Countable Sums and Discrete (Sub)Distributions

Gergely Buday Andrei Popescu

March 19, 2025

Abstract

We provide elementary formalizations of countable sums over positive real numbers, and of discrete probabilistic subdistributions and distributions. This is intended as a lightweight alternative to the corresponding concepts from the Isabelle distribution, which are defined using their continuous counterparts (namely Lebesgue integral and general probability distributions) and therefore have significant dependencies.

Contents

1 Infinite Sums of Positive Reals **1**
1.1 Preliminaries 1
1.2 Positivity, boundedness and infinite summation 7

2 Discrete Subdistributions and Distributions **18**
2.1 Definitions and Basic Properties 19
2.2 Monadic structure 21
2.3 Expectation 36

1 Infinite Sums of Positive Reals

This is a theory of infinite sums of positive reals defined as limits of finite sums. The goal is to make reasoning about these infinite sums almost as easy as that about finite sums.

theory *Infinite-Sums-of-Positive-Reals*
imports *Complex-Main HOL−Library.Countable-Set*
begin

1.1 Preliminaries

lemma *real-pm-iff*:
$\bigwedge a\ b\ c.\ (a::real) + b \leq c \longleftrightarrow a \leq c - b$
$\bigwedge a\ b\ c.\ (a::real) + b \leq c \longleftrightarrow b \leq c - a$

$\bigwedge a\ b\ c.\ (a::real) \leq b - c \longleftrightarrow c \leq b - a$
by *auto*

lemma *real-md-iff*:
$\bigwedge a\ b\ c.\ a \geq 0 \Longrightarrow b > 0 \Longrightarrow c \geq 0 \Longrightarrow (a::real) * b \leq c \longleftrightarrow a \leq c\ /\ b$
$\bigwedge a\ b\ c.\ a > 0 \Longrightarrow b \geq 0 \Longrightarrow c \geq 0 \Longrightarrow (a::real) * b \leq c \longleftrightarrow b \leq c\ /\ a$
$\bigwedge a\ b\ c.\ a > 0 \Longrightarrow b \geq 0 \Longrightarrow c > 0 \Longrightarrow (a::real) \leq b\ /\ c \longleftrightarrow c \leq b\ /\ a$
by (*auto simp*: *field-simps*)

lemma *disjoint-finite-aux*:
$\forall i{\in}I.\ \forall j{\in}I.\ i \neq j \longrightarrow A\ i \cap A\ j = \{\} \Longrightarrow B \subseteq \bigcup\ (A\ `\ I) \Longrightarrow$ *finite* $B \Longrightarrow$ *finite* $\{i \in I.\ B \cap A\ i \neq \{\}\}$
apply(*rule inj-on-finite*[*of* $\lambda i.$ *SOME* $b.\ b \in B \cap A\ i$ - B])
subgoal unfolding *inj-on-def* **by** *auto* (*metis* (*no-types*, *lifting*) *IntI empty-iff someI*)
subgoal by *auto* (*metis* (*no-types*, *lifting*) *someI*)
subgoal by *auto* **.**

lemma *incl-UNION-aux*: $B \subseteq \bigcup\ (A\ `\ I) \Longrightarrow B = \bigcup\ ((\lambda i.\ (B \cap A\ i))\ `\ \{i \in I.\ B \cap A\ i \neq \{\}\})$
by *fastforce*

lemma *incl-UNION-aux2*: $B \subseteq \bigcup\ (A\ `\ I) \longleftrightarrow B = \bigcup\ ((\lambda i.\ (B \cap A\ i))\ `\ I)$
by *fastforce*

lemma *sum-singl*[*simp*]: *sum* $f\ \{a\} = f\ a$
by *simp*

lemma *sum-two*[*simp*]: $a1 \neq a2 \Longrightarrow$ *sum* $f\ \{a1,a2\} = f\ a1 + f\ a2$
by *simp*

lemma *sum-three*[*simp*]: $a1 \neq a2 \Longrightarrow a1 \neq a3 \Longrightarrow a2 \neq a3 \Longrightarrow$
sum $f\ \{a1,a2,a3\} = f\ a1 + f\ a2 + f\ a3$
by (*simp add*: *add.assoc*)

lemma *Sup-leq*:
$A \neq \{\} \Longrightarrow \forall\ a{\in}A.\ \exists\ b{\in}B.\ (a::real) \leq b \Longrightarrow$ *bdd-above* $B \Longrightarrow$ *Sup* $A \leq$ *Sup* B
by (*simp add*: *cSup-mono*)

lemma *Sup-image-leq*:
$A \neq \{\} \Longrightarrow\ \forall\ a{\in}A.\ \exists\ b{\in}B.\ (f\ a::real) \leq g\ b \Longrightarrow$ *bdd-above* $(g\ `\ B) \Longrightarrow$
Sup $(f\ `\ A) \leq$ *Sup* $(g\ `\ B)$
by (*rule Sup-leq*) *auto*

lemma *Sup-cong*:
assumes $A \neq \{\} \vee B \neq \{\}\ \forall\ a{\in}A.\ \exists\ b{\in}B.\ (a::real) \leq b\ \forall\ b{\in}B.\ \exists\ a{\in}A.\ (b::real)$

$\leq a$
bdd-above A ∨ bdd-above B
shows *Sup A = Sup B*
proof−
have *A ≠ {} ∧ B ≠ {} bdd-above A ∧ bdd-above B*
using *assms* **unfolding** *bdd-above-def* **using** *order.trans* **by** *blast+*
thus *?thesis* **using** *assms* **by** (*metis Sup-leq order-antisym*)
qed

lemma *Sup-image-cong*:
A ≠ {} ∨ B ≠ {} ⟹ ∀ a∈A. ∃ b∈B. (f a::real) ≤ g b ⟹ ∀ b∈B. ∃ a∈A. (g b::real) ≤ f a ⟹
bdd-above (f ‘ A) ∨ bdd-above (g ‘ B) ⟹
Sup (f ‘ A) = Sup (g ‘ B)
by (*rule Sup-cong*) *auto*

lemma *Sup-congL*:
A ≠ {} ⟹ ∀ a∈A. ∃ b∈B. (a::real) ≤ b ⟹ ∀ b∈B. b ≤ Sup A ⟹ Sup A = Sup B
by (*metis Collect-empty-eq Collect-mem-eq Sup-leq bdd-aboveI cSup-least dual-order.antisym*)

lemma *Sup-image-congL*:
A ≠ {} ⟹ ∀ a∈A. ∃ b∈B. (f a::real) ≤ g b ⟹ ∀ b∈B. g b ≤ Sup (f ‘ A) ⟹ Sup (f ‘ A) = Sup (g ‘ B)
by (*rule Sup-congL*) *auto*

lemma *Sup-congR*:
B ≠ {} ⟹ ∀ a∈A. a ≤ Sup B ⟹ ∀ b∈B. ∃ a∈A. (b::real) ≤ a ⟹ Sup A = Sup B
by (*metis Collect-empty-eq Collect-mem-eq Sup-leq bdd-aboveI cSup-least dual-order.antisym*)

lemma *Sup-image-congR*:
B ≠ {} ⟹ ∀ a∈A. f a ≤ Sup (g ‘ B) ⟹ ∀ b∈B. ∃ a∈A. (g b::real) ≤ f a ⟹ Sup (f ‘ A) = Sup (g ‘ B)
by (*rule Sup-congR*) *auto*

lemma *Sup-eq-0-iff*:
assumes *A ≠ {} bdd-above A (∀ a∈A. (a::real) ≥ 0)*
shows *Sup A = 0 ⟷ (∀ a∈A. a = 0)*
using *assms* **by** (*metis all-not-in-conv cSup-eq-maximum cSup-upper leD less-eq-real-def*)

lemma *plus-Sup-commute*:
assumes *f1*: *{f1 b1 | b1. φ1 b1} ≠ {} bdd-above {f1 b1 | b1. φ1 b1}* **and**
f2: *{f2 b2 | b2. φ2 b2} ≠ {} bdd-above {f2 b2 | b2. φ2 b2}*
shows
Sup {(f1 b1::real) | b1 . φ1 b1} + Sup {f2 b2 | b2 . φ2 b2} =
Sup {f1 b1 + f2 b2 | b1 b2. φ1 b1 ∧ φ2 b2} (**is** *?L1 + ?L2 = ?R*)

proof−
have *f*:
{f1 b1 + f2 b2 | b1 b2. φ1 b1 ∧ φ2 b2} ≠ {}
bdd-above {f1 b1 + f2 b2 | b1 b2. φ1 b1 ∧ φ2 b2}
subgoal using *f1 f2* **by** *auto*
subgoal using *f1(2) f2(2)* **unfolding** *bdd-above-def* **apply** *safe*
subgoal for *M1 M2*
apply(*rule exI*[*of - M1 + M2*]) **using** *add-mono* **by** *blast* **. .**
show *?thesis* **proof**(*rule order.antisym*)
show *?L1 + ?L2 ≤ ?R* **unfolding** *real-pm-iff(1) cSup-le-iff*[*OF f1*] **apply** *safe*
apply(*subst real-pm-iff(3)*) **unfolding** *cSup-le-iff*[*OF f2*] **apply** *safe*
unfolding *real-pm-iff(2)*[*symmetric*] **apply**(*rule cSup-upper*[*OF - f(2)*]) **by** *auto*
next
show *?R ≤ ?L1 + ?L2* **unfolding** *cSup-le-iff*[*OF f*] **apply** *safe*
subgoal for *- b1 b2*
using *cSup-upper*[*OF - f1(2), of f1 b1*] *cSup-upper*[*OF - f2(2), of f2 b2*]
by *fastforce* **.**
qed
qed

lemma *plus-Sup-commute′*:
assumes *f1*: *A1 ≠ {} bdd-above A1* **and**
f2: *A2 ≠ {} bdd-above A2*
shows *Sup A1 + Sup A2 = Sup {(a1*::*real) + a2 | a1 a2. a1 ∈ A1 ∧ a2 ∈ A2}*

using *plus-Sup-commute*[*of id λa1. a1 ∈ A1 id λa2. a2 ∈ A2*] *assms*
by *auto*

lemma *plus-SupR*: *A ≠ {} ⟹ bdd-above A ⟹ Sup A + (b*::*real) = Sup {a + b | a. a ∈ A}*
apply(*subst cSup-singleton*[*of b, symmetric*])
apply(*subst plus-Sup-commute′*) **by** *auto*

lemma *plus-SupL*: *A ≠ {} ⟹ bdd-above A ⟹ (b*::*real) + Sup A = Sup {b + a | a. a ∈ A}*
apply(*subst cSup-singleton*[*of b, symmetric*])
apply(*subst plus-Sup-commute′*) **by** *auto*

lemma *mult-Sup-commute*:
assumes *f1*: *{f1 b1 | b1. φ1 b1} ≠ {} bdd-above {f1 b1 | b1. φ1 b1} ∀ b1. φ1 b1 ⟶ f1 b1 ≥ 0* **and**
f2: *{f2 b2 | b2. φ2 b2} ≠ {} bdd-above {f2 b2 | b2. φ2 b2} ∀ b2. φ2 b2 ⟶ f2 b2 ≥ 0*
shows
Sup {(f1 b1::*real) | b1 . φ1 b1} * Sup {f2 b2 | b2 . φ2 b2} =*

```
 Sup {f1 b1 * f2 b2 | b1 b2. φ1 b1 ∧ φ2 b2} (is ?L1 * ?L2 = ?R)
proof−
  have f:
    {f1 b1 * f2 b2 | b1 b2. φ1 b1 ∧ φ2 b2} ≠ {}
    bdd-above {f1 b1 * f2 b2 | b1 b2. φ1 b1 ∧ φ2 b2}
    ∀ b1 b2. φ1 b1 ∧ φ2 b2 ⟶ f1 b1 * f2 b2 ≥ 0
    subgoal using f1 f2 by auto
    subgoal using f1(2) f2(2) unfolding bdd-above-def apply safe
     subgoal for M1 M2
       apply(rule exI[of - M1 * M2]) using mult-mono f1(3) f2(3) by fastforce
.
    subgoal using f1(3) f2(3) by auto .
  show ?thesis
  proof(rule order.antisym)
    show ?L1 * ?L2 ≤ ?R
    proof(cases ?L1 = 0 ∨ ?L2 = 0)
      case True thus ?thesis using f
       by (smt (verit) Collect-empty-eq cSup-upper mem-Collect-eq mult-not-zero)
    next
      case False
      have gez: ?L1 > 0 ?L2 > 0 ?R ≥ 0
         apply (smt (verit) Collect-empty-eq False cSup-upper f1 mem-Collect-eq)
        apply (smt (verit) Collect-empty-eq False cSup-upper f2 mem-Collect-eq)
       by (smt (verit) Collect-empty-eq cSup-upper f(1) f(2) f(3) mem-Collect-eq)
      hence ggez: ?L1 ≥ 0 ?L2 ≥ 0 by linarith+
      show ?thesis
        unfolding real-md-iff(1)[OF ggez(1) gez(2) gez(3)]
        unfolding cSup-le-iff[OF f1(1,2)] apply safe
        subgoal for - b1
          apply(cases f1 b1 = 0)
          subgoal using divide-nonneg-pos gez(2) gez(3) by auto
          subgoal apply(subst real-md-iff(3))
            subgoal using f1(3) by auto
            subgoal using gez(3) by blast
            subgoal using gez(2) by blast
            subgoal unfolding cSup-le-iff[OF f2(1,2)] apply safe
              apply(subst real-md-iff(2)[symmetric])
              using f1(3) f2(3) gez(3) by (auto intro: cSup-upper[OF - f(2)]) . . .
    qed
  next
    show ?R ≤ ?L1 * ?L2 unfolding cSup-le-iff[OF f(1,2)] apply safe subgoal
for - b1 b2
       using cSup-upper[OF - f1(2), of f1 b1] cSup-upper[OF - f2(2), of f2 b2]
       by (metis (mono-tags, lifting) f1(3) f2(3) mem-Collect-eq mult-mono') .
  qed
qed

lemma mult-Sup-commute':
  assumes A1 ≠ {} bdd-above A1 ∀ a1∈A1. a1 ≥ 0 and
```

