Von Neumann Morgenstern Utility Theorem *

Julian Parsert   Cezary Kaliszyk

December 14, 2021

Abstract

Utility functions form an essential part of game theory and economics. In order to guarantee the existence of utility functions most of the time sufficient properties are assumed in an axiomatic manner. One famous and very common set of such assumptions is that of expected utility theory. Here, the rationality, continuity, and independence of preferences is assumed. The von-Neumann-Morgenstern Utility theorem shows that these assumptions are necessary and sufficient for an expected utility function to exists. This theorem was proven by Neumann and Morgenstern in "Theory of Games and Economic Behavior" which is regarded as one of the most influential works in game theory.

We formalize these results in Isabelle/HOL. The formalization includes formal definitions of the underlying concepts including continuity and independence of preferences.

Contents

*This work is supported by the Austrian Science Fund (FWF) project P26201 and the European Research Council (ERC) grant no 714034 *SMART*.

theory *PMF-Composition*
imports
HOL−Probability.Probability
begin

1 Composition of Probability Mass functions

definition *mix-pmf* :: *real* ⇒ *'a pmf* ⇒ *'a pmf* ⇒ *'a pmf* **where**
mix-pmf α *p q* = (*bernoulli-pmf* α) ≫= (λX. *if X then p else q*)

lemma *pmf-mix*: *a* ∈ {*0..1*} ⟹ *pmf* (*mix-pmf a p q*) *x* = *a* ∗ *pmf p x* + (*1* − *a*) ∗ *pmf q x*
by (*simp add*: *mix-pmf-def pmf-bind*)

lemma *pmf-mix-deeper*: *a* ∈ {*0..1*} ⟹ *pmf* (*mix-pmf a p q*) *x* = *a* ∗ *pmf p x* + *pmf q x* − *a* ∗ *pmf q x*
by (*simp add*: *left-diff-distrib' pmf-mix*)

lemma *bernoulli-pmf-0* [*simp*]: *bernoulli-pmf 0* = *return-pmf False*
by (*intro pmf-eqI*) (*auto simp*: *bernoulli-pmf.rep-eq*)

lemma *bernoulli-pmf-1* [*simp*]: *bernoulli-pmf 1* = *return-pmf True*
by (*intro pmf-eqI*) (*auto simp*: *bernoulli-pmf.rep-eq*)

lemma *pmf-mix-0* [*simp*]: *mix-pmf 0 p q* = *q*
by (*simp add*: *mix-pmf-def bind-return-pmf*)

lemma *pmf-mix-1* [*simp*]: *mix-pmf 1 p q* = *p*
by (*simp add*: *mix-pmf-def bind-return-pmf*)

lemma *set-pmf-mix*: *a* ∈ {*0*<..<*1*} ⟹ *set-pmf* (*mix-pmf a p q*) = *set-pmf p* ∪ *set-pmf q*
by (*auto simp add*: *mix-pmf-def split*: *if-splits*)

lemma *set-pmf-mix-eq*: *a* ∈ {*0..1*} ⟹ *mix-pmf a p p* = *p*
by (*simp add*: *mix-pmf-def*)

lemma *pmf-equiv-intro*[*intro*]:
assumes ⋀*e*. *e* ∈ *set-pmf p* ⟹ *pmf p e* = *pmf q e*
assumes ⋀*e*. *e* ∈ *set-pmf q* ⟹ *pmf q e* = *pmf p e*

shows $p = q$
by (*metis assms*(*2*) *less-irrefl pmf-neq-exists-less pmf-not-neg set-pmf-iff*)

lemma *pmf-equiv-intro1*[*intro*]:
assumes $\bigwedge e.\ e \in$ *set-pmf* $p \Longrightarrow$ *pmf* $p\ e =$ *pmf* $q\ e$
shows $p = q$
by (*standard*, *auto simp*: *assms*, *metis assms set-pmf-iff assms linorder-not-le order-refl pmf-neq-exists-less pmf-not-neg set-pmf-iff*)

lemma *pmf-inverse-switch-eqals*:
assumes $a \in \{0..1\}$
shows *mix-pmf* $a\ p\ q =$ *mix-pmf* $(1-a)\ q\ p$
proof −
have *fst*: $\forall x \in$ *set-pmf* p. *pmf* (*mix-pmf* $a\ p\ q$) $x =$ *pmf* (*mix-pmf* $(1-a)\ q\ p$) x
proof
fix x
assume $x \in$ *set-pmf* p
have *pmf* (*mix-pmf* $a\ p\ q$) $x = a *$ *pmf* $p\ x + (1 - a) *$ *pmf* $q\ x$
using *pmf-mix*[*of a p q x*] *assms* **by** *blast*
also have ... $= a *$ *pmf* $p\ x +$ *pmf* $q\ x - a *$ *pmf* $q\ x$
by (*simp add*: *left-diff-distrib*)
from *pmf-mix*[*of* $1-a$ *q p x*] *assms*
have *pmf* (*mix-pmf* $(1 - a)\ q\ p$) $x = (1 - a) *$ *pmf* $q\ x + (1 - (1 - a)) *$ *pmf* $p\ x$
by *auto*
then show *pmf* (*mix-pmf* $a\ p\ q$) $x =$ *pmf* (*mix-pmf* $(1 - a)\ q\ p$) x
using *calculation* **by** *auto*
qed
have $\forall x \in$ *set-pmf* q. *pmf* (*mix-pmf* $a\ p\ q$) $x =$ *pmf* (*mix-pmf* $(1-a)\ q\ p$) x
proof
fix x
assume $x \in$ *set-pmf* q
have *pmf* (*mix-pmf* $a\ p\ q$) $x = a *$ *pmf* $p\ x + (1 - a) *$ *pmf* $q\ x$
using *pmf-mix*[*of a p q x*] *assms* **by** *blast*
also have ... $= a *$ *pmf* $p\ x +$ *pmf* $q\ x - a *$ *pmf* $q\ x$
by (*simp add*: *left-diff-distrib*)
from *pmf-mix*[*of* $1-a$ *q p x*] *assms*
show *pmf* (*mix-pmf* $a\ p\ q$) $x =$ *pmf* (*mix-pmf* $(1 - a)\ q\ p$) x
using *calculation* **by** *auto*
qed
then have $\forall x \in$ *set-pmf* (*mix-pmf* $a\ p\ q$). *pmf* (*mix-pmf* $a\ p\ q$) $x =$ *pmf* (*mix-pmf* $(1-a)\ q\ p$) x
by (*metis* (*no-types*) *fst add-0-left assms mult-eq-0-iff pmf-mix set-pmf-iff*)
thus *?thesis*
by (*simp add*: *pmf-equiv-intro1*)
qed

lemma *mix-pmf-comp-left-div*:
assumes $\alpha \in \{0..(1::real)\}$

and $\beta \in \{0..(1::real)\}$
assumes $\alpha > \beta$
shows *pmf* (*mix-pmf* (β/α) (*mix-pmf* α *p q*) *q*) *e* $= \beta *$ *pmf p e* + *pmf q e* $-$ $\beta *$ *pmf q e*
proof−
let *?l* = (*mix-pmf* (β/α) (*mix-pmf* α *p q*) *q*)
have *fst*: *pmf ?l e* $= (\beta/\alpha) *$ *pmf* (*mix-pmf* α *p q*) *e* $+ (1-\beta/\alpha) *$ *pmf q e*
by (*meson assms*(*1*) *assms*(*2*) *assms*(*3*) *atLeastAtMost-iff less-divide-eq-1 less-eq-real-def not-less pmf-mix zero-le-divide-iff*)
then have *pmf* (*mix-pmf* α *p q*) *e* $= \alpha *$ *pmf p e* $+ (1 - \alpha) *$ *pmf q e*
using *pmf-mix*[*of* α *p q*] *assms*(*2*) *assms*(*3*) *assms*(*1*) **by** *blast*
have *pmf ?l e* $= (\beta/\alpha) * (\alpha *$ *pmf p e* $+ (1 - \alpha) *$ *pmf q e*) $+ (1-\beta/\alpha) *$ *pmf q e*
using *fst assms*(*1*) *pmf-mix* **by** *fastforce*
then have *pmf ?l e* $= ((\beta/\alpha) *\alpha *$ *pmf p e* $+ (\beta/\alpha) *(1 - \alpha) *$ *pmf q e*) $+ (1-\beta/\alpha) *$ *pmf q e*
using *fst assms*(*1*) **by** (*metis mult.assoc ring-class.ring-distribs*(*1*))
then have $*$: *pmf ?l e* $= (\beta *$ *pmf p e* $+ (\beta/\alpha) *(1 - \alpha) *$ *pmf q e*) $+ (1-\beta/\alpha) *$ *pmf q e*
using *fst assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *auto*
then have *pmf ?l e* $= (\beta *$ *pmf p e* $+ ((\beta/\alpha) - (\beta/\alpha)*\alpha) *$ *pmf q e*) $+ (1-\beta/\alpha) *$ *pmf q e*
using *fst assms*(*1*) *assms*(*2*) *assms*(*3*) **by** (*simp add*: $*$ *diff-divide-distrib right-diff-distrib'*)
then have *pmf ?l e* $= (\beta *$ *pmf p e* $+ ((\beta/\alpha) - \beta) *$ *pmf q e*) $+ (1-\beta/\alpha) *$ *pmf q e*
using *fst assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *auto*
then have *pmf ?l e* $= (\beta *$ *pmf p e* $+ (\beta/\alpha) *$ *pmf q e* $- \beta *$ *pmf q e*) $+ 1*$ *pmf q e*$-\beta/\alpha*$ *pmf q e*
by (*simp add*: *left-diff-distrib*)
thus *?thesis*
by *linarith*
qed

lemma *mix-pmf-comp-with-dif-equiv*:
assumes $\alpha \in \{0..(1::real)\}$
and $\beta \in \{0..(1::real)\}$
assumes $\alpha > \beta$
shows *mix-pmf* (β/α) (*mix-pmf* α *p q*) *q* = *mix-pmf* β *p q* (**is** *?l* = *?r*)
proof (*rule pmf-equiv-intro1*[*symmetric*])
fix *e*
assume *a*: $e \in$ *set-pmf ?r*
have $e \in$ *set-pmf ?l*
using *a pmf-mix-deeper* **by** (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *mix-pmf-comp-left-div pmf-eq-0-set-pmf*)
then have *pmf ?l e* $= \beta *$ *pmf p e* $- \beta *$ *pmf q e* + *pmf q e*
using *pmf-mix-deeper*[*of* β/α *p q e*] *mix-pmf-comp-left-div*[*of* α β *p q e*] *assms* **by** *auto*
then show *pmf* (*mix-pmf* β *p q*) *e* = *pmf* (*mix-pmf* $(\beta\ /\ \alpha)$ (*mix-pmf* α *p q*)

q) *e*
  by (*metis* (*full-types*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *mix-pmf-comp-left-div pmf-mix-deeper*)
qed

lemma *product-mix-pmf-prob-distrib*:
  assumes $a \in \{0..1\}$
    and $b \in \{0..1\}$
  shows *mix-pmf a* (*mix-pmf b p q*) *q* = *mix-pmf* (*a*∗*b*) *p q*
proof −
  define γ **where** *g*: $\gamma = (a * b)$
  define *l* **where** *l*: *l* = (*mix-pmf b p q*)
  define *r* **where** *r*: *r* = *mix-pmf* (*a*∗*b*) *p q*
  have *y*: $\gamma \in \{0..1\}$
    using *assms*(*2*) *mult-le-one assms g* **by** *auto*
  have *alt-*: ∀ *e* ∈ *set-pmf l*. *pmf r e* = γ ∗ *pmf p e* + *pmf q e* − γ ∗ *pmf q e*
  proof
    fix *e*
    have *pmf r e* = γ ∗ *pmf p e* + (*1*−γ) ∗ *pmf q e*
      using ‹γ ∈ {*0..1*}› *g pmf-mix r* **by** *fastforce*
    moreover have ... = γ ∗ *pmf p e* + *1* ∗ *pmf q e* − γ ∗ *pmf q e*
      by (*simp add*: *algebra-simps*)
    moreover have ... = *pmf* (*mix-pmf* γ *p q*) *e*
      using *calculation g r* **by** *auto*
    moreover have ... = γ ∗ *pmf p e* + *pmf q e* − γ ∗ *pmf q e*
      using *calculation* **by** *auto*
    ultimately show *pmf r e* = γ ∗ *pmf p e* + *pmf q e* − γ ∗ *pmf q e*
      by *auto*
  qed
  have ∀ *e* ∈ *set-pmf r*. *pmf l e* = *b* ∗ *pmf p e* + *pmf q e* − *b* ∗ *pmf q e*
    using *allI pmf-mix-deeper assms*(*2*) *l* **by** *fastforce*
  have *mix-pmf a* (*mix-pmf b p q*) *q* = *mix-pmf* (*a* ∗ *b*) *p q*
  proof (*rule ccontr*)
    assume *neg*:¬*mix-pmf a* (*mix-pmf b p q*) *q* = *mix-pmf* (*a* ∗ *b*) *p q*
    then have *b*: $b \neq 0$
      by (*metis* (*no-types*) *assms*(*1*) *mult-cancel-right2 pmf-mix-0 set-pmf-mix-eq*)
    have *f3*: *b* − (*a* ∗ *b*) > *0* ⟶ *mix-pmf a* (*mix-pmf b p q*) *q* = *mix-pmf* (*a* ∗ *b*)
p q
      by (*metis assms*(*2*) *diff-le-0-iff-le g mix-pmf-comp-with-dif-equiv mult-eq-0-iff nonzero-mult-div-cancel-right not-le order-refl y*)
    thus *False*
      using *b neg assms*(*1*) *assms*(*2*) **by** *auto*
  qed
  then show *?thesis* **by** *auto*
qed

lemma *mix-pmf-subset-of-original*:
  assumes $a \in \{0..1\}$
  shows (*set-pmf* (*mix-pmf a p q*)) ⊆ *set-pmf p* ∪ *set-pmf q*
proof −

```
  have a ∈ {0<..<1} ⟹ ?thesis
    by (simp add: set-pmf-mix)
  moreover have a = 1 ⟹ ?thesis
    by simp
  moreover have a = 0 ⟹ ?thesis
    by simp
  ultimately show ?thesis
    using assms less-eq-real-def by auto
qed

lemma mix-pmf-preserves-finite-support:
  assumes a ∈ {0..1}
  assumes finite (set-pmf p)
    and finite (set-pmf q)
  shows finite (set-pmf (mix-pmf a p q))
 by (meson assms(1) assms(2) assms(3) finite-Un finite-subset mix-pmf-subset-of-original)

lemma ex-certain-iff-singleton-support:
  shows (∃ x. pmf p x = 1) ⟷ card (set-pmf p) = 1
proof (rule iffI, goal-cases)
  case 1
  show ?case
  proof (rule ccontr)
    assume neg: ¬ card (set-pmf p) = 1
    then have card (set-pmf p) ≠ 1
      by blast
    have finite (set-pmf p)
      by (metis 1 empty-iff finite.emptyI finite-insert insert-iff
          not-le pmf-le-1 pmf-neq-exists-less pmf-nonneg set-pmf-iff set-return-pmf)
    then have sumeq-1: (∑ i ∈ set-pmf p. pmf p i) = 1
      using sum-pmf-eq-1[of set-pmf p p] by auto
    have set-pmf-nemtpy: set-pmf p ≠ {}
      by (simp add: set-pmf-not-empty)
    then have g1: card (set-pmf p) > 1
     by (metis card-0-eq less-one nat-neq-iff neg sum.infinite sumeq-1 zero-neq-one)
    have card (set-pmf p) > 1 ⟶ (∑ i ∈ set-pmf p. pmf p i) > 1
    proof
      assume card (set-pmf p) > 1
      have ∃ x y. pmf p x = 1 ∧ y ≠ x ∧ y ∈ set-pmf p
        using  set-pmf-nemtpy is-singletonI' is-singleton-altdef
        by (metis 1 neg)
      then show (∑ i ∈ set-pmf p. pmf p i) > 1
        by (metis AE-measure-pmf-iff UNIV-I empty-iff insert-iff
            measure-pmf.prob-eq-1 pmf.rep-eq sets-measure-pmf)
    qed
    then have card (set-pmf p) < 1
      using sumeq-1 neg by linarith
    then show False
      using g1 by linarith
```

qed
qed (*metis card-1-singletonE less-numeral-extra*(*1*) *pmf.rep-eq subset-eq sum-pmf-eq-1*[*of set-pmf p p*] *card-gt-0-iff*[*of set-pmf p*] *measure-measure-pmf-finite*[*of set-pmf p*])

We thank Manuel Eberl for suggesting the following two lemmas.

