*)})) *?L*
proof (*intro bij-betwI*[**where** *g*= *λx*. *λi*∈*J*. *restrict x* (*A i*)] *Pi-I*, *goal-cases*)
case (*1 x*)
have *bij-betw* (*λi*∈*I*. *x* (*g i*) *i*) (*A j*) (*B j*) **if** *j* ∈ *J* **for** *j*
proof –
have *last*:*bij-betw* (*x j*) (*A j*) (*B j*) **using** *that 1* **by** *auto*
have *A j* ⊆ *I* **using** *that assms*(*2*) **by** *auto*
thus *?thesis* **using** *g that* **by** (*intro iffD2*[*OF bij-betw-cong last*]) *auto*
qed
thus *?case* **using** *1* **by** *auto*
next
case (*2 x*)
thus *?case* **by** (*intro iffD2*[*OF restrict-PiE-iff*] *ballI*) *simp*
next
case (*3 x*)
have *restrict* (*λi*∈*I*. *x* (*g i*) *i*) (*A j*) = *x j* **if** *j* ∈ *J* **for** *j*
proof –
have *A j* ⊆ *I* **using** *that assms*(*2*) **by** *auto*
moreover have *x j* ∈ *extensional* (*A j*) **using** *that 3* **by** *auto*
hence *restrict* (*λi*. *x* (*g i*) *i*) (*A j*) = *x j*
using *g that* **unfolding** *restrict-def extensional-def* **by** *auto*
ultimately show *?thesis* **unfolding** *restrict-restrict* **using** *Int-absorb1* **by** *metis*
qed
thus *?case* **using** *3* **unfolding** *extensional-def PiE-def* **by** *auto*
next
case (*4 y*)
have (*λj*∈*J*. *restrict y* (*A j*)) (*g i*) *i* = *y i* **if** *that'*:*i* ∈ *I* **for** *i*
proof –
obtain *j* **where** *i* ∈ *A j j* ∈ *J* **using** *that' assms*(*2*) **by** *auto*
thus *?thesis* **using** *g* **by** *simp*
qed
thus *?case* **using** *4* **unfolding** *extensional-def* **by** *auto*
qed

hence *card ?L* = *card* (*PiE J* (*λj*. {*f*. *bij-betw f* (*A j*) (*B j*)∧ *f*∈*extensional* (*A j*)}))
using *bij-betw-same-card*[*symmetric*] **by** *auto*
also have . . . = (∏*i*∈*J*. *card* {*f*. *bij-betw f* (*A i*) (*B i*) ∧ *f* ∈ *extensional* (*A i*)})
unfolding *card-PiE*[*OF assms*(*1*)] **by** *simp*
also have . . . = (∏*i*∈*J*. *fact* (*card* (*A i*)))
using *assms*(*4*) **by** (*intro prod.cong card-bijections'*) *auto*
finally show *?thesis* **by** *simp*
qed

lemma *map-bij-pmf-non-inj*:
fixes *I* :: *'a set*
fixes *F* :: *'b set*
fixes φ :: *'b* ⇒ *'c*
assumes *finite I finite F card I* = *card F*
defines *q* ≡ {*f*. *f* ∈ *extensional I* ∧ {#*f x*. *x* ∈# *mset-set I*#} = {#φ *x*. *x*∈# *mset-set F*#}}
shows *map-pmf* (λ*f*. (λ*x*∈*I*. φ(*f x*))) (*bij-pmf I F*) = *pmf-of-set q* (**is** *?L* = -)
proof −
let *?G* = {# φ *x*. *x* ∈# *mset-set F* #}
let *?G'* = *set-mset ?G*
define *c* :: *nat* **where** *c* = (∏ *i* ∈ *set-mset ?G*. *fact* (*count ?G i*))

note *ne* = *bij-betw-non-empty-finite*[*OF assms*(*1*−*3*)]
note *cim* = *count-image-mset-eq-card-vimage*

have *c* ≥ *1* **unfolding** *c-def* **by** (*intro prod-ge-1*) *auto*
hence *c-gt-0*: *c* > *0* **by** *simp*

have *?L* = *pmf-of-multiset* {#λ*x*∈*I*. φ (*f x*). *f* ∈# *mset-set* {*f*. *bij-betw f I F*∧*f*∈*extensional I*}#}
unfolding *bij-pmf-def* **by** (*intro map-pmf-of-set*[*OF ne*])
also have ... = *pmf-of-set q* **unfolding** *q-def*
proof (*rule pmf-of-multiset-eq-pmf-of-setI*[*OF c-gt-0*],*goal-cases*)
case *1*
have *card* {*f*. *bij-betw f I F* ∧ *f* ∈ *extensional I*} > *0* **using** *ne* **by** *fastforce*
thus *?case* **by** (*simp add*:*nonempty-has-size*)
next
case (*2 f*)

hence *a*: *image-mset f* (*mset-set I*) = *image-mset* φ (*mset-set F*) **by** *simp*
hence *card* {*x* ∈ *F*. φ *x* = *g*} = *card* {*x* ∈ *I*. *f x* = *g*} **for** *g*
using *cim*[*OF assms*(*1*)] *cim*[*OF assms*(*2*)] **by** *metis*
hence *b*: *card* (φ −‘ {*g*} ∩ *F*) = *card* (*f* −‘ {*g*} ∩ *I*) **for** *g*
by (*auto simp add*:*Int-def conj-commute*)

have *c*:*bij-betw* ω *I F* ∧ (λ*i*∈*I*. φ (ω *i*))=*f* ⟷ (∀ *g*∈*?G'*. *bij-betw* ω (*f* −‘{*g*} ∩*I*) (φ−‘{*g*} ∩*F*))
(**is** *?L1* = *?R1*) **for** ω
proof
assume *?L1*
hence *d*:*bij-betw* ω *I F* **and** *e*: ∀ *i* ∈ *I*. φ (ω *i*) = *f i* **by** *auto*
have *bij-betw* ω (*f* −‘{*g*} ∩ *I*) (φ −‘ {*g*} ∩ *F*) **if** *g* ∈ *?G'* **for** *g*
proof −
have *card* (φ −‘ {*g*} ∩ *F*) = *card* (ω ‘ (*f* −‘ {*g*} ∩ *I*))
unfolding *b* **using** *d*
by (*intro card-image*[*symmetric*]) (*simp add*: *bij-betw-imp-inj-on inj-on-Int*)
hence ω ‘ (*f* −‘ {*g*} ∩ *I*) = φ −‘ {*g*} ∩ *F*

