Pricing in discrete financial models

Mnacho Echenim

September 1, 2025

Contents

1 Generated subalgebras

This section contains definitions and properties related to generated subalgebras.

theory *Generated-Subalgebra* **imports** *HOL−Probability.Probability*

begin

definition *gen-subalgebra* **where**
gen-subalgebra M G = sigma (space M) G

lemma *gen-subalgebra-space*:
shows *space (gen-subalgebra M G) = space M*
by (*simp add*: *gen-subalgebra-def space-measure-of-conv*)

lemma *gen-subalgebra-sets*:
assumes *G ⊆ sets M*
and *A ∈ G*
shows *A ∈ sets (gen-subalgebra M G)*
by (*metis assms gen-subalgebra-def sets.space-closed sets-measure-of sigma-sets.Basic subset-trans*)

lemma *gen-subalgebra-sig-sets*:
assumes *G ⊆ Pow (space M)*
shows *sets (gen-subalgebra M G) = sigma-sets (space M) G* **unfolding** *gen-subalgebra-def*
by (*metis assms gen-subalgebra-def sets-measure-of*)

lemma *gen-subalgebra-sigma-sets*:
assumes *G ⊆ sets M*
and *sigma-algebra (space M) G*
shows *sets (gen-subalgebra M G) = G*
using *assms* **by** (*simp add*: *gen-subalgebra-def sigma-algebra.sets-measure-of-eq*)

lemma *gen-subalgebra-is-subalgebra*:
assumes *sub*: *G ⊆ sets M*
and *sigal*:*sigma-algebra (space M) G*
shows *subalgebra M (gen-subalgebra M G)* (**is** *subalgebra M ?N*)
unfolding *subalgebra-def*
proof (*intro conjI*)
show *space ?N = space M* **using** *space-measure-of-conv*[*of* (*space M*)] **unfolding** *gen-subalgebra-def* **by** *simp*
have *geqn*: *G = sets ?N* **using** *assms* **by** (*simp add*:*gen-subalgebra-sigma-sets*)
thus *sets ?N ⊆ sets M* **using** *assms* **by** *simp*
qed

definition *fct-gen-subalgebra* :: *'a measure ⇒ 'b measure ⇒ ('a ⇒ 'b) ⇒ 'a measure* **where**
fct-gen-subalgebra M N X = gen-subalgebra M (sigma-sets (space M) {X −‘ B ∩ (space M) | B. B ∈ sets N})

lemma *fct-gen-subalgebra-sets*:
shows *sets* (*fct-gen-subalgebra M N X*) = *sigma-sets* (*space M*) {*X* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets N*}
unfolding *fct-gen-subalgebra-def gen-subalgebra-def*
proof −
have {*X* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets N*} ⊆ *Pow* (*space M*)
by *blast*
then show *sets* (*sigma* (*space M*) (*sigma-sets* (*space M*) {*X* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets N*})) = *sigma-sets* (*space M*) {*X* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets N*}
by (*meson sigma-algebra.sets-measure-of-eq sigma-algebra-sigma-sets*)
qed

lemma *fct-gen-subalgebra-space*:
shows *space* (*fct-gen-subalgebra M N X*) = *space M*
unfolding *fct-gen-subalgebra-def* **by** (*simp add*: *gen-subalgebra-space*)

lemma *fct-gen-subalgebra-eq-sets*:
assumes *sets M* = *sets P*
shows *fct-gen-subalgebra M N X* = *fct-gen-subalgebra P N X*
proof −
have *space M* = *space P* **using** *sets-eq-imp-space-eq assms* **by** *auto*
thus *?thesis* **unfolding** *fct-gen-subalgebra-def gen-subalgebra-def* **by** *simp*
qed

lemma *fct-gen-subalgebra-sets-mem*:
assumes *B*∈ *sets N*
shows *X* −‘ *B* ∩ (*space M*) ∈ *sets* (*fct-gen-subalgebra M N X*) **unfolding** *fct-gen-subalgebra-def*
proof −
have *f1*: {*X* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets N*} ⊆ *Pow* (*space M*)
by *blast*
have ∃ *A*. *X* −‘ *B* ∩ *space M* = *X* −‘ *A* ∩ *space M* ∧ *A* ∈ *sets N*
by (*metis assms*)
then show *X* −‘ *B* ∩ *space M* ∈ *sets* (*gen-subalgebra M* (*sigma-sets* (*space M*) {*X* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets N*}))
using *f1* **by** (*simp add*: *gen-subalgebra-def sigma-algebra.sets-measure-of-eq sigma-algebra-sigma-sets*)
qed

lemma *fct-gen-subalgebra-is-subalgebra*:
assumes *X*∈ *measurable M N*
shows *subalgebra M* (*fct-gen-subalgebra M N X*)
unfolding *fct-gen-subalgebra-def*
proof (*rule gen-subalgebra-is-subalgebra*)
show *sigma-sets* (*space M*) {*X* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets N*} ⊆ *sets M* (**is** *?L* ⊆ *?R*)
proof (*rule sigma-algebra.sigma-sets-subset*)

show $\{X -` B \cap space\ M\ |B.\ B \in sets\ N\} \subseteq sets\ M$
proof
fix a
assume $a \in \{X -` B \cap (space\ M) \mid B.\ B \in sets\ N\}$
then obtain B **where** $B \in sets\ N$ **and** $a = X -` B \cap (space\ M)$ **by** *auto*
thus $a \in sets\ M$ **using** *measurable-sets assms* **by** *simp*
qed
show *sigma-algebra* (*space M*) (*sets M*) **using** *measure-space* **by** (*auto simp add*: *measure-space-def*)
qed
show *sigma-algebra* (*space M*) *?L*
proof (*rule sigma-algebra-sigma-sets*)
let *?preimages* $= \{X -` B \cap (space\ M) \mid B.\ B \in sets\ N\}$
show *?preimages* $\leq$ *Pow* (*space M*) **using** *assms* **by** *auto*
qed
qed

lemma *fct-gen-subalgebra-fct-measurable*:
assumes $X \in space\ M \to space\ N$
shows $X \in$ *measurable* (*fct-gen-subalgebra M N X*) *N*
unfolding *measurable-def*
proof ((*intro CollectI*), (*intro conjI*))
have *speq*: *space M* = *space* (*fct-gen-subalgebra M N X*)
by (*simp add*: *fct-gen-subalgebra-space*)
show $X \in space$ (*fct-gen-subalgebra M N X*) $\to$ *space N*
proof −
have $X \in space\ M \to space\ N$ **using** *assms* **by** *simp*
thus *?thesis* **using** *speq* **by** *simp*
qed
show $\forall\, y \in sets\ N.$
$X -` y \cap space$ (*fct-gen-subalgebra M N X*) $\in sets$ (*fct-gen-subalgebra M N X*)
using *fct-gen-subalgebra-sets-mem speq* **by** *metis*
qed

lemma *fct-gen-subalgebra-min*:
assumes *subalgebra M P*
and $f \in$ *measurable P N*
shows *subalgebra P* (*fct-gen-subalgebra M N f*)
unfolding *subalgebra-def*
proof (*intro conjI*)
let *?Mf* = *fct-gen-subalgebra M N f*
show *space ?Mf* = *space P* **using** *assms*
by (*simp add*: *fct-gen-subalgebra-def gen-subalgebra-space subalgebra-def*)
show *inc*: *sets ?Mf* $\subseteq$ *sets P*
proof −

have *space M = space P* **using** *assms* **by** (*simp add*:*subalgebra-def*)
have *f*∈ *measurable M N* **using** *assms* **using** *measurable-from-subalg* **by** *blast*
have *sigma-algebra* (*space P*) (*sets P*) **using** *assms measure-space measure-space-def* **by** *auto*
have ∀ *A* ∈ *sets N*. *f*−‘*A* ∩ *space P* ∈ *sets P* **using** *assms* **by** *simp*
hence {*f* −‘ *A* ∩ (*space M*) | *A*. *A* ∈ *sets N*} ⊆ *sets P* **using** ‹*space M = space P*› **by** *auto*
hence *sigma-sets* (*space M*) {*f* −‘ *A* ∩ (*space M*) | *A*. *A* ∈ *sets N*} ⊆ *sets P*
by (*simp add*: ‹*sigma-algebra* (*space P*) (*sets P*)› ‹*space M = space P*› *sigma-algebra.sigma-sets-subset*)
thus *?thesis* **using** *fct-gen-subalgebra-sets* ‹*f* ∈ *M* →$_M$ *N*› ‹*space M = space P*› *assms*(*2*)
measurable-sets mem-Collect-eq sets.sigma-sets-subset subsetI **by** *blast*
qed
qed

lemma *fct-preimage-sigma-sets*:
assumes *X*∈ *space M* → *space N*
shows *sigma-sets* (*space M*) {*X* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets N*} = {*X* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets N*} (**is** *?L = ?R*)
proof
show *?R*⊆ *?L* **by** *blast*
show *?L*⊆ *?R*
proof
fix *A*
assume *A*∈ *?L*
thus *A*∈ *?R*
proof (*induct rule*:*sigma-sets.induct*, *auto*)
{
fix *B*
assume *B*∈ *sets N*
let *?cB* = *space N* − *B*
have *?cB* ∈ *sets N* **by** (*simp add*: ‹*B* ∈ *sets N*› *sets.compl-sets*)
have *space M* − *X* −‘ *B* ∩ *space M* = *X* −‘ *?cB* ∩ *space M*
proof
show *space M* − *X* −‘ *B* ∩ *space M* ⊆ *X* −‘ (*space N* − *B*) ∩ *space M*
proof
fix *w*
assume *w* ∈ *space M* − *X* −‘ *B* ∩ *space M*
hence *X w* ∈ (*space N* − *B*) **using** *assms* **by** *blast*
thus *w*∈ *X* −‘ (*space N* − *B*) ∩ *space M* **using** ‹*w* ∈ *space M* − *X* −‘ *B* ∩ *space M*› **by** *blast*
qed
show *X* −‘ (*space N* − *B*) ∩ *space M* ⊆ *space M* − *X* −‘ *B* ∩ *space M*
proof
fix *w*
assume *w*∈ *X* −‘ (*space N* − *B*) ∩ *space M*
thus *w* ∈ *space M* − *X* −‘ *B* ∩ *space M* **by** *blast*
qed

qed
thus $\exists Ba.\ space\ M - X - `\ B \cap space\ M = X - `\ Ba \cap space\ M \wedge Ba \in sets\ N$ **using** ‹*?cB* $\in$ *sets N*› **by** *auto*
}
{
fix *S*::*nat* $\Rightarrow$ *'a set*
assume $(\bigwedge i.\ \exists B.\ S\ i = X - `\ B \cap space\ M \wedge B \in sets\ N)$
hence $(\forall i.\ \exists B.\ S\ i = X - `\ B \cap space\ M \wedge B \in sets\ N)$ **by** *auto*
hence $\exists\ f.\ \forall\ x.\ S\ x = X - `(f\ x) \cap space\ M \wedge (f\ x) \in sets\ N$
using *choice*[*of* $\lambda i\ B\ .\ S\ i = X - `\ B \cap space\ M \wedge B \in sets\ N$] **by** *simp*
from *this* **obtain** *rep* **where** $\forall i.\ S\ i = X - `\ (rep\ i) \cap space\ M \wedge (rep\ i) \in sets\ N$ **by** *auto* **note** *rProp* = *this*
let *?uB* = $\bigcup i \in UNIV.\ rep\ i$
have *?uB* $\in$ *sets N*
by (*simp add*: ‹$\forall i.\ S\ i = X - `\ rep\ i \cap space\ M \wedge rep\ i \in sets\ N$› *countable-Un-Int*(*1*))
have $(\bigcup x.\ S\ x) = X - `\ ?uB \cap space\ M$
proof
show $(\bigcup x.\ S\ x) \subseteq X - `\ (\bigcup i.\ rep\ i) \cap space\ M$
proof
fix *w*
assume *w*$\in$ $(\bigcup x.\ S\ x)$
hence $\exists x.\ w \in S\ x$ **by** *auto*
from *this* **obtain** *x* **where** $w \in S\ x$ **by** *auto*
hence *w*$\in$ $X - `\ rep\ x \cap space\ M$ **using** *rProp* **by** *simp*
hence *w*$\in$ $(\bigcup i.\ (X - `(rep\ i) \cap space\ M))$ **by** *blast*
also have ... = $X - `\ (\bigcup i.\ rep\ i) \cap space\ M$ **by** *auto*
finally show $w \in X - `\ (\bigcup i.\ rep\ i) \cap space\ M$ **.**
qed
show $X - `\ (\bigcup i.\ rep\ i) \cap space\ M \subseteq (\bigcup x.\ S\ x)$
proof
fix *w*
assume *w*$\in$ $X - `\ (\bigcup i.\ rep\ i) \cap space\ M$
hence $\exists\ x.$ *w*$\in$ $X - `\ (rep\ x) \cap space\ M$ **by** *auto*
from *this* **obtain** *x* **where** *w*$\in$ $X - `\ (rep\ x) \cap space\ M$ **by** *auto*
hence *w*$\in$ $S\ x$ **using** *rProp* **by** *simp*
thus *w*$\in$ $(\bigcup x.\ S\ x)$ **by** *blast*
qed
qed
thus $\exists B.\ (\bigcup x.\ S\ x) = X - `\ B \cap space\ M \wedge B \in sets\ N$ **using** ‹*?uB* $\in$ *sets N*› **by** *auto*
}
qed
qed
qed

lemma *fct-gen-subalgebra-sigma-sets*:
assumes *X*$\in$ *space M* $\rightarrow$ *space N*
shows *sets* (*fct-gen-subalgebra M N X*) = $\{X - `\ B \cap space\ M\ |B.\ B \in sets\ N\}$

by (*simp add*: *assms fct-gen-subalgebra-sets fct-preimage-sigma-sets*)

lemma *fct-gen-subalgebra-info*:
assumes $f \in$ *space* $M \to$ *space* N
and $x \in$ *space* M
and $w \in$ *space* M
and $f\ x = f\ w$
shows $\bigwedge A.\ A \in$ *sets* (*fct-gen-subalgebra* $M\ N\ f$) $\Longrightarrow (x \in A) = (w \in A)$
proof −
{fix A
assume $A \in$ *sigma-sets* (*space* M) $\{f\ -`\ B \cap$ (*space* M) $\mid B.\ B \in$ *sets* $N\}$
from *this* **have** $(x \in A) = (w \in A)$
proof (*induct rule*:*sigma-sets.induct*)
{
fix a
assume $a \in \{f\ -`\ B \cap$ *space* $M \mid B.\ B \in$ *sets* $N\}$
hence $\exists\ B \in$ *sets* $N.\ a = f\ -`\ B \cap$ *space* M **by** *auto*
from *this* **obtain** B **where** $B \in$ *sets* N **and** $a = f\ -`\ B \cap$ *space* M **by** *blast*
note *bhyps* = *this*
show $(x \in a) = (w \in a)$ **by** (*simp add*: *assms*(2) *assms*(3) *assms*(4) *bhyps*(2))
}
{
fix a
assume $a \in$ *sigma-sets* (*space* M) $\{f\ -`\ B \cap$ *space* $M \mid B.\ B \in$ *sets* $N\}$
and $(x \in a) = (w \in a)$ **note** *xh* = *this*
show $(x \in$ *space* $M - a) = (w \in$ *space* $M - a)$ **by** (*simp add*: *assms*(2) *assms*(3) *xh*(2))
}
{
fix *a*::*nat* ⇒ *'a set*
assume $(\bigwedge i.\ a\ i \in$ *sigma-sets* (*space* M) $\{f\ -`\ B \cap$ *space* $M \mid B.\ B \in$ *sets* $N\})$
and $(\bigwedge i.\ (x \in a\ i) = (w \in a\ i))$
show $(x \in \bigcup (a\ `\ UNIV)) = (w \in \bigcup (a\ `\ UNIV))$ **by** (*simp add*: ‹$\bigwedge i.\ (x \in a\ i) = (w \in a\ i)$›)
}
{show $(x \in \{\}) = (w \in \{\})$ **by** *simp***}**
qed} note *eqsig* = *this*
fix A
assume $A \in$ *sets* (*fct-gen-subalgebra* $M\ N\ f$)
hence $A \in$ *sigma-sets* (*space* M) $\{f\ -`\ B \cap$ (*space* M) $\mid B.\ B \in$ *sets* $N\}$
using *assms*(1) *fct-gen-subalgebra-sets* **by** *blast*
thus $(x \in A) = (w \in A)$ **using** *eqsig* **by** *simp*
qed

1.1 Independence between a random variable and a subalgebra.

definition (**in** *prob-space*) *subalgebra-indep-var* :: (*'a* ⇒ *real*) ⇒ *'a measure* ⇒ *bool* **where**
subalgebra-indep-var X N ⟷
X∈ *borel-measurable M* &
(*subalgebra M N*) &
(*indep-set* (*sigma-sets* (*space M*) { *X* −‘ *A* ∩ *space M* | *A*. *A* ∈ *sets borel*}) (*sets N*))

lemma (**in** *prob-space*) *indep-set-mono*:
assumes *indep-set A B*
assumes *A'* ⊆ *A*
assumes *B'* ⊆ *B*
shows *indep-set A' B'*
by (*meson indep-sets2-eq assms subsetCE subset-trans*)

lemma (**in** *prob-space*) *subalgebra-indep-var-indicator*:
fixes *X*::*'a*⇒*real*
assumes *subalgebra-indep-var X N*
and *X* ∈ *borel-measurable M*
and *A* ∈ *sets N*
shows *indep-var borel X borel* (*indicator A*)
proof ((*rule indep-var-eq*[*THEN iffD2*]), (*intro conjI*))
let *?IA* = (*indicator A*)::*'a*⇒ *real*
show *bm*:*random-variable borel X* **by** (*simp add*: *assms*(*2*))
show *random-variable borel ?IA* **using** *assms indep-setD-ev2* **unfolding** *subalgebra-indep-var-def* **by** *auto*
show *indep-set* (*sigma-sets* (*space M*) {*X* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets borel*}) (*sigma-sets* (*space M*) {*?IA* −‘ *Aa* ∩ *space M* |*Aa*. *Aa* ∈ *sets borel*})
proof (*rule indep-set-mono*)
show *sigma-sets* (*space M*) {*X* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets borel*} ⊆ *sigma-sets* (*space M*) {*X* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets borel*} **by** *simp*
show *sigma-sets* (*space M*) {*?IA* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets borel*} ⊆ *sets N*
proof −
have *sigma-algebra* (*space M*) (*sets N*) **using** *assms*
by (*metis subalgebra-indep-var-def sets.sigma-algebra-axioms subalgebra-def*)
have *sigma-sets* (*space M*) {*?IA* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets borel*} ⊆ *sigma-sets* (*space M*) (*sets N*)
proof (*rule sigma-sets-subseteq*)
show {*?IA* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets borel*} ⊆ *sets N*
proof
fix *x*
assume *x* ∈ {*?IA* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets borel*}
then obtain *B* **where** *B* ∈ *sets borel* **and** *x* = *?IA* −‘ *B* ∩ *space M* **by** *auto*
thus *x* ∈ *sets N*

by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*3*) *borel-measurable-indicator measurable-sets subalgebra-indep-var-def subalgebra-def*)
qed
qed
also have ... = *sets N*
by (*simp add*: ‹*sigma-algebra* (*space M*) (*sets N*)› *sigma-algebra.sigma-sets-eq*)
finally show *sigma-sets* (*space M*) {*?IA* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets borel*} ⊆ *sets N* **.**
qed
show *indep-set* (*sigma-sets* (*space M*) {*X* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets borel*}) (*sets N*)
using *assms* **unfolding** *subalgebra-indep-var-def* **by** *simp*
qed
qed

lemma *fct-gen-subalgebra-cong*:
assumes *space M* = *space P*
and *sets N* = *sets Q*
shows *fct-gen-subalgebra M N X* = *fct-gen-subalgebra P Q X*
proof −
have *space M* = *space P* **using** *assms* **by** *simp*
thus *?thesis* **using** *assms* **unfolding** *fct-gen-subalgebra-def gen-subalgebra-def* **by** *simp*
qed

end

2 Filtrations

This theory introduces basic notions about filtrations, which permit to define adaptable processes and predictable processes in the case where the filtration is indexed by natural numbers.

theory *Filtration* **imports** *HOL−Probability.Probability*
begin

2.1 Basic definitions

class *linorder-bot* = *linorder* + *bot*
instantiation *nat*::*linorder-bot*
begin
instance proof qed
end

definition *filtration* :: *'a measure* ⇒ (*'i*::*linorder-bot* ⇒ *'a measure*) ⇒ *bool* **where**
filtration M F ⟷

$(\forall\, t.\ subalgebra\ M\ (F\ t))\ \wedge$
$(\forall\ s\ t.\ s \leq t \longrightarrow subalgebra\ (F\ t)\ (F\ s))$

lemma *filtrationI*:
assumes $\forall\, t.\ subalgebra\ M\ (F\ t)$
and $\forall\, s\ t.\ s \leq t \longrightarrow subalgebra\ (F\ t)\ (F\ s)$
shows *filtration M F* **unfolding** *filtration-def* **using** *assms* **by** *simp*

lemma *filtrationE1*:
assumes *filtration M F*
shows *subalgebra M* $(F\ t)$ **using** *assms* **unfolding** *filtration-def* **by** *simp*

lemma *filtrationE2*:
assumes *filtration M F*
shows $s \leq t \Longrightarrow$ *subalgebra* $(F\ t)\ (F\ s)$ **using** *assms* **unfolding** *filtration-def* **by** *simp*

locale *filtrated-prob-space* = *prob-space* +
fixes *F*
assumes *filtration*: *filtration M F*

lemma (**in** *filtrated-prob-space*) *filtration-space*:
assumes $s \leq t$
shows *space* $(F\ s)$ = *space* $(F\ t)$ **by** (*metis filtration filtration-def subalgebra-def*)

lemma (**in** *filtrated-prob-space*) *filtration-measurable*:
assumes $f \in$ *measurable* $(F\ t)$ *N*
shows $f \in$ *measurable M N* **unfolding** *measurable-def*
proof
show $f \in space\ M \to space\ N \wedge (\forall\, y \in sets\ N.\ f\ -`\ y \cap space\ M \in sets\ M)$
proof (*intro conjI ballI*)
have *space* $(F\ t)$ = *space M* **using** *assms filtration* **unfolding** *filtration-def subalgebra-def* **by** *auto*
thus $f \in space\ M \to space\ N$ **using** *assms* **unfolding** *measurable-def* **by** *simp*
fix *y*
assume $y \in sets\ N$
hence $f\ -`y \cap space\ M \in sets\ (F\ t)$ **using** *assms* **unfolding** *measurable-def*
using ‹*space* $(F\ t)$ = *space M*› **by** *auto*
thus $f\ -`y \cap space\ M \in sets\ M$ **using** *assms filtration* **unfolding** *filtration-def subalgebra-def* **by** *auto*
qed
qed

lemma (**in** *filtrated-prob-space*) *increasing-measurable-info*:
assumes $f \in$ *measurable* $(F\ s)$ *N*
and $s \leq t$
shows $f \in$ *measurable* $(F\ t)$ *N*
proof (*rule measurableI*)

have *inc*: *sets* (*F s*) ⊆ *sets* (*F t*)
using *assms*(*2*) *filtration* **by** (*simp add*: *filtration-def subalgebra-def*)
have *sp*: *space* (*F s*) = *space* (*F t*) **by** (*metis filtration filtration-def subalgebra-def*)
thus ⋀*x*. *x* ∈ *space* (*F t*) ⟹ *f x* ∈ *space N* **using** *assms* **by** (*simp add*: *measurable-space*)
show ⋀*A*. *A* ∈ *sets N* ⟹ *f* −‘ *A* ∩ *space* (*F t*) ∈ *sets* (*F t*)
proof −
fix *A*
assume *A*∈ *sets N*
hence *f* −‘ *A* ∩ *space* (*F s*) ∈ *sets* (*F s*) **using** *assms* **using** *measurable-sets* **by** *blast*
hence *f* −‘ *A* ∩ *space* (*F s*) ∈ *sets* (*F t*) **using** *subsetD*[*of F s F t*] *inc* **by** *blast*
thus *f* −‘ *A* ∩ *space* (*F t*) ∈ *sets* (*F t*) **using** *sp* **by** *simp*
qed
qed

definition *disc-filtr* :: *'a measure* ⇒ (*nat* ⇒ *'a measure*) ⇒ *bool* **where**
disc-filtr M F ⟷
(∀ *n*. *subalgebra M* (*F n*)) ∧
(∀ *n m*. *n* ≤ *m* ⟶ *subalgebra* (*F m*) (*F n*))

locale *disc-filtr-prob-space* = *prob-space* +
fixes *F*
assumes *discrete-filtration*: *disc-filtr M F*

lemma (**in** *disc-filtr-prob-space*) *subalgebra-filtration*:
assumes *subalgebra N M*
and *filtration M F*
shows *filtration N F*
proof (*rule filtrationI*)
show ∀ *s t*. *s* ≤ *t* ⟶ *subalgebra* (*F t*) (*F s*) **using** *assms* **unfolding** *filtration-def* **by** *simp*
show ∀ *t*. *subalgebra N* (*F t*)
proof
fix *t*
have *subalgebra M* (*F t*) **using** *assms* **unfolding** *filtration-def* **by** *auto*
thus *subalgebra N* (*F t*) **using** *assms* **by** (*metis subalgebra-def subsetCE subsetI*)
qed
qed

sublocale *disc-filtr-prob-space* ⊆ *filtrated-prob-space*
proof *unfold-locales*
show *filtration M F*

using *discrete-filtration* **by** (*simp add*: *filtration-def disc-filtr-def*)
qed

2.2 Stochastic processes

Stochastic processes are collections of measurable functions. Those of a particular interest when there is a filtration are the adapted stochastic processes.

definition *stoch-procs* **where**
stoch-procs M N = {*X*. ∀ *t*. (*X t*) ∈ *measurable M N*}

2.2.1 Adapted stochastic processes

definition *adapt-stoch-proc* **where**
(*adapt-stoch-proc F X N*) ⟷ (∀ *t*. (*X t*) ∈ *measurable* (*F t*) *N*)

abbreviation *borel-adapt-stoch-proc F X* ≡ *adapt-stoch-proc F X borel*

lemma (**in** *filtrated-prob-space*) *adapted-is-dsp*:
assumes *adapt-stoch-proc F X N*
shows *X* ∈ *stoch-procs M N*
unfolding *stoch-procs-def*
by (*intro CollectI*, (*meson adapt-stoch-proc-def assms filtration filtration-def measurable-from-subalg*))

lemma (**in** *filtrated-prob-space*) *adapt-stoch-proc-borel-measurable*:
assumes *adapt-stoch-proc F X N*
shows ∀ *n*. (*X n*) ∈ *measurable M N*
proof
fix *n*
have *X n* ∈ *measurable* (*F n*) *N* **using** *assms* **unfolding** *adapt-stoch-proc-def* **by** *simp*
moreover have *subalgebra M* (*F n*) **using** *filtration* **unfolding** *filtration-def* **by** *simp*
ultimately show *X n* ∈ *measurable M N* **by** (*simp add*:*measurable-from-subalg*)
qed

lemma (**in** *filtrated-prob-space*) *borel-adapt-stoch-proc-borel-measurable*:
assumes *borel-adapt-stoch-proc F X*
shows ∀ *n*. (*X n*) ∈ *borel-measurable M*
proof
fix *n*
have *X n* ∈ *borel-measurable* (*F n*) **using** *assms* **unfolding** *adapt-stoch-proc-def* **by** *simp*
moreover have *subalgebra M* (*F n*) **using** *filtration* **unfolding** *filtration-def* **by** *simp*
ultimately show *X n* ∈ *borel-measurable M* **by** (*simp add*:*measurable-from-subalg*)

qed

lemma (**in** *filtrated-prob-space*) *constant-process-borel-adapted*:
shows *borel-adapt-stoch-proc F* (λ *n w*. *c*)
unfolding *adapt-stoch-proc-def*
proof
fix *t*
show (λ*w*. *c*) ∈ *borel-measurable* (*F t*) **using** *borel-measurable-const* **by** *blast*
qed

lemma (**in** *filtrated-prob-space*) *borel-adapt-stoch-proc-add*:
fixes *X*::*'b* ⇒ *'a* ⇒ (*'c*::{*second-countable-topology*, *topological-monoid-add*})
assumes *borel-adapt-stoch-proc F X*
and *borel-adapt-stoch-proc F Y*
shows *borel-adapt-stoch-proc F* (λ*t w*. *X t w* + *Y t w*) **unfolding** *adapt-stoch-proc-def*
proof
fix *t*
have *X t* ∈ *borel-measurable* (*F t*) **using** *assms* **unfolding** *adapt-stoch-proc-def*
by *simp*
moreover have *Y t* ∈ *borel-measurable* (*F t*) **using** *assms* **unfolding** *adapt-stoch-proc-def*
by *simp*
ultimately show (λ*w*. *X t w* + *Y t w*) ∈ *borel-measurable* (*F t*) **by** *simp*
qed

lemma (**in** *filtrated-prob-space*) *borel-adapt-stoch-proc-sum*:
fixes *A*::*'d* ⇒ *'b* ⇒ *'a* ⇒ (*'c*::{*second-countable-topology*, *topological-comm-monoid-add*})
assumes ⋀*i*. *i*∈ *S* ⟹ *borel-adapt-stoch-proc F* (*A i*)
shows *borel-adapt-stoch-proc F* (λ *t w*. (∑ *i*∈ *S*. *A i t w*)) **unfolding** *adapt-stoch-proc-def*
proof
fix *t*
have ⋀*i*. *i*∈ *S*⟹ *A i t* ∈ *borel-measurable* (*F t*) **using** *assms* **unfolding** *adapt-stoch-proc-def* **by** *simp*
thus (λ *w*. (∑ *i*∈ *S*. *A i t w*)) ∈ *borel-measurable* (*F t*) **by** (*simp add*:*borel-measurable-sum*)
qed

lemma (**in** *filtrated-prob-space*) *borel-adapt-stoch-proc-times*:
fixes *X*::*'b* ⇒ *'a* ⇒ (*'c*::{*second-countable-topology*, *real-normed-algebra*})
assumes *borel-adapt-stoch-proc F X*
and *borel-adapt-stoch-proc F Y*
shows *borel-adapt-stoch-proc F* (λ*t w*. *X t w* ∗ *Y t w*) **unfolding** *adapt-stoch-proc-def*
proof
fix *t*
have *X t* ∈ *borel-measurable* (*F t*) **using** *assms* **unfolding** *adapt-stoch-proc-def*
by *simp*
moreover have *Y t* ∈ *borel-measurable* (*F t*) **using** *assms* **unfolding** *adapt-stoch-proc-def*
by *simp*

ultimately show $(\lambda w.\ X\ t\ w * Y\ t\ w) \in$ *borel-measurable* $(F\ t)$ **by** *simp*
qed

lemma (**in** *filtrated-prob-space*) *borel-adapt-stoch-proc-prod*:
fixes $A::'d \Rightarrow 'b \Rightarrow 'a \Rightarrow ('c::\{$*second-countable-topology*, *real-normed-field*$\})$
assumes $\bigwedge i.\ i \in S \Longrightarrow$ *borel-adapt-stoch-proc* $F\ (A\ i)$
shows *borel-adapt-stoch-proc* $F\ (\lambda\ t\ w.\ (\prod\ i \in S.\ A\ i\ t\ w))$ **unfolding** *adapt-stoch-proc-def*
proof
fix t
have $\bigwedge i.\ i \in S \Longrightarrow A\ i\ t \in$ *borel-measurable* $(F\ t)$ **using** *assms* **unfolding** *adapt-stoch-proc-def* **by** *simp*
thus $(\lambda\ w.\ (\prod\ i \in S.\ A\ i\ t\ w)) \in$ *borel-measurable* $(F\ t)$ **by** *simp*
qed

2.2.2 Predictable stochastic processes

definition *predict-stoch-proc* **where**
$($*predict-stoch-proc* $F\ X\ N) \longleftrightarrow (X\ 0 \in$ *measurable* $(F\ 0)\ N \wedge (\forall\, n.\ (X\ (Suc\ n)) \in$ *measurable* $(F\ n)\ N))$

abbreviation *borel-predict-stoch-proc* $F\ X \equiv$ *predict-stoch-proc* $F\ X$ *borel*

lemma (**in** *disc-filtr-prob-space*) *predict-imp-adapt*:
assumes *predict-stoch-proc* $F\ X\ N$
shows *adapt-stoch-proc* $F\ X\ N$ **unfolding** *adapt-stoch-proc-def*
proof
fix n
show $X\ n \in$ *measurable* $(F\ n)\ N$
proof $(cases\ n = 0)$
case *True*
thus *?thesis* **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *auto*
next
case *False*
thus *?thesis* **using** *assms* **unfolding** *predict-stoch-proc-def*
by (*metis Suc-n-not-le-n increasing-measurable-info nat-le-linear not0-implies-Suc*)
qed
qed

lemma (**in** *disc-filtr-prob-space*) *predictable-is-dsp*:
assumes *predict-stoch-proc* $F\ X\ N$
shows $X \in$ *stoch-procs* $M\ N$
unfolding *stoch-procs-def*
proof
show $\forall\, n.$ *random-variable* $N\ (X\ n)$
proof
fix n
show *random-variable* $N\ (X\ n)$

proof (*cases n=0*)
case *True*
thus *?thesis* **using** *assms* **unfolding** *predict-stoch-proc-def*
using *filtration filtration-def measurable-from-subalg* **by** *blast*
next
case *False*
thus *?thesis* **using** *assms* **unfolding** *predict-stoch-proc-def*
by (*metis filtration filtration-def measurable-from-subalg not0-implies-Suc*)
qed
qed
qed

lemma (**in** *disc-filtr-prob-space*) *borel-predict-stoch-proc-borel-measurable*:
assumes *borel-predict-stoch-proc F X*
shows $\forall n.\ (X\ n) \in$ *borel-measurable M* **using** *assms predictable-is-dsp* **unfolding** *stoch-procs-def* **by** *auto*

lemma (**in** *disc-filtr-prob-space*) *constant-process-borel-predictable*:
shows *borel-predict-stoch-proc F* (λ *n w. c*)
unfolding *predict-stoch-proc-def*
proof
show ($\lambda w.\ c$) $\in$ *borel-measurable* (*F 0*) **using** *borel-measurable-const* **by** *blast*
next
show $\forall n.\ (\lambda w.\ c) \in$ *borel-measurable* (*F n*) **using** *borel-measurable-const* **by** *blast*
qed

lemma (**in** *disc-filtr-prob-space*) *borel-predict-stoch-proc-add*:
fixes *X*::*nat* $\Rightarrow$ *'a* $\Rightarrow$ (*'c*::{*second-countable-topology*, *topological-monoid-add*})
assumes *borel-predict-stoch-proc F X*
and *borel-predict-stoch-proc F Y*
shows *borel-predict-stoch-proc F* ($\lambda t\ w.\ X\ t\ w + Y\ t\ w$) **unfolding** *predict-stoch-proc-def*
proof
show ($\lambda w.\ X\ 0\ w + Y\ 0\ w$) $\in$ *borel-measurable* (*F 0*)
using *assms*(*1*) *assms*(*2*) *borel-measurable-add predict-stoch-proc-def* **by** *blast*
next
show $\forall n.\ (\lambda w.\ X\ (Suc\ n)\ w + Y\ (Suc\ n)\ w) \in$ *borel-measurable* (*F n*)
proof
fix *n*
have *X* (*Suc n*) $\in$ *borel-measurable* (*F n*) **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *simp*
moreover have *Y* (*Suc n*) $\in$ *borel-measurable* (*F n*) **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *simp*
ultimately show ($\lambda w.\ X\ (Suc\ n)\ w + Y\ (Suc\ n)\ w$) $\in$ *borel-measurable* (*F n*) **by** *simp*

qed
qed

lemma (**in** *disc-filtr-prob-space*) *borel-predict-stoch-proc-sum*:
fixes $A::'d \Rightarrow nat \Rightarrow 'a \Rightarrow ('c::\{second\text{-}countable\text{-}topology, topological\text{-}comm\text{-}monoid\text{-}add\})$
assumes $\bigwedge i.\ i \in S \Longrightarrow$ *borel-predict-stoch-proc* $F\ (A\ i)$
shows *borel-predict-stoch-proc* $F\ (\lambda\ t\ w.\ (\sum\ i \in S.\ A\ i\ t\ w))$ **unfolding** *predict-stoch-proc-def*
proof
show $(\lambda w.\ \sum i \in S.\ A\ i\ 0\ w) \in$ *borel-measurable* $(F\ 0)$
proof
have $\bigwedge i.\ i \in S \Longrightarrow A\ i\ 0 \in$ *borel-measurable* $(F\ 0)$ **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *simp*
thus $(\lambda\ w.\ (\sum\ i \in S.\ A\ i\ 0\ w)) \in$ *borel-measurable* $(F\ 0)$ **by** (*simp add*:*borel-measurable-sum*)
qed *simp*
next
show $\forall n.\ (\lambda w.\ \sum i \in S.\ A\ i\ (Suc\ n)\ w) \in$ *borel-measurable* $(F\ n)$
proof
fix *n*
have $\bigwedge i.\ i \in S \Longrightarrow A\ i\ (Suc\ n) \in$ *borel-measurable* $(F\ n)$ **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *simp*
thus $(\lambda\ w.\ (\sum\ i \in S.\ A\ i\ (Suc\ n)\ w)) \in$ *borel-measurable* $(F\ n)$ **by** (*simp add*:*borel-measurable-sum*)
qed
qed

lemma (**in** *disc-filtr-prob-space*) *borel-predict-stoch-proc-times*:
fixes $X::nat \Rightarrow 'a \Rightarrow ('c::\{second\text{-}countable\text{-}topology, real\text{-}normed\text{-}algebra\})$
assumes *borel-predict-stoch-proc F X*
and *borel-predict-stoch-proc F Y*
shows *borel-predict-stoch-proc F* $(\lambda t\ w.\ X\ t\ w * Y\ t\ w)$ **unfolding** *predict-stoch-proc-def*
proof
show $(\lambda w.\ X\ 0\ w * Y\ 0\ w) \in$ *borel-measurable* $(F\ 0)$
proof −
have $X\ 0 \in$ *borel-measurable* $(F\ 0)$ **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *simp*
moreover have $Y\ 0 \in$ *borel-measurable* $(F\ 0)$ **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *simp*
ultimately show $(\lambda w.\ X\ 0\ w * Y\ 0\ w) \in$ *borel-measurable* $(F\ 0)$ **by** *simp*
qed
next
show $\forall n.\ (\lambda w.\ X\ (Suc\ n)\ w * Y\ (Suc\ n)\ w) \in$ *borel-measurable* $(F\ n)$
proof
fix *n*
have $X\ (Suc\ n) \in$ *borel-measurable* $(F\ n)$ **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *simp*

moreover have $Y\ (Suc\ n) \in borel\text{-}measurable\ (F\ n)$ **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *simp*
ultimately show $(\lambda w.\ X\ (Suc\ n)\ w * Y\ (Suc\ n)\ w) \in borel\text{-}measurable\ (F\ n)$ **by** *simp*
qed
qed

lemma (**in** *disc-filtr-prob-space*) *borel-predict-stoch-proc-prod*:
fixes $A::'d \Rightarrow nat \Rightarrow 'a \Rightarrow ('c::\{second\text{-}countable\text{-}topology, real\text{-}normed\text{-}field\})$
assumes $\bigwedge i.\ i \in S \Longrightarrow borel\text{-}predict\text{-}stoch\text{-}proc\ F\ (A\ i)$
shows $borel\text{-}predict\text{-}stoch\text{-}proc\ F\ (\lambda\ t\ w.\ (\prod\ i \in S.\ A\ i\ t\ w))$ **unfolding** *predict-stoch-proc-def*
proof
show $(\lambda w.\ \prod i \in S.\ A\ i\ 0\ w) \in borel\text{-}measurable\ (F\ 0)$
proof −
have $\bigwedge i.\ i \in S \Longrightarrow A\ i\ 0 \in borel\text{-}measurable\ (F\ 0)$ **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *simp*
thus $(\lambda\ w.\ (\prod\ i \in S.\ A\ i\ 0\ w)) \in borel\text{-}measurable\ (F\ 0)$ **by** *simp*
qed
next
show $\forall n.\ (\lambda w.\ \prod i \in S.\ A\ i\ (Suc\ n)\ w) \in borel\text{-}measurable\ (F\ n)$
proof
fix n
have $\bigwedge i.\ i \in S \Longrightarrow A\ i\ (Suc\ n) \in borel\text{-}measurable\ (F\ n)$ **using** *assms* **unfolding** *predict-stoch-proc-def* **by** *simp*
thus $(\lambda\ w.\ (\prod\ i \in S.\ A\ i\ (Suc\ n)\ w)) \in borel\text{-}measurable\ (F\ n)$ **by** *simp*
qed
qed