lemma *mix-pmf-partition*:
fixes p :: $'a$ *pmf*
assumes $y \in$ *set-pmf p* *set-pmf p* $- \{y\} \neq \{\}$
obtains *a q* **where** $a \in \{0<..<1\}$ *set-pmf q* = *set-pmf p* $- \{y\}$
p = *mix-pmf a q* (*return-pmf y*)
proof −
from *assms* **obtain** x **where** x: $x \in$ *set-pmf p* $- \{y\}$ **by** *auto*
define a **where** $a = 1 -$ *pmf p y*
have *a-n1*:$a \neq 1$
by (*simp add*: *a-def assms*(*1*) *pmf-eq-0-set-pmf*)
have *pmf p y* $\neq 1$
using *ex-certain-iff-singleton-support* **by** (*metis* (*full-types*) *Diff-cancel assms*(*1*) *assms*(*2*) *card-1-singletonE singletonD*)
hence y: *pmf p y* < 1 **using** *pmf-le-1*[*of p y*] **unfolding** *a-def* **by** *linarith*
hence a: $a > 0$ **by** (*simp add*: *a-def*)
define q **where** q = *embed-pmf* (λz. *if* $z = y$ *then 0 else pmf p z* / a)
have q: *pmf q z* = (*if* $z = y$ *then 0 else pmf p z* / a) **for** z
unfolding *q-def*
proof (*rule pmf-embed-pmf*)
have $1 = (\int^+ x.$ *ennreal* (*pmf p x*) ∂*count-space UNIV*)
by (*rule nn-integral-pmf-eq-1* [*symmetric*])
also have ... = ($\int^+ x.$ *ennreal* (*pmf p x*) ∗ *indicator* $\{y\}$ x +
ennreal (*pmf p x*) ∗ *indicator* ($-\{y\}$) x ∂*count-space UNIV*)
by (*intro nn-integral-cong*) (*auto simp*: *indicator-def*)
also have ... = ($\int^+ x.$ *ennreal* (*pmf p x*) ∗ *indicator* $\{y\}$ x ∂*count-space UNIV*) +
($\int^+ x.$ *ennreal* (*pmf p x*) ∗ *indicator* ($-\{y\}$) x ∂*count-space UNIV*)
by (*subst nn-integral-add*) *auto*
also have ($\int^+ x.$ *ennreal* (*pmf p x*) ∗ *indicator* $\{y\}$ x ∂*count-space UNIV*) = *pmf p y*
by (*subst nn-integral-indicator-finite*) *auto*
also have *ennreal* (*pmf p y*) + ($\int^+ x.$ *ennreal* (*pmf p x*) ∗ *indicator* ($-\{y\}$) x ∂*count-space UNIV*)
− *ennreal* (*pmf p y*) = ($\int^+ x.$ *ennreal* (*pmf p x*) ∗ *indicator* ($-\{y\}$) x ∂*count-space UNIV*)
by *simp*
also have $1 -$ *ennreal* (*pmf p y*) = *ennreal* ($1 -$ *pmf p y*)
by (*subst ennreal-1* [*symmetric*], *subst ennreal-minus*) *auto*
finally have *eq*: ($\int^+ x \in -\{y\}.$ *ennreal* (*pmf p x*)∂*count-space UNIV*) = $1 -$ *pmf p y* **..**
have ($\int^+ x.$ *ennreal* (*if* $x = y$ *then 0 else pmf p x* / a) ∂*count-space UNIV*) =
($\int^+ x.$ *inverse a* ∗ (*ennreal* (*pmf p x*) ∗ *indicator* ($-\{y\}$) x) ∂*count-space UNIV*)

using *a* **by** (*intro nn-integral-cong*) (*auto simp*: *divide-simps ennreal-mult′* [*symmetric*])
also have … = *inverse a* ∗ ($\int^+$ *x*∈−{*y*}. *ennreal* (*pmf p x*) ∂*count-space UNIV*)
using *a* **by** (*subst nn-integral-cmult* [*symmetric*]) (*auto simp*: *ennreal-mult′*)
also note *eq*
also have *ennreal* (*inverse a*) ∗ *ennreal* (*1* − *pmf p y*) = *ennreal* ((*1* − *pmf p y*) / *a*)
using *a* **by** (*subst ennreal-mult′* [*symmetric*]) (*auto simp*: *field-simps*)
also have (*1* − *pmf p y*) / *a* = *1* **using** *y* **by** (*simp add*: *a-def*)
finally show ($\int^+$ *x*. *ennreal* (*if x* = *y then 0 else pmf p x* / *a*) ∂*count-space UNIV*) = *1*
by *simp*
qed (*insert a*, *auto*)
have *mix-pmf* (*1* − *pmf p y*) *q* (*return-pmf y*) = *p*
using *y* **by** (*intro pmf-eqI*) (*auto simp*: *q pmf-mix pmf-le-1 a-def*)
moreover have *set-pmf q* = *set-pmf p* − {*y*}
using *y* **by** (*auto simp*: *q set-pmf-eq a-def*)
ultimately show *?thesis* **using** *that*[*of 1* − *pmf p y q*] *y assms* **by** (*auto simp*: *set-pmf-eq*)
qed

lemma *pmf-mix-induct* [*consumes 2*, *case-names degenerate mix*]:
assumes *finite A set-pmf p* ⊆ *A*
assumes *degenerate*: ⋀*x*. *x* ∈ *A* ⟹ *P* (*return-pmf x*)
assumes *mix*: ⋀*p a y*. *set-pmf p* ⊆ *A* ⟹ *a* ∈ {*0*<..<*1*} ⟹ *y* ∈ *A* ⟹ *P p* ⟹ *P* (*mix-pmf a p* (*return-pmf y*))
shows *P p*
proof −
have *finite* (*set-pmf p*) *set-pmf p* ≠ {} *set-pmf p* ⊆ *A*
using *assms*(*1*,*2*) **by** (*auto simp*: *set-pmf-not-empty dest*: *finite-subset*)
thus *?thesis*
proof (*induction set-pmf p arbitrary*: *p rule*: *finite-ne-induct*)
case (*singleton x p*)
hence *p* = *return-pmf x* **using** *set-pmf-subset-singleton*[*of p x*] **by** *auto*
thus *?case* **using** *singleton* **by** (*auto intro*: *degenerate*)
next
case (*insert x B p*)
from *insert.hyps* **have** *x* ∈ *set-pmf p set-pmf p* − {*x*} ≠ {} **by** *auto*
from *mix-pmf-partition*[*OF this*] **obtain** *a q*
where *decomp*: *a* ∈ {*0*<..<*1*} *set-pmf q* = *set-pmf p* − {*x*}
p = *mix-pmf a q* (*return-pmf x*) **by** *blast*
have *P* (*mix-pmf a q* (*return-pmf x*))
using *insert.prems decomp*(*1*) *insert.hyps*
by (*intro mix insert*) (*auto simp*: *decomp*(*2*))
with *decomp*(*3*) **show** *?case* **by** *simp*
qed
qed

lemma *pmf-mix-induct'* [*consumes 2*, *case-names degenerate mix*]:
  assumes *finite A set-pmf p* $\subseteq$ *A*
  assumes *degenerate*: $\bigwedge x.\ x \in A \Longrightarrow P$ (*return-pmf x*)
  assumes *mix*: $\bigwedge p\ q\ a.$ *set-pmf p* $\subseteq A \Longrightarrow$ *set-pmf q* $\subseteq A \Longrightarrow a \in \{0<..<1\}$
$\Longrightarrow$
    $P\ p \Longrightarrow P\ q \Longrightarrow P$ (*mix-pmf a p q*)
  shows *P p*
  using *assms* **by** (*induct p rule*: *pmf-mix-induct*)(*auto*)+

lemma *finite-sum-distribute-mix-pmf*:
  assumes *finite* (*set-pmf* (*mix-pmf a p q*))
  assumes *finite* (*set-pmf p*)
  assumes *finite* (*set-pmf q*)
  shows ($\sum$ *i* $\in$ *set-pmf* (*mix-pmf a p q*). *pmf* (*mix-pmf a p q*) *i*) = ($\sum$ *i*$\in$*set-pmf p*. *a* $*$ *pmf p i*) + ($\sum$ *i*$\in$*set-pmf q*. (*1*$-$*a*) $*$ *pmf q i*)
proof $-$
  have *fst*: ($\sum$ *i* $\in$ *set-pmf* (*mix-pmf a p q*). *pmf* (*mix-pmf a p q*) *i*) = *1*
    using *sum-pmf-eq-1 assms* **by** *auto*
  have ($\sum$ *i*$\in$*set-pmf p*. *a* $*$ *pmf p i*) = *a* $*$ ($\sum$ *i*$\in$*set-pmf p*. *pmf p i*)
    by (*simp add*: *sum-distrib-left*)
  also have ... = *a* $*$ *1*
    using *assms sum-pmf-eq-1* **by** (*simp add*: *sum-pmf-eq-1*)
  then show *?thesis*
    by (*metis fst add.assoc add-diff-cancel-left' add-uminus-conv-diff assms*(*3*)
      mult.right-neutral order-refl sum-distrib-left sum-pmf-eq-1)
qed

lemma *distribute-alpha-over-sum*:
  shows ($\sum$ *i*$\in$*set-pmf T*. *a* $*$ *pmf p i* $*$ *f i*) = *a* $*$ ($\sum$ *i*$\in$*set-pmf T*. *pmf p i* $*$ *f i*)
  by (*metis* (*mono-tags*, *lifting*) *semiring-normalization-rules*(*18*) *sum.cong sum-distrib-left*)

lemma *sum-over-subset-pmf-support*:
  assumes *finite T*
  assumes *set-pmf p* $\subseteq$ *T*
  shows ($\sum$ *i*$\in$*T*. *a* $*$ *pmf p i* $*$ *f i*) = ($\sum$ *i*$\in$*set-pmf p*. *a* $*$ *pmf p i* $*$ *f i*)
proof $-$
  consider (*eq*) *set-pmf p* = *T* | (*sub*) *set-pmf p* $\subset$ *T*
    using *assms* **by** *blast*
  then show *?thesis*
  proof (*cases*)
  next
    case *sub*
    define *A* **where** *A* = *T* $-$ (*set-pmf p*)
    have *finite* (*set-pmf p*)
      using *assms*(*1*) *assms*(*2*) *finite-subset* **by** *auto*
    moreover have *finite A*
      using *A-def assms*(*1*) **by** *blast*
    moreover have *A* $\cap$ *set-pmf p* = {}
      using *A-def assms*(*1*) **by** *blast*

ultimately have ∗: (∑ *i*∈*T*. *a* ∗ *pmf p i* ∗ *f i*) = (∑ *i*∈*set-pmf p*. *a* ∗ *pmf p i* ∗ *f i*) + (∑ *i*∈*A*. *a* ∗ *pmf p i* ∗ *f i*)
using *sum.union-disjoint* **by** (*metis* (*no-types*) *A-def Un-Diff-cancel2 Un-absorb2 assms*(*2*) *inf.commute inf-sup-aci*(*5*) *sum.union-disjoint*)
have ∀ *e* ∈ *A*. *pmf p e* = *0*
by (*simp add*: *A-def pmf-eq-0-set-pmf*)
hence (∑ *i*∈*A*. *a* ∗ *pmf p i* ∗ *f i*) = *0*
by *simp*
then show *?thesis*
by (*simp add*: ∗)
qed (*auto*)
qed

lemma *expected-value-mix-pmf-distrib*:
assumes *finite* (*set-pmf p*)
and *finite* (*set-pmf q*)
assumes *a* ∈ {*0*<..<*1*}
shows *measure-pmf.expectation* (*mix-pmf a p q*) *f* = *a* ∗ *measure-pmf.expectation p f* + (*1*−*a*) ∗ *measure-pmf.expectation q f*
proof −
have *fn*: *finite* (*set-pmf* (*mix-pmf a p q*))
using *mix-pmf-preserves-finite-support assms less-eq-real-def* **by** *auto*
have *subsets*: *set-pmf p* ⊆ *set-pmf* (*mix-pmf a p q*) *set-pmf q* ⊆ *set-pmf* (*mix-pmf a p q*)
using *assms assms set-pmf-mix* **by**(*fastforce*)+
have ∗: (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). *a* ∗ *pmf p i* ∗ *f i*) = *a* ∗ (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). *pmf p i* ∗ *f i*)
by (*metis* (*mono-tags*, *lifting*) *mult.assoc sum.cong sum-distrib-left*)
have ∗∗: (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). (*1*−*a*) ∗ *pmf q i* ∗ *f i*) = (*1*−*a*) ∗ (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). *pmf q i* ∗ *f i*)
using *distribute-alpha-over-sum*[*of* (*1* − *a*) *q f* (*mix-pmf a p q*)] **by** *auto*
have ∗∗∗: *measure-pmf.expectation* (*mix-pmf a p q*) *f* = (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). *pmf* (*mix-pmf a p q*) *i* ∗ *f i*)
by (*metis fn pmf-integral-code-unfold pmf-integral-pmf-set-integral pmf-set-integral-code-alt-finite*)
also have *g*: ... = (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). (*a* ∗ *pmf p i* + (*1*−*a*) ∗ *pmf q i*) ∗ *f i*)
using *pmf-mix*[*of a p q*] *assms*(*3*) **by** *auto*
also have ∗∗∗∗: ... = (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). *a* ∗ *pmf p i* ∗ *f i* + (*1*−*a*) ∗ *pmf q i* ∗ *f i*)
by (*simp add*: *mult.commute ring-class.ring-distribs*(*1*))
also have *f*: ... = (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). *a* ∗ *pmf p i* ∗ *f i*) + (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). (*1*−*a*) ∗ *pmf q i* ∗ *f i*)
by (*simp add*: *sum.distrib*)
also have ... = *a* ∗ (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). *pmf p i* ∗ *f i*) + (*1*−*a*) ∗ (∑ *i* ∈ *set-pmf* (*mix-pmf a p q*). *pmf q i* ∗ *f i*)
using ∗ ∗∗ **by** *simp*
also have *h*: ... = *a* ∗ (∑ *i* ∈ *set-pmf p*. *pmf p i* ∗ *f i*) + (*1*−*a*) ∗ (∑ *i* ∈ *set-pmf q*. *pmf q i* ∗ *f i*)

proof −
have ($\sum$ *i* ∈ *set-pmf* (*mix-pmf a p q*). *pmf p i* ∗ *f i*) = ($\sum$ *i* ∈ *set-pmf p*. *pmf p i* ∗ *f i*)
using *subsets sum-over-subset-pmf-support*[*of* (*mix-pmf a p q*) *p 1 f*] *fn* **by** *auto*
moreover have ($\sum$ *i* ∈ *set-pmf* (*mix-pmf a p q*). *pmf q i* ∗ *f i*) = ($\sum$ *i* ∈ *set-pmf q*. *pmf q i* ∗ *f i*)
using *subsets sum-over-subset-pmf-support*[*of* (*mix-pmf a p q*) *q 1 f*] *fn* **by** *auto*
ultimately show *?thesis*
by (*simp*)
qed
finally show *?thesis*
proof −
have ($\sum$ *i*∈*set-pmf q*. *pmf q i* ∗ *f i*) = *measure-pmf.expectation q f*
by (*metis* (*full-types*) *assms*(*2*) *pmf-integral-code-unfold pmf-integral-pmf-set-integral pmf-set-integral-code-alt-finite*)
moreover have ($\sum$ *i*∈*set-pmf p*. *pmf p i* ∗ *f i*) = *measure-pmf.expectation p f*
by (*metis* (*full-types*) *assms*(*1*) *pmf-integral-code-unfold pmf-integral-pmf-set-integral pmf-set-integral-code-alt-finite*)
ultimately show *?thesis*
by (*simp add*: ∗ ∗∗ ∗∗∗ ∗∗∗∗ *f g h*)
qed
qed

lemma *expected-value-mix-pmf*:
assumes *finite* (*set-pmf p*)
and *finite* (*set-pmf q*)
assumes *a* ∈ {*0..1*}
shows *measure-pmf.expectation* (*mix-pmf a p q*) *f* = *a* ∗ *measure-pmf.expectation p f* + (*1*−*a*) ∗ *measure-pmf.expectation q f*
proof −
consider (*0*) *a* = *0* | (*b*) *a* ∈ {*0*<..<*1*} | (*1*) *a* = *1*
using *assms*(*3*) *less-eq-real-def* **by** *auto*
then show *?thesis*
proof (*cases*)
case *0*
have (*mix-pmf a p q*) = *q*
using *0 pmf-mix-0* **by** *blast*
have *measure-pmf.expectation q f* = (*1*−*a*) ∗ *measure-pmf.expectation q f*
by (*simp add*: *0*)
show *?thesis*
using *0* **by** *auto*
next
case *b*
show *?thesis*
using *assms*(*1*) *assms*(*2*) *b expected-value-mix-pmf-distrib* **by** *blast*
next
case *1*

have (*mix-pmf a p q*) = *p*
using *1 pmf-mix-0* **by** *simp*
then show *?thesis*
by (*simp add*: *1*)
qed
qed

end

theory *Lotteries*
imports
PMF-Composition
HOL−Probability.Probability
begin

2 Lotteries

definition *lotteries-on*
where
lotteries-on Oc = {*p* . (*set-pmf p*) ⊆ *Oc*}

lemma *lotteries-on-subset*:
assumes *A* ⊆ *B*
shows *lotteries-on A* ⊆ *lotteries-on B*
by (*metis* (*no-types*, *lifting*) *Collect-mono assms gfp.leq-trans lotteries-on-def*)

lemma *support-in-outcomes*:
∀ *oc*. ∀ *p* ∈ *lotteries-on oc*. ∀ *a* ∈ *set-pmf p*. *a* ∈ *oc*
by (*simp add*: *lotteries-on-def subsetD*)

lemma *lotteries-on-nonempty*:
assumes *outcomes* ≠ {}
shows *lotteries-on outcomes* ≠ {}
by (*auto simp*: *lotteries-on-def*) (*metis* (*full-types*) *assms*
empty-subsetI ex-in-conv insert-subset set-return-pmf)

lemma *finite-support-one-oc*:
assumes *card outcomes* = *1*
shows ∀ *l* ∈ *lotteries-on outcomes*. *finite* (*set-pmf l*)
by (*metis assms card.infinite finite-subset lotteries-on-def mem-Collect-eq zero-neq-one*)

lemma *one-outcome-card-support-1*:
assumes *card outcomes* = *1*
shows ∀ *l* ∈ *lotteries-on outcomes*. *card* (*set-pmf l*) = *1*
proof
fix *l*
assume *l* ∈ *lotteries-on outcomes*

have *finite outcomes*
  using *assms card.infinite* **by** *force*
then have $l \in$ *lotteries-on outcomes* $\longrightarrow$ *1* $=$ *card* (*set-pmf l*)
  by (*metis assms card-eq-0-iff card-mono finite-support-one-oc le-neq-implies-less*

    less-one lotteries-on-def mem-Collect-eq set-pmf-not-empty)
then show *card* (*set-pmf l*) $=$ *1*
  by (*simp add*: ‹$l \in$ *lotteries-on outcomes*›)
qed

lemma *finite-nempty-ex-degernate-in-lotteries*:
  assumes *out* $\neq$ {}
  assumes *finite out*
  shows $\exists\, e \in$ *lotteries-on out*. $\exists\, x \in$ *out*. *pmf e x* $=$ *1*
proof (*rule ccontr*)
  assume *a*: $\neg$ ($\exists\, e \in$ *lotteries-on out*. $\exists\, x \in$ *out*. *pmf e x* $=$ *1*)
  then have *subset*: $\forall\, e \in$ *lotteries-on out*. *set-pmf e* $\subseteq$ *out*
    using *lotteries-on-def* **by** (*simp add*: *lotteries-on-def*)
  then have $\forall$ *e*. $e \in$ *lotteries-on out* $\longrightarrow$ (($\sum$ *i*$\in$*set-pmf e*. *pmf e i*) $=$ *1*)
    using *sum-pmf-eq-1* **by** (*metis subset assms*(*2*) *finite-subset order-refl*)
  then show *False*
    by (*metis* (*no-types*, *lifting*) *a assms*(*1*) *assms*(*2*) *card.empty card-gt-0-iff*
card-seteq
    empty-subsetI finite.emptyI finite-insert insert-subset lotteries-on-def subsetI
    measure-measure-pmf-finite mem-Collect-eq nat-less-le pmf.rep-eq set-pmf-of-set
)
qed