A2 $\neq \{\}$ *bdd-above A2* $\forall$ *a2*$\in$*A2*. *a2* $\geq$ *0*
shows *Sup A1* $*$ *Sup A2* = *Sup* $\{$(*a1*::*real*) $*$ *a2* | *a1 a2*. *a1* $\in$ *A1* $\wedge$ *a2* $\in$ *A2*$\}$

using *mult-Sup-commute*[*of id* λ*a1*. *a1* $\in$ *A1 id* λ*a2*. *a2* $\in$ *A2*] *assms*
by *auto*

lemma *mult-SupR*: *A* $\neq \{\}$ $\Longrightarrow$ *bdd-above A* $\Longrightarrow$ $\forall$ *a*$\in$*A*. *a* $\geq$ *0* $\Longrightarrow$ *b* $\geq$ *0* $\Longrightarrow$
Sup A $*$ (*b*::*real*) = *Sup* $\{$*a* $*$ *b* | *a*. *a* $\in$ *A*$\}$
apply(*subst cSup-singleton*[*of b*, *symmetric*])
apply(*subst mult-Sup-commute*$'$) **by** *auto*

lemma *mult-SupL*: *A* $\neq \{\}$ $\Longrightarrow$ *bdd-above A* $\Longrightarrow$ $\forall$ *a*$\in$*A*. *a* $\geq$ *0* $\Longrightarrow$ *b* $\geq$ *0* $\Longrightarrow$
(*b*::*real*) $*$ *Sup A* = *Sup* $\{$*b* $*$ *a* | *a*. *a* $\in$ *A*$\}$
apply(*subst cSup-singleton*[*of b*, *symmetric*])
apply(*subst mult-Sup-commute*$'$) **by** *auto*

lemma *sum-mono3*:
finite B $\Longrightarrow$ *A* $\subseteq$ *B* $\Longrightarrow$ ($\bigwedge$*b*. *b* $\in$ *B* $-$ *A* $\Longrightarrow$ *0* $\leq$ *g b*) $\Longrightarrow$ ($\bigwedge$*a*. *a* $\in$ *A* $\Longrightarrow$ (*f* *a*::*real*) $\leq$ *g a*) $\Longrightarrow$
sum f A $\leq$ *sum g B*
by (*smt* (*verit*, *best*) *sum-mono sum-mono2*)

lemma *sum-Sup-commute*:
fixes *h* :: $'a \Rightarrow$ *real*
assumes *finite J* **and** $\forall$ *i*$\in$*J*. $\{$*h b* | *b*. φ *i b*$\}$ $\neq \{\}$ $\wedge$ *bdd-above* $\{$*h b* | *b*. φ *i b*$\}$
shows *sum* (λ*i*. *Sup* $\{$*h b* | *b*. φ *i b*$\}$) *J* =
Sup $\{$*sum* (λ*i*. *h* (*b i*)) *J* | *b* . $\forall$ *i*$\in$*J*. φ *i* (*b i*)$\}$
using *assms* **proof**(*induction J*)
case (*insert j J*)

{assume $\forall$ *i*$\in$*J*. *Ex* (φ *i*) $\wedge$ *bdd-above* $\{$*h b* |*b*. φ *i b*$\}$
and *0*: $\{\sum$ *i*$\in$*J*. *h* (*b2 i*) |*b2*. $\forall$ *i*$\in$*J*. φ *i* (*b2 i*)$\}$ = $\{\}$
hence $\forall$ *i*$\in$*J*. $\exists$ *b*. φ *i b* **by** *auto*
hence $\exists$ *b2*. $\forall$ *i*$\in$*J*. φ *i* (*b2 i*)
by (*intro exI*[*of* - λ*i*. *SOME b*. φ *i b*]) (*simp add*: *some-eq-ex*)
hence *False* **using** *0* **by** *blast* **} note** *1* = *this*

{assume $\forall$ *i*$\in$*J*. *Ex* (φ *i*) $\wedge$ ($\exists$ *M*. $\forall$ *x*$\in\{$*h b* |*b*. φ *i b*$\}$. *x* $\leq$ *M*)
then obtain *Ms* **where** *0*: $\forall$ *i*$\in$*J*. $\forall$ *b*. φ *i b* $\longrightarrow$ *h b* $\leq$ *Ms i*
by *simp metis*
have $\exists$ *M*. $\forall$ *x*$\in\{\sum$ *i*$\in$*J*. *h* (*b2 i*) |*b2*. $\forall$ *i*$\in$*J*. φ *i* (*b2 i*)$\}$. *x* $\leq$ *M*
apply(*rule exI*[*of* - *sum Ms J*]) **using** *insert 0*
by (*auto simp*: *sum-mono*)
} note *2* = *this*

show *?case* **apply**(*subst sum.insert*)
subgoal by *fact*

subgoal by *fact*
subgoal apply(*subst sum.insert*)
subgoal by *fact*
subgoal by *fact*
subgoal apply(*subst insert.IH*)
subgoal using *insert* **by** *auto*
subgoal using *insert* **apply**(*subst plus-Sup-commute*)
subgoal by *auto*
subgoal by *auto*
subgoal using *1* **by** *auto*
subgoal unfolding *bdd-above-def* **using** *2* **by** *auto*
subgoal apply(*rule arg-cong*[*of - - Sup*]) **apply**(*rule Collect-eqI*) **apply** *safe*
subgoal for *- b1 b2* **apply**(*rule exI*[*of - λj'. if j'=j then b1 else b2 j'*])
by (*smt* (*verit*, *best*) *insertE sum.cong*)
subgoal by *auto* . . . . .
qed *auto*

1.2 Positivity, boundedness and infinite summation

definition *positive* :: ('*a* ⇒ *real*) ⇒ '*a set* ⇒ *bool* **where**
positive f A ≡ ∀ *a*∈*A*. *f a* ≥ *0*

definition *sbounded* :: ('*a* ⇒ *real*) ⇒ '*a set* ⇒ *bool* **where**
sbounded f A ≡ ∃ *r*. ∀ *B*. *B* ⊆ *A* ∧ *finite B* ⟶ *sum f B* ≤ *r*

definition *isum* :: ('*a* ⇒ *real*) ⇒ '*a set* ⇒ *real* **where**
isum f A ≡ *Sup* (*sum f* ` {*B* | *B* . *B* ⊆ *A* ∧ *finite B*})

lemma *positive-mono*: *positive p A* ⟹ *B* ⊆ *A* ⟹ *positive p B*
unfolding *positive-def* **by** *auto*

lemma *positive-eq*:
assumes *positive f A* **and** ∀ *a*∈*A*. *f1 a* = *f a*
shows *positive f1 A*
using *assms* **unfolding** *positive-def* **by** *auto*

lemma *sbounded-eq*:
assumes *sbounded f A* **and** ∀ *a*∈*A*. *f1 a* = *f a*
shows *sbounded f1 A*
using *assms* **unfolding** *sbounded-def* **by** (*simp add*: *subset-eq*)

lemma *finite-imp-sbounded*: *positive f A* ⟹ *finite A* ⟹ *sbounded f A*
unfolding *sbounded-def positive-def* **apply**(*rule exI*[*of - sum f A*])
by *simp* (*meson DiffD1 sum-mono2*)

lemma *sbounded-empty*[*simp*,*intro*!]: *sbounded f* {}
unfolding *sbounded-def* **by** *auto*

lemma *sbounded-insert*[*simp*]: *sbounded f* (*insert a A*) ⟷ *sbounded f A*
unfolding *sbounded-def* **apply** *safe*
subgoal by (*meson subset-insertI2*)
subgoal for *r* **apply**(*rule exI*[*of - r + max 0* (*f a*)])
by *simp* (*smt* (*verit*) *finite-Diff subset-insert-iff sum.remove*) **.**

lemma *sbounded-Un*[*simp*]: *sbounded f* (*A1* ∪ *A2*) ⟷ *sbounded f A1* ∧ *sbounded f A2*
unfolding *sbounded-def* **apply** *safe*
subgoal by (*meson sup.coboundedI1*)
subgoal by (*meson sup.coboundedI2*)
subgoal for *r1 r2* **apply**(*rule exI*[*of - r1+r2*]) **apply** *safe* **apply**(*elim subset-UnE*)
by *simp* (*smt* (*verit*) *Diff-subset finite-Diff order.trans sum.union-diff2 sum.union-inter*)
.

lemma *sbounded-UNION*:
assumes *finite I* **shows** *sbounded f* (⋃*i*∈*I*. *A i*) ⟷ (∀*i*∈*I*. *sbounded f* (*A i*))
using *assms* **by** (*induction I*) *auto*

lemma *sbounded-mono*: *A* ⊆ *B* ⟹ *sbounded f B* ⟹ *sbounded f A*
unfolding *sbounded-def* **by** (*meson order-trans*)

lemma *sbounded-reindex*: *sbounded* (*f o u*) *A* ⟹ *sbounded f* (*u* ‘ *A*)
unfolding *sbounded-def* **apply** *safe* **subgoal for** *r*
apply(*rule exI*[*of - r*]) **apply** *safe*
by (*metis finite-image-iff subset-image-inj sum.reindex*) **.**

lemma *sbounded-reindex-inj-on*: *inj-on u A* ⟹ *sbounded f* (*u* ‘ *A*) ⟷ *sbounded* (*f o u*) *A*
by (*smt* (*verit*, *best*) *comp-apply f-the-inv-into-f sbounded-eq sbounded-reindex the-inv-into-onto*)

lemma *sbounded-swap*:
sbounded (λ(*a*,*b*). *f a b*) (*A* × *B*) ⟷ *sbounded* (λ(*b*,*a*). *f a b*) (*B* × *A*)
proof−
have *0*: *A* × *B* = (λ(*a*,*b*). (*b*,*a*)) ‘ (*B* × *A*) **by** *auto*
show *?thesis* **unfolding** *0* **apply**(*subst sbounded-reindex-inj-on*)
by (*auto simp*: *o-def inj-on-def intro*!: *arg-cong2*[*of - - - - sbounded*])
qed

lemma *sbounded-constant-0*:
assumes ∀ *a*∈*A*. *f a* = (*0*::*real*)
shows *sbounded f A*
using *assms* **unfolding** *sbounded-def*

by (*metis in-mono order-refl sum-nonpos*)

lemma *sbounded-setminus*:
assumes *sbounded f A* **and** $\forall\, b{\in}B{-}A.\ f\ b = 0$
shows *sbounded f B*
by (*metis Un-Diff-cancel2 assms sbounded-Un sbounded-constant-0*)

lemma *isum-eq-sum*:
positive f A $\Longrightarrow$ *finite A* $\Longrightarrow$ *isum f A* = *sum f A*
unfolding *isum-def positive-def* **apply**(*subst cSup-eq-maximum*[*of sum f A*])
subgoal by *simp*
subgoal using *sum-mono3* **by** *fastforce*
subgoal by *simp* **.**

lemma *isum-cong*:
assumes $A = B$ **and** $\bigwedge x.\ x \in B \Longrightarrow g\ x = h\ x$
shows *isum g A* = *isum h B*
using *assms* **unfolding** *isum-def*
by (*auto intro*!: *SUP-cong comm-monoid-add-class.sum.cong*)

lemma *isum-mono*:
assumes *sbounded h A* **and** $\bigwedge x.\ x \in A \Longrightarrow g\ x \leq h\ x$
shows *isum g A* $\leq$ *isum h A*
using *assms* **unfolding** *isum-def*
apply(*intro cSup-mono*)
subgoal by *auto*
subgoal unfolding *bdd-above-def sbounded-def* **by** *auto*
subgoal for *r* **apply** *safe*
subgoal for - *B* **apply**(*rule bexI*[*of* - *sum h B*]) **apply** (*auto intro*: *sum-mono*)
. . .

lemma *isum-mono′*:
assumes *sbounded g B* **and** $A \subseteq B$
shows *isum g A* $\leq$ *isum g B*
using *assms* **unfolding** *isum-def*
apply(*intro cSup-subset-mono*)
unfolding *bdd-above-def sbounded-def* **by** *auto*

lemma *isum-empty*[*simp*]: *isum g* {} = *0*
unfolding *isum-def* **by** *auto*

lemma *isum-const-zero*[*simp*]: *isum* (λ*x*. *0*) *A* = *0*
unfolding *isum-def*
by *simp* (*metis cSUP-const empty-iff empty-subsetI finite.emptyI mem-Collect-eq*)

lemma *isum-const-zero'*: ∀ *x*∈*A*. *g x* = *0* ⟹ *isum g A* = *0*
by (*simp add*: *isum-cong*)

lemma *isum-eq-0-iff*: *positive f A* ⟹ *sbounded f A* ⟹ *isum f A* = *0* ⟷ (∀ *a*∈*A*. *f a* = *0*)
unfolding *isum-def* **apply**(*subst Sup-eq-0-iff*)
subgoal by *auto*
subgoal unfolding *bdd-above-def sbounded-def* **by** *auto*
subgoal by (*auto simp*: *positive-def in-mono sum-nonneg*)
subgoal by (*auto simp*: *positive-def in-mono sum-nonneg*) **.**

lemma *isum-reindex*: *inj-on h A* ⟹ *isum g* (*h* ‘ *A*) = *isum* (*g* ∘ *h*) *A*
unfolding *isum-def* **apply**(*rule arg-cong*[*of - - Sup*])
unfolding *image-def*[*of sum g*] *image-def*[*of sum* (*g* ∘ *h*)]
apply(*rule Collect-eqI*)
by (*smt* (*verit*, *del-insts*) *finite-image-iff inj-on-subset mem-Collect-eq subset-image-inj sum.reindex*)

lemma *isum-reindex-cong*: *inj-on l B* ⟹ *A* = *l* ‘ *B* ⟹
(⋀*x*. *x* ∈ *B* ⟹ *g* (*l x*) = *h x*) ⟹ *isum g A* = *isum h B*
by (*metis isum-cong comp-def isum-reindex*)

lemma *isum-reindex-cong'*:
(⋀*x y*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* ≠ *y* ⟹ *h x* = *h y* ⟹ *g* (*h x*) = *0*) ⟹ *isum g* (*h* ‘ *A*) = *isum* (*g* ∘ *h*) *A*
unfolding *isum-def* **apply**(*rule arg-cong*[*of - - Sup*])
unfolding *image-def*[*of sum g*] *image-def*[*of sum* (*g* ∘ *h*)] **apply**(*rule Collect-eqI*)
apply *safe*
subgoal by(*metis* (*no-types*, *lifting*) *finite-image-iff mem-Collect-eq subset-image-inj sum.reindex*)
subgoal by (*smt* (*verit*) *finite-imageI image-mono in-mono mem-Collect-eq sum.reindex-nontrivial*)
.