definition (**in** *prob-space*) *constant-image* **where**
constant-image f = (*if* $\exists\ c::'b::\{t2\text{-}space\}.\ \forall x \in space\ M.\ f\ x = c$ *then*
SOME c. $\forall x \in space\ M.\ f\ x = c$ *else undefined*)

lemma (**in** *prob-space*) *constant-imageI*:
assumes $\exists\ c::'b::\{t2\text{-}space\}.\ \forall x \in space\ M.\ f\ x = c$
shows $\forall x \in space\ M.\ f\ x = (constant\text{-}image\ f)$
proof
fix x
assume $x \in space\ M$
let $?c = SOME\ c.\ \forall x \in space\ M.\ f\ x = c$
have $f\ x = ?c$ **using** $\langle x \in space\ M \rangle$ *someI-ex*[*of* $\lambda c.\ \forall x \in space\ M.\ f\ x = c$] *assms* **by** *blast*
thus $f\ x = (constant\text{-}image\ f)$ **by** (*simp add*: *assms prob-space.constant-image-def prob-space-axioms*)
qed

lemma (**in** *prob-space*) *constant-image-pos*:
assumes $\forall x \in space\ M.\ (0::real) < f\ x$

and $\exists\, c{::}real.\ \forall\, x{\in}\ space\ M.\ f\ x = c$
shows $0 < (constant\text{-}image\ f)$
proof −
{
fix x
assume $x{\in}\ space\ M$
hence $0 < f\ x$ **using** *assms* **by** *simp*
also have $... = constant\text{-}image\ f$ **using** *assms constant-imageI* ‹$x{\in}\ space\ M$›
by *auto*
finally have *?thesis* **.**
}
thus *?thesis* **using** *subprob-not-empty* **by** *auto*
qed

definition *open-except* **where**
$open\text{-}except\ x\ y = (if\ x = y\ then\ \{\}\ else\ SOME\ A.\ open\ A \wedge x{\in}\ A \wedge y{\notin}\ A)$

lemma *open-exceptI*:
assumes $(x{::}'b{::}\{t1\text{-}space\}) \neq y$
shows $open\ (open\text{-}except\ x\ y)$ **and** $x{\in}\ open\text{-}except\ x\ y$ **and** $y{\notin}\ open\text{-}except\ x\ y$
proof−
have *ex*:$\exists\, U.\ open\ U \wedge x \in U \wedge y \notin U$ **using** ‹$x{\neq}\ y$› **by** (*simp add*:*t1-space*)
let $?V = SOME\ A.\ open\ A \wedge x{\in}\ A \wedge y{\notin}\ A$
have *vprop*: $open\ ?V \wedge x \in ?V \wedge y \notin ?V$ **using** *someI-ex*[*of* $\lambda U.\ open\ U \wedge x \in U \wedge y \notin U$] *ex* **by** *blast*
show $open\ (open\text{-}except\ x\ y)$ **by** (*simp add*: *open-except-def vprop*)
show $x{\in}\ open\text{-}except\ x\ y$ **by** (*metis* (*full-types*) *open-except-def vprop*)
show $y{\notin}\ open\text{-}except\ x\ y$ **by** (*metis* (*full-types*) *open-except-def vprop*)
qed

lemma *open-except-set*:
assumes *finite A*
and $(x{::}'b{::}\{t1\text{-}space\}) \notin A$
shows $\exists\, U.\ open\ U \wedge x{\in}\ U \wedge U{\cap}\ A = \{\}$
proof(*intro exI conjI*)
have $\forall\, y{\in}\ A.\ x{\neq}\ y$ **using** *assms* **by** *auto*
let $?U = \bigcap\ y \in A.\ open\text{-}except\ x\ y$
show *open ?U*
proof (*intro open-INT ballI*, (*simp add*: *assms*))
fix y
assume $y{\in}\ A$
show $open\ (open\text{-}except\ x\ y)$ **using** ‹$\forall\, y{\in}\ A.\ x{\neq}\ y$› **by** (*simp add*: ‹$y \in A$›
open-exceptI)
qed
show $x \in (\bigcap y{\in}A.\ open\text{-}except\ x\ y)$
proof
fix y
assume $y{\in}\ A$

show $x \in$*open-except x y* **using** ‹$\forall y \in A.\ x \neq y$› **by** (*simp add*: ‹$y \in A$› *open-exceptI*)
qed
have $\forall y \in A.\ y \notin ?U$ **using** ‹$\forall y \in A.\ x \neq y$› *open-exceptI*(*3*) **by** *auto*
thus $(\bigcap y \in A.$ *open-except x y*$) \cap A = \{\}$ **by** *auto*
qed

definition *open-exclude-set* **where**
open-exclude-set x A = (*if* $(\exists U.$ *open* $U \wedge U \cap A = \{x\})$ *then SOME U. open U* $\wedge$ $U \cap A = \{x\}$ *else* $\{\}$)

lemma *open-exclude-setI*:
assumes $\exists U.$ *open* $U \wedge U \cap A = \{x\}$
shows *open* (*open-exclude-set x A*) **and** (*open-exclude-set x A*) $\cap A = \{x\}$
proof −
let *?V* = *SOME U. open* $U \wedge U \cap A = \{x\}$
have *vprop*: *open ?V* $\wedge$ *?V* $\cap A = \{x\}$ **using** *someI-ex*[*of* $\lambda U.$ *open* $U \wedge U \cap A = \{x\}$] *assms* **by** *blast*
show *open* (*open-exclude-set x A*) **by** (*simp add*: *open-exclude-set-def vprop*)
show *open-exclude-set x A* $\cap A = \{x\}$ **by** (*metis* (*mono-tags*, *lifting*) *open-exclude-set-def vprop*)
qed

lemma *open-exclude-finite*:
assumes *finite A*
and (x::'b::{*t1-space*})$\in A$
shows *open-set*: *open* (*open-exclude-set x A*) **and** *inter-x*:(*open-exclude-set x A*) $\cap$ $A = \{x\}$
proof −
have $\exists U.$ *open* $U \wedge U \cap A = \{x\}$
proof −
have $\exists U.$ *open* $U \wedge x \in U \wedge U \cap (A - \{x\}) = \{\}$
proof (*rule open-except-set*)
show *finite* $(A - \{x\})$ **using** *assms* **by** *auto*
show $x \notin A - \{x\}$ **by** *simp*
qed
thus *?thesis* **using** *assms* **by** *auto*
qed
thus *open* (*open-exclude-set x A*) **and** (*open-exclude-set x A*) $\cap A = \{x\}$ **by** (*auto simp add*: *open-exclude-setI*)
qed

2.3 Initially trivial filtrations

Intuitively, these are filtrations that can be used to denote the fact that there is no information at the start.

definition *init-triv-filt*::'*a measure* ⇒ ('*i*::*linorder-bot* ⇒ '*a measure*) ⇒ *bool* **where**
init-triv-filt M F ⟷ *filtration M F* $\wedge$ *sets* (*F bot*) = {{}, *space M*}

lemma *triv-measurable-cst*:
fixes $f::'a \Rightarrow 'b::\{t2\text{-}space\}$
assumes *space* N = *space* M
and *space* $M \neq \{\}$
and *sets* $N = \{\{\}, space\ M\}$
and $f \in$ *measurable* N *borel*
shows $\exists\ c::'b.\ \forall x \in space\ N.\ f\ x = c$
proof −
have $f\ `(space\ N) \neq \{\}$ **using** *assms* **by** (*simp add*: *assms*)
hence $\exists\ c.\ c \in f`(space\ N)$ **by** *auto*
from *this* **obtain** c **where** $c \in f`(space\ N)$ **by** *auto*
have $\forall x \in space\ N.\ f\ x = c$
proof
fix x
assume $x \in space\ N$
show $f\ x = c$
proof (*rule ccontr*)
assume $f\ x \neq c$
hence ($\exists\ U\ V.$ *open* $U \wedge$ *open* $V \wedge (f\ x) \in U \wedge c \in V \wedge U \cap V = \{\}$) **by** (*simp add*: *separation-t2*)
from *this* **obtain** U **and** V **where** *open* U **and** *open* V **and** $(f\ x) \in U$ **and** $c \in V$ **and** $U \cap V = \{\}$ **by** *blast*
have $(f\ -`V) \cap space\ N = space\ N$
proof −
have $V \in$ *sets borel* **using** ‹*open* V› **unfolding** *borel-def* **by** *simp*
hence $(f\ -`V) \cap space\ N \in sets\ N$ **using** *assms* **unfolding** *measurable-def* **by** *simp*
show $(f\ -`V) \cap space\ N = space\ N$
proof (*rule ccontr*)
assume $(f\ -`V) \cap space\ N \neq space\ N$
hence $(f\ -`V) \cap space\ N = \{\}$ **using** *assms* ‹$(f\ -`V) \cap space\ N \in sets\ N$› **by** *simp*
thus *False* **using** ‹$c \in V$› **using** ‹$c \in f\ `\ space\ N$› **by** *blast*
qed
qed
have $((f-`U) \cap space\ N) \cap ((f-`V) \cap space\ N) = \{\}$ **using** ‹$U \cap V = \{\}$› **by** *auto*
moreover have $(f\ -`U) \cap space\ N \in sets\ N$ **using** *assms* ‹*open* U› **unfolding** *measurable-def* **by** *simp*
ultimately have $(f\ -`U) \cap space\ N = \{\}$ **using** *assms* ‹$(f\ -`V) \cap space\ N = space\ N$› **by** *simp*
thus *False* **using** ‹$f\ x \in U$› ‹$x \in space\ N$› **by** *blast*
qed
qed
thus $\exists\ c.\ \forall x \in space\ N.\ f\ x = c$ **by** *auto*
qed

locale *trivial-init-filtrated-prob-space* = *prob-space* +
fixes F

assumes *info-filtration*: *init-triv-filt M F*

sublocale *trivial-init-filtrated-prob-space* ⊆ *filtrated-prob-space*
using *info-filtration* **unfolding** *init-triv-filt-def* **by** (*unfold-locales*, *simp*)

locale *triv-init-disc-filtr-prob-space* = *prob-space* +
fixes *F*
assumes *info-disc-filtr*: *disc-filtr M F* ∧ *sets* (*F bot*) = {{}, *space M*}

sublocale *triv-init-disc-filtr-prob-space* ⊆ *trivial-init-filtrated-prob-space*
proof *unfold-locales*
show *init-triv-filt M F* **using** *info-disc-filtr bot-nat-def* **unfolding** *init-triv-filt-def disc-filtr-def*
by (*simp add*: *filtrationI*)

qed

sublocale *triv-init-disc-filtr-prob-space* ⊆ *disc-filtr-prob-space*
proof *unfold-locales*
show *disc-filtr M F* **using** *info-disc-filtr* **by** *simp*
qed

lemma (**in** *triv-init-disc-filtr-prob-space*) *adapted-init*:
assumes *borel-adapt-stoch-proc F x*
shows ∃ *c*. ∀ *w* ∈ *space M*. ((*x 0 w*)::*real*) = *c*
proof −
have *space M* = *space* (*F 0*) **using** *filtration*
by (*simp add*: *filtration-def subalgebra-def*)
moreover have ∃ *c*. ∀ *w* ∈ *space* (*F 0*). *x 0 w* = *c*
proof (*rule triv-measurable-cst*)
show *space* (*F 0*) = *space M* **using** ‹*space M* = *space* (*F 0*)› **..**
show *sets* (*F 0*) = {{}, *space M*} **using** *info-disc-filtr*
by (*simp add*: *init-triv-filt-def bot-nat-def*)
show *x 0* ∈ *borel-measurable* (*F 0*) **using** *assms* **by** (*simp add*: *adapt-stoch-proc-def*)
show *space M* ≠ {} **by** (*simp add*:*not-empty*)
qed
ultimately show *?thesis* **by** *simp*
qed

2.4 Filtration-equivalent measure spaces

This is a relaxation of the notion of equivalent probability spaces, where equivalence is tested modulo a filtration. Equivalent measure spaces agree on events that have a zero probability of occurring; here, filtration-equivalent measure spaces agree on such events when they belong to the filtration under consideration.

definition *filt-equiv* **where**
filt-equiv F M N $\longleftrightarrow$ *sets M* = *sets N* $\wedge$ *filtration M F* $\wedge$ ($\forall$ *t A*. *A* $\in$ *sets* (*F t*) $\longrightarrow$ (*emeasure M A* = *0*) $\longleftrightarrow$ (*emeasure N A* = *0*))

lemma *filt-equiv-space*:
assumes *filt-equiv F M N*
shows *space M* = *space N* **using** *assms* **unfolding** *filt-equiv-def filtration-def subalgebra-def* **by** (*meson sets-eq-imp-space-eq*)

lemma *filt-equiv-sets*:
assumes *filt-equiv F M N*
shows *sets M* = *sets N* **using** *assms* **unfolding** *filt-equiv-def* **by** *simp*

lemma *filt-equiv-filtration*:
assumes *filt-equiv F M N*
shows *filtration N F* **using** *assms* **unfolding** *filt-equiv-def filtration-def subalgebra-def*
by (*metis sets-eq-imp-space-eq*)

lemma (**in** *filtrated-prob-space*) *AE-borel-eq*:
fixes *f*::*'a*$\Rightarrow$*real*
assumes *f*$\in$ *borel-measurable* (*F t*)
and *g*$\in$ *borel-measurable* (*F t*)
and *AE w in M*. *f w* = *g w*
shows {*w*$\in$ *space M*. *f w* $\neq$ *g w*} $\in$ *sets* (*F t*) $\wedge$ *emeasure M* {*w*$\in$ *space M*. *f w* $\neq$ *g w*} = *0*
proof
show {*w* $\in$ *space M*. *f w* $\neq$ *g w*} $\in$ *sets* (*F t*)
proof −
define *minus* **where** *minus* = (λ*w*. (*f w*) − (*g w*))
have *minus* $\in$ *borel-measurable* (*F t*) **unfolding** *minus-def* **using** *assms* **by** *simp*
hence {*w*$\in$ *space* (*F t*). *0* < *minus w*} $\in$ *sets* (*F t*) **using** *borel-measurable-iff-greater* **by** *auto*
moreover have {*w*$\in$ *space* (*F t*). *minus w* < *0*} $\in$ *sets* (*F t*) **using** *borel-measurable-iff-less* ‹*minus* $\in$ *borel-measurable* (*F t*)› **by** *auto*
ultimately have {*w*$\in$ *space* (*F t*). *0* < *minus w*} $\cup$ {*w*$\in$ *space* (*F t*). *minus w* < *0*} $\in$ *sets* (*F t*) **by** *simp*
moreover have {*w*$\in$ *space* (*F t*). *f w* $\neq$ *g w*} = {*w*$\in$ *space* (*F t*). *0* < *minus w*} $\cup$ {*w*$\in$ *space* (*F t*). *minus w* < *0*}
proof
show {*w* $\in$ *space* (*F t*). *f w* $\neq$ *g w*} $\subseteq$ {*w* $\in$ *space* (*F t*). *0* < *minus w*} $\cup$ {*w* $\in$ *space* (*F t*). *minus w* < *0*}

proof
fix w
assume $w \in \{w \in space\ (F\ t).\ f\ w \neq g\ w\}$
hence $w \in space\ (F\ t)$ **and** $f\ w \neq g\ w$ **by** *auto*
thus $w \in \{w \in space\ (F\ t).\ 0 < minus\ w\} \cup \{w \in space\ (F\ t).\ minus\ w < 0\}$ **unfolding** *minus-def*
by (*cases f w < g w*) *auto*
qed
have $\{w \in space\ (F\ t).\ 0 < minus\ w\} \subseteq \{w \in space\ (F\ t).\ f\ w \neq g\ w\}$ **unfolding** *minus-def* **by** *auto*
moreover have $\{w \in space\ (F\ t).\ minus\ w < 0\} \subseteq \{w \in space\ (F\ t).\ f\ w \neq g\ w\}$ **unfolding** *minus-def* **by** *auto*
ultimately show $\{w \in space\ (F\ t).\ 0 < minus\ w\} \cup \{w \in space\ (F\ t).\ minus\ w < 0\} \subseteq \{w \in space\ (F\ t).\ f\ w \neq g\ w\}$
by *simp*
qed
moreover have *space* $(F\ t) = space\ M$ **using** *filtration* **unfolding** *filtration-def subalgebra-def* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
show *emeasure* $M\ \{w \in space\ M.\ f\ w \neq g\ w\} = 0$ **by** (*metis* (*no-types*) *AE-iff-measurable assms*(*3*) *emeasure-notin-sets*)
qed

lemma (**in** *prob-space*) *filt-equiv-borel-AE-eq*:
fixes $f :: 'a \Rightarrow real$
assumes *filt-equiv F M N*
and $f \in$ *borel-measurable* $(F\ t)$
and $g \in$ *borel-measurable* $(F\ t)$
and *AE w in M*. $f\ w = g\ w$
shows *AE w in N*. $f\ w = g\ w$
proof −
have *set0*: $\{w \in space\ M.\ f\ w \neq g\ w\} \in sets\ (F\ t) \wedge emeasure\ M\ \{w \in space\ M.\ f\ w \neq g\ w\} = 0$
proof (*rule filtrated-prob-space.AE-borel-eq*, (*auto simp add*: *assms*))
show *filtrated-prob-space M F* **using** *assms* **unfolding** *filt-equiv-def*
by (*simp add*: *filtrated-prob-space-axioms.intro filtrated-prob-space-def prob-space-axioms*)
qed
hence *emeasure N* $\{w \in space\ M.\ f\ w \neq g\ w\} = 0$ **using** *assms* **unfolding** *filt-equiv-def* **by** *auto*
moreover have $\{w \in space\ M.\ f\ w \neq g\ w\} \in sets\ N$ **using** *set0 assms* **unfolding** *filt-equiv-def*
filtration-def subalgebra-def **by** *auto*
ultimately show *?thesis*
proof −
have *space M* = *space N*
by (*metis assms*(*1*) *filt-equiv-space*)
then have $\forall p.$ *almost-everywhere N p* $\vee$ $\{a \in space\ N.\ \neg\ p\ a\} \neq \{a \in space$

$N.\ f\ a \neq g\ a\}$
using *AE-iff-measurable* ‹*emeasure N* $\{w \in$ *space M*. $f\ w \neq g\ w\} = 0$› ‹$\{w \in$ *space M*. $f\ w \neq g\ w\} \in$ *sets N*›
by *auto*
then show *?thesis*
by *metis*
qed
qed

lemma *filt-equiv-prob-space-subalgebra*:
assumes *prob-space N*
and *filt-equiv F M N*
and *sigma-finite-subalgebra M G*
shows *sigma-finite-subalgebra N G* **unfolding** *sigma-finite-subalgebra-def*
proof
show *subalgebra N G*
by (*metis assms*(*2*) *assms*(*3*) *filt-equiv-space filt-equiv-def sigma-finite-subalgebra-def subalgebra-def*)
show *sigma-finite-measure* (*restr-to-subalg N G*) **unfolding** *restr-to-subalg-def*
by (*metis* ‹*subalgebra N G*› *assms*(*1*) *finite-measure-def finite-measure-restr-to-subalg prob-space-def restr-to-subalg-def*)
qed

lemma *filt-equiv-measurable*:
assumes *filt-equiv F M N*
and *f*∈ *measurable M P*
shows *f*∈ *measurable N P* **using** *assms* **unfolding** *filt-equiv-def measurable-def*
proof −
assume *a1*: *sets M* = *sets N* ∧ *Filtration.filtration M F* ∧ (∀ *t A*. *A* ∈ *sets* (*F t*) ⟶ (*emeasure M A* = *0*) = (*emeasure N A* = *0*))
assume *a2*: *f* ∈ {*f* ∈ *space M* → *space P*. ∀ *y*∈*sets P*. *f* −‘ *y* ∩ *space M* ∈ *sets M*}
have *space N* = *space M*
using *a1* **by** (*metis* (*lifting*) *sets-eq-imp-space-eq*)
then show *f* ∈ {*f* ∈ *space N* → *space P*. ∀ *C*∈*sets P*. *f* −‘ *C* ∩ *space N* ∈ *sets N*}
using *a2 a1* **by** *force*
qed

lemma *filt-equiv-imp-subalgebra*:
assumes *filt-equiv F M N*
shows *subalgebra N M* **unfolding** *subalgebra-def*
using *assms filt-equiv-space filt-equiv-def* **by** *blast*

end

3 Martingales

theory *Martingale* **imports** *Filtration*
begin

definition *martingale* **where**
martingale M F X $\longleftrightarrow$
(*filtration M F*) $\wedge$ ($\forall$ *t. integrable M* (*X t*)) $\wedge$ (*borel-adapt-stoch-proc F X*) $\wedge$
($\forall$ *t s. t* $\leq$ *s* $\longrightarrow$ (*AE w in M. real-cond-exp M* (*F t*) (*X s*) *w* = *X t w*))

lemma *martingaleAE*:
assumes *martingale M F X*
and *t* $\leq$ *s*
shows *AE w in M. real-cond-exp M* (*F t*) (*X s*) *w* = (*X t*) *w* **using** *assms* **unfolding** *martingale-def* **by** *simp*

lemma *martingale-add*:
assumes *martingale M F X*
and *martingale M F Y*
and $\forall$ *m. sigma-finite-subalgebra M* (*F m*)
shows *martingale M F* (λ*n w. X n w* + *Y n w*) **unfolding** *martingale-def*
proof (*intro conjI*)
let *?sum* = λ*n w. X n w* + *Y n w*
show $\forall$ *n. integrable M* (λ*w. X n w* + *Y n w*)
proof
fix *n*
show *integrable M* (λ*w. X n w* + *Y n w*)
by (*metis Bochner-Integration.integrable-add assms*(*1*) *assms*(*2*) *martingale-def*)
qed
show $\forall$ *n m. n* $\leq$ *m* $\longrightarrow$ (*AE w in M. real-cond-exp M* (*F n*) (λ*w. X m w* + *Y m w*) *w* = *X n w* + *Y n w*)
proof (*intro allI impI*)
fix *n*::*'b*
fix *m*
assume *n* $\leq$ *m*
show *AE w in M. real-cond-exp M* (*F n*) (λ*w. X m w* + *Y m w*) *w* = *X n w* + *Y n w*
proof −
have *integrable M* (*X m*) **using** *assms* **unfolding** *martingale-def* **by** *simp*
moreover have *integrable M* (*Y m*) **using** *assms* **unfolding** *martingale-def* **by** *simp*
moreover have *sigma-finite-subalgebra M* (*F n*) **using** *assms* **by** *simp*
ultimately have *AE w in M. real-cond-exp M* (*F n*) (λ*w. X m w* + *Y m w*)

```
w =
        real-cond-exp M (F n) (X m) w + real-cond-exp M (F n) (Y m) w
       using sigma-finite-subalgebra.real-cond-exp-add[of M F n X m Y m] by simp
      moreover have AE w in M. real-cond-exp M (F n) (X m) w = X n w using
‹n≤ m› assms
        unfolding martingale-def by simp
      moreover have AE w in M. real-cond-exp M (F n) (Y m) w = Y n w using
‹n≤ m› assms
        unfolding martingale-def by simp
      ultimately show ?thesis by auto
    qed
  qed
  show filtration M F using assms unfolding martingale-def by simp
 show borel-adapt-stoch-proc F (λn w. X n w + Y n w) unfolding adapt-stoch-proc-def
  proof
    fix n
    show (λw. X n w + Y n w) ∈ borel-measurable (F n) using assms unfolding
martingale-def adapt-stoch-proc-def
      by (simp add: borel-measurable-add)
  qed
qed

lemma disc-martingale-charact:
  assumes (∀ n. integrable M (X n))
  and filtration M F
  and ∀ m. sigma-finite-subalgebra M (F m)
  and ∀ m. X m ∈ borel-measurable (F m)
  and (∀ n. AE w in M. real-cond-exp M (F n) (X (Suc n)) w = (X n) w)
shows martingale M F X  unfolding martingale-def
proof (intro conjI)
  have ∀ k m. k ≤ m ⟶ (AE w in M. real-cond-exp M (F (m−k)) (X m) w =
X (m−k) w)
  proof (intro allI impI)
    fix m
    fix k::nat
    show k≤m ⟹ AE w in M. real-cond-exp M (F (m−k)) (X m) w = X (m−k)
w
    proof (induct k)
      case 0
      have X m ∈ borel-measurable (F m) using assms by simp
      moreover have integrable M (X m) using assms by simp
      moreover have sigma-finite-subalgebra M (F m) using assms by simp
      ultimately have AE w in M. real-cond-exp M (F m) (X m) w = X m w
       using sigma-finite-subalgebra.real-cond-exp-F-meas[of M F m X m] by simp
      thus ?case using 0 by simp
    next
      case (Suc k)
      have Suc (m − (Suc k)) = m − k using Suc by simp
      hence AE w in M. real-cond-exp M (F (m − (Suc k))) (X (Suc (m − (Suc
```

```
k)))) w = (X (m − (Suc k))) w
        using assms by blast
      hence AE w in M. real-cond-exp M (F (m − (Suc k))) (X ((m − k))) w =
(X (m − (Suc k))) w
        using assms(3) ‹Suc (m − (Suc k)) = m − k› by simp
     moreover have AE w in M. real-cond-exp M (F (m − (Suc k))) (real-cond-exp
M (F (m − k)) (X m)) w =
        real-cond-exp M (F (m − (Suc k))) (X m) w
        using sigma-finite-subalgebra.real-cond-exp-nested-subalg[of M F (m− (Suc
k)) F (m−k) X m]
        by (metis Filtration.filtration-def Suc-n-not-le-n ‹Suc (m − Suc k) = m −
k› assms(1) assms(2) assms(3)
            filtrationE1 nat-le-linear)
     moreover have AE w in M. real-cond-exp M (F (m − (Suc k))) (real-cond-exp
M (F (m − k)) (X m)) w =
        real-cond-exp M (F (m − (Suc k))) (X (m−k)) w using Suc
       sigma-finite-subalgebra.real-cond-exp-cong[of M F (m − (Suc k)) real-cond-exp
M (F (m − k)) (X m) X (m − k)]
        borel-measurable-cond-exp[of M F (m−k) X m]
        using Suc-leD assms(1) assms(3) borel-measurable-cond-exp2 by blast
      ultimately show ?case by auto
    qed
  qed
  thus ∀ n m. n ≤ m ⟶ (AE w in M. real-cond-exp M (F n) (X m) w = X n w)
    by (metis diff-diff-cancel diff-le-self)
  show ∀ t. integrable M (X t) using assms by simp
  show filtration M F using assms by simp
   show borel-adapt-stoch-proc F X using assms unfolding adapt-stoch-proc-def
by simp
qed


lemma (in finite-measure) constant-martingale:
  assumes ∀ t. sigma-finite-subalgebra M (F t)
and filtration M F
shows martingale M F (λn w. c) unfolding martingale-def
proof (intro allI conjI impI)
  show filtration M F using assms by simp
  {
    fix t
    show integrable M (λw. c) by simp
  }
  {
    fix t::'b
    fix s
    assume t ≤ s
    show AE w in M. real-cond-exp M (F t) (λw. c) w = c
     by (intro sigma-finite-subalgebra.real-cond-exp-F-meas, (auto simp add: assms))
  }
```

show *borel-adapt-stoch-proc F* (*λn w. c*) **unfolding** *adapt-stoch-proc-def* **by** *simp*
qed

end

4 Discrete Conditional Expectation

theory *Disc-Cond-Expect* **imports** *HOL−Probability.Probability Generated-Subalgebra*

begin

4.1 Preliminary measurability results

These are some useful results, in particular when working with functions that have a countable codomain.

definition *disc-fct* **where**
disc-fct f ≡ *countable* (*range f*)

definition *point-measurable* **where**
point-measurable M S f ≡ (*f*‘(*space M*)⊆ *S*) ∧ (∀ *r* ∈ (*range f*) ∩ *S* . *f*−‘{*r*} ∩ (*space M*) ∈ *sets M*)

lemma *singl-meas-if*:
assumes *f* ∈ *space M* → *space N*
and ∀ *r*∈ *range f*∩ *space N*. ∃ *A*∈ *sets N*. *range f*∩ *A* = {*r*}
shows *point-measurable* (*fct-gen-subalgebra M N f*) (*space N*) *f* **unfolding** *point-measurable-def*
proof
show *f*‘*space* (*fct-gen-subalgebra M N f*)⊆ *space N* **using** *assms*
by (*simp add*: *Pi-iff fct-gen-subalgebra-space image-subsetI*)
show (∀ *r*∈*range f* ∩ *space N*. *f* −‘ {*r*} ∩ *space* (*fct-gen-subalgebra M N f*) ∈ *sets* (*fct-gen-subalgebra M N f*))
proof
fix *r*
assume *r*∈ *range f* ∩ *space N*
hence ∃ *A*∈ *sets N*. *range f*∩ *A* = {*r*} **using** *assms* **by** *blast*
from *this* **obtain** *A* **where** *A*∈ *sets N* **and** *range f* ∩ *A* = {*r*} **by** *auto* **note** *Aprops* = *this*
hence *f*−‘*A* = *f*−‘{*r*} **by** *auto*
hence *f*−‘*A* ∩ *space M* = *f*−‘{*r*} ∩ *space* (*fct-gen-subalgebra M N f*) **by** (*simp add*: *fct-gen-subalgebra-space*)
thus *f* −‘ {*r*} ∩ *space* (*fct-gen-subalgebra M N f*) ∈ *sets* (*fct-gen-subalgebra M N f*)
using *Aprops fct-gen-subalgebra-sets-mem*[*of A N f M*] **by** *simp*
qed
qed

lemma *meas-single-meas*:
assumes $f \in$ *measurable M N*
and $\forall r \in$ *range* $f \cap$ *space N*. $\exists A \in$ *sets N*. *range* $f \cap A = \{r\}$
shows *point-measurable M* (*space N*) *f*
proof −
have *subalgebra M* (*fct-gen-subalgebra M N f*) **using** *assms fct-gen-subalgebra-is-subalgebra* **by** *blast*
hence *sets* (*fct-gen-subalgebra M N f*) ⊆ *sets M* **by** (*simp add*: *subalgebra-def*)
moreover have *point-measurable* (*fct-gen-subalgebra M N f*) (*space N*) *f* **using** *assms singl-meas-if*
by (*metis* (*no-types*, *lifting*) *Pi-iff measurable-space*)
ultimately show *?thesis*
proof −
obtain *bb* :: *'a measure* ⇒ *'b set* ⇒ (*'a* ⇒ *'b*) ⇒ *'b* **where**
f1: ∀ *m B f*. (¬ *point-measurable m B f* ∨ *f* ‘ *space m* ⊆ *B* ∧ (∀ *b*. *b* ∉ *range f* ∩ *B* ∨ *f* −‘ {*b*} ∩ *space m* ∈ *sets m*)) ∧ (¬ *f* ‘ *space m* ⊆ *B* ∨ *bb m B f* ∈ *range f* ∩ *B* ∧ *f* −‘ {*bb m B f*} ∩ *space m* ∉ *sets m* ∨ *point-measurable m B f*)
by (*metis* (*no-types*) *point-measurable-def*)
moreover
{ assume *f* −‘ {*bb M* (*space N*) *f*} ∩ *space* (*fct-gen-subalgebra M N f*) ∈ *sets* (*fct-gen-subalgebra M N f*)
then have *f* −‘ {*bb M* (*space N*) *f*} ∩ *space M* ∈ *sets* (*fct-gen-subalgebra M N f*)
by (*metis* ‹*subalgebra M* (*fct-gen-subalgebra M N f*)› *subalgebra-def*)
then have *f* −‘ {*bb M* (*space N*) *f*} ∩ *space M* ∈ *sets M*
using ‹*sets* (*fct-gen-subalgebra M N f*) ⊆ *sets M*› **by** *blast*
then have *f* ‘ *space M* ⊆ *space N* ∧ *f* −‘ {*bb M* (*space N*) *f*} ∩ *space M* ∈ *sets M*
using *f1* **by** (*metis* ‹*point-measurable* (*fct-gen-subalgebra M N f*) (*space N*) *f*› ‹*subalgebra M* (*fct-gen-subalgebra M N f*)› *subalgebra-def*)
then have *?thesis*
using *f1* **by** *metis* **}**
ultimately show *?thesis*
by (*metis* (*no-types*) ‹*point-measurable* (*fct-gen-subalgebra M N f*) (*space N*) *f*› ‹*subalgebra M* (*fct-gen-subalgebra M N f*)› *subalgebra-def*)
qed
qed

definition *countable-preimages* **where**
countable-preimages B Y = (λ*n*. *if* ((*infinite B*) ∨ (*finite B* ∧ *n* < *card B*)) *then* *Y* −‘ {(*from-nat-into B*) *n*} *else* {})

lemma *count-pre-disj*:
fixes *i*::*nat*
assumes *countable B*
and *i* ≠ *j*

shows (*countable-preimages B Y*) *i* ∩ (*countable-preimages B Y*) *j* = {}
proof (*cases* (*countable-preimages B Y*) *i* = {} ∨ (*countable-preimages B Y*) *j* = {})
case *True*
thus *?thesis* **by** *auto*
next
case *False*
hence *Y* −‘ {(*from-nat-into B*) *i*} ≠ {} ∧ *Y* −‘ {(*from-nat-into B*) *j*} ≠ {}
unfolding *countable-preimages-def* **by** *meson*
have (*infinite B*) ∨ (*finite B* ∧ *i* < *card B* ∧ *j* < *card B*) **using** *False* **unfolding** *countable-preimages-def*
by *meson*
have (*from-nat-into B*) *i* ≠ (*from-nat-into B*) *j*
by (*metis False assms*(*1*) *assms*(*2*) *bij-betw-def countable-preimages-def from-nat-into-inj from-nat-into-inj-infinite lessThan-iff to-nat-on-finite*)
thus *?thesis*
proof −
have *f1*: ∀ *A f n*. *if infinite A* ∨ *finite A* ∧ *n* < *card A then countable-preimages A f n* = *f* −‘ {*from-nat-into A n*::*'a*} *else countable-preimages A f n* = ({}::*'b set*)
by (*meson countable-preimages-def*)
then have *f2*: *infinite B* ∨ *finite B* ∧ *i* < *card B*
by (*metis* (*no-types*) *False*)
have *infinite B* ∨ *finite B* ∧ *j* < *card B*
using *f1* **by** (*meson False*)
then show *?thesis*
using *f2 f1* ‹*from-nat-into B i* ≠ *from-nat-into B j*› **by** *fastforce*
qed
qed

lemma *count-pre-surj*:
assumes *countable B*
and *w* ∈ *Y* −‘*B*
shows ∃ *i*. *w* ∈ (*countable-preimages B Y*) *i*
proof (*cases finite B*)
case *True*
have ∃ *i* < *card B*. (*from-nat-into B*) *i* = *Y w*
by (*metis True assms*(*1*) *assms*(*2*) *bij-betw-def from-nat-into-to-nat-on image-eqI lessThan-iff*
to-nat-on-finite vimageE)
from *this* **obtain** *i* **where** *i*< *card B* **and** (*from-nat-into B*) *i* = *Y w* **by** *blast*
hence *w* ∈ (*countable-preimages B Y*) *i*
by (*simp add*: *countable-preimages-def*)
thus ∃ *i*. *w* ∈ (*countable-preimages B Y*) *i* **by** *auto*
next
case *False*
hence ∃ *i*. (*from-nat-into B*) *i* = *Y w*
by (*meson assms*(*1*) *assms*(*2*) *from-nat-into-to-nat-on vimageE*)
from *this* **obtain** *i* **where** (*from-nat-into B*) *i* = *Y w* **by** *blast*
hence *w* ∈ (*countable-preimages B Y*) *i*

by (*simp add*: *False countable-preimages-def*)
thus ∃ *i*. *w* ∈ (*countable-preimages B Y*) *i* **by** *auto*
qed

lemma *count-pre-img*:
assumes *x* ∈ (*countable-preimages B Y*) *n*
shows *Y x* = (*from-nat-into B*) *n*
proof −
have *x*∈ *Y* −‘ {(*from-nat-into B*) *n*} **using** *assms* **unfolding** *countable-preimages-def*
by (*meson empty-iff*)
thus *?thesis* **by** *simp*
qed

lemma *count-pre-union-img*:
assumes *countable B*
shows *Y* −‘*B* = (⋃ *i*. (*countable-preimages B Y*) *i*)
proof (*cases B* = {})
case *False*
have *Y* −‘*B* ⊆ (⋃ *i*. (*countable-preimages B Y*) *i*)
by (*simp add*: *assms count-pre-surj subset-eq*)
moreover have (⋃ *i*. (*countable-preimages B Y*) *i*) ⊆ *Y* −‘*B*
proof −
have *f1*: ∀ *b A f n*. (*b*::′*b*) ∉ *countable-preimages A f n* ∨ (*f b*::′*a*) = *from-nat-into A n*
by (*meson count-pre-img*)
have *range* (*from-nat-into B*) = *B*
by (*meson False assms range-from-nat-into*)
then show *?thesis*
using *f1* **by** *blast*
qed
ultimately show *?thesis* **by** *simp*
next
case *True*
hence ∀ *i*. (*countable-preimages B Y*) *i* = {} **unfolding** *countable-preimages-def* **by** *simp*
hence (⋃ *i*. (*countable-preimages B Y*) *i*) = {} **by** *auto*
moreover have *Y* −‘*B* = {} **using** *True* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *count-pre-meas*:
assumes *point-measurable M* (*space N*) *Y*
and *B*⊆ *space N*
and *countable B*
shows ∀ *i*. (*countable-preimages B Y*) *i* ∩ *space M* ∈ *sets M*
proof
fix *i*

have $Y -`B = (\bigcup\ i.\ (countable\text{-}preimages\ B\ Y)\ i)$ **using** *assms*
by (*simp add*: *count-pre-union-img*)
show *countable-preimages B Y i* $\cap$ *space M* $\in$ *sets M*
proof (*cases countable-preimages B Y i* $= \{\}$)
case *True*
thus *?thesis* **by** *simp*
next
case *False*
from *this* **obtain** *y* **where** $y \in$ *countable-preimages B Y i* **by** *auto*
hence *countable-preimages B Y i* $= Y -`\{Y\ y\}$
by (*metis False count-pre-img countable-preimages-def*)
have $Y\ y =$ *from-nat-into B i*
by (*meson* ‹*y* $\in$ *countable-preimages B Y i*› *count-pre-img*)
hence $Y\ y \in$ *space N*
by (*metis UNIV-I UN-I* ‹*y* $\in$ *countable-preimages B Y i*› ‹$Y -`B = (\bigcup$ *i*. (*countable-preimages B Y*) *i*)› *assms*(*2*) *empty-iff from-nat-into subsetCE vimage-empty*)
moreover have $Y\ y \in$ *range Y* **by** *simp*
thus *?thesis*
by (*metis IntI* ‹*countable-preimages B Y i* $= Y -`\{Y\ y\}$› *assms*(*1*) *calculation point-measurable-def*)
qed
qed

lemma *disct-fct-point-measurable*:
assumes *disc-fct f*
and *point-measurable M* (*space N*) *f*
shows *f*$\in$ *measurable M N* **unfolding** *measurable-def*
proof
show *f* $\in$ *space M* $\to$ *space N* $\wedge$ ($\forall$ *y*$\in$*sets N*. *f* $-`$ *y* $\cap$ *space M* $\in$ *sets M*)
proof
show *f* $\in$ *space M* $\to$ *space N* **using** *assms* **unfolding** *point-measurable-def* **by** *auto*
show $\forall$ *y*$\in$*sets N*. *f* $-`$ *y* $\cap$ *space M* $\in$ *sets M*
proof
fix *y*
assume *y*$\in$ *sets N*
let *?imY* $=$ *range f* $\cap$ *y*
have *f*$-`y = f-`?imY$ **by** *auto*
moreover have *countable ?imY* **using** *assms* **unfolding** *disc-fct-def* **by** *auto*
ultimately have $f -`y = (\bigcup\ i.\ (countable\text{-}preimages\ ?imY\ f)\ i)$ **using** *assms count-pre-union-img* **by** *metis*
hence *yeq*: $f -`\ y \cap$ *space M* $= (\bigcup\ i.\ ((countable\text{-}preimages\ ?imY\ f)\ i) \cap$ *space M*) **by** *auto*
have $\forall$ *i*. *countable-preimages ?imY f i* $\cap$ *space M* $\in$ *sets M*
by (*metis* ‹*countable* (*range f* $\cap$ *y*)› ‹*y* $\in$ *sets N*› *assms*(*2*) *inf-le2 le-inf-iff count-pre-meas sets.Int-space-eq1*)
hence $(\bigcup\ i.\ ((countable\text{-}preimages\ ?imY\ f)\ i) \cap$ *space M*$) \in$ *sets M* **by** *blast*

thus $f -` y \cap space\ M \in sets\ M$ **using** *yeq* **by** *simp*
qed
qed
qed

lemma *set-point-measurable*:
assumes *point-measurable M* (*space N*) *Y*
and $B \subseteq space\ N$
and *countable B*
shows $(Y -` B) \cap space\ M \in sets\ M$
proof −
have $Y -` B = (\bigcup\ i.\ (countable\text{-}preimages\ B\ Y)\ i)$ **using** *assms*
by (*simp add*: *count-pre-union-img*)
hence $Y -` B \cap space\ M = (\bigcup\ i.\ ((countable\text{-}preimages\ B\ Y)\ i \cap space\ M))$
by *auto*
have $\forall i.\ (countable\text{-}preimages\ B\ Y)\ i \cap space\ M \in sets\ M$ **using** *assms* **by** (*simp add*: *count-pre-meas*)
hence $(\bigcup\ i.\ ((countable\text{-}preimages\ B\ Y)\ i \cap space\ M)) \in sets\ M$ **by** *blast*
show *?thesis*
using ‹$(\bigcup i.\ countable\text{-}preimages\ B\ Y\ i \cap space\ M) \in sets\ M$› ‹$Y -` B \cap space\ M = (\bigcup i.\ countable\text{-}preimages\ B\ Y\ i \cap space\ M)$› **by** *auto*
qed