lemma *card-support-1-probability-1*:
  assumes *card* (*set-pmf p*) $=$ *1*
  shows $\forall\, e \in$ *set-pmf p*. *pmf p e* $=$ *1*
  by(*auto*) (*metis assms card-1-singletonE card-ge-0-finite*
    card-subset-eq ex-card le-numeral-extra(*4*) *measure-measure-pmf-finite*
    pmf.rep-eq singletonD sum-pmf-eq-1 zero-less-one)

lemma *one-outcome-card-lotteries-1*:
  assumes *card outcomes* $=$ *1*
  shows *card* (*lotteries-on outcomes*) $=$ *1*
proof $-$
  obtain x :: $'a$ **where**
    x: *outcomes* $=$ {*x*}
    using *assms card-1-singletonE* **by** *blast*
  have *exl*: $\exists\, l \in$ *lotteries-on outcomes*. *pmf l x* $=$ *1*
    by (*metis x assms card.infinite empty-iff*
      finite-nempty-ex-degernate-in-lotteries insert-iff zero-neq-one)
  have *pmfs*: $\forall\, l \in$ *lotteries-on outcomes*. *set-pmf l* $=$ {*x*}
    by (*simp add*: *lotteries-on-def set-pmf-subset-singleton x*)
  have $\forall\, l \in$ *lotteries-on outcomes*. *pmf l x* $=$ *1*
    by (*simp add*: *lotteries-on-def set-pmf-subset-singleton x*)

then show *?thesis*
by (*metis exl empty-iff is-singletonI′ is-singleton-altdef order-refl pmfs set-pmf-subset-singleton*)
qed

lemma *return-pmf-card-equals-set*:
shows *card* {*return-pmf x* |*x*. *x* ∈ *S*} = *card S*
proof−
have {*return-pmf x* |*x*. *x* ∈ *S*} = *return-pmf* ‘ *S*
by *blast*
also have *card* ... = *card S*
by (*intro card-image*) (*auto simp*: *inj-on-def*)
finally show *card* {*return-pmf x* |*x*. *x* ∈ *S*} = *card S* **.**
qed

lemma *mix-pmf-in-lotteries*:
assumes *p* ∈ *lotteries-on A*
and *q* ∈ *lotteries-on A*
and *a* ∈ {*0*<..<*1*}
shows (*mix-pmf a p q*) ∈ *lotteries-on A*
proof −
have *set-pmf* (*mix-pmf a p q*) = *set-pmf p* ∪ *set-pmf q*
by (*meson assms*(*3*) *set-pmf-mix*)
then show *?thesis*
by (*metis Un-subset-iff assms*(*1*) *assms*(*2*) *lotteries-on-def mem-Collect-eq*)
qed

lemma *card-degen-lotteries-equals-outcomes*:
shows *card* {*x* ∈ *lotteries-on out*. *card* (*set-pmf x*) = *1*} = *card out*
proof −
consider (*empty*) *out* = {} | (*not-empty*) *out* ≠ {}
by *blast*
then show *?thesis*
proof (*cases*)
case *not-empty*
define *DG* **where**
DG: *DG* = {*x* ∈ *lotteries-on out*. *card* (*set-pmf x*) = *1*}
define *AP* **where**
AP: *AP* = {*return-pmf x* |*x*. *x* ∈ *out*}
have ∗∗: *card AP* = *card out*
using *AP return-pmf-card-equals-set* **by** *blast*
have ∗: ∀ *d* ∈ *DG*. *d* ∈ *AP*
proof
fix *l*
assume *l* ∈ *DG*
then have *l* ∈ *lotteries-on out* ∧ *1* = *card* (*set-pmf l*)
using *DG* **by** *force*
then obtain *x* **where**
x: *x* ∈ *out set-pmf l* = {*x*}

by (*metis* (*no-types*) *card-1-singletonE singletonI support-in-outcomes*)
have *return-pmf* $x = l$
using *set-pmf-subset-singleton* $x(2)$ **by** *fastforce*
then show $l \in AP$
using $AP\ x(1)$ **by** *blast*
qed
moreover have $AP = DG$
proof
have $\forall\, e \in AP.\ e \in$ *lotteries-on out*
by(*auto simp*: *lotteries-on-def AP*)
then show $AP \subseteq DG$ **using** $DG\ AP$ **by** *force*
qed (*auto simp*: $*$)
ultimately show *?thesis*
using DG $**$ **by** *blast*
qed (*simp add*: *lotteries-on-def set-pmf-not-empty*)
qed

end

theory *Neumann-Morgenstern-Utility-Theorem*
imports
HOL−Probability.Probability
First-Welfare-Theorem.Utility-Functions
Lotteries
begin

3 Properties of Preferences

3.1 Independent Preferences

Independence is sometimes called substitution

Notice how r is "added" to the right of mix-pmf and the element to the left q/p changes

definition *independent-vnm*
where
independent-vnm $C\ P =$
$(\forall\, p \in C.\ \forall\, q \in C.\ \forall\, r \in C.\ \forall\,(\alpha{::}real) \in \{0{<}..1\}.\ p \succeq[P]\ q \longleftrightarrow$ *mix-pmf* $\alpha\ p\ r \succeq[P]$ *mix-pmf* $\alpha\ q\ r)$

lemma *independent-vnmI1*:
assumes $(\forall\, p \in C.\ \forall\, q \in C.\ \forall\, r \in C.\ \forall\,\alpha \in \{0{<}..1\}.\ p \succeq[P]\ q \longleftrightarrow$ *mix-pmf* $\alpha\ p\ r \succeq[P]$ *mix-pmf* $\alpha\ q\ r)$
shows *independent-vnm* $C\ P$
using *assms independent-vnm-def* **by** *blast*

lemma *independent-vnmI2*:
  assumes $\bigwedge p\ q\ r\ \alpha.\ p \in C \Longrightarrow q \in C \Longrightarrow r \in C \Longrightarrow \alpha \in \{0{<}..1\} \Longrightarrow p \succeq[P]$ $q \longleftrightarrow$ *mix-pmf* $\alpha\ p\ r \succeq[P]$ *mix-pmf* $\alpha\ q\ r$
  shows *independent-vnm C P*
  by (*rule independent-vnmI1*, *standard*, *standard*, *standard*, *standard*, *simp add*: *assms*) (*meson assms greaterThanAtMost-iff*)

lemma *independent-vnm-alt-def*:
  shows *independent-vnm C P* $\longleftrightarrow$ ($\forall p \in C.\ \forall q \in C.\ \forall r \in C.\ \forall \alpha \in \{0{<}..{<}1\}.$ $p \succeq[P]\ q \longleftrightarrow$ *mix-pmf* $\alpha\ p\ r \succeq[P]$ *mix-pmf* $\alpha\ q\ r$) (**is** *?L* $\longleftrightarrow$ *?R*)
proof (*rule iffI*)
  assume *a*: *?R*
  have *independent-vnm C P*
    by(*rule independent-vnmI2*, *simp add*: *a*) (*metis a greaterThanLessThan-iff linorder-neqE-linordered-idom not-le pmf-mix-1*)
  then show *?L* **by** *auto*
qed (*simp add*: *independent-vnm-def*)

lemma *independece-dest-alt*:
  assumes *independent-vnm C P*
  shows ($\forall p \in C.\ \forall q \in C.\ \forall r \in C.\ \forall (\alpha{::}real) \in \{0{<}..1\}.\ p \succeq[P]\ q \longleftrightarrow$ *mix-pmf* $\alpha\ p\ r \succeq[P]$ *mix-pmf* $\alpha\ q\ r$)
proof (*standard*, *standard*, *standard*, *standard*)
  fix $p\ q\ r\ \alpha$
  assume *as1*: $p \in C$
  assume *as2*: $q \in C$
  assume *as3*: $r \in C$
  assume *as4*: $(\alpha{::}real) \in \{0{<}..1\}$
  then show $p \succeq[P]\ q =$ *mix-pmf* $\alpha\ p\ r \succeq[P]$ *mix-pmf* $\alpha\ q\ r$
    using *as1 as2 as3 assms*(*1*) *independent-vnm-def* **by** *blast*
qed

lemma *independent-vnmD1*:
  assumes *independent-vnm C P*
  shows ($\forall p \in C.\ \forall q \in C.\ \forall r \in C.\ \forall \alpha \in \{0{<}..1\}.\ p \succeq[P]\ q \longleftrightarrow$ *mix-pmf* $\alpha\ p\ r \succeq[P]$ *mix-pmf* $\alpha\ q\ r$)
  using *assms independent-vnm-def* **by** *blast*

lemma *independent-vnmD2*:
  fixes $p\ q\ r\ \alpha$
  assumes $\alpha \in \{0{<}..1\}$
    and $p \in C$
    and $q \in C$
    and $r \in C$
  assumes *independent-vnm C P*
  assumes $p \succeq[P]\ q$
  shows *mix-pmf* $\alpha\ p\ r \succeq[P]$ *mix-pmf* $\alpha\ q\ r$
  using *assms independece-dest-alt* **by** *blast*

lemma *independent-vnmD3*:
  fixes $p\ q\ r\ \alpha$
  assumes $\alpha \in \{0{<}..1\}$
    and $p \in C$
    and $q \in C$
    and $r \in C$
  assumes *independent-vnm C P*
  assumes *mix-pmf* α p r $\succeq[P]$ *mix-pmf* α q r
  shows $p \succeq[P]\ q$
  using *assms independece-dest-alt* **by** *blast*

lemma *independent-vnmD4*:
  assumes *independent-vnm C P*
  assumes *refl-on C P*
  assumes $p \in C$
    and $q \in C$
    and $r \in C$
    and $\alpha \in \{0..1\}$
    and $p \succeq[P]\ q$
  shows *mix-pmf* α p r $\succeq[P]$ *mix-pmf* α q r
  using *assms*
  by (*cases* $\alpha = 0 \lor \alpha \in \{0{<}..1\}$,*metis assms*(*1*,*2*,*3*,*4*)
      independece-dest-alt pmf-mix-0 refl-onD, *auto*)

lemma *approx-indep-ge*:
  assumes $x \approx[\mathcal{R}]\ y$
  assumes $\alpha \in \{0..(1{::}real)\}$
  assumes *rpr*: *rational-preference* (*lotteries-on outcomes*) $\mathcal{R}$
    and *ind*: *independent-vnm* (*lotteries-on outcomes*) $\mathcal{R}$
  shows $\forall\, r \in$ *lotteries-on outcomes*. (*mix-pmf* α y r) $\succeq[\mathcal{R}]$ (*mix-pmf* α x r)
proof
  fix r
  assume *a*: $r \in$ *lotteries-on outcomes* (**is** $r \in$ *?lo*)
  have *clct*: $y \succeq[\mathcal{R}]\ x \land$ *independent-vnm ?lo* $\mathcal{R} \land y \in$ *?lo* $\land\ x \in$ *?lo* $\land\ r \in$ *?lo*
    by (*meson a assms*(*1*) *assms*(*2*) *atLeastAtMost-iff greaterThanAtMost-iff*
      ind preference-def rational-preference-def rpr)
  then have *in-lo*: *mix-pmf* α y $r \in$ *?lo* (*mix-pmf* α x r) $\in$ *?lo*
    by (*metis assms*(*2*) *atLeastAtMost-iff greaterThanLessThan-iff*
      less-eq-real-def mix-pmf-in-lotteries pmf-mix-0 pmf-mix-1 a)+
  have $0 = \alpha \lor 0 < \alpha$
    using *assms* **by** *auto*
  then show *mix-pmf* α y r $\succeq[\mathcal{R}]$ *mix-pmf* α x r
    using *in-lo*(*2*) *rational-preference.compl rpr*
    by (*auto*,*blast*) (*meson assms*(*2*) *atLeastAtMost-iff clct*
      greaterThanAtMost-iff independent-vnmD2)
qed

lemma *approx-imp-approx-ind*:

```
  assumes x ≈[ℛ] y
  assumes α ∈ {0..(1::real)}
  assumes rpr: rational-preference (lotteries-on outcomes) ℛ
    and ind: independent-vnm (lotteries-on outcomes) ℛ
  shows ∀ r ∈ lotteries-on outcomes. (mix-pmf α y r) ≈[ℛ] (mix-pmf α x r)
  using approx-indep-ge assms(1) assms(2) ind rpr by blast

lemma geq-imp-mix-geq-right:
  assumes x ≽[ℛ] y
  assumes rpr: rational-preference (lotteries-on outcomes) ℛ
  assumes ind: independent-vnm (lotteries-on outcomes) ℛ
  assumes α ∈ {0..(1::real)}
  shows (mix-pmf α x y) ≽[ℛ] y
proof −
  have xy-p: x ∈ (lotteries-on outcomes) y ∈ (lotteries-on outcomes)
    by (meson assms(1) preference.not-outside rational-preference-def rpr)
      (meson assms(1) preference-def rational-preference-def rpr)
  have (mix-pmf α x y) ∈ (lotteries-on outcomes) (is ?mpf ∈ ?lot)
    using mix-pmf-in-lotteries [of x outcomes y α] xy-p assms(2)
    by (meson approx-indep-ge assms(4) ind preference.not-outside
        rational-preference.compl rational-preference-def)
  have all: ∀ r ∈ ?lot. (mix-pmf α x r) ≽[ℛ] (mix-pmf α y r)
     by (metis assms assms(2) atLeastAtMost-iff greaterThanAtMost-iff indepen-
dece-dest-alt
        less-eq-real-def pmf-mix-0 rational-preference.compl rpr ind xy-p)
  thus ?thesis
    by (metis all assms(4) set-pmf-mix-eq xy-p(2))
qed

lemma geq-imp-mix-geq-left:
  assumes x ≽[ℛ] y
  assumes rpr: rational-preference (lotteries-on outcomes) ℛ
  assumes ind: independent-vnm (lotteries-on outcomes) ℛ
  assumes α ∈ {0..(1::real)}
  shows (mix-pmf α y x) ≽[ℛ] y
proof −
  define β where
    b: β = 1 − α
  have β ∈ {0..1}
    using assms(4) b by auto
  then have mix-pmf β x y ≽[ℛ] y
   using geq-imp-mix-geq-right[OF assms] assms(1) geq-imp-mix-geq-right ind rpr
by blast
  moreover have mix-pmf β x y = mix-pmf α y x
    by (metis assms(4) b pmf-inverse-switch-eqals)
  ultimately show ?thesis
    by simp
qed
```

lemma *sg-imp-mix-sg*:
  assumes $x \succ[\mathcal{R}]\ y$
  assumes *rpr*: *rational-preference* (*lotteries-on outcomes*) $\mathcal{R}$
  assumes *ind*: *independent-vnm* (*lotteries-on outcomes*) $\mathcal{R}$
  assumes $\alpha \in \{0<..(1::real)\}$
  shows (*mix-pmf* α *x y*) $\succ[\mathcal{R}]\ y$
proof −
  have *xy-p*: $x \in$ (*lotteries-on outcomes*) $y \in$ (*lotteries-on outcomes*)
    by (*meson assms*(*1*) *preference.not-outside rational-preference-def rpr*)
      (*meson assms*(*1*) *preference-def rational-preference-def rpr*)
  have (*mix-pmf* α *x y*) $\in$ (*lotteries-on outcomes*) (**is** *?mpf* $\in$ *?lot*)
    using *mix-pmf-in-lotteries* [*of x outcomes y* α] *xy-p assms*(*2*)
    using *assms*(*4*) **by** *fastforce*
  have *all*: $\forall$ *r* $\in$ *?lot*. (*mix-pmf* α *x r*) $\succeq[\mathcal{R}]$ (*mix-pmf* α *y r*)
    by (*metis assms*(*1*,*3*,*4*) *independece-dest-alt ind xy-p*)
  have (*mix-pmf* α *x y*) $\succeq[\mathcal{R}]\ y$
    by (*metis all assms*(*4*) *atLeastAtMost-iff greaterThanAtMost-iff*
      less-eq-real-def set-pmf-mix-eq xy-p(*2*))
  have *all2*: $\forall$ *r* $\in$ *?lot*. (*mix-pmf* α *x r*) $\succ[\mathcal{R}]$ (*mix-pmf* α *y r*)
    using *assms*(*1*) *assms*(*4*) *ind independece-dest-alt xy-p*(*1*) *xy-p*(*2*) **by** *blast*
  then show *?thesis*
    by (*metis assms*(*4*) *atLeastAtMost-iff greaterThanAtMost-iff*
      less-eq-real-def set-pmf-mix-eq xy-p(*2*))
qed

3.2 Continuity

Continuity is sometimes called Archimedean Axiom

definition *continuous-vnm*
  where
    continuous-vnm C P = ($\forall p \in C.\ \forall q \in C.\ \forall r \in C.\ p \succeq[P]\ q \land q \succeq[P]\ r \longrightarrow$
    ($\exists \alpha \in \{0..1\}$. (*mix-pmf* α *p r*) $\approx[P]\ q$))

lemma *continuous-vnmD*:
  assumes *continuous-vnm C P*
  shows ($\forall p \in C.\ \forall q \in C.\ \forall r \in C.\ p \succeq[P]\ q \land q \succeq[P]\ r \longrightarrow$
    ($\exists \alpha \in \{0..1\}$. (*mix-pmf* α *p r*) $\approx[P]\ q$))
  using *continuous-vnm-def assms* **by** *blast*

lemma *continuous-vnmI*:
  assumes $\bigwedge p\ q\ r.\ p \in C \Longrightarrow q \in C \Longrightarrow r \in C \Longrightarrow p \succeq[P]\ q \land q \succeq[P]\ r \Longrightarrow$
    $\exists \alpha \in \{0..1\}$. (*mix-pmf* α *p r*) $\approx[P]\ q$
  shows *continuous-vnm C P*
  by (*simp add*: *assms continuous-vnm-def*)

lemma *mix-in-lot*:
  assumes $x \in$ *lotteries-on outcomes*
    and $y \in$ *lotteries-on outcomes*
    and $\alpha \in \{0..1\}$

shows (*mix-pmf* α *x y*) $\in$ *lotteries-on outcomes*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *less-eq-real-def mix-pmf-in-lotteries* **by** *fast-force*

lemma *non-unique-continuous-unfolding*:
assumes *cnt*: *continuous-vnm* (*lotteries-on outcomes*) $\mathcal{R}$
assumes *rational-preference* (*lotteries-on outcomes*) $\mathcal{R}$
assumes *p* $\succeq[\mathcal{R}]$ *q*
and *q* $\succeq[\mathcal{R}]$ *r*
and *p* $\succ[\mathcal{R}]$ *r*
shows $\exists\,\alpha \in \{0..1\}.\ q \approx[\mathcal{R}]$ *mix-pmf* α *p r*
using *assms*(*1*) *assms*(*2*) *cnt continuous-vnmD assms*
proof −
have $\forall$ *p q. p*$\in$ (*lotteries-on outcomes*) $\wedge$ *q* $\in$ (*lotteries-on outcomes*) $\longleftrightarrow$ *p* $\succeq[\mathcal{R}]$ *q* $\vee$ *q* $\succeq[\mathcal{R}]$ *p*
using *assms rational-preference.compl*[*of lotteries-on outcomes* $\mathcal{R}$]
by (*metis* (*no-types*, *opaque-lifting*) *preference-def rational-preference-def*)
then show *?thesis*
using *continuous-vnmD*[*OF assms*(*1*)] **by** (*metis assms*(*3*) *assms*(*4*))
qed