using *assms*(*2*) *e* *bij-betw-imp-surj-on*[*OF d*] **by** (*intro card-seteq image-subsetI*) *auto*
thus *?thesis* **by** (*intro bij-betw-subset*[*OF d*]) *auto*
qed
thus *?R1* **by** *auto*
next
assume *f*:*?R1*

have *g*: φ (ω *i*) = *f i* **if** *i* ∈ *I* **for** *i*
proof −
have *f i* ∈ *?G'* **unfolding** *a*[*symmetric*] **using** *that assms*(*1*) **by** *auto*
hence ω ‘ (*f* −‘ {*f i*} ∩ *I*) = (φ −‘ {*f i*} ∩ *F*)
using *bij-betw-imp-surj-on* **using** *f* **by** *metis*
thus *?thesis* **using** *that* **by** (*auto simp add*:*vimage-def*)
qed
have *x* = *y* **if** *x* ∈ *I* *y* ∈ *I* ω *x* = ω *y* **for** *x y*
proof −
have *f x* ∈ *?G'* **unfolding** *a*[*symmetric*] **using** *that assms*(*1*) **by** *auto*
hence *inj-on* ω (*f* −‘ {*f x*} ∩ *I*) **using** *f bij-betw-imp-inj-on* **by** *blast*
moreover have *f x* = *f y* **using** *that g* **by** *metis*
ultimately show *x* = *y* **using** *that*(*1*,*2*,*3*) *inj-onD*[**where** *f*=ω, *OF - that*(*3*)] **by** *fastforce*
qed
hence *h*:*inj-on* ω *I* **by** (*rule inj-onI*)

have *i*: ω ‘ *I* ⊆ *F*
proof (*rule image-subsetI*)
fix *x* **assume** *x* ∈ *I*
hence *f x* ∈ *?G'* *x* ∈ (*f* −‘ {*f x*} ∩ *I*) **using** *assms*(*1*) **unfolding** *a*[*symmetric*] **by** *auto*
thus ω *x* ∈ *F* **using** *bij-betw-imp-surj-on f* **by** *fast*
qed
have *bij-betw* ω *I F*
using *card-image*[*OF h*] *assms*(*3*) **unfolding** *bij-betw-def*
by (*intro conjI card-seteq i h assms*) *auto*
thus *?L1* **using** *g 2* **unfolding** *restrict-def extensional-def* **by** *auto*
qed

have *j*: *f* ‘ *I* ⊆ φ ‘ *F* **using** *a*
by (*metis assms*(*1*,*2*) *finite-set-mset-mset-set multiset.set-map set-eq-subset*)

have *c* = ($\prod$ *g* ∈ *?G'*. *fact* (*card* (*f* −‘ {*g*} ∩ *I*)))
unfolding *b*[*symmetric*] *c-def cim*[*OF assms*(*2*)]
by (*simp add*:*vimage-def Int-def conj-commute*)
also have ... = *card* {ω. (∀ *g* ∈ *?G'*. *bij-betw* ω (*f*−‘{*g*} ∩ *I*) (φ−‘{*g*} ∩ *F*)) ∧ ω ∈ *extensional I*}
using *assms*(*1*,*2*) *j b*
by (*intro card-multi-bij*[*symmetric*]) (*auto simp*: *vimage-def disjoint-family-on-def*)
also have ... = *card* {ω. *bij-betw* ω *I F* ∧ ω ∈ *extensional I* ∧ (λ*i*∈*I*. φ (ω *i*))

= *f*}
using *c* **by** (*intro arg-cong*[**where** *f*=*card*] *Collect-cong*) *auto*
finally show *?case* **using** *ne* **by** (*subst count-image-mset-eq-card-vimage*) *auto*
next
case (*3 f*)
then obtain *u* **where** *u-def*:*bij-betw u I F u* ∈ *extensional I f* = (λ*x*. λ*xa*∈*I*. φ (*x xa*)) *u*
using *ne* **by** *auto*

have *image-mset f* (*mset-set I*) = *image-mset* φ (*image-mset u* (*mset-set I*))
using *assms*(*1*) **unfolding** *u-def*(*3*) *multiset.map-comp* **by** (*intro image-mset-cong*) *auto*
also have ... = *image-mset* φ (*mset-set F*) **using** *image-mset-mset-set u-def*(*1*)
unfolding *bij-betw-def* **by** (*intro arg-cong2*[**where** *f*=*image-mset*] *refl*) *auto*
finally have *image-mset f* (*mset-set I*) = *image-mset* φ (*mset-set F*) **by** *simp*

moreover have *f* ∈ *extensional I* **unfolding** *u-def*(*3*) **by** *auto*
ultimately show *?case* **by** *simp*
qed
finally show *?thesis* **by** *simp*
qed

lemmas *fkg-inequality-pmf-internalized* = *fkg-inequality-pmf*[*unoverload-type 'a*]

lemma *permutation-distributions-are-neg-associated*:
fixes *F* :: (*'a* :: *linorder-topology*) *set*
fixes *I* :: *'b set*
assumes *finite F finite I card I* = *card F*
shows *measure-pmf.neg-assoc* (*bij-pmf I F*) (λ*i* ω. ω *i*) *I*
proof (*rule measure-pmf.neg-assocI2*, *goal-cases*)
case (*1 i*) **thus** *?case* **by** *simp*
next
case (*2 f g J*)

have *fin-J*: *finite J* **using** *2*(*1*) *assms*(*2*) *finite-subset* **by** *metis*
have *fin-I-J*: *finite* (*I*−*J*) **using** *2*(*1*) *assms*(*2*) *finite-subset* **by** *blast*

define *k* **where** *k* = *card J*

have *k-le-F*: *k* ≤ *card F* **unfolding** *k-def* **using** *2*(*1*) *assms*(*2*,*3*) *card-mono* **by** *force*

let *?p0* = *bij-pmf I F*
let *?p1* = *pmf-of-set* {*S*. *card S* = *card J* ∧ *S* ⊆ *F*}
let *?p2* = λ*S*. *bij-pmf J S*
let *?p3* = λ*S*. *bij-pmf* (*I* − *J*) (*F* − *S*)

note *set-pmf-p0* = *bij-pmf*[*OF assms*(*2*,*1*,*3*)]