4.2 Definition of explicit conditional expectation

This section is devoted to an explicit computation of a conditional expectation for random variables that have a countable codomain. More precisely, the computed random variable is almost everywhere equal to a conditional expectation of the random variable under consideration.

definition *img-dce* **where**
img-dce M Y X = (λ *y*. *if measure M* ((*Y* −` {*y*}) ∩ *space M*) = *0 then 0 else*
((*integral*^{L} *M* (λ*w*. ((*X w*) ∗ (*indicator* ((*Y* −`{*y*})∩ *space M*) *w*))))/(*measure M* ((*Y* −` {*y*}) ∩ *space M*))))

definition *expl-cond-expect* **where**
expl-cond-expect M Y X = (*img-dce M Y X*) ∘ *Y*

lemma *nn-expl-cond-expect-pos*:
assumes $\forall\ w \in space\ M.\ 0 \leq X\ w$
shows $\forall\ w \in space\ M.\ 0 \leq (expl\text{-}cond\text{-}expect\ M\ Y\ X)\ w$
proof
fix *w*
assume *space*: *w*∈ *space M*
show $0 \leq (expl\text{-}cond\text{-}expect\ M\ Y\ X)\ w$
proof (*cases measure M* ((*Y* −` {*Y w*})∩ *space M*) = *0*)
case *True*
thus $0 \leq (expl\text{-}cond\text{-}expect\ M\ Y\ X)\ w$ **unfolding** *expl-cond-expect-def img-dce-def*

by *simp*
next
case *False*
hence $Y -`\{Y\ w\} \cap$ *space* $M \in$ *sets* M **using** *measure-notin-sets* **by** *blast*
let *?indA* $= ((\lambda\ x.$ *indicator* $((Y -`\{Y\ w\}) \cap$ *space* $M)\ x))$
have $\forall\ w \in$ *space* $M.\ 0 \leq (X\ w) * ($*?indA* $w)$ **by** (*simp add*: *assms*)
hence $0 \leq ($*integral*$^L\ M\ (\lambda w.\ ((X\ w) * ($*?indA* $w))))$ **by** *simp*
moreover have (*expl-cond-expect* $M\ Y\ X)\ w = ($*integral*$^L\ M\ (\lambda w.\ ((X\ w) *$ (*?indA* $w)))) /$ (*measure* $M\ ((Y -`\{Y\ w\}) \cap$ *space* $M))$
unfolding *expl-cond-expect-def img-dce-def* **using** *False* **by** *simp*
moreover have $0 <$ *measure* $M\ ((Y -`\{Y\ w\}) \cap$ *space* $M)$ **using** *False* **by** (*simp add*: *zero-less-measure-iff*)
ultimately show $0 \leq$ (*expl-cond-expect* $M\ Y\ X)\ w$ **by** *simp*
qed
qed

lemma *expl-cond-expect-const*:
assumes $Y\ w = Y\ y$
shows *expl-cond-expect* $M\ Y\ X\ w =$ *expl-cond-expect* $M\ Y\ X\ y$
unfolding *expl-cond-expect-def img-dce-def*
by (*simp add*: *assms*)

lemma *expl-cond-exp-cong*:
assumes $\forall\ w \in$ *space* $M.\ X\ w = Z\ w$
shows $\forall\ w \in$ *space* $M.$ *expl-cond-expect* $M\ Y\ X\ w =$ *expl-cond-expect* $M\ Y\ Z\ w$
unfolding *expl-cond-expect-def img-dce-def*
by (*metis* (*no-types*, *lifting*) *Bochner-Integration.integral-cong assms*(*1*) *o-apply*)

lemma *expl-cond-exp-add*:
assumes *integrable* $M\ X$
and *integrable* $M\ Z$
shows $\forall\ w \in$ *space* $M.$ *expl-cond-expect* $M\ Y\ (\lambda x.\ X\ x + Z\ x)\ w =$ *expl-cond-expect* $M\ Y\ X\ w +$ *expl-cond-expect* $M\ Y\ Z\ w$
proof
fix w
assume $w \in$ *space* M
define prY **where** $prY =$ *measure* $M\ ((Y -`\{Y\ w\}) \cap$ *space* $M)$
show *expl-cond-expect* $M\ Y\ (\lambda x.\ X\ x + Z\ x)\ w =$ *expl-cond-expect* $M\ Y\ X\ w +$ *expl-cond-expect* $M\ Y\ Z\ w$
proof (*cases* $prY = 0$)
case *True*
thus *?thesis* **unfolding** *expl-cond-expect-def img-dce-def prY-def* **by** *simp*
next
case *False*
hence $(Y -`\{Y\ w\}) \cap$ *space* $M \in$ *sets* M **unfolding** *prY-def* **using** *mea-*

sure-notin-sets **by** *blast*
let *?indA* = *indicator* ((*Y* −‘ {*Y w*}) ∩ *space M*)::(*'a*⇒*real*)
have *integrable M* (λ*x*. *X x* ∗ *?indA x*)
using ‹*Y* −‘ {*Y w*} ∩ *space M* ∈ *sets M*› *assms*(*1*) *integrable-real-mult-indicator*
by *blast*
moreover have *integrable M* (λ*x*. *Z x* ∗ *?indA x*)
using ‹*Y* −‘ {*Y w*} ∩ *space M* ∈ *sets M*› *assms*(*2*) *integrable-real-mult-indicator*
by *blast*
ultimately have $integral^L$ *M* (λ*x*. *X x* ∗ *?indA x* + *Z x* ∗ *?indA x*) = $integral^L$ *M* (λ*x*. *X x* ∗ *?indA x*) + $integral^L$ *M* (λ*x*. *Z x* ∗ *?indA x*)
using *Bochner-Integration.integral-add* **by** *blast*
moreover have ∀ *x*∈ *space M*. *X x* ∗ *?indA x* + *Z x* ∗ *?indA x* = (*X x* + *Z x*) ∗ *?indA x*
by (*simp add*: *indicator-def*)
ultimately have *fsteq*: $integral^L$ *M* (λ*x*. (*X x* + *Z x*) ∗ *?indA x*) = $integral^L$ *M* (λ*x*. *X x* ∗ *?indA x*) + $integral^L$ *M* (λ*x*. *Z x* ∗ *?indA x*)
by (*metis* (*no-types*, *lifting*) *Bochner-Integration.integral-cong*)
have $integral^L$ *M* (λ*x*. (*X x* + *Z x*) ∗ *?indA x*/*prY*) = $integral^L$ *M* (λ*x*. (*X x* + *Z x*) ∗ *?indA x*)/*prY*
by *simp*
also have ... = $integral^L$ *M* (λ*x*. *X x* ∗ *?indA x*)/*prY* + $integral^L$ *M* (λ*x*. *Z x* ∗ *?indA x*)/*prY* **using** *fsteq*
by (*simp add*: *add-divide-distrib*)
also have ... = $integral^L$ *M* (λ*x*. *X x* ∗ *?indA x*/*prY*) + $integral^L$ *M* (λ*x*. *Z x* ∗ *?indA x*/*prY*) **by** *auto*
finally have $integral^L$ *M* (λ*x*. (*X x* + *Z x*) ∗ *?indA x*/*prY*) = $integral^L$ *M* (λ*x*. *X x* ∗ *?indA x*/*prY*) + $integral^L$ *M* (λ*x*. *Z x* ∗ *?indA x*/*prY*) **.**
thus *?thesis* **using** *False* **unfolding** *expl-cond-expect-def img-dce-def*
by (*simp add*: *add-divide-distrib fsteq*)
qed
qed

lemma *expl-cond-exp-diff*:
assumes *integrable M X*
and *integrable M Z*
shows ∀ *w*∈ *space M*. *expl-cond-expect M Y* (λ*x*. *X x* − *Z x*) *w* = *expl-cond-expect M Y X w* − *expl-cond-expect M Y Z w*
proof
fix *w*
assume *w*∈ *space M*
define *prY* **where** *prY* = *measure M* ((*Y* −‘ {*Y w*}) ∩ *space M*)
show *expl-cond-expect M Y* (λ*x*. *X x* − *Z x*) *w* = *expl-cond-expect M Y X w* − *expl-cond-expect M Y Z w*
proof (*cases prY* = *0*)
case *True*
thus *?thesis* **unfolding** *expl-cond-expect-def img-dce-def prY-def* **by** *simp*
next
case *False*

hence $(Y -` \{Y\ w\}) \cap$ *space* $M \in$ *sets* M **unfolding** *prY-def* **using** *measure-notin-sets* **by** *blast*
let *?indA* = *indicator* $((Y -` \{Y\ w\}) \cap$ *space* $M)::('a \Rightarrow real)$
have *integrable* M $(\lambda x.\ X\ x * ?indA\ x)$
using ‹$Y -` \{Y\ w\} \cap$ *space* $M \in$ *sets* M› *assms*(*1*) *integrable-real-mult-indicator*
by *blast*
moreover have *integrable* M $(\lambda x.\ Z\ x * ?indA\ x)$
using ‹$Y -` \{Y\ w\} \cap$ *space* $M \in$ *sets* M› *assms*(*2*) *integrable-real-mult-indicator*
by *blast*
ultimately have $integral^L\ M\ (\lambda x.\ X\ x * ?indA\ x - Z\ x * ?indA\ x) = integral^L\ M\ (\lambda x.\ X\ x * ?indA\ x) - integral^L\ M\ (\lambda x.\ Z\ x * ?indA\ x)$
using *Bochner-Integration.integral-diff* **by** *blast*
moreover have $\forall x \in$ *space* $M.\ X\ x * ?indA\ x - Z\ x * ?indA\ x = (X\ x - Z\ x) * ?indA\ x$
by (*simp add*: *indicator-def*)
ultimately have *fsteq*: $integral^L\ M\ (\lambda x.\ (X\ x - Z\ x) * ?indA\ x) = integral^L\ M\ (\lambda x.\ X\ x * ?indA\ x) - integral^L\ M\ (\lambda x.\ Z\ x * ?indA\ x)$
by (*metis* (*no-types*, *lifting*) *Bochner-Integration.integral-cong*)
have $integral^L\ M\ (\lambda x.\ (X\ x - Z\ x) * ?indA\ x/prY) = integral^L\ M\ (\lambda x.\ (X\ x - Z\ x) * ?indA\ x)/prY$
by *simp*
also have $... = integral^L\ M\ (\lambda x.\ X\ x * ?indA\ x)/prY - integral^L\ M\ (\lambda x.\ Z\ x * ?indA\ x)/prY$ **using** *fsteq*
by (*simp add*: *diff-divide-distrib*)
also have $... = integral^L\ M\ (\lambda x.\ X\ x * ?indA\ x/prY) - integral^L\ M\ (\lambda x.\ Z\ x * ?indA\ x/prY)$ **by** *auto*
finally have $integral^L\ M\ (\lambda x.\ (X\ x - Z\ x) * ?indA\ x/prY) = integral^L\ M\ (\lambda x.\ X\ x * ?indA\ x/prY) - integral^L\ M\ (\lambda x.\ Z\ x * ?indA\ x/prY)$ **.**
thus *?thesis* **using** *False* **unfolding** *expl-cond-expect-def img-dce-def*
by (*simp add*: *diff-divide-distrib fsteq*)
qed
qed

lemma *expl-cond-expect-prop-sets*:
assumes *disc-fct* Y
and *point-measurable* M (*space* N) Y
and $D = \{w \in$ *space* $M.\ Y\ w \in$ *space* $N \wedge (P$ (*expl-cond-expect* $M\ Y\ X\ w))\}$
shows $D \in$ *sets* M
proof −
let *?C* = $\{y \in (Y`(space\ M)) \cap (space\ N).\ P\ (img\text{-}dce\ M\ Y\ X\ y)\}$
have *space* $M \subseteq$ *UNIV* **by** *simp*
hence $Y`(space\ M) \subseteq range\ Y$ **by** *auto*
hence *countable* $(Y`(space\ M))$ **using** *assms countable-subset* **unfolding** *disc-fct-def*
by *auto*
hence *countable ?C* **using** *assms* **unfolding** *disc-fct-def* **by** *auto*
have *eqset*: $D = (\bigcup\ b \in ?C.\ Y -` \{b\}) \cap$ *space* M
proof
show $D \subseteq (\bigcup\ b \in ?C.\ Y -` \{b\}) \cap$ *space* M
proof

fix w
assume $w\in D$
hence $w\in$ *space* $M \wedge Y\ w \in$ (*space* N) $\wedge$ (P (*expl-cond-expect* $M\ Y\ X\ w$))
by (*simp add*: *assms*)
hence P (*img-dce* $M\ Y\ X$ ($Y\ w$)) **by** (*simp add*: *expl-cond-expect-def*)
hence $Y\ w \in ?C$ **using** ‹$w \in$ *space* $M \wedge Y\ w \in$ *space* $N \wedge P$ (*expl-cond-expect* $M\ Y\ X\ w$)› **by** *blast*
thus $w\in$ ($\bigcup\ b\in ?C.\ Y-`\{b\}$)$\cap$ *space* M
using ‹$w \in$ *space* $M \wedge Y\ w \in$ *space* $N \wedge P$ (*expl-cond-expect* $M\ Y\ X\ w$)› **by** *blast*
qed
show ($\bigcup\ b\in ?C.\ Y-`\{b\}$)$\cap$ *space* $M \subseteq D$
proof
fix w
assume $w\in$ ($\bigcup\ b\in ?C.\ Y-`\{b\}$)$\cap$ *space* M
from *this* **obtain** b **where** $b\in ?C \wedge w\in Y-`\{b\}$ **by** *auto* **note** *bprops* = *this*
hence $Y\ w = b$ **by** *auto*
hence $Y\ w\in$ *space* N **using** *bprops* **by** *simp*
show $w \in D$
by (*metis* (*mono-tags*, *lifting*) *IntE* ‹$Y\ w = b$› ‹$w \in$ ($\bigcup b\in ?C.\ Y\ -`\ \{b\}$) $\cap$ *space* M› *assms*(*3*)
bprops mem-Collect-eq o-apply expl-cond-expect-def)
qed
qed
also have ... = ($\bigcup\ b\in ?C.\ Y-`\{b\}\cap$ *space* M) **by** *blast*
finally have D = ($\bigcup\ b\in ?C.\ Y-`\{b\}\cap$ *space* M)**.**
have $\forall\, b\in ?C.\ Y-`\{b\} \cap$ *space* $M \in$ *sets* M **using** *assms* **unfolding** *point-measurable-def* **by** *auto*
hence ($\bigcup\ b\in ?C.\ Y-`\{b\}\cap$ *space* M) $\in$ *sets* M **using** ‹*countable* $?C$› **by** *blast*
thus *?thesis*
using ‹D = ($\bigcup b\in ?C.\ Y\ -`\ \{b\} \cap$ *space* M)› **by** *blast*
qed

lemma *expl-cond-expect-prop-sets2*:
assumes *disc-fct* Y
and *point-measurable* (*fct-gen-subalgebra* $M\ N\ Y$) (*space* N) Y
and D = {$w\in$ *space* $M.\ Y\ w \in$ *space* $N \wedge$ (P (*expl-cond-expect* $M\ Y\ X\ w$))}
shows $D\in$ *sets* (*fct-gen-subalgebra* $M\ N\ Y$)
proof −
let $?C$ = {$y \in$ (Y‘(*space* M)) $\cap$ (*space* N). P (*img-dce* $M\ Y\ X\ y$)}
have *space* $M \subseteq$ *UNIV* **by** *simp*
hence Y‘(*space* M) $\subseteq$ *range* Y **by** *auto*
hence *countable* (Y‘(*space* M)) **using** *assms countable-subset* **unfolding** *disc-fct-def* **by** *auto*
hence *countable* $?C$ **using** *assms* **unfolding** *disc-fct-def* **by** *auto*
have *eqset*: D = ($\bigcup\ b\in ?C.\ Y-`\{b\}$)$\cap$ *space* M
proof
show $D\subseteq$ ($\bigcup\ b\in ?C.\ Y-`\{b\}$)$\cap$ *space* M
proof

fix w
assume $w \in D$
hence $w \in$ *space* $M \wedge Y\ w \in$ (*space* N) $\wedge$ (P (*expl-cond-expect* $M\ Y\ X\ w$))
by (*simp add*: *assms*)
hence P (*img-dce* $M\ Y\ X$ ($Y\ w$)) **by** (*simp add*: *expl-cond-expect-def*)
hence $Y\ w \in ?C$ **using** ‹$w \in$ *space* $M \wedge Y\ w \in$ *space* $N \wedge P$ (*expl-cond-expect* $M\ Y\ X\ w$)› **by** *blast*
thus $w \in (\bigcup\ b \in ?C.\ Y - `\{b\}) \cap$ *space* M
using ‹$w \in$ *space* $M \wedge Y\ w \in$ *space* $N \wedge P$ (*expl-cond-expect* $M\ Y\ X\ w$)› **by** *blast*
qed
show $(\bigcup\ b \in ?C.\ Y - `\{b\}) \cap$ *space* $M \subseteq D$
proof
fix w
assume $w \in (\bigcup\ b \in ?C.\ Y - `\{b\}) \cap$ *space* M
from *this* **obtain** b **where** $b \in ?C \wedge w \in Y - `\{b\}$ **by** *auto* **note** *bprops* = *this*
hence $Y\ w = b$ **by** *auto*
hence $Y\ w \in$ *space* N **using** *bprops* **by** *simp*
show $w \in D$
by (*metis* (*mono-tags*, *lifting*) *IntE* ‹$Y\ w = b$› ‹$w \in (\bigcup b \in ?C.\ Y - `\{b\}) \cap$ *space* M› *assms*(*3*)
bprops mem-Collect-eq o-apply expl-cond-expect-def)
qed
qed
also have ... = $(\bigcup\ b \in ?C.\ Y - `\{b\} \cap$ *space* M) **by** *blast*
finally have $D = (\bigcup\ b \in ?C.\ Y - `\{b\} \cap$ *space* M)**.**
have *space* M = *space* (*fct-gen-subalgebra* $M\ N\ Y$)
by (*simp add*: *fct-gen-subalgebra-space*)
hence $\forall\, b \in ?C.\ Y - `\{b\} \cap$ *space* $M \in$ *sets* (*fct-gen-subalgebra* $M\ N\ Y$) **using** *assms* **unfolding** *point-measurable-def* **by** *auto*
hence $(\bigcup\ b \in ?C.\ Y - `\{b\} \cap$ *space* $M) \in$ *sets* (*fct-gen-subalgebra* $M\ N\ Y$) **using** ‹*countable* $?C$› **by** *blast*
thus *?thesis*
using ‹$D = (\bigcup b \in ?C.\ Y - `\{b\} \cap$ *space* $M)$› **by** *blast*
qed

lemma *expl-cond-expect-disc-fct*:
assumes *disc-fct* Y
shows *disc-fct* (*expl-cond-expect* $M\ Y\ X$)
using *assms* **unfolding** *disc-fct-def expl-cond-expect-def*
by (*metis countable-image image-comp*)

lemma *expl-cond-expect-point-meas*:
assumes *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
shows *point-measurable M UNIV* (*expl-cond-expect M Y X*)
proof −
have *disc-fct* (*expl-cond-expect M Y X*) **using** *assms* **by** (*simp add*: *expl-cond-expect-disc-fct*)
show *?thesis* **unfolding** *point-measurable-def*
proof
show (*expl-cond-expect M Y X*)‘*space M* ⊆ *UNIV* **by** *simp*
show ∀ *r*∈*range* (*expl-cond-expect M Y X*) ∩ *UNIV*. *expl-cond-expect M Y X*
−‘ {*r*} ∩ *space M* ∈ *sets M*
proof
fix *r*
assume *r*∈ *range* (*expl-cond-expect M Y X*) ∩ *UNIV*
let *?D* = {*w* ∈ *space M*. *Y w* ∈ *space N* ∧ (*expl-cond-expect M Y X w*) = *r*}
have *?D* ∈ *sets M* **using** *expl-cond-expect-prop-sets*[*of Y M N ?D* λ*x*. *x* = *r*
X] **using** *assms* **by** *simp*
moreover have *expl-cond-expect M Y X* −‘{*r*}∩ *space M* = *?D*
proof
show *expl-cond-expect M Y X* −‘{*r*}∩ *space M* ⊆ *?D*
proof
fix *w*
assume *w*∈ *expl-cond-expect M Y X* −‘{*r*}∩ *space M*
hence *Y w* ∈ *space N*
by (*meson IntD2 assms*(*1*) *assms*(*2*) *disct-fct-point-measurable measur-*
able-space)
thus *w* ∈ *?D*
using ‹*w* ∈ *expl-cond-expect M Y X* −‘ {*r*} ∩ *space M*› **by** *blast*
qed
show *?D* ⊆ *expl-cond-expect M Y X* −‘{*r*}∩ *space M*
proof
fix *w*
assume *w*∈ *?D*
thus *w*∈ *expl-cond-expect M Y X* −‘{*r*}∩ *space M* **by** *blast*
qed
qed
ultimately show *expl-cond-expect M Y X* −‘ {*r*} ∩ *space M* ∈ *sets M* **by**
simp
qed
qed
qed

lemma *expl-cond-expect-borel-measurable*:
assumes *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
shows (*expl-cond-expect M Y X*) ∈ *borel-measurable M* **using** *expl-cond-expect-point-meas*[*of*

Y M] *assms*
disct-fct-point-measurable[*of expl-cond-expect M Y X*]
by (*simp add*: *expl-cond-expect-disc-fct*)

lemma *expl-cond-exp-borel*:
assumes $Y \in$ *space* $M \to$ *space* N
and *disc-fct Y*
and $\forall\, r \in$ *range* $Y \cap$ *space* $N.\ \exists\, A \in$ *sets* $N.$ *range* $Y \cap A = \{r\}$
shows (*expl-cond-expect M Y X*) $\in$ *borel-measurable* (*fct-gen-subalgebra M N Y*)
proof (*intro borel-measurableI*)
fix *S*::*real set*
assume *open S*
show *expl-cond-expect M Y X* $-`$ *S* $\cap$ *space* (*fct-gen-subalgebra M N Y*) $\in$ *sets* (*fct-gen-subalgebra M N Y*)
proof (*rule expl-cond-expect-prop-sets2*)
show *disc-fct Y* **using** *assms* **by** *simp*
show *point-measurable* (*fct-gen-subalgebra M N Y*) (*space N*) *Y* **using** *assms*
by (*simp add*: *singl-meas-if*)
show *expl-cond-expect M Y X* $-`$ *S* $\cap$ *space* (*fct-gen-subalgebra M N Y*) $= \{w \in$ *space* $M.\ Y\ w \in$ *space* $N \wedge$ (*expl-cond-expect M Y X w*) $\in S\}$
proof
show *expl-cond-expect M Y X* $-`$ *S* $\cap$ *space* (*fct-gen-subalgebra M N Y*) $\subseteq \{w \in$ *space* $M.\ Y\ w \in$ *space* $N \wedge$ *expl-cond-expect M Y X w* $\in S\}$
proof
fix *x*
assume *asm*: $x \in$ *expl-cond-expect M Y X* $-`$ *S* $\cap$ *space* (*fct-gen-subalgebra M N Y*)
hence *expl-cond-expect M Y X x* $\in S$ **by** *auto*
moreover have $x \in$ *space M* **using** *asm* **by** (*simp add*:*fct-gen-subalgebra-space*)
ultimately show $x \in \{w \in$ *space* $M.\ Y\ w \in$ *space* $N \wedge$ *expl-cond-expect M Y X w* $\in S\}$ **using** *assms* **by** *auto*
qed
show $\{w \in$ *space* $M.\ Y\ w \in$ *space* $N \wedge$ *expl-cond-expect M Y X w* $\in S\} \subseteq$ *expl-cond-expect M Y X* $-`$ *S* $\cap$ *space* (*fct-gen-subalgebra M N Y*)
proof
fix *x*
assume *asm2*: $x \in \{w \in$ *space* $M.\ Y\ w \in$ *space* $N \wedge$ *expl-cond-expect M Y X w* $\in S\}$
hence $x \in$ *space* (*fct-gen-subalgebra M N Y*) **by** (*simp add*:*fct-gen-subalgebra-space*)
moreover have $x \in$ *expl-cond-expect M Y X* $-`S$ **using** *asm2* **by** *simp*
ultimately show $x \in$ *expl-cond-expect M Y X* $-`$ *S* $\cap$ *space* (*fct-gen-subalgebra M N Y*) **by** *simp*
qed
qed
qed
qed

lemma *expl-cond-expect-indic-borel-measurable*:
assumes *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
and *B*⊆ *space N*
and *countable B*
shows (λ*w*. *expl-cond-expect M Y X w* ∗ *indicator* (*countable-preimages B Y n* ∩ *space M*) *w*)∈ *borel-measurable M*
proof −
have *countable-preimages B Y n* ∩ *space M* ∈ *sets M* **using** *assms* **by** (*auto simp add*: *count-pre-meas*)
have (*expl-cond-expect M Y X*)∈ *borel-measurable M* **using** *expl-cond-expect-point-meas*[*of Y M N X*] *assms*
disct-fct-point-measurable[*of expl-cond-expect M Y X*]
by (*simp add*: *expl-cond-expect-disc-fct*)
moreover have (*indicator* (*countable-preimages B Y n* ∩ *space M*))∈ *borel-measurable M*
using ‹*countable-preimages B Y n* ∩ *space M* ∈ *sets M*› *borel-measurable-indicator*
by *blast*
ultimately show *?thesis*
using *borel-measurable-times* **by** *blast*
qed

lemma (**in** *finite-measure*) *dce-prod*:
assumes *point-measurable M* (*space N*) *Y*
and *integrable M X*
and ∀ *w*∈ *space M*. *0* ≤ *X w*
shows ∀ *w*. (*Y w*) ∈ *space N* ⟶ (*expl-cond-expect M Y X*) *w* ∗ *measure M* ((*Y* −‘ {*Y w*})∩ *space M*) = *integral*^{L} *M* (λ*y*. (*X y*) ∗ (*indicator* ((*Y* −‘{*Y w*})∩ *space M*) *y*))
proof (*intro allI impI*)
fix *w*
assume *Y w*∈ *space N*
let *?indY* = (λ*y*. *indicator* ((*Y* −‘{*Y w*})∩ *space M*) *y*)::′*a* ⇒ *real*
show *expl-cond-expect M Y X w* ∗ *measure M* ((*Y* −‘ {*Y w*})∩ *space M*) = *integral*^{L} *M* (λ*y*. (*X y*) ∗ *?indY y*)
proof (*cases AE y in M*. *?indY y* = *0*)
case *True*

hence *emeasure M* ((*Y* −‘ {*Y w*})∩ *space M*) = *0*
proof −
have *AE y in M*. *y* ∉ *Y* −‘ {*Y w*} ∩ *space M*
using *True eventually-elim2* **by** *auto*
hence ∃ *N*∈ *null-sets M*.{*x*∈ *space M*. ¬(*x*∉ *Y* −‘ {*Y w*} ∩ *space M*)} ⊆ *N*
using *eventually-ae-filter*[*of* λ*x*. *x* ∉ *Y* −‘ {*Y w*} ∩ *space M M*] **by** *simp*

hence $\exists N \in$ *null-sets M*. $\{x \in$ *space M*. $x \in Y -` \{Y\ w\} \cap$ *space M*$\} \subseteq N$ **by** *simp*

from *this* **obtain** *N* **where** $N \in$ *null-sets M* **and** $\{x \in$ *space M*. $x \in Y -` \{Y\ w\} \cap$ *space M*$\} \subseteq N$ **by** *auto*

note *Nprops* = *this*

have $\{x \in$ *space M*. $x \in Y -` \{Y\ w\}\} \subseteq N$ **using** *Nprops* **by** *auto*

hence *emeasure M* $\{x \in$ *space M*. $x \in Y -` \{Y\ w\}\} \leq$ *emeasure M N*

by (*simp add*: *emeasure-mono Nprops*(*1*) *null-setsD2*)

thus *?thesis*

by (*metis* (*no-types*, *lifting*) *Collect-cong Int-def Nprops*(*1*) *le-zero-eq null-setsD1*)

qed

hence *enn2real* (*emeasure M* $((Y -` \{Y\ w\}) \cap$ *space M*$) = 0$ **by** *simp*

hence *measure M* $((Y -` \{Y\ w\}) \cap$ *space M*$) = 0$ **unfolding** *measure-def* **by** *simp*

hence *lhs*: *expl-cond-expect M Y X w* $= 0$ **unfolding** *expl-cond-expect-def img-dce-def* **by** *simp*

have *zer*: *AE y in M*. $(X\ y) * ?indY\ y = (\lambda y.\ 0)\ y$ **using** *True* **by** *auto*

hence *rhs*: *integral*^{L} *M* $(\lambda y.\ (X\ y) * ?indY\ y) = 0$

proof −

have $\forall\ w \in$ *space M*. $0 \leq X\ w * ?indY\ w$ **using** *assms* **by** *simp*

have *integrable M* $(\lambda y.\ (X\ y) * ?indY\ y)$ **using** *assms*

by (*metis* (*mono-tags*, *lifting*) *IntI UNIV-I* ‹*Y w* ∈ *space N*› *image-eqI Bochner-Integration.integrable-cong integrable-real-mult-indicator point-measurable-def*)

hence $(\lambda y.\ (X\ y) * ?indY\ y) \in$ *borel-measurable M* **by** *blast*

thus *?thesis* **using** *zer integral-cong-AE*[*of* $(\lambda y.\ (X\ y) * ?indY\ y)$ *M* $\lambda y.\ 0$] **by** *simp*

qed

thus *expl-cond-expect M Y X w*∗*measure M* $((Y -` \{Y\ w\}) \cap$ *space M*$) =$ *integral*^{L} *M* $(\lambda y.\ (X\ y) * ?indY\ y)$ **using** *lhs rhs* **by** *simp*

next

case *False*

hence $\neg$(*AE y in M*. $y \notin (Y -` \{Y\ w\}) \cap$ *space M*)

by (*simp add*: *indicator-eq-0-iff*)

hence *emeasure M* $((Y -` \{Y\ w\}) \cap$ *space M*$) \neq 0$

proof −

have $(Y -` \{Y\ w\}) \cap$ *space M* ∈ *sets M*

by (*meson IntI UNIV-I* ‹*Y w* ∈ *space N*› *assms*(*1*) *image-eqI point-measurable-def*)

have $(Y -` \{Y\ w\}) \cap$ *space M* $\notin$ *null-sets M*

using ‹¬ (*AE y in M*. $y \notin Y -` \{Y\ w\} \cap$ *space M*)› *eventually-ae-filter* **by** *blast*

thus *?thesis*

using ‹*Y* $-` \{Y\ w\} \cap$ *space M* ∈ *sets M*› **by** *blast*

qed

hence *measure M* $((Y -` \{Y\ w\}) \cap$ *space M*$) \neq 0$

by (*simp add*: *emeasure-eq-measure*)

thus *expl-cond-expect M Y X w*∗ *measure M* $((Y -` \{Y\ w\}) \cap$ *space M*$) =$ *integral*^{L} *M* $(\lambda y.\ (X\ y) * ?indY\ y)$ **unfolding** *expl-cond-expect-def img-dce-def*

using *o-apply* **by** *auto*

qed
qed

lemma *expl-cond-expect-const-exp*:
shows $integral^L\ M\ (\lambda y.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * (indicator\ (Y -` \{Y\ w\} \cap space\ M))\ y) =$
$integral^L\ M\ (\lambda y.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ y * (indicator\ (Y -` \{Y\ w\} \cap space\ M))\ y)$
proof −
let *?ind* = $(indicator\ (Y -` \{Y\ w\} \cap space\ M))$
have $\forall\ y \in space\ M.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * ?ind\ y = expl\text{-}cond\text{-}expect\ M\ Y\ X\ y * ?ind\ y$
proof
fix *y*
assume $y \in space\ M$
show $expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * ?ind\ y = expl\text{-}cond\text{-}expect\ M\ Y\ X\ y * ?ind\ y$
proof (*cases* $y \in Y -` \{Y\ w\} \cap space\ M$)
case *False*
thus *?thesis* **by** *simp*
next
case *True*
hence $Y\ w = Y\ y$ **by** *auto*
hence $expl\text{-}cond\text{-}expect\ M\ Y\ X\ w = expl\text{-}cond\text{-}expect\ M\ Y\ X\ y$
using *expl-cond-expect-const*[*of Y w y M X*] **by** *simp*
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis*
by (*meson Bochner-Integration.integral-cong*)
qed

lemma *nn-expl-cond-expect-const-exp*:
assumes $\forall\ w \in space\ M.\ 0 \le X\ w$
shows $integral^N\ M\ (\lambda y.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * (indicator\ (Y -` \{Y\ w\} \cap space\ M))\ y) =$
$integral^N\ M\ (\lambda y.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ y * (indicator\ (Y -` \{Y\ w\} \cap space\ M))\ y)$
proof −
let *?ind* = $(indicator\ (Y -` \{Y\ w\} \cap space\ M))$
have *forall*: $\forall\ y \in space\ M.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * ?ind\ y = expl\text{-}cond\text{-}expect\ M\ Y\ X\ y * ?ind\ y$
proof
fix *y*
assume $y \in space\ M$
show $expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * ?ind\ y = expl\text{-}cond\text{-}expect\ M\ Y\ X\ y * ?ind\ y$
proof (*cases* $y \in Y -` \{Y\ w\} \cap space\ M$)
case *False*

thus *?thesis* **by** *simp*
next
case *True*
hence $Y\ w = Y\ y$ **by** *auto*
hence *expl-cond-expect M Y X w = expl-cond-expect M Y X y*
using *expl-cond-expect-const*[*of Y*] **by** *blast*
thus *?thesis* **by** *simp*
qed
qed
show *?thesis*
by (*metis* (*no-types*, *lifting*) *forall nn-integral-cong*)
qed

lemma (**in** *finite-measure*) *nn-expl-cond-bounded*:
assumes $\forall\, w{\in}$ *space M*. $0 \le X\ w$
and *integrable M X*
and *point-measurable M* (*space N*) *Y*
and $w{\in}$ *space M*
and $Y\ w{\in}$ *space N*
shows $integral^N\ M\ (\lambda y.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ y * (indicator\ (Y\ -`\ \{Y\ w\} \cap space\ M))\ y) < \infty$
proof −
let *?ind* = (*indicator* $(Y\ -`\ \{Y\ w\} \cap space\ M)$)::$'a{\Rightarrow}real$
have $0 \le$ *expl-cond-expect M Y X w* **using** *assms nn-expl-cond-expect-pos*[*of M X Y*] **by** *simp*
have *integrable M* $(\lambda y.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * ?ind\ y)$
proof −
have *eq*: $(Y\ -`\{Y\ w\} \cap space\ M) \cap space\ M = (Y\ -`\{Y\ w\} \cap space\ M)$ **by** *auto*
have $(Y\ -`\ \{Y\ w\} \cap space\ M) \in sets\ M$ **using** *assms*
by (*simp add*: *point-measurable-def*)
moreover have *emeasure M* $(Y\ -`\{Y\ w\} \cap space\ M) < \infty$ **by** (*simp add*: *inf.strict-order-iff*)
ultimately have *integrable M* $(\lambda y.\ ?ind\ y)$
using *integrable-indicator-iff*[*of M* $(Y\ -`\{Y\ w\} \cap space\ M)$] **by** *simp*
thus *?thesis* **using** *integrable-mult-left-iff*[*of M expl-cond-expect M Y X w ?ind*] **by** *blast*
qed
have $\forall\, y{\in}$ *space M*. $0 \le$ *expl-cond-expect M Y X w * ?ind y*
using ‹$0 \le$ *expl-cond-expect M Y X w*› *mult-nonneg-nonneg* **by** *blast*
hence $\forall\, y{\in}$ *space M*. *expl-cond-expect M Y X w * ?ind y = norm* (*expl-cond-expect M Y X w * ?ind y*) **by** *auto*
hence *inf*: $integral^N\ M\ (\lambda y.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * ?ind\ y) < \infty$
using *integrable-iff-bounded*[*of M* $(\lambda y.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * ?ind\ y)$]
‹$0 \le$ *expl-cond-expect M Y X w*› *real-norm-def nn-integral-cong*
by (*metis* (*no-types*, *lifting*) ‹*integrable M* $(\lambda y.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * indicator\ (Y\ -`\ \{Y\ w\} \cap space\ M)\ y)$›)
have $integral^N\ M\ (\lambda y.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ y * ?ind\ y) =$

integral^N *M* (λ*y*. *expl-cond-expect M Y X w* ∗ *?ind y*) **using** *assms*
by (*simp add*: *nn-expl-cond-expect-const-exp*)
also have ... < ∞ **using** *inf* **by** *simp*
finally show *?thesis* **.**
qed

lemma (**in** *finite-measure*) *count-prod*:
fixes *Y*::′*a*⇒′*b*
assumes *B*⊆ *space N*
and *point-measurable M* (*space N*) *Y*
and *integrable M X*
and ∀ *w* ∈ *space M*. *0* ≤ *X w*
shows ∀ *i*. *integral*^{L} *M* (λ*y*. (*X y*) ∗ (*indicator* (*countable-preimages B Y i* ∩ *space M*)) *y*) =
integral^L *M* (λ*y*. (*expl-cond-expect M Y X y*) ∗ (*indicator* (*countable-preimages B Y i* ∩ *space M*)) *y*)
proof
fix *i*
show *integral*^{L} *M* (λ*y*. *X y* ∗ *indicator* (*countable-preimages B Y i* ∩ *space M*) *y*) =
integral^L *M* (λ*y*. *expl-cond-expect M Y X y* ∗ *indicator* (*countable-preimages B Y i* ∩ *space M*) *y*)
proof (*cases countable-preimages B Y i* ∩ *space M* = {})
case *True*
thus *?thesis* **by** *simp*
next
case *False*
from *this* **obtain** *w* **where** *w*∈ *countable-preimages B Y i* **by** *auto*
hence *Y w* = (*from-nat-into B*) *i* **by** (*meson count-pre-img*)
hence *Y w* ∈ *B*
proof (*cases infinite B*)
case *True*
thus *?thesis*
by (*simp add*: ‹*Y w* = *from-nat-into B i*› *from-nat-into infinite-imp-nonempty*)
next
case *False*
thus *?thesis*
by (*metis Finite-Set.card-0-eq* ‹*Y w* = *from-nat-into B i*› ‹*w* ∈ *countable-preimages B Y i*› *countable-preimages-def equals0D from-nat-into gr-implies-not0*)
qed
let *?ind* = (*indicator* (*Y* −‘ {*Y w*} ∩ *space M*))::′*a*⇒*real*
have *integral*^{L} *M* (λ*y*. (*X y*) ∗ (*indicator* (*countable-preimages B Y i* ∩ *space M*)) *y*) = *integral*^{L} *M* (λ*y*. *X y* ∗ *?ind y*)
by (*metis* (*no-types*, *opaque-lifting*) ‹*Y w* = *from-nat-into B i*› ‹⋀*thesis*. (⋀*w*. *w* ∈ *countable-preimages B Y i* ⟹ *thesis*) ⟹ *thesis*› *countable-preimages-def empty-iff*)
also have ... =
expl-cond-expect M Y X w ∗ *measure M* (*Y* −‘ {*Y w*} ∩ *space M*) **using**

dce-prod[*of N Y X*]
by (*metis* (*no-types*, *lifting*) ‹*Y w* ∈ *B*› *assms subsetCE*)
also have ... = *expl-cond-expect M Y X w* ∗ (*integral*^{L} *M ?ind*)
by *auto*
also have ... = *integral*^{L} *M* (λ*y*. *expl-cond-expect M Y X w* ∗ *?ind y*)
by *auto*
also have ... = *integral*^{L} *M* (λ*y*. *expl-cond-expect M Y X y* ∗ *?ind y*)
proof −
have ∀ *y*∈ *space M*. *expl-cond-expect M Y X w* ∗ *?ind y* = *expl-cond-expect M Y X y* ∗ *?ind y*
proof
fix *y*
assume *y*∈ *space M*
show *expl-cond-expect M Y X w* ∗ *?ind y* = *expl-cond-expect M Y X y* ∗ *?ind y*
proof (*cases y*∈ *Y* −‘ {*Y w*} ∩ *space M*)
case *False*
thus *?thesis* **by** *simp*
next
case *True*
hence *Y w* = *Y y* **by** *auto*
hence *expl-cond-expect M Y X w* = *expl-cond-expect M Y X y*
using *expl-cond-expect-const*[*of Y*] **by** *blast*
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis*
by (*meson Bochner-Integration.integral-cong*)
qed
also have ... = *integral*^{L} *M* (λ*y*. *expl-cond-expect M Y X y* ∗ *indicator* (*countable-preimages B Y i* ∩ *space M*) *y*)
by (*metis* (*no-types*, *opaque-lifting*) ‹*Y w* = *from-nat-into B i*› ‹⋀*thesis*. (⋀*w*. *w* ∈ *countable-preimages B Y i* ⟹ *thesis*) ⟹ *thesis*› *countable-preimages-def empty-iff*)
finally show *?thesis* **.**
qed
qed