4 System U start, as per vNM

These are the first two assumptions which we use to derive the first results. We assume rationality and independence. In this system U the von-Neumann-Morgenstern Utility Theorem is proven.

context
fixes *outcomes* :: *'a set*
fixes $\mathcal{R}$
assumes *rpr*: *rational-preference* (*lotteries-on outcomes*) $\mathcal{R}$
assumes *ind*: *independent-vnm* (*lotteries-on outcomes*) $\mathcal{R}$
begin

abbreviation $\mathcal{P} \equiv$ *lotteries-on outcomes*

lemma *relation-in-carrier*:
$x \succeq[\mathcal{R}]\ y \Longrightarrow x \in \mathcal{P} \wedge y \in \mathcal{P}$
by (*meson preference-def rational-preference-def rpr*)

lemma *mix-pmf-preferred-independence*:
assumes $r \in \mathcal{P}$
and $\alpha \in \{0..1\}$
assumes *p* $\succeq[\mathcal{R}]$ *q*
shows *mix-pmf* α *p r* $\succeq[\mathcal{R}]$ *mix-pmf* α *q r*
using *ind* **by** (*metis relation-in-carrier antisym-conv1 assms atLeastAtMost-iff greaterThanAtMost-iff independece-dest-alt pmf-mix-0 rational-preference.no-better-thansubset-rel rpr subsetI*)

```
lemma mix-pmf-strict-preferred-independence:
  assumes r ∈ 𝒫
    and α ∈ {0<..1}
  assumes p ≻[ℛ] q
  shows mix-pmf α p r ≻[ℛ] mix-pmf α q r
  by (meson assms(1) assms(2) assms(3) ind independent-vnmD2
      independent-vnmD3 relation-in-carrier)

lemma mix-pmf-preferred-independence-rev:
  assumes p ∈ 𝒫
    and q ∈ 𝒫
    and r ∈ 𝒫
    and α ∈ {0<..1}
  assumes mix-pmf α p r ≽[ℛ] mix-pmf α q r
  shows p ≽[ℛ] q
proof −
  have mix-pmf α p r ∈ 𝒫
    using assms mix-in-lot relation-in-carrier by blast
  moreover have mix-pmf α q r ∈ 𝒫
    using assms mix-in-lot assms(2) relation-in-carrier by blast
  ultimately show ?thesis
    using ind independent-vnmD3[of α p 𝒫 q r ℛ] assms by blast
qed

lemma x-sg-y-sg-mpmf-right:
  assumes x ≻[ℛ] y
  assumes b ∈ {0<..(1::real)}
  shows x ≻[ℛ] mix-pmf b y x
proof −
  consider b = 1 | b ≠ 1
    by blast
  then show ?thesis
  proof (cases)
    case 2
    have sg: (mix-pmf b x y) ≻[ℛ] y
      using assms(1) assms(2) assms ind rpr sg-imp-mix-sg 2 by fastforce
    have mix-pmf b x y ∈ 𝒫
      by (meson sg preference-def rational-preference-def rpr)
    have mix-pmf b x x ∈ 𝒫
      using relation-in-carrier assms(2) mix-in-lot assms by fastforce
    have b ∈ {0<..<1}
      using 2 assms(2)  by auto
    have mix-pmf b x x ≻[ℛ] mix-pmf b y x
      using mix-pmf-preferred-independence[of x b] assms
       by (meson ‹b ∈ {0<..<1}› greaterThanAtMost-iff greaterThanLessThan-iff
ind
          independece-dest-alt less-eq-real-def preference-def
          rational-preference.axioms(1) relation-in-carrier rpr)
```

then show *?thesis*
using *mix-pmf-preferred-independence*
by (*metis assms*(*2*) *atLeastAtMost-iff greaterThanAtMost-iff less-eq-real-def set-pmf-mix-eq*)
qed (*simp add*: *assms*(*1*))
qed

lemma *neumann-3B-b*:
assumes $u \succ[\mathcal{R}]\ v$
assumes $\alpha \in \{0{<}..{<}1\}$
shows $u \succ[\mathcal{R}]$ *mix-pmf* α *u v*
proof −
have ∗: *preorder-on* $\mathcal{P}$ $\mathcal{R}$ ∧ *rational-preference-axioms* $\mathcal{P}$ $\mathcal{R}$
by (*metis* (*no-types*) *preference-def rational-preference-def rpr*)
have $1 - \alpha \in \{0{<}..1\}$
using *assms*(*2*) **by** *auto*
then show *?thesis*
using ∗ *assms* **by** (*metis atLeastAtMost-iff greaterThanLessThan-iff less-eq-real-def pmf-inverse-switch-eqals x-sg-y-sg-mpmf-right*)
qed

lemma *neumann-3B-b-non-strict*:
assumes $u \succeq[\mathcal{R}]\ v$
assumes $\alpha \in \{0..1\}$
shows $u \succeq[\mathcal{R}]$ *mix-pmf* α *u v*
proof −
have *f2*: *mix-pmf* α (*u*::'*a pmf*) *v* = *mix-pmf* $(1 - \alpha)$ *v u*
using *pmf-inverse-switch-eqals assms*(*2*) **by** *auto*
have $1 - \alpha \in \{0..1\}$
using *assms*(*2*) **by** *force*
then show *?thesis*
using *f2 relation-in-carrier*
by (*metis* (*no-types*) *assms*(*1*) *mix-pmf-preferred-independence set-pmf-mix-eq*)
qed

lemma *greater-mix-pmf-greater-step-1-aux*:
assumes $v \succ[\mathcal{R}]\ u$
assumes $\alpha \in \{0{<}..{<}(1::real)\}$
and $\beta \in \{0{<}..{<}(1::real)\}$
assumes $\beta > \alpha$
shows (*mix-pmf* β *v u*) $\succ[\mathcal{R}]$ (*mix-pmf* α *v u*)
proof −
define *t* **where**
t: *t* = *mix-pmf* β *v u*
obtain γ **where**
g: $\alpha = \beta * \gamma$
by (*metis assms*(*2*) *assms*(*4*) *greaterThanLessThan-iff mult.commute nonzero-eq-divide-eq not-less-iff-gr-or-eq*)
have *g1*: $\gamma > 0 \land \gamma < 1$

by (*metis* (*full-types*) *assms*(*2*) *assms*(*4*) *g greaterThanLessThan-iff less-trans mult.right-neutral mult-less-cancel-left-pos not-le sgn-le-0-iff sgn-pos zero-le-one zero-le-sgn-iff zero-less-mult-iff*)
have *t-in*: *mix-pmf* β *v u* $\in \mathcal{P}$
by (*meson assms*(*1*) *assms*(*3*) *mix-pmf-in-lotteries preference-def rational-preference-def rpr*)
have *v* $\succ$[$\mathcal{R}$] *mix-pmf* (*1* $-$ β) *v u*
using *x-sg-y-sg-mpmf-right*[*of u v 1*$-\beta$] *assms*
by (*metis atLeastAtMost-iff greaterThanAtMost-iff greaterThanLessThan-iff less-eq-real-def pmf-inverse-switch-eqals x-sg-y-sg-mpmf-right*)
have *t* $\succ$[$\mathcal{R}$] *u*
using *assms*(*1*) *assms*(*3*) *ind rpr sg-imp-mix-sg t* **by** *fastforce*
hence *t-s*: *t* $\succ$[$\mathcal{R}$] (*mix-pmf* γ *t u*)
proof $-$
have (*mix-pmf* γ *t u*) $\in \mathcal{P}$
by (*metis assms*(*1*) *assms*(*3*) *atLeastAtMost-iff g1 mix-in-lot mix-pmf-in-lotteries*

not-less order.asym preference-def rational-preference-def rpr t)
have *t* $\succ$[$\mathcal{R}$] *mix-pmf* γ (*mix-pmf* β *v u*) *u*
using *neumann-3B-b*[*of t u* γ] *assms t g1*
by (*meson greaterThanAtMost-iff greaterThanLessThan-iff ind less-eq-real-def rpr sg-imp-mix-sg*)
thus *?thesis*
using *t* **by** *blast*
qed
from *product-mix-pmf-prob-distrib*[*of - β v u*] *assms*
have *mix-pmf* β *v u* $\succ$[$\mathcal{R}$] *mix-pmf* α *v u*
by (*metis t-s atLeastAtMost-iff g g1 greaterThanLessThan-iff less-eq-real-def mult.commute t*)
then show *?thesis* **by** *blast*
qed

5 This lemma is in called step 1 in literature. In Von Neumann and Morgenstern's book this is A:A (albeit more general)

lemma *step-1-most-general*:
assumes *x* $\succ$[$\mathcal{R}$] *y*
assumes $\alpha \in \{0..(1::real)\}$
and $\beta \in \{0..(1::real)\}$
assumes $\alpha > \beta$
shows (*mix-pmf* α *x y*) $\succ$[$\mathcal{R}$] (*mix-pmf* β *x y*)
proof $-$
consider (*ex*) $\alpha = 1 \wedge \beta = 0$ | (*m*) $\alpha \neq 1 \vee \beta \neq 0$
by *blast*
then show *?thesis*
proof (*cases*)
case *m*

```
    consider  β = 0 |  β ≠ 0
      by blast
    then show ?thesis
    proof (cases)
      case 1
      then show ?thesis
        using assms(1) assms(2) assms(4) ind rpr sg-imp-mix-sg by fastforce
    next
      case 2
      let ?d = (β/α)
      have sg: (mix-pmf α x y) ≻[ℛ] y
          using assms(1) assms(2) assms(3) assms(4) ind rpr sg-imp-mix-sg by
fastforce
      have a: α > 0
        using assms(3) assms(4) by auto
      then have div-in: ?d ∈ {0<..1}
        using assms(3) assms(4) 2 by auto
      have mx-p: (mix-pmf α x y) ∈ 𝒫
        by (meson sg preference-def rational-preference-def rpr)
      have y-P: y ∈ 𝒫
        by (meson assms(1) preference-def rational-preference-def rpr)
      hence (mix-pmf ?d (mix-pmf α x y) y) ∈ 𝒫
        using div-in mx-p by (simp add: mix-in-lot)
      have  mix-pmf β (mix-pmf α x y) y ≻[ℛ] y
        using sg-imp-mix-sg[of (mix-pmf α x y) y ℛ outcomes β] sg div-in rpr ind
          a assms(2) 2 assms(3) by auto
      have al1: ∀ r ∈ 𝒫. (mix-pmf α x r) ≻[ℛ] (mix-pmf α y r)
         by (meson a assms(1) assms(2) atLeastAtMost-iff greaterThanAtMost-iff
ind
          independece-dest-alt preference.not-outside rational-preference-def rpr y-P)
      then show ?thesis
        using greater-mix-pmf-greater-step-1-aux assms
        by (metis a div-in divide-less-eq-1-pos greaterThanAtMost-iff
           greaterThanLessThan-iff mix-pmf-comp-with-dif-equiv neumann-3B-b sg)
    qed
  qed (simp add: assms(1))
qed
```

Kreps refers to this lemma as 5.6 c. The lemma after that is also significant.

```
lemma approx-remains-after-same-comp:
  assumes p ≈[ℛ] q
    and r ∈ 𝒫
    and α ∈ {0..1}
  shows mix-pmf α p r ≈[ℛ] mix-pmf α q r
  using approx-indep-ge assms(1) assms(2) assms(3) ind rpr by blast
```

This lemma is the symmetric version of the previous lemma. This lemma is never mentioned in literature anywhere. Even though it looks trivial now, due to the asymmetric nature of the independence axiom, it is not so trivial, and definitely worth mentioning.

lemma *approx-remains-after-same-comp-left*:
  assumes $p \approx[\mathcal{R}]\ q$
    and $r \in \mathcal{P}$
    and $\alpha \in \{0..1\}$
  shows *mix-pmf* $\alpha\ r\ p \approx[\mathcal{R}]$ *mix-pmf* $\alpha\ r\ q$
proof −
  have *1*: $\alpha \leq 1 \wedge \alpha \geq 0\ 1 - \alpha \in \{0..1\}$
    using *assms*(*3*) **by** *auto+*
  have *fst*: *mix-pmf* $\alpha\ r\ p \approx[\mathcal{R}]$ *mix-pmf* $(1-\alpha)\ p\ r$
    using *assms* **by** (*metis mix-in-lot pmf-inverse-switch-eqals rational-preference.compl relation-in-carrier rpr*)
  moreover have *mix-pmf* $\alpha\ r\ p \approx[\mathcal{R}]$ *mix-pmf* $\alpha\ r\ q$
    using *approx-remains-after-same-comp*[*of* - - - α] *pmf-inverse-switch-eqals*[*of* α *p q*] *1*
      pmf-inverse-switch-eqals rpr mix-pmf-preferred-independence[*of* - α - -]
    by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *mix-pmf-preferred-independence*)
  thus *?thesis*
    by *blast*
qed

lemma *mix-of-preferred-is-preferred*:
  assumes $p \succeq[\mathcal{R}]\ w$
  assumes $q \succeq[\mathcal{R}]\ w$
  assumes $\alpha \in \{0..1\}$
  shows *mix-pmf* $\alpha\ p\ q \succeq[\mathcal{R}]\ w$
proof −
  consider $p \succeq[\mathcal{R}]\ q \mid q \succeq[\mathcal{R}]\ p$
    using *rpr assms*(*1*) *assms*(*2*) *rational-preference.compl relation-in-carrier* **by** *blast*
  then show *?thesis*
  proof (*cases*)
    case *1*
    have *mix-pmf* $\alpha\ p\ q \succeq[\mathcal{R}]\ q$
      using *1 assms*(*3*) *geq-imp-mix-geq-right ind rpr* **by** *blast*
    moreover have $q \succeq[\mathcal{R}]\ w$
      using *assms* **by** *auto*
    ultimately show *?thesis* **using** *rpr preference.transitivity*[*of* $\mathcal{P}$ $\mathcal{R}$]
      by (*meson rational-preference-def transE*)
  next
    case *2*
    have *mix-pmf* $\alpha\ p\ q \succeq[\mathcal{R}]\ p$
      using *2 assms geq-imp-mix-geq-left ind rpr* **by** *blast*
    moreover have $p \succeq[\mathcal{R}]\ w$
      using *assms* **by** *auto*
    ultimately show *?thesis* **using** *rpr preference.transitivity*[*of* $\mathcal{P}$ $\mathcal{R}$]
      by (*meson rational-preference-def transE*)
  qed
qed

lemma *mix-of-not-preferred-is-not-preferred*:
  assumes $w \succeq[\mathcal{R}]\ p$
  assumes $w \succeq[\mathcal{R}]\ q$
  assumes $\alpha \in \{0..1\}$
  shows $w \succeq[\mathcal{R}]$ *mix-pmf* $\alpha\ p\ q$
proof −
  consider $p \succeq[\mathcal{R}]\ q \mid q \succeq[\mathcal{R}]\ p$
    using *rpr assms*(*1*) *assms*(*2*) *rational-preference.compl relation-in-carrier* **by** *blast*
  then show *?thesis*
  proof (*cases*)
    case *1*
    moreover have $p \succeq[\mathcal{R}]$ *mix-pmf* $\alpha\ p\ q$
      using *assms*(*3*) *neumann-3B-b-non-strict calculation* **by** *blast*
    moreover show *?thesis*
      using *rpr preference.transitivity*[*of* $\mathcal{P}$ $\mathcal{R}$]
      by (*meson assms*(*1*) *calculation*(*2*) *rational-preference-def transE*)
  next
    case *2*
    moreover have $q \succeq[\mathcal{R}]$ *mix-pmf* $\alpha\ p\ q$
      using *assms*(*3*) *neumann-3B-b-non-strict calculation*
      by (*metis mix-pmf-preferred-independence relation-in-carrier set-pmf-mix-eq*)
    moreover show *?thesis*
      using *rpr preference.transitivity*[*of* $\mathcal{P}$ $\mathcal{R}$]
      by (*meson assms*(*2*) *calculation*(*2*) *rational-preference-def transE*)
  qed
qed

private definition *degenerate-lotteries* **where**
  degenerate-lotteries = $\{x \in \mathcal{P}.$ *card* (*set-pmf* x) = *1*$\}$

private definition *best* **where**
  best = $\{x \in \mathcal{P}.\ (\forall y \in \mathcal{P}.\ x \succeq[\mathcal{R}]\ y)\}$

private definition *worst* **where**
  worst = $\{x \in \mathcal{P}.\ (\forall y \in \mathcal{P}.\ y \succeq[\mathcal{R}]\ x)\}$

lemma *degenerate-total*:
  $\forall e \in$ *degenerate-lotteries*. $\forall m \in \mathcal{P}.\ e \succeq[\mathcal{R}]\ m \vee m \succeq[\mathcal{R}]\ e$
  using *degenerate-lotteries-def rational-preference.compl rpr* **by** *fastforce*

lemma *degen-outcome-cardinalities*:
  card degenerate-lotteries = *card outcomes*
  using *card-degen-lotteries-equals-outcomes degenerate-lotteries-def* **by** *auto*