lemma *isum-zeros-cong*:
assumes *sbounded g* (*S* ∩ *T*) ∨ *sbounded h* (*S* ∩ *T*)
and (⋀*i*. *i* ∈ *T* − *S* ⟹ *h i* = *0*) **and** (⋀*i*. *i* ∈ *S* − *T* ⟹ *g i* = *0*)
and (⋀*x*. *x* ∈ *S* ∩ *T* ⟹ *g x* = *h x*)
shows *isum g S* = *isum h T*

proof−
have *0*: *sbounded g S* ∨ *sbounded h T* **using** *assms*
by (*metis Diff-Int2 Int-absorb Int-commute sbounded-setminus*)
show *?thesis* **unfolding** *isum-def* **apply**(*rule Sup-image-cong*)
subgoal by *auto*
subgoal apply *safe*
subgoal for - *B*
apply(*rule bexI*[*of* - *B* ∩ *T*], *rule order-eq-refl*)
apply(*rule sum.mono-neutral-cong*) **using** *assms* **by** *auto* **.**
subgoal apply *safe*
subgoal for - *B*
apply(*rule bexI*[*of* - *B* ∩ *S*], *rule order-eq-refl*)
apply(*rule sum.mono-neutral-cong*) **using** *assms* **by** *auto* **.**
subgoal using *assms*(*2*,*3*,*4*) *0* **unfolding** *bdd-above-def sbounded-def* **apply**
(*elim disjE exE*)
subgoal for *r* **apply**(*rule disjI1*) **apply**(*rule exI*[*of* - *r*]) **by** *auto*
subgoal for *r* **apply**(*rule disjI2*) **apply**(*rule exI*[*of* - *r*]) **by** *auto* **. .**
qed

lemma *isum-zeros-congL*:
sbounded g S ⟹ *S* ⊆ *T* ⟹ ∀ *i*∈*T* − *S*. *g i* = *0* ⟹ *isum g S* = *isum g T*
by (*metis Diff-eq-empty-iff emptyE isum-zeros-cong le-iff-inf*)

lemma *isum-zeros-congR*:
sbounded g S ⟹ *S* ⊆ *T* ⟹ ∀ *i*∈*T* − *S*. *g i* = *0* ⟹ *isum g T* = *isum g S*
by (*simp add*: *isum-zeros-congL*)

lemma *isum-singl*[*simp*]: *f a* ≥ (*0*::*real*) ⟹ *isum f* {*a*} = *f a*
by (*simp add*: *positive-def isum-eq-sum*)

lemma *isum-two*[*simp*]: *a1* ≠ *a2* ⟹ *f a1* ≥ (*0*::*real*) ⟹ *f a2* ≥ *0* ⟹ *isum f*
{*a1*,*a2*} = *f a1* + *f a2*
by (*simp add*: *positive-def isum-eq-sum*)

lemma *isum-three*[*simp*]: *a1* ≠ *a2* ⟹ *a1* ≠ *a3* ⟹ *a2* ≠ *a3* ⟹ *f a1* ≥ *0* ⟹
f a2 ≥ (*0*::*real*) ⟹ *f a3* ≥ *0* ⟹
isum f {*a1*,*a2*,*a3*} = *f a1* + *f a2* + *f a3*
by (*simp add*: *positive-def isum-eq-sum*)

lemma *isum-ge-0*: *positive f A* ⟹ *sbounded f A* ⟹ *isum f A* ≥ *0*
using *isum-mono′* **by** *fastforce*

lemma *in-le-isum*: *positive f A* ⟹ *sbounded f A* ⟹ *a* ∈ *A* ⟹ *f a* ≤ *isum f A*
by (*metis* (*full-types*) *DiffE positive-def Set.set-insert equals0I*
insert-mono isum-mono′ isum-singl subsetI)

lemma *isum-eq-singl*:

assumes *fx*: $f\ a = x$ **and** *f*: $\forall\, a'.\ a' \neq a \longrightarrow f\ a' = 0$ **and** *x*: $x \geq 0$ **and** *a*: $a \in A$
shows $isum\ f\ A = x$
apply(*subst fx*[*symmetric*])
apply(*subst isum-singl*[*of f a*, *symmetric*])
subgoal using *assms* **by** *simp*
subgoal apply(*rule isum-zeros-cong*) **using** *assms* **by** *auto* **.**

lemma *isum-le-singl*:
assumes *fx*: $f\ a \leq x$ **and** *f*: $\forall\, a'.\ a' \neq a \longrightarrow f\ a' = 0$ **and** *x*: $f\ a \geq 0$ **and** *a*: $a \in A$
shows $isum\ f\ A \leq x$
by (*metis a f fx isum-eq-singl x*)

lemma *isum-insert*[*simp*]: $a \notin A \Longrightarrow sbounded\ f\ A \Longrightarrow f\ a \geq 0 \Longrightarrow isum\ f\ (insert\ a\ A) = isum\ f\ A + f\ a$
unfolding *isum-def* **apply**(*subst plus-SupR*)
subgoal by *auto*
subgoal unfolding *sbounded-def bdd-above-def* **by** *auto*
subgoal apply(*rule Sup-cong*)
subgoal by *auto*
subgoal apply *safe* **subgoal for** - - *B*
apply(*rule bexI*[*of* - *sum f* $(B - \{a\}) + f\ a$])
subgoal by (*cases* $a \in B$) (*auto simp*: *image-def sum.remove*)
subgoal by *blast* **. .**
subgoal apply *safe* **subgoal for** - - - *B*
apply(*rule bexI*[*of* - *sum f* (*insert a B*)])
subgoal by (*cases* $a \in B$) (*auto simp*: *image-def sum.remove*)
subgoal by *blast* **. .**
subgoal unfolding *sbounded-def bdd-above-def* **apply** (*elim exE*)
subgoal for *r* **apply**(*rule disjI2*, *rule exI*[*of* - $r + f\ a$]) **by** *auto* **. . .**

lemma *isum-UNION*:
assumes *dsj*: $\forall\, i{\in}I.\ \forall\, j{\in}I.\ i \neq j \longrightarrow A\ i \cap A\ j = \{\}$ **and** *sb*: $sbounded\ g\ (\bigcup\ (A\ `\ I))$
shows $isum\ g\ (\bigcup\ (A\ `\ I)) = isum\ (\lambda i.\ isum\ g\ (A\ i))\ I$
proof−
{fix *J* **assume** *J*: $J \subseteq I$ *finite J*
have $sum\ (\lambda i.\ Sup\ \{sum\ g\ B \mid B.\ B \subseteq A\ i \wedge finite\ B\})\ J =$
$Sup\ \{sum\ (\lambda i.\ sum\ g\ (B\ i))\ J \mid B\ .\ \forall\, i{\in}J.\ B\ i \subseteq A\ i \wedge finite\ (B\ i)\}$
using *J sb* **apply**(*subst sum-Sup-commute*)
subgoal by *auto*
subgoal unfolding *sbounded-def bdd-above-def*
by *simp* (*metis SUP-upper2 bot.extremum finite.emptyI in-mono*)
subgoal by *auto* **.**
also have $\ldots =$
$Sup\ \{sum\ g\ (\bigcup\ (B\ `\ J)) \mid B\ .\ \forall\, i{\in}J.\ B\ i \subseteq A\ i \wedge finite\ (B\ i)\}$
apply(*rule arg-cong*[*of* - - *Sup*]) **apply**(*rule Collect-eqI*) **apply**(*rule ex-cong1*)

```
    apply safe
    subgoal
      apply(rule sum.UNION-disjoint[symmetric])
      using J dsj by auto (metis IntI empty-iff subsetD)
    subgoal
      apply(rule sum.UNION-disjoint)
      using J dsj by auto (metis IntI empty-iff subsetD) .
  also have ... =
    Sup {sum g B | B . B ⊆ ⋃ (A ` J) ∧ finite B}
    apply(rule Sup-cong) apply safe
    subgoal by auto
    subgoal for - B
      apply(rule bexI[of - sum g (⋃ (B ` J))]) using J by auto
    subgoal for - B
      apply(rule bexI[of - sum g (⋃ ((λi. B ∩ A i) ` J))])
      subgoal using J unfolding incl-UNION-aux2 by auto
      subgoal by blast .
    subgoal using J sb unfolding bdd-above-def sbounded-def
      by simp (metis UN-mono finite-UN-I) .
  also have ... ≤
    Sup {sum g B | B . B ⊆ ⋃ (A ` I) ∧ finite B}
    apply(rule cSup-subset-mono) apply safe
    subgoal by auto
    subgoal using J sb unfolding sbounded-def bdd-above-def by auto
    subgoal for - B apply(rule exI[of - B]) using J by auto .
  finally
  have (∑ i∈J. Sup {y. ∃ B⊆A i. finite B ∧ y = sum g B}) ≤
        Sup {y. ∃ B⊆⋃ (A ` I). finite B ∧ y = sum g B}
    by (smt (verit) Collect-cong sum-mono)
} note 1 = this

show ?thesis
  unfolding isum-def apply(rule Sup-image-congL)
  unfolding image-def[of sum g] image-def[of sum (λi. Sup {y. ∃ x∈{B |B. B ⊆
A i ∧ finite B}. y = sum g x})]
    apply safe
  subgoal by auto
  subgoal for - B using assms
    apply(intro bexI[of - {i. i ∈ I ∧ B ∩ A i ≠ {}}])
    subgoal apply(subst incl-UNION-aux[of B A I], assumption)
      apply(subst comm-monoid-add-class.sum.UNION-disjoint)
      subgoal by (simp add: disjoint-finite-aux)
      subgoal by auto
      subgoal by auto
      subgoal apply(rule sum-mono)
        subgoal for i apply(rule cSup-upper)
          subgoal by auto
            subgoal unfolding bdd-above-def sbounded-def by simp (metis UN-I
```

in-mono subsetI) . . .
    subgoal by (*simp add*: *disjoint-finite-aux*) .
  subgoal using *1* **by** *auto* .
qed

lemma *isum-Un*[*simp*]:
  assumes *positive f A1 sbounded f A1 positive f A2 sbounded f A2 A1 ∩ A2 = {}*
  shows *isum f (A1 ∪ A2) = isum f A1 + isum f A2*
proof−
  have [*simp*]: *isum* (λ*i*. *isum f* (*case i of 0 ⇒ A1 | Suc - ⇒ A2*)) {*0*, *Suc 0*} = *isum f A1 + isum f A2*
    apply(*subst isum-two*) **using** *assms* **by** (*auto intro*!: *isum-ge-0*)
  show *?thesis* **using** *assms isum-UNION*[*of* {*0*,*1*::*nat*} λ*i*. *case i of 0 ⇒ A1 |- ⇒ A2 f*] **by** *auto*
qed

lemma *isum-Sigma*:
  assumes *sbd*: *sbounded* (λ(*a*,*b*). *f a b*) (*Sigma A Bs*)
  shows *isum* (λ(*a*,*b*). *f a b*) (*Sigma A Bs*) = *isum* (λ*a*. *isum* (*f a*) (*Bs a*)) *A*
proof−
  define *g* **where** *g* ≡ λ(*a*,*b*). *f a b*
  define *Cs* **where** *Cs* ≡ λ*a*. {*a*} × *Bs a*
  have *1*: ⋀*a*. {*a*} × *Bs a* = *Pair a* ‘ (*Bs a*) **by** *auto*
  have *0*: *Sigma A Bs* = (⋃ *a*∈*A*. *Cs a*) **unfolding** *Cs-def* **by** *auto*
  show *?thesis* **unfolding** *0* **apply**(*subst isum-UNION*)
    subgoal unfolding *Cs-def* **by** *auto*
    subgoal using *sbd* **unfolding** *0* .
    subgoal unfolding *Cs-def* **apply**(*rule arg-cong2*[*of - - - - isum*])
      subgoal unfolding *1*
        apply(*subst isum-reindex-cong'*) **by** (*auto simp*: *o-def*)
      subgoal .. . .
qed

lemma *isum-Times*:
  assumes *sbounded* (λ(*a*,*b*). *f a b*) (*A* × *B*)
  shows *isum* (λ(*a*,*b*). *f a b*) (*A* × *B*) = *isum* (λ*a*. *isum* (*f a*) *B*) *A*
  using *isum-Sigma*[*OF assms*] .