lemma (**in** *finite-measure*) *count-pre-integrable*:
assumes *point-measurable M* (*space N*) *Y*
and *disc-fct Y*
and *B*⊆ *space N*
and *countable B*
and *integrable M X*
and ∀ *w* ∈ *space M*. *0* ≤ *X w*
shows *integrable M* (λ*w*. *expl-cond-expect M Y X w* ∗ *indicator* (*countable-preimages B Y n* ∩ *space M*) *w*)

proof−
have $integral^L\ M\ (\lambda y.\ (X\ y) * (indicator\ (countable\text{-}preimages\ B\ Y\ n \cap space\ M))\ y) =$
$integral^L\ M\ (\lambda y.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ y) * (indicator\ (countable\text{-}preimages\ B\ Y\ n \cap space\ M))\ y)$ **using** *assms count-prod*
by *blast*
have $\forall\, w \in space\ M.\ 0 \leq (expl\text{-}cond\text{-}expect\ M\ Y\ X\ w) * (indicator\ (countable\text{-}preimages\ B\ Y\ n \cap space\ M))\ w$
by (*simp add*: *assms nn-expl-cond-expect-pos*)
have $countable\text{-}preimages\ B\ Y\ n \cap space\ M \in sets\ M$ **using** *count-pre-meas*[*of M*] *assms* **by** *auto*
hence $integrable\ M\ (\lambda w.\ X\ w * indicator\ (countable\text{-}preimages\ B\ Y\ n \cap space\ M)\ w)$
using *assms integrable-real-mult-indicator* **by** *blast*
show *?thesis*
proof (*rule integrableI-nonneg*)
show $(\lambda w.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * indicator\ (countable\text{-}preimages\ B\ Y\ n \cap space\ M)\ w) \in borel\text{-}measurable\ M$
proof −
have $(expl\text{-}cond\text{-}expect\ M\ Y\ X) \in borel\text{-}measurable\ M$ **using** *expl-cond-expect-point-meas*[*of Y M N X*] *assms*
disct-fct-point-measurable[*of expl-cond-expect M Y X*]
by (*simp add*: *expl-cond-expect-disc-fct*)
moreover have $(indicator\ (countable\text{-}preimages\ B\ Y\ n \cap space\ M)) \in borel\text{-}measurable\ M$
using ‹$countable\text{-}preimages\ B\ Y\ n \cap space\ M \in sets\ M$› *borel-measurable-indicator*
by *blast*
ultimately show *?thesis*
using *borel-measurable-times* **by** *blast*
qed
show $AE\ x\ in\ M.\ 0 \leq expl\text{-}cond\text{-}expect\ M\ Y\ X\ x * indicator\ (countable\text{-}preimages\ B\ Y\ n \cap space\ M)\ x$
by (*simp add*: ‹$\forall\, w \in space\ M.\ 0 \leq expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * indicator\ (countable\text{-}preimages\ B\ Y\ n \cap space\ M)\ w$›)
show $(\int^+ x.\ ennreal\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ x * indicator\ (countable\text{-}preimages\ B\ Y\ n \cap space\ M)\ x)\ \partial M) < \infty$
proof (*cases countable-preimages B Y n ∩ space M = {}*)
case *True*
thus *?thesis* **by** *simp*
next
case *False*
from *this* **obtain** *w* **where** *w*∈ *countable-preimages B Y n*∩ *space M* **by** *auto*
hence $countable\text{-}preimages\ B\ Y\ n = Y - `\{Y\ w\}$
by (*metis IntD1 count-pre-img countable-preimages-def equals0D*)
have *w*∈ *space M* **using** *False*
using ‹$w \in countable\text{-}preimages\ B\ Y\ n \cap space\ M$› **by** *blast*
moreover have $Y\ w \in space\ N$
by (*meson* ‹$w \in space\ M$› *assms*(*1*) *assms*(*2*) *disct-fct-point-measurable measurable-space*)

ultimately show *?thesis* **using** *assms nn-expl-cond-bounded*[*of X N Y*]
using ‹*countable-preimages B Y n* = *Y* −‘ {*Y w*}› **by** *presburger*
qed
qed
qed

lemma (**in** *finite-measure*) *nn-cond-expl-is-cond-exp-tmp*:
assumes ∀ *w*∈ *space M*. 0 ≤ *X w*
and *integrable M X*
and *disc-fct Y*
and *point-measurable M* (*space M′*) *Y*
shows ∀ *A* ∈ *sets M′*. *integrable M* (λ*w*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator* ((*Y* −‘*A*)∩ (*space M*)) *w*)) ∧
integral^L *M* (λ*w*. (*X w*) ∗ (*indicator* ((*Y* −‘*A*)∩ (*space M*)) *w*)) =
integral^L *M* (λ*w*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator* ((*Y* −‘*A*) ∩ (*space M*))) *w*)
proof
fix *A*
assume *A* ∈ *sets M′*
let *?imA* = *A* ∩ (*range Y*)
have *countable ?imA* **using** *assms disc-fct-def* **by** *blast*
have *Y* −‘*A* = *Y* −‘*?imA* **by** *auto*
define *prY* **where** *prY* = *countable-preimages ?imA Y*
have *un*: *Y* −‘*?imA* = (⋃ *i*. *prY i*) **using** ‹*countable ?imA*›
by (*metis count-pre-union-img prY-def*)
have (*Y* −‘*?imA*) ∩ (*space M*) = (⋃ *i*. *prY i*) ∩ (*space M*) **using** ‹*Y* −‘*A* = *Y* −‘*?imA*› *un* **by** *simp*
also have ... = (⋃ *i*. (*prY i*) ∩ (*space M*)) **by** *blast*
finally have *eq2*: (*Y* −‘*?imA*) ∩ (*space M*) = (⋃ *i*. (*prY i*) ∩ (*space M*)).
define *indpre*::*nat* ⇒ *′a* ⇒ *real* **where** *indpre* = (λ *i x*. (*indicator* ((*prY i*) ∩ (*space M*))) *x*)
have ∀ *i*. *indpre i* ∈ *borel-measurable M*
proof
fix *i*
show *indpre i* ∈ *borel-measurable M* **unfolding** *indpre-def prY-def*
proof (*rule borel-measurable-indicator*, *cases countable-preimages* (*A* ∩ *range Y*) *Y i* ∩ *space M* = {})
case *True*
thus *countable-preimages* (*A* ∩ *range Y*) *Y i* ∩ *space M* ∈ *sets M* **by** *simp*
next
case *False*
from *this* **obtain** *x* **where** *x*∈ *countable-preimages* (*A* ∩ *range Y*) *Y i* ∩ *space M* **by** *blast*
hence *Y x* ∈ *space M′*
by (*metis Int-iff UN-I* ‹*A* ∈ *sets M′*› ‹*prY* ≡ *countable-preimages* (*A*

$\cap$ *range Y*) *Y*› *imageE*
rangeI sets.sets-into-space subset-eq un vimage-eq)
thus *countable-preimages* (*A* $\cap$ *range Y*) *Y i* $\cap$ *space M* $\in$ *sets M*
by (*metis IntE IntI* ‹*x* $\in$ *countable-preimages* (*A* $\cap$ *range Y*) *Y i* $\cap$ *space M*› *assms*(*4*)
count-pre-img countable-preimages-def empty-iff point-measurable-def rangeI)
qed
qed
have $\forall$ *i*. *integrable M* (λ*w*. (*X w*) $*$ *indpre i w*)
proof
fix *i*
show *integrable M* (λ*w*. (*X w*) $*$ *indpre i w*) **unfolding** *indpre-def prY-def*
proof (*rule integrable-real-mult-indicator*)
show *countable-preimages* (*A* $\cap$ *range Y*) *Y i* $\cap$ *space M* $\in$ *sets M*
proof (*cases countable-preimages* (*A* $\cap$ *range Y*) *Y i* = {})
case *True*
thus *countable-preimages* (*A* $\cap$ *range Y*) *Y i* $\cap$ *space M* $\in$ *sets M* **by** (*simp add*: *True*)
next
case *False*
hence *Y* $-$‘ {(*from-nat-into* (*A* $\cap$ *range Y*)) *i*} $\neq$ {} **unfolding** *countable-preimages-def* **by** *meson*
have (*infinite* (*A* $\cap$ *range Y*)) $\vee$ (*finite* (*A* $\cap$ *range Y*) $\wedge$ *i* $<$ *card* (*A* $\cap$ *range Y*)) **using** *False* **unfolding** *countable-preimages-def*
by *meson*
show *?thesis*
by (*metis* ‹*A* $\in$ *sets M'*› ‹*countable* (*A* $\cap$ *range Y*)› *assms*(*4*) *count-pre-meas le-inf-iff*
range-from-nat-into sets.Int-space-eq1 sets.empty-sets sets.sets-into-space subset-range-from-nat-into)
qed
show *integrable M X* **using** *assms* **by** *simp*
qed
qed
hence *prod-bm*: $\forall$ *i*. (λ*w*. (*X w*) $*$ *indpre i w*) $\in$ *borel-measurable M*
by (*simp add*: *assms*(*2*) *borel-measurable-integrable borel-measurable-times*)
have *posprod*: $\forall$ *i w*. *0* $\leq$ (*X w*) $*$ *indpre i w*
proof (*intro allI*)
fix *i*
fix *w*
show *0* $\leq$ (*X w*) $*$ *indpre i w*
by (*metis IntE assms*(*1*) *indicator-pos-le indicator-simps*(*2*) *indpre-def mult-eq-0-iff mult-sign-intros*(*1*))
qed
let *?indA* = *indicator* ((*Y* $-$‘(*A* $\cap$ *range Y*))$\cap$ (*space M*))::*'a*$\Rightarrow$*real*

have $\forall$ *i j*. *i* $\neq$ *j* $\longrightarrow$ ((*prY i*) $\cap$ (*space M*)) $\cap$ ((*prY j*) $\cap$ (*space M*)) = {}
by (*simp add*: ‹*countable* (*A* $\cap$ *range Y*)› ‹*prY* $\equiv$ *countable-preimages* (*A* $\cap$

range Y) Y› count-pre-disj inf-commute inf-sup-aci(3))
hence *sumind*: $\forall x.\ (\lambda i.\ indpre\ i\ x)\ sums\ ?indA\ x$ **using** ‹*countable ?imA*› *eq2* **unfolding** *prY-def indpre-def*
by (*metis indicator-sums*)
hence *sumxlim*: $\forall x.\ (\lambda i.\ (X\ x) * indpre\ i\ x{::}real)\ sums\ ((X\ x) * indicator\ ((Y - `?imA) \cap (space\ M))\ x)$ **using** ‹*countable ?imA*› **unfolding** *prY-def*
using *sums-mult* **by** *blast*
hence *sum*: $\forall x.\ (\sum\ i.((X\ x) * indpre\ i\ x){::}real) = (X\ x) * indicator\ ((Y - `?imA) \cap (space\ M))\ x$ **by** (*metis sums-unique*)
hence *b*: $\forall\ w.\ 0 \leq (\sum\ i.((X\ w) * indpre\ i\ w))$ **using** *suminf-nonneg*
by (*metis* ‹$\forall x.\ (\lambda i.\ X\ x * indpre\ i\ x)\ sums\ (X\ x * indicator\ (Y - `(A \cap range\ Y) \cap space\ M)\ x)$› *posprod summable-def*)
have *sumcondlim*: $\forall x.\ (\lambda i.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ x) * indpre\ i\ x{::}real)\ sums\ ((expl\text{-}cond\text{-}expect\ M\ Y\ X\ x) * ?indA\ x)$ **using** ‹*countable ?imA*› **unfolding** *prY-def*
using *sums-mult sumind* **by** *blast*

have *integrable M* $(\lambda w.\ (X\ w) * ?indA\ w)$
proof (*rule integrable-real-mult-indicator*)
show $Y - `(A \cap range\ Y) \cap space\ M \in sets\ M$
using ‹$A \in sets\ M'$› *assms(3) assms(4) disct-fct-point-measurable measurable-sets*
by (*metis* ‹$Y - `A = Y - `(A \cap range\ Y)$›)
show *integrable M X* **using** *assms* **by** *simp*
qed
hence *intsum*: *integrable M* $(\lambda w.\ (\sum i.\ ((X\ w) * indpre\ i\ w)))$ **using** *sum Bochner-Integration.integrable-cong*[*of M M* $\lambda\ w.(X\ w) * (indicator\ ((Y - `A) \cap (space\ M))\ w)\ \lambda w.\ (\sum\ i.((X\ w) * indpre\ i\ w))$]
using ‹$Y - `A = Y - `(A \cap range\ Y)$› **by** *presburger*
have $integral^L\ M\ (\lambda w.\ (X\ w) * ?indA\ w) = integral^L\ M\ (\lambda w.\ (\sum\ i.((X\ w) * indpre\ i\ w)))$
using ‹$Y - `A = Y - `(A \cap range\ Y)$› *sum* **by** *auto*
also have ... =
$\int^+ w.\ \ ((\sum\ i.\ ((X\ w) * indpre\ i\ w)))\ \partial M$ **using** *nn-integral-eq-integral*
by (*metis* (*mono-tags*, *lifting*) *AE-I2 intsum b nn-integral-cong*)
also have $(\int^+ w.\ \ ((\sum\ i.\ ((X\ w) * indpre\ i\ w)))\ \partial M) = \int^+ w.\ \ ((\sum\ i.\ ennreal\ ((X\ w) * indpre\ i\ w)))\ \partial M$ **using** *suminf-ennreal2 summable-def posprod sum sumxlim*
proof −
{ fix *aa* :: *'a*
have $\forall a.\ ennreal\ (\sum n.\ X\ a * indpre\ n\ a) = (\sum n.\ ennreal\ (X\ a * indpre\ n\ a))$
by (*metis* (*full-types*) *posprod suminf-ennreal2 summable-def sumxlim*)
then have $(\int^+ a.\ ennreal\ (\sum n.\ X\ a * indpre\ n\ a)\ \partial M) = (\int^+ a.\ (\sum n.\ ennreal\ (X\ a * indpre\ n\ a))\ \partial M) \vee ennreal\ (\sum n.\ X\ aa * indpre\ n\ aa) = (\sum n.\ ennreal\ (X\ aa * indpre\ n\ aa))$
by *metis* **}**
then show *?thesis*
by *presburger*

qed
also have ... = $(\sum i.\ integral^N\ M\ ((\lambda i\ w.\ (X\ w) * indpre\ i\ w)\ i))$
proof (*intro nn-integral-suminf*)
fix *i*
show $(\lambda x.\ ennreal\ (X\ x * indpre\ i\ x)) \in borel\text{-}measurable\ M$
using *measurable-compose-rev measurable-ennreal prod-bm* **by** *blast*
qed
also have ... = $(\sum i.\ integral^N\ M\ ((\lambda i\ w.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ w) * indpre\ i\ w)\ i))$
proof (*intro suminf-cong*)
fix *n*
have $A \cap range\ Y \subseteq space\ M'$
using ‹$A \in sets\ M'$› *sets.Int-space-eq1* **by** *auto*
have $integral^N\ M\ (\lambda w.\ (X\ w) * indpre\ n\ w) = integral^L\ M\ (\lambda w.\ (X\ w) * indpre\ n\ w)$
using *nn-integral-eq-integral*[*of M* $\lambda w.\ (X\ w) * indpre\ n\ w$]
by (*simp add*: ‹$\forall i.\ integrable\ M\ (\lambda w.\ X\ w * indpre\ i\ w)$› *posprod*)
also have ... = $integral^L\ M\ (\lambda w.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X)\ w * indpre\ n\ w)$
proof −
have $integral^L\ M\ (\lambda w.\ X\ w * indicator\ (countable\text{-}preimages\ (A \cap range\ Y)\ Y\ n \cap space\ M)\ w) =$
$integral^L\ M\ (\lambda w.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * indicator\ (countable\text{-}preimages\ (A \cap range\ Y)\ Y\ n \cap space\ M)\ w)$
using *count-prod*[*of A*∩ *range Y M′ Y X*] *assms* ‹$A \cap range\ Y \subseteq space\ M'$› **by** *blast*
thus *?thesis*
using ‹$indpre \equiv \lambda i.\ indicator\ (prY\ i \cap space\ M)$› *prY-def* **by** *presburger*
qed
also have ... = $integral^N\ M\ (\lambda w.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X)\ w * indpre\ n\ w)$
proof −
have $integrable\ M\ (\lambda w.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X)\ w * indpre\ n\ w)$ **unfolding** *indpre-def prY-def*
using *count-pre-integrable assms* ‹$A \cap range\ Y \subseteq space\ M'$› ‹$countable\ (A \cap range\ Y)$› **by** *blast*
moreover have $AE\ w\ in\ M.\ 0 \leq (expl\text{-}cond\text{-}expect\ M\ Y\ X)\ w * indpre\ n\ w$
by (*simp add*: ‹$indpre \equiv \lambda i.\ indicator\ (prY\ i \cap space\ M)$› *assms*(*1*) *nn-expl-cond-expect-pos*)
ultimately show *?thesis* **by** (*simp add*:*nn-integral-eq-integral*)
qed
finally show $integral^N\ M\ (\lambda w.\ (X\ w) * indpre\ n\ w) = integral^N\ M\ (\lambda w.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X)\ w * indpre\ n\ w)$ **.**
qed
also have ... = $integral^N\ M\ (\lambda w.\ \sum i.\ ((expl\text{-}cond\text{-}expect\ M\ Y\ X\ w) * indpre\ i\ w))$
proof −
have $(\lambda x.\ (\sum i.\ ennreal\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ x * indpre\ i\ x))) =$
$(\lambda x.\ ennreal\ (\sum i.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ x * indpre\ i\ x)))$
proof−
have *posex*: $\forall\ i\ x.\ 0 \leq (expl\text{-}cond\text{-}expect\ M\ Y\ X\ x) * (indpre\ i\ x)$

by (*metis IntE* ‹*indpre* ≡ λ*i*. *indicator* (*prY i* ∩ *space M*)› *assms*(*1*) *indicator-pos-le indicator-simps*(*2*) *mult-eq-0-iff mult-sign-intros*(*1*) *nn-expl-cond-expect-pos*)
have ∀ *x*. (∑ *i*. *ennreal* (*expl-cond-expect M Y X x* ∗ *indpre i x*)) = (*ennreal* (∑ *i*. (*expl-cond-expect M Y X x* ∗ *indpre i x*)))
proof
fix *x*
show (∑ *i*. *ennreal* (*expl-cond-expect M Y X x* ∗ *indpre i x*)) = (*ennreal* (∑ *i*. (*expl-cond-expect M Y X x* ∗ *indpre i x*)))
using *suminf-ennreal2*[*of* λ*i*. (*expl-cond-expect M Y X x* ∗ *indpre i x*)] *sumcondlim summable-def posex*
proof −
have *f1*: *summable* (λ*n*. *expl-cond-expect M Y X x* ∗ *indpre n x*)
using *sumcondlim summable-def* **by** *blast*
obtain *nn* :: *nat* **where**
¬ *0* ≤ *expl-cond-expect M Y X x* ∗ *indpre nn x* ∨ ¬ *summable* (λ*n*. *expl-cond-expect M Y X x* ∗ *indpre n x*) ∨ *ennreal* (∑ *n*. *expl-cond-expect M Y X x* ∗ *indpre n x*) = (∑ *n*. *ennreal* (*expl-cond-expect M Y X x* ∗ *indpre n x*))
by (*metis* (*full-types*) ‹⟦⋀*i*. *0* ≤ *expl-cond-expect M Y X x* ∗ *indpre i x*; *summable* (λ*i*. *expl-cond-expect M Y X x* ∗ *indpre i x*)⟧ ⟹ (∑ *i*. *ennreal* (*expl-cond-expect M Y X x* ∗ *indpre i x*)) = *ennreal* (∑ *i*. *expl-cond-expect M Y X x* ∗ *indpre i x*)›)
then show *?thesis*
using *f1 posex* **by** *presburger*
qed
qed
thus *?thesis* **by** *simp*
qed
have ∀ *i*. (λ*w*. (*expl-cond-expect M Y X w*) ∗ *indpre i w*) ∈ *borel-measurable M*
proof −
show *?thesis*
using ‹∀ *i*. (*indpre i*)∈ *borel-measurable M*› *assms*(*3*) *assms*(*4*) *borel-measurable-times expl-cond-expect-borel-measurable* **by** *blast*
qed
hence ⋀*i*. (λ*x*. *ennreal* (*expl-cond-expect M Y X x* ∗ *indpre i x*))∈ *borel-measurable M*
using *measurable-compose-rev measurable-ennreal* **by** *blast*
thus *?thesis* **using** *nn-integral-suminf*[*of* (λ*i w*. (*expl-cond-expect M Y X w*) ∗ *indpre i w*) *M*, *symmetric*]
using ‹(λ*x*. ∑ *i*. *ennreal* (*expl-cond-expect M Y X x* ∗ *indpre i x*)) = (λ*x*. *ennreal* (∑ *i*. *expl-cond-expect M Y X x* ∗ *indpre i x*))› **by** *auto*
qed
also have ... = *integral*^{N} *M* (λ*w*. (*expl-cond-expect M Y X w*) ∗ *?indA w*)
using *sumcondlim*
by (*metis* (*no-types*, *lifting*) *sums-unique*)
also have ... = *integral*^{L} *M* (λ*w*. (*expl-cond-expect M Y X w*) ∗ *?indA w*)
proof −
have *scdint*: *integrable M* (λ*w*. (*expl-cond-expect M Y X w*) ∗ *?indA w*)
proof −
have *rv*: (λ*w*. (*expl-cond-expect M Y X w*) ∗ *indicator* ((*Y* −‘*?imA*) ∩ (*space*

M)) *w*) ∈ *borel-measurable M*
proof −
have *expl-cond-expect M Y X* ∈ *borel-measurable M* **using** *expl-cond-expect-borel-measurable*
using *assms* **by** *blast*
moreover have (*Y* −‘*?imA*) ∩ (*space M*) ∈ *sets M*
by (*metis* ‹*A* ∈ *sets M*′› ‹*Y* −‘ *A* = *Y* −‘ (*A* ∩ *range Y*)› *assms*(*3*) *assms*(*4*) *disct-fct-point-measurable measurable-sets*)
ultimately show *?thesis*
using *borel-measurable-indicator-iff borel-measurable-times* **by** *blast*
qed
moreover have *born*: *integral*^{N} *M* (λ*w*. *ennreal* (*norm* (*expl-cond-expect M Y X w* ∗ *?indA w*))) < ∞
proof −
have *integral*^{N} *M* (λ*w*. *ennreal* (*norm* (*expl-cond-expect M Y X w* ∗ *?indA w*))) =
integral^N *M* (λ*w*. *ennreal* (*expl-cond-expect M Y X w* ∗ *?indA w*))
proof −
have ∀ *w*∈ *space M*. *norm* (*expl-cond-expect M Y X w* ∗ *?indA w*) = *expl-cond-expect M Y X w* ∗ *?indA w*
using *nn-expl-cond-expect-pos* **by** (*simp add*: *nn-expl-cond-expect-pos assms*(*1*))
thus *?thesis* **by** (*metis* (*no-types*, *lifting*) *nn-integral-cong*)
qed
thus *?thesis*
by (*metis* (*no-types*, *lifting*)
‹(∑ *i*. ∫$^+$ *x*. *ennreal* (*X x* ∗ *indpre i x*) ∂*M*) = (∑ *i*. ∫$^+$ *x*. *ennreal* (*expl-cond-expect M Y X x* ∗ *indpre i x*) ∂*M*)›
‹(∑ *i*. ∫$^+$ *x*. *ennreal* (*expl-cond-expect M Y X x* ∗ *indpre i x*) ∂*M*) = (∫$^+$ *x*. *ennreal* (∑ *i*. *expl-cond-expect M Y X x* ∗ *indpre i x*) ∂*M*)›
‹(∫$^+$ *w*. (∑ *i*. *ennreal* (*X w* ∗ *indpre i w*)) ∂*M*) = (∑ *i*. ∫$^+$ *x*. *ennreal* (*X x* ∗ *indpre i x*) ∂*M*)›
‹(∫$^+$ *x*. *ennreal* (∑ *i*. *X x* ∗ *indpre i x*) ∂*M*) = (∫$^+$ *w*. (∑ *i*. *ennreal* (*X w* ∗ *indpre i w*)) ∂*M*)›
‹(∫$^+$ *x*. *ennreal* (∑ *i*. *expl-cond-expect M Y X x* ∗ *indpre i x*) ∂*M*) = (∫$^+$ *x*. *ennreal* (*expl-cond-expect M Y X x* ∗ *indicator* (*Y* −‘ (*A* ∩ *range Y*) ∩ *space M*) *x*) ∂*M*)›
‹*ennreal* (*integral*^{L} *M* (λ*w*. ∑ *i*. *X w* ∗ *indpre i w*)) = (∫$^+$ *x*. *ennreal* (∑ *i*. *X x* ∗ *indpre i x*) ∂*M*)›
ennreal-less-top infinity-ennreal-def)
qed
show *integrable M* (λ*w*. (*expl-cond-expect M Y X w*) ∗ *?indA w*)
proof (*rule iffD2*[*OF integrable-iff-bounded*])
show ((λ*w*. *expl-cond-expect M Y X w* ∗ *indicator* (*Y* −‘ (*A* ∩ *range Y*) ∩ *space M*) *w*) ∈ *borel-measurable M*) ∧
((∫$^+$ *x*. (*ennreal* (*norm* (*expl-cond-expect M Y X x* ∗ *indicator* (*Y* −‘ (*A* ∩ *range Y*) ∩ *space M*) *x*))) ∂*M*) < ∞)
proof
show (λ*w*. *expl-cond-expect M Y X w* ∗ *indicator* (*Y* −‘ (*A* ∩ *range Y*) ∩ *space M*) *w*)∈ *borel-measurable M*

using *rv* **by** *simp*
show $(\int^+ x.\ ennreal\ (norm\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ x * indicator\ (Y -` (A \cap range\ Y) \cap space\ M)\ x))\ \partial M) < \infty$
using *born* **by** *simp*
qed
qed
qed
moreover have $\forall w \in space\ M.\ 0 \leq (expl\text{-}cond\text{-}expect\ M\ Y\ X\ w) * indicator\ ((Y -` ?imA) \cap (space\ M))\ w$
by (*simp add*: *assms*(*1*) *nn-expl-cond-expect-pos*)
ultimately show *?thesis* **using** *nn-integral-eq-integral*
by (*metis* (*mono-tags*, *lifting*) *AE-I2 nn-integral-cong*)
qed
finally have *myeq*: $ennreal\ (integral^L\ M\ (\lambda w.\ (X\ w) * ?indA\ w)) = integral^L\ M\ (\lambda w.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ w) * ?indA\ w)$ **.**

thus $integrable\ M\ (\lambda w.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * indicator\ (Y -` A \cap space\ M)\ w) \wedge integral^L\ M\ (\lambda w.\ X\ w * indicator\ (Y -` A \cap space\ M)\ w) =$
$integral^L\ M\ (\lambda w.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * indicator\ (Y -` A \cap space\ M)\ w)$
proof −
have $0 \leq integral^L\ M\ (\lambda w.\ X\ w * indicator\ (Y -` A \cap space\ M)\ w)$
using ‹$Y -` A = Y -` (A \cap range\ Y)$› *b sum* **by** *fastforce*
moreover have $0 \leq integral^L\ M\ (\lambda w.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * indicator\ (Y -` A \cap space\ M)\ w)$
by (*simp add*: *assms*(*1*) *nn-expl-cond-expect-pos*)
ultimately have *expeq*: $integral^L\ M\ (\lambda w.\ X\ w * indicator\ (Y -` A \cap space\ M)\ w) =$
$integral^L\ M\ (\lambda w.\ expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * indicator\ (Y -` A \cap space\ M)\ w)$
by (*metis* (*mono-tags*, *lifting*) *Bochner-Integration.integral-cong* ‹$Y -` A = Y -` (A \cap range\ Y)$› *ennreal-inj myeq*)
have $integrable\ M\ (\lambda w.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ w) * ?indA\ w)$
proof −
have *rv*: $(\lambda w.\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ w) * indicator\ ((Y -` ?imA) \cap (space\ M))\ w) \in borel\text{-}measurable\ M$
proof −
have $expl\text{-}cond\text{-}expect\ M\ Y\ X \in borel\text{-}measurable\ M$ **using** *expl-cond-expect-borel-measurable*
using *assms* **by** *blast*
moreover have $(Y -` ?imA) \cap (space\ M) \in sets\ M$
by (*metis* ‹$A \in sets\ M'$› ‹$Y -` A = Y -` (A \cap range\ Y)$› *assms*(*3*) *assms*(*4*) *disct-fct-point-measurable measurable-sets*)
ultimately show *?thesis*
using *borel-measurable-indicator-iff borel-measurable-times* **by** *blast*
qed
moreover have *born*: $integral^N\ M\ (\lambda w.\ ennreal\ (norm\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * ?indA\ w))) < \infty$
proof −
have $integral^N\ M\ (\lambda w.\ ennreal\ (norm\ (expl\text{-}cond\text{-}expect\ M\ Y\ X\ w * ?indA$

$w))) =$
integral^N M (λw. ennreal (expl-cond-expect M Y X w $*$?indA w))
proof −
have $\forall w \in$ space M. norm (expl-cond-expect M Y X w $*$?indA w) = expl-cond-expect M Y X w $*$?indA w
using nn-expl-cond-expect-pos **by** (simp add: nn-expl-cond-expect-pos assms(1))
thus ?thesis **by** (metis (no-types, lifting) nn-integral-cong)
qed
thus ?thesis
by (metis (no-types, lifting)
‹($\sum i$. $\int^+ x$. ennreal (X x $*$ indpre i x) ∂M) = ($\sum i$. $\int^+ x$. ennreal (expl-cond-expect M Y X x $*$ indpre i x) ∂M)›
‹($\sum i$. $\int^+ x$. ennreal (expl-cond-expect M Y X x $*$ indpre i x) ∂M) = ($\int^+ x$. ennreal ($\sum i$. expl-cond-expect M Y X x $*$ indpre i x) ∂M)›
‹($\int^+ w$. ($\sum i$. ennreal (X w $*$ indpre i w)) ∂M) = ($\sum i$. $\int^+ x$. ennreal (X x $*$ indpre i x) ∂M)›
‹($\int^+ x$. ennreal ($\sum i$. X x $*$ indpre i x) ∂M) = ($\int^+ w$. ($\sum i$. ennreal (X w $*$ indpre i w)) ∂M)›
‹($\int^+ x$. ennreal ($\sum i$. expl-cond-expect M Y X x $*$ indpre i x) ∂M) = ($\int^+ x$. ennreal (expl-cond-expect M Y X x $*$ indicator (Y −‘ ($A \cap$ range Y) $\cap$ space M) x) ∂M)›
‹ennreal (integral^L M (λw. $\sum i$. X w $*$ indpre i w)) = ($\int^+ x$. ennreal ($\sum i$. X x $*$ indpre i x) ∂M)›
ennreal-less-top infinity-ennreal-def)
qed
show integrable M (λw. (expl-cond-expect M Y X w) $*$?indA w)
proof (rule iffD2[OF integrable-iff-bounded])
show ((λw. expl-cond-expect M Y X w $*$ indicator (Y −‘ ($A \cap$ range Y) $\cap$ space M) w) $\in$ borel-measurable M) $\wedge$
(($\int^+ x$. (ennreal (norm (expl-cond-expect M Y X x $*$ indicator (Y −‘ ($A \cap$ range Y) $\cap$ space M) x))) ∂M) $< \infty$)
proof
show (λw. expl-cond-expect M Y X w $*$ indicator (Y −‘ ($A \cap$ range Y) $\cap$ space M) w)$\in$ borel-measurable M
using rv **by** simp
show ($\int^+ x$. ennreal (norm (expl-cond-expect M Y X x $*$ indicator (Y −‘ ($A \cap$ range Y) $\cap$ space M) x)) ∂M) $< \infty$
using born **by** simp
qed
qed
qed
hence integrable M (λw. expl-cond-expect M Y X w $*$ indicator (Y −‘ $A \cap$ space M) w)
using ‹Y −‘ A = Y −‘ ($A \cap$ range Y)› **by** presburger
thus ?thesis **using** expeq **by** simp
qed
qed

lemma (**in** *finite-measure*) *nn-expl-cond-exp-integrable*:
assumes $\forall\ w{\in}\ space\ M.\ 0 \leq X\ w$
and *integrable M X*
and *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
shows *integrable M* (*expl-cond-expect M Y X*)
proof −
have *Y*−‘*space N* ∩ *space M* = *space M*
by (*meson assms*(*3*) *assms*(*4*) *disct-fct-point-measurable inf-absorb2 measurable-space subsetI vimageI*)
let *?indA* = *indicator* ((*Y* −‘*space N*)∩ (*space M*))::*'a*⇒*real*
have ∀ *w*∈ *space M*. (*?indA w*)= (*1*::*real*) **by** (*simp add*: ‹*Y* −‘ *space N* ∩ *space M* = *space M*›)
hence ∀ *w*∈ *space M*. ((*expl-cond-expect M Y X*) *w*) ∗ (*?indA w*) = (*expl-cond-expect M Y X*) *w* **by** *simp*
moreover have *integrable M* (λ*w*. ((*expl-cond-expect M Y X*) *w*) ∗ (*?indA w*))
using *assms*
nn-cond-expl-is-cond-exp-tmp[*of X Y N*] **by** *blast*
ultimately show *?thesis* **by** (*metis* (*no-types*, *lifting*) *Bochner-Integration.integrable-cong*)
qed

lemma (**in** *finite-measure*) *nn-cond-expl-is-cond-exp*:
assumes $\forall\ w{\in}\ space\ M.\ 0 \leq X\ w$
and *integrable M X*
and *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
shows ∀ *A* ∈ *sets N*. *integral*^{L} *M* (λ*w*. (*X w*) ∗ (*indicator* ((*Y* −‘*A*)∩ (*space M*)) *w*)) =
integral^L *M* (λ*w*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator* ((*Y* −‘*A*) ∩ (*space M*))) *w*)
by (*metis* (*mono-tags*, *lifting*) *assms nn-cond-expl-is-cond-exp-tmp*)

lemma (**in** *finite-measure*) *expl-cond-exp-integrable*:
assumes *integrable M X*
and *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
shows *integrable M* (*expl-cond-expect M Y X*)
proof −
let *?zer* = λ*w*. *0*
let *?Xp* = λ*w*. *max* (*?zer w*) (*X w*)
let *?Xn* = λ*w*. *max* (*?zer 0*) (−*X w*)
have ∀ *w*. *X w* = *?Xp w* − *?Xn w* **by** *auto*
have *ints*: *integrable M ?Xp integrable M ?Xn* **using** *integrable-max assms* **by** *auto*
hence *integrable M* (*expl-cond-expect M Y ?Xp*) **using** *assms nn-expl-cond-exp-integrable*
by (*metis max.cobounded1*)
moreover have *integrable M* (*expl-cond-expect M Y ?Xn*) **using** *ints assms*

nn-expl-cond-exp-integrable
by (*metis max.cobounded1*)
ultimately have *integr*: *integrable M* (λ*w*. (*expl-cond-expect M Y ?Xp*) *w* − (*expl-cond-expect M Y ?Xn*) *w*) **by** *auto*
have ∀ *w*∈ *space M*. (*expl-cond-expect M Y ?Xp*) *w* − (*expl-cond-expect M Y ?Xn*) *w* = (*expl-cond-expect M Y X*) *w*
proof
fix *w*
assume *w*∈ *space M*
hence (*expl-cond-expect M Y ?Xp*) *w* − (*expl-cond-expect M Y ?Xn*) *w* = (*expl-cond-expect M Y* (λ*x*. *?Xp x* − *?Xn x*)) *w*
using *ints* **by** (*simp add*: *expl-cond-exp-diff*)
also have ... = *expl-cond-expect M Y X w* **using** *expl-cond-exp-cong* ‹∀ *w*. *X w* = *?Xp w* − *?Xn w*› **by** *auto*
finally show (*expl-cond-expect M Y ?Xp*) *w* − (*expl-cond-expect M Y ?Xn*) *w* = *expl-cond-expect M Y X w* **.**
qed
thus *?thesis* **using** *integr*
by (*metis* (*mono-tags*, *lifting*) *Bochner-Integration.integrable-cong*)
qed

lemma (**in** *finite-measure*) *is-cond-exp*:
assumes *integrable M X*
and *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
shows ∀ *A* ∈ *sets N*. *integral*^{L} *M* (λ*w*. (*X w*) ∗ (*indicator* ((*Y* −‘*A*)∩ (*space M*)) *w*)) =
integral^L *M* (λ*w*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator* ((*Y* −‘*A*) ∩ (*space M*))) *w*)
proof −
let *?zer* = λ*w*. *0*
let *?Xp* = λ*w*. *max* (*?zer w*) (*X w*)
let *?Xn* = λ*w*. *max* (*?zer 0*) (−*X w*)
have ∀ *w*. *X w* = *?Xp w* − *?Xn w* **by** *auto*
have *ints*: *integrable M ?Xp integrable M ?Xn* **using** *integrable-max assms* **by** *auto*
hence *posint*: *integrable M* (*expl-cond-expect M Y ?Xp*) **using** *assms nn-expl-cond-exp-integrable*
by (*metis max.cobounded1*)
have *negint*: *integrable M* (*expl-cond-expect M Y ?Xn*) **using** *ints assms nn-expl-cond-exp-integrable*
by (*metis max.cobounded1*)
have *eqp*: ∀ *A* ∈ *sets N*. *integral*^{L} *M* (λ*w*. (*?Xp w*) ∗ (*indicator* ((*Y* −‘*A*)∩ (*space M*)) *w*)) =
integral^L *M* (λ*w*. ((*expl-cond-expect M Y ?Xp*) *w*) ∗ (*indicator* ((*Y* −‘*A*) ∩ (*space M*))) *w*)
using *nn-cond-expl-is-cond-exp*[*of ?Xp Y N*] *assms* **by** *auto*
have *eqn*: ∀ *A* ∈ *sets N*. *integral*^{L} *M* (λ*w*. (*?Xn w*) ∗ (*indicator* ((*Y* −‘*A*)∩ (*space M*)) *w*)) =
integral^L *M* (λ*w*. ((*expl-cond-expect M Y ?Xn*) *w*) ∗ (*indicator* ((*Y* −‘*A*) ∩

(*space M*))) *w*)
using *nn-cond-expl-is-cond-exp*[*of ?Xn Y N*] *assms* **by** *auto*

show ∀ *A* ∈ *sets N*. *integral*^{L} *M* (λ*w*. (*X w*) ∗ (*indicator* ((*Y* −‘*A*)∩ (*space M*)) *w*)) =
integral^L *M* (λ*w*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator* ((*Y* −‘*A*) ∩ (*space M*))) *w*)
proof
fix *A*
assume *A*∈ *sets N*
let *?imA* = *A* ∩ (*range Y*)
have *countable ?imA* **using** *assms disc-fct-def* **by** *blast*
have *Y* −‘*A* = *Y* −‘*?imA* **by** *auto*
have *yev*: *Y* −‘ (*A*∩ *range Y*) ∩ *space M* ∈ *sets M*
using ‹*A* ∈ *sets N*› *assms*(*3*) *assms*(*2*) *disct-fct-point-measurable measurable-sets*
by (*metis* ‹*Y* −‘ *A* = *Y* −‘ (*A* ∩ *range Y*)›)
let *?indA* = *indicator* ((*Y* −‘(*A* ∩ *range Y*))∩ (*space M*))::′*a*⇒*real*
have *intp*: *integrable M* (λ*w*. (*?Xp w*) ∗ *?indA w*)
proof (*rule integrable-real-mult-indicator*)
show *Y* −‘ (*A*∩ *range Y*) ∩ *space M* ∈ *sets M* **using** *yev* **by** *simp*
show *integrable M ?Xp* **using** *assms* **by** *simp*
qed
have *intn*: *integrable M* (λ*w*. (*?Xn w*) ∗ *?indA w*)
proof (*rule integrable-real-mult-indicator*)
show *Y* −‘ (*A*∩ *range Y*) ∩ *space M* ∈ *sets M* **using** *yev* **by** *simp*
show *integrable M ?Xn* **using** *assms* **by** *simp*
qed
have *exintp*: *integrable M* (λ*w*. (*expl-cond-expect M Y ?Xp w*) ∗ *?indA w*)
proof (*rule integrable-real-mult-indicator*)
show *Y* −‘ (*A*∩ *range Y*) ∩ *space M* ∈ *sets M* **using** *yev* **by** *simp*
show *integrable M* (*expl-cond-expect M Y ?Xp*) **using** *posint* **by** *simp*
qed
have *exintn*: *integrable M* (λ*w*. (*expl-cond-expect M Y ?Xn w*) ∗ *?indA w*)
proof (*rule integrable-real-mult-indicator*)
show *Y* −‘ (*A*∩ *range Y*) ∩ *space M* ∈ *sets M* **using** *yev* **by** *simp*
show *integrable M* (*expl-cond-expect M Y ?Xn*) **using** *negint* **by** *simp*
qed
have *integral*^{L} *M* (λ*w*. *X w* ∗ *indicator* (*Y* −‘ *A* ∩ *space M*) *w*) =
integral^L *M* (λ*w*. (*?Xp w* − *?Xn w*) ∗ *indicator* (*Y* −‘ *A* ∩ *space M*) *w*)
using ‹∀ *w*. *X w* =*?Xp w* − *?Xn w*› **by** *auto*
also have ... = *integral*^{L} *M* (λ*w*. (*?Xp w* ∗ *indicator* (*Y* −‘ *A* ∩ *space M*) *w*) − *?Xn w* ∗ *indicator* (*Y* −‘ *A* ∩ *space M*) *w*)
by (*simp add*: *left-diff-distrib*)
also have ... = *integral*^{L} *M* (λ*w*. (*?Xp w* ∗ *indicator* (*Y* −‘ *A* ∩ *space M*) *w*)) −
integral^L *M* (λ*w*. *?Xn w* ∗ *indicator* (*Y* −‘ *A* ∩ *space M*) *w*)
using ‹*Y* −‘ *A* = *Y* −‘ (*A* ∩ *range Y*)› *intp intn* **by** *auto*
also have ... = *integral*^{L} *M* (λ*w*. ((*expl-cond-expect M Y ?Xp*) *w*) ∗ (*indicator*