note *integrable-p0*[*simp*] = *integrable-measure-pmf-finite*[*OF set-pmf-p0*(*2*), **where** $'b$=*real*]

note *dep-f* = *2*(*2*)
note *dep-g* = *2*(*3*)

have *bounded-f*: *bounded* (*f* ‘ *S*) **for** *S* **using** *bounded-subset*[*OF 2*(*6*) *image-mono*] **by** *simp*
have *bounded-g*: *bounded* (*g* ‘ *S*) **for** *S* **using** *bounded-subset*[*OF 2*(*7*) *image-mono*] **by** *simp*

note *mono-f* = *2*(*4*)
note *mono-g* = *2*(*5*)

let *?L* = *ordered-set-lattice F k*

define f' **where** f' *S* = ($\int$ φ. *f* φ ∂*?p2 S*) **for** *S*
define g' **where** g' *S* = ($\int$ φ. *g* φ ∂*?p3 S*) **for** *S*

interpret *L*: *finite-ne-distrib-lattice ordered-set-lattice F k*
by (*intro ordered-set-lattice-lattice assms*(*1*) *k-le-F*)

have *carr-L-ne*: *carrier ?L* $\neq$ {} **and** *fin-L*: *finite* (*carrier ?L*)
using *ordered-set-lattice-carrier-finite-ne*[*OF assms*(*1*) *k-le-F*] **by** *auto*

have *mono-f'*: *monotone-on* (*carrier ?L*) ($\sqsubseteq_{?L}$) ($\leq$) f'
proof (*rule monotone-onI*)
fix *S T*
assume *a*:*S* $\sqsubseteq_{?L}$ *T S* $\in$ *carrier ?L T* $\in$ *carrier ?L*
then obtain ϱ **where** ϱ*-bij*: *bij-betw* ϱ *S T* **and** ϱ*-inc*: $\bigwedge$*e*. ϱ *e* $\geq$ *e*
using *bij-betw-ord-set-lattice-pairs*[*OF assms*(*1*) *k-le-F*] **by** *blast*

note *S-carr* = *ordered-set-lattice-carrier*[*OF a*(*2*)]
have *c*:*card J* = *card S* **using** *S-carr k-def* **by** *auto*

note *set-pmf-p2* = *bij-pmf*[*OF fin-J S-carr*(*1*) *c*]
note *int* = *integrable-measure-pmf-finite*[*OF set-pmf-p2*(*2*)]

have f' *S* = ($\int$ φ. *f* ($\lambda\omega\in J$. φ ω) ∂*?p2 S*) **unfolding** f'*-def*
using *set-pmf-p2 extensional-restrict* **by** (*intro integral-cong-AE AE-pmfI*) *force+*
also have ... $\leq$ ($\int$ φ. *f* ($\lambda\omega\in J$. ϱ(φ ω)) ∂*?p2 S*) **unfolding** f'*-def*
using ϱ*-inc* **unfolding** *restrict-def*
by (*intro integral-mono-AE AE-pmfI monoD*[*OF mono-f*] *int*) (*auto simp*: *le-fun-def*)
also have ... = ($\int$ φ. *f* φ ∂(*map-pmf* ($\lambda\varphi$. ($\lambda\omega\in J$. ϱ(φ ω))) (*?p2 S*))) **by** *simp*
also have ... = ($\int$ φ. *f* φ ∂(*?p2* (ϱ ‘ *S*)))
using *ordered-set-lattice-carrier*[*OF a*(*2*)] *k-def*
by (*intro arg-cong2*[**where** *f*=*measure-pmf.expectation*] *map-bij-pmf refl*

bij-betw-imp-inj-on[*OF ϱ-bij*] *fin-J*) *auto*
also have ... = ($\int \varphi$. *f* φ ∂*?p2 T*) **using** *bij-betw-imp-surj-on*[*OF ϱ-bij*] **by** *simp*
finally show f' *S* $\leq$ f' *T* **unfolding** *f′-def* **by** *simp*
qed

have *mono-g′*: *monotone-on* (*carrier ?L*) ($\sqsubseteq_{?L}$) ($\leq$) ((*)(−*1*) ∘ g')
proof (*rule monotone-onI*)
fix *S T*
let *?M* = *ordered-set-lattice F* (*card F*−*k*)
assume *a*:*S* $\sqsubseteq_{?L}$ *T S* ∈ *carrier ?L T* ∈ *carrier ?L*
hence *a′*: (*F*−*T*) $\sqsubseteq_{?M}$ (*F*−*S*) (*F*−*S*) ∈ *carrier ?M* (*F*−*T*) ∈ *carrier ?M*
using *ordered-set-lattice-dual*[*OF assms*(*1*) *k-le-F*] **by** *auto*
then obtain *ϱ* **where** *ϱ-bij*: *bij-betw ϱ* (*F*−*T*) (*F*−*S*) **and** *ϱ-inc*: $\bigwedge$*e*. *ϱ e* ≥ *e*
using *bij-betw-ord-set-lattice-pairs*[*OF assms*(*1*)] *k-le-F* **by** (*meson diff-le-self*)
note *T-carr* = *ordered-set-lattice-carrier*[*OF a′*(*3*)]

have *c*: *card* (*I*−*J*) = *card* (*F*−*T*)
using *assms ordered-set-lattice-carrier*[*OF a*(*3*)] *k-def 2*(*1*) *fin-J*
by (*simp add*: *card-Diff-subset*)
note *set-pmf-p3* = *bij-pmf*[*OF fin-I-J T-carr*(*1*) *c*]
note *int* = *integrable-measure-pmf-finite*[*OF set-pmf-p3*(*2*)]