lemma *isum-swap*:
  assumes *sbounded* (λ(*a*,*b*). *f a b*) (*A* × *B*)
  shows *isum* (λ*a*. *isum* (*f a*) *B*) *A* = *isum* (λ*b*. *isum* (λ*a*. *f a b*) *A*) *B* (**is** *?L* = *?R*)
proof−
  have *0*: *A* × *B* = (λ(*a*,*b*). (*b*,*a*)) ‘ (*B* × *A*) **by** *auto*
  have *?L* = *isum* (λ(*a*,*b*). *f a b*) (*A* × *B*) **using** *isum-Times*[*OF assms*] **..**

```
 also have ... = isum (λ(b,a). f a b) (B × A) unfolding 0 apply(subst isum-reindex-cong')

    by (auto simp: o-def intro!: arg-cong2[of - - - - isum])
  also have ... = ?R apply(subst isum-Times) using assms sbounded-swap by
auto
  finally show ?thesis .
qed

lemma isum-plus:
  assumes f1: positive f1 A sbounded f1 A
    and f2: positive f2 A sbounded f2 A
  shows isum (λa. f1 a + f2 a) A = isum f1 A + isum f2 A
  unfolding isum-def apply(subst plus-Sup-commute')
  subgoal by auto
  subgoal using f1 unfolding sbounded-def bdd-above-def by auto
  subgoal by auto
  subgoal using f2 unfolding sbounded-def bdd-above-def by auto
  subgoal apply(rule Sup-cong)
    subgoal by auto
    subgoal apply safe
      subgoal for - - B
        apply(rule bexI[of - ∑ a∈B. f1 a + f2 a])
        subgoal by auto
         subgoal apply clarify apply(rule exI[of - sum f1 B]) apply(rule exI[of -
sum f2 B])
          using sum.distrib by auto . .
    subgoal apply safe
      subgoal for - - - - - B1 B2
        apply(rule bexI[of - ∑ a∈B1 ∪ B2. f1 a + f2 a])
        subgoal apply(subst sum.distrib) apply(rule add-mono)
           subgoal apply(rule sum-mono2) using f1 unfolding positive-def by
auto
           subgoal apply(rule sum-mono2) using f2 unfolding positive-def by
auto .
        subgoal by auto . .
    subgoal apply(rule disjI2)
      using f1 f2 unfolding sbounded-def bdd-above-def apply (elim exE)
      subgoal for r1 r2 apply(rule exI[of - r1 + r2]) apply safe
        by (auto simp: add-mono) . . .



lemma sbounded-product:
  assumes f: positive f A sbounded f A and g: positive g B sbounded g B
  shows sbounded (λ(a,b). f a * g b) (A × B)
  using assms unfolding sbounded-def positive-def apply safe
  subgoal for r1 r2
    apply(rule exI[of - r1*r2]) apply safe subgoal for Ba
```

apply(*rule order.trans*[*OF sum-mono2*[*of fst'Ba* × *snd'Ba Ba* λ(*a*,*b*). *f a* ∗ *g b*]])
subgoal by *auto*
subgoal by (*simp add*: *subset-fst-snd*)
subgoal for *ab* **apply**(*cases ab*, *simp*)
by (*metis* (*no-types*, *lifting*) *imageE mem-Sigma-iff mult-nonneg-nonneg prod.collapse subset-eq*)
subgoal apply(*subst sum.cartesian-product*[*symmetric*])
apply(*subst sum-product*[*symmetric*])
by (*smt* (*verit*) *finite-imageI imageE mem-Sigma-iff mult-mono prod.collapse subset-eq sum-nonneg*) **. . .**

lemma *sbounded-multL*: *x* ≥ *0* ⟹ *sbounded f A* ⟹ *sbounded* (λ*a*. *x* ∗ *f a*) *A*
unfolding *sbounded-def* **apply** *safe*
subgoal for *r* **apply**(*rule exI*[*of* - *x* ∗ *r*])
by (*metis mult.commute mult-right-mono sum-distrib-left*) **.**

lemma *sbounded-multL-strict*[*simp*]:
assumes *x*: *x* > *0*
shows *sbounded* (λ*a*. *x* ∗ *f a*) *A* ⟷ *sbounded f A*
proof−
have *0*: *f* = (λ*a*. (*1* / *x*) ∗ (*x* ∗ *f a*)) **unfolding** *fun-eq-iff* **using** *x* **by** *auto*
show *?thesis* **apply** *safe*
subgoal apply(*subst 0*) **apply**(*rule sbounded-multL*) **using** *x* **by** *auto*
subgoal apply(*rule sbounded-multL*) **using** *x* **by** *auto* **.**
qed

lemma *sbounded-multR*: *x* ≥ *0* ⟹ *sbounded f A* ⟹ *sbounded* (λ*a*. *f a* ∗ *x*) *A*
unfolding *sbounded-def* **apply** *safe*
subgoal for *r* **apply**(*rule exI*[*of* - *r* ∗ *x*])
by (*metis mult.commute mult-left-mono sum-distrib-right*) **.**

lemma *sbounded-multR-strict*[*simp*]:
assumes *x*: *x* > *0*
shows *sbounded* (λ*a*. *f a* ∗ *x*) *A* ⟷ *sbounded f A*
proof−
have *0*: *f* = (λ*a*. (*f a* ∗ *x*) ∗ (*1* / *x*)) **unfolding** *fun-eq-iff* **using** *x* **by** *auto*
show *?thesis* **apply** *safe*
subgoal apply(*subst 0*) **apply**(*rule sbounded-multR*) **using** *x* **by** *auto*
subgoal apply(*rule sbounded-multR*) **using** *x* **by** *auto* **.**
qed

lemma *positive-sbounded-multL*:
assumes *f*: *positive f A sbounded f A* **and** *g*: ∀ *a*∈*A*. *g a* ≤ *x*
shows *sbounded* (λ*a*. *f a* ∗ *g a*) *A*
proof−
define *y* **where** *y* ≡ *max x 0*
have *g*: ∀ *a*∈*A*. *g a* ≤ *y* ∧ *y* ≥ *0* **using** *g* **unfolding** *y-def* **by** *auto*
show *?thesis* **using** *assms* **unfolding** *sbounded-def* **apply** *safe*

```
    subgoal for r  apply(intro exI[of - r * y]) apply safe
      subgoal for B
        apply(rule order.trans[OF sum-mono[of B λa. f a * g a λa. f a * y]])
        subgoal unfolding positive-def
          by (simp add: g in-mono mult-left-mono)
        subgoal apply(subst sum-distrib-right[symmetric])
          by (metis max.cobounded2 mult.commute mult-left-mono y-def) . . .
qed

lemma positive-sbounded-multR:
  assumes f: positive f A sbounded f A and g: ∀ a∈A. g a ≤ x
  shows sbounded (λa. g a * f a) A
  using positive-sbounded-multL[OF assms]
  unfolding sbounded-def apply safe
  subgoal for r apply(rule exI[of - r]) apply safe
    subgoal for B
      apply(subst sum.cong[of B B - λa. f a * g a]) by auto . .


lemma isum-product-Times:
  assumes f: positive f A sbounded f A and g: positive g B sbounded g B
  shows isum f A * isum g B = isum (λ(a,b). f a * g b) (A × B)
  unfolding isum-def apply(subst mult-Sup-commute′)
  subgoal by auto
  subgoal using f(2) unfolding bdd-above-def sbounded-def by auto
  subgoal using f by (auto simp: positive-def subsetD sum-nonneg)
  subgoal by auto
  subgoal using g(2) unfolding bdd-above-def sbounded-def by auto
  subgoal using g by (auto simp: positive-def subsetD sum-nonneg)
  subgoal apply(rule Sup-cong)
    subgoal by auto
    subgoal using f(2) g(2) unfolding sbounded-def apply safe
      subgoal for a r1 r2 a1 a2 x xa A1 B1
        apply(rule bexI[of - sum (λ(a, b). f a * g b) (A1 × B1)])
        subgoal apply(subst sum-product)
          apply(subst sum.cartesian-product)
          apply(subst sum-mono2) by auto
        subgoal by (simp add: image-def) (metis Sigma-mono finite-SigmaI) . .
    subgoal using f(2) g(2) unfolding sbounded-def apply safe
      subgoal for b x r ra AB1
        apply(rule bexI[of - sum f (fst ‘ AB1) * sum g (snd ‘ AB1)])
        subgoal apply(subst sum-product)
          apply(subst sum.cartesian-product)
          apply(subst sum-mono2)
          subgoal by (simp-all add: subset-fst-snd)
          subgoal by (simp-all add: subset-fst-snd)
          subgoal for ab using f(1) g(1) unfolding positive-def apply(cases ab)
        by simp (metis (no-types, lifting) imageE mem-Sigma-iff mult-nonneg-nonneg
prod.collapse subsetD)
```

```
        subgoal .. .
       subgoal apply (simp add: image-def)
        apply(rule exI[of - sum f (fst ‘ AB1)], rule exI[of - sum g (snd ‘ AB1)])
        apply (simp add: image-def) apply safe
        subgoal apply(rule exI[of - fst ‘ AB1]) by (fastforce simp: image-def)
        subgoal apply(rule exI[of - snd ‘ AB1]) by (fastforce simp: image-def) .
. .
   subgoal apply(rule disjI1) using sbounded-product[OF assms]
     unfolding sbounded-def bdd-above-def image-def apply safe
     subgoal for r apply(rule exI[of - r])
    by simp (metis (no-types) Sigma-mono finite-cartesian-product sum.cartesian-product
sum-product)
    . . .


lemma isum-product:
  assumes f: positive f A sbounded f A and g: positive g B sbounded g B
  shows isum f A * isum g B = isum (λa. isum (λb. f a * g b) B) A
  by (simp add: assms isum-Times isum-product-Times sbounded-product)


lemma isum-distribR:
  assumes f: positive f (A::'a set) sbounded f A and r: r ≥ 0
  shows isum f A * r = isum (λa. f a * r) A
proof−
  have 0: A × {0} = (λa. (a, 0::nat)) ‘ A by auto
  show ?thesis
    apply(subst isum-singl[of λ-. r 0::nat,symmetric, OF r])
    apply(subst isum-product-Times)
    subgoal by fact
    subgoal by fact
    subgoal using r unfolding positive-def by auto
    subgoal by simp
   subgoal apply (subst 0) apply(subst isum-reindex) unfolding inj-on-def o-def
by auto .
qed


lemma isum-distribL:
  assumes f: positive f (A::'a set) sbounded f A and r: r ≥ 0
  shows r * isum f A = isum (λa. r * f a) A
proof−
  have 0: r * isum f A = isum f A * r
    (λa. r * f a) = (λa. f a * r) unfolding fun-eq-iff by auto
  show ?thesis unfolding 0 using isum-distribR[OF assms] .
qed


end
```

2 Discrete Subdistributions and Distributions

This theory defines countably discrete probability (sub)distributions and their monadic operators, namely:

- Kleisli extension, "ext"
- functorial action, the lifting operator "lift"
- monad unit, the indicator function "ind"
- monad counit, the flattening operators "flat" for subdistributions and "dflat" for subdistributions

Basic facts about them are proved, including the monadic laws.

In all operators except the monad counit (flattening/averaging), the operators for distributions are restrictions of those for subdistributions. For flattening, as explained later we must use two distinct operators "flat" and "dflat".

We also define the expectation operator, "expd", which is the Lebesgue integral for the discrete case.

theory *Discrete-Subdistributions-and-Distributions*
imports *Infinite-Sums-of-Positive-Reals*
begin

2.1 Definitions and Basic Properties

definition *Subdis* :: *'a set* ⇒ (*'a* ⇒ *real*) *set* **where**
Subdis A ≡ {*p*. *positive p A* ∧ *sbounded p A* ∧ *isum p A* ≤ *1*}

definition *Dis* :: *'a set* ⇒ (*'a* ⇒ *real*) *set* **where**
Dis A ≡ {*p*. *p* ∈ *Subdis A* ∧ *isum p A* ≥ *1*}

lemma *Dis-incl-Subdis*: *Dis A* ⊆ *Subdis A* **unfolding** *Dis-def* **by** *auto*

lemma *Subdis-mono*: *p* ∈ *Subdis A* ⟹ *B* ⊆ *A* ⟹ *p* ∈ *Subdis B*
unfolding *Subdis-def* **by** *simp* (*meson isum-mono' order-trans positive-mono sbounded-mono*)

lemma *Subdis-Dis2*: *Subdis* (*Subdis A*) ⊆ *Subdis* (*Dis A*)
by (*meson Dis-incl-Subdis Subdis-mono subsetI*)

lemma *Subdis-ge-0*: *p* ∈ *Subdis A* ⟹ *a* ∈ *A* ⟹ *p a* ≥ *0*
unfolding *Subdis-def positive-def* **by** *auto*

lemma *Subdis-le-1*: *p* ∈ *Subdis A* ⟹ *a* ∈ *A* ⟹ *p a* ≤ *1*

unfolding *Subdis-def* **using** *in-le-isum* **by** *fastforce*

lemma *Subdis-eq*:
assumes $p \in Subdis\ A$ **and** $\forall\, a{\in}A.\ p1\ a = p\ a$
shows $p1 \in Subdis\ A$
using *Subdis-def assms isum-cong positive-eq sbounded-eq* **by** *fastforce*

lemma *Dis-Subdis-mono*: $p \in Dis\ A \Longrightarrow B \subseteq A \Longrightarrow p \in Subdis\ B$
using *Dis-def Subdis-mono* **by** *blast*

lemma *Dis-zeros-mono*: $p \in Dis\ A \Longrightarrow B \subseteq A \Longrightarrow \forall\, a{\in}A{-}B.\ p\ a = 0 \Longrightarrow p \in Dis\ B$
by (*metis* (*no-types*, *lifting*) *Dis-Subdis-mono Dis-def Subdis-def isum-zeros-congL mem-Collect-eq*)

lemma *Dis-ge-0*: $p \in Dis\ A \Longrightarrow a \in A \Longrightarrow p\ a \geq 0$
using *Dis-def Subdis-ge-0* **by** *fastforce*

lemma *Dis-le-1*: $p \in Dis\ A \Longrightarrow a \in A \Longrightarrow p\ a \leq 1$
using *Dis-def Subdis-le-1* **by** *fastforce*

lemma *Dis-isum-1*: $p \in Dis\ A \Longrightarrow isum\ p\ A = 1$
using *Dis-def Subdis-def isum-eq-sum* **by** *fastforce*