lemma *degenerate-lots-subset-all*: *degenerate-lotteries* $\subseteq \mathcal{P}$
  by (*simp add*: *degenerate-lotteries-def*)

lemma *alt-definition-of-degenerate-lotteries*[*iff*]:
  {*return-pmf x* |*x*. *x*∈ *outcomes*} = *degenerate-lotteries*
proof (*standard*, *goal-cases*)
  case *1*
  have ∀ *x* ∈ {*return-pmf x* |*x*. *x* ∈ *outcomes*}. *x* ∈ *degenerate-lotteries*
  proof
    fix *x*
    assume *a*: *x* ∈ {*return-pmf x* |*x*. *x* ∈ *outcomes*}
    then have *card* (*set-pmf x*) = *1*
      by *auto*
    moreover have *set-pmf x* ⊆ *outcomes*
      using *a set-pmf-subset-singleton* **by** *auto*
    moreover have *x* ∈ $\mathcal{P}$
      by (*simp add*: *lotteries-on-def calculation*)
    ultimately show *x* ∈ *degenerate-lotteries*
      by (*simp add*: *degenerate-lotteries-def*)
  qed
  then show *?case* **by** *blast*
next
  case *2*
  have ∀ *x* ∈ *degenerate-lotteries*. *x* ∈ {*return-pmf x* |*x*. *x* ∈ *outcomes*}
  proof
    fix *x*
    assume *a*: *x* ∈ *degenerate-lotteries*
    hence *card* (*set-pmf x*) = *1*
      using *degenerate-lotteries-def* **by** *blast*
    moreover have *set-pmf x* ⊆ *outcomes*
      by (*meson a degenerate-lots-subset-all subset-iff support-in-outcomes*)
    moreover obtain *e* **where** {*e*} = *set-pmf x*
      using *calculation*
      by (*metis card-1-singletonE*)
    moreover have *e* ∈ *outcomes*
      using *calculation*(*2*) *calculation*(*3*) **by** *blast*
    moreover have *x* = *return-pmf e*
      using *calculation*(*3*) *set-pmf-subset-singleton* **by** *fast*
    ultimately show *x* ∈ {*return-pmf x* |*x*. *x* ∈ *outcomes*}
      by *blast*
  qed
  then show *?case* **by** *blast*
qed

lemma *best-indifferent*:
  ∀ *x* ∈ *best*. ∀ *y* ∈ *best*. *x* ≈[$\mathcal{R}$] *y*
  by (*simp add*: *best-def*)

lemma *worst-indifferent*:
  ∀ *x* ∈ *worst*. ∀ *y* ∈ *worst*. *x* ≈[$\mathcal{R}$] *y*
  by (*simp add*: *worst-def*)

lemma *best-worst-indiff-all-indiff*:
  assumes $b \in$ *best*
    and $w \in$ *worst*
    and $b \approx[\mathcal{R}]\ w$
  shows $\forall\, e \in \mathcal{P}.\ e \approx[\mathcal{R}]\ w\ \forall\, e \in \mathcal{P}.\ e \approx[\mathcal{R}]\ b$
proof −
  show $\forall\, e \in \mathcal{P}.\ e \approx[\mathcal{R}]\ w$
  proof (*standard*)
    fix e
    assume a: $e \in \mathcal{P}$
    then have $b \succeq[\mathcal{R}]\ e$
      using *a best-def assms* **by** *blast*
    moreover have $e \succeq[\mathcal{R}]\ w$
      using *a assms worst-def* **by** *auto*
    moreover have $b \succeq[\mathcal{R}]\ e$
      by (*simp add*: *calculation*(*1*))
    moreover show $e \approx[\mathcal{R}]\ w$
    proof (*rule ccontr*)
      assume $\neg\ e \approx[\mathcal{R}]\ w$
      then consider $e \succ[\mathcal{R}]\ w \mid w \succ[\mathcal{R}]\ e$
        by (*simp add*: *calculation*(*2*))
      then show *False*
      proof (*cases*)
        case *2*
        then show *?thesis*
          using *calculation*(*2*) **by** *blast*
      qed (*meson assms*(*3*) *calculation*(*1*)
          rational-preference.strict-is-neg-transitive relation-in-carrier rpr)
    qed
  qed
  then show $\forall\, e{\in}local.\mathcal{P}.\ e \approx[\mathcal{R}]\ b$
    using *assms* **by** (*meson rational-preference.compl*
        rational-preference.strict-is-neg-transitive relation-in-carrier rpr)
qed

Like Step 1 most general but with IFF.

lemma *mix-pmf-pref-iff-more-likely* [*iff*]:
  assumes $b \succ[\mathcal{R}]\ w$
  assumes $\alpha \in \{0..1\}$
    and $\beta \in \{0..1\}$
  shows $\alpha > \beta \longleftrightarrow$ *mix-pmf* α b $w \succ[\mathcal{R}]$ *mix-pmf* β b w (**is** *?L* $\longleftrightarrow$ *?R*)
  using *assms step-1-most-general*[*of* b w α β]
  by (*metis linorder-neqE-linordered-idom step-1-most-general*)

lemma *better-worse-good-mix-preferred*[*iff*]:
  assumes $b \succeq[\mathcal{R}]\ w$
  assumes $\alpha \in \{0..1\}$
    and $\beta \in \{0..1\}$
  assumes $\alpha \geq \beta$

shows *mix-pmf* α *b w* $\succeq$[$\mathcal{R}$] *mix-pmf* β *b w*
proof−
have (*0*::*real*) $\leq$ *1*
by *simp*
then show *?thesis*
by (*metis* (*no-types*) *assms assms*(*1*) *assms*(*2*) *assms*(*3*) *atLeastAtMost-iff less-eq-real-def mix-of-not-preferred-is-not-preferred mix-of-preferred-is-preferred mix-pmf-preferred-independence pmf-mix-0 relation-in-carrier step-1-most-general*)
qed

5.1 Add finiteness and non emptyness of outcomes

context
assumes *fnt*: *finite outcomes*
assumes *nempty*: *outcomes* $\neq$ {}
begin

lemma *finite-degenerate-lotteries*:
finite degenerate-lotteries
using *degen-outcome-cardinalities fnt nempty* **by** *fastforce*

lemma *degenerate-has-max-preferred*:
{$x \in$ *degenerate-lotteries*. ($\forall\, y \in$ *degenerate-lotteries*. $x \succeq$[$\mathcal{R}$] y)} $\neq$ {} (**is** *?l* $\neq$ {})
proof
assume *a*: *?l* = {}
let *?DG* = *degenerate-lotteries*
obtain *R* **where**
R: *rational-preference ?DG R R* $\subseteq \mathcal{R}$
using *degenerate-lots-subset-all rational-preference.all-carrier-ex-sub-rel rpr* **by** *blast*
then have $\exists\, e \in$ *?DG*. $\forall\, e' \in$ *?DG*. $e \succeq$[$\mathcal{R}$] e'
by (*metis R*(*1*) *R*(*2*) *card-0-eq degen-outcome-cardinalities finite-degenerate-lotteries fnt nempty subset-eq rational-preference.finite-nonempty-carrier-has-maximum*)
then show *False*
using *a* **by** *auto*
qed

lemma *degenerate-has-min-preferred*:
{$x \in$ *degenerate-lotteries*. ($\forall\, y \in$ *degenerate-lotteries*. $y \succeq$[$\mathcal{R}$] x)} $\neq$ {} (**is** *?l* $\neq$ {})
proof
assume *a*: *?l* = {}
let *?DG* = *degenerate-lotteries*
obtain *R* **where**
R: *rational-preference ?DG R R* $\subseteq \mathcal{R}$
using *degenerate-lots-subset-all rational-preference.all-carrier-ex-sub-rel rpr* **by**

blast
  have $\exists\, e \in$ *?DG*. $\forall\, e' \in$ *?DG*. $e' \succeq[\mathcal{R}]\ e$
    by (*metis R*(*1*) *R*(*2*) *card-0-eq degen-outcome-cardinalities*
      finite-degenerate-lotteries fnt nempty subset-eq
      rational-preference.finite-nonempty-carrier-has-minimum)
  then show *False*
    using *a* **by** *auto*
qed

lemma *exists-best-degenerate*:
  $\exists\, x \in$ *degenerate-lotteries*. $\forall\, y \in$ *degenerate-lotteries*. $x \succeq[\mathcal{R}]\ y$
  using *degenerate-has-max-preferred* **by** *blast*

lemma *exists-worst-degenerate*:
  $\exists\, x \in$ *degenerate-lotteries*. $\forall\, y \in$ *degenerate-lotteries*. $y \succeq[\mathcal{R}]\ x$
  using *degenerate-has-min-preferred* **by** *blast*

lemma *best-degenerate-in-best-overall*:
  $\exists\, x \in$ *degenerate-lotteries*. $\forall\, y \in \mathcal{P}$. $x \succeq[\mathcal{R}]\ y$
proof −
  obtain *b* **where**
    b: $b \in$ *degenerate-lotteries* $\forall\, y \in$ *degenerate-lotteries*. $b \succeq[\mathcal{R}]\ y$
    using *exists-best-degenerate* **by** *blast*
  have *asm*: *finite outcomes set-pmf b* $\subseteq$ *outcomes*
   by (*simp add*: *fnt*) (*meson b*(*1*) *degenerate-lots-subset-all subset-iff support-in-outcomes*)
  obtain *B* **where** *B*: *set-pmf b* $= \{B\}$
    using *b card-1-singletonE degenerate-lotteries-def* **by** *blast*
  have *deg*: $\forall\, d{\in}$*outcomes*. $b \succeq[\mathcal{R}]$ *return-pmf d*
    using *alt-definition-of-degenerate-lotteries b*(*2*) **by** *blast*
  define *P* **where**
    $P = (\lambda p.\ p \in \mathcal{P} \longrightarrow$ *return-pmf B* $\succeq[\mathcal{R}]\ p)$
  have *P p* **for** *p*
  proof −
    consider *set-pmf p* $\subseteq$ *outcomes* | $\neg$*set-pmf p* $\subseteq$ *outcomes*
      by *blast*
    then show *?thesis*
    proof (*cases*)
      case *1*
      have *finite outcomes set-pmf p* $\subseteq$ *outcomes*
        by (*auto simp*: *1 asm*)
      then show *?thesis*
      proof (*induct rule*: *pmf-mix-induct'*)
        case (*degenerate x*)
        then show *?case*
          using *B P-def deg set-pmf-subset-singleton* **by** *fastforce*
      qed (*simp add*: *P-def lotteries-on-def mix-of-not-preferred-is-not-preferred*
            mix-of-not-preferred-is-not-preferred[*of b p q a*])
    qed (*simp add*: *lotteries-on-def P-def*)
  qed

moreover have $\forall\, e \in \mathcal{P}.\ b \succeq[\mathcal{R}]\ e$
  using *calculation B P-def set-pmf-subset-singleton* **by** *fastforce*
ultimately show *?thesis*
  using *b degenerate-lots-subset-all* **by** *blast*
qed

lemma *worst-degenerate-in-worst-overall*:
$\exists\, x \in$ *degenerate-lotteries*. $\forall\, y \in \mathcal{P}.\ y \succeq[\mathcal{R}]\ x$
proof −
  obtain *b* **where**
    b: $b \in$ *degenerate-lotteries* $\forall\, y \in$ *degenerate-lotteries*. $y \succeq[\mathcal{R}]\ b$
    using *exists-worst-degenerate* **by** *blast*
  have *asm*: *finite outcomes set-pmf b* $\subseteq$ *outcomes*
   by (*simp add*: *fnt*) (*meson b*(*1*) *degenerate-lots-subset-all subset-iff support-in-outcomes*)
  obtain *B* **where** *B*: *set-pmf b* = $\{B\}$
    using *b card-1-singletonE degenerate-lotteries-def* **by** *blast*
  have *deg*: $\forall\, d \in$ *outcomes*. *return-pmf d* $\succeq[\mathcal{R}]\ b$
    using *alt-definition-of-degenerate-lotteries b*(*2*) **by** *blast*
  define *P* **where**
    $P = (\lambda p.\ p \in \mathcal{P} \longrightarrow p \succeq[\mathcal{R}]$ *return-pmf B*$)$
  have *P p* **for** *p*
  proof −
    consider *set-pmf p* $\subseteq$ *outcomes* | $\neg$*set-pmf p* $\subseteq$ *outcomes*
      by *blast*
    then show *?thesis*
    proof (*cases*)
      case *1*
      have *finite outcomes set-pmf p* $\subseteq$ *outcomes*
        by (*auto simp*: *1 asm*)
      then show *?thesis*
      proof (*induct rule*: *pmf-mix-induct'*)
        case (*degenerate x*)
        then show *?case*
          using *B P-def deg set-pmf-subset-singleton* **by** *fastforce*
      next
      qed (*simp add*: *P-def lotteries-on-def mix-of-preferred-is-preferred*
          mix-of-not-preferred-is-not-preferred[*of b p*])
    qed (*simp add*: *lotteries-on-def P-def*)
  qed
  moreover have $\forall\, e \in \mathcal{P}.\ e \succeq[\mathcal{R}]\ b$
    using *calculation B P-def set-pmf-subset-singleton* **by** *fastforce*
  ultimately show *?thesis*
    using *b degenerate-lots-subset-all* **by** *blast*
qed

lemma *overall-best-nonempty*:
  best $\neq \{\}$
  using *best-def best-degenerate-in-best-overall degenerate-lots-subset-all* **by** *blast*

lemma *overall-worst-nonempty*:
  worst ≠ {}
  using *degenerate-lots-subset-all worst-def worst-degenerate-in-worst-overall* **by** *auto*

lemma *trans-approx*:
  assumes *x*≈[𝓡] *y*
    and *y* ≈[𝓡] *z*
  shows *x* ≈[𝓡] *z*
  using *preference.indiff-trans*[*of* 𝓟 𝓡 *x y z*] *assms rpr rational-preference-def* **by** *blast*

First EXPLICIT use of the axiom of choice

private definition *some-best* **where**
  some-best = (*SOME x. x* ∈ *degenerate-lotteries* ∧ *x* ∈ *best*)

private definition *some-worst* **where**
  some-worst = (*SOME x. x* ∈ *degenerate-lotteries* ∧ *x* ∈ *worst*)

private definition *my-U* :: *'a pmf* ⇒ *real*
  where
    my-U p = (*SOME α. α*∈{*0..1*} ∧ *p* ≈[𝓡] *mix-pmf α some-best some-worst*)

lemma *exists-best-and-degenerate*: *degenerate-lotteries* ∩ *best* ≠ {}
  using *best-def best-degenerate-in-best-overall degenerate-lots-subset-all* **by** *blast*

lemma *exists-worst-and-degenerate*: *degenerate-lotteries* ∩ *worst* ≠ {}
  using *worst-def worst-degenerate-in-worst-overall degenerate-lots-subset-all* **by** *blast*

lemma *some-best-in-best*: *some-best* ∈ *best*
  using *exists-best-and-degenerate some-best-def*
  by (*metis* (*mono-tags*, *lifting*) *Int-emptyI some-eq-ex*)

lemma *some-worst-in-worst*: *some-worst* ∈ *worst*
  using *exists-worst-and-degenerate some-worst-def*
  by (*metis* (*mono-tags*, *lifting*) *Int-emptyI some-eq-ex*)

lemma *best-always-at-least-as-good-mix*:
  assumes *α* ∈ {*0..1*}
    and *p* ∈ 𝓟
  shows *mix-pmf α some-best p* ≿[𝓡] *p*
  using *assms*(*1*) *assms*(*2*) *best-def mix-of-preferred-is-preferred*
    rational-preference.compl rpr some-best-in-best **by** *fastforce*

lemma *geq-mix-imp-weak-pref*:

assumes $\alpha \in \{0..1\}$
and $\beta \in \{0..1\}$
assumes $\alpha \geq \beta$
shows *mix-pmf* α *some-best some-worst* $\succeq[\mathcal{R}]$ *mix-pmf* β *some-best some-worst*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *best-def some-best-in-best some-worst-in-worst worst-def* **by** *auto*

lemma *gamma-inverse*:
assumes $\alpha \in \{0<..<1\}$
and $\beta \in \{0<..<1\}$
shows $(1::real) - (\alpha - \beta) / (1 - \beta) = (1 - \alpha) / (1 - \beta)$
proof −
have $1 - (\alpha - \beta) / (1 - \beta) = (1 - \beta)/(1 - \beta) - (\alpha - \beta) / (1 - \beta)$
using *assms*(*2*) **by** *auto*
also have ... $= (1 - \beta - (\alpha - \beta)) / (1 - \beta)$
by (*metis diff-divide-distrib*)
also have ... $= (1 - \alpha) / (1 - \beta)$
by *simp*
finally show *?thesis* **.**
qed

lemma *all-mix-pmf-indiff-indiff-best-worst*:
assumes $l \in \mathcal{P}$
assumes $b \in$ *best*
assumes $w \in$ *worst*
assumes $b \approx[\mathcal{R}]\ w$
shows $\forall \alpha \in \{0..1\}.\ l \approx[\mathcal{R}]$ *mix-pmf* $\alpha\ b\ w$
by (*meson assms best-worst-indiff-all-indiff*(*1*) *mix-of-preferred-is-preferred best-worst-indiff-all-indiff*(*2*) *mix-of-not-preferred-is-not-preferred*)

lemma *indiff-imp-same-utility-value*:
assumes *some-best* $\succ[\mathcal{R}]$ *some-worst*
assumes $\alpha \in \{0..1\}$
assumes $\beta \in \{0..1\}$
assumes *mix-pmf* β *some-best some-worst* $\approx[\mathcal{R}]$ *mix-pmf* α *some-best some-worst*
shows $\beta = \alpha$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *linorder-neqE-linordered-idom* **by** *blast*

lemma *leq-mix-imp-weak-inferior*:
assumes *some-best* $\succ[\mathcal{R}]$ *some-worst*
assumes $\alpha \in \{0..1\}$
and $\beta \in \{0..1\}$
assumes *mix-pmf* β *some-best some-worst* $\succeq[\mathcal{R}]$ *mix-pmf* α *some-best some-worst*
shows $\beta \geq \alpha$
proof −
have $*$: *mix-pmf* β *some-best some-worst* $\approx[\mathcal{R}]$ *mix-pmf* α *some-best some-worst* $\Longrightarrow \alpha \leq \beta$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *indiff-imp-same-utility-value* **by** *blast*

```
 consider mix-pmf β some-best some-worst ≻[ℛ] mix-pmf α some-best some-worst
|
    mix-pmf β some-best some-worst ≈[ℛ] mix-pmf α some-best some-worst
    using assms(4) by blast
  then show ?thesis
    by(cases) (meson assms(2) assms(3) geq-mix-imp-weak-pref le-cases ∗)+
qed

lemma ge-mix-pmf-preferred:
  assumes x ≻[ℛ] y
  assumes α ∈ {0..1}
    and β ∈ {0..1}
  assumes α ≥ β
  shows (mix-pmf α x y) ≽[ℛ] (mix-pmf β x y)
  using assms(1) assms(2) assms(3) assms(4) by blast
```