$((Y - `A) \cap (space\ M))))\ w) -$
$integral^L\ M\ (\lambda w.\ ((expl\text{-}cond\text{-}expect\ M\ Y\ ?Xn)\ w) * (indicator\ ((Y - `A) \cap (space\ M)))\ w)$
using *eqp eqn* **by** (*simp add*: ‹$A \in sets\ N$›)
also have ... = $integral^L\ M\ (\lambda w.\ ((expl\text{-}cond\text{-}expect\ M\ Y\ ?Xp)\ w) * (indicator\ ((Y - `A) \cap (space\ M)))\ w -$
$((expl\text{-}cond\text{-}expect\ M\ Y\ ?Xn)\ w) * (indicator\ ((Y - `A) \cap (space\ M)))\ w)$
using ‹$Y - `A = Y - `(A \cap range\ Y)$› *exintn exintp* **by** *auto*
also have ... = $integral^L\ M\ (\lambda w.\ ((expl\text{-}cond\text{-}expect\ M\ Y\ ?Xp)\ w - (expl\text{-}cond\text{-}expect\ M\ Y\ ?Xn)\ w) * (indicator\ ((Y - `A) \cap (space\ M)))\ w)$
by (*simp add*: *left-diff-distrib*)
also have ... = $integral^L\ M\ (\lambda w.\ ((expl\text{-}cond\text{-}expect\ M\ Y\ (\lambda x.\ ?Xp\ x - ?Xn\ x)\ w) * (indicator\ ((Y - `A) \cap (space\ M)))\ w))$
using *expl-cond-exp-diff*[*of M ?Xp ?Xn Y*] *ints* **by** (*metis* (*mono-tags*, *lifting*) *Bochner-Integration.integral-cong*)
also have ... = $integral^L\ M\ (\lambda w.\ ((expl\text{-}cond\text{-}expect\ M\ Y\ X\ w) * (indicator\ ((Y - `A) \cap (space\ M)))\ w))$
using ‹$\forall w.\ X\ w = ?Xp\ w - ?Xn\ w$› *expl-cond-exp-cong*[*of M X* $\lambda x.$ *?Xp x - ?Xn x Y*] **by** *presburger*
finally show $integral^L\ M\ (\lambda w.\ X\ w * indicator\ (Y - `A \cap space\ M)\ w) = integral^L\ M\ (\lambda w.\ ((expl\text{-}cond\text{-}expect\ M\ Y\ X\ w) * (indicator\ ((Y - `A) \cap (space\ M)))\ w))$ **.**
qed
qed

lemma (**in** *finite-measure*) *charact-cond-exp*:
assumes *disc-fct Y*
and *integrable M X*
and *point-measurable M* (*space N*) *Y*
and $Y \in space\ M \to space\ N$
and $\forall r \in range\ Y \cap space\ N.\ \exists A \in sets\ N.\ range\ Y \cap A = \{r\}$
shows $AE\ w\ in\ M.\ real\text{-}cond\text{-}exp\ M\ (fct\text{-}gen\text{-}subalgebra\ M\ N\ Y)\ X\ w = expl\text{-}cond\text{-}expect\ M\ Y\ X\ w$
proof (*rule sigma-finite-subalgebra.real-cond-exp-charact*)
have $Y \in measurable\ M\ N$
by (*simp add*: *assms*(*1*) *assms*(*3*) *disct-fct-point-measurable*)
have *point-measurable M* (*space N*) *Y* **by** (*simp add*: *assms*(*3*))
show *sigma-finite-subalgebra M* (*fct-gen-subalgebra M N Y*) **unfolding** *sigma-finite-subalgebra-def*
proof
show *subalgebra M* (*fct-gen-subalgebra M N Y*) **using** ‹$Y \in measurable\ M\ N$›
by (*simp add*: *fct-gen-subalgebra-is-subalgebra*)
show *sigma-finite-measure* (*restr-to-subalg M* (*fct-gen-subalgebra M N Y*))
unfolding *sigma-finite-measure-def*
proof (*intro exI conjI*)
let *?A* = {*space M*}
show *countable ?A* **by** *simp*
show *?A* $\subseteq$ *sets* (*restr-to-subalg M* (*fct-gen-subalgebra M N Y*))
by (*metis empty-subsetI insert-subset sets.top space-restr-to-subalg*)

show $\bigcup$ *?A* = *space* (*restr-to-subalg M* (*fct-gen-subalgebra M N Y*))
by (*simp add*: *space-restr-to-subalg*)
show ∀ *a*∈{*space M*}. *emeasure* (*restr-to-subalg M* (*fct-gen-subalgebra M N Y*)) *a* ≠ ∞
by (*metis* ‹*subalgebra M* (*fct-gen-subalgebra M N Y*)› *emeasure-finite emeasure-restr-to-subalg infinity-ennreal-def sets.top singletonD subalgebra-def*)
qed
qed
show *integrable M X* **using** *assms* **by** *simp*
show *expl-cond-expect M Y X* ∈ *borel-measurable* (*fct-gen-subalgebra M N Y*) **using** *assms* **by** (*simp add*:*expl-cond-exp-borel*)
show *integrable M* (*expl-cond-expect M Y X*)
using *assms expl-cond-exp-integrable* **by** *blast*
have ∀ *A*∈ *sets M*. *integral*^{L} *M* (λ*w*. (*X w*) ∗ (*indicator A w*)) = *set-lebesgue-integral M A X*
by (*metis* (*no-types*, *lifting*) *Bochner-Integration.integral-cong mult-commute-abs real-scaleR-def set-lebesgue-integral-def*)
have ∀ *A*∈ *sets M*. *integral*^{L} *M* (λ*w*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator A w*)) = *set-lebesgue-integral M A* (*expl-cond-expect M Y X*)
by (*metis* (*no-types*, *lifting*) *Bochner-Integration.integral-cong mult-commute-abs real-scaleR-def set-lebesgue-integral-def*)
have ∀ *A*∈ *sets* (*fct-gen-subalgebra M N Y*). *integral*^{L} *M* (λ*w*. (*X w*) ∗ (*indicator A w*)) =
integral^L *M* (λ*w*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator A w*))
proof
fix *A*
assume *A* ∈ *sets* (*fct-gen-subalgebra M N Y*)
hence *A* ∈ {*Y* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets N*} **using** *assms* **by** (*simp add*:*fct-gen-subalgebra-sigma-sets*)
hence ∃ *B* ∈ *sets N*. *A* = *Y* −‘*B* ∩ *space M* **by** *auto*
from *this* **obtain** *B* **where** *B*∈ *sets N* **and** *A* = *Y* −‘*B*∩ *space M* **by** *auto*
thus *integral*^{L} *M* (λ*w*. (*X w*) ∗ (*indicator A w*)) =
integral^L *M* (λ*w*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator A w*)) **using** *is-cond-exp*
using *Bochner-Integration.integral-cong* ‹*point-measurable M* (*space N*) *Y*› *assms*(*1*) *assms*(*2*) **by** *blast*
qed
thus ⋀*A*. *A* ∈ *sets* (*fct-gen-subalgebra M N Y*) ⟹ *set-lebesgue-integral M A X* = *set-lebesgue-integral M A* (*expl-cond-expect M Y X*)
by (*smt Bochner-Integration.integral-cong Groups.mult-ac*(*2*) *real-scaleR-def set-lebesgue-integral-def*)
qed

lemma (**in** *finite-measure*) *charact-cond-exp'*:
assumes *disc-fct Y*
and *integrable M X*
and *Y*∈ *measurable M N*
and ∀ *r*∈ *range Y*∩ *space N*. ∃ *A*∈ *sets N*. *range Y*∩ *A* = {*r*}

shows *AE w in M. real-cond-exp M (fct-gen-subalgebra M N Y) X w = expl-cond-expect M Y X w*
proof (*rule charact-cond-exp*)
show *disc-fct Y* **using** *assms* **by** *simp*
show *integrable M X* **using** *assms* **by** *simp*
show $\forall r \in$*range Y* $\cap$ *space N*. $\exists A \in$*sets N*. *range Y* $\cap$ *A* = $\{r\}$ **using** *assms* **by** *simp*
show *Y* $\in$ *space M* $\rightarrow$ *space N*
by (*meson Pi-I assms*(*3*) *measurable-space*)
show *point-measurable M* (*space N*) *Y* **using** *assms* **by** (*simp add*: *meas-single-meas*)
qed

end

5 Infinite coin toss space

This section contains the formalization of the infinite coin toss space, i.e., the probability space constructed on infinite sequences of independent coin tosses.

theory *Infinite-Coin-Toss-Space* **imports** *Filtration Generated-Subalgebra Disc-Cond-Expect*

begin

5.1 Preliminary results

lemma *decompose-init-prod*:
fixes *n*::*nat*
shows ($\prod$ *i*$\in$ {*0*..*n*}. *f i*) = *f 0* $*$ ($\prod$ *i*$\in$ {*1*..*n*}. *f i*)
proof (*cases Suc 0* $\leq$ *n*)
case *True*
thus *?thesis*
by (*metis One-nat-def Suc-le-D True prod.atLeast0-atMost-Suc-shift prod.atLeast-Suc-atMost-Suc-shift*)
next
case *False*
thus *?thesis*
by (*metis One-nat-def atLeastLessThanSuc-atLeastAtMost prod.atLeast0-lessThan-Suc-shift prod.atLeast-Suc-lessThan-Suc-shift*)
qed

lemma *Inter-nonempty-distrib*:
assumes *A* $\neq$ {}
shows $\bigcap$ *A* $\cap$ *B* = ($\bigcap$ *C*$\in$ *A*. (*C*$\cap$ *B*))
proof
show ($\bigcap$ *C*$\in$*A*. *C* $\cap$ *B*) $\subseteq$ $\bigcap$ *A* $\cap$ *B*
proof

fix x
assume *mem*: $x \in (\bigcap C{\in}A.\ C \cap B)$
from ‹$A \neq \{\}$› **obtain** C **where** $C{\in}\ A$ **by** *blast*
hence $x{\in}\ C{\cap}\ B$ **using** *mem* **by** *blast*
hence *in1*: $x{\in}\ B$ **by** *auto*
have $\bigwedge C.\ C{\in}\ A \Longrightarrow x \in C{\cap}B$ **using** *mem* **by** *blast*
hence $\bigwedge C.\ C{\in}\ A \Longrightarrow x{\in}\ C$ **by** *auto*
hence *in2*: $x{\in} \bigcap A$ **by** *auto*
thus $x{\in} \bigcap A \cap B$ **using** *in1 in2* **by** *simp*
qed
qed *auto*

lemma *enn2real-sum*: **shows** *finite* $A \Longrightarrow (\bigwedge a.\ a{\in}\ A{\Longrightarrow}\ f\ a < top) \Longrightarrow$*enn2real* (*sum* f A) = ($\sum$ $a{\in}$ A. *enn2real* (f a))
proof (*induct rule*:*finite-induct*)
case *empty*
thus *?case* **by** *simp*
next
case (*insert* a A)
have *enn2real* (*sum* f (*insert* a A)) = *enn2real* (f a + (*sum* f A))
by (*simp add*: *insert.hyps*(*1*) *insert.hyps*(*2*))
also have ... = *enn2real* (f a) + *enn2real* (*sum* f A)
by (*simp add*: *enn2real-plus insert.hyps*(*1*) *insert.prems*)
also have ... = *enn2real* (f a) + ($\sum$ $a{\in}$ A. *enn2real* (f a))
by (*simp add*: *insert.hyps*(*3*) *insert.prems*)
also have ... = ($\sum$ $a{\in}$ (*insert* a A)). *enn2real* (f a))
by (*metis calculation insert.hyps*(*1*) *insert.hyps*(*2*) *sum.insert*)
finally show *?case* **.**
qed

lemma *ennreal-sum*: **shows** *finite* $A \Longrightarrow (\bigwedge a.\ 0 \leq f\ a) \Longrightarrow (\sum a{\in}\ A.\ ennreal\ (f\ a)) = ennreal\ (\sum a{\in}\ A.\ f\ a)$
proof (*induct rule*:*finite-induct*)
case *empty*
thus *?case* **by** *simp*
next
case (*insert* a A)
have ($\sum a{\in}$ (*insert* a A)). *ennreal* (f a)) = *ennreal* (f a) + ($\sum a{\in}$ A. *ennreal* (f a))
by (*simp add*: *insert.hyps*(*1*) *insert.hyps*(*2*))
also have ... = *ennreal* (f a) + *ennreal* ($\sum a{\in}$ A. f a)
by (*simp add*: *insert.prems*)
also have ... = *ennreal* (f a + ($\sum a{\in}$ A. f a))
by (*simp add*: *insert.prems sum-nonneg*)
also have ... = *ennreal* ($\sum a{\in}$ (*insert* a A)). (f a))
using *insert.hyps*(*1*) *insert.hyps*(*2*) **by** *auto*
finally show *?case* **.**

qed

lemma *stake-snth*:
assumes *stake n w = stake n x*
shows *Suc i ≤ n ⟹ snth w i = snth x i*
by (*metis Suc-le-eq assms stake-nth*)

lemma *stake-snth-charact*:
assumes *stake n w = stake n x*
shows *∀ i < n. snth w i = snth x i*
proof (*intro allI impI*)
fix *i*
assume *i < n*
thus *snth w i = snth x i* **using** *Suc-leI assms stake-snth* **by** *blast*
qed

lemma *stake-snth′*:
shows (*⋀i. Suc i ≤ n ⟹ snth w i = snth x i*) *⟹stake n w = stake n x*
proof (*induct n arbitrary:w x*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
hence *mh*: *⋀i. Suc i ≤ Suc n ⟹ w !! i = x !! i* **by** *auto*
hence *seq*:*snth w n = snth x n* **by** *auto*
have *stake n w = stake n x* **using** *mh Suc* **by** (*meson Suc-leD Suc-le-mono*)
thus *stake* (*Suc n*) *w = stake* (*Suc n*) *x* **by** (*metis seq stake-Suc*)
qed

lemma *stake-inter-snth*:
fixes *x*
assumes *Suc 0 ≤ n*
shows {*w∈ space M*. (*stake n w = stake n x*)} = (⋂ *i* ∈ {*0*.. *n−1*}. {*w∈ space M*. (*snth w i = snth x i*)})
proof
let *?S* = {*w∈ space M*. (*stake n w = stake n x*)}
show *?S* ⊆ (⋂*i*∈{*0..n−1*}. {*w* ∈ *space M*. *w !! i = x !! i*}) **using** *stake-snth assms* **by** *fastforce*
show (⋂*i*∈{*0..n−1*}. {*w* ∈ *space M*. *w !! i = x !! i*}) ⊆ *?S*
proof
fix *w*
assume *inter*: *w* ∈ (⋂*i*∈{*0..n−1*}. {*w* ∈ *space M*. *w !! i = x !! i*})
have *∀ i*. *0 ≤ i ∧ i ≤ n−1 ⟶ snth w i = snth x i*
proof (*intro allI impI*)
fix *i*
assume *0 ≤ i ∧ i ≤ n−1*
thus *snth w i = snth x i* **using** *inter* **by** *auto*
qed

hence *stake n w = stake n x*
by (*metis One-nat-def Suc-le-D diff-Suc-Suc diff-is-0-eq diff-zero le0 stake-snth'*)
thus *w∈ ?S* **using** *inter* **by** *auto*
qed
qed

lemma *streams-stake-set*:
shows *pw ∈ streams A ⟹ set (stake n pw) ⊆ A*
proof (*induct n arbitrary*: *pw*)
case (*Suc n*) **note** *hyp = this*
have *set (stake (Suc 0) pw) ⊆ A*
proof −
have *shd pw ∈ A* **using** *hyp streams-shd*[*of pw A*] **by** *simp*
have *stake (Suc 0) pw = [shd pw]* **by** *auto*
hence *set (stake (Suc 0) pw) = {shd pw}* **by** *auto*
thus *?thesis* **using** ‹*shd pw ∈ A*› **by** *auto*
qed
thus *?case* **by** (*simp add*: *Suc.hyps Suc.prems streams-stl*)
qed *simp*

lemma *stake-finite-universe-induct*:
assumes *finite A*
and *A ≠ {}*
shows (*stake (Suc n) ‘(streams A)) = {s#w| s w. s∈ A ∧ w∈ (stake n ‘(streams A))}* (**is** *?L = ?R*)
proof
show *?L ⊆ ?R*
proof
fix *l*::*'a list*
assume *l∈ ?L*
hence ∃*pw. pw ∈ streams A ∧ l = stake (Suc n) pw* **by** *auto*
from *this* **obtain** *pw* **where** *pw ∈ streams A* **and** *l = stake (Suc n) pw* **by** *blast*
hence *l = shd pw # stake n (stl pw)* **unfolding** *stake-def* **by** *auto*
thus *l∈ ?R* **by** (*simp add*: ‹*pw ∈ streams A*› *streams-shd streams-stl*)
qed
show *?R ⊆ ?L*
proof
fix *l*::*'a list*
assume *l∈ ?R*
hence ∃ *s w. s∈ A ∧ w∈ (stake n ‘(streams A)) ∧ l = s# w* **by** *auto*
from *this* **obtain** *s* **and** *w* **where** *s∈ A* **and** *w∈ (stake n ‘(streams A))* **and** *l = s# w* **by** *blast*
note *swprop = this*
have ∃*pw. pw ∈ streams A ∧ w = stake n pw* **using** *swprop* **by** *auto*
from *this* **obtain** *pw* **where** *pw∈ streams A* **and** *w = stake n pw* **by** *blast*
note *pwprop = this*
have *l ∈ lists A*

```
    proof −
      have s∈ A using swprop by simp
      have set w ⊆ A using pwprop streams-stake-set by simp
      have set l = set w ∪ {s} using swprop by auto
      thus ?thesis using ‹s∈ A› ‹set w ⊆ A› by auto
    qed
    have ∃ x. x ∈ A using assms by auto
    from this obtain x where x∈ A by blast
    let ?sx = sconst x
    let ?st = shift l ?sx
    have l = stake (Suc n) ?st by (simp add: pwprop(2) stake-shift swprop(3))
    have sset ?sx = {x} by simp
    hence sset ?sx ⊆ A using ‹x∈ A› by simp
    hence ?sx ∈ streams A using sset-streams[of sconst x] by simp
     hence ?st ∈ streams A using ‹l ∈ lists A› shift-streams[of l A sconst x] by
simp
    thus l∈ ?L using ‹l = stake (Suc n) ?st› by blast
  qed
qed


lemma stake-finite-universe-finite:
  assumes finite A
  and A ≠ {}
  shows finite (stake n `(streams A))
proof (induction n)
  let ?L = (stake 0 `(streams A))
  have streams A ≠ {}
  proof
    assume streams A = {}
    have ∃ x. x ∈ A using assms by auto
    from this obtain x where x∈ A by blast
    let ?sx = sconst x
    have sset ?sx = {x} by simp
    hence sset ?sx ⊆ A using ‹x∈ A› by simp
    hence ?sx ∈ streams A using sset-streams[of sconst x] by simp
    thus False using ‹streams A = {}› by simp
  qed
  have stake 0 = (λs. []) unfolding stake-def by simp
  hence ?L = {[]} using ‹streams A ≠ {}› by auto
 show finite (stake 0 `(streams A)) by (simp add: ‹?L = {[]}› image-constant-conv)
next
  fix n assume finite (stake n `(streams A)) note hyp = this
  have (stake (Suc n) `(streams A)) = {s#w| s w. s∈ A ∧ w∈ (stake n `(streams
A))} (is ?L = ?R)
  using assms stake-finite-universe-induct[of A n] by simp
  have finite ?R  by (simp add: assms(1) finite-image-set2 hyp)
  thus finite ?L using ‹?L = ?R›by simp
qed
```

lemma *diff-streams-only-if*:
assumes $w \neq x$
shows $\exists n.\ snth\ w\ n \neq snth\ x\ n$
proof −
have *f1*: $smap\ (\lambda n.\ x\ !!\ (n - Suc\ 0))\ (fromN\ (Suc\ 0)) \neq w$
by (*metis assms stream-smap-fromN*)
obtain $nn :: 'a\ stream \Rightarrow nat\ stream \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat$ **where**
$\forall x0\ x1\ x2.\ (\exists v3.\ x2\ (x1\ !!\ v3) \neq x0\ !!\ v3) = (x2\ (x1\ !!\ nn\ x0\ x1\ x2) \neq x0\ !!\ nn\ x0\ x1\ x2)$
by *moura*
then have $x\ !!\ (fromN\ (Suc\ 0)\ !!\ nn\ w\ (fromN\ (Suc\ 0))\ (\lambda n.\ x\ !!\ (n - Suc\ 0)) - Suc\ 0) \neq w\ !!\ nn\ w\ (fromN\ (Suc\ 0))\ (\lambda n.\ x\ !!\ (n - Suc\ 0))$
using *f1* **by** (*meson smap-alt*)
then show *?thesis*
by (*metis* (*no-types*) *snth-smap stream-smap-fromN*)
qed

lemma *diff-streams-if*:
assumes $\exists n.\ snth\ w\ n \neq snth\ x\ n$
shows $w \neq x$
using *assms* **by** *auto*

lemma *sigma-set-union-count*:
assumes $\forall\ y \in A.\ B\ y \in sigma\text{-}sets\ X\ Y$
and *countable A*
shows $(\bigcup\ y \in A.\ B\ y) \in sigma\text{-}sets\ X\ Y$
by (*metis* (*mono-tags*, *lifting*) *assms countable-image imageE sigma-sets-UNION*)

lemma *sigma-set-inter-init*:
assumes $\bigwedge i.\ i \leq (n::nat) \Longrightarrow A\ i \in sigma\text{-}sets\ sp\ B$
and $B \subseteq Pow\ sp$
shows $(\bigcap\ i \in \{m.\ m \leq n\}.\ A\ i) \in sigma\text{-}sets\ sp\ B$
by (*metis* (*full-types*) *assms*(*1*) *assms*(*2*) *bot.extremum empty-iff mem-Collect-eq sigma-sets-INTER*)

lemma *adapt-sigma-sets*:
assumes $\bigwedge i.\ i \leq n \Longrightarrow (X\ i) \in measurable\ M\ N$
shows $sigma\text{-}algebra\ (space\ M)\ (sigma\text{-}sets\ (space\ M)\ (\bigcup i \in \{m.\ m \leq n\}.\ \{X\ i\ -`\ A \cap space\ M\ |A.\ A \in sets\ N\}))$
proof (*rule sigma-algebra-sigma-sets*)
show $(\bigcup i \in \{m.\ m \leq n\}.\ \{X\ i\ -`\ A \cap space\ M\ |A.\ A \in sets\ N\}) \subseteq Pow\ (space\ M)$
proof (*rule UN-subset-iff*[*THEN iffD2*], *intro ballI*)
fix *i*
assume $i \in \{m.\ m \leq n\}$

show $\{X\ i\ -`\ A \cap space\ M\ |A.\ A \in sets\ N\} \subseteq Pow\ (space\ M)$ **by** *auto*
qed
qed

5.2 Bernoulli streams

Bernoulli streams represent the formal definition of the infinite coin toss space. The parameter p represents the probability of obtaining a head after a coin toss.

definition *bernoulli-stream*::*real* ⇒ (*bool stream*) *measure* **where**
bernoulli-stream p = *stream-space* (*measure-pmf* (*bernoulli-pmf p*))

lemma *bernoulli-stream-space*:
assumes N = *bernoulli-stream p*
shows *space N* = *streams UNIV*::*bool*
using *assms* **unfolding** *bernoulli-stream-def stream-space-def*
by (*simp add*: *assms bernoulli-stream-def space-stream-space*)

lemma *bernoulli-stream-preimage*:
assumes N = *bernoulli-stream p*
shows *f* −` *A* ∩ (*space N*) = *f*−`*A*
using *assms* **by** (*simp add*: *bernoulli-stream-space*)

lemma *bernoulli-stream-component-probability*:
assumes N = *bernoulli-stream p* **and** $0 \leq p$ **and** $p \leq 1$
shows ∀ *n*. *emeasure N* {*w*∈ *space N*. (*snth w n*)} = *p*
proof
have *prob-space N* **using** *assms* **by** (*simp add*: *bernoulli-stream-def prob-space.prob-space-stream-space prob-space-measure-pmf*)
fix *n*::*nat*
let *?S* = {*w*∈ *space N*. (*snth w n*)}
have *s*: *?S* ∈ *sets N*
proof −
have (λ*w*. *snth w n*) ∈ *measurable N* (*measure-pmf* (*bernoulli-pmf p*)) **using** *assms* **by** (*simp add*: *bernoulli-stream-def*)
moreover have {*True*} ∈ *sets* (*measure-pmf* (*bernoulli-pmf p*)) **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
let *?PM* = (λ*i*::*nat*. (*measure-pmf* (*bernoulli-pmf p*)))
have *isps*: *product-prob-space ?PM* **by** *unfold-locales*
let *?Z* = {*X*::*nat*⇒*bool*. *X n* = *True*}
let *?wPM* = Pi_M *UNIV ?PM*
have *space ?wPM* = *UNIV* **using** *space-PiM* **by** *fastforce*
hence (*to-stream* −` *?S* ∩ (*space ?wPM*)) = *to-stream* −` *?S* **by** *simp*
also have ... = *?Z* **using** *assms* **by** (*simp add*:*bernoulli-stream-space to-stream-def*)
also have ... = *prod-emb UNIV ?PM* {*n*} (Pi_E {*n*} (λ*x*::*nat*. {*True*}))
proof
{

```
    fix X
    assume X ∈ prod-emb UNIV ?PM {n} (Pi_E {n} (λx::nat. {True}))
     hence restrict X {n} ∈ (Pi_E {n} (λx::nat. {True})) using prod-emb-iff[of
X] by simp
    hence X n = True
      unfolding PiE-iff by auto
    hence X ∈ ?Z by simp
   }
   thus prod-emb UNIV ?PM {n} (Pi_E {n} (λx::nat. {True})) ⊆ ?Z by auto
   {
    fix X
    assume X ∈ ?Z
    hence X n = True by simp
    hence restrict X {n} ∈ (Pi_E {n} (λx::nat. {True}))
      unfolding PiE-iff by auto
    moreover have X ∈ extensional UNIV by simp
    moreover have ∀ i ∈ UNIV. X i ∈ space (?PM i) by auto
   ultimately have X ∈ prod-emb UNIV ?PM {n} (Pi_E {n} (λx::nat. {True}))
using prod-emb-iff[of X] by simp
   }
   thus ?Z ⊆ prod-emb UNIV ?PM {n} (Pi_E {n} (λx::nat. {True})) by auto
  qed
  finally have inteq: (to-stream −‘ ?S ∩ (space ?wPM)) = prod-emb UNIV ?PM
{n} (Pi_E {n} (λx::nat. {True})) .
  have emeasure N ?S = emeasure ?wPM (to-stream −‘ ?S ∩ (space ?wPM))
   using assms emeasure-distr[of to-stream ?wPM (vimage-algebra (streams (space
(measure-pmf (bernoulli-pmf p)))) (!!)
       (Pi_M UNIV (λi. measure-pmf (bernoulli-pmf p)))) ?S] measurable-to-stream[of
(measure-pmf (bernoulli-pmf p))] s
    unfolding bernoulli-stream-def stream-space-def by auto
  also have ... = emeasure ?wPM (prod-emb UNIV ?PM {n} (Pi_E {n} (λx::nat.
{True}))) using inteq by simp
  also have ... =
    (∏ i∈{n}. emeasure (?PM i) ((λx::nat. {True}) i)) using isps
   by (auto simp add: product-prob-space.emeasure-PiM-emb simp del: ext-funcset-to-sing-iff)
  also have ... = emeasure (?PM n) {True} by simp
  also have ... = p using assms by (simp add: emeasure-pmf-single)
  finally show emeasure N ?S = p .
qed


lemma bernoulli-stream-component-probability-compl:
  assumes N = bernoulli-stream p and 0 ≤ p and p ≤ 1
  shows ∀ n. emeasure N {w∈ space N. ¬(snth w n)} = 1− p
proof
  fix n
  let ?A = {w ∈ space N. ¬ w !! n}
  let ?B = {w ∈ space N. w !! n}
  have ?A ∪ ?B = space N by auto
```

have *?A∩?B = {}* **by** *auto*
hence *eqA*: *?A = (?A∪ ?B) − ?B* **using** *Diff-cancel* **by** *blast*
moreover have *?A ∈ sets N*
proof −
have *(λw. snth w n) ∈ measurable N (measure-pmf (bernoulli-pmf p))* **using** *assms* **by** *(simp add: bernoulli-stream-def)*
moreover have *{True} ∈ sets (measure-pmf (bernoulli-pmf p))* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
moreover have *?B ∈ sets N*
proof −
have *(λw. snth w n) ∈ measurable N (measure-pmf (bernoulli-pmf p))* **using** *assms* **by** *(simp add: bernoulli-stream-def)*
moreover have *{True} ∈ sets (measure-pmf (bernoulli-pmf p))* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
ultimately have *emeasure N ((?A∪ ?B) − ?B) = emeasure N (?A∪ ?B) − emeasure N ?B*
proof −
have *f1*: *⋀S m. (S::bool stream set) ∉ sets m ∨ emeasure m S = ⊤ ∨ emeasure m (space m) − emeasure m S = emeasure m (space m − S)*
by *(metis emeasure-compl infinity-ennreal-def)*
have *emeasure N {s ∈ space N. s !! n} ≠ ⊤*
using *assms(1) assms(2) assms(3) ennreal-neq-top bernoulli-stream-component-probability* **by** *presburger*
then have *emeasure N (space N) − emeasure N {s ∈ space N. s !! n} = emeasure N (space N − {s ∈ space N. s !! n})*
using *f1 ‹{w ∈ space N. w !! n} ∈ sets N›* **by** *blast*
then show *?thesis*
using *‹{w ∈ space N. ¬ w !! n} ∪ {w ∈ space N. w !! n} = space N›* **by** *presburger*
qed
hence *emeasure N ?A = emeasure N (?A∪ ?B) − emeasure N ?B* **using** *eqA* **by** *simp*
also have *... = 1 − emeasure N ?B*
by *(metis (no-types, lifting) ‹?A ∪ ?B = space N› assms(1) bernoulli-stream-def prob-space.emeasure-space-1 prob-space.prob-space-stream-space prob-space-measure-pmf)*
also have *... = 1 − p* **using** *bernoulli-stream-component-probability[of N p] assms*
by *(metis (mono-tags) ennreal-1 ennreal-minus-if)*
finally show *emeasure N ?A = 1 − p* **.**
qed

lemma *bernoulli-stream-sets*:
assumes *0 < q*
and *q < 1*
and *0 < p*
and *p < 1*
shows *sets (bernoulli-stream p) = sets (bernoulli-stream q)* **unfolding** *bernoulli-stream-def*
by *(rule sets-stream-space-cong, simp)*

locale *infinite-coin-toss-space* =
  fixes *p*::*real* **and** *M*::*bool stream measure*
  assumes *p-gt-0*: *0* ≤ *p*
  and *p-lt-1*: *p* ≤ *1*
  and *bernoulli*: *M* = *bernoulli-stream p*

sublocale *infinite-coin-toss-space* ⊆ *prob-space*
by (*simp add*: *bernoulli bernoulli-stream-def prob-space.prob-space-stream-space prob-space-measure-pmf*)

5.3 Natural filtration on the infinite coin toss space

The natural filtration on the infinite coin toss space is the discrete filtration F such that $F\ n$ represents the restricted measure space in which the outcome of the first n coin tosses is known.

5.3.1 The projection function

Intuitively, the restricted measure space in which the outcome of the first n coin tosses is known can be defined by any measurable function that maps all infinite sequences that agree on the first n coin tosses to the same element.

definition (**in** *infinite-coin-toss-space*) *pseudo-proj-True*:: *nat* ⇒ *bool stream* ⇒ *bool stream* **where**
  pseudo-proj-True n = (λ*w*. *shift* (*stake n w*) (*sconst True*))

definition (**in** *infinite-coin-toss-space*) *pseudo-proj-False*:: *nat* ⇒ *bool stream* ⇒ *bool stream* **where**
  pseudo-proj-False n = (λ*w*. *shift* (*append* (*stake n w*) [*False*]) (*sconst True*))

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-neq-True*:
  shows *pseudo-proj-False n w* ≠ *pseudo-proj-True n w*
proof (*rule diff-streams-if*, *intro exI*)
  have *snth* (*pseudo-proj-False n w*) *n* = *False* **unfolding** *pseudo-proj-False-def*
by *simp*
 moreover have *snth* (*pseudo-proj-True n w*) *n* = *True* **unfolding** *pseudo-proj-True-def*
by *simp*
  ultimately show *snth* (*pseudo-proj-False n w*) *n* ≠ *snth* (*pseudo-proj-True n w*) *n* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-measurable*:

shows *pseudo-proj-False n* ∈ *measurable* (*bernoulli-stream p*) (*bernoulli-stream p*)
proof −
let *?N* = *bernoulli-stream p*
have *id* ∈ *measurable ?N ?N* **by** *simp*
moreover have (λ*w*. (*sconst True*)) ∈ *measurable ?N ?N* **using** *bernoulli-stream-space* **by** *simp*
ultimately show *?thesis* **using** *measurable-shift p-gt-0 p-lt-1*
unfolding *bernoulli-stream-def pseudo-proj-False-def* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-stake*:
shows *stake n* (*pseudo-proj-True n w*) = *stake n w* **by** (*simp add*: *pseudo-proj-True-def stake-shift*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-stake*:
shows *stake n* (*pseudo-proj-False n w*) = *stake n w* **by** (*simp add*: *pseudo-proj-False-def stake-shift*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-stake-image*:
assumes (*stake n w*) = *stake n x*
shows *pseudo-proj-True n w* = *pseudo-proj-True n x* **by** (*simp add*: *assms pseudo-proj-True-def*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-prefix*:
assumes *n* ≤ *m*
and *pseudo-proj-True m x* = *pseudo-proj-True m y*
shows *pseudo-proj-True n x* = *pseudo-proj-True n y*
by (*metis assms diff-is-0-eq id-stake-snth-sdrop length-stake pseudo-proj-True-def stake.simps*(*1*) *stake-shift*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-measurable*:
shows *pseudo-proj-True n* ∈ *measurable* (*bernoulli-stream p*) (*bernoulli-stream p*)
proof −
let *?N* = *bernoulli-stream p*
have *id* ∈ *measurable ?N ?N* **by** *simp*
moreover have (λ*w*. (*sconst True*)) ∈ *measurable ?N ?N* **using** *bernoulli-stream-space* **by** *simp*
ultimately show *?thesis* **using** *measurable-shift p-gt-0 p-lt-1*
unfolding *bernoulli-stream-def pseudo-proj-True-def* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-finite-image*:
shows *finite* (*range* (*pseudo-proj-True n*))
proof −
let *?U* = *UNIV*::*bool set*
have *?U* = {*True*, *False*} **by** *auto*
hence *finite ?U* **by** *simp*

moreover have *?U* ≠ {} **by** *auto*
ultimately have *fi*: *finite* (*stake n 'streams ?U*) **using** *stake-finite-universe-finite*[*of ?U*] **by** *simp*
let *?sh*= (λ*l*. *shift l* (*sconst True*))
have *finite* {*?sh l*|*l*. *l*∈(*stake n 'streams ?U*)} **using** *fi* **by** *simp*
moreover have {*?sh l*|*l*. *l*∈(*stake n 'streams ?U*)} = *range* (*pseudo-proj-True n*) **unfolding** *pseudo-proj-True-def* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-finite-image*:
shows *finite* (*range* (*pseudo-proj-False n*))
proof −
let *?U* = *UNIV*::*bool set*
have *?U* = {*True*, *False*} **by** *auto*
hence *finite ?U* **by** *simp*
moreover have *?U* ≠ {} **by** *auto*
ultimately have *fi*: *finite* (*stake n 'streams ?U*) **using** *stake-finite-universe-finite*[*of ?U*] **by** *simp*
let *?sh*= (λ*l*. *shift* (*l* @ [*False*]) (*sconst True*))
have *finite* {*?sh l*|*l*. *l*∈(*stake n 'streams ?U*)} **using** *fi* **by** *simp*
moreover have {*?sh l*|*l*. *l*∈(*stake n 'streams ?U*)} = *range* (*pseudo-proj-False n*) **unfolding** *pseudo-proj-False-def* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-proj*:
shows *pseudo-proj-True n* (*pseudo-proj-True n w*) = *pseudo-proj-True n w*
by (*metis pseudo-proj-True-def pseudo-proj-True-stake*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-Suc-False-proj*:
shows *pseudo-proj-True* (*Suc n*) (*pseudo-proj-False n w*) = *pseudo-proj-False n w*
by (*metis append-Nil2 cancel-comm-monoid-add-class.diff-cancel length-append-singleton length-stake order-refl pseudo-proj-False-def pseudo-proj-True-def stake.simps*(*1*) *stake-shift take-all*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-Suc-proj*:
shows *pseudo-proj-True* (*Suc n*) (*pseudo-proj-True n w*) = *pseudo-proj-True n w*
by (*metis id-apply id-stake-snth-sdrop pseudo-proj-True-def pseudo-proj-True-stake shift-left-inj siterate.code stake-sdrop stream.sel*(*2*))