lemma *Dis-sum-1*: $p \in Dis\ A \Longrightarrow finite\ A \Longrightarrow sum\ p\ A = 1$
using *Dis-def Subdis-def isum-eq-sum* **by** *fastforce*

lemma *Dis-eq*:
assumes $p \in Dis\ A$ **and** $\forall\, a{\in}A.\ p1\ a = p\ a$
shows $p1 \in Dis\ A$
unfolding *Dis-def*
using *Dis-def Subdis-eq assms isum-cong* **by** *fastforce*

lemma *Subdis-le-1-eq-1*: $p \in Subdis\ A \Longrightarrow 1 \leq isum\ p\ A \Longrightarrow isum\ p\ A = 1$
unfolding *Subdis-def* **by** *auto*

lemma *Subdis-sum-le-1*: $p \in Subdis\ A \Longrightarrow finite\ A \Longrightarrow sum\ p\ A \leq 1$
using *Subdis-def isum-eq-sum* **by** *fastforce*

lemma *Subdis-sum-ge-0*: $p \in Subdis\ A \Longrightarrow finite\ A \Longrightarrow sum\ p\ A \geq 0$
by (*simp add*: *Subdis-ge-0 sum-nonneg*)

lemma *Subdis-sum-ge-0-sub*: $p \in Subdis\ A \Longrightarrow B \subseteq A \Longrightarrow finite\ B \Longrightarrow sum\ p\ B \geq 0$
using *Subdis-mono Subdis-sum-ge-0* **by** *blast*

lemma *Subdis-sum-le-1-sub*: $p \in Subdis\ A \Longrightarrow B \subseteq A \Longrightarrow finite\ B \Longrightarrow sum\ p\ B \leq 1$
using *Subdis-mono Subdis-sum-le-1* **by** *blast*

lemma *Subdis-sboundedL*:
assumes $p \in$ *Subdis* A $\forall\, a{\in}A.\ g\ a \leq x$
shows *sbounded* $(\lambda a.\ p\ a * g\ a)\ A$
using *Subdis-def assms positive-sbounded-multL* **by** *fastforce*

lemma *Subdis-sboundedR*:
assumes $p \in$ *Subdis* A $\forall\, a{\in}A.\ g\ a \leq x$
shows *sbounded* $(\lambda a.\ g\ a * p\ a)\ A$
using *Subdis-def assms positive-sbounded-multR* **by** *fastforce*

lemma *Subdis-isum-leL*:
assumes *p*: $p \in$ *Subdis* A **and** *g*: *positive* $g\ A$ $\forall\, a{\in}A.\ g\ a \leq x$ **and** *x*: $x \geq 0$
shows *isum* $(\lambda a.\ p\ a * g\ a)\ A \leq x$
apply(*rule order.trans*[*OF isum-mono*[*of* $\lambda a.\ p\ a * x$]])
subgoal apply(*rule sbounded-multR*) **using** $x\ p$ **unfolding** *Subdis-def* **by** *auto*
subgoal using $x\ p\ g$ **unfolding** *Subdis-def positive-def* **by** (*simp add*: *mult-mono*)
subgoal apply(*subst isum-distribR*[*symmetric*])
using *assms* **unfolding** *Subdis-def* **by** (*auto simp*: *mult.commute mult-left-le*)
.

lemma *Subdis-isum-leR*:
assumes *p*: $p \in$ *Subdis* A **and** *g*: *positive* $g\ A$ $\forall\, a{\in}A.\ g\ a \leq x$ **and** *x*: $x \geq 0$
shows *isum* $(\lambda a.\ g\ a * p\ a)\ A \leq x$
proof−
have *0*: $(\lambda a.\ g\ a * p\ a) = (\lambda a.\ p\ a * g\ a)$ **unfolding** *fun-eq-iff* **by** *auto*
show *?thesis* **unfolding** *0* **using** *Subdis-isum-leL*[*OF assms*] **.**
qed

lemma *Subdis-sum-le-Max*:
assumes *finite* A $p \in$ *Subdis* A *positive* $g\ A$ $A \neq \{\}$
shows $(\sum a{\in}A.\ p\ a * g\ a) \leq$ *Max* $(g\ `\ A)$
proof−
have *0*: *Max* $(g\ `\ A) \geq 0$
by (*metis positive-def Max-ge-iff assms*(*1*,*3*,*4*) *ex-in-conv finite-imageI image-eqI*)
have $(\sum a{\in}A.\ p\ a * g\ a) \leq (\sum a{\in}A.\ p\ a *$ *Max* $(g\ `\ A))$
apply(*rule sum-mono*) **using** *assms* **unfolding** *Subdis-def positive-def*
by (*simp add*: *mult-left-mono*)
also have $\ldots = (\sum a{\in}A.\ p\ a) *$ *Max* $(g\ `\ A)$
using *sum-distrib-right*[*symmetric*] **.**
also have $\ldots \leq$ *Max* $(g\ `\ A)$ **using** *assms*
using *Subdis-sum-le-1*[*of* p A] *0 Subdis-sum-ge-0 mult-left-le-one-le* **by** *blast*
finally show *?thesis* **.**
qed

lemma *Subdis-sum-le*:
assumes *finite* A $p \in$ *Subdis* A *positive* $g\ A$ $A \neq \{\}$ $\forall\, a{\in}A.\ g\ a \leq x$

shows $(\sum a{\in}A.\ p\ a * g\ a) \leq x$
using *Subdis-sum-le-Max*[*OF assms*(1−4)]
by (*metis assms*(1,4,5) *dual-order.trans obtains-MAX*)

2.2 Monadic structure

definition *ind* :: $'a \Rightarrow ('a \Rightarrow real)$ **where**
ind a ≡ $\lambda a'$. *if* $a' = a$ *then 1 else 0*

lemma *ind-simps*[*simp*]: $\bigwedge a$. *ind a a* = *1*
$\bigwedge a\ a'$. $a' \neq a \Longrightarrow$ *ind* a' *a* = *0*
unfolding *ind-def* **by** *auto*

lemma *ind-eq-0-iff*[*simp*]: *ind a* $a' = 0 \longleftrightarrow a \neq a'$
unfolding *ind-def* **by** *auto*

lemma *ind-eq-1-iff*[*simp*]: *ind a* $a' = 1 \longleftrightarrow a = a'$
unfolding *ind-def* **by** *auto*

lemma *ind-ge-0*: *ind a* $a' \geq 0$
unfolding *ind-def* **by** *auto*

lemma *ind-le-1*: *ind a* $a' \leq 1$
unfolding *ind-def* **by** *auto*

lemma *positive-ind*[*simp*]: *positive* (*ind a*) *A*
unfolding *ind-def positive-def* **by** *auto*

lemma *sbounded-ind*[*simp*]: *sbounded* (*ind a*) *A*
unfolding *ind-def sbounded-def* **by** *auto*

lemma *sum-ind*[*simp*]: $\bigwedge a\ B$. *finite B* $\Longrightarrow$ *a* ∈ *B* $\Longrightarrow$ *sum* (*ind a*) *B* = *1*
$\bigwedge a\ B$. *finite B* $\Longrightarrow$ *a* ∉ *B* $\Longrightarrow$ *sum* (*ind a*) *B* = *0*
subgoal for *a B*
unfolding *ind-simps*(*1*)[*symmetric*, *of a*] *sum-singl*[*of ind a a*, *symmetric*]
apply(*rule sum.mono-neutral-right*) **by** *auto*
subgoal by (*metis ind-eq-0-iff sum.neutral*) **.**

lemma *isum-ind*[*simp*]: $\bigwedge a\ A$. *a* ∈ *A* $\Longrightarrow$ *isum* (*ind a*) *A* = *1*
$\bigwedge a\ A$. *a* ∉ *A* $\Longrightarrow$ *isum* (*ind a*) *A* = *0*
apply (*simp add*: *ind-ge-0 isum-eq-singl*)
by (*metis ind-simps*(*2*) *isum-const-zero*′)

lemma *ind-Subdis*[*simp*, *intro*!]: *ind a* ∈ *Subdis A*
unfolding *Subdis-def*
by (*metis* (*lifting*) *isum-ind*(*1*,*2*) *le-numeral-extra*(*4*) *linordered-nonzero-semiring-class.zero-le-one mem-Collect-eq positive-ind sbounded-ind*)

lemma *Dis-ind*[*simp*, *intro*!]: *a* ∈ *A* $\Longrightarrow$ *ind a* ∈ *Dis A*

unfolding *Dis-def* **by** *simp*

lemma *ind-mult-SubdisL*:
assumes *p*: $p \in Subdis\ A$
shows $(\lambda a.\ p\ a * ind\ (f\ a)\ a') \in Subdis\ A$
unfolding *Subdis-def mem-Collect-eq* **using** *p* **apply** *safe*
subgoal by (*simp add*: *positive-def Subdis-ge-0 ind-ge-0*)
subgoal apply(*rule Subdis-sboundedL*[*of - - - 1*]) **by** (*auto simp*: *ind-le-1*)
subgoal apply(*rule order.trans*[*of - isum p A*])
apply(*rule isum-mono*) **unfolding** *Subdis-def* **using** *ind-le-1*
by (*auto simp*: *positive-def ind-le-1 mult-left-le*) **.**

lemma *ind-mult-SubdisR*:
assumes *p*: $p \in Subdis\ A$
shows $(\lambda a.\ ind\ (f\ a)\ a' * p\ a) \in Subdis\ A$
proof−
have *0*: $(\lambda a.\ ind\ (f\ a)\ a' * p\ a) = (\lambda a.\ p\ a * ind\ (f\ a)\ a')$ **by** *fastforce*
show *?thesis* **unfolding** *0* **using** *ind-mult-SubdisL*[*OF p*] **.**
qed

lemma *isum-ind-multL*: $a' \in A \Longrightarrow f\ a' \geq 0 \Longrightarrow isum\ (\lambda a.\ f\ a * ind\ a'\ a)\ A = f\ a'$
apply(*rule isum-eq-singl*[*of - a'*]) **by** *auto*

lemma *isum-ind-multR*: $a' \in A \Longrightarrow f\ a' \geq 0 \Longrightarrow isum\ (\lambda a.\ ind\ a'\ a * f\ a)\ A = f\ a'$
apply(*rule isum-eq-singl*[*of - a'*]) **by** *auto*

definition *ext* :: $'a\ set \Rightarrow ('a \Rightarrow ('b \Rightarrow real)) \Rightarrow (('a \Rightarrow real) \Rightarrow ('b \Rightarrow real))$
where
$ext\ A\ f \equiv \lambda p\ b.\ isum\ (\lambda a.\ p\ a * f\ a\ b)\ A$

lemma *ext-ge-0*:
assumes *f*: $\forall a{\in}A.\ f\ a \in Subdis\ B$ **and** *p*: $p \in Subdis\ A$ **and** *b*: $b \in B$
shows $ext\ A\ f\ p\ b \geq 0$
unfolding *ext-def* **apply**(*rule isum-ge-0*)
subgoal using *assms* **unfolding** *Subdis-def positive-def* **by** *auto*
subgoal apply(*rule positive-sbounded-multL*[*of - - - 1*])
using *assms* **unfolding** *Subdis-def*
by *auto* (*meson Subdis-le-1 f*) **.**