5.2 Add continuity to assumptions

```
context
  assumes cnt: continuous-vnm (lotteries-on outcomes) ℛ
begin
```

In Literature this is referred to as step 2.

```
lemma step-2-unique-continuous-unfolding:
  assumes p ≽[ℛ] q
    and q ≽[ℛ] r
    and p ≻[ℛ] r
  shows ∃!α ∈ {0..1}. q ≈[ℛ] mix-pmf α p r
proof (rule ccontr)
  assume neg-a: ∄!α. α ∈ {0..1} ∧ q ≈[ℛ] mix-pmf α p r
  have ∃α ∈ {0..1}. q ≈[ℛ] mix-pmf α p r
    using non-unique-continuous-unfolding[of outcomes ℛ p q r]
      assms cnt rpr by blast
  then obtain α β :: real where
    a-b: α∈{0..1} β ∈{0..1} q ≈[ℛ] mix-pmf α p r q ≈[ℛ] mix-pmf β p r α ≠ β
    using neg-a by blast
  consider α > β | β > α
    using a-b by linarith
  then show False
  proof (cases)
    case 1
    with step-1-most-general[of p r α β] assms
    have mix-pmf α p r ≻[ℛ] mix-pmf β p r
      using a-b(1) a-b(2) by blast
    then show ?thesis using a-b
      by (meson rational-preference.strict-is-neg-transitive relation-in-carrier rpr)
  next
    case 2
    with step-1-most-general[of p r β α] assms have mix-pmf β p r ≻[ℛ]mix-pmf
```

α p r
      using *a-b*(*1*) *a-b*(*2*) **by** *blast*
    then show *?thesis* **using** *a-b*
      by (*meson rational-preference.strict-is-neg-transitive relation-in-carrier rpr*)
  qed
qed

These folowing two lemmas are referred to sometimes called step 2.

lemma *create-unique-indiff-using-distinct-best-worst*:
  assumes $l \in \mathcal{P}$
  assumes $b \in$ *best*
  assumes $w \in$ *worst*
  assumes $b \succ[\mathcal{R}]\ w$
  shows $\exists!\alpha \in \{0..1\}.\ l \approx[\mathcal{R}]$ *mix-pmf* $\alpha\ b\ w$
proof −
  have $b \succeq[\mathcal{R}]\ l$
    using *best-def*
    using *assms* **by** *blast*
  moreover have $l \succeq[\mathcal{R}]\ w$
    using *worst-def assms* **by** *blast*
  ultimately show $\exists!\alpha\in\{0..1\}.\ l \approx[\mathcal{R}]$ *mix-pmf* $\alpha\ b\ w$
    using *step-2-unique-continuous-unfolding*[*of b l w*] *assms* **by** *linarith*
qed

lemma *exists-element-bw-mix-is-approx*:
  assumes $l \in \mathcal{P}$
  assumes $b \in$ *best*
  assumes $w \in$ *worst*
  shows $\exists\alpha \in\{0..1\}.\ l \approx[\mathcal{R}]$ *mix-pmf* $\alpha\ b\ w$
proof −
  consider $b \succ[\mathcal{R}]\ w \mid b \approx[\mathcal{R}]\ w$
    using *assms*(*2*) *assms*(*3*) *best-def worst-def* **by** *auto*
  then show *?thesis*
  proof (*cases*)
    case *1*
    then show *?thesis*
      using *create-unique-indiff-using-distinct-best-worst assms* **by** *blast*
  qed (*auto simp*: *all-mix-pmf-indiff-indiff-best-worst assms*)
qed

lemma *my-U-is-defined*:
  assumes $p \in \mathcal{P}$
  shows *my-U p* $\in \{0..1\}$ $p \approx[\mathcal{R}]$ *mix-pmf* (*my-U p*) *some-best some-worst*
proof −
  have *some-best* $\in$ *best*
    by (*simp add*: *some-best-in-best*)
  moreover have *some-worst* $\in$ *worst*
    by (*simp add*: *some-worst-in-worst*)
  with *exists-element-bw-mix-is-approx*[*of p some-best some-worst*] *calculation assms*

have *e*: $\exists\alpha\in\{0..1\}$. $p \approx[\mathcal{R}]$ *mix-pmf* α *some-best some-worst* **by** *blast*
then show *my-U p* $\in \{0..1\}$
by (*metis* (*mono-tags*, *lifting*) *my-U-def someI-ex*)
show $p \approx[\mathcal{R}]$ *mix-pmf* (*my-U p*) *some-best some-worst*
by (*metis* (*mono-tags*, *lifting*) *e my-U-def someI-ex*)
qed

lemma *weak-pref-mix-with-my-U-weak-pref*:
assumes $p \succeq[\mathcal{R}]$ q
shows *mix-pmf* (*my-U p*) *some-best some-worst* $\succeq[\mathcal{R}]$ *mix-pmf* (*my-U q*) *some-best some-worst*
by (*meson assms my-U-is-defined*(*2*) *relation-in-carrier rpr rational-preference.weak-is-transitive*)

lemma *preferred-greater-my-U*:
assumes $p \in \mathcal{P}$
and $q \in \mathcal{P}$
assumes *mix-pmf* (*my-U p*) *some-best some-worst* $\succ[\mathcal{R}]$ *mix-pmf* (*my-U q*) *some-best some-worst*
shows *my-U p* $>$ *my-U q*
proof (*rule ccontr*)
assume $\neg$ *my-U p* $>$ *my-U q*
then consider *my-U p* $=$ *my-U q* | *my-U p* $<$ *my-U q*
by *linarith*
then show *False*
proof (*cases*)
case *1*
then have *mix-pmf* (*my-U p*) *some-best some-worst* $\approx[\mathcal{R}]$ *mix-pmf* (*my-U q*) *some-best some-worst*
using *assms* **by** *auto*
then show *?thesis* **using** *assms* **by** *blast*
next
case *2*
moreover have *my-U q* $\in \{0..1\}$
using *assms*(*2*) *my-U-is-defined*(*1*) **by** *blast*
moreover have *my-U p* $\in \{0..1\}$
using *assms*(*1*) *my-U-is-defined*(*1*) **by** *blast*
moreover have *mix-pmf* (*my-U q*) *some-best some-worst* $\succeq[\mathcal{R}]$ *mix-pmf* (*my-U p*) *some-best some-worst*
using *calculation geq-mix-imp-weak-pref* **by** *auto*
then show *?thesis* **using** *assms* **by** *blast*
qed
qed

lemma *geq-my-U-imp-weak-preference*:
assumes $p \in \mathcal{P}$
and $q \in \mathcal{P}$
assumes *some-best* $\succ[\mathcal{R}]$ *some-worst*
assumes *my-U p* $\geq$ *my-U q*

shows *p* $\succeq$[$\mathcal{R}$] *q*
proof –
have *p-q*: *my-U p* $\in$ {*0..1*} *my-U q* $\in$ {*0..1*}
using *assms my-U-is-defined*(*1*) **by** *blast+*
with *ge-mix-pmf-preferred*[*of some-best some-worst my-U p my-U q*]
p-q assms(*1*) *assms*(*3*) *assms*(*4*)
have *mix-pmf* (*my-U p*) *some-best some-worst* $\succeq$[$\mathcal{R}$] *mix-pmf* (*my-U q*) *some-best some-worst* **by** *blast*
consider *my-U p* = *my-U q* | *my-U p* > *my-U q*
using *assms* **by** *linarith*
then show *?thesis*
proof (*cases*)
case *2*
then show *?thesis*
by (*meson assms*(*1*) *assms*(*2*) *assms*(*3*) *p-q*(*1*) *p-q*(*2*) *rational-preference.compl*

rpr step-1-most-general weak-pref-mix-with-my-U-weak-pref)
qed (*metis assms*(*1*) *assms*(*2*) *my-U-is-defined*(*2*) *trans-approx*)
qed

lemma *my-U-represents-pref*:
assumes *some-best* $\succ$[$\mathcal{R}$] *some-worst*
assumes *p* $\in \mathcal{P}$
and *q* $\in \mathcal{P}$
shows *p* $\succeq$[$\mathcal{R}$] *q* $\longleftrightarrow$ *my-U p* $\geq$ *my-U q* (**is** *?L* $\longleftrightarrow$ *?R*)
proof –
have *p-def*: *my-U p*$\in$ {*0..1*} *my-U q* $\in$ {*0..1*}
using *assms my-U-is-defined* **by** *blast+*
show *?thesis*
proof
assume *a*: *?L*
hence *mix-pmf* (*my-U p*) *some-best some-worst* $\succeq$[$\mathcal{R}$] *mix-pmf* (*my-U q*) *some-best some-worst*
using *weak-pref-mix-with-my-U-weak-pref* **by** *auto*
then show *?R* **using** *leq-mix-imp-weak-inferior*[*of my-U p my-U q*] *p-def a*
assms(*1*) *leq-mix-imp-weak-inferior* **by** *blast*
next
assume *?R*
then show *?L* **using** *geq-my-U-imp-weak-preference*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *blast*
qed
qed

lemma *first-iff-u-greater-strict-preff*:
assumes *p* $\in \mathcal{P}$
and *q* $\in \mathcal{P}$
assumes *some-best* $\succ$[$\mathcal{R}$] *some-worst*
shows *my-U p* > *my-U q* $\longleftrightarrow$ *mix-pmf* (*my-U p*) *some-best some-worst* $\succ$[$\mathcal{R}$] *mix-pmf* (*my-U q*) *some-best some-worst*

proof
  assume *a*: *my-U p* > *my-U q*
  have *my-U p* ∈ {*0..1*} *my-U q* ∈ {*0..1*}
    using *assms my-U-is-defined*(*1*) **by** *blast+*
  then show *mix-pmf* (*my-U p*) *some-best some-worst* ≻[ℛ] *mix-pmf* (*my-U q*) *some-best some-worst*
    using *a assms*(*3*) **by** *blast*
next
  assume *a*: *mix-pmf* (*my-U p*) *some-best some-worst* ≻[ℛ] *mix-pmf* (*my-U q*) *some-best some-worst*
  have *my-U p* ∈ {*0..1*} *my-U q* ∈ {*0..1*}
    using *assms my-U-is-defined*(*1*) **by** *blast+*
  then show *my-U p* > *my-U q*
    using *preferred-greater-my-U*[*of p q*] *assms a* **by** *blast*
qed

lemma *second-iff-calib-mix-pref-strict-pref*:
  assumes *p* ∈ 𝒫
    and *q* ∈ 𝒫
  assumes *some-best* ≻[ℛ] *some-worst*
  shows *mix-pmf* (*my-U p*) *some-best some-worst* ≻[ℛ] *mix-pmf* (*my-U q*) *some-best some-worst* ⟷ *p* ≻[ℛ] *q*
proof
  assume *a*: *mix-pmf* (*my-U p*) *some-best some-worst* ≻[ℛ] *mix-pmf* (*my-U q*) *some-best some-worst*
  have *my-U p* ∈ {*0..1*} *my-U q* ∈ {*0..1*}
    using *assms my-U-is-defined*(*1*) **by** *blast+*
  then show *p* ≻[ℛ] *q*
    using *a assms*(*3*) *assms*(*1*) *assms*(*2*) *geq-my-U-imp-weak-preference*
      leq-mix-imp-weak-inferior weak-pref-mix-with-my-U-weak-pref **by** *blast*
next
  assume *a*: *p* ≻[ℛ] *q*
  have *my-U p* ∈ {*0..1*} *my-U q* ∈ {*0..1*}
    using *assms my-U-is-defined*(*1*) **by** *blast+*
  then show *mix-pmf* (*my-U p*) *some-best some-worst* ≻[ℛ] *mix-pmf* (*my-U q*) *some-best some-worst*
    using *a assms*(*1*) *assms*(*2*) *assms*(*3*) *leq-mix-imp-weak-inferior my-U-represents-pref* **by** *blast*
qed

lemma *my-U-is-linear-function*:
  assumes *p* ∈ 𝒫
    and *q* ∈ 𝒫
    and α ∈ {*0..1*}
  assumes *some-best* ≻[ℛ] *some-worst*
  shows *my-U* (*mix-pmf* α *p q*) = α ∗ *my-U p* + (*1* − α) ∗ *my-U q*
proof −
  define *B* **where** *B*: *B* = *some-best*
  define *W* **where** *W*: *W* = *some-worst*

define *Up* **where** *Up*: $Up = my\text{-}U\ p$
define *Uq* **where** *Uq*: $Uq = my\text{-}U\ q$
have *long-in*: $(\alpha * Up + (1 - \alpha) * Uq) \in \{0..1\}$
proof −
have $Up \in \{0..1\}$
using *assms Up my-U-is-defined*(*1*) **by** *blast*
moreover have $Uq \in \{0..1\}$
using *assms Uq my-U-is-defined*(*1*) **by** *blast*
moreover have $\alpha * Up \in \{0..1\}$
using ‹$Up \in \{0..1\}$› *assms*(*3*) *mult-le-one* **by** *auto*
moreover have $1-\alpha \in \{0..1\}$
using *assms*(*3*) **by** *auto*
moreover have $(1 - \alpha) * Uq \in \{0..1\}$
using *mult-le-one*[*of* $1-\alpha$ *Uq*] *calculation*(*2*) *calculation*(*4*) **by** *auto*
ultimately show *?thesis*
using *add-nonneg-nonneg*[*of* $\alpha * Up\ (1 - \alpha) * Uq$]
convex-bound-le[*of* *Up 1 Uq* α $1-\alpha$] **by** *simp*
qed
have *fst*: $p \approx[\mathcal{R}]$ (*mix-pmf Up B W*)
using *assms my-U-is-defined*[*of p*] *B W Up* **by** *simp*
have *snd*: $q \approx[\mathcal{R}]$ (*mix-pmf Uq B W*)
using *assms my-U-is-defined*[*of q*] *B W Uq* **by** *simp*
have *mp-in*: (*mix-pmf Up B W*) $\in \mathcal{P}$
using *fst relation-in-carrier* **by** *blast*
have *f2*: *mix-pmf* α *p q* $\approx[\mathcal{R}]$ *mix-pmf* α (*mix-pmf Up B W*) *q*
using *fst assms*(*2*) *assms*(*3*) *mix-pmf-preferred-independence* **by** *blast*
have $**$: *mix-pmf* α (*mix-pmf Up B W*) (*mix-pmf Uq B W*) =
mix-pmf $(\alpha * Up + (1-\alpha) * Uq)$ *B W* (**is** *?L* = *?R*)
proof −
let *?mixPQ* = (*mix-pmf* $(\alpha * Up + (1 - \alpha) * Uq)$ *B W*)
have ∀ *e*∈*set-pmf ?L*. *pmf* (*?L*) *e* = *pmf ?mixPQ e*
proof
fix *e*
assume *asm*: *e* ∈ *set-pmf ?L*
have *i1*: *pmf* (*?L*) *e* = α ∗ *pmf* (*mix-pmf Up B W*) *e* +
pmf (*mix-pmf Uq B W*) *e* − α ∗ *pmf* (*mix-pmf Uq B W*) *e*
using *pmf-mix-deeper*[*of* α *mix-pmf Up B W* (*mix-pmf Uq B W*) *e*] *assms*(*3*) **by** *blast*
have *i3*: ... = α ∗ *Up* ∗ *pmf B e* + α ∗ *pmf W e* − α ∗ *Up* ∗ *pmf W e* + *Uq* ∗ *pmf B e* +
pmf W e − *Uq* ∗ *pmf W e* − α ∗ *Uq* ∗ *pmf B e* − α ∗ *pmf W e* + α ∗ *Uq* ∗ *pmf W e*
using *left-diff-distrib′ pmf-mix-deeper*[*of Up B W e*] *pmf-mix-deeper*[*of Uq B W e*]
assms Up Uq my-U-is-defined(*1*) **by** (*simp add*: *distrib-left right-diff-distrib*)
have *j4*: *pmf ?mixPQ e* = $(\alpha * Up + (1 - \alpha) * Uq)$ ∗ *pmf B e* +
pmf W e − $(\alpha * Up + (1 - \alpha) * Uq)$ ∗ *pmf W e*
using *pmf-mix-deeper*[*of* $(\alpha * Up + (1 - \alpha) * Uq)$ *B W e*] *long-in* **by** *blast*
then show *pmf* (*?L*) *e* = *pmf ?mixPQ e*

by (*simp add*: *i1 i3 mult.commute right-diff-distrib′ ring-class.ring-distribs*(*1*))
qed
then show *?thesis* **using** *pmf-equiv-intro1* **by** *blast*
qed
have *mix-pmf* α (*mix-pmf Up B W*) *q* $\approx$[$\mathcal{R}$] *?L*
using *approx-remains-after-same-comp-left assms*(*3*) *mp-in snd* **by** *blast*
hence $*$: *mix-pmf* α *p q* $\approx$[$\mathcal{R}$] *mix-pmf* α (*mix-pmf* (*my-U p*) *B W*) (*mix-pmf* (*my-U q*) *B W*)
using *Up Uq f2 trans-approx* **by** *blast*
have *mix-pmf* α (*mix-pmf* (*my-U p*) *B W*) (*mix-pmf* (*my-U q*) *B W*) = *?R*
using *Up Uq* $**$ **by** *blast*
hence *my-U* (*mix-pmf* α *p q*) = α $*$ *Up* + ($1-\alpha$) $*$ *Uq*
by (*metis* $*$ *B W assms*(*4*) *indiff-imp-same-utility-value long-in my-U-is-defined*(*1*) *my-U-is-defined*(*2*) *my-U-represents-pref relation-in-carrier*)
then show *?thesis*
using *Up Uq* **by** *blast*
qed