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-proj-Suc*:
shows *pseudo-proj-True n* (*pseudo-proj-True* (*Suc n*) *w*) = *pseudo-proj-True n w*

by (*meson Suc-n-not-le-n nat-le-linear pseudo-proj-True-prefix pseudo-proj-True-stake pseudo-proj-True-stake-image*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-shift*:
shows *length l* = *n* ⟹ *pseudo-proj-True n* (*shift l* (*sconst True*)) = *shift l* (*sconst True*)
by (*simp add*: *pseudo-proj-True-def stake-shift*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-suc-img*:
shows *pseudo-proj-True* (*Suc n*) *w* ∈ {*pseudo-proj-True n w*, *pseudo-proj-False n w*}
by (*metis* (*full-types*) *cycle-decomp insertCI list.distinct*(*1*) *pseudo-proj-True-def pseudo-proj-False-def sconst-cycle shift-append stake-Suc*)

lemma (**in** *infinite-coin-toss-space*) *measurable-snth-count-space*:
shows (λ*w*. *snth w n*) ∈ *measurable* (*bernoulli-stream p*) (*count-space* (*UNIV*::*bool set*))
by (*simp add*: *bernoulli-stream-def*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-same-img*:
assumes *pseudo-proj-True n w* = *pseudo-proj-True n x*
shows *stake n w* = *stake n x* **by** (*metis assms pseudo-proj-True-stake*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-snth*:
assumes *pseudo-proj-True n x* = *pseudo-proj-True n w*
shows ⋀*i*. *Suc i* ≤ *n* ⟹ *snth x i* = *snth w i*
proof −
fix *i*
have *stake n w*= *stake n x* **using** *assms* **by** (*metis pseudo-proj-True-stake*)
assume *Suc i* ≤ *n*
thus *snth x i* = *snth w i* **using** ‹*stake n w* = *stake n x*› *stake-snth* **by** *auto*
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-snth'*:
assumes (⋀*i*. *Suc i* ≤ *n* ⟹ *snth w i* = *snth x i*)
shows *pseudo-proj-True n w* = *pseudo-proj-True n x*
proof −
have *stake n w* = *stake n x* **using** *stake-snth'*[*of n w x*] **using** *assms* **by** *simp*
moreover have *stake n w* = *stake n x* ⟹ *pseudo-proj-True n w* = *pseudo-proj-True n x* **using** *pseudo-proj-True-stake-image*[*of n w x*] **by** *simp*
ultimately show *?thesis* **by** *auto*
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-preimage*:
assumes $w = pseudo\text{-}proj\text{-}True\ n\ w$
shows $(pseudo\text{-}proj\text{-}True\ n) -` \{w\} = (\bigcap i{\in}\ \{m.\ Suc\ m \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\})$
proof
show $(pseudo\text{-}proj\text{-}True\ n) -` \{w\} \subseteq (\bigcap i{\in}\{m.\ Suc\ m \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\})$
proof
fix x
assume $x \in (pseudo\text{-}proj\text{-}True\ n) -`\{w\}$
hence $pseudo\text{-}proj\text{-}True\ n\ x = pseudo\text{-}proj\text{-}True\ n\ w$ **using** *assms* **by** *auto*
hence $\bigwedge i.\ i \in\{m.\ Suc\ m \leq n\} \Longrightarrow x\ \in (\lambda x.\ snth\ x\ i) -`\{snth\ w\ i\}$
by (*metis* (*mono-tags*) *Suc-le-eq mem-Collect-eq*
pseudo-proj-True-stake stake-nth vimage-singleton-eq)
thus $x \in (\bigcap i{\in}\{m.\ Suc\ m \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\})$ **by** *auto*
qed
show $(\bigcap i{\in}\{m.\ Suc\ m \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\}) \subseteq (pseudo\text{-}proj\text{-}True\ n) -` \{w\}$
proof
fix x
assume $x{\in}\ (\bigcap i{\in}\{m.\ Suc\ m \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\})$
hence $\bigwedge i.\ i \in\{m.\ Suc\ m \leq n\} \Longrightarrow x\ \in (\lambda x.\ snth\ x\ i) -`\{snth\ w\ i\}$ **by** *simp*
hence $\bigwedge i.\ i \in\{m.\ Suc\ m \leq n\} \Longrightarrow snth\ x\ i = snth\ w\ i$ **by** *simp*
hence $\bigwedge i.\ Suc\ i \leq n \Longrightarrow snth\ x\ i = snth\ w\ i$ **by** *auto*
hence $pseudo\text{-}proj\text{-}True\ n\ x = pseudo\text{-}proj\text{-}True\ n\ w$ **using** $pseudo\text{-}proj\text{-}True\text{-}snth'[of\ n\ x\ w]$ **by** *simp*
also have $... = w$ **using** *assms* **by** *simp*
finally have $pseudo\text{-}proj\text{-}True\ n\ x = w$ **.**
thus $x \in (pseudo\text{-}proj\text{-}True\ n) -`\{w\}$ **by** *auto*
qed
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-preimage*:
assumes $w = pseudo\text{-}proj\text{-}False\ n\ w$
shows $(pseudo\text{-}proj\text{-}False\ n) -` \{w\} = (\bigcap i{\in}\ \{m.\ Suc\ m \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\})$
proof
show $(pseudo\text{-}proj\text{-}False\ n) -` \{w\} \subseteq (\bigcap i{\in}\{m.\ Suc\ m \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\})$
proof
fix x
assume $x \in (pseudo\text{-}proj\text{-}False\ n) -`\{w\}$
hence $pseudo\text{-}proj\text{-}False\ n\ x = pseudo\text{-}proj\text{-}False\ n\ w$ **using** *assms* **by** *auto*
hence $\bigwedge i.\ i \in\{m.\ Suc\ m \leq n\} \Longrightarrow x\ \in (\lambda x.\ snth\ x\ i) -`\{snth\ w\ i\}$
by (*metis* (*mono-tags*) *Suc-le-eq mem-Collect-eq*
pseudo-proj-False-stake stake-nth vimage-singleton-eq)
thus $x \in (\bigcap i{\in}\{m.\ Suc\ m \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\})$ **by** *auto*

qed
show $(\bigcap i{\in}\{m.\ Suc\ m \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\}) \subseteq (pseudo\text{-}proj\text{-}False\ n) -` \{w\}$
proof
fix x
assume $x{\in}\ (\bigcap i{\in}\{m.\ Suc\ m \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\})$
hence $\bigwedge i.\ i \in \{m.\ Suc\ m \leq n\} \Longrightarrow x\ \in (\lambda x.\ snth\ x\ i) -` \{snth\ w\ i\}$ **by** *simp*
hence $\bigwedge i.\ i \in \{m.\ Suc\ m \leq n\} \Longrightarrow snth\ x\ i = snth\ w\ i$ **by** *simp*
hence $\bigwedge i.\ Suc\ i \leq n \Longrightarrow snth\ x\ i = snth\ w\ i$ **by** *auto*
hence $pseudo\text{-}proj\text{-}False\ n\ x = pseudo\text{-}proj\text{-}False\ n\ w$
by (*metis* (*full-types*) *pseudo-proj-False-def stake-snth'*)
also have $... = w$ **using** *assms* **by** *simp*
finally have $pseudo\text{-}proj\text{-}False\ n\ x = w$ **.**
thus $x \in (pseudo\text{-}proj\text{-}False\ n) -` \{w\}$ **by** *auto*
qed
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-preimage-stake*:
assumes $w = pseudo\text{-}proj\text{-}True\ n\ w$
shows $(pseudo\text{-}proj\text{-}True\ n) -` \{w\} = \{x.\ stake\ n\ x = stake\ n\ w\}$
proof
show $\{x.\ stake\ n\ x = stake\ n\ w\} \subseteq (pseudo\text{-}proj\text{-}True\ n) -` \{w\}$
proof
fix x
assume $x \in \{x.\ stake\ n\ x = stake\ n\ w\}$
hence $stake\ n\ x = stake\ n\ w$ **by** *auto*
hence $pseudo\text{-}proj\text{-}True\ n\ x = w$ **using** *assms pseudo-proj-True-def* **by** *auto*
thus $x \in (pseudo\text{-}proj\text{-}True\ n) -` \{w\}$ **by** *auto*
qed
show $(pseudo\text{-}proj\text{-}True\ n) -` \{w\} \subseteq \{x.\ stake\ n\ x = stake\ n\ w\}$
proof
fix x
assume $x{\in}\ pseudo\text{-}proj\text{-}True\ n -` \{w\}$
hence $pseudo\text{-}proj\text{-}True\ n\ x = pseudo\text{-}proj\text{-}True\ n\ w$ **using** *assms* **by** *auto*
hence $stake\ n\ x = stake\ n\ w$ **by** (*metis pseudo-proj-True-stake*)
thus $x{\in}\ \{x.\ stake\ n\ x = stake\ n\ w\}$ **by** *simp*
qed
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-preimage-stake*:
assumes $w = pseudo\text{-}proj\text{-}False\ n\ w$
shows $(pseudo\text{-}proj\text{-}False\ n) -` \{w\} = \{x.\ stake\ n\ x = stake\ n\ w\}$
proof
show $\{x.\ stake\ n\ x = stake\ n\ w\} \subseteq (pseudo\text{-}proj\text{-}False\ n) -` \{w\}$
proof
fix x
assume $x \in \{x.\ stake\ n\ x = stake\ n\ w\}$

hence *stake n x = stake n w* **by** *auto*
hence *pseudo-proj-False n x = w* **using** *assms pseudo-proj-False-def* **by** *auto*
thus *x ∈ (pseudo-proj-False n) −‘ {w}* **by** *auto*
qed
show *(pseudo-proj-False n) −‘ {w} ⊆ {x. stake n x = stake n w}*
proof
fix *x*
assume *x∈ pseudo-proj-False n −‘{w}*
hence *pseudo-proj-False n x = pseudo-proj-False n w* **using** *assms* **by** *auto*
hence *stake n x = stake n w* **by** *(metis pseudo-proj-False-stake)*
thus *x∈ {x. stake n x = stake n w}* **by** *simp*
qed
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-preimage-stake-space*:
assumes *w = pseudo-proj-True n w*
shows *(pseudo-proj-True n) −‘ {w} ∩ space M = {x∈ space M. stake n x = stake n w}*
proof −
have *(pseudo-proj-True n) −‘ {w} = {x. stake n x = stake n w}* **using** *assms*
by *(simp add:pseudo-proj-True-preimage-stake)*
hence *(pseudo-proj-True n) −‘ {w}∩ space M = {x. stake n x = stake n w}∩ space M*
by *simp*
also have *... = {x∈ space M. stake n x = stake n w}* **by** *auto*
finally show *?thesis* **.**
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-singleton*:
assumes *w = pseudo-proj-True n w*
shows *(pseudo-proj-True n) −‘{w} ∩ (space (bernoulli-stream p)) ∈ sets (bernoulli-stream p)*
proof *(cases {m. (Suc m) ≤ n} = {})*
case *False*
have *⋀i. (λx. snth x i) ∈ measurable (bernoulli-stream p) (count-space UNIV)* **by** *(simp add: measurable-snth-count-space)*
have *fi: ⋀i. Suc i ≤ n ⟹ (λw. snth w i) −‘ {snth w i} ∩ (space (bernoulli-stream p)) ∈ sets (bernoulli-stream p)*
proof −
fix *i*
assume *Suc i ≤ n*
have *(λx. snth x i) ∈ measurable (bernoulli-stream p) (count-space UNIV)* **by** *(simp add: measurable-snth-count-space)*
moreover have *{snth w i} ∈ sets (count-space UNIV)* **by** *auto*
ultimately show *(λx. snth x i) −‘ {snth w i}∩ (space (bernoulli-stream p)) ∈ sets (bernoulli-stream p)*
unfolding *measurable-def* **by** *(simp add: measurable-snth-count-space)*
qed

have $(\bigcap i\in \{m.\ (Suc\ m) \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\} \cap (space\ (bernoulli\text{-}stream\ p))) \in sets\ (bernoulli\text{-}stream\ p)$
proof $((rule\ sigma\text{-}algebra.countable\text{-}INT'')$, $auto)$
show $sigma\text{-}algebra\ (space\ (bernoulli\text{-}stream\ p))\ (sets\ (bernoulli\text{-}stream\ p))$
using $measure\text{-}space\ measure\text{-}space\text{-}def$ **by** $auto$
show $UNIV \in sets\ (bernoulli\text{-}stream\ p)$ **by** $(metis\ bernoulli\text{-}stream\text{-}space\ sets.top\ streams\text{-}UNIV)$
show $\bigwedge i.\ Suc\ i \leq n \Longrightarrow (\lambda w.\ w\ !!\ i) -` \{w\ !!\ i\} \cap space\ (bernoulli\text{-}stream\ p) \in sets\ (bernoulli\text{-}stream\ p)$ **using** fi **by** $simp$
qed
moreover have $(\bigcap i\in \{m.\ (Suc\ m) \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\} \cap (space\ (bernoulli\text{-}stream\ p))) =$
$(\bigcap i\in \{m.\ (Suc\ m) \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\}) \cap (space\ (bernoulli\text{-}stream\ p))$
using $False\ Inter\text{-}nonempty\text{-}distrib$ **by** $auto$
ultimately show $?thesis$ **using** $assms\ pseudo\text{-}proj\text{-}True\text{-}preimage[of\ w\ n]$ **by** $simp$
next
case $True$
hence $n = 0$ **using** $less\text{-}eq\text{-}Suc\text{-}le$ **by** $auto$
hence $pseudo\text{-}proj\text{-}True\ n = (\lambda w.\ sconst\ True)$ **by** $(simp\ add{:}\ pseudo\text{-}proj\text{-}True\text{-}def)$
hence $w = sconst\ True$ **using** $assms$ **by** $simp$
hence $(pseudo\text{-}proj\text{-}True\ n) -` \{w\} \cap (space\ (bernoulli\text{-}stream\ p)) = (space\ (bernoulli\text{-}stream\ p))$ **by** $(simp\ add{:}\ ‹pseudo\text{-}proj\text{-}True\ n = (\lambda w.\ sconst\ True)›)$
thus $(pseudo\text{-}proj\text{-}True\ n) -` \{w\} \cap (space\ (bernoulli\text{-}stream\ p)) \in sets\ (bernoulli\text{-}stream\ p)$ **by** $simp$
qed

lemma (**in** $infinite\text{-}coin\text{-}toss\text{-}space$) $pseudo\text{-}proj\text{-}False\text{-}singleton$:
assumes $w = pseudo\text{-}proj\text{-}False\ n\ w$
shows $(pseudo\text{-}proj\text{-}False\ n) -` \{w\} \cap (space\ (bernoulli\text{-}stream\ p)) \in sets\ (bernoulli\text{-}stream\ p)$
proof $(cases\ \{m.\ (Suc\ m) \leq n\} = \{\})$
case $False$
have $\bigwedge i.\ (\lambda x.\ snth\ x\ i) \in measurable\ (bernoulli\text{-}stream\ p)\ (count\text{-}space\ UNIV)$ **by** $(simp\ add{:}\ measurable\text{-}snth\text{-}count\text{-}space)$
have $fi{:}\ \bigwedge i.\ Suc\ i \leq n \Longrightarrow (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\} \cap (space\ (bernoulli\text{-}stream\ p)) \in sets\ (bernoulli\text{-}stream\ p)$
proof $-$
fix i
assume $Suc\ i \leq n$
have $(\lambda x.\ snth\ x\ i) \in measurable\ (bernoulli\text{-}stream\ p)\ (count\text{-}space\ UNIV)$ **by** $(simp\ add{:}\ measurable\text{-}snth\text{-}count\text{-}space)$
moreover have $\{snth\ w\ i\} \in sets\ (count\text{-}space\ UNIV)$ **by** $auto$
ultimately show $(\lambda x.\ snth\ x\ i) -` \{snth\ w\ i\} \cap (space\ (bernoulli\text{-}stream\ p)) \in sets\ (bernoulli\text{-}stream\ p)$
unfolding $measurable\text{-}def$ **by** $(simp\ add{:}\ measurable\text{-}snth\text{-}count\text{-}space)$
qed

have $(\bigcap i\in \{m.\ (Suc\ m) \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\} \cap (space\ (bernoulli\text{-}stream\ p))) \in sets\ (bernoulli\text{-}stream\ p)$
proof $((rule\ sigma\text{-}algebra.countable\text{-}INT'')$, *auto*)
show *sigma-algebra* $(space\ (bernoulli\text{-}stream\ p))\ (sets\ (bernoulli\text{-}stream\ p))$
using *measure-space measure-space-def* **by** *auto*
show $UNIV \in sets\ (bernoulli\text{-}stream\ p)$ **by** (*metis bernoulli-stream-space sets.top streams-UNIV*)
show $\bigwedge i.\ Suc\ i \leq n \Longrightarrow (\lambda w.\ w\ !!\ i) -` \{w\ !!\ i\} \cap space\ (bernoulli\text{-}stream\ p) \in sets\ (bernoulli\text{-}stream\ p)$ **using** *fi* **by** *simp*
qed
moreover have $(\bigcap i\in \{m.\ (Suc\ m) \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\} \cap (space\ (bernoulli\text{-}stream\ p))) =$
$(\bigcap i\in \{m.\ (Suc\ m) \leq n\}.\ (\lambda w.\ snth\ w\ i) -` \{snth\ w\ i\}) \cap (space\ (bernoulli\text{-}stream\ p))$
using *False Inter-nonempty-distrib* **by** *auto*
ultimately show *?thesis* **using** *assms pseudo-proj-False-preimage*[*of w n*] **by** *simp*
next
case *True*
hence $n = 0$ **using** *less-eq-Suc-le* **by** *auto*
hence $pseudo\text{-}proj\text{-}False\ n = (\lambda w.\ False\ \#\#\ sconst\ True)$ **by** (*simp add*: *pseudo-proj-False-def*)
hence $w = False\ \#\#\ sconst\ True$ **using** *assms* **by** *simp*
hence $(pseudo\text{-}proj\text{-}False\ n) -` \{w\} \cap (space\ (bernoulli\text{-}stream\ p)) = (space\ (bernoulli\text{-}stream\ p))$
by (*simp add*: ‹$pseudo\text{-}proj\text{-}False\ n = (\lambda w.\ False\#\#sconst\ True)$›)
thus $(pseudo\text{-}proj\text{-}False\ n) -` \{w\} \cap (space\ (bernoulli\text{-}stream\ p)) \in sets\ (bernoulli\text{-}stream\ p)$ **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-inverse-induct*:
assumes $w \in range\ (pseudo\text{-}proj\text{-}True\ n)$
shows $(pseudo\text{-}proj\text{-}True\ n) -` \{w\} =$
$(pseudo\text{-}proj\text{-}True\ (Suc\ n)) -` \{w\} \cup (pseudo\text{-}proj\text{-}True\ (Suc\ n)) -` \{pseudo\text{-}proj\text{-}False\ n\ w\}$
proof
let $?y = pseudo\text{-}proj\text{-}False\ n\ w$
show $(pseudo\text{-}proj\text{-}True\ n) -` \{w\} \subseteq (pseudo\text{-}proj\text{-}True\ (Suc\ n)) -` \{w\} \cup (pseudo\text{-}proj\text{-}True\ (Suc\ n)) -` \{?y\}$
proof
fix *z*
assume $z \in pseudo\text{-}proj\text{-}True\ n -` \{w\}$
thus $z \in pseudo\text{-}proj\text{-}True\ (Suc\ n) -` \{w\} \cup pseudo\text{-}proj\text{-}True\ (Suc\ n) -` \{?y\}$
using *pseudo-proj-False-def pseudo-proj-True-def pseudo-proj-True-stake pseudo-proj-True-suc-img* **by** *fastforce*
qed
{
fix *z*
assume $z \in pseudo\text{-}proj\text{-}True\ (Suc\ n) -` \{w\}$
hence $pseudo\text{-}proj\text{-}True\ (Suc\ n)\ z = w$ **by** *simp*

hence *pseudo-proj-True n z = pseudo-proj-True n w* **by** (*metis pseudo-proj-True-proj-Suc*)
also have *... = w* **using** *assms pseudo-proj-True-def pseudo-proj-True-stake* **by** *auto*
finally have *pseudo-proj-True n z = w* **.**
}
hence *fst*: *pseudo-proj-True (Suc n) −‘ {w} ⊆ (pseudo-proj-True n) −‘ {w}* **by** *blast*
{
fix *z*
assume *z ∈ pseudo-proj-True (Suc n) −‘ {?y}*
hence *pseudo-proj-True n z = pseudo-proj-True n w*
by (*metis append1-eq-conv append-Nil2 cancel-comm-monoid-add-class.diff-cancel length-append-singleton length-stake order-refl pseudo-proj-False-def pseudo-proj-True-stake pseudo-proj-True-stake-image stake-Suc stake-invert-Nil stake-shift take-all vimage-singleton-eq*)

also have *... = w* **using** *assms pseudo-proj-True-def pseudo-proj-True-stake* **by** *auto*
finally have *pseudo-proj-True n z = w* **.**
}
hence *scd*: *pseudo-proj-True (Suc n) −‘ {?y} ⊆ (pseudo-proj-True n) −‘ {w}* **by** *blast*
show *(pseudo-proj-True (Suc n)) −‘ {w} ∪ (pseudo-proj-True (Suc n)) −‘{?y} ⊆ (pseudo-proj-True n) −‘ {w}*
using *fst scd* **by** *auto*
qed

5.3.2 Natural filtration locale

This part is mainly devoted to the proof that the projection function defined above indeed permits to obtain a filtration on the infinite coin toss space, and that this filtration is initially trivial.

definition (**in** *infinite-coin-toss-space*) *nat-filtration*::*nat ⇒ bool stream measure* **where**
nat-filtration n = fct-gen-subalgebra M M (pseudo-proj-True n)

locale *infinite-cts-filtration = infinite-coin-toss-space +*
fixes *F*
assumes *natural-filtration*: *F = nat-filtration*

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-space*:
shows *space (nat-filtration n) = UNIV*
by (*metis bernoulli bernoulli-stream-space fct-gen-subalgebra-space nat-filtration-def streams-UNIV*)

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-sets*:
shows *sets* (*nat-filtration n*) =
sigma-sets (*space* (*bernoulli-stream p*))
{*pseudo-proj-True n* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets* (*bernoulli-stream p*)}
proof −
have *sigma-sets* (*space M*) {*pseudo-proj-True n* −‘ *S* ∩ *space M* |*S*. *S* ∈ *sets* (*bernoulli-stream p*)} =
sets (*fct-gen-subalgebra M M* (*pseudo-proj-True n*))
using *bernoulli fct-gen-subalgebra-sets pseudo-proj-True-measurable* **by** *blast*
then show *?thesis*
using *bernoulli nat-filtration-def* **by** *force*
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-singleton*:
assumes *pseudo-proj-True n w* = *w*
shows *pseudo-proj-True n* −‘{*w*} ∈ *sets* (*nat-filtration n*)
proof −
let *?pw* = *pseudo-proj-True n* −‘{*w*}
have *memset*:*?pw* ∈ *sets M* **using** *bernoulli assms bernoulli-stream-preimage*[*of* - - *pseudo-proj-True n*]
pseudo-proj-True-singleton[*of w n*] **by** *simp*
have *pseudo-proj-True n* −‘*?pw* ∈ *sets* (*nat-filtration n*)
proof −
have *pseudo-proj-True n* −‘*?pw* ∩ (*space M*) ∈ *sets* (*nat-filtration n*) **using** *memset*
by (*metis fct-gen-subalgebra-sets-mem nat-filtration-def*)
moreover have *pseudo-proj-True n* −‘*?pw* ∩ (*space M*) = *pseudo-proj-True n* −‘*?pw* **using**
bernoulli-stream-preimage[*of* - - *pseudo-proj-True n*] *bernoulli* **by** *simp*
ultimately show *pseudo-proj-True n* −‘*?pw* ∈ *sets* (*nat-filtration n*) **by** *auto*
qed
moreover have *pseudo-proj-True n* −‘*?pw* = *?pw* **using** *pseudo-proj-True-proj* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-pseudo-proj-True-measurable*:
shows *pseudo-proj-True n* ∈ *measurable* (*nat-filtration n*) *M* **unfolding** *nat-filtration-def*
using *bernoulli fct-gen-subalgebra-fct-measurable*[*of pseudo-proj-True n M M*] *pseudo-proj-True-measurable*[*of n*]
bernoulli-stream-space **by** *auto*

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-comp-measurable*:

assumes $f \in$ *measurable M N*
and $f \circ$ *pseudo-proj-True n* $= f$
shows $f \in$ *measurable* (*nat-filtration n*) *N*
by (*metis assms measurable-comp nat-filtration-pseudo-proj-True-measurable*)

definition (**in** *infinite-coin-toss-space*) *set-discriminating* **where**
set-discriminating n f N $\equiv$ ($\forall$ *w*. *f w* $\neq$ *f* (*pseudo-proj-True n w*) $\longrightarrow$
($\exists$ *A*$\in$*sets N*. (*f w* $\in$ *A*) = (*f* (*pseudo-proj-True n w*) $\notin$ *A*)))

lemma (**in** *infinite-coin-toss-space*) *set-discriminating-if*:
fixes *f*::*bool stream* $\Rightarrow$ *'b*::{*t0-space*}
assumes *f*$\in$ *borel-measurable* (*nat-filtration n*)
shows *set-discriminating n f borel* **unfolding** *set-discriminating-def*
proof (*intro allI impI*)
{
fix *w*
assume *f w* $\neq$ (*f* $\circ$ (*pseudo-proj-True n*)) *w*
hence $\exists$ *U*. *open U* $\wedge$ (*f w* $\in$ *U* = ((*f* $\circ$ (*pseudo-proj-True n*)) *w* $\notin$ *U*)) **using** *separation-t0* **by** *auto*
from *this* **obtain** *A* **where** *open A* **and** *f w*$\in$ *A* = ((*f* $\circ$ (*pseudo-proj-True n*)) *w* $\notin$ *A*) **by** *blast* **note** *Ah* = *this*
have *A*$\in$ *sets borel* **using** *Ah* **by** *simp*
hence $\exists$ *A*$\in$*sets borel*. (*f w* $\in$ *A*) = ((*f* $\circ$ (*pseudo-proj-True n*)) *w* $\notin$ *A*) **using** *Ah* **by** *blast*
}
thus $\bigwedge$*w*. *f w* $\neq$ *f* (*pseudo-proj-True n w*) $\Longrightarrow$ $\exists$ *A*$\in$*sets borel*. (*f w* $\in$ *A*) = (*f* (*pseudo-proj-True n w*) $\notin$ *A*) **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-not-borel-info*:
assumes *f*$\in$ *measurable* (*nat-filtration n*) *N*
and *set-discriminating n f N*
shows *f*$\circ$ *pseudo-proj-True n* $= f$
proof (*rule ccontr*)
assume *f*$\circ$ *pseudo-proj-True n* $\neq f$
hence $\exists$ *w*. (*f*$\circ$ (*pseudo-proj-True n*)) *w* $\neq$ *f w* **by** *auto*
from *this* **obtain** *w* **where** (*f*$\circ$ (*pseudo-proj-True n*)) *w* $\neq$ *f w* **by** *blast* **note** *wh* = *this*
let *?x* = *pseudo-proj-True n w*
have *pseudo-proj-True n ?x* = *pseudo-proj-True n w* **by** (*simp add*: *pseudo-proj-True-proj*)
have *f w* $\neq$ *f* (*pseudo-proj-True n w*) **using** *wh* **by** *simp*
hence $\exists$ *A* $\in$ *sets N*. (*f w* $\in$ *A* = (*f ?x* $\notin$ *A*)) **using** *assms* **unfolding** *set-discriminating-def* **by** *simp*
from *this* **obtain** *A* **where** *A* $\in$ *sets N* **and** *f w*$\in$ *A* = (*f ?x* $\notin$ *A*) **by** *blast* **note** *Ah* = *this*
have *f*$-$‘ *A*$\cap$ (*space* (*nat-filtration n*)) $\in$ *sets* (*nat-filtration n*)
using *Ah assms borel-open measurable-sets* **by** *blast*
hence *fn*:*f*$-$‘ *A* $\in$ *sets* (*nat-filtration n*) **using** *nat-filtration-space* **by** *simp*
have *?x*$\in$ *f*$-$‘*A* = (*w* $\in$ *f* $-$‘*A*) **using** ‹*pseudo-proj-True n ?x* = *pseudo-proj-True*

n w› assms
fct-gen-subalgebra-info[*of pseudo-proj-True n M*] *bernoulli-stream-space*
by (*metis Pi-I UNIV-I bernoulli fn nat-filtration-def streams-UNIV*)
also have ... = (*f w* ∈ *A*) **by** *simp*
also have ... = (*f ?x* ∉ *A*) **using** *Ah* **by** *simp*
also have ... = (*?x* ∉ *f* −‘*A*) **by** *simp*
finally have *?x*∈ *f*−‘*A* = (*?x* ∉ *f* −‘*A*) **.**
thus *False* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-info*:
fixes *f*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows *f*∘ *pseudo-proj-True n* = *f*
proof (*rule nat-filtration-not-borel-info*)
show *f*∈ *borel-measurable* (*nat-filtration n*) **using** *assms* **by** *simp*
show *set-discriminating n f borel* **using** *assms* **by** (*simp add*: *set-discriminating-if*)
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-not-borel-info'*:
assumes *f*∈ *measurable* (*nat-filtration n*) *N*
and *set-discriminating n f N*
shows *f*∘ *pseudo-proj-False n* = *f*
proof
fix *x*
have (*f* ∘ *pseudo-proj-False n*) *x* = *f* (*pseudo-proj-False n x*) **by** *simp*
also have ... = *f* (*pseudo-proj-True n* (*pseudo-proj-False n x*)) **using** *assms nat-filtration-not-borel-info*
by (*metis comp-apply*)
also have ... = *f* (*pseudo-proj-True n x*)
proof −
have *pseudo-proj-True n* (*pseudo-proj-False n x*) = *pseudo-proj-True n x*
by (*simp add*: *pseudo-proj-False-stake pseudo-proj-True-def*)
thus *?thesis* **by** *simp*
qed
also have ... = *f x* **using** *assms nat-filtration-not-borel-info* **by** (*metis comp-apply*)
finally show (*f* ∘ *pseudo-proj-False n*) *x* = *f x* **.**
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-info'*:
fixes *f*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)

shows $f\circ$ *pseudo-proj-False n* = *f*
proof
fix *x*
have (*f* ∘ *pseudo-proj-False n*) *x* = *f* (*pseudo-proj-False n x*) **by** *simp*
also have ... = *f* (*pseudo-proj-True n* (*pseudo-proj-False n x*)) **using** *assms nat-filtration-info*
by (*metis comp-apply*)
also have ... = *f* (*pseudo-proj-True n x*)
proof −
have *pseudo-proj-True n* (*pseudo-proj-False n x*) = *pseudo-proj-True n x*
by (*simp add*: *pseudo-proj-False-stake pseudo-proj-True-def*)
thus *?thesis* **by** *simp*
qed
also have ... = *f x* **using** *assms nat-filtration-info* **by** (*metis comp-apply*)
finally show (*f* ∘ *pseudo-proj-False n*) *x* = *f x* **.**
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-borel-measurable-characterization*:
fixes *f*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *f*∈ *borel-measurable M*
shows *f*∈ *borel-measurable* (*nat-filtration n*) ⟷ *f*∘ *pseudo-proj-True n* = *f*
using *assms nat-filtration-comp-measurable nat-filtration-info* **by** *blast*

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-borel-measurable-init*:
fixes *f*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration 0*)
shows *f* = (λ*w*. *f* (*sconst True*))
proof
fix *w*
have *f w* = *f* ((*pseudo-proj-True 0*) *w*) **using** *assms nat-filtration-info*[*of f 0*] **by** (*metis comp-apply*)
also have ... = *f* (*sconst True*) **by** (*simp add*: *pseudo-proj-True-def*)
finally show *f w* = *f* (*sconst True*) **.**
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-Suc-sets*:
shows *sets* (*nat-filtration n*) ⊆ *sets* (*nat-filtration* (*Suc n*))
proof −
{
fix *x*
assume *x*∈ {*pseudo-proj-True n* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets M*}

hence $\exists B.\ B \in sets\ M \wedge x = pseudo\text{-}proj\text{-}True\ n\ -`\ B \cap space\ M$ **by** *auto*
from *this* **obtain** B **where** $B \in sets\ M$ **and** $x = pseudo\text{-}proj\text{-}True\ n\ -`\ B \cap space\ M$
by *blast* **note** *xhyps* = *this*
let *?Bim* = $B \cap (range\ (pseudo\text{-}proj\text{-}True\ n))$
let *?preT* = $(\lambda n\ w.\ (pseudo\text{-}proj\text{-}True\ n)\ -`\ \{w\})$
have *finite ?Bim* **using** *pseudo-proj-True-finite-image* **by** *simp*
have $pseudo\text{-}proj\text{-}True\ n\ -`B \cap (space\ M) = pseudo\text{-}proj\text{-}True\ n\ -`B$
using *bernoulli bernoulli-stream-preimage*[*of - - pseudo-proj-True n*] **by** *simp*
also have ... = $pseudo\text{-}proj\text{-}True\ n\ -`?Bim$ **by** *auto*
also have ... = $(\bigcup\ w \in ?Bim.\ ?preT\ n\ w)$ **by** *auto*
also have ... = $(\bigcup\ w \in ?Bim.\ (?preT\ (Suc\ n)\ w \cup ?preT\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)))$
by (*simp add*:*pseudo-proj-True-inverse-induct*)
also have ... = $(\bigcup\ w \in ?Bim.\ ?preT\ (Suc\ n)\ w) \cup (\bigcup\ w \in ?Bim.\ ?preT\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w))$ **by** *auto*
finally have *tmpeq*: $pseudo\text{-}proj\text{-}True\ n\ -`B \cap (space\ M) = (\bigcup\ w \in ?Bim.\ ?preT\ (Suc\ n)\ w) \cup (\bigcup\ w \in ?Bim.\ ?preT\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w))$ **.**
have $(\bigcup\ w \in ?Bim.\ ?preT\ (Suc\ n)\ w) \in sets\ (nat\text{-}filtration\ (Suc\ n))$
using ‹*finite ?Bim*› *nat-filtration-singleton pseudo-proj-True-Suc-proj* **by** *auto*
moreover have $(\bigcup\ w \in ?Bim.\ ?preT\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)) \in sets\ (nat\text{-}filtration\ (Suc\ n))$ **using** ‹*finite ?Bim*›
by (*simp add*: *nat-filtration-singleton pseudo-proj-True-Suc-False-proj sets.finite-UN*)
ultimately have $x \in sets\ (nat\text{-}filtration\ (Suc\ n))$
using *tmpeq xhyps* **by** *simp*
} note *xmem* = *this*
have $sets\ (nat\text{-}filtration\ n) = sigma\text{-}sets\ (space\ M)\ \{pseudo\text{-}proj\text{-}True\ n\ -`\ B \cap space\ M\ |B.\ B \in sets\ M\}$
using *bernoulli nat-filtration-sets* **by** *blast*
also have ... $\subseteq (nat\text{-}filtration\ (Suc\ n))$
proof (*rule sigma-algebra.sigma-sets-subset*)
show $\{pseudo\text{-}proj\text{-}True\ n\ -`\ B \cap space\ M\ |B.\ B \in sets\ M\}$
$\subseteq sets\ (nat\text{-}filtration\ (Suc\ n))$ **using** *xmem* **by** *auto*
show $sigma\text{-}algebra\ (space\ M)\ (sets\ (nat\text{-}filtration\ (Suc\ n)))$
by (*metis bernoulli bernoulli-stream-space nat-filtration-space sets.sigma-algebra-axioms streams-UNIV*)
qed
finally show *?thesis* **.**
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-subalgebra*:
shows $subalgebra\ M\ (nat\text{-}filtration\ n)$ **using** *bernoulli fct-gen-subalgebra-is-subalgebra nat-filtration-def*
pseudo-proj-True-measurable **by** *metis*

lemma (**in** *infinite-coin-toss-space*) *nat-discrete-filtration*:
shows *filtration M nat-filtration*

```
  unfolding filtration-def
proof((intro conjI), (intro allI)+)
  {
    fix n
    let ?F = nat-filtration n
    show subalgebra M ?F
      using bernoulli fct-gen-subalgebra-is-subalgebra nat-filtration-def
      pseudo-proj-True-measurable by metis
  } note allrm = this
  show ∀ n m. n ≤ m ⟶ subalgebra (nat-filtration m) (nat-filtration n)
  proof (intro allI impI)
    let ?F = nat-filtration
    fix n::nat
    fix m
    show n ≤ m ⟹ subalgebra (nat-filtration m) (nat-filtration n)
    proof (induct m)
      case (Suc m)
      have subalgebra (?F (Suc m)) (?F m) unfolding subalgebra-def
      proof (intro conjI)
     show speq: space (?F m) = space (?F (Suc m)) by (simp add: nat-filtration-space)
        show sets (?F m) ⊆ sets (?F (Suc m)) using nat-filtration-Suc-sets by
simp
      qed

      thus n ≤ Suc m ⟹ subalgebra (?F (Suc m)) (?F n) using Suc
        using Suc.hyps le-Suc-eq subalgebra-def by fastforce
      next
      case 0
        thus ?case by (simp add: subalgebra-def)
    qed
  qed
qed

lemma (in infinite-coin-toss-space) nat-info-filtration:
  shows init-triv-filt M nat-filtration unfolding init-triv-filt-def
proof
  show filtration M nat-filtration by (simp add:nat-discrete-filtration)
   have img: ∀  w∈ space M. pseudo-proj-True 0 w = sconst True unfolding
pseudo-proj-True-def by simp
  show sets (nat-filtration bot) = {{}, space M}
  proof
    show {{}, space M} ⊆ sets (nat-filtration bot)
      by (metis empty-subsetI insert-subset nat-filtration-subalgebra sets.empty-sets
sets.top subalgebra-def)
    show sets (nat-filtration bot) ⊆ {{}, space M}
    proof −
      have ∀ B ∈ sets (bernoulli-stream p). pseudo-proj-True 0 −‘ B ∩ space M ∈
{{}, space M}
      proof
```

```
      fix B
      assume B ∈ sets (bernoulli-stream p)
      show pseudo-proj-True 0 −‘ B ∩ space M ∈ {{}, space M}
      proof (cases sconst True ∈ B)
        case True
       hence pseudo-proj-True 0 −‘ B ∩ space M = space M using img by auto
        thus ?thesis by auto
      next
        case False
        hence pseudo-proj-True 0 −‘ B ∩ space M = {} using img by auto
        thus ?thesis by auto
      qed
    qed
     hence {pseudo-proj-True 0 −‘ B ∩ space M |B. B ∈ sets (bernoulli-stream
p)} ⊆ {{}, space M} by auto
    hence sigma-sets (space (bernoulli-stream p))
        {pseudo-proj-True 0 −‘ B ∩ space M |B. B ∈ sets (bernoulli-stream p)}
⊆ {{}, space M}
    using sigma-algebra.sigma-sets-subset[of space (bernoulli-stream p) {{}, space
M}]
      by (simp add: bernoulli sigma-algebra-trivial)
    thus ?thesis by (simp add:nat-filtration-sets bot-nat-def)
  qed
 qed
qed
```

```
sublocale infinite-cts-filtration ⊆ triv-init-disc-filtr-prob-space
proof (unfold-locales, intro conjI)
 show disc-filtr M F unfolding disc-filtr-def
  using filtrationE2 nat-discrete-filtration nat-filtration-subalgebra natural-filtration
by auto
 show sets (F bot) = {{}, space M} using nat-info-filtration natural-filtration
   unfolding init-triv-filt-def by simp
qed
```

```
lemma (in infinite-coin-toss-space) nat-filtration-vimage-finite:
 fixes f::bool stream ⇒ 'b::{t2-space}
 assumes f∈ borel-measurable (nat-filtration n)
 shows finite (f‘(space M)) using pseudo-proj-True-finite-image nat-filtration-info[of
f n]
    by (metis assms bernoulli bernoulli-stream-space finite-imageI fun.set-map
streams-UNIV)
```