have g' *T* = ($\int \varphi$. *g* (λω∈*I*−*J*. φ ω) ∂*?p3 T*) **unfolding** *g′-def*
using *set-pmf-p3 extensional-restrict* **by** (*intro integral-cong-AE AE-pmfI*) *force+*
also have ... ≤ ($\int \varphi$. *g* (λω∈*I*−*J*. *ϱ*(φ ω)) ∂*?p3 T*) **unfolding** *g′-def restrict-def* **using** *ϱ-inc*
by (*intro integral-mono-AE AE-pmfI monoD*[*OF mono-g*] *int*) (*auto simp*: *le-fun-def*)
also have ... = ($\int \varphi$. *g* φ ∂(*map-pmf* (λφ. (λω∈*I*−*J*. *ϱ*(φ ω))) (*?p3 T*))) **by** *simp*
also have ... = ($\int \varphi$. *g* φ ∂(*bij-pmf* (*I* − *J*) (*ϱ* ‘ (*F*−*T*)))) **using** *assms*
by (*intro arg-cong2*[**where** *f*=*measure-pmf.expectation*] *map-bij-pmf refl*
bij-betw-imp-inj-on[*OF ϱ-bij*] *fin-J c*) *auto*
also have ... = ($\int \varphi$. *g* φ ∂*?p3 S*) **using** *bij-betw-imp-surj-on*[*OF ϱ-bij*] **by** *simp*
finally have g' *T* ≤ g' *S* **unfolding** *g′-def* **by** *simp*
thus ((*) (− *1*) ∘ g') *S* ≤ ((*) (− *1*) ∘ g') *T* **by** *simp*
qed

have ($\int$ *S*. f' *S* * g' *S* ∂*?p1*) ≤ ($\int$ *S*. f' *S* ∂*?p1*) * ($\int$ *S*. g' *S* ∂*?p1*)
if *td*: ∃(*Rep* :: ′*x* ⇒ ′*a set*) *Abs*. *type-definition Rep Abs* (*carrier ?L*)
proof −
obtain *Rep* :: ′*x* ⇒ ′*a set* **and** *Abs* **where** *td*:*type-definition Rep Abs* (*carrier ?L*)
using *td* **by** *auto*
interpret *type-definition Rep Abs carrier ?L* **using** *td* **by** *auto*

have *carr-L*: *carrier ?L* = {*S*. *card S* = *card J* ∧ *S* ⊆ *F*}
using *finite-subset*[*OF* - *assms*(*1*)] **unfolding** *ordered-set-lattice-def k-def*
by (*auto simp add*:*set-eq-iff*)

have *Rep-bij*: *bij-betw Rep UNIV* {*S*. *card S* = *card J* ∧ *S* ⊆ *F*}
using *Rep-range Rep-inject carr-L* **unfolding** *bij-betw-def* **by** (*intro conjI inj-onI*) *auto*

have *fin-UNIV*: *finite* (*UNIV* :: *'x set*)
using *fin-L carr-L Rep-bij bij-betw-finite* **by** *metis*

let *?p1'* = *pmf-of-set* (*UNIV* :: *'x set*)
have *rep-p1*: *?p1* = *map-pmf Rep ?p1'*
by (*intro UNIV-not-empty map-pmf-of-set-bij-betw*[*symmetric*] *Rep-bij fin-UNIV*)

note ∗ = *L.transfer-to-type*[*OF fin-L td*]

note *fkg* = *fkg-inequality-pmf-internalized*[*OF* ∗]

have *mono-rep-f'*: *monotone* (*λS T*. *Rep S* $\sqsubseteq_{?L}$ *Rep T*) (≤) (*f'* ∘ *Rep*)
using *mono-f' Rep* **unfolding** *monotone-on-def* **by** *simp*
have *mono-rep-g'*: *monotone* (*λS T*. *Rep S* $\sqsubseteq_{?L}$ *Rep T*) (≥) (*g'* ∘ *Rep*)
using *mono-g' Rep* **unfolding** *monotone-on-def* **by** *simp*
have *pmf-const*: *pmf ?p1' x* = *1*/(*real* (*CARD*(*'x*))) **for** *x*
by (*subst pmf-of-set*[*OF* - *fin-UNIV*]) *auto*

have (∫ *S*. *f' S* ∗ *g' S* ∂*?p1*) = (∫ *S*. *f'* (*Rep S*) ∗ *g'* (*Rep S*) ∂*?p1'*)
unfolding *rep-p1* **by** *simp*
also have ... ≤ (∫ *S*. *f'* (*Rep S*) ∂*?p1'*) ∗ (∫ *S*. *g'* (*Rep S*) ∂*?p1'*)
using *mono-rep-f' mono-rep-g'*
by (*intro fkg*[**where** *τ*=*Fwd* **and** *σ*=*Rev*, *simplified*]) (*simp-all add*:*comp-def pmf-const*)
also have ... = (∫ *S*. *f' S* ∂*?p1*) ∗ (∫ *S*. *g' S* ∂*?p1*)
unfolding *rep-p1* **by** *simp*
finally show (∫ *S*. *f' S* ∗ *g' S* ∂*?p1*) ≤ (∫ *S*. *f' S* ∂*?p1*) ∗ (∫ *S*. *g' S* ∂*?p1*) **by** *simp*
qed

note *core-result* = *this*[*cancel-type-definition*, *OF carr-L-ne*]

note *split-p0* = *split-bij-pmf*[*OF assms*(*2*,*1*,*3*) *2*(*1*)]

have (∫ *x*. *f x* ∗ *g x* ∂*bij-pmf I F*) =
(∫ *S*. (∫ *φ*. (∫ *ψ*. *f*(*merge J* (*I*−*J*) (*φ*,*ψ*))∗*g*(*merge J* (*I*−*J*) (*φ*,*ψ*)) ∂*?p3 S*) ∂*?p2 S*) ∂*?p1*)
unfolding *k-def* **by** (*simp add*:*split-p0 bounded-intros bounded-f bounded-g integral-bind-pmf*)
also have ... = (∫ *S*. (∫ *φ*. (∫ *ψ*. *f φ*∗*g ψ* ∂*?p3 S*) ∂*?p2 S*) ∂*?p1*)
by (*intro integral-cong-AE AE-pmfI arg-cong2*[**where** *f*=(∗)] *depends-onD2*[*OF*