Now we define a more general Utility function that also takes the degenerate case into account

private definition *general-U*
where
general-U p = (*if some-best* $\approx$[$\mathcal{R}$] *some-worst then 1 else my-U p*)

lemma *general-U-is-linear-function*:
assumes $p \in \mathcal{P}$
and $q \in \mathcal{P}$
and $\alpha \in \{0..1\}$
shows *general-U* (*mix-pmf* α *p q*) = α $*$ (*general-U p*) + ($1 - \alpha$) $*$ (*general-U q*)
proof −
consider *some-best* $\succ$[$\mathcal{R}$] *some-worst* | *some-best* $\approx$[$\mathcal{R}$] *some-worst*
using *best-def some-best-in-best some-worst-in-worst worst-def* **by** *auto*
then show *?thesis*
proof (*cases*, *goal-cases*)
case *1*
then show *?case*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *general-U-def my-U-is-linear-function* **by** *auto*
next
case *2*
then show *?case*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *general-U-def* **by** *auto*
qed
qed

lemma *general-U-ordinal-Utility*:
shows *ordinal-utility* $\mathcal{P}$ $\mathcal{R}$ *general-U*
proof (*standard*, *goal-cases*)

case (*1 x y*)
consider (*a*) *some-best* $\succ$[$\mathcal{R}$] *some-worst* | (*b*) *some-best* $\approx$[$\mathcal{R}$] *some-worst*
using *best-def some-best-in-best some-worst-in-worst worst-def* **by** *auto*
then show *?case*
proof (*cases*, *goal-cases*)
case *a*
have *some-best* $\succ$[$\mathcal{R}$] *some-worst*
using *a* **by** *auto*
then show $x \succeq$[$\mathcal{R}$] y = (*general-U y* $\leq$ *general-U x*)
using *1 my-U-represents-pref*[*of x y*] *general-U-def* **by** *simp*
next
case *b*
have *general-U x* = *1 general-U y* = *1*
by (*simp add*: *b general-U-def*)+
moreover have $x \approx$[$\mathcal{R}$] y **using** *b*
by (*meson 1*(*1*) *1*(*2*) *best-worst-indiff-all-indiff*(*1*)
some-best-in-best some-worst-in-worst trans-approx)
ultimately show $x \succeq$[$\mathcal{R}$] y = (*general-U y* $\leq$ *general-U x*)
using *general-U-def* **by** *linarith*
qed
next
case (*2 x y*)
then show *?case*
using *relation-in-carrier* **by** *blast*
next
case (*3 x y*)
then show *?case*
using *relation-in-carrier* **by** *blast*
qed

Proof of the linearity of general-U. If we consider the definition of expected utility functions from Maschler, Solan, Zamir we are done.

theorem *is-linear*:
assumes $p \in \mathcal{P}$
and $q \in \mathcal{P}$
and $\alpha \in \{0..1\}$
shows $\exists$ *u*. *u* (*mix-pmf* α *p q*) = α * (*u p*) + (*1*−α) * (*u q*)
proof
let *?u* = *general-U*
consider *some-best* $\succ$[$\mathcal{R}$] *some-worst* | *some-best* $\approx$[$\mathcal{R}$] *some-worst*
using *best-def some-best-in-best some-worst-in-worst worst-def* **by** *auto*
then show *?u* (*mix-pmf* α *p q*) = α * *?u p* + (*1* − α) * *?u q*
proof (*cases*)
case *1*
then show *?thesis*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *general-U-def my-U-is-linear-function* **by**
auto
next
case *2*

then show *?thesis*
by (*simp add*: *general-U-def*)
qed
qed

Now I define a Utility function that assigns a utility to all outcomes. These are only finitely many

private definition *ocU*
where
ocU p = general-U (return-pmf p)

lemma *geral-U-is-expected-value-of-ocU*:
assumes *set-pmf p* $\subseteq$ *outcomes*
shows *general-U p = measure-pmf.expectation p ocU*
using *fnt assms*
proof (*induct rule*: *pmf-mix-induct'*)
case (*mix p q a*)
hence *general-U (mix-pmf a p q) = a ∗ general-U p + (1−a) ∗ general-U q*
using *general-U-is-linear-function[of p q a] mix.hyps assms lotteries-on-def mix.hyps* **by** *auto*
also have ... = *a ∗ measure-pmf.expectation p ocU + (1−a) ∗ measure-pmf.expectation q ocU*
by (*simp add*: *mix.hyps(4) mix.hyps(5)*)
also have ... = *measure-pmf.expectation (mix-pmf a p q) ocU*
using *general-U-is-linear-function expected-value-mix-pmf-distrib fnt infinite-super mix.hyps(1)*
by (*metis fnt mix.hyps(2) mix.hyps(3)*)
finally show *?case* **.**
qed (*auto simp*: *support-in-outcomes assms fnt integral-measure-pmf-real ocU-def*)

lemma *ordinal-utility-expected-value*:
ordinal-utility $\mathcal{P}$ $\mathcal{R}$ (λx. *measure-pmf.expectation x ocU*)
proof (*standard*, *goal-cases*)
case (*1 x y*)
have *ocs*: *set-pmf x* $\subseteq$ *outcomes set-pmf y* $\subseteq$ *outcomes*
by (*meson 1 subsetI support-in-outcomes*)+
have $x \succeq[\mathcal{R}]\ y \Longrightarrow$ (*measure-pmf.expectation y ocU* $\le$ *measure-pmf.expectation x ocU*)
proof −
assume $x \succeq[\mathcal{R}]\ y$
have *general-U x* $\ge$ *general-U y*
by (*meson* ‹$x \succeq[\mathcal{R}]\ y$› *general-U-ordinal-Utility ordinal-utility-def*)
then show (*measure-pmf.expectation y ocU* $\le$ *measure-pmf.expectation x ocU*)
using *geral-U-is-expected-value-of-ocU ocs* **by** *auto*
qed
moreover have (*measure-pmf.expectation y ocU* $\le$ *measure-pmf.expectation x ocU*) $\Longrightarrow x \succeq[\mathcal{R}]\ y$
proof −
assume (*measure-pmf.expectation y ocU* $\le$ *measure-pmf.expectation x ocU*)

then have *general-U x* $\geq$ *general-U y*
by (*simp add*: *geral-U-is-expected-value-of-ocU ocs*(*1*) *ocs*(*2*))
then show $x \succeq[\mathcal{R}]$ *y*
by (*meson 1*(*1*) *1*(*2*) *general-U-ordinal-Utility ordinal-utility.util-def*)
qed
ultimately show *?case*
by *blast*
next
case (*2 x y*)
then show *?case*
using *relation-in-carrier* **by** *blast*
next
case (*3 x y*)
then show *?case*
using *relation-in-carrier* **by** *auto*
qed

lemma *ordinal-utility-expected-value*′:
$\exists$ *u*. *ordinal-utility* $\mathcal{P}$ $\mathcal{R}$ (λx. *measure-pmf.expectation x u*)
using *ordinal-utility-expected-value* **by** *blast*

lemma *ocU-is-expected-utility-bernoulli*:
shows $\forall x \in \mathcal{P}$. $\forall y \in \mathcal{P}$. $x \succeq[\mathcal{R}]$ $y \longleftrightarrow$
measure-pmf.expectation x ocU $\geq$ *measure-pmf.expectation y ocU*
using *ordinal-utility-expected-value* **by** (*meson ordinal-utility.util-def*)

end

end

end

lemma *expected-value-is-utility-function*:
assumes *fnt*: *finite outcomes* **and** *outcomes* $\neq \{\}$
assumes $x \in$ *lotteries-on outcomes* **and** $y \in$ *lotteries-on outcomes*
assumes *ordinal-utility* (*lotteries-on outcomes*) $\mathcal{R}$ (λx. *measure-pmf.expectation x u*)
shows *measure-pmf.expectation x u* $\geq$ *measure-pmf.expectation y u* $\longleftrightarrow x \succeq[\mathcal{R}]$ *y* (**is** *?L* $\longleftrightarrow$ *?R*)
using *assms*(*3*) *assms*(*4*) *assms*(*5*) *ordinal-utility.util-def-conf*
ordinal-utility.ordinal-utility-left iffI **by** (*metis* (*no-types*, *lifting*))

lemma *system-U-implies-vNM-utility*:
assumes *fnt*: *finite outcomes* **and** *outcomes* $\neq \{\}$
assumes *rpr*: *rational-preference* (*lotteries-on outcomes*) $\mathcal{R}$
assumes *ind*: *independent-vnm* (*lotteries-on outcomes*) $\mathcal{R}$

assumes *cnt*: *continuous-vnm* (*lotteries-on outcomes*) $\mathcal{R}$
shows $\exists$ *u*. *ordinal-utility* (*lotteries-on outcomes*) $\mathcal{R}$ (λx. *measure-pmf.expectation x u*)
using *ordinal-utility-expected-value′*[*of outcomes* $\mathcal{R}$] *assms* **by** *blast*

lemma *vNM-utility-implies-rationality*:
assumes *fnt*: *finite outcomes* **and** *outcomes* $\neq$ {}
assumes $\exists$ *u*. *ordinal-utility* (*lotteries-on outcomes*) $\mathcal{R}$ (λx. *measure-pmf.expectation x u*)
shows *rational-preference* (*lotteries-on outcomes*) $\mathcal{R}$
using *assms*(*3*) *ordinal-util-imp-rat-prefs* **by** *blast*

theorem *vNM-utility-implies-independence*:
assumes *fnt*: *finite outcomes* **and** *outcomes* $\neq$ {}
assumes $\exists$ *u*. *ordinal-utility* (*lotteries-on outcomes*) $\mathcal{R}$ (λx. *measure-pmf.expectation x u*)
shows *independent-vnm* (*lotteries-on outcomes*) $\mathcal{R}$
proof (*rule independent-vnmI2*)
fix *p q r*
and α::*real*
assume *a1*: $p \in \mathcal{P}$ *outcomes*
assume *a2*: $q \in \mathcal{P}$ *outcomes*
assume *a3*: $r \in \mathcal{P}$ *outcomes*
assume *a4*: $\alpha \in \{0<..1\}$
have *in-lots*: *mix-pmf* α *p r* $\in$ *lotteries-on outcomes mix-pmf* α *q r* $\in$ *lotteries-on outcomes*
using *a1 a3 a4 mix-in-lot* **apply** *fastforce*
using *a2 a3 a4 mix-in-lot* **by** *fastforce*
have *fnts*: *finite* (*set-pmf p*) *finite* (*set-pmf q*) *finite* (*set-pmf r*)
using *a1 a2 a3 fnt infinite-super lotteries-on-def* **by** *blast+*
obtain *u* **where**
u: *ordinal-utility* (*lotteries-on outcomes*) $\mathcal{R}$ (λx. *measure-pmf.expectation x u*)
using *assms* **by** *blast*
have $p \succeq[\mathcal{R}]$ $q \Longrightarrow$ *mix-pmf* α *p r* $\succeq[\mathcal{R}]$ *mix-pmf* α *q r*
proof −
assume $p \succeq[\mathcal{R}]$ *q*
hence *f*: *measure-pmf.expectation p u* $\geq$ *measure-pmf.expectation q u*
using *u a1 a2 ordinal-utility.util-def* **by** *fastforce*
have *measure-pmf.expectation* (*mix-pmf* α *p r*) *u* $\geq$ *measure-pmf.expectation* (*mix-pmf* α *q r*) *u*
proof −
have *measure-pmf.expectation* (*mix-pmf* α *p r*) *u* =
α ∗ *measure-pmf.expectation p u* + (*1* − α) ∗ *measure-pmf.expectation r u*
using *expected-value-mix-pmf-distrib*[*of p r* α *u*] *assms fnts a4* **by** *fastforce*
moreover have *measure-pmf.expectation* (*mix-pmf* α *q r*) *u* =
α ∗ *measure-pmf.expectation q u* + (*1* − α) ∗ *measure-pmf.expectation r u*
using *expected-value-mix-pmf-distrib*[*of q r* α *u*] *assms fnts a4* **by** *fastforce*
ultimately show *?thesis* **using** *f* **using** *a4* **by** *auto*
qed

then show *mix-pmf* α *p r* $\succeq[\mathcal{R}]$ *mix-pmf* α *q r*
using *u ordinal-utility-expected-value′ ocU-is-expected-utility-bernoulli in-lots*
by (*simp add*: *in-lots ordinal-utility-def*)
qed
moreover have *mix-pmf* α *p r* $\succeq[\mathcal{R}]$ *mix-pmf* α *q r* $\Longrightarrow$ *p* $\succeq[\mathcal{R}]$ *q*
proof −
assume *mix-pmf* α *p r* $\succeq[\mathcal{R}]$ *mix-pmf* α *q r*
hence *f*:*measure-pmf.expectation* (*mix-pmf* α *p r*) *u* $\geq$ *measure-pmf.expectation* (*mix-pmf* α *q r*) *u*
using *ordinal-utility.ordinal-utility-left u* **by** *fastforce*
hence *measure-pmf.expectation p u* $\geq$ *measure-pmf.expectation q u*
proof −
have *measure-pmf.expectation* (*mix-pmf* α *p r*) *u* =
α ∗ *measure-pmf.expectation p u* + (*1* − α) ∗ *measure-pmf.expectation r u*
using *expected-value-mix-pmf-distrib*[*of p r* α *u*] *assms fnts a4* **by** *fastforce*
moreover have *measure-pmf.expectation* (*mix-pmf* α *q r*) *u* =
α ∗ *measure-pmf.expectation q u* + (*1* − α) ∗ *measure-pmf.expectation r u*
using *expected-value-mix-pmf-distrib*[*of q r* α *u*] *assms fnts a4* **by** *fastforce*
ultimately show *?thesis* **using** *f* **using** *a4* **by** *auto*
qed
then show *p* $\succeq[\mathcal{R}]$ *q*
using *a1 a2 ordinal-utility.util-def-conf u* **by** *fastforce*
qed
ultimately show *p* $\succeq[\mathcal{R}]$ *q* = *mix-pmf* α *p r* $\succeq[\mathcal{R}]$ *mix-pmf* α *q r*
by *blast*
qed

lemma *exists-weight-for-equality*:
assumes *a* > *c* **and** *a* $\geq$ *b* **and** *b* $\geq$ *c*
shows $\exists$ (*e*::*real*) $\in$ {*0..1*}. (*1*−*e*) ∗ *a* + *e* ∗ *c* = *b*
proof −
from *assms* **have** *b* $\in$ *closed-segment a c*
by (*simp add*: *closed-segment-eq-real-ivl*)
thus *?thesis* **by** (*auto simp*: *closed-segment-def*)
qed

lemma *vNM-utilty-implies-continuity*:
assumes *fnt*: *finite outcomes* **and** *outcomes* $\neq$ {}
assumes $\exists$ *u*. *ordinal-utility* (*lotteries-on outcomes*) $\mathcal{R}$ (λ*x*. *measure-pmf.expectation x u*)
shows *continuous-vnm* (*lotteries-on outcomes*) $\mathcal{R}$
proof (*rule continuous-vnmI*)
fix *p q r*
assume *a1*: *p* $\in$ $\mathcal{P}$ *outcomes*
assume *a2*: *q* $\in$ $\mathcal{P}$ *outcomes*
assume *a3*: *r* $\in$ $\mathcal{P}$ *outcomes*
assume *a4*: *p* $\succeq[\mathcal{R}]$ *q* $\wedge$ *q* $\succeq[\mathcal{R}]$ *r*
then have *g*: *p* $\succeq[\mathcal{R}]$ *r*
by (*meson assms*(*3*) *ordinal-utility.util-imp-trans transD*)

```
  obtain u where
    u: ordinal-utility (lotteries-on outcomes) ℛ (λx. measure-pmf.expectation x u)
    using assms by blast
  have geqa: measure-pmf.expectation p u ≥ measure-pmf.expectation q u
    measure-pmf.expectation q u ≥ measure-pmf.expectation r u
    using a4 u by (meson ordinal-utility.ordinal-utility-left)+
  have fnts: finite p finite q finite r
    using a1 a2 a3 fnt infinite-super lotteries-on-def by auto+
  consider p ≻[ℛ] r | p ≈[ℛ] r
    using g by auto
  then show ∃α∈{0..1}. mix-pmf α p r ≈[ℛ] q
  proof (cases)
    case 1
    define a where a: a = measure-pmf.expectation p u
    define b where b: b = measure-pmf.expectation r u
    define c where c: c = measure-pmf.expectation q u
    have a > b
      using 1 a1 a2 a3 a b ordinal-utility.util-def-conf u by force
    have c ≤ a b ≤ c
      using geqa a b c by blast+
    then obtain e ::real where
      e: e ∈ {0..1} (1−e) ∗ a + e ∗ b = c
      using exists-weight-for-equality[of b a c] ‹b < a› by blast
    have ∗:1−e ∈ {0..1}
      using e(1) by auto
    hence measure-pmf.expectation (mix-pmf (1−e) p r) u =
     (1−e) ∗ measure-pmf.expectation p u + e ∗ measure-pmf.expectation r u
      using expected-value-mix-pmf-distrib[of p r 1−e u] fnts by fastforce
    also have ... = (1−e) ∗ a + e ∗ b
      using a b by auto
    also have ... = c
      using c e by auto
     finally have f: measure-pmf.expectation (mix-pmf (1−e) p r) u = mea-
sure-pmf.expectation q u
      using c by blast
    hence mix-pmf (1−e) p r ≈[ℛ] q
       using expected-value-is-utility-function[of outcomes mix-pmf (1−e) p r q ℛ
u] ∗
    proof −
      have mix-pmf (1 − e) p r ∈ 𝒫 outcomes
        using ‹1 − e ∈ {0..1}› a1 a3 mix-in-lot by blast
      then show ?thesis
        using f a2 ordinal-utility.util-def u by fastforce
    qed
    then show ?thesis
      using exists-weight-for-equality expected-value-mix-pmf-distrib ∗ by blast
  next
    case 2
    have r ≈[ℛ] q
```

by (*meson 2 a4 assms*(*3*) *ordinal-utility.util-imp-trans transD*)
then show *?thesis* **by** *force*
qed
qed

theorem *Von-Neumann-Morgenstern-Utility-Theorem*:
assumes *fnt*: *finite outcomes* **and** *outcomes* $\neq$ {}
shows *rational-preference* (*lotteries-on outcomes*) $\mathcal{R}$ $\wedge$
independent-vnm (*lotteries-on outcomes*) $\mathcal{R}$ $\wedge$
continuous-vnm (*lotteries-on outcomes*) $\mathcal{R}$ $\longleftrightarrow$
($\exists$ *u*. *ordinal-utility* (*lotteries-on outcomes*) $\mathcal{R}$ (λ*x*. *measure-pmf.expectation x u*))
using *vNM-utility-implies-independence*[*OF assms*, *of* $\mathcal{R}$]
system-U-implies-vNM-utility[*OF assms*, *of* $\mathcal{R}$]
vNM-utilty-implies-continuity[*OF assms*, *of* $\mathcal{R}$]
ordinal-util-imp-rat-prefs[*of lotteries-on outcomes* $\mathcal{R}$] **by** *auto*

end

theory *Expected-Utility*
imports
Neumann-Morgenstern-Utility-Theorem
begin

6 Definition of vNM-utility function

We define a version of the vNM Utility function using the locale mechanism. Currently this definition and system U have no proven relation yet.