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-borel-measurable-simple*:
fixes *f*::*bool stream* ⇒ *'b*::{*t2-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows *simple-function M f*
proof −
have *f1*: ∀ *m ma*. (*m*::*bool stream measure*) →$_M$ (*ma*::*'b measure*) = {*f* ∈ *space m* → *space ma*. ∀ *B*. *B* ∈ *sets ma* ⟶ *f* −‘ *B* ∩ *space m* ∈ *sets m*}
by (*metis measurable-def*)
then have *f* ∈ *space* (*nat-filtration n*) → *space borel* ∧ (∀ *B*. *B* ∈ *sets borel* ⟶ *f* −‘ *B* ∩ *space* (*nat-filtration n*) ∈ *sets* (*nat-filtration n*))
using *assms* **by** *blast*
then have *f* ∈ *space M* → *space borel* ∧ (∀ *B*. *B* ∈ *sets borel* ⟶ *f* −‘ *B* ∩ *space M* ∈ *events*)
by (*metis* (*no-types*) *contra-subsetD nat-filtration-subalgebra subalgebra-def*)
then have *random-variable borel f*
using *f1* **by** *blast*
then show *?thesis*
using *assms nat-filtration-vimage-finite simple-function-borel-measurable* **by** *blast*
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-singleton-range-set*:
fixes *f*::*bool stream* ⇒ *'b*::{*t2-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows ∃ *A*∈ *sets borel*. *range f* ∩ *A* = {*f x*}
proof −
let *?Ax* = *range f* − {*f x*}
have *range f* = *f*‘*space M* **using** *bernoulli bernoulli-stream-space* **by** *simp*
hence *finite ?Ax* **using** *assms nat-filtration-vimage-finite* **by** *auto*
hence ∃ *U*. *open U* ∧ *f x*∈ *U* ∧ *U*∩ *?Ax* = {} **by** (*simp add*:*open-except-set*)
then obtain *U* **where** *open U* **and** *f x*∈ *U* **and** *U*∩ *?Ax* = {} **by** *auto*
have *U* ∈ *sets borel* **using** ‹*open U*› **by** *simp*
have *range f* ∩ *U* = {*f x*} **using** ‹*f x* ∈ *U*› ‹*U*∩ *?Ax* = {}› **by** *blast*
thus ∃ *A*∈ *sets borel*. *range f* ∩ *A* = {*f x*} **using** ‹*U*∈ *sets borel*› **by** *auto*
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-borel-measurable-singleton*:
fixes *f*::*bool stream* ⇒ *'b*::{*t2-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows *f* −‘{*f x*} ∈ *sets* (*nat-filtration n*)
proof −
let *?Ax* = *f*‘*space M* − {*f x*}
have *finite ?Ax*
using *assms nat-filtration-vimage-finite* **by** *blast*
hence ∃ *U*. *open U* ∧ *f x*∈ *U* ∧ *U*∩ *?Ax* = {} **by** (*simp add*:*open-except-set*)
then obtain *U* **where** *open U* **and** *f x*∈ *U* **and** *U*∩ *?Ax* = {} **by** *auto*
have *f x* ∈ *f* ‘ *space M* **using** *bernoulli-stream-space bernoulli* **by** *simp*
hence *f*‘*space M* ∩ *U* = {*f x*} **using** ‹*f x*∈ *U*› ‹*U*∩ *?Ax* = {}› **by** *blast*

hence $\exists A.$ *open* $A \wedge$ *f'space* $M \cap A = \{f\, x\}$ **using** ‹*open* U› **by** *auto*
from *this* **obtain** A **where** *open* A **and** *inter*: *f'space* $M \cap A = \{f\, x\}$ **by** *auto*
have $A \in$ *sets borel* **using** ‹*open* A› **by** *simp*
hence $f\, -`A \cap$ *space* $M \in$ *sets* (*nat-filtration* n) **using** *assms nat-filtration-space*
by (*simp add*: *bernoulli bernoulli-stream-space in-borel-measurable-borel*)
hence $f\, -`A \cap$ *space* $M \in$ *events* **using** *nat-filtration-subalgebra*
by (*meson subalgebra-def subset-eq*)
have $f\, -`\{f\, x\} \cap$ *space* $M = f\, -`A \cap$ *space* M
proof
have $f\, x \in A$ **using** *inter* **by** *auto*
thus $f\, -`\{f\, x\} \cap$ *space* $M \subseteq f\, -`A \cap$ *space* M **by** *auto*
show $f\, -`A \cap$ *space* $M \subseteq f\, -`\{f\, x\} \cap$ *space* M
proof
fix y
assume $y \in f\, -`A \cap$ *space* M
hence $f\, y \in A \cap$ *f'space* M **by** *simp*
hence $f\, y = f\, x$ **using** *inter* **by** *auto*
thus $y \in f\, -`\{f\, x\} \cap$ *space* $M$ **using** ‹$y \in f\, -`A \cap$ *space* M› **by** *auto*
qed
qed
moreover have $f\, -`A \cap$ *space* $M \in$ (*nat-filtration* n) **using** *assms* ‹$A \in$ *sets borel*›
using ‹$f\, -`A \cap$ *space* $M \in$ *sets* (*nat-filtration* n)› **by** *blast*
ultimately show *?thesis* **using** *bernoulli-stream-space bernoulli* **by** *simp*
qed

lemma (**in** *infinite-cts-filtration*) *borel-adapt-nat-filtration-info*:
fixes X::*nat* $\Rightarrow$ *bool stream* $\Rightarrow$ $'b$::{*t0-space*}
assumes *borel-adapt-stoch-proc* F X
and $m \leq n$
shows $X\ m$ (*pseudo-proj-True* n w) $= X\ m\ w$
proof −
have $X\ m \in$ *borel-measurable* ($F\ n$) **using** *assms natural-filtration*
using *increasing-measurable-info*
by (*metis adapt-stoch-proc-def*)
thus *?thesis* **using** *nat-filtration-info natural-filtration*
by (*metis comp-apply*)
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-borel-measurable-integrable*:
assumes $f \in$ *borel-measurable* (*nat-filtration* n)
shows *integrable* M f
proof −
have *simple-function* M f **using** *assms* **by** (*simp add*: *nat-filtration-borel-measurable-simple*)
moreover have *emeasure* M $\{y \in$ *space* $M.\ f\, y \neq 0\} \neq \infty$ **by** *simp*
ultimately have *Bochner-Integration.simple-bochner-integrable* M f
using *Bochner-Integration.simple-bochner-integrable.simps* **by** *blast*
hence *has-bochner-integral* M f (*Bochner-Integration.simple-bochner-integral* M f)

using *has-bochner-integral-simple-bochner-integrable* **by** *auto*
thus *?thesis* **using** *integrable.simps* **by** *auto*
qed

definition (**in** *infinite-coin-toss-space*) *spick*:: *bool stream* ⇒ *nat* ⇒ *bool* ⇒ *bool stream* **where**
spick w n v = *shift* (*stake n w*) (*v##* *sconst True*)

lemma (**in** *infinite-coin-toss-space*) *spickI*:
shows *stake n* (*spick w n v*) = *stake n w* ∧ *snth* (*spick w n v*) *n* = *v*
by (*simp add*: *spick-def stake-shift*)

lemma (**in** *infinite-coin-toss-space*) *spick-eq-pseudo-proj-True*:
shows *spick w n True* = *pseudo-proj-True n w* **unfolding** *spick-def pseudo-proj-True-def*
by (*metis* (*full-types*) *id-apply siterate.code*)

lemma (**in** *infinite-coin-toss-space*) *spick-eq-pseudo-proj-False*:
shows *spick w n False* = *pseudo-proj-False n w* **unfolding** *spick-def pseudo-proj-False-def*
by *simp*

lemma (**in** *infinite-coin-toss-space*) *spick-pseudo-proj*:
shows *spick* (*pseudo-proj-True* (*Suc n*) *w*) *n v* = *spick w n v*
by (*metis pseudo-proj-True-proj-Suc pseudo-proj-True-stake spick-def*)

lemma (**in** *infinite-coin-toss-space*) *spick-pseudo-proj-gen*:
shows *m* < *n* ⟹ *spick* (*pseudo-proj-True n w*) *m v* = *spick w m v*
by (*metis Suc-leI pseudo-proj-True-proj pseudo-proj-True-prefix spick-pseudo-proj*)

lemma (**in** *infinite-coin-toss-space*) *spick-nat-filtration-measurable*:
shows (λ*w*. *spick w n v*) ∈ *measurable* (*nat-filtration n*) *M*
proof (*rule nat-filtration-comp-measurable*)
show (λ*w*. *spick w n v*) ∈ *measurable M M*
proof −
let *?N* = *bernoulli-stream p*
have *id* ∈ *measurable ?N ?N* **by** *simp*
moreover have (λ*w*. *v##* (*sconst True*)) ∈ *measurable ?N ?N* **using** *bernoulli-stream-space*
by *simp*
ultimately show *?thesis* **using** *measurable-shift bernoulli p-gt-0 p-lt-1*
unfolding *bernoulli-stream-def spick-def* **by** *simp*
qed
{
fix *w*
have *spick* (*pseudo-proj-True n w*) *n v* = *spick w n v*

by (*simp add*: *pseudo-proj-True-stake spick-def*)
}
thus (λ*w*. *spick w n v*) ∘ *pseudo-proj-True n* = (λ*w*. *spick w n v*) **by** *auto*
qed

definition (**in** *infinite-coin-toss-space*) *proj-rep-set*:
proj-rep-set n = *range* (*pseudo-proj-True n*)

lemma (**in** *infinite-coin-toss-space*) *proj-rep-set-finite*:
shows *finite* (*proj-rep-set n*) **using** *pseudo-proj-True-finite-image*
by (*simp add*: *proj-rep-set*)

lemma (**in** *infinite-coin-toss-space*) *set-filt-contain*:
assumes *A*∈ *sets* (*nat-filtration n*)
and *w*∈ *A*
shows *pseudo-proj-True n* −‘ {*pseudo-proj-True n w*} ⊆ *A*
proof
define *indA* **where** *indA* = ((*indicator A*)::*bool stream*⇒*real*)
have *indA* ∈ *borel-measurable* (*nat-filtration n*) **unfolding** *indA-def*
by (*simp add*: *assms*(*1*) *borel-measurable-indicator*)
fix *x*
assume *x* ∈ *pseudo-proj-True n* −‘ {*pseudo-proj-True n w*}
have *indA x* = *indA* (*pseudo-proj-True n x*)
using *nat-filtration-info*[*symmetric*, *of indicator A n*] ‹*indA* ∈ *borel-measurable* (*nat-filtration n*)›
unfolding *indA-def* **by** (*metis comp-apply*)
also have ... = *indA* (*pseudo-proj-True n w*) **using** ‹*x* ∈ *pseudo-proj-True n* −‘ {*pseudo-proj-True n w*}›
by *simp*
also have ... = *indA w* **using** *nat-filtration-info*[*of indicator A n*]
‹*indA* ∈ *borel-measurable* (*nat-filtration n*)› **unfolding** *indA-def* **by** (*metis comp-apply*)
also have ... = *1* **using** *assms* **unfolding** *indA-def* **by** *simp*
finally have *indA x* = *1* **.**
thus *x*∈ *A* **unfolding** *indA-def* **by** (*simp add*: *indicator-eq-1-iff*)
qed

lemma (**in** *infinite-cts-filtration*) *measurable-range-rep*:
fixes *f*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *f* ∈ *borel-measurable* (*nat-filtration n*)
shows *range f* = (⋃ *r*∈(*proj-rep-set n*). {*f*(*r*)})
proof −
have *f* = *f* ∘ (*pseudo-proj-True n*) **using** *assms nat-filtration-info*[*of f n*] **by** *simp*
hence *range f* = *f* ‘(*proj-rep-set n*) **by** (*metis fun.set-map proj-rep-set*)

also have ... = ($\bigcup$ *r*∈*proj-rep-set n*. {*f r*}) **by** *blast*
finally show *range f* = ($\bigcup$ *r*∈*proj-rep-set n*. {*f r*}) **.**
qed

lemma (**in** *infinite-coin-toss-space*) *borel-measurable-stake*:
fixes *f*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
and *stake n w* = *stake n y*
shows *f w* = *f y*
proof −
have *pseudo-proj-True n w* = *pseudo-proj-True n y* **unfolding** *pseudo-proj-True-def*
using *assms* **by** *simp*
thus *?thesis* **using** *assms nat-filtration-info* **by** (*metis comp-apply*)
qed

5.3.3 Probability component

The probability component permits to compute measures of subspaces in a straightforward way.

definition *prob-component* **where**
prob-component (*p*::*real*) *w n* = (*if* (*snth w n*) *then p else 1*−*p*)

lemma *prob-component-neq-zero*:
assumes *0* < *p*
and *p* < *1*
shows *prob-component p w n* ≠ *0* **using** *assms prob-component-def* **by** *auto*

lemma *prob-component-measure*:
fixes *x*::*bool stream*
assumes *0* ≤ *p*
and *p* ≤ *1*
shows *emeasure* (*measure-pmf* (*bernoulli-pmf p*)) {*snth x i*} = *prob-component p x i* **unfolding** *prob-component-def* **using** *emeasure-pmf-single pmf-bernoulli-False pmf-bernoulli-True*
by (*simp add*: *emeasure-pmf-single assms*)

lemma *stake-preimage-measurable*:
fixes *x*::*bool stream*
assumes *Suc 0* ≤ *n* **and** *M* = *bernoulli-stream p*
shows {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∈ *sets M*
proof −
let *?S* = {*w*∈ *space M*. (*stake n w* = *stake n x*)}
have *?S* = ($\bigcap$ *i* ∈ {*0*.. *n*−*1*}. {*w*∈ *space M*. (*snth w i* = *snth x i*)}) **using** *stake-inter-snth assms* **by** *simp*
moreover have ($\bigcap$ *i* ∈ {*0*.. *n*−*1*}. {*w*∈ *space M*. (*snth w i* = *snth x i*)}) ∈ *sets M*
proof −

```
    have ∀ i ≤ n−1. {w∈ space M. (snth w i = snth x i)} ∈ sets M
    proof (intro allI impI)
      fix i
      assume i ≤ n−1
      thus {w ∈ space M. w !! i = x !! i} ∈ sets M
      proof −
       have (λw. snth w i) ∈ measurable M (measure-pmf (bernoulli-pmf p)) using
assms by (simp add: assms bernoulli-stream-def)
        thus ?thesis by simp
      qed
    qed
    thus ?thesis by auto
  qed
  ultimately show ?thesis by simp
qed

lemma snth-as-fct:
  fixes b
  assumes M = bernoulli-stream p
  shows to-stream −‘ {w∈ space M. snth w i = b} = {X::nat⇒bool. X i = b}
proof −
  let ?S = {w∈ space M. snth w i = b}
  let ?PM = (λi::nat. (measure-pmf (bernoulli-pmf p)))
  have isps: product-prob-space ?PM by unfold-locales
  let ?Z = {X::nat⇒bool. X i = b}
 show to-stream −‘?S = ?Z by (simp add: assms bernoulli-stream-space to-stream-def)
qed

lemma stake-as-fct:
  assumes Suc 0 ≤ n and M= bernoulli-stream p
  shows to-stream −‘{w∈ space M. (stake n w = stake n x)} = {X::nat⇒bool. ∀ i.
0 ≤ i ∧ i ≤ n−1 ⟶ X i = snth x i}
proof −
  let ?S = {w∈ space M. (stake n w = stake n x)}
  let ?Z = {X::nat⇒bool. ∀ i. 0 ≤ i ∧ i ≤ n−1 ⟶ X i = snth x i}
  have to-stream −‘ ?S = to-stream −‘ (⋂ i ∈ {0.. n−1}. {w∈ space M. (snth w
i = snth x i)})
    using ‹Suc 0 ≤ n› stake-inter-snth by blast
  also have ... = (⋂ i ∈ {0.. n−1}. to-stream −‘{w∈ space M. (snth w i = snth
x i)}) by auto
   also have ... = (⋂ i ∈ {0.. n−1}. {X::nat⇒bool. X i = snth x i}) using
snth-as-fct assms by simp
  also have ... = ?Z by auto
  finally show ?thesis .
qed

lemma bernoulli-stream-npref-prob:
  fixes x
  assumes M = bernoulli-stream p
```

shows *emeasure M* {*w*∈ *space M*. (*stake 0 w* = *stake 0 x*)} = *1*
proof −
define *S* **where** *S* = {*w*∈ *space M*. (*stake 0 w* = *stake 0 x*)}
have *S* = *space M* **unfolding** *S-def* **by** *simp*
thus *?thesis*
by (*simp add*: *assms bernoulli-stream-def prob-space.emeasure-space-1*
prob-space.prob-space-stream-space prob-space-measure-pmf)
qed

lemma *bernoulli-stream-pref-prob*:
fixes *x*
assumes *M* =*bernoulli-stream p*
and *0* ≤ *p* **and** *p* ≤ *1*
shows *n*≥ *Suc 0*⟹ *emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} = (∏ *i*∈{*0*..*n*−*1*}. *prob-component p x i*)
proof −
have *prob-space M*
by (*simp add*: *assms bernoulli-stream-def prob-space.prob-space-stream-space prob-space-measure-pmf*)
fix *n*::*nat*
assume *n*≥ *Suc 0*
define *S* **where** *S* = {*w*∈ *space M*. (*stake n w* = *stake n x*)}
have *s*: *S* ∈ *sets M* **unfolding** *S-def* **by** (*simp add*: *assms stake-preimage-measurable* ‹*Suc 0* ≤ *n*›)
define *PM* **where** *PM* = (λ*i*::*nat*. (*measure-pmf* (*bernoulli-pmf p*)))
have *isps*: *product-prob-space PM* **unfolding** *PM-def* **by** *unfold-locales*
define *Z* **where** *Z* = {*X*::*nat*⇒*bool*. ∀ *i*. *0* ≤ *i* ∧ *i* ≤ *n*−*1* ⟶ *X i* = *snth x i*}
let *?wPM* = *Pi*$_M$ *UNIV PM*
define *imgSbs* **where** *imgSbs* = *prod-emb UNIV PM* {*0*..*n*−*1*} (*Pi*$_E$ {*0*..*n*−*1*} (λ*i*::*nat*. {*snth x i*}))
have *space ?wPM* = *UNIV* **using** *space-PiM* **unfolding** *PM-def* **by** *fastforce*
hence (*to-stream* −‘ *S* ∩ (*space ?wPM*)) = *to-stream* −‘ *S* **by** *simp*
also have ... = *Z* **using** *stake-as-fct* ‹*Suc 0* ≤ *n*› *assms* **unfolding** *Z-def S-def* **by** *simp*
also have ... = *imgSbs*
proof
{
fix *X*
assume *X* ∈ *imgSbs*
hence *restrict X* {*0*..*n*−*1*} ∈ (*Pi*$_E$ {*0*..*n*−*1*} (λ*i*::*nat*. {*snth x i*})) **using** *prod-emb-iff*[*of X*] **unfolding** *imgSbs-def* **by** *simp*
hence ∀ *i*. *0* ≤ *i* ∧ *i* ≤ *n*−*1* ⟶ *X i* = *snth x i* **by** *auto*
hence *X* ∈ *Z* **unfolding** *Z-def* **by** *simp*
}
thus *imgSbs* ⊆ *Z* **by** *blast*
{
fix *X*

assume $X \in Z$
hence $\forall i.\ 0 \leq i \wedge i \leq n-1 \longrightarrow X\ i = snth\ x\ i$ **unfolding** *Z-def* **by** *simp*
hence *restrict* $X\ \{0..n-1\} \in (Pi_E\ \{0..n-1\}\ (\lambda i{::}nat.\ \{snth\ x\ i\}))$ **by** *simp*
moreover have $X \in$ *extensional UNIV* **by** *simp*
moreover have $\forall i \in UNIV.\ X\ i \in space\ (PM\ i)$ **unfolding** *PM-def* **by** *auto*
ultimately have $X \in imgSbs$
using *prod-emb-iff*[*of X*] **unfolding** *imgSbs-def* **by** *simp*
}
thus $Z \subseteq imgSbs$ **by** *auto*
qed
finally have *inteq*: $(to\text{-}stream -` S \cap (space\ ?wPM)) = imgSbs$ **.**

have *emeasure M S* = *emeasure ?wPM* (*to-stream* −‘ *S* ∩ (*space ?wPM*))
using *emeasure-distr*[*of to-stream ?wPM M S*] *measurable-to-stream*[*of* (*measure-pmf* (*bernoulli-pmf p*))] *s assms*
unfolding *bernoulli-stream-def stream-space-def PM-def*
by (*simp add*: *emeasure-distr*)
also have ... = *emeasure ?wPM imgSbs* **using** *inteq* **by** *simp*
also have ... = $(\prod i{\in}\{0..n-1\}.\ emeasure\ (PM\ i)\ ((\lambda m{::}nat.\ \{snth\ x\ m\})\ i))$
using *isps* **unfolding** *imgSbs-def PM-def* **by** (*auto simp add*:*product-prob-space.emeasure-PiM-emb*)
also have ... = $(\prod i{\in}\{0..n-1\}.\ prob\text{-}component\ p\ x\ i)$ **using** *prob-component-measure* **unfolding** *PM-def*
proof −
have *f1*: $\forall N\ f.\ (\exists n.\ (n{::}nat) \in N \wedge \neg\ 0 \leq f\ n) \vee (\prod n{\in}N.\ ennreal\ (f\ n)) = ennreal\ (prod\ f\ N)$
by (*metis* (*no-types*) *prod-ennreal*)
obtain *nn* :: (*nat* ⇒ *real*) ⇒ *nat set* ⇒ *nat* **where**
f2: $\forall x0\ x1.\ (\exists v2.\ v2 \in x1 \wedge \neg\ 0 \leq x0\ v2) = (nn\ x0\ x1 \in x1 \wedge \neg\ 0 \leq x0\ (nn\ x0\ x1))$
by *moura*
have *f3*: $\forall s\ n.$ *if s !! n then prob-component p s n = p else p + prob-component p s n = 1*
by (*simp add*: *prob-component-def*)
{ **assume** *prob-component p x* (*nn* (*prob-component p x*) {*0..n − 1*}) ≠ *p*
then have *p + prob-component p x* (*nn* (*prob-component p x*) {*0..n − 1*}) = *1*
using *f3* **by** *metis*
then have *nn* (*prob-component p x*) {*0..n − 1*} ∉ {*0..n − 1*} ∨ *0* ≤ *prob-component p x* (*nn* (*prob-component p x*) {*0..n − 1*})
using *assms* **by** *linarith* }
then have *nn* (*prob-component p x*) {*0..n − 1*} ∉ {*0..n − 1*} ∨ *0* ≤ *prob-component p x* (*nn* (*prob-component p x*) {*0..n − 1*})
using *assms* **by** *linarith*
then have $(\prod n = 0..n - 1.\ ennreal\ (prob\text{-}component\ p\ x\ n)) = ennreal\ (prod\ (prob\text{-}component\ p\ x)\ \{0..n - 1\})$
using *f2 f1* **by** *meson*
moreover have $(\prod n = 0..n - 1.\ ennreal\ (prob\text{-}component\ p\ x\ n)) =$
$(\prod n = 0..n - 1.\ emeasure\ (measure\text{-}pmf\ (bernoulli\text{-}pmf\ p))\ \{x\ !!\ n\})$ **using** *prob-component-measure*[*of p x*]

assms **by** *simp*
ultimately show ($\prod n = 0..n - 1$. *emeasure* (*measure-pmf* (*bernoulli-pmf p*)) {*x* !! *n*}) = *ennreal* (*prod* (*prob-component p x*) {*0..n − 1*})
using *prob-component-measure*[*of p x*] **by** *simp*
qed
finally show *emeasure M S* = ($\prod i \in \{0..n-1\}$. *prob-component p x i*) **.**
qed

lemma *bernoulli-stream-pref-prob'*:
fixes *x*
assumes *M* = *bernoulli-stream p*
and *p* ≤ *1* **and** *0* ≤ *p*
shows *emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} = ($\prod i \in \{0..<n\}$. *prob-component p x i*)
proof (*cases Suc 0* ≤ *n*)
case *True*
hence *emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} = ($\prod i \in \{0..n-1\}$. *prob-component p x i*) **using** *assms*
by (*simp add*: *bernoulli-stream-pref-prob*)
moreover have ($\prod i \in \{0..n-1\}$. *prob-component p x i*) = ($\prod i \in \{0..<n\}$. *prob-component p x i*)
proof (*rule prod.cong*)
show {*0..n − 1*} = {*0..<n*} **using** *True* **by** *auto*
show $\bigwedge xa.\ xa \in \{0..<n\} \Longrightarrow$ *prob-component p x xa* = *prob-component p x xa*
by *simp*
qed
ultimately show *?thesis* **by** *simp*
next
case *False*
hence *n* = *0* **using** *False* **by** *simp*
have {*w*∈ *space M*. (*stake n w* = *stake n x*)} = *space M*
proof
show {*w* ∈ *space M*. *stake n w* = *stake n x*} ⊆ *space M*
proof
fix *w*
assume *w*∈ {*w* ∈ *space M*. *stake n w* = *stake n x*}
thus *w* ∈ *space M* **by** *auto*
qed
show *space M* ⊆ {*w* ∈ *space M*. *stake n w* = *stake n x*}
proof
fix *w*
assume *w*∈ *space M*
have *stake 0 w* = *stake 0 x* **by** *simp*
hence *stake n w* = *stake n x* **using** ‹*n* = *0*› **by** *simp*
thus *w*∈ {*w* ∈ *space M*. *stake n w* = *stake n x*} **using** ‹*w*∈ *space M*› **by** *auto*
qed
qed
hence *emeasure M* {*w* ∈ *space M*. *stake n w* = *stake n x*} = *emeasure M* (*space*

M) **by** *simp*
also have ... = *1* **using** *assms*
by (*simp add*: *bernoulli-stream-def prob-space.emeasure-space-1*
prob-space.prob-space-stream-space prob-space-measure-pmf)
also have ... = ($\prod$ *i*∈{*0*..<*n*}. *prob-component p x i*) **using** ‹*n* = *0*› **by** *simp*
finally show *?thesis* **.**
qed

lemma *bernoulli-stream-stake-prob*:
fixes *x*
assumes *M* = *bernoulli-stream p*
and *p* ≤ *1* **and** *0* ≤ *p*
shows *measure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} = ($\prod$ *i*∈{*0*..<*n*}.
prob-component p x i)
proof −
have *measure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} = *emeasure M* {*w*∈
space M. (*stake n w* = *stake n x*)}
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *bernoulli-stream-def emeasure-eq-ennreal-measure*
emeasure-space
ennreal-one-neq-top neq-top-trans prob-space.emeasure-space-1 prob-space.prob-space-stream-space
prob-space-measure-pmf)
also have ... = ($\prod$ *i*∈{*0*..<*n*}. *prob-component p x i*) **using** *bernoulli-stream-pref-prob'*
assms **by** *simp*
finally show *?thesis* **by** (*simp add*: *assms*(*2*) *assms*(*3*) *prob-component-def*
prod-nonneg)
qed

lemma (**in** *infinite-coin-toss-space*) *bernoulli-stream-pseudo-prob*:
fixes *x*
assumes *M* = *bernoulli-stream p*
and *p* ≤ *1* **and** *0* ≤ *p*
and *w*∈ *range* (*pseudo-proj-True n*)
shows *measure M* (*pseudo-proj-True n* −‘{*w*} ∩ *space M*) = ($\prod$ *i*∈{*0*..<*n*}. *prob-component*
p w i)
proof −
have (*pseudo-proj-True n* −‘{*w*}) ∩ *space M* = {*x*∈ *space M*. (*stake n w* = *stake*
n x)}
using *assms*(*4*) *infinite-coin-toss-space.pseudo-proj-True-def infinite-coin-toss-space-axioms*
pseudo-proj-True-preimage-stake pseudo-proj-True-stake **by** *force*
thus *?thesis* **using** *bernoulli-stream-stake-prob assms*
proof −
have *pseudo-proj-True n w* = *w*
using ‹*w* ∈ *range* (*pseudo-proj-True n*)› *pseudo-proj-True-proj* **by** *blast*
then show *?thesis*
using *bernoulli bernoulli-stream-stake-prob p-gt-0 p-lt-1 pseudo-proj-True-preimage-stake-space*
by *presburger*
qed
qed

lemma *bernoulli-stream-element-prob-rec*:
fixes *x*
assumes M = *bernoulli-stream p*
and $0 \leq p$ **and** $p \leq 1$
shows $\bigwedge$ *n. emeasure M* {*w*∈ *space M*. (*stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*)} =
(*emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∗ *prob-component p x n*)
proof −
fix *n*
define *S* **where** *S* = {*w*∈ *space M*. (*stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*)}
define *precS* **where** *precS* = {*w*∈ *space M*. (*stake n w* = *stake n x*)}
show *emeasure M S* = *emeasure M precS* ∗ *prob-component p x n*
proof (*cases* *n*≤ *0*)
case *True*
hence *n*=*0* **by** *simp*
hence *emeasure M S* = (∏ *i*∈{*0*..*n*}. *prob-component p x i*) **unfolding** *S-def*
using *bernoulli-stream-pref-prob assms diff-Suc-1 le-refl* **by** *presburger*
also have ... = *prob-component p x 0* **using** *True* **by** *simp*
also have ... = *emeasure M precS* ∗ *prob-component p x n* **using** *bernoulli-stream-npref-prob*
assms
by (*simp add*: ‹*n*=*0*› *precS-def*)
finally show *emeasure M S* = *emeasure M precS* ∗ *prob-component p x n* **.**
next
case *False*
hence *n* ≥ *Suc 0* **by** *simp*
hence *emeasure M S* = (∏ *i*∈{*0*..*n*}. *prob-component p x i*) **unfolding** *S-def*
using *bernoulli-stream-pref-prob diff-Suc-1 le-refl assms* **by** *fastforce*
also have ... = (∏ *i*∈{*0*..*n*−*1*}. *prob-component p x i*) ∗ *prob-component p x n*
using ‹*n* ≥ *Suc 0*›
by (*metis One-nat-def Suc-le-lessD Suc-pred prod.atLeast0-atMost-Suc*)
also have ... = *emeasure M precS* ∗ *prob-component p x n* **using** *bernoulli-stream-pref-prob*
unfolding *precS-def*
using ‹*Suc 0* ≤ *n*› *ennreal-mult''* *assms prob-component-def* **by** *auto*
finally show *emeasure M S* = *emeasure M precS* ∗ *prob-component p x n* **.**
qed
qed

lemma *bernoulli-stream-element-prob-rec'*:
fixes *x*
assumes M = *bernoulli-stream p*
and $0 \leq p$ **and** $p \leq 1$
shows $\bigwedge$ *n. measure M* {*w*∈ *space M*. (*stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*)} =
(*measure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∗ *prob-component p x n*)
proof −
fix *n*
have *ennreal* (*measure M* {*w*∈ *space M*. (*stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*)})
=
emeasure M {*w*∈ *space M*. (*stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*)}
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *bernoulli-stream-def emeasure-eq-ennreal-measure*

emeasure-space ennreal-top-neq-one neq-top-trans prob-space.emeasure-space-1
prob-space.prob-space-stream-space prob-space-measure-pmf)
also have ... = (*emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∗ *prob-component p x n*)
using *bernoulli-stream-element-prob-rec assms* **by** *simp*
also have ... = (*measure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∗ *prob-component p x n*)
proof −
have *prob-space M*
using *assms*(*1*) *bernoulli-stream-def prob-space.prob-space-stream-space prob-space-measure-pmf* **by** *auto*
then show *?thesis*
by (*simp add*: *ennreal-mult'' finite-measure.emeasure-eq-measure mult.commute prob-space-def*)
qed
finally have *ennreal* (*measure M* {*w*∈ *space M*. (*stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*)}) =
(*measure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∗ *prob-component p x n*) **.**
thus *measure M* {*w*∈ *space M*. (*stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*)} =
(*measure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∗ *prob-component p x n*)
using *assms prob-component-def* **by** *auto*
qed

lemma (**in** *infinite-coin-toss-space*) *bernoulli-stream-pseudo-prob-rec'*:
fixes *x*
assumes *pseudo-proj-True n x* = *x*
shows *measure M* (*pseudo-proj-True* (*Suc n*) −‘{*x*}) =
(*measure M* (*pseudo-proj-True n*−‘{*x*}) ∗ *prob-component p x n*)
proof −
have *pseudo-proj-True* (*Suc n*) −‘{*x*} = {*w*. (*stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*)} **using** *pseudo-proj-True-preimage-stake*
assms **by** (*metis pseudo-proj-True-Suc-proj*)
moreover have *pseudo-proj-True n* −‘{*x*} = {*w*. (*stake n w* = *stake n x*)} **using** *pseudo-proj-True-preimage-stake*
assms **by** *simp*
ultimately show *?thesis* **using** *assms bernoulli-stream-element-prob-rec'*
by (*simp add*: *bernoulli bernoulli-stream-space p-gt-0 p-lt-1*)
qed

lemma (**in** *infinite-coin-toss-space*) *bernoulli-stream-pref-prob-pos*:
fixes *x*
assumes *0* < *p*
and *p* < *1*
shows *emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} > *0*
proof (*induct n*)
case *0*
hence *emeasure M* {*w*∈ *space M*. (*stake 0 w* = *stake 0 x*)} = *1* **using** *bernoulli-stream-npref-prob*[*of M p x*]

bernoulli **by** *simp*
thus *?case* **by** *simp*
next
case (*Suc n*)
have *emeasure M* {*w* ∈ *space M*. *stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*} =
(*emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∗ *prob-component p x n*)
using *bernoulli-stream-element-prob-rec*
bernoulli p-gt-0 p-lt-1 **by** *simp*
thus *?case* **using** *Suc* **using** *assms p-gt-0 p-lt-1 prob-component-def*
by (*simp add*: *ennreal-zero-less-mult-iff*)
qed

lemma (**in** *infinite-coin-toss-space*) *bernoulli-stream-pref-prob-neq-zero*:
fixes *x*
assumes *0* < *p*
and *p* < *1*
shows *emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ≠ *0*
proof (*induct n*)
case *0*
hence *emeasure M* {*w*∈ *space M*. (*stake 0 w* = *stake 0 x*)} = *1* **using** *bernoulli-stream-npref-prob*[*of M p x*]
bernoulli **by** *simp*
thus *?case* **by** *simp*
next
case (*Suc n*)
have *emeasure M* {*w* ∈ *space M*. *stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*} =
(*emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∗ *prob-component p x n*)
using *bernoulli-stream-element-prob-rec*
bernoulli assms **by** *simp*
thus *?case* **using** *Suc* **using** *assms p-gt-0 p-lt-1 prob-component-def* **by** *auto*
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-element-prob-pref*:
assumes *w*∈ *range* (*pseudo-proj-True n*)
shows *emeasure M* {*y*∈ *space M*. ∃ *x* ∈ (*pseudo-proj-True n* −‘{*w*}). *y* = *c* ## *x*} =
prob-component p (*c*##*w*) *0* ∗ *emeasure M* ((*pseudo-proj-True n*) −‘{*w*} ∩ *space M*)
proof −
have *pseudo-proj-True n w* = *w* **using** *assms pseudo-proj-True-def pseudo-proj-True-stake* **by** *auto*
have *pseudo-proj-True* (*Suc n*) (*c*##*w*) = *c*##*w* **using** *assms*
pseudo-proj-True-def pseudo-proj-True-stake **by** *auto*
have {*y*∈ *space M*. ∃ *x* ∈ (*pseudo-proj-True n* −‘{*w*}). *y* = *c* ## *x*} = *pseudo-proj-True* (*Suc n*) −‘{*c*##*w*} ∩ *space M*
proof
show {*y*∈ *space M*. ∃ *x*∈*pseudo-proj-True n* −‘ {*w*}. *y* = *c* ## *x*} ⊆ *pseudo-proj-True*

$(Suc\ n) -` \{c\ \#\#\ w\} \cap space\ M$
proof
fix y
assume $y \in \{y \in space\ M.\ \exists x \in pseudo\text{-}proj\text{-}True\ n -` \{w\}.\ y = c\ \#\#\ x\}$
hence $y \in space\ M$ **and** $\exists x \in pseudo\text{-}proj\text{-}True\ n -` \{w\}.\ y = c\ \#\#\ x$ **by** *auto*
from *this* **obtain** x **where** $x \in pseudo\text{-}proj\text{-}True\ n -` \{w\}$ **and** $y = c\#\#\ x$ **by** *auto*
have $pseudo\text{-}proj\text{-}True\ (Suc\ n)\ y = c\#\#w$ **using** ‹$x \in pseudo\text{-}proj\text{-}True\ n -` \{w\}$› ‹$y = c\#\#\ x$›
unfolding *pseudo-proj-True-def* **by** *simp*
thus $y \in pseudo\text{-}proj\text{-}True\ (Suc\ n) -` \{c\ \#\#\ w\} \cap space\ M$ **using** ‹$y \in space\ M$› **by** *auto*
qed
show $pseudo\text{-}proj\text{-}True\ (Suc\ n) -` \{c\ \#\#\ w\} \cap space\ M \subseteq \{y \in space\ M.\ \exists x \in pseudo\text{-}proj\text{-}True\ n -` \{w\}.\ y = c\ \#\#\ x\}$
proof
fix y
assume $y \in pseudo\text{-}proj\text{-}True\ (Suc\ n) -` \{c\ \#\#\ w\} \cap space\ M$
hence $pseudo\text{-}proj\text{-}True\ (Suc\ n)\ y = c\#\#w$ **and** $y \in space\ M$ **by** *auto*
have $pseudo\text{-}proj\text{-}True\ n\ (stl\ y) = pseudo\text{-}proj\text{-}True\ n\ w$
proof (*rule pseudo-proj-True-snth'*)
have $pseudo\text{-}proj\text{-}True\ (Suc\ n)\ (c\#\#w) = c\#\#w$ **using** ‹$pseudo\text{-}proj\text{-}True\ (Suc\ n)\ (c\#\#w) = c\#\#w$› **.**
also have ... $= pseudo\text{-}proj\text{-}True\ (Suc\ n)\ y$ **using** ‹$pseudo\text{-}proj\text{-}True\ (Suc\ n)\ y = c\#\#w$› **by** *simp*
finally have $pseudo\text{-}proj\text{-}True\ (Suc\ n)\ (c\#\#w) = pseudo\text{-}proj\text{-}True\ (Suc\ n)\ y$ **.**
hence $\bigwedge i.\ Suc\ i \leq Suc\ n \Longrightarrow (c\#\#w)!!\ i = y!!\ i$ **by** (*simp add*: *pseudo-proj-True-snth*)
thus $\bigwedge i.\ Suc\ i \leq n \Longrightarrow stl\ y\ !!\ i = w\ !!\ i$ **by** *fastforce*
qed
also have ... $= w$ **using** *assms pseudo-proj-True-def pseudo-proj-True-stake* **by** *auto*
finally have $pseudo\text{-}proj\text{-}True\ n\ (stl\ y) = w$ **.**
hence $stl\ y \in (pseudo\text{-}proj\text{-}True\ n) -` \{w\}$ **by** *simp*
moreover have $y = c\#\#(stl\ y)$
proof −
have $stake\ (Suc\ n)\ y = stake\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)\ y)$ **unfolding** *pseudo-proj-True-def*
using *pseudo-proj-True-def pseudo-proj-True-stake* **by** *auto*
hence $shd\ y = shd\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)\ y)$ **by** *simp*
also have ... $= shd\ (c\#\#w)$ **using** ‹$pseudo\text{-}proj\text{-}True\ (Suc\ n)\ y = c\#\#w$› **by** *simp*
also have ... $= c$ **by** *simp*
finally have $shd\ y = c$ **.**
thus *?thesis* **by** (*simp add*: *stream-eq-Stream-iff*)
qed
ultimately show $y \in \{y \in space\ M.\ \exists x \in pseudo\text{-}proj\text{-}True\ n -` \{w\}.\ y = c\ \#\#\ x\}$ **using** ‹$y \in space\ M$› **by** *auto*

```
    qed
  qed
  hence emeasure M {y∈ space M. ∃ x ∈ (pseudo-proj-True n −‘{w}). y = c ##
x} =
    emeasure M (pseudo-proj-True (Suc n) −‘{c##w}∩ space M) by simp
   also have ... = emeasure M {y∈ space M. stake (Suc n) y = stake (Suc n)
(c##w)}
    using ‹pseudo-proj-True (Suc n) (c##w) = c##w› by (simp add:pseudo-proj-True-preimage-stake-space)
  also have ... = (∏ i∈{0..n}. prob-component p (c##w) i)
     using bernoulli-stream-pref-prob[of M p Suc n c##w] bernoulli p-lt-1 p-gt-0
diff-Suc-1 le-refl by simp
  also have ... = prob-component p (c##w) 0 ∗ (∏ i∈{1..n}. prob-component p
(c##w) i)
    by (simp add: decompose-init-prod)
 also have ... = prob-component p (c##w) 0 ∗ (∏ i∈{1..< Suc n}. prob-component
p (c##w) i)
  proof −
   have (∏ i∈{1..n}. prob-component p (c##w) i) = (∏ i∈{1..< Suc n}. prob-component
p (c##w) i)
    proof (rule prod.cong)
      show {1..n} = {1..<Suc n} by auto
    show ⋀x. x ∈ {1..<Suc n} ⟹ prob-component p (c ## w) x = prob-component
p (c ## w) x by simp
    qed
    thus ?thesis by simp
  qed
  also have ... = prob-component p (c##w) 0 ∗ (∏ i∈{0..< n}. prob-component
p w i)
  proof −
     have (∏ i∈{1..< Suc n}. prob-component p (c##w) i) = (∏ i∈{0..< n}.
prob-component p w i)
    proof (rule prod.reindex-cong)
      show inj-on (λn. Suc n) {0..<n} by simp
      show {1..< Suc n} = Suc ‘ {0..< n} by auto
        show ⋀x. x ∈ {0..< n} ⟹ prob-component p (c ## w) (Suc x) =
prob-component p w x
       by (simp add: prob-component-def)
    qed
    thus ?thesis by simp
  qed
  also have ... = prob-component p (c##w) 0 ∗ emeasure M {y ∈ space M. stake
n y = stake n w}
    using bernoulli-stream-pref-prob'[symmetric, of M p w n] ennreal-mult' p-gt-0
p-lt-1 bernoulli
    prob-component-def by auto
  also have ... = prob-component p (c##w) 0 ∗ emeasure M (pseudo-proj-True n
−‘ {w} ∩ space M)
    using pseudo-proj-True-preimage-stake-space ‹pseudo-proj-True n w = w›
    by (simp add: pseudo-proj-True-preimage-stake-space)
```

finally show *?thesis* **.**
qed

5.3.4 Filtration equivalence for the natural filtration

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-null-set*:
assumes $A \in$ *sets* (*nat-filtration n*)
and $0 < p$
and $p < 1$
and *emeasure M A* $= 0$
shows $A = \{\}$
proof (*rule ccontr*)
assume $A \neq \{\}$
hence $\exists w.\ w \in A$ **by** *auto*
from *this* **obtain** w **where** $w \in A$ **by** *auto*
hence *inc*: *pseudo-proj-True n* $-$ ‘ $\{$*pseudo-proj-True n w*$\} \subseteq A$ **using** *assms* **by** (*simp add*: *set-filt-contain*)
have $0 <$ *emeasure M* $\{x \in$ *space M*. (*stake n x* = *stake n* (*pseudo-proj-True n w*))$\}$ **using** *assms* **by** (*simp add*: *bernoulli-stream-pref-prob-pos*)
also have ... = *emeasure M* (*pseudo-proj-True n* $-$ ‘ $\{$*pseudo-proj-True n w*$\}$)
using *pseudo-proj-True-preimage-stake*
pseudo-proj-True-proj bernoulli bernoulli-stream-space **by** *simp*
also have ... $\leq$ *emeasure M A*
proof (*rule emeasure-mono*, (*simp add*: *inc*))
show $A \in$ *events* **using** *assms nat-discrete-filtration* **unfolding** *filtration-def subalgebra-def* **by** *auto*
qed
finally have $0 <$ *emeasure M A* **.**
thus *False* **using** *assms* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-AE-zero*:
fixes *f*::*bool stream* $\Rightarrow$ *real*
assumes *AE w in M*. *f w* $= 0$
and $f \in$ *borel-measurable* (*nat-filtration n*)
and $0 < p$
and $p < 1$
shows $\forall w.\ f\ w = 0$
proof $-$
from ‹*AE w in M*. *f w* $= 0$› **obtain** N' **where** *Nprops*: $\{w \in$ *space M*. $\neg f\ w = 0\} \subseteq N'$ $N' \in$ *sets M emeasure M* $N' = 0$
by (*force elim*:*AE-E*)
have $\{w \in$ *space M*. $f\ w < 0\} \in$ *sets* (*nat-filtration n*)
by (*metis* (*no-types*) *assms*(*2*) *bernoulli bernoulli-stream-space borel-measurable-iff-less nat-filtration-space streams-UNIV*)
moreover have $\{w \in$ *space M*. $f\ w > 0\} \in$ *sets* (*nat-filtration n*)
by (*metis* (*no-types*) *assms*(*2*) *bernoulli bernoulli-stream-space borel-measurable-iff-greater nat-filtration-space streams-UNIV*)
moreover have $\{w \in$ *space M*. $\neg f\ w = 0\} = \{w \in$ *space M*. $f\ w < 0\} \cup \{w \in$

space M. f w > 0} **by** *auto*
ultimately have {*w*∈ *space M*. ¬*f w* = *0*} ∈ *sets* (*nat-filtration n*) **by** *auto*
hence *emeasure M* {*w*∈ *space M*. ¬*f w* = *0*} = *0* **using** *Nprops* **by** (*metis* (*no-types*, *lifting*) *emeasure-eq-0*)
hence {*w*∈ *space M*. ¬*f w* = *0*} = {} **using** ‹{*w*∈ *space M*. ¬*f w* = *0*} ∈ *sets* (*nat-filtration n*)›
nat-filtration-null-set[*of* {*w* ∈ *space M*. *f w* ≠ *0*} *n*] *assms* **by** *simp*
hence {*w*. *f w*≠ *0*} = {} **by** (*simp add*:*bernoulli-stream-space bernoulli*)
thus *?thesis* **by** *auto*
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-AE-eq*:
fixes *f*::*bool stream* ⇒ *real*
assumes *AE w in M*. *f w* = *g w*
and *0* < *p*
and *p* < *1*
and *f*∈ *borel-measurable* (*nat-filtration n*)
and *g*∈ *borel-measurable* (*nat-filtration n*)
shows *f w* = *g w*
proof −
define *diff* **where** *diff* = (λ*w*. *f w* − *g w*)
have *AE w in M*. *diff w* = *0*
proof (*rule AE-mp*)
show *AE w in M*. *f w* = *g w* **using** *assms* **by** *simp*
show *AE w in M*. *f w* = *g w* ⟶ *diff w* = *0*
by (*rule AE-I2*, *intro impI*, (*simp add*: *diff-def*))
qed
have ∀ *w*. *diff w* = *0*
proof (*rule nat-filtration-AE-zero*)
show *AE w in M*. *diff w* = *0* **using** ‹*AE w in M*. *diff w* = *0*› **.**
show *diff* ∈ *borel-measurable* (*nat-filtration n*) **using** *assms* **unfolding** *diff-def* **by** *simp*
show *0* < *p* **and** *p* < *1* **using** *assms* **by** *auto*
qed
thus *f w* = *g w* **unfolding** *diff-def* **by** *auto*
qed

lemma (**in** *infinite-coin-toss-space*) *bernoulli-stream-equiv*:
assumes *N* = *bernoulli-stream q*
and *0* < *p*
and *p* < *1*
and *0* < *q*
and *q* < *1*
shows *filt-equiv nat-filtration M N* **unfolding** *filt-equiv-def*
proof (*intro conjI*)
have *sets* (*stream-space* (*measure-pmf* (*bernoulli-pmf p*))) = *sets* (*stream-space*