dep-f]
    depends-onD2[*OF dep-g*]) *simp-all*
  also have ... = (∫ *S*. (∫ *φ*. *f φ* ∂*?p2 S*) ∗ (∫ *ψ*. *g ψ* ∂*?p3 S*) ∂*?p1*) **by** *simp*
  also have ... ≤ (∫ *S*. (∫ *φ*. *f φ* ∂*?p2 S*) ∂*?p1*) ∗ (∫ *S*. (∫ *φ*. *g φ* ∂*?p3 S*) ∂*?p1*)
    using *core-result* **unfolding** *f′-def g′-def* **by** *simp*
  also have ... = (∫ *S*.(∫ *φ*.(∫ *ψ*. *f φ* ∂*?p3 S*) ∂*?p2 S*) ∂*?p1*) ∗ (∫ *S*.(∫ *φ*.(∫ *ψ*. *g ψ* ∂*?p3 S*) ∂*?p2 S*) ∂*?p1*)
    by *simp*
  also have ... =
    (∫ *S*. (∫ *φ*. (∫ *ψ*. *f* (*merge J* (*I*−*J*) (*φ*,*ψ*)) ∂*?p3 S*) ∂*?p2 S*) ∂*?p1*) ∗
    (∫ *S*. (∫ *φ*. (∫ *ψ*. *g* (*merge J* (*I*−*J*) (*φ*,*ψ*)) ∂*?p3 S*) ∂*?p2 S*) ∂*?p1*)
    by (*intro arg-cong2*[**where** *f*=(∗)] *integral-cong-AE AE-pmfI depends-onD2*[*OF dep-f*]
    depends-onD2[*OF dep-g*]) *simp-all*
  also have ... = (∫ *x*. *f x* ∂*?p0*) ∗ (∫ *x*. *g x* ∂*?p0*)
    unfolding *k-def* **by** (*simp add*:*split-p0 bounded-intros bounded-f bounded-g integral-bind-pmf*)
  finally show (∫ *x*. *f x* ∗ *g x* ∂*?p0*) ≤ (∫ *x*. *f x* ∂*?p0*)∗(∫ *x*. *g x* ∂*?p0*) **by** *simp*
qed

lemma *multiset-permutation-distributions-are-neg-associated*:
  fixes *F* :: (′*a* :: *linorder-topology*) *multiset*
  fixes *I* :: ′*b set*
  assumes *finite I card I* = *size F*
  defines *p* ≡ *pmf-of-set* {*φ*. *φ* ∈ *extensional I* ∧ *image-mset φ* (*mset-set I*) = *F*}
  shows *measure-pmf.neg-assoc p* (*λi ω*. *ω i*) *I*
proof −
  let *?xs* = *sorted-list-of-multiset F*
  define *α* **where** *α k* = *?xs* ! (*min k* (*length ?xs* −*1*)) **for** *k*

  let *?N* = {..<*size F*}
  let *?h* = (*λf*. (*λi*∈*I*. *α* (*f i*)))

  have *sorted-xs*: *sorted ?xs* **by** (*induction F*, *auto simp*:*sorted-insort*)

  have *mono-α*: *mono α*
  proof (*cases ?xs* = [])
    case *True* **thus** *?thesis* **unfolding** *α-def* **by** *simp*
  next
    case *False* **thus** *?thesis* **unfolding** *α-def*
    by (*intro monoI sorted-nth-mono*[*OF sorted-xs*]) (*simp-all add*: *min.strict-coboundedI2*)
  qed

  have *l-xs*: *length ?xs* = *size F* **by** (*metis mset-sorted-list-of-multiset size-mset*)

  have *image-mset α* (*mset-set* {..<*size F*}) = *image-mset* ((!) *?xs*) (*mset-set* {..<*size F*})
    unfolding *α-def l-xs*[*symmetric*] **by** (*intro image-mset-cong*) *auto*
  also have ... = *mset ?xs* **unfolding** *l-xs*[*symmetric*]

by (*metis map-nth mset-map mset-set-upto-eq-mset-upto*)
also have … = *F* **by** *simp*
finally have *0*:*image-mset* α (*mset-set* {..<*size F*}) = *F* **by** *simp*

have *map-pmf* (λ*f*. (λ*i*∈*I*. α (*f i*))) (*bij-pmf I ?N*) =
pmf-of-set {*f* ∈ *extensional I*. *image-mset f* (*mset-set I*) = *image-mset* α (*mset-set* {..<*size F*})}
using *assms* **by** (*intro map-bij-pmf-non-inj*) *auto*
also have … = *p* **unfolding** *p-def 0* **by** *simp*
finally have *1*:*map-pmf* (λ*f*. (λ*i*∈*I*. α (*f i*))) (*bij-pmf I ?N*) = *p* **by** *simp*

have *2*:*measure-pmf.neg-assoc* (*bij-pmf I* {..<*size F*}) (λ*i* ω. ω *i*) *I*
using *assms*(*1*,*2*) **by** (*intro permutation-distributions-are-neg-associated*) *auto*

have *measure-pmf.neg-assoc* (*bij-pmf I* {..<*size F*}) (λ*i* ω. *if i* ∈ *I then* α(ω *i*) *else undefined*) *I*
using *mono-*α **by** (*intro measure-pmf.neg-assoc-compose-simple*[*OF assms*(*1*) *2*, **where** η=*Fwd*]
borel-measurable-continuous-onI) *simp-all*
hence *measure-pmf.neg-assoc* (*map-pmf* (λ*f*. (λ*i*∈*I*. α(*f i*))) (*bij-pmf I* {..<*size F*})) (λ*i* ω. ω *i*) *I*
by (*simp add*:*neg-assoc-map-pmf restrict-def if-distrib if-distribR*)
thus *?thesis* **unfolding** *1* **by** *simp*
qed

lemma *n-subsets-prob*:
assumes *d* ≤ *card S finite S s* ∈ *S*
shows
measure-pmf.prob (*pmf-of-set* {*a*. *a* ⊆ *S* ∧ *card a* = *d*}) {ω. *s* ∉ ω} = (*1* − *real d*/*card S*)
measure-pmf.prob (*pmf-of-set* {*a*. *a* ⊆ *S* ∧ *card a* = *d*}) {ω. *s* ∈ ω} = *real d*/*card S*
proof −
let *?C* = {*a*. *a* ⊆ *S* ∧ *card a* = *d*}

have *card ?C* > *0* **unfolding** *n-subsets*[*OF assms*(*2*)] **using** *zero-less-binomial*[*OF assms*(*1*)] **by** *simp*
hence *ne*:*?C* ≠ {} *finite ?C* **using** *card-gt-0-iff* **by** *blast+*

have *card-S-gt-0*: *card S* > *0* **using** *assms*(*2*,*3*) *card-gt-0-iff* **by** *auto*