Important: u is actually not the von Neuman Utility Function, but a Bernoulli Utility Function. The Expected value p given u is the von Neumann Utility Function.

locale *vNM-utility* =
fixes *outcomes* :: *'a set*
fixes *relation* :: *'a pmf relation*
fixes *u* :: *'a* $\Rightarrow$ *real*
assumes *relation* $\subseteq$ (*lotteries-on outcomes* $\times$ *lotteries-on outcomes*)
assumes $\bigwedge$*p q*. *p* $\in$ *lotteries-on outcomes* $\Longrightarrow$
q $\in$ *lotteries-on outcomes* $\Longrightarrow$
p $\succeq$[*relation*] *q* $\longleftrightarrow$ *measure-pmf.expectation p u* $\geq$ *measure-pmf.expectation q u*
begin

lemma *vNM-utilityD*:
shows *relation* $\subseteq$ (*lotteries-on outcomes* $\times$ *lotteries-on outcomes*)
and *p* $\in$ *lotteries-on outcomes* $\Longrightarrow$ *q* $\in$ *lotteries-on outcomes* $\Longrightarrow$

p ⪰[*relation*] *q* ⟷ *measure-pmf.expectation p u* ≥ *measure-pmf.expectation q u*
using *vNM-utility-axioms vNM-utility-def* **by** (*blast*+)

lemma *not-outside*:
assumes *p* ⪰[*relation*] *q*
shows *p* ∈ *lotteries-on outcomes*
and *q* ∈ *lotteries-on outcomes*
proof (*goal-cases*)
case *1*
then show *?case*
by (*meson assms contra-subsetD mem-Sigma-iff vNM-utility-axioms vNM-utility-def*)
next
case *2*
then show *?case*
by (*metis assms mem-Sigma-iff subsetCE vNM-utility-axioms vNM-utility-def*)
qed

lemma *utility-ge*:
assumes *p* ⪰[*relation*] *q*
shows *measure-pmf.expectation p u* ≥ *measure-pmf.expectation q u*
using *assms vNM-utility-axioms vNM-utility-def*
by (*metis* (*no-types*, *lifting*) *not-outside*(*1*) *not-outside*(*2*))

end

sublocale *vNM-utility* ⊆ *ordinal-utility* (*lotteries-on outcomes*) *relation* (λ*p*. *measure-pmf.expectation p u*)
proof (*standard*, *goal-cases*)
case (*2 x y*)
then show *?case*
using *not-outside*(*1*) **by** *blast*
next
case (*3 x y*)
then show *?case*
by (*auto simp add*: *not-outside*(*2*))
qed (*metis* (*mono-tags*, *lifting*) *vNM-utility-axioms vNM-utility-def*)

context *vNM-utility*
begin

lemma *strict-preference-iff-strict-utility*:
assumes *p* ∈ *lotteries-on outcomes*
assumes *q* ∈ *lotteries-on outcomes*
shows *p* ≻[*relation*] *q* ⟷ *measure-pmf.expectation p u* > *measure-pmf.expectation q u*
by (*meson assms*(*1*) *assms*(*2*) *less-eq-real-def not-le util-def*)

lemma *pos-distrib-left*:

assumes $c > 0$
shows $(\sum z{\in}outcomes.\ pmf\ q\ z * (c * u\ z)) = c * (\sum z{\in}outcomes.\ pmf\ q\ z * (u\ z))$
proof −
have $(\sum z{\in}outcomes.\ pmf\ q\ z * (c * u\ z)) = (\sum z{\in}outcomes.\ pmf\ q\ z * c * u\ z)$
by (*simp add*: *ab-semigroup-mult-class.mult-ac*(*1*))
also have ... $= (\sum z{\in}outcomes.\ c * pmf\ q\ z * \ u\ z)$
by (*simp add*: *mult.commute*)
also have ... $= c * (\sum z{\in}outcomes.\ pmf\ q\ z * \ u\ z)$
by (*simp add*: *ab-semigroup-mult-class.mult-ac*(*1*) *sum-distrib-left*)
finally show *?thesis* **.**
qed

lemma *sum-pmf-util-commute*:
$(\sum a{\in}outcomes.\ pmf\ p\ a * u\ a) = (\sum a{\in}outcomes.\ u\ a * pmf\ p\ a)$
by (*simp add*: *mult.commute*)

7 Finite outcomes

context
assumes *fnt*: *finite outcomes*
begin

lemma *sum-equals-pmf-expectation*:
assumes $p \in$ *lotteries-on outcomes*
shows$(\sum z{\in}outcomes.\ (pmf\ p\ z) * (u\ z)) = measure\text{-}pmf.expectation\ p\ u$
proof −
have *fnt*: *finite outcomes*
by (*simp add*: *vNM-utilityD*(*1*) *fnt*)
have $measure\text{-}pmf.expectation\ p\ u = (\sum a{\in}outcomes.\ pmf\ p\ a * u\ a)$
using *support-in-outcomes assms fnt integral-measure-pmf-real*
sum-pmf-util-commute **by** *fastforce*
then show *?thesis*
using *real-scaleR-def* **by** *presburger*
qed

lemma *expected-utility-weak-preference*:
assumes $p \in$ *lotteries-on outcomes*
and $q \in$ *lotteries-on outcomes*
shows $p \succeq[relation]\ q \longleftrightarrow (\sum z{\in}outcomes.\ (pmf\ p\ z) * (u\ z)) \geq (\sum z{\in}outcomes.\ (pmf\ q\ z) * (u\ z))$
using *sum-equals-pmf-expectation*[*of* p, *OF assms*(*1*)]
sum-equals-pmf-expectation[*of* q, *OF assms*(*2*)]
vNM-utility-def assms(*1*) *assms*(*2*) *util-def-conf* **by** *presburger*

lemma *diff-leq-zero-weak-preference*:
assumes $p \in$ *lotteries-on outcomes*
and $q \in$ *lotteries-on outcomes*
shows $p \succeq q \longleftrightarrow ((\sum a{\in}outcomes.\ pmf\ q\ a * u\ a) - (\sum a{\in}outcomes.\ pmf\ p\ a$

$*$ *u a*) $\leq$ *0*)
  using *assms*(*1*) *assms*(*2*) *diff-le-0-iff-le*
  by (*metis* (*mono-tags*, *lifting*) *expected-utility-weak-preference*)

lemma *expected-utility-strict-preference*:
  assumes $p \in$ *lotteries-on outcomes*
    and $q \in$ *lotteries-on outcomes*
  shows $p \succ$[*relation*] $q \longleftrightarrow$ *measure-pmf.expectation p u* $>$ *measure-pmf.expectation q u*
  using *assms expected-utility-weak-preference less-eq-real-def not-le*
  by (*metis* (*no-types*, *lifting*) *util-def-conf*)

lemma *scale-pos-left*:
  assumes $c > 0$
  shows *vNM-utility outcomes relation* ($\lambda x.\ c * u\ x$)
proof(*standard*, *goal-cases*)
  case *1*
  then show *?case*
    using *vNM-utility-axioms vNM-utility-def* **by** *blast*
next
  case (*2 p q*)
  have $q \in$ *lotteries-on outcomes* **and** $p \in$ *lotteries-on outcomes*
    using *2*(*2*) **by** (*simp add*: *fnt 2*(*1*))+
  then have $*$: $p \succeq q$ = (*measure-pmf.expectation q u* $\leq$ *measure-pmf.expectation p u*)
    using *expected-utility-weak-preference*[*of p q*] *assms* **by** *blast*
  have *dist-c*: ($\sum z \in$*outcomes*. (*pmf q z*) $*$ ($c * u\ z$)) $= c *$ ($\sum z \in$*outcomes*. (*pmf q z*) $*$ (*u z*))
    using *pos-distrib-left*[*of c q*] *assms* **by** *blast*
  have *dist-c′*: ($\sum z \in$*outcomes*. (*pmf p z*) $*$ ($c * u\ z$)) $= c *$ ($\sum z \in$*outcomes*. (*pmf p z*) $*$ (*u z*))
    using *pos-distrib-left*[*of c p*] *assms* **by** *blast*
  have $p \succeq q \longleftrightarrow$ (($\sum z \in$*outcomes*. (*pmf q z*) $*$ ($c * u\ z$)) $\leq$ ($\sum z \in$*outcomes*. (*pmf p z*) $*$ ($c * u\ z$)))
  proof (*rule iffI*)
    assume $p \succeq q$
    then have ($\sum z \in$*outcomes*. *pmf q z* $*$ (*u z*)) $\leq$ ($\sum z \in$*outcomes*. *pmf p z* $*$ (*u z*))
      using *utility-ge*
      using *2*(*1*) *2*(*2*) *sum-equals-pmf-expectation* **by** *presburger*
    then show ($\sum z \in$*outcomes*. *pmf q z* $*$ ($c * u\ z$)) $\leq$ ($\sum z \in$*outcomes*. *pmf p z* $*$ ($c * u\ z$))
      using *dist-c dist-c′*
      by (*simp add*: *assms*)
  next
    assume ($\sum z \in$*outcomes*. *pmf q z* $*$ ($c * u\ z$)) $\leq$ ($\sum z \in$*outcomes*. *pmf p z* $*$ ($c * u\ z$))
    then have ($\sum z \in$*outcomes*. *pmf q z* $*$ (*u z*)) $\leq$ ($\sum z \in$*outcomes*. *pmf p z* $*$ (*u z*))

    using *2*(*1*) *mult-le-cancel-iff2 assms* **by** (*simp add*: *dist-c dist-c′*)
  then show $p \succeq q$
    using *2*(*2*) *assms 2*(*1*) **by** (*simp add*: ∗ *sum-equals-pmf-expectation*)
  qed
  then show *?case*
    by (*simp add*: ∗ *assms*)
qed

lemma *strict-alt-def*:
  assumes $p \in$ *lotteries-on outcomes*
    and $q \in$ *lotteries-on outcomes*
  shows $p \succ$[*relation*] $q \longleftrightarrow$
      ($\sum z \in$*outcomes*. (*pmf p z*) ∗ (*u z*)) > ($\sum z \in$*outcomes*. (*pmf q z*) ∗ (*u z*))
  using *sum-equals-pmf-expectation*[*of p*, *OF assms*(*1*)] *assms*(*1*) *assms*(*2*)
    sum-equals-pmf-expectation[*of q*, *OF assms*(*2*)] *strict-prefernce-iff-strict-utility*
  by *presburger*

lemma *strict-alt-def-utility-g*:
  assumes $p \succ$[*relation*] q
  shows ($\sum z \in$*outcomes*. (*pmf p z*) ∗ (*u z*)) > ($\sum z \in$*outcomes*. (*pmf q z*) ∗ (*u z*))
  using *assms not-outside*(*1*) *not-outside*(*2*) *strict-alt-def*
  by *meson*

end

end

lemma *vnm-utility-is-ordinal-utility*:
  assumes *vNM-utility outcomes relation u*
  shows *ordinal-utility* (*lotteries-on outcomes*) *relation* (λp. *measure-pmf.expectation p u*)
proof (*standard*, *goal-cases*)
  case (*1 x y*)
  then show *?case*
    using *assms vNM-utility-def* **by** *blast*
next
  case (*2 x y*)
  then show *?case*
    using *assms vNM-utility.not-outside*(*1*) **by** *blast*
next
  case (*3 x y*)
  then show *?case*
    using *assms vNM-utility.not-outside*(*2*) **by** *blast*
qed

lemma *vnm-utility-imp-reational-prefs*:
  assumes *vNM-utility outcomes relation u*
  shows *rational-preference* (*lotteries-on outcomes*) *relation*
proof (*standard*,*goal-cases*)

```
  case (1 x y)
  then show ?case
    using assms vNM-utility.not-outside(1) by blast
next
  case (2 x y)
  then show ?case
    using assms vNM-utility.not-outside(2) by blast
next
  case 3
  have t: trans relation
    using assms ordinal-utility.util-imp-trans vnm-utility-is-ordinal-utility by blast
  have refl-on (lotteries-on outcomes) relation
    by (meson assms order-refl refl-on-def vNM-utility-def)
  then show ?case
    using preorder-on-def t by blast
next
  case 4
  have total-on (lotteries-on outcomes) relation
    using ordinal-utility.util-imp-total[of lotteries-on outcomes
        relation (λp. (∑ z∈outcomes. (pmf p z) ∗ (u z)))]
      assms vnm-utility-is-ordinal-utility
    using ordinal-utility.util-imp-total by blast
  then show ?case
    by simp
qed

theorem expected-utilty-theorem-form-vnm-utility:
  assumes fnt: finite outcomes and outcomes ≠ {}
  shows rational-preference (lotteries-on outcomes) ℛ ∧
        independent-vnm (lotteries-on outcomes) ℛ ∧
        continuous-vnm (lotteries-on outcomes) ℛ ⟷
        (∃ u. vNM-utility outcomes ℛ u)
proof
  assume rational-preference (𝒫 outcomes) ℛ ∧ independent-vnm (𝒫 outcomes)
ℛ ∧ continuous-vnm (𝒫 outcomes) ℛ
   with Von-Neumann-Morgenstern-Utility-Theorem[of outcomes ℛ, OF assms]
have
  (∃ u. ordinal-utility (𝒫 outcomes) ℛ (λx. measure-pmf.expectation x u)) using
assms by blast
  then obtain u where
    u: ordinal-utility (𝒫 outcomes) ℛ (λx. measure-pmf.expectation x u)
    by auto
  have vNM-utility outcomes ℛ u
  proof (standard, goal-cases)
    case 1
    then show ?case
      using u ordinal-utility.relation-subset-crossp by blast
  next
    case (2 p q)
```

```
    then show ?case
      using assms(2) expected-value-is-utility-function fnt u by blast
  qed
  then show ∃ u. vNM-utility outcomes ℛ u
    by blast
next
  assume a: ∃ u. vNM-utility outcomes ℛ u
  then have rational-preference (𝒫 outcomes) ℛ
    using vnm-utility-imp-reational-prefs by auto
  moreover have independent-vnm (𝒫 outcomes) ℛ
   using a by (meson assms(2) fnt vNM-utility-implies-independence vnm-utility-is-ordinal-utility)
  moreover have continuous-vnm (𝒫 outcomes) ℛ
   using a by (meson assms(2) fnt vNM-utilty-implies-continuity vnm-utility-is-ordinal-utility)
   ultimately show rational-preference (𝒫 outcomes) ℛ ∧ independent-vnm (𝒫
outcomes) ℛ ∧ continuous-vnm (𝒫 outcomes) ℛ
    by auto
qed

end
```

8 Related work

Formalizations in Social choice theory has been formalized by Wiedijk [13], Nipkow [7], and Gammie [4, 5]. Vestergaard [12], Le Roux, Martin-Dorel, and Soloviev [10, 11] provide formalizations of results in game theory. A library for algorithmic game theory in Coq is described in[1].

Related work in economics includes the verification of financial systems [9], binomial pricing models [3], and VCG-Auctions [6]. In microeconomics we discussed a formalization of two economic models and the First Welfare Theorem [8].

To our knowledge the only work that uses expected utility theory is that of Eberl [2]. Since we focus on the underlying theory of expected utility, we found that there is only little overlap.

References

[1] A. Bagnall, S. Merten, and G. Stewart. A library for algorithmic game theory in ssreflect/coq. *Journal of Formalized Reasoning*, 10(1):67–95, 2017.

[2] M. Eberl. Randomised social choice theory. *Archive of Formal Proofs*, May 2016. http://isa-afp.org/entries/Randomised_Social_Choice.shtml, Formal proof development.

[3] M. Echenim and N. Peltier. The binomial pricing model in finance: A formalization in isabelle. In L. de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *LNCS*, pages 546–562. Springer, 2017.

[4] P. Gammie. Some classical results in social choice theory. *Archive of Formal Proofs*, Nov. 2008. http://isa-afp.org/entries/SenSocialChoice.html, Formal proof development.

[5] P. Gammie. Stable matching. *Archive of Formal Proofs*, Oct. 2016. http://isa-afp.org/entries/Stable_Matching.html, Formal proof development.

[6] M. Kerber, C. Lange, C. Rowat, and W. Windsteiger. Developing an auction theory toolbox. *AISB 2013*, pages 1–4, 2013.

[7] T. Nipkow. Arrow and Gibbard-Satterthwaite. *Archive of Formal Proofs*, 2008.

[8] J. Parsert and C. Kaliszyk. Formal Microeconomic Foundations and the First Welfare Theorem. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, pages 91–101. ACM, 2018.

[9] G. O. Passmore and D. Ignatovich. Formal verification of financial algorithms. In L. de Moura, editor, *Automated Deduction – CADE 26*, pages 26–41. Springer, 2017.

[10] S. L. Roux. Acyclic Preferences and Existence of Sequential Nash Equilibria: A formal and constructive equivalence. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *LNCS*, pages 293–309. Springer, 2009.

[11] S. L. Roux, É. Martin-Dorel, and J. Smaus. An existence theorem of Nash Equilibrium in Coq and Isabelle. In P. Bouyer, A. Orlandini, and P. S. Pietro, editors, *Proceedings Eighth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017, Roma, Italy, 20-22 September 2017.*, volume 256 of *EPTCS*, pages 46–60, 2017.

[12] R. Vestergaard. A constructive approach to sequential nash equilibria. *Inf. Process. Lett.*, 97(2):46–51, 2006.

[13] F. Wiedijk. Formalizing Arrow's theorem. *Sadhana*, 34(1):193–220, Feb 2009.