lemma *Subdis-sum-isum-le-1*:
assumes *B*: *finite B* **and** *f*: $\forall a{\in}A.\ f\ a \in Subdis\ B$ **and** *p*: $p \in Subdis\ A$

```
  shows (∑ b∈B. isum (λa. p a * f a b) A) ≤ 1
proof−
  have 0: (λ(b, a). p a * f a b) = (λba. p (snd ba) * f (snd ba) (fst ba))
    unfolding fun-eq-iff by auto
  have 1: (λba. p (snd ba)) = (λ(b,a). 1 * p a)
    unfolding fun-eq-iff by auto
  have 2: ⋀AB. AB ⊆ A × B ⟹ fst ‘ AB ⊆ A by auto
  have 3: ⋀AB. AB ⊆ A × B ⟹ finite AB ⟹ sum p (fst ‘ AB) ≤ 1
    using p unfolding Subdis-def sbounded-def
    by (metis (mono-tags, lifting) 2 Subdis-sum-le-1-sub finite-imageI p)
  show ?thesis apply(subst isum-eq-sum[symmetric])
    subgoal unfolding Subdis-def positive-def apply safe
      subgoal for a apply(rule isum-ge-0)
        subgoal using f p unfolding Subdis-def positive-def by auto
        subgoal apply(rule Subdis-sboundedL[of - - - 1])
          subgoal using p .
          subgoal using f by (meson Subdis-le-1) . . .
    subgoal by fact
    subgoal apply(subst isum-swap[symmetric])
      subgoal unfolding sbounded-def apply(rule exI[of - real (card B)]) apply
safe
        subgoal for AB apply(rule order.trans[of - ∑ (a, b)∈(fst‘AB)×B. p a *
1])
          subgoal apply(rule sum-mono3)
            subgoal using B by auto
           subgoal by (smt (verit, ccfv-threshold) SigmaE mem-Sigma-iff subset-eq
subset-fst-snd)
            subgoal for ab using p f unfolding Subdis-def positive-def by (cases
ab, auto)
          subgoal for ab using p f unfolding Subdis-def positive-def apply (cases
ab, auto)
              by (meson Subdis-le-1 f in-mono mem-Sigma-iff mult-left-le) .
          subgoal apply(subst sum.cartesian-product[symmetric]) apply simp
            apply(subst sum-distrib-left[symmetric])
            by (metis 3 real-of-card sum-bounded-below) . .
      subgoal apply(subst order.trans[OF isum-mono[of p]])
        subgoal using Subdis-def p by blast
        subgoal for a apply(subst isum-distribL[symmetric])
          subgoal using Subdis-def f by blast
          subgoal using Subdis-def f by blast
          subgoal by (meson Subdis-ge-0 p)
        subgoal using Subdis-def p f mult-left-le unfolding Subdis-def positive-def
by fastforce .
        subgoal using Subdis-def p by blast
        subgoal by simp . . .
qed

lemma sbounded-prod-Subdis:
  assumes f: ∀ a∈A. f a ∈ Subdis B and p: p ∈ Subdis A
```

```
  shows sbounded (λ(a, b). p b * f b a) (B × A)
proof−
  have 0: ⋀BA. BA ⊆ B × A ⟹ fst ‘ BA ⊆ B ∧ snd ‘ BA ⊆ A by auto
  show ?thesis
    unfolding sbounded-def apply(rule exI[of - 1]) apply safe
    subgoal for BA apply(rule order.trans[OF sum-mono2[of fst‘BA × snd‘BA]])
      subgoal by auto
      subgoal by (simp add: subset-fst-snd)
      subgoal for ab using assms unfolding Subdis-def positive-def
        using in-mono by fastforce
      subgoal unfolding sum.cartesian-product[symmetric]
        apply(rule order.trans[OF - Subdis-sum-isum-le-1[of fst‘BA snd‘BA f p]])
        subgoal apply(rule sum-mono)
          subgoal for b apply(subst isum-eq-sum)
            subgoal using assms unfolding positive-def
              by (metis 0 Subdis-ge-0 in-mono zero-le-mult-iff)
            subgoal by auto
            subgoal by auto . .
        subgoal by auto
        subgoal using f by (metis 0 Subdis-mono in-mono)
        subgoal using p using 0 Subdis-mono by force . . .
qed



lemma ext-eq: ∀ a∈A. p1 a = p2 a ⟹ ∀ a∈A. ∀ b∈B. f1 a b = f2 a b ⟹
 b ∈ B ⟹ ext A f1 p1 b = ext A f2 p2 b
  by (metis (mono-tags, lifting) ext-def isum-cong)

lemma ext-Subdis:
  assumes f: ∀ a∈A. f a ∈ Subdis B and p: p ∈ Subdis A
  shows ext A f p ∈ Subdis B
proof−
  have 0: (λ(b, a). p a * f a b) = (λba. p (snd ba) * f (snd ba) (fst ba))
    unfolding fun-eq-iff by auto
  have 1: (λba. p (snd ba)) = (λ(b,a). 1 * p a)
    unfolding fun-eq-iff by auto
  show ?thesis
    unfolding Subdis-def mem-Collect-eq apply safe
    subgoal using ext-ge-0[OF assms] unfolding positive-def by auto
    subgoal unfolding sbounded-def ext-def
      apply(rule exI[of - 1]) apply safe
      subgoal for Ba
        apply(rule Subdis-sum-isum-le-1[of Ba])
        using assms Subdis-mono by force+ .
    subgoal unfolding ext-def apply(subst isum-swap)
      subgoal using sbounded-prod-Subdis[OF assms] .
      subgoal apply(subst order.trans[OF isum-mono[of p]])
        subgoal using Subdis-def p by blast
```

```
      subgoal for a apply(subst isum-distribL[symmetric])
        subgoal using Subdis-def f by blast
        subgoal using Subdis-def f by blast
        subgoal by (meson Subdis-ge-0 p)
       subgoal using Subdis-def p f mult-left-le unfolding Subdis-def positive-def
by fastforce .
      subgoal using Subdis-def p by blast
      subgoal by simp . . .
qed

lemma ext-Dis:
  assumes f: ∀ a∈A. f a ∈ Dis B and p: p ∈ Dis A
  shows ext A f p ∈ Dis B
proof−
  have isum (ext A f p) B = 1 unfolding ext-def apply(subst isum-swap)
    subgoal apply(rule sbounded-prod-Subdis)
      using assms unfolding Dis-def by auto
    subgoal apply(rule trans[of - isum (λb. p b ∗ isum (f b) B) A])
      subgoal apply(rule isum-cong, rule refl)
        apply(rule isum-distribL[symmetric])
        subgoal by (meson Dis-ge-0 positive-def f)
        subgoal using Dis-def Subdis-def f by blast
        subgoal by (meson Dis-ge-0 p) .
      subgoal apply(rule trans[of - isum p A])
        subgoal apply(rule isum-cong, rule refl)
          subgoal for a using f apply(subst Dis-isum-1) by auto .
        subgoal using p using Dis-isum-1 by auto . . .
  thus ?thesis using assms unfolding Dis-def by (simp add: ext-Subdis)
qed


lemma ext-ind: p ∈ Subdis A ⟹ a ∈ A ⟹ ext A ind p a = p a
  unfolding ext-def fun-eq-iff
 by (smt (verit) Subdis-ge-0 ind-simps isum-eq-singl mult-cancel-left2 mult-minus-right)



lemma ext-o:
  assumes f: ∀ a∈A. f a ∈ B and gg: ∀ b∈B. gg b ∈ Subdis C and p: p ∈ Subdis
A and c: c ∈ C
  shows ext A (gg o f) p c = ext B gg (ext A (ind o f) p) c
proof−
  have 0: ⋀a. a ∈ A ⟹ positive (λaa. ind (f a) aa ∗ gg aa c) B ∧
                       sbounded (λaa. ind (f a) aa ∗ gg aa c) B
    subgoal for a apply safe
      subgoal using assms unfolding positive-def apply safe
        subgoal for b by (cases b = f a, auto dest: Subdis-ge-0) .
    subgoal using assms by (metis Subdis-le-1 positive-ind positive-sbounded-multL
```

sbounded-ind) . .

show *?thesis*
unfolding *ext-def*
apply(*rule trans*[*of - isum* (λ*b*. *isum* (λ*a*. *p a* ∗ (*ind* (*f a*) *b* ∗ *gg b c*)) *A*) *B*])
subgoal apply(*subst isum-swap*)
subgoal apply(*subst sbounded-prod-Subdis*)
subgoal unfolding *Subdis-def mem-Collect-eq* **apply** *safe*
subgoal using *0* **by** *auto*
subgoal using *0* **by** *auto*
subgoal for *a* **apply**(*rule isum-le-singl*[*of - f a*])
using *assms* **by** (*auto dest*: *Subdis-ge-0 Subdis-le-1*) .
subgoal by *fact*
subgoal by *simp* .
subgoal apply(*rule isum-cong*, *rule refl*)
subgoal for *a* **apply**(*subst isum-distribL*[*symmetric*])
subgoal using *0* **by** *auto*
subgoal using *0* **by** *auto*
subgoal by (*meson Subdis-ge-0 p*)
subgoal unfolding *mult-cancel-left* **apply**(*rule disjI2*)
unfolding *o-def* **apply**(*subst isum-singl*[*of* λ*b*. *gg b c f a*, *symmetric*])
subgoal by (*meson Subdis-ge-0 c f gg*)
subgoal apply(*rule isum-zeros-cong*)
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal using *f* **by** *auto*
subgoal by *auto* . . . . .
subgoal apply(*rule isum-cong*, *rule refl*)
apply(*subst isum-distribR*)
subgoal using *Subdis-def ind-mult-SubdisL p* **by** *fastforce*
subgoal apply(*rule Subdis-sboundedL*[*of - - - 1*]) **using** *assms*
by (*simp-all add*: *ind-le-1*)
subgoal by (*meson Subdis-ge-0 c gg*)
subgoal apply(*rule isum-cong*) **by** *auto* . .
qed

definition *lift* :: *'a set* ⇒ (*'a* ⇒ *'b*) ⇒ (*'a* ⇒ *real*) ⇒ (*'b* ⇒ *real*) **where**
lift A f p ≡ λ*b*. *isum* (λ*a*. *p a*) {*a*. *a* ∈ *A* ∧ *f a* = *b*}

lemma *lift-ext*:
assumes *p*: *p* ∈ *Subdis A*
shows *lift A f p* = *ext A* (*ind o f*) *p*
apply(*rule ext*) **unfolding** *lift-def ext-def* **apply**(*rule isum-zeros-cong*)
subgoal by (*metis* (*no-types*, *lifting*) *Subdis-def inf-le2 mem-Collect-eq p sbounded-mono*)
subgoal using *assms* **by** *auto*
subgoal by *auto*

subgoal using *assms* **by** *auto* **.**

lemma *lift-eq*:
assumes *f*: $\forall\, a{\in}A.\ f1\ a = f2\ a$ **and** *p*: $\forall\, a{\in}A.\ p1\ a = p2\ a$ **and** *b*: $b \in B$
shows $lift\ A\ f1\ p1\ b = lift\ A\ f2\ p2\ b$
unfolding *lift-def* **apply**(*rule isum-cong*) **using** *assms* **by** *auto*

lemma *lift-Subdis*:
assumes *p*: $p \in Subdis\ A$
shows $lift\ A\ f\ p \in Subdis\ B$
by (*simp add*: *ext-Subdis lift-ext p*)

lemma *lift-Dis*:
assumes *f*: $\forall\, a{\in}A.\ f\ a \in B$ **and** *p*: $p \in Dis\ A$
shows $lift\ A\ f\ p \in Dis\ B$
by (*smt* (*verit*, *best*) *Dis-incl-Subdis Dis-ind comp-apply ext-Dis f lift-ext p subset-iff*)

lemma *lift-id*[*simp*]:
assumes *p*: $p \in Subdis\ A$ **and** $a{\in}A$
shows $lift\ A\ id\ p\ a = p\ a$
unfolding *lift-ext*[*OF p*] **using** *ext-ind*[*OF assms*] **by** *simp*

lemma *lift-o*[*simp*]:
assumes *f*: $\forall\, a{\in}A.\ f\ a \in B$ **and** *g*: $\forall\, b{\in}B.\ g\ b \in C$ **and** *p*: $p \in Subdis\ A$ **and** *c*: $c{\in}C$
shows $lift\ A\ (g\ o\ f)\ p\ c = lift\ B\ g\ (lift\ A\ f\ p)\ c$
apply(*subst lift-ext*[*OF p*])
apply(*subst o-assoc*)
apply(*subst ext-o*[*of A f B ind* $\circ$ *g C*])
subgoal using *f* **by** *auto*
subgoal using *g* **by** *auto*
subgoal by *fact*
subgoal by *fact*
by (*metis lift-Subdis lift-ext p*)

lemma *lift-ind*:
assumes *a*: $a \in A$
shows $lift\ A\ f\ (ind\ a) = ind\ (f\ a)$
apply(*rule ext*)
subgoal for *b*
apply(*cases f a* = *b*)
subgoal using *a* **unfolding** *lift-def* **by** *auto*

subgoal unfolding *lift-def* **by** *auto* **. .**

lemma *isum-lift*:
assumes *f*: $\forall\, a{\in}A.\ f\ a \in B$ **and** *p*: $p \in Subdis\ A$
shows $isum\ (lift\ A\ f\ p)\ B = isum\ p\ A$
proof−
have *A*: $isum\ p\ A = isum\ p\ (\bigcup\ ((\lambda b.\ \{a\ .\ a \in A \wedge f\ a = b\})\ `\ B))$
apply(*rule arg-cong*[*of* - - *isum p*]) **using** *f* **by** *auto*
show *?thesis* **apply**(*subst A*) **apply**(*subst isum-UNION*)
subgoal by *auto*
subgoal using *p* **unfolding** *Subdis-def*
by (*metis* (*no-types*, *lifting*) *SUP-least mem-Collect-eq sbounded-mono subset-eq*)
subgoal unfolding *lift-def* **.. .**
qed

lemma *lift-reflects-Dis*:
assumes *f*: $\forall\, a{\in}A.\ f\ a \in B$ **and** *p*: $p \in Subdis\ A$
shows $lift\ A\ f\ p \in Dis\ B \longleftrightarrow p \in Dis\ A$
by (*metis* (*no-types*, *lifting*) *Dis-def f isum-lift lift-Dis mem-Collect-eq p*)

definition *flatP* :: $('a \Rightarrow real)\ set \Rightarrow$
$(('a \Rightarrow real) \Rightarrow real) \Rightarrow ('a \Rightarrow real)$ **where**
$flatP\ Da\ pp \equiv \lambda a.\ isum\ (\lambda p.\ pp\ p * p\ a)\ Da$

lemma *flatP-ext*: $flatP\ Da = ext\ Da\ id$
unfolding *flatP-def ext-def* **by** *simp*

lemma *flatP-eq*: $\forall\, p{\in}Da.\ pp1\ p = pp2\ p \Longrightarrow a \in A \Longrightarrow flatP\ Da\ pp1\ a = flatP\ Da\ pp2\ a$
unfolding *flatP-ext* **apply**(*rule ext-eq*[*of* - - - *A*]) **by** *auto*

lemma *flatP-Subdis*: $Da \subseteq Subdis\ A \Longrightarrow$
$pp \in Subdis\ Da \Longrightarrow flatP\ Da\ pp \in Subdis\ A$
unfolding *flatP-ext* **apply**(*rule ext-Subdis*) **unfolding** *Dis-def* **by** *auto*

lemma *flatP-Da*: $\forall\, pp{\in}Dis\ Da.\ ext\ Da\ id\ pp \in Da \Longrightarrow$
$pp \in Dis\ Da \Longrightarrow flatP\ Da\ pp \in Da$
by (*simp add*: *flatP-ext*)

lemma *flatP-lift-ind*:
assumes *Da*: $Da \subseteq Subdis\ A\ ind\ `\ A \subseteq Da$
and *p*: $p \in Subdis\ A$ **and** *a*: $a \in A$
shows $flatP\ Da\ (lift\ A\ ind\ p)\ a = p\ a$
unfolding *flatP-ext lift-ext*[*OF p*]
apply(*subst ext-o*[*symmetric*, *of* - - - - *A*])
using *assms* **by** (*auto simp*: *ext-ind*)