have *measure* (*pmf-of-set ?C*) {*x*. *s* ∉ *x*} = *real* (*card* {*T*. *T*⊆*S* ∧ *card T* = *d* ∧ *s* ∉ *T*}) / *card ?C*
by (*subst measure-pmf-of-set*[*OF ne*]) (*simp-all add*:*Int-def*)
also have … = *real* (*card* {*T*. *T*⊆(*S*−{*s*}) ∧ *card T* = *d*}) / *card ?C*
by (*intro arg-cong2*[**where** *f*=(λ*x y*. *real* (*card x*)/*y*)] *Collect-cong*) *auto*
also have … = *real*(*card* (*S* − {*s*}) *choose d*) / *real* (*card S choose d*)
using *assms*(*1*,*2*) **by** (*subst* (*1 2*) *n-subsets*) *auto*
also have … = *real*((*card S* − *1*) *choose d*) / *real* (*card S choose d*) **using**

assms **by** *simp*
also have ... = *real*(*card S* ∗((*card S*−*1*) *choose d*)) / *real* (*card S* ∗ (*card S* *choose d*))
using *card-S-gt-0* **by** *simp*
also have ... = *real* (*card S* − *d*) / *real* (*card S*)
unfolding *binomial-absorb-comp*[*symmetric*] **by** *simp*
also have ... = (*real* (*card S*) − *real d*) / *real* (*card S*)
using *assms* **by** (*subst of-nat-diff*) *auto*
also have ... = (*1* − *real d*/*card S*) **using** *card-S-gt-0* **by** (*simp add*:*field-simps*)
finally show *measure* (*pmf-of-set ?C*) {*x*. *s* ∉ *x*} = (*1* − *real d*/*card S*) **by** *simp*

hence ‹*1*−*measure* (*pmf-of-set ?C*) {*x*. *s* ∉ *x*} = *real d*/*card S*› **by** *simp*
thus *measure-pmf.prob* (*pmf-of-set ?C*) {ω. *s* ∈ ω} = *real d*/*card S*
by (*subst* (*asm*) *measure-pmf.prob-compl*[*symmetric*]) (*auto simp*:*diff-eq Compl-eq*)
qed

lemma *n-subsets-distribution-neg-assoc*:
assumes *finite S k* ≤ *card S*
defines *p* ≡ *pmf-of-set* {*T*. *T* ⊆ *S* ∧ *card T* = *k*}
shows *measure-pmf.neg-assoc p* (∈) *S*
proof −
define *F* :: *bool multiset* **where** *F* = *replicate-mset k True* + *replicate-mset* (*card S* − *k*) *False*
let *?qset* = { φ ∈ *extensional S*. *image-mset* φ (*mset-set S*) = *F* }
define *q* **where** *q* = *pmf-of-set ?qset*

have *a*: *card S* = *size F* **unfolding** *F-def* **using** *assms*(*2*) **by** *simp*

have *b*: *image-mset* φ (*mset-set S*) = *F* ⟷ *card* (φ −‘ {*True*} ∩ *S*) = *k*
(**is** *?L* ⟷ *?R*) **for** φ
proof −
have *de*:*card* (φ−‘{*False*}∩*S*) + *card* (φ−‘{*True*}∩*S*) = *card S*
using *assms*(*1*) **by** (*subst card-Un-disjoint*[*symmetric*]) (*auto intro*:*arg-cong*[**where** *f*=*card*])

have *?L* ⟷ (∀ *i*. *count* {#φ *x*. *x*∈#*mset-set S*#} *i* = *count F i*) **using** *multiset-eq-iff* **by** *blast*
also have ... ⟷ (∀ *i*. *card* (φ −‘ {*i*} ∩ *S*) = *count F i*)
unfolding *count-image-mset-eq-card-vimage*[*OF assms*(*1*)] *vimage-def Int-def*
by (*simp add*:*conj-commute*)
also have ... ⟷ *card* (φ −‘ {*True*} ∩ *S*) = *k* ∧ *card* (φ −‘ {*False*} ∩ *S*) = (*card S*−*k*)
unfolding *F-def* **using** *assms*(*1*) **by** *auto*
also have ... ⟷ *?R* **using** *assms*(*2*) *de* **by** *auto*
finally show *?thesis* **by** *simp*
qed

have *bij-betw* (λω. λ*s*∈*S*. *s*∈ω) {*T*. *T*⊆*S* ∧ *card T* = *k*} *?qset* **unfolding** *b*
by (*intro bij-betwI*[**where** *g*=λφ. {*x*. *x* ∈ *S* ∧ φ *x*}] *Pi-I ext*)

(*auto intro*: *arg-cong*[**where** *f*=*card*] *simp*:*extensional-def vimage-def Int-def conj-commute*)
moreover have *card* {*T*. *T* ⊆ *S* ∧ *card T* = *k*} > *0*
unfolding *n-subsets*[*OF assms*(*1*)] **by** (*intro zero-less-binomial assms*(*2*))
hence {*T*. *T* ⊆ *S* ∧ *card T* = *k*} ≠ {} ∧ *finite* {*T*. *T* ⊆ *S* ∧ *card T* = *k*}
using *card-gt-0-iff* **by** *blast*
ultimately have *c*: *map-pmf* (λω. λs∈*S*. *s*∈ω) *p* = *q*
unfolding *p-def q-def* **by** (*intro map-pmf-of-set-bij-betw*) *auto*

have *measure-pmf.neg-assoc* (*map-pmf* (λω. λs∈*S*. *s*∈ω) *p*) (λ*i* ω. ω *i*) *S*
unfolding *c q-def* **by** (*intro multiset-permutation-distributions-are-neg-associated a assms*(*1*))
hence *d*:*measure-pmf.neg-assoc p* (λ*s* ω. *if s* ∈ *S then* (*s* ∈ ω) *else undefined*) *S*
unfolding *neg-assoc-map-pmf* **by** (*simp add*:*restrict-def cong*:*if-cong*)
show *?thesis* **by** (*intro measure-pmf.neg-assoc-cong*[*OF assms*(*1*) - *d*] *AE-pmfI*) *auto*
qed

end

7 Application: Bloom Filters

The false positive probability of Bloom Filters is a case where negative association is really useful. Traditionally it is derived only approximately. Bloom [4] first derives the expected number of bits set to true given the number of elements inserted, then the false positive probability is computed, pretending that the expected number of bits is the actual number of bits.

Both Blooms original derivation and Mitzenmacher and Upfal [15] use this method.

A more correct approach would be to derive a tail bound for the number of set bits and derive a false-positive probability based on that, which unfortunately leads to a complex formula.