lemma *flatP-ind*:
assumes *Da*: $Da \subseteq Subdis\ A$
and $p \in Da$ **and** $a \in A$
shows $flatP\ Da\ (ind\ p)\ a = p\ a$
unfolding *flatP-def* **apply**(*rule isum-eq-singl*[*of* - *p p a*]) **using** *assms*
by (*auto simp*: *Subdis-ge-0*)

lemma *flatP-lift*:
assumes *Da*: $Da \subseteq Subdis\ A$
and *Db*: $Db \subseteq Subdis\ B$
and *Dab*: $\forall\, p{\in}Da.\ lift\ A\ f\ p \in Db$
assumes *f*: $\forall\, a{\in}A.\ f\ a \in B$ **and** *pp*: $pp \in Subdis\ Da$ **and** *b*: $b \in B$
shows $flatP\ Db\ (lift\ Da\ (lift\ A\ f)\ pp)\ b = lift\ A\ f\ (flatP\ Da\ pp)\ b$
proof−
have *0*: $\bigwedge p.\ p \in Subdis\ A \Longrightarrow positive\ (\lambda a.\ p\ a * ind\ (f\ a)\ b)\ A$
$\wedge\ sbounded\ (\lambda a.\ p\ a * ind\ (f\ a)\ b)\ A$
apply *safe*
subgoal using *assms* **unfolding** *positive-def* **by** (*simp add*: *Subdis-ge-0*)
subgoal apply(*rule Subdis-sboundedL*[*of* - - - *1*]) **using** *assms*
by (*auto simp*: *ind-le-1*) **.**
show *?thesis*
unfolding *lift-ext*[*OF pp*] **unfolding** *ext-def flatP-def*
apply(*subst lift-ext*)
subgoal unfolding *Subdis-def* **apply**(*subst Subdis-def*[*symmetric*])
by (*metis* (*no-types*, *lifting*) *Da Subdis-eq flatP-Subdis flatP-def isum-cong pp*)

subgoal unfolding *ext-def*
subgoal
apply(*rule trans*[*of* - *isum* ($\lambda p.\ isum\ (\lambda a.\ pp\ a * (ind \circ lift\ A\ f)\ a\ p * p\ b)$
Da) *Db*])
subgoal apply(*rule isum-cong*)

```
        subgoal by simp
        subgoal for p apply(subst isum-distribR)
         subgoal using assms unfolding positive-def by (simp add: Subdis-ge-0)
          subgoal apply(rule Subdis-sboundedL[of - - - 1]) using assms by (auto
simp: ind-le-1)
          subgoal by (meson Db Subdis-ge-0 b subsetD)
          subgoal by simp . .
       subgoal
        apply(rule trans[of - isum (λa. isum (λp. pp p * p a * (ind ∘ f) a b) Da)
A])
        subgoal apply(subst isum-swap[of - - Da])
         subgoal unfolding mult.assoc apply(rule sbounded-prod-Subdis)
          subgoal unfolding Subdis-def[of Db] mem-Collect-eq apply safe
           subgoal using assms unfolding positive-def
          by (metis Subdis-ge-0 comp-apply in-mono ind-ge-0 mult-nonneg-nonneg)
            subgoal apply(rule Subdis-sboundedL[of - - - 1]) using assms by
(auto simp: Subdis-le-1)
           subgoal for p apply(rule isum-le-singl[of - lift A f p - Db])
            subgoal by simp (meson Dab Db Subdis-le-1 b in-mono)
            subgoal by simp
            subgoal by simp (meson Dab Db Subdis-ge-0 b in-mono)
            subgoal using Dab by simp . .
          subgoal by fact .
         subgoal apply(subst isum-swap[of - - Da])
          subgoal unfolding mult.assoc apply(rule sbounded-prod-Subdis)
           subgoal using Da by (auto intro!: ind-mult-SubdisL)
           subgoal by fact .
          subgoal apply(rule isum-cong)
           subgoal by simp
           subgoal for p unfolding o-def lift-def mult.assoc
            subgoal apply(subst isum-distribL[symmetric])
             subgoal using assms unfolding positive-def
              by (meson Subdis-ge-0 in-mono ind-ge-0 mult-nonneg-nonneg)
             subgoal apply(rule Subdis-sboundedL[of - - - 1]) using assms by
(auto simp: Subdis-le-1)
              subgoal using Subdis-ge-0 pp by fastforce
              subgoal apply(subst isum-distribL[symmetric])
               subgoal using assms unfolding positive-def
                by (meson Subdis-ge-0 in-mono ind-ge-0 mult-nonneg-nonneg)
               subgoal apply(rule Subdis-sboundedL[of - - - 1]) using assms
by (auto simp: Subdis-le-1)
                subgoal using Subdis-ge-0 pp by fastforce
                subgoal unfolding mult-cancel-left apply(rule disjI2)
                 apply(rule isum-eq-singl[of - λb. isum p {a ∈ A. f a = b}])
                 subgoal unfolding Subdis-def using Da
                  by (fastforce intro: isum-zeros-cong sbounded-mono[of - A p]
simp: Subdis-def)
                subgoal by simp
                subgoal apply(rule isum-ge-0) using Da 0 by auto
```

subgoal unfolding *lift-def*[*symmetric*] **by** (*simp add*: *Dab*) **. .**
. . . . .
subgoal apply(*rule isum-cong*)
subgoal by *simp*
subgoal for *p* **apply**(*subst isum-distribR*)
subgoal using *assms* **unfolding** *positive-def*
by (*metis Subdis-ge-0 in-mono mult-nonneg-nonneg*)
subgoal apply(*rule Subdis-sboundedL*[*of - - - 1*]) **using** *assms* **by** (*auto simp*: *Subdis-le-1*)
subgoal using *assms* **by** (*simp add*: *ind-ge-0*)
subgoal by *simp* **. . . . . .**
qed

lemma *flatP-flatP-lift*:
assumes *Da*: *Da* ⊆ *Subdis A*
and *fDa*: ∀ *pp* ∈ *Daa*. *flatP Da pp* ∈ *Da*
and *Daa*: *Daa* ⊆ *Subdis Da*
assumes *ppp*: *ppp* ∈ *Subdis Daa* **and** *a*: *a* ∈ *A*
shows *flatP Da* (*flatP Daa ppp*) *a* = *flatP Da* (*lift Daa* (*flatP Da*) *ppp*) *a*
unfolding *flatP-def lift-ext*[*OF ppp*] *ext-def*
apply(*rule trans*[*of - isum* (λ*p*. *isum* (λ*pp*. *ppp pp* ∗ *pp p* ∗ *p a*) *Daa*) *Da*])
subgoal apply(*rule isum-cong*)
subgoal by *simp*
subgoal apply(*rule isum-distribR*)
subgoal using *assms* **unfolding** *Subdis-def*[*of Daa*] *positive-def*
by (*metis* (*no-types*, *lifting*) *Subdis-ge-0 in-mono mult-nonneg-nonneg ppp*)
subgoal apply(*rule Subdis-sboundedL*[*of - - - 1*]) **using** *assms* **by** (*auto simp*: *Subdis-le-1*)
subgoal by (*meson Da Subdis-ge-0 a subsetD*) **. .**
subgoal apply(*subst isum-swap*)
subgoal apply(*subst mult.assoc*) **apply**(*rule sbounded-prod-Subdis*)
subgoal unfolding *Subdis-def mem-Collect-eq* **apply** *safe* **unfolding** *Subdis-def*[*symmetric*]
subgoal using *assms* **unfolding** *positive-def*
by (*metis Subdis-ge-0 split-mult-pos-le subset-iff*)
subgoal apply(*rule Subdis-sboundedL*[*of - - - 1*])
using *assms Subdis-def* **by** (*auto simp*: *Subdis-le-1*)
subgoal by (*metis* (*full-types*) *Da Daa Subdis-le-1 a flatP-Subdis flatP-def in-mono*) **.**
subgoal by *fact* **.**

subgoal apply(*rule trans*[*of -*
isum (λ*p*. *isum* (λ*pp*. *ppp pp* ∗ (*ind* ∘ (λ*pp a*. *isum* (λ*p*. *pp p* ∗ *p a*) *Da*)) *pp p* ∗ *p a*)
Daa) *Da*])
subgoal apply(*subst isum-swap*[*of - Da*])
subgoal apply(*subst mult.assoc*) **apply**(*rule sbounded-prod-Subdis*)
subgoal unfolding *Subdis-def mem-Collect-eq* **apply** *safe* **unfolding**

Subdis-def[*symmetric*]
subgoal using *assms* **unfolding** *positive-def*
by *simp* (*meson Subdis-ge-0 in-mono ind-ge-0 mult-nonneg-nonneg*)
subgoal apply(*rule Subdis-sboundedL*[*of - - - 1*])
using *assms Subdis-def* **by** (*auto simp*: *Subdis-le-1*)
subgoal for *p* **unfolding** *o-def*
by (*metis* (*no-types*) *Da Subdis-le-1 a flatP-Subdis flatP-def ind-Subdis*)
.
subgoal by *fact* .
subgoal apply(*rule isum-cong*)
subgoal by *simp*
subgoal for *pp* **unfolding** *mult.assoc* **apply**(*subst isum-distribL*[*symmetric*])
subgoal using *assms* **unfolding** *positive-def*
by (*metis Subdis-ge-0 mult-nonneg-nonneg subset-eq*)
subgoal apply(*rule Subdis-sboundedL*[*of - - - 1*]) **using** *assms* **by** (*auto simp*: *Subdis-le-1*)
subgoal using *Subdis-ge-0 ppp* **by** *fastforce*
subgoal apply(*subst isum-distribL*[*symmetric*])
subgoal using *assms* **unfolding** *positive-def*
by (*metis* (*lifting*) *Subdis-ge-0 comp-eq-dest-lhs ind-Subdis split-mult-pos-le subsetD*)
subgoal apply(*rule Subdis-sboundedL*[*of - - - 1*]) **using** *assms* **by** (*auto simp*: *Subdis-le-1*)
subgoal using *Subdis-ge-0 ppp* **by** *fastforce*
subgoal unfolding *mult-cancel-left* **apply**(*rule disjI2*)
unfolding *o-def* **apply**(*subst isum-ind-multR*)
subgoal unfolding *flatP-def*[*symmetric*] **using** *Daa fDa* **by** *auto*
subgoal apply(*rule isum-ge-0*)
subgoal using *assms* **unfolding** *positive-def*
by (*metis Subdis-ge-0 mult-nonneg-nonneg subset-iff*)
subgoal apply(*rule Subdis-sboundedL*[*of - - - 1*]) **using** *assms* **by** (*auto simp*: *Subdis-le-1*) .
subgoal by *simp* . . . . .
subgoal apply(*rule isum-cong*)
subgoal by *auto*
apply(*rule isum-distribR*[*symmetric*])
subgoal using *assms* **unfolding** *positive-def* **by** (*simp add*: *Subdis-ge-0 ind-ge-0*)
subgoal apply(*rule Subdis-sboundedL*[*of - - - 1*]) **using** *assms* **by** (*auto simp*: *Subdis-le-1 ind-le-1*)
subgoal using *Da* **by** (*auto simp*: *Subdis-ge-0 a*) . . . .

definition *flat* :: *'a set* ⇒ ((*'a* ⇒ *real*) ⇒ *real*) ⇒ (*'a* ⇒ *real*) **where**
flat A pp ≡ λ*a*. *isum* (λ*p*. *pp p* ∗ *p a*) (*Subdis A*)

lemma *flat-flatP*: *flat A* = *flatP* (*Subdis A*)
by (*auto simp*: *flat-def flatP-def*)

lemma *flat-ext*: *flat A* = *ext* (*Subdis A*) *id*
unfolding *flat-flatP* **using** *flatP-ext* **.**

lemma *flat-eq*: ∀ *p*∈*Subdis A*. *pp1 p* = *pp2 p* ⟹ *a* ∈ *A* ⟹ *flat A pp1 a* = *flat A pp2 a*
by (*simp add*: *flatP-eq flat-flatP*)

lemma *flat-Subdis*: *pp* ∈ *Subdis* (*Subdis A*) ⟹ *flat A pp* ∈ *Subdis A*
unfolding *flat-flatP* **apply**(*rule flatP-Subdis*) **by** *auto*

lemma *flat-lift-ind*:
assumes *p*: *p* ∈ *Subdis A* **and** *a*: *a* ∈ *A*
shows *flat A* (*lift A ind p*) *a* = *p a*
by (*simp add*: *a flatP-lift-ind flat-flatP image-subsetI p subset-eq*)

lemma *flat-ind*:
assumes *p* ∈ *Subdis A* **and** *a* ∈ *A*
shows *flat A* (*ind p*) *a* = *p a*
by (*metis assms flatP-ind flat-flatP subset-iff*)

lemma *flat-lift*:
assumes *f*: ∀ *a*∈*A*. *f a* ∈ *B* **and** *pp*: *pp* ∈ *Subdis* (*Subdis A*) **and** *b*: *b* ∈ *B*
shows *flat B* (*lift* (*Subdis A*) (*lift A f*) *pp*) *b* = *lift A f* (*flat A pp*) *b*
unfolding *flat-flatP* **apply**(*rule flatP-lift*)
using *assms* **using** *lift-Subdis* **by** *fastforce*+

lemma *flat-flat-lift*:
assumes *ppp*: *ppp* ∈ *Subdis* (*Subdis* (*Subdis A*)) **and** *a*: *a* ∈ *A*
shows *flat A* (*flat* (*Subdis A*) *ppp*) *a* = *flat A* (*lift* (*Subdis* (*Subdis A*)) (*flat A*) *ppp*) *a*
unfolding *flat-flatP* **apply**(*rule flatP-flatP-lift*)
using *assms* **by** (*auto intro*: *flatP-Subdis*)

definition *dflat* :: *'a set* ⇒ ((*'a* ⇒ *real*) ⇒ *real*) ⇒ (*'a* ⇒ *real*) **where**
dflat A pp ≡ λ*a*. *isum* (λ*p*. *pp p* ∗ *p a*) (*Dis A*)