An exact result has later been derived using combinatorial methods by Gopinathan and Sergey [10]. However their formula is less useful, as it consists of a sum with Stirling numbers and binomial coefficients.

It is however easy to see that the original bound derived by Bloom is a correct upper bound for the false positive probability using negative association. (This is pointed out by Bao et al. [**?**].)

In this section, we derive the same bound using this library as an example for the applicability of this library.

theory *Negative-Association-Bloom-Filters*
imports *Negative-Association-Permutation-Distributions*
begin

fun *bloom-filter-pmf* **where**

bloom-filter-pmf 0 d N = *return-pmf* {} |
bloom-filter-pmf (*Suc n*) *d N* = *do* {
  h ← *bloom-filter-pmf n d N*;
  a ← *pmf-of-set* {*a*. *a* ⊆ {..<(*N*::*nat*)} ∧ *card a* = *d*};
  return-pmf (*a* ∪ *h*)
}

lemma *bloom-filter-neg-assoc*:
assumes *d* ≤ *N*
shows *measure-pmf.neg-assoc* (*bloom-filter-pmf n d N*) (λ*i* ω. *i* ∈ ω) {..<*N*}
proof (*induction n*)
case *0*

have *a*:*measure-pmf.neg-assoc* (*bloom-filter-pmf 0 d N*) (λ- -. *False*) {..<*N*}
by (*intro measure-pmf.indep-imp-neg-assoc measure-pmf.indep-vars-const*) *auto*
show *?case* **by** (*intro measure-pmf.neg-assoc-cong*[*OF - - a*] *AE-pmfI*) *simp-all*
next
case (*Suc n*)
let *?l* = *bloom-filter-pmf n d N*
let *?r* = *pmf-of-set* {*a*. *a* ⊆ {..<*N*} ∧ *card a* = *d*}

define *f* **where** *f j* ω = (ω (*True*,*j*) ∨ ω (*False*,*j*)) **for** ω **and** *j* :: *nat*

have *f-borel*: *f i* ∈ *borel-measurable* (Pi_M (*UNIV* × {*i*}) (λ-. *borel*)) (**is** *?L* ∈ *?R*) **for** *i*
proof −
have *f i* = (λω. *max*(*fst* ω) (*snd* ω)) ∘ (λω. (ω (*True*,*i*),ω (*False*,*i*))) **unfolding** *f-def* **by** *auto*
also have ... ∈ *?R* **by** (*intro measurable-comp*[**where** *N*=*borel* $\bigotimes_M$ *borel*]) *measurable*
finally show *?thesis* **by** *simp*
qed

have *0*:{*True*} ×{..<*N*} ∪ {*False*} ×{..<*N*} = *UNIV*×{..<*N*} **by** *auto*

have *s*:{*b*} × {..<*N*} = *Pair b* ‘ {..<*N*} **for** *b* :: *bool* **by** *auto*

have *measure-pmf.neg-assoc* (*map-pmf snd* (*pair-pmf ?l ?r*)) (λ*i* ω. *i* ∈ ω) ({..<*N*})
unfolding *map-snd-pair-pmf* **using** *assms* **by** (*intro n-subsets-distribution-neg-assoc*) *auto*
hence *na-l*:
measure-pmf.neg-assoc (*pair-pmf ?l ?r*) (λ*i* ω. *snd i* ∈ *case-bool fst snd* (*fst i*) ω) ({*False*} × {..<*N*})
unfolding *s neg-assoc-map-pmf* **by** (*subst measure-pmf.neg-assoc-reindex*) (*auto intro*:*inj-onI*)

have *measure-pmf.neg-assoc* (*map-pmf fst* (*pair-pmf ?l ?r*)) (∈) ({..<*N*})
unfolding *map-fst-pair-pmf* **using** *Suc* **by** *simp*

hence na-r:
measure-pmf.neg-assoc (pair-pmf ?l ?r) (λi ω. snd i ∈ case-bool fst snd (fst i) ω) ({True} × {..<N})
unfolding s neg-assoc-map-pmf **by** (subst measure-pmf.neg-assoc-reindex) (auto intro:inj-onI)

have c: prob-space.indep-var (pair-pmf ?l ?r)
(PiM ({True} × {..<N}) (λ-. borel)) x (PiM ({False} × {..<N}) (λ-. borel)) y
if x = ((λω. λi∈{True} × {..<N}. snd i∈ ω)∘fst) y=((λω. λi∈{False} × {..<N}. snd i ∈ ω)∘snd)
for x y
unfolding that **by** (intro prob-space.indep-var-compose[OF - indep-var-pair-pmf] prob-space-measure-pmf)
(auto simp:space-PiM)

have a:measure-pmf.neg-assoc (pair-pmf ?l ?r) (λi ω. snd i ∈ case-bool fst snd (fst i) ω) (UNIV × {..<N})
by (intro measure-pmf.neg-assoc-combine[OF - 0] na-l na-r c) (auto simp: restrict-def mem-Times-iff)
have measure-pmf.neg-assoc (pair-pmf ?l ?r) (λi ω. f i (λi. snd i ∈ case-bool fst snd (fst i) ω)) {..<N}
by (intro measure-pmf.neg-assoc-compose[OF - a, **where** deps=λj. UNIV×{j} **and** η=Fwd]
monotoneI depends-onI f-borel) (auto simp:f-def)
hence measure-pmf.neg-assoc (pair-pmf ?l ?r) (λi ω. i ∈ fst ω ∨ i ∈ snd ω) {..<N}
unfolding f-def **by** (simp add:case-prod-beta′)
hence measure-pmf.neg-assoc (map-pmf (case-prod (∪)) (pair-pmf ?l ?r)) (∈) {..<N}
unfolding neg-assoc-map-pmf **by** (simp add:case-prod-beta′)
thus ?case **by** (simp add:pair-pmf-def map-bind-pmf Un-commute)
qed

lemma bloom-filter-cell-prob:
assumes d ≤ N i < N
shows measure (bloom-filter-pmf n d N) {ω. i ∈ ω} = 1 − (1 − real d/real N)^n
proof −
have measure (bloom-filter-pmf n d N) {ω. i ∉ ω} = (1 − real d/real N)^n
proof (induction n)
case 0 **thus** ?case **by** simp
next
case (Suc n)
let ?p = pair-pmf (bloom-filter-pmf n d N) (pmf-of-set {a. a ⊆ {..<N} ∧ card a = d})