lemma *dflat-flatP*: *dflat A = flatP (Dis A)*
by (*auto simp*: *dflat-def flatP-def*)

lemma *dflat-ext*: *dflat A = ext (Dis A) id*
unfolding *dflat-flatP* **using** *flatP-ext* **.**

lemma *dflat-eq*: $\forall p{\in}Dis\ A.\ pp1\ p = pp2\ p \Longrightarrow a \in A \Longrightarrow dflat\ A\ pp1\ a = dflat\ A\ pp2\ a$
by (*metis dflat-flatP flatP-eq*)

lemma *dflat-Subdis*: *pp ∈ Subdis (Dis A) ⟹ dflat A pp ∈ Subdis A*
by (*simp add*: *Dis-incl-Subdis dflat-flatP flatP-Subdis*)

lemma *dflat-Dis*: *pp ∈ Dis (Dis A) ⟹ dflat A pp ∈ Dis A*
by (*simp add*: *dflat-ext ext-Dis*)

lemma *dflat-lift-ind*:
assumes *p*: *p ∈ Dis A* **and** *a*: *a ∈ A*
shows *dflat A (lift A ind p) a = p a*
by (*metis Dis-Subdis-mono Dis-ind a dflat-flatP flatP-lift-ind image-subsetI p subsetI*)

lemma *dflat-ind*:
assumes *p ∈ Dis A* **and** *a ∈ A*
shows *dflat A (ind p) a = p a*
by (*metis Dis-incl-Subdis assms dflat-flatP flatP-ind*)

lemma *dflat-lift-Subdis*:
assumes *f*: $\forall a{\in}A.\ f\ a \in B$ **and** *pp*: *pp ∈ Subdis (Dis A)* **and** *b*: *b ∈ B*
shows *dflat B (lift (Dis A) (lift A f) pp) b = lift A f (dflat A pp) b*
unfolding *dflat-flatP* **apply**(*rule flatP-lift*)
using *assms* **using** *lift-Subdis Dis-incl-Subdis* **by** (*fastforce simp*: *lift-Dis*)+

corollary *dflat-lift*:
assumes *f*: $\forall a{\in}A.\ f\ a \in B$ **and** *pp*: *pp ∈ Dis (Dis A)* **and** *b*: *b ∈ B*
shows *dflat B (lift (Dis A) (lift A f) pp) b = lift A f (dflat A pp) b*
by (*meson Dis-incl-Subdis b dflat-lift-Subdis f in-mono pp*)

lemma *dflat-dflat-lift-Subdis*:
assumes *ppp*: *ppp ∈ Subdis (Dis (Dis A))* **and** *a*: *a ∈ A*
shows *dflat A (dflat (Dis A) ppp) a = dflat A (lift (Dis (Dis A)) (dflat A) ppp)*

a
by (*metis Dis-incl-Subdis Dis-incl-Subdis a dflat-Dis dflat-flatP dflat-flatP flatP-flatP-lift ppp*)

corollary *dflat-dflat-lift*:
assumes *ppp*: *ppp* ∈ *Dis* (*Dis* (*Dis A*)) **and** *a*: *a* ∈ *A*
shows *dflat A* (*dflat* (*Dis A*) *ppp*) *a* = *dflat A* (*lift* (*Dis* (*Dis A*)) (*dflat A*) *ppp*)
a
by (*meson Dis-incl-Subdis a dflat-dflat-lift-Subdis in-mono ppp*)

lemma *dflat-from-flat*:
assumes *pp*: *pp* ∈ *Subdis* (*Dis A*) **and** *a*: *a* ∈ *A*
shows *dflat A pp a* = *flat A* (λ*p*. *if p* ∈ *Dis A then pp p else 0*) *a*
using *assms* **unfolding** *dflat-def flat-def* **apply**(*intro isum-zeros-cong*)
subgoal apply(*rule disjI1*) **apply**(*rule Subdis-sboundedL*[*of - - - 1*])
by (*auto simp*: *Subdis-mono Subdis-le-1*)
subgoal by *auto*
subgoal using *Dis-incl-Subdis* **by** *auto*
subgoal by *auto* **.**

lemma *dflat-flat*:
assumes *pp*: *pp* ∈ *Subdis* (*Dis A*) **and** *a*: *a* ∈ *A* **and** ∀ *p*∈*Subdis A* − *Dis A*. *pp p* = *0*
shows *dflat A pp a* = *flat A pp a*
by (*simp add*: *assms dflat-from-flat flat-eq*)

lemma *dflat-flat′*:
assumes *pp*: *pp* ∈ *Dis* (*Dis A*) **and** *a*: *a* ∈ *A* **and** ∀ *p*∈*Subdis A* − *Dis A*. *pp p* = *0*
shows *dflat A pp a* = *flat A pp a*
by (*meson assms Dis-incl-Subdis dflat-flat in-mono*)

2.3 Expectation

definition *expd* :: *′a set* ⇒ (*′a* ⇒ *real*) ⇒ (*′a* ⇒ *real*) ⇒ *real* **where**
expd A p X ≡ *isum* (λ*a*. *p a* ∗ *X a*) *A*

lemma *ext-expd*: *ext A f p b* = *expd A p* (λ*a*. *f a b*)
unfolding *ext-def expd-def* **by** *auto*

lemma *expd-ge-0′*:

assumes $p \in$ *Subdis A* **and** *positive f A* **and** *sbounded* $(\lambda a.\ p\ a * f\ a)\ A$
shows *expd A p f* ≥ 0
by (*metis* (*mono-tags*, *lifting*) *positive-def Subdis-def assms expd-def isum-ge-0 mem-Collect-eq mult-nonneg-nonneg*)

lemma *expd-ge-0*:
assumes *p*: $p \in$ *Subdis A* **and** *f*: *positive f A* $\forall\, a{\in}A.\ f\ a \leq x$
shows *expd A p f* ≥ 0
by (*meson Subdis-sboundedL expd-ge-0' assms*)

lemma *expd-le-upper*:
assumes *p*: $p \in$ *Subdis A* **and** *f*: *positive f A* $\forall\, a{\in}A.\ f\ a \leq x$ **and** *x*: $x \geq 0$
shows *expd A p f* $\leq x$
unfolding *expd-def* **apply**(*rule order.trans*[*of - isum* $(\lambda a.\ p\ a * x)\ A$])
subgoal apply(*rule isum-mono*)
subgoal apply(*rule sbounded-multR*)
using *x f*(*2*) *Subdis-def p* **by** *auto*
subgoal by (*meson Subdis-ge-0 f*(*2*) *mult-left-mono p*) **.**
subgoal apply(*subst isum-distribR*[*symmetric*])
using *assms x* **unfolding** *Subdis-def*
by (*auto simp*: *mult.commute mult-left-le*) **.**

lemma *expd-ge-lower-Subdis*:
assumes *p*: $p \in$ *Subdis A* **and** *f*: $\forall\, a{\in}A.\ f\ a \geq x$ **and** *x*: $x \geq 0$
and *pf*: *sbounded* $(\lambda a.\ p\ a * f\ a)\ A$
shows *expd A p f* $\geq x *$ *isum p A*
proof−
have *f2*: *positive f A* **using** *f x* **unfolding** *positive-def* **by** *auto*
show *?thesis*
unfolding *expd-def* **apply**(*rule order.trans*[*of - isum* $(\lambda a.\ p\ a * x)\ A$])
apply (*smt* (*verit*, *best*) *Subdis-def isum-cong isum-distribL mem-Collect-eq mult-commute-abs p x*)
by (*smt* (*verit*) *Subdis-ge-0 f isum-mono mult-mono p pf x*)
qed

lemma *expd-ge-lower-Dis'*:
assumes *p*: $p \in$ *Dis A* **and** *f*: $\forall\, a{\in}A.\ f\ a \geq x$ **and** *x*: $x \geq 0$
and *pf*: *sbounded* $(\lambda a.\ p\ a * f\ a)\ A$
shows *expd A p f* $\geq x$
apply(*rule order.trans*[*OF - expd-ge-lower-Subdis*])
using *assms Dis-Subdis-mono* **by** (*auto simp*: *Dis-isum-1*)

lemma *expd-ge-lower-Dis*:
assumes *p*: $p \in$ *Dis A* **and** *f*: $\forall\, a{\in}A.\ f\ a \geq x\ \forall\, a{\in}A.\ f\ a \leq y$
and *xy*: $x \geq 0\ y \geq 0$
shows *expd A p f* $\geq x$
proof−
have *sbounded* $(\lambda a.\ p\ a * f\ a)\ A$
using *Dis-incl-Subdis Subdis-sboundedL f*(*2*) *p* **by** *blast*

thus *?thesis* **using** *assms* **apply**(*intro expd-ge-lower-Dis'*) **by** *auto*
qed

lemma *expd-ge01*:
assumes *p*: $p \in Subdis\ A$ **and** *f*: $\forall\, a{\in}A.\ f\ a \geq 0\ \forall\, a{\in}A.\ f\ a \leq 1$
shows $expd\ A\ p\ f \geq 0$
apply(*rule order.trans*[*OF - expd-ge-lower-Subdis*])
using *assms Dis-Subdis-mono* **by** (*auto simp*: *Subdis-sboundedL*)

lemma *expd-le01*:
assumes *p*: $p \in Subdis\ A$ **and** *f*: $\forall\, a{\in}A.\ f\ a \geq 0\ \forall\, a{\in}A.\ f\ a \leq 1$
shows $expd\ A\ p\ f \leq 1$
by (*simp add*: *positive-def expd-le-upper assms*)

lemma *expd-const-Subdis*[*simp*]:
assumes *p*: $p \in Subdis\ A$ **and** $c \geq 0$
shows $expd\ A\ p\ (\lambda\text{-}.\ c) = c * isum\ p\ A$
unfolding *expd-def* **apply**(*subst isum-distribR*[*symmetric*])
using *assms* **unfolding** *Subdis-def* **by** *auto*

lemma *expd-const-le*:
assumes *p*: $p \in Subdis\ A$ **and** $c \geq 0$
shows $expd\ A\ p\ (\lambda\text{-}.\ c) \leq c$
apply(*subst expd-const-Subdis*[*OF assms*])
using *assms* **by** (*simp add*: *Subdis-def mult-left-le*)

lemma *expd-const-Dis*[*simp*]:
assumes *p*: $p \in Dis\ A$ **and** $c \geq 0$
shows $expd\ A\ p\ (\lambda\text{-}.\ c) = c$
apply(*subst expd-const-Subdis*)
using *assms* **unfolding** *Dis-def Subdis-def* **by** *auto*

lemma *expd-eq-ct-iff*[*simp*]:
assumes $p \in Subdis\ A\ c > 0$
shows $expd\ A\ p\ (\lambda\text{-}.\ c) = c \longleftrightarrow p \in Dis\ A$
proof −
have $c = c * isum\ p\ A \longrightarrow p \in Dis\ A$
using *Dis-def assms* **by** *fastforce*
then show *?thesis*
by (*metis* (*no-types*) *assms expd-const-Dis expd-const-Subdis less-le*)
qed

lemma *expd-0*[*simp*]: $expd\ A\ p\ (\lambda\text{-}.\ 0) = 0$
unfolding *expd-def* **by** *simp*

lemma *expd-1-le-1*: $p \in Subdis\ A \Longrightarrow expd\ A\ p\ (\lambda\text{-}.\ 1) \leq 1$
using *expd-const-le zero-le-one* **by** *blast*

lemma *expd-1-eq-1*[*simp*]: $p \in Dis\ A \Longrightarrow expd\ A\ p\ (\lambda\text{-}.\ 1) = 1$

by *simp*

lemma *expd-plus′*:
assumes *p*: *p* ∈ *Subdis A*
and *f1*: *positive f1 A sbounded* (λ*a*. *p a* ∗ *f1 a*) *A*
and *f2*: *positive f2 A sbounded* (λ*a*. *p a* ∗ *f2 a*) *A*
shows *expd A p* (λ*a*. *f1 a* + *f2 a*) = *expd A p f1* + *expd A p f2*
unfolding *expd-def* **apply**(*rule trans*[*of* - *isum* (λ*a*. *p a* ∗ *f1 a* + *p a* ∗ *f2 a*) *A*])

subgoal apply(*rule isum-cong*) **by** (*simp-all add*: *distrib-left*)
subgoal apply(*subst isum-plus*)
using *assms* **unfolding** *Subdis-def positive-def* **by** *auto* **.**

lemma *expd-plus*:
assumes *p*: *p* ∈ *Subdis A*
and *f1*: *positive f1 A bdd-above* (*f1*‘*A*)
and *f2*: *positive f2 A bdd-above* (*f2*‘*A*)
shows *expd A p* (λ*a*. *f1 a* + *f2 a*) = *expd A p f1* + *expd A p f2*
apply(*rule expd-plus′*)
using *assms* **unfolding** *bdd-above-def* **by** (*auto simp add*: *Subdis-sboundedL*)

lemma *expd-mult′*:
assumes *p*: *p* ∈ *Subdis A*
and *f*: *positive f A sbounded* (λ*a*. *p a* ∗ *f a*) *A* **and** *c*: *c* ≥ *0*
shows *expd A p* (λ*a*. *c* ∗ *f a*) = *c* ∗ *expd A p f*
unfolding *expd-def* **unfolding** *mult.left-commute* **apply**(*rule isum-distribL*[*symmetric*])
using *assms* **unfolding** *Subdis-def positive-def* **by** *auto*

lemma *expd-mult*:
assumes *p*: *p* ∈ *Subdis A*
and *f*: *positive f A bdd-above* (*f*‘*A*) **and** *c*: *c* ≥ *0*
shows *expd A p* (λ*a*. *c* ∗ *f a*) = *c* ∗ *expd A p f*
apply(*rule expd-mult′*)
using *assms* **unfolding** *bdd-above-def* **by** (*auto simp*: *Subdis-sboundedL*)

end