have a: {ω. i ∉ fst ω ∧ i ∉ snd ω} = ({ω. i ∉ ω}) × ({ω. i ∉ ω}) **by** auto

have measure ?p {ω. i ∉ fst ω ∧ i ∉ snd ω} = (1−real d/N) ^ n ∗ (1−real

d/card {..<*N*})
      using *assms* **unfolding** *a measure-pair-pmf*
      by (*intro Suc n-subsets-prob*(*1*) *arg-cong2*[**where** *f*=(∗)]) *auto*
    also have ... = (*1*−*real d*/*N*) ^ (*n*+*1*) **by** *simp*
    finally have *measure ?p* {ω. *i* ∉ *fst* ω ∧ *i* ∉ *snd* ω} = (*1*−*real d*/*N*) ^ (*n*+*1*)
by *simp*

    hence *measure* (*map-pmf* (λω. *snd* ω ∪ *fst* ω) *?p*) {ω. *i* ∉ ω} = (*1*−*real*
d/*N*)^(*n*+*1*)
      by (*simp add*:*disj-commute*)
    thus *?case* **by** (*simp add*:*pair-pmf-def map-bind-pmf*)
  qed
  hence *1* − *measure* (*bloom-filter-pmf n d N*) {ω. *i* ∈ ω} = (*1* − *real d*/*real N*)^*n*
    by (*subst measure-pmf.prob-compl*[*symmetric*]) (*auto simp*:*set-diff-eq*)
  thus *?thesis* **by** *simp*
qed

lemma *bloom-filter-false-positive-prob*:
  assumes *d* ≤ *N T* ⊆ {..<*N*} *card T* = *d*
  shows *measure* (*bloom-filter-pmf n d N*) {ω. *T* ⊆ ω} ≤ (*1* − (*1* − *real d*/*real*
N)^*n*)^*d*
    (**is** *?L* ≤ *?R*)
proof −
  let *?p* = *bloom-filter-pmf n d N*
  have *na*: *measure-pmf.neg-assoc* (*bloom-filter-pmf n d N*) (λ*i* ω. *i* ∈ ω) *T*
    by (*intro measure-pmf.neg-assoc-subset*[*OF assms*(*2*) *bloom-filter-neg-assoc*]
assms(*1*))

  have *fin-T*: *finite T* **using** *assms*(*2*) *finite-subset* **by** *auto*
  hence *a*: *of-bool* (*T* ⊆ *y*) = (∏*t*∈*T*. *of-bool* (*t* ∈ *y*)::*real*) **for** *y*
    by (*induction T*) *auto*

  have *?L* = *measure ?p* ({ω. *T* ⊆ ω} ∩ *space ?p*) **by** *simp*
  also have ... = (∫ω. (∏*t* ∈ *T*. *of-bool*(*t* ∈ ω)) ∂*?p*)
    unfolding *Bochner-Integration.integral-indicator*[*symmetric*] *indicator-def*
    using *a* **by** (*intro integral-cong-AE AE-pmfI*) *auto*
  also have ... ≤ (∏*t* ∈ *T*. (∫ω. *of-bool*(*t* ∈ ω) ∂*?p*))
    by (*intro has-int-thatD*(*2*)[*OF measure-pmf.neg-assoc-imp-prod-mono*[*OF - na*,
where η=*Fwd*]]
        integrable-bounded-pmf bounded-range-imp[*OF bounded-of-bool*] *fin-T*
        borel-measurable-continuous-onI) (*auto intro*:*monoI*)
  also have ... = (∏*t* ∈ *T*. *measure ?p* ({ω. *t* ∈ ω} ∩ *space ?p*))
    unfolding *Bochner-Integration.integral-indicator*[*symmetric*] *indicator-def* **by**
simp
  also have ... = (∏*t* ∈ *T*. *measure ?p* {ω. *t* ∈ ω}) **by** *simp*
  also have ... = (∏*t* ∈ *T*. *1* − (*1* − *real d*/*real N*)^*n*)
    using *assms*(*1*,*2*) **by** (*intro prod.cong bloom-filter-cell-prob*) *auto*
  also have ... = *?R* **using** *assms*(*3*) **by** *simp*
  finally show *?thesis* **by** *simp*

qed

end

References

[1] R. Ahlswede and D. E. Daykin. An inequality for the weights of two families of sets, their unions and intersections. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 43:183–185, 1978.

[2] N. Alon and J. H. Spencer. *The Probabilistic Method, Second Edition*. John Wiley & Sons, Ltd, 2nd edition, 2000.

[3] G. Birkhoff. *Lattice Theory*. AMS, 3rd edition, 1967.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[5] M. Doty. Birkhoff's representation theorem for finite distributive lattices. *Archive of Formal Proofs*, December 2022. https://isa-afp.org/entries/Birkhoff_Finite_Distributive_Lattices.html, Formal proof development.

[6] D. Dubhashi, J. Jonasson, and D. Ranjan. Positive influence and negative dependence. *Combinatorics, Probability and Computing*, 16(1):29—41, 2007.

[7] D. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *Random Structures & Algorithms*, 13(2):99–124, 1998.

[8] D. P. Dubhashi, V. Priebe, and D. Ranjan. Negative dependence through the fkg inequality. *BRICS Report Series*, 3, 1996.

[9] C. Fortuin, P. Kastelyn, and J. Ginibre. Correlation inequalities on some partially ordered sets. *Commun. Math. Phys.*, 22:89–103, jun 1971.

[10] K. Gopinathan and I. Sergey. Certifying certainty and uncertainty in approximate membership query structures. In S. K. Lahiri and C. Wang, editors, *Computer Aided Verification*, pages 279–303, Cham, 2020. Springer International Publishing.

[11] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[12] R. Impagliazzo and V. Kabanets. Constructive proofs of concentration bounds. In M. Serna, R. Shaltiel, K. Jansen, and J. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 617–631, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[13] K. Joag-Dev and F. Proschan. Negative association of random variables with applications. *Annals of Statistics*, 11:286–295, 1983.

[14] S. Lisawadi and T.-C. Hu. On the negative association property for the dependent bootstrap random variables. *Lobachevskii Journal of Mathematics*, 32:32–38, 2011.

[15] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis.* Cambridge University Press, USA, 2nd edition, 2017.

[16] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[17] R. Pemantle. Towards a theory of negative dependence. *Journal of Mathematical Physics*, 41(3):1371–1390, 03 2000.