(*measure-pmf* (*bernoulli-pmf q*)))
by (*rule sets-stream-space-cong*, *simp*)
thus *events* = *sets N* **using** *assms bernoulli* **unfolding** *bernoulli-stream-def* **by** *simp*
show *filtration M nat-filtration* **by** (*simp add*:*nat-discrete-filtration*)
show ∀ *t A*. *A* ∈ *sets* (*nat-filtration t*) ⟶ (*emeasure M A* = *0*) = (*emeasure N A* = *0*)
proof (*intro allI impI*)
fix *n*
fix *A*
assume *A*∈ *sets* (*nat-filtration n*)
show (*emeasure M A* = *0*) = (*emeasure N A* = *0*)
proof
{
assume *emeasure M A* = *0*
hence *A* = {} **using** ‹*A*∈ *sets* (*nat-filtration n*)› **using** *assms* **by** (*simp add*:*nat-filtration-null-set*)
thus *emeasure N A* = *0* **by** *simp*
}
{
assume *emeasure N A* = *0*
hence *A* = {} **using** ‹*A*∈ *sets* (*nat-filtration n*)› *infinite-coin-toss-space.nat-filtration-null-set*[*of q N A n*]
assms
using ‹*events* = *sets N*› *bernoulli bernoulli-stream-space infinite-coin-toss-space.nat-filtration-sets*
infinite-coin-toss-space-def nat-filtration-sets **by** *force*
thus *emeasure M A* = *0* **by** *simp*
}
qed
qed
qed

lemma (**in** *infinite-coin-toss-space*) *bernoulli-nat-filtration*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *0* < *p*
and *p* < *1*
shows *infinite-cts-filtration q N nat-filtration*
proof (*unfold-locales*)
have *0* < *q* **using** *assms* **by** *simp*
thus *0* ≤ *q* **by** *simp*
have *q* < *1* **using** *assms* **by** *simp*
thus *q* ≤ *1* **by** *simp*
show *N* = *bernoulli-stream q* **using** *assms* **by** *simp*
show *nat-filtration* = *infinite-coin-toss-space.nat-filtration N*
proof −
have *filt-equiv nat-filtration M N* **using** ‹*q* < *1*› ‹*0* < *q*›
by (*simp add*: *assms bernoulli-stream-equiv*)

hence *sets M = sets N* **unfolding** *filt-equiv-def* **by** *simp*
hence *space M = space N* **using** *sets-eq-imp-space-eq* **by** *auto*
have $\forall$ *m. nat-filtration m = infinite-coin-toss-space.nat-filtration N m*
proof
fix *m*
have *infinite-coin-toss-space.nat-filtration N m = fct-gen-subalgebra N N (pseudo-proj-True m)*
using ‹*0 ≤ q*› ‹*N = bernoulli-stream q*› ‹*q ≤ 1*› *infinite-coin-toss-space.intro infinite-coin-toss-space.nat-filtration-def* **by** *blast*
thus *nat-filtration m = infinite-coin-toss-space.nat-filtration N m*
unfolding *nat-filtration-def*
using *fct-gen-subalgebra-cong*[*of M N M N pseudo-proj-True m*] ‹*sets M = sets N*› ‹*space M = space N*›
by *simp*
qed
thus *?thesis* **by** *auto*
qed
qed

5.3.5 More results on the projection function

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-Suc-prefix*:
shows *pseudo-proj-True (Suc n) w = (w!!0)## pseudo-proj-True n (stl w)*
proof −
have *pseudo-proj-True (Suc n) w = shift (stake (Suc n) w) (sconst True)* **unfolding** *pseudo-proj-True-def* **by** *simp*
also have *... = shift (w!!0 # (stake n (stl w))) (sconst True)* **by** *simp*
also have *... = w!!0 ## shift (stake n (stl w)) (sconst True)* **by** *simp*
also have *... = w!!0 ## pseudo-proj-True n (stl w)* **unfolding** *pseudo-proj-True-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-img*:
assumes *pseudo-proj-True n w = w*
shows *w∈ range (pseudo-proj-True n)*
by (*metis assms rangeI*)

lemma (**in** *infinite-coin-toss-space*) *sconst-if*:
assumes $\bigwedge$*n. snth w n = True*
shows *w = sconst True*
by (*metis* (*full-types*) *assms diff-streams-only-if id-apply id-funpow snth-siterate*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-suc-img-pref*:
shows *range (pseudo-proj-True (Suc n)) = {y. ∃ w ∈ range (pseudo-proj-True n). y = True ## w} ∪*
{y. ∃ w ∈ range (pseudo-proj-True n). y = False ## w}
proof
show *range (pseudo-proj-True (Suc n))*

$\subseteq \{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = True\ \#\#\ w\} \cup \{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = False\ \#\#\ w\}$
proof
fix x
assume $x \in range\ (pseudo\text{-}proj\text{-}True\ (Suc\ n))$
hence $x = pseudo\text{-}proj\text{-}True\ (Suc\ n)\ x$ **using** *pseudo-proj-True-proj* **by** *auto*
define xp **where** $xp = stl\ x$
have $xp = stl\ (shift\ (stake\ (Suc\ n)\ x)\ (sconst\ True))$ **using** ‹$x = pseudo\text{-}proj\text{-}True\ (Suc\ n)\ x$›
unfolding *xp-def pseudo-proj-True-def* **by** *simp*
also have $... = shift\ ((stake\ n\ (stl\ x)))\ (sconst\ True)$ **by** *simp*
finally have $xp = shift\ ((stake\ n\ (stl\ x)))\ (sconst\ True)$ **.**
hence $xp \in range\ (pseudo\text{-}proj\text{-}True\ n)$ **using** *pseudo-proj-True-def* **by** *auto*
show $x \in \{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n)\ .\ y = True\ \#\#\ w\} \cup \{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = False\ \#\#\ w\}$
proof (*cases snth x 0*)
case *True*
have $x = True\ \#\#\ xp$ **unfolding** *xp-def* **using** *True* **by** (*simp add*: *stream-eq-Stream-iff*)
hence $x \in \{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = True\ \#\#\ w\}$ **using** ‹$xp \in range\ (pseudo\text{-}proj\text{-}True\ n)$› **by** *auto*
thus *?thesis* **by** *auto*
next
case *False*
have $x = False\ \#\#\ xp$ **unfolding** *xp-def* **using** *False* **by** (*simp add*: *stream-eq-Stream-iff*)
hence $x \in \{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = False\ \#\#\ w\}$ **using** ‹$xp \in range\ (pseudo\text{-}proj\text{-}True\ n)$› **by** *auto*
thus *?thesis* **by** *auto*
qed
qed
have $\{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n)\ .\ y = True\ \#\#\ w\} \subseteq range\ (pseudo\text{-}proj\text{-}True\ (Suc\ n))$
proof
fix y
assume $y \in \{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n)\ .\ y = True\ \#\#\ w\}$
hence $\exists\, w.\ w \in range\ (pseudo\text{-}proj\text{-}True\ n) \wedge y = True\ \#\#\ w$ **by** *auto*
from *this* **obtain** w **where** $w \in range\ (pseudo\text{-}proj\text{-}True\ n)$ **and** $y = True\ \#\#\ w$ **by** *auto*
have $w = pseudo\text{-}proj\text{-}True\ n\ w$ **using** *pseudo-proj-True-proj* ‹$w \in range\ (pseudo\text{-}proj\text{-}True\ n)$› **by** *auto*
hence $y = True\ \#\#\ (shift\ (stake\ n\ w)\ (sconst\ True))$ **using** ‹$y = True\ \#\#\ w$› **unfolding** *pseudo-proj-True-def* **by** *simp*
also have $... = shift\ (stake\ (Suc\ n)\ (True\ \#\#\ w))\ (sconst\ True)$ **by** *simp*
also have $... = pseudo\text{-}proj\text{-}True\ (Suc\ n)\ (True\ \#\#\ w)$ **unfolding** *pseudo-proj-True-def* **by** *simp*
finally have $y = pseudo\text{-}proj\text{-}True\ (Suc\ n)\ (True \#\# w)$ **.**
thus $y \in range\ (pseudo\text{-}proj\text{-}True\ (Suc\ n))$ **by** *simp*
qed

moreover have $\{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n)\ .\ y = False\ \#\#\ w\} \subseteq range\ (pseudo\text{-}proj\text{-}True\ (Suc\ n))$
proof
fix y
assume $y \in \{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n)\ .\ y = False\ \#\#\ w\}$
hence $\exists\, w.\ w \in range\ (pseudo\text{-}proj\text{-}True\ n) \wedge y = False\ \#\#\ w$ **by** *auto*
from *this* **obtain** w **where** $w \in range\ (pseudo\text{-}proj\text{-}True\ n)$ **and** $y = False\ \#\#\ w$ **by** *auto*
have $w = pseudo\text{-}proj\text{-}True\ n\ w$ **using** *pseudo-proj-True-proj* ‹$w \in range\ (pseudo\text{-}proj\text{-}True\ n)$› **by** *auto*
hence $y = False\ \#\#\ (shift\ (stake\ n\ w)\ (sconst\ True))$ **using** ‹$y = False\ \#\#\ w$› **unfolding** *pseudo-proj-True-def* **by** *simp*
also have $... = shift\ (stake\ (Suc\ n)\ (False\ \#\#\ w))\ (sconst\ True)$ **by** *simp*
also have $... = pseudo\text{-}proj\text{-}True\ (Suc\ n)\ (False\ \#\#\ w)$ **unfolding** *pseudo-proj-True-def* **by** *simp*
finally have $y = pseudo\text{-}proj\text{-}True\ (Suc\ n)\ (False\#\#w)$ **.**
thus $y \in range\ (pseudo\text{-}proj\text{-}True\ (Suc\ n))$ **by** *simp*
qed
ultimately show $\{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n)\ .\ y = True\ \#\#\ w\} \cup \{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n)\ .\ y = False\ \#\#\ w\} \subseteq range\ (pseudo\text{-}proj\text{-}True\ (Suc\ n))$ **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *reindex-pseudo-proj*:
shows $(\sum w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ f\ (c\ \#\#\ w)) = (\sum y \in \{y.\ \exists\, w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = c\ \#\#\ w\}.f\ y)$
proof (*rule sum.reindex-cong*[*symmetric*],*auto*)
define *ccons* **where** $ccons = (\lambda w.\ c\#\#\ w)$
show *inj-on ccons* $(range\ (pseudo\text{-}proj\text{-}True\ n))$
proof
fix $x\ y$
assume $x \in range\ (pseudo\text{-}proj\text{-}True\ n)$ **and** $y \in range\ (pseudo\text{-}proj\text{-}True\ n)$ **and** $ccons\ x = ccons\ y$
hence $c\#\#x = c\#\#y$ **unfolding** *ccons-def* **by** *simp*
thus $x = y$ **by** *simp*
qed
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-imp-False*:
assumes $pseudo\text{-}proj\text{-}True\ n\ w = pseudo\text{-}proj\text{-}True\ n\ x$
shows $pseudo\text{-}proj\text{-}False\ n\ w = pseudo\text{-}proj\text{-}False\ n\ x$
by (*metis assms pseudo-proj-False-def pseudo-proj-True-stake*)

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-Suc-prefix*:
assumes $pseudo\text{-}proj\text{-}True\ n\ w = pseudo\text{-}proj\text{-}True\ n\ x$
shows $pseudo\text{-}proj\text{-}True\ (Suc\ n)\ w \in \{pseudo\text{-}proj\text{-}True\ n\ x,\ pseudo\text{-}proj\text{-}False\ n\ x\}$

proof −
have *pseudo-proj-False n w* = *pseudo-proj-False n x* **using** *assms pseudo-proj-True-imp-False*[*of n w x*] **by** *simp*
hence {*pseudo-proj-True n w*, *pseudo-proj-False n w*} = {*pseudo-proj-True n x*, *pseudo-proj-False n x*} **using** *assms* **by** *simp*
thus *?thesis* **using** *pseudo-proj-True-suc-img*[*of n w*] **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-Suc-preimage*:
shows *range* (*pseudo-proj-True* (*Suc n*)) ∩ (*pseudo-proj-True n*) −‘ {*pseudo-proj-True n x*} =
{*pseudo-proj-True n x*, *pseudo-proj-False n x*}
proof
show *range* (*pseudo-proj-True* (*Suc n*)) ∩ *pseudo-proj-True n* −‘ {*pseudo-proj-True n x*}
⊆ {*pseudo-proj-True n x*, *pseudo-proj-False n x*}
proof
fix *w*
assume *w*∈ *range* (*pseudo-proj-True* (*Suc n*)) ∩ *pseudo-proj-True n* −‘ {*pseudo-proj-True n x*}
hence *w*∈ *range* (*pseudo-proj-True* (*Suc n*)) **and** *w*∈ *pseudo-proj-True n* −‘ {*pseudo-proj-True n x*} **by** *auto*
hence *pseudo-proj-True n w* = *pseudo-proj-True n x* **by** *simp*
have *w* = *pseudo-proj-True* (*Suc n*) *w* **using** ‹*w*∈ *range* (*pseudo-proj-True* (*Suc n*))›
using *pseudo-proj-True-proj* **by** *auto*
also have ... ∈ {*pseudo-proj-True n x*, *pseudo-proj-False n x*} **using** ‹*pseudo-proj-True n w* = *pseudo-proj-True n x*›
pseudo-proj-Suc-prefix **by** *simp*
finally show *w* ∈ {*pseudo-proj-True n x*, *pseudo-proj-False n x*} **.**
qed
show {*pseudo-proj-True n x*, *pseudo-proj-False n x*}
⊆ *range* (*pseudo-proj-True* (*Suc n*)) ∩ *pseudo-proj-True n* −‘ {*pseudo-proj-True n x*}
proof −
have *pseudo-proj-True n x* ∈ *range* (*pseudo-proj-True* (*Suc n*)) ∩ *pseudo-proj-True n* −‘ {*pseudo-proj-True n x*}
by (*simp add*: *pseudo-proj-True-Suc-proj pseudo-proj-True-img pseudo-proj-True-proj*)
moreover have *pseudo-proj-False n x* ∈ *range* (*pseudo-proj-True* (*Suc n*)) ∩ *pseudo-proj-True n* −‘ {*pseudo-proj-True n x*}
by (*metis* (*no-types*, *lifting*) *Int-iff UnI2 infinite-coin-toss-space.pseudo-proj-False-def infinite-coin-toss-space-axioms*
pseudo-proj-True-Suc-False-proj pseudo-proj-True-inverse-induct pseudo-proj-True-stake rangeI singletonI vimage-eq)
ultimately show *?thesis* **by** *auto*
qed
qed

lemma (**in** *infinite-cts-filtration*) *f-borel-Suc-preimage*:
assumes $f \in$ *measurable* $(F\ n)\ N$
and *set-discriminating* $n\ f\ N$
shows *range* (*pseudo-proj-True* (*Suc* n)) $\cap\ f\ -`\ \{f\ x\} =$
(*pseudo-proj-True* n) `` $(f\ -`\ \{f\ x\}) \cup$ (*pseudo-proj-False* $n$) `` $(f\ -`\ \{f\ x\})$
proof −
have *range* (*pseudo-proj-True* (*Suc* n)) $\cap\ f\ -`\ \{f\ x\} =$
$(\bigcup\ w \in \{y.\ f\ y = f\ x\}.\{$*pseudo-proj-True* $n\ w$, *pseudo-proj-False* $n\ w\}$)
proof
show *range* (*pseudo-proj-True* (*Suc* n)) $\cap\ f\ -`\ \{f\ x\} \subseteq (\bigcup w \in \{y.\ f\ y = f\ x\}.$
$\{$*pseudo-proj-True* $n\ w$, *pseudo-proj-False* $n\ w\}$)
proof
fix w
assume $w \in$ *range* (*pseudo-proj-True* (*Suc* n)) $\cap\ f\ -`\ \{f\ x\}$
hence $w \in$ *range* (*pseudo-proj-True* (*Suc* n)) **and** $w \in f\ -`\ \{f\ x\}$ **by** *auto*
hence $f\ w = f\ x$ **by** *simp*
hence $w \in \{y.\ f\ y = f\ x\}$ **by** *simp*
have $w =$ *pseudo-proj-True* (*Suc* n) w **using** ‹$w \in$ *range* (*pseudo-proj-True* (*Suc* n))›
using *pseudo-proj-True-proj* **by** *auto*
also have ... $\in \{$*pseudo-proj-True* $n\ w$, *pseudo-proj-False* $n\ w\}$
using *pseudo-proj-Suc-prefix* **by** *auto*
also have ... $\subseteq (\bigcup w \in \{y.\ f\ y = f\ x\}.\ \{$*pseudo-proj-True* $n\ w$, *pseudo-proj-False* $n\ w\})$ **using** ‹$w \in \{y.\ f\ y = f\ x\}$›
by *auto*
finally show $w \in (\bigcup w \in \{y.\ f\ y = f\ x\}.\ \{$*pseudo-proj-True* $n\ w$, *pseudo-proj-False* $n\ w\})$ **.**
qed
show $(\bigcup w \in \{y.\ f\ y = f\ x\}.\ \{$*pseudo-proj-True* $n\ w$, *pseudo-proj-False* $n\ w\})$
$\subseteq$ *range* (*pseudo-proj-True* (*Suc* n)) $\cap\ f\ -`\ \{f\ x\}$
proof
fix w
assume $w \in (\bigcup w \in \{y.\ f\ y = f\ x\}.\ \{$*pseudo-proj-True* $n\ w$, *pseudo-proj-False* $n\ w\})$
hence $\exists\, y.\ f\ y = f\ x \wedge w \in \{$*pseudo-proj-True* $n\ y$, *pseudo-proj-False* $n\ y\}$ **by** *auto*
from *this* **obtain** y **where** $f\ y = f\ x$ **and** $w \in \{$*pseudo-proj-True* $n\ y$, *pseudo-proj-False* $n\ y\}$ **by** *auto*
hence $w =$ *pseudo-proj-True* $n\ y \vee w =$ *pseudo-proj-False* $n\ y$ **by** *auto*
show $w \in$ *range* (*pseudo-proj-True* (*Suc* n)) $\cap\ f\ -`\ \{f\ x\}$
proof (*cases* $w =$ *pseudo-proj-True* $n\ y$)
case *True*
hence $f\ w = f\ y$ **using** *assms nat-filtration-not-borel-info natural-filtration*
by (*metis comp-apply*)
thus *?thesis* **using** ‹$f\ y = f\ x$›
by (*simp add*: *True pseudo-proj-True-Suc-proj pseudo-proj-True-img*)
next
case *False*

hence $f\ w = f\ y$ **using** *assms nat-filtration-not-borel-info natural-filtration*
by (*metis Int-iff* ‹$w \in \{$*pseudo-proj-True n y*, *pseudo-proj-False n y*$\}$› *comp-apply pseudo-proj-Suc-preimage singletonD vimage-eq*)
thus *?thesis* **using** ‹$f\ y = f\ x$›
using ‹$w \in \{$*pseudo-proj-True n y*, *pseudo-proj-False n y*$\}$› *pseudo-proj-Suc-preimage*
by *auto*
qed
qed
qed
also have ... =
($\bigcup$ $w \in \{y.\ f\ y = f\ x\}.\{$*pseudo-proj-True n w*$\}$) $\cup$ ($\bigcup$ $w \in \{y.\ f\ y = f\ x\}.\{$*pseudo-proj-False n w*$\}$) **by** *auto*
also have ... = (*pseudo-proj-True n*) ‘ $\{y.\ f\ y = f\ x\}$ $\cup$ (*pseudo-proj-False n*) ‘$\{y.\ f\ y = f\ x\}$ **by** *auto*
also have ... = (*pseudo-proj-True n*) ‘ (f −‘ $\{f\ x\}$) $\cup$ (*pseudo-proj-False n*) ‘ (f −‘ $\{f\ x\}$) **by** *auto*
finally show *?thesis* **.**
qed

lemma (**in** *infinite-cts-filtration*) *pseudo-proj-preimage*:
assumes $g \in$ *measurable* ($F\ n$) N
and *set-discriminating n g N*
shows *pseudo-proj-True n* −‘ (g −‘ $\{g\ z\}$) = *pseudo-proj-True n* −‘ (*pseudo-proj-True n* ‘(g −‘ $\{g\ z\}$))
proof
show *pseudo-proj-True n* −‘ g −‘ $\{g\ z\} \subseteq$ *pseudo-proj-True n* −‘ *pseudo-proj-True n* ‘ g −‘ $\{g\ z\}$
proof
fix w
assume $w \in$ *pseudo-proj-True n* −‘ g −‘ $\{g\ z\}$
have *pseudo-proj-True n w* = *pseudo-proj-True n* (*pseudo-proj-True n w*)
by (*simp add*: *pseudo-proj-True-proj*)
also have ... $\in$ *pseudo-proj-True n* ‘(g −‘ $\{g\ z\}$) **using** ‹$w \in$ *pseudo-proj-True n* −‘ g −‘ $\{g\ z\}$›
by *simp*
finally have *pseudo-proj-True n w* $\in$ *pseudo-proj-True n* ‘(g −‘ $\{g\ z\}$) **.**
thus $w \in$ *pseudo-proj-True n* −‘ (*pseudo-proj-True n* ‘(g −‘ $\{g\ z\}$)) **by** *simp*
qed
show *pseudo-proj-True n* −‘ *pseudo-proj-True n* ‘ g −‘ $\{g\ z\} \subseteq$ *pseudo-proj-True n* −‘ g −‘ $\{g\ z\}$
proof
fix w
assume $w \in$ *pseudo-proj-True n* −‘ *pseudo-proj-True n* ‘ g −‘ $\{g\ z\}$
hence $\exists\, y.$ *pseudo-proj-True n w* = *pseudo-proj-True n y* $\wedge$ $g\ y = g\ z$ **by** *auto*
from *this* **obtain** y **where** *pseudo-proj-True n w* = *pseudo-proj-True n y* **and** $g\ y = g\ z$ **by** *auto*
have g (*pseudo-proj-True n w*) = g (*pseudo-proj-True n y*) **using** ‹*pseudo-proj-True*

n w = pseudo-proj-True n y›
by *simp*
also have ... = *g y* **using** *assms nat-filtration-not-borel-info natural-filtration*
by (*metis comp-apply*)
also have ... = *g z* **using** ‹*g y = g z*› **.**
finally have *g* (*pseudo-proj-True n w*) = *g z* **.**
thus *w*∈ *pseudo-proj-True n −‘ g −‘* {*g z*} **by** *simp*
qed
qed

lemma (**in** *infinite-cts-filtration*) *borel-pseudo-proj-preimage*:
fixes *g*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *g*∈ *borel-measurable* (*F n*)
shows *pseudo-proj-True n −‘* (*g −‘* {*g z*}) = *pseudo-proj-True n −‘* (*pseudo-proj-True n ‘*(*g −‘* {*g z*}))
using *pseudo-proj-preimage*[*of g n borel z*] *set-discriminating-if*[*of g n*] *natural-filtration assms* **by** *simp*

lemma (**in** *infinite-cts-filtration*) *pseudo-proj-False-preimage*:
assumes *g*∈ *measurable* (*F n*) *N*
and *set-discriminating n g N*
shows *pseudo-proj-False n −‘* (*g −‘* {*g z*}) = *pseudo-proj-False n −‘* (*pseudo-proj-False n ‘*(*g −‘* {*g z*}))
proof
show *pseudo-proj-False n −‘ g −‘* {*g z*} ⊆ *pseudo-proj-False n −‘ pseudo-proj-False n ‘ g −‘* {*g z*}
proof
fix *w*
assume *w*∈ *pseudo-proj-False n −‘ g −‘* {*g z*}
have *pseudo-proj-False n w = pseudo-proj-False n* (*pseudo-proj-False n w*)
using *pseudo-proj-False-def pseudo-proj-False-stake* **by** *auto*
also have ... ∈ *pseudo-proj-False n ‘*(*g −‘* {*g z*}) **using** ‹*w*∈ *pseudo-proj-False n −‘ g −‘* {*g z*}›
by *simp*
finally have *pseudo-proj-False n w* ∈ *pseudo-proj-False n ‘*(*g −‘* {*g z*}) **.**
thus *w*∈ *pseudo-proj-False n −‘* (*pseudo-proj-False n ‘*(*g −‘* {*g z*})) **by** *simp*
qed
show *pseudo-proj-False n −‘ pseudo-proj-False n ‘ g −‘* {*g z*} ⊆ *pseudo-proj-False n −‘ g −‘* {*g z*}
proof
fix *w*
assume *w* ∈ *pseudo-proj-False n −‘ pseudo-proj-False n ‘ g −‘* {*g z*}
hence ∃ *y*. *pseudo-proj-False n w = pseudo-proj-False n y* ∧ *g y = g z* **by** *auto*
from *this* **obtain** *y* **where** *pseudo-proj-False n w = pseudo-proj-False n y* **and** *g y = g z* **by** *auto*
have *g* (*pseudo-proj-False n w*) = *g* (*pseudo-proj-False n y*) **using** ‹*pseudo-proj-False n w = pseudo-proj-False n y*›
by *simp*

also have ... = *g y* **using** *assms nat-filtration-not-borel-info′ natural-filtration*
by (*metis comp-apply*)
also have ... = *g z* **using** ‹*g y* = *g z*› **.**
finally have *g* (*pseudo-proj-False n w*) = *g z* **.**
thus *w*∈ *pseudo-proj-False n* −‘ *g* −‘ {*g z*} **by** *simp*
qed
qed

lemma (**in** *infinite-cts-filtration*) *borel-pseudo-proj-False-preimage*:
fixes *g*::*bool stream* ⇒ *′b*::{*t0-space*}
assumes *g*∈ *borel-measurable* (*F n*)
shows *pseudo-proj-False n* −‘ (*g* −‘ {*g z*}) = *pseudo-proj-False n* −‘ (*pseudo-proj-False n* ‘(*g* −‘ {*g z*}))
using *pseudo-proj-False-preimage*[*of g n borel z*] *set-discriminating-if*[*of g n*] *natural-filtration assms* **by** *simp*

lemma (**in** *infinite-cts-filtration*) *pseudo-proj-preimage′*:
assumes *g*∈ *measurable* (*F n*) *N*
and *set-discriminating n g N*
shows *pseudo-proj-True n* −‘ (*g* −‘ {*g z*}) = *g* −‘ {*g z*}
proof
show *pseudo-proj-True n* −‘ *g* −‘ {*g z*} ⊆ *g* −‘ {*g z*}
proof
fix *w*
assume *w*∈ *pseudo-proj-True n* −‘ *g* −‘ {*g z*}
have *g w* = *g* (*pseudo-proj-True n w*) **using** *assms nat-filtration-not-borel-info natural-filtration*
by (*metis comp-apply*)
also have ... = *g z* **using** ‹*w*∈ *pseudo-proj-True n* −‘ *g* −‘ {*g z*}› **by** *simp*
finally have *g w* = *g z***.**
thus *w*∈ *g* −‘{*g z*} **by** *simp*
qed
show *g* −‘ {*g z*} ⊆ *pseudo-proj-True n* −‘ *g* −‘ {*g z*}
proof
fix *w*
assume *w* ∈ *g* −‘ {*g z*}
have *g* (*pseudo-proj-True n w*) = *g w* **using** *assms nat-filtration-not-borel-info natural-filtration*
by (*metis comp-apply*)
also have ... = *g z* **using** ‹*w*∈ *g* −‘{*g z*}› **by** *simp*
finally have *g* (*pseudo-proj-True n w*) = *g z* **.**
thus *w*∈ *pseudo-proj-True n* −‘ *g* −‘ {*g z*} **by** *simp*
qed
qed

lemma (**in** *infinite-cts-filtration*) *borel-pseudo-proj-preimage′*:
fixes *g*::*bool stream* ⇒ *′b*::{*t0-space*}
assumes *g*∈ *borel-measurable* (*F n*)

shows *pseudo-proj-True n −‘ (g −‘ {g z}) = g −‘ {g z}*
using *assms natural-filtration* **by** (*simp add*: *set-discriminating-if pseudo-proj-preimage′*)

lemma (**in** *infinite-cts-filtration*) *pseudo-proj-False-preimage′*:
assumes *g∈ measurable (F n) N*
and *set-discriminating n g N*
shows *pseudo-proj-False n −‘ (g −‘ {g z}) = g −‘ {g z}*
proof
show *pseudo-proj-False n −‘ g −‘ {g z} ⊆ g −‘ {g z}*
proof
fix *w*
assume *w∈ pseudo-proj-False n −‘ g −‘ {g z}*
have *g w = g (pseudo-proj-False n w)* **using** *assms nat-filtration-not-borel-info′ natural-filtration*
by (*metis comp-apply*)
also have *... = g z* **using** ‹*w∈ pseudo-proj-False n −‘ g −‘ {g z}*› **by** *simp*
finally have *g w = g z*.
thus *w∈ g −‘{g z}* **by** *simp*
qed
show *g −‘ {g z} ⊆ pseudo-proj-False n −‘ g −‘ {g z}*
proof
fix *w*
assume *w ∈ g −‘ {g z}*
have *g (pseudo-proj-False n w) = g w* **using** *assms nat-filtration-not-borel-info′ natural-filtration*
by (*metis comp-apply*)
also have *... = g z* **using** ‹*w∈ g −‘{g z}*› **by** *simp*
finally have *g (pseudo-proj-False n w) = g z* .
thus *w∈ pseudo-proj-False n −‘ g −‘ {g z}* **by** *simp*
qed
qed

lemma (**in** *infinite-cts-filtration*) *borel-pseudo-proj-False-preimage′*:
fixes *g*::*bool stream ⇒ ′b*::{*t0-space*}
assumes *g∈ borel-measurable (F n)*
shows *pseudo-proj-False n −‘ (g −‘ {g z}) = g −‘ {g z}*
using *assms natural-filtration* **by** (*simp add*: *set-discriminating-if pseudo-proj-False-preimage′*)

5.3.6 Integrals and conditional expectations on the natural filtration

lemma (**in** *infinite-cts-filtration*) *cst-integral*:
fixes *f*::*bool stream⇒real*
assumes *f ∈ borel-measurable (F 0)*
and *f (sconst True) = c*
shows *has-bochner-integral M f c*
proof −

have *space M = space (F 0)* **using** *filtration* **by** (*simp add*: *filtration-def subalgebra-def*)
have *f∈ borel-measurable M*
using *assms(1) nat-filtration-borel-measurable-integrable natural-filtration* **by** *blast*
have ∃ *d*. ∀ *x∈ space (F 0)*. *f x = d*
proof (*rule triv-measurable-cst*)
show *space (F 0) = space M* **using** ‹*space M = space (F 0)*› **..**
show *sets (F 0) = {{}, space M}* **using** *info-disc-filtr*
by (*simp add*: *init-triv-filt-def bot-nat-def*)
show *f ∈ borel-measurable (F 0)* **using** *assms* **by** *simp*
show *space M ≠ {}* **by** (*simp add*:*not-empty*)
qed
from *this* **obtain** *d* **where** ∀ *x∈ space (F 0)*. *f x = d* **by** *auto*
hence ∀ *x∈ space M*. *f x = d* **using** ‹*space M = space (F 0)*› **by** *simp*
hence *f* (*sconst True*) = *d* **using** *bernoulli-stream-space bernoulli* **by** *simp*
hence *c = d* **using** *assms* **by** *simp*
hence ∀ *x∈ space M*. *f x = c* **using** ‹∀ *x∈ space M*. *f x = d*› ‹*c = d*› **by** *simp*
have *f∈ borel-measurable M*
using *assms(1) nat-filtration-borel-measurable-integrable natural-filtration* **by** *blast*
have *integral*^{N} *M f = integral*^{N} *M* (λ*w*. *c*)
proof (*rule nn-integral-cong*)
fix *x*
assume *x∈ space M*
thus *ennreal* (*f x*) = *ennreal c* **using** ‹∀ *x∈ space M*. *f x = d*› ‹*c = d*› **by** *auto*
qed
also have ... = *integral*^{N} *M* (λ*w*. *c* ∗ (*indicator* (*space M*)) *w*)
by (*simp add*: *nn-integral-cong*)
also have ... = *ennreal c* ∗ *emeasure M* (*space M*) **using** *nn-integral-cmult-indicator*[*of space M M c*]
by (*simp add*: *nn-integral-cong*)
also have ... = *ennreal c* **by** (*simp add*: *emeasure-space-1*)
finally have *integral*^{N} *M f = ennreal c* **.**
hence *integral*^{N} *M* (λ*x*. − *f x*) = *ennreal* (−*c*)
by (*simp add*: ‹∀ *x∈space M*. *f x = d*› ‹*c = d*› *emeasure-space-1 nn-integral-cong*)
show *has-bochner-integral M f c*
proof (*cases 0 ≤ c*)
case *True*
hence *AE x in M*. *0 ≤ f x* **using** ‹∀ *x∈ space M*. *f x = c*› **by** *simp*
thus *?thesis* **using** ‹*random-variable borel f*› *True*
‹*integral*^{N} *M f = ennreal c*› **by** (*simp add*: *has-bochner-integral-nn-integral*)
next
case *False*
let *?mf* = λ*w*. − *f w*
have *AE x in M*. *0 ≤ ?mf x* **using** ‹∀ *x∈ space M*. *f x = c*› *False* **by** *simp*
hence *has-bochner-integral M ?mf* (−*c*) **using** ‹*random-variable borel f*› *False*
‹*integral*^{N} *M* (λ*x*. − *f x*) = *ennreal* (−*c*)› **by** (*simp add*: *has-bochner-integral-nn-integral*)
thus *?thesis* **using** *has-bochner-integral-minus* **by** *fastforce*

qed
qed

lemma (**in** *infinite-cts-filtration*) *cst-nn-integral*:
fixes f::*bool stream*⇒*real*
assumes $f \in$ *borel-measurable* $(F\ 0)$
and $\bigwedge w.\ 0 \leq f\ w$
and f (*sconst True*) $= c$
shows $integral^N\ M\ f =$ *ennreal* c **using** *assms cst-integral*
by (*simp add*: *assms*(*1*) *has-bochner-integral-iff nn-integral-eq-integral*)

lemma (**in** *infinite-cts-filtration*) *suc-measurable*:
fixes f::*bool stream* ⇒ *'b*::{*t0-space*}
assumes $f\in$ *borel-measurable* $(F\ (Suc\ n))$
shows $(\lambda w.\ f\ (c\ \#\#\ w)) \in$ *borel-measurable* $(F\ n)$
proof −
have $(\lambda w.\ f\ (c\ \#\#\ w)) \in$ *borel-measurable* (*nat-filtration n*)
proof (*rule nat-filtration-comp-measurable*)
have $f\in$ *borel-measurable M* **using** *assms*
using *measurable-from-subalg nat-filtration-subalgebra natural-filtration* **by** *blast*
hence $f\in$ *borel-measurable* (*stream-space* (*measure-pmf* (*bernoulli-pmf p*))) **using** *bernoulli* **unfolding** *bernoulli-stream-def* **by** *simp*
have $(\lambda w.\ c\ \#\#\ w) \in$ (*stream-space* (*measure-pmf* (*bernoulli-pmf p*)) $\rightarrow_M$ *stream-space* (*measure-pmf* (*bernoulli-pmf p*)))
proof (*rule measurable-Stream*)
show $(\lambda x.\ c) \in$ *stream-space* (*measure-pmf* (*bernoulli-pmf p*)) $\rightarrow_M$ *measure-pmf* (*bernoulli-pmf p*) **by** *simp*
show $(\lambda x.\ x) \in$ *stream-space* (*measure-pmf* (*bernoulli-pmf p*)) $\rightarrow_M$ *stream-space* (*measure-pmf* (*bernoulli-pmf p*)) **by** *simp*
qed
hence $(\lambda w.\ f\ (c\ \#\#\ w)) \in$ (*stream-space* (*measure-pmf* (*bernoulli-pmf p*)) $\rightarrow_M$ *borel*) **using** ‹$f\in$ *borel-measurable* (*stream-space* (*measure-pmf* (*bernoulli-pmf p*)))›
measurable-comp[*of* $(\lambda w.\ c\ \#\#\ w)$ *stream-space* (*measure-pmf* (*bernoulli-pmf p*)) *stream-space* (*measure-pmf* (*bernoulli-pmf p*)) *f borel*]
by *simp*
thus *random-variable borel* $(\lambda w.\ f\ (c\ \#\#\ w))$ **using** *bernoulli* **unfolding** *bernoulli-stream-def* **by** *simp*
have $\forall\, w.\ f\ (c\ \#\#\ (pseudo\text{-}proj\text{-}True\ n\ w)) = f\ (c\#\#w)$
proof
fix *w*
have $c\#\#\ (pseudo\text{-}proj\text{-}True\ n\ w) = pseudo\text{-}proj\text{-}True\ (Suc\ n)\ (c\#\#w)$ **unfolding** *pseudo-proj-True-def* **by** *simp*
hence $f\ (c\ \#\#\ (pseudo\text{-}proj\text{-}True\ n\ w)) = f\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)\ (c\#\#w))$ **by** *simp*
also have ... $= f\ (c\#\#w)$ **using** *assms nat-filtration-info*[*of f Suc n*] *natural-filtration*
by (*metis comp-apply*)

finally show $f\ (c\ \#\#\ (pseudo\text{-}proj\text{-}True\ n\ w)) = f\ (c\#\#w)$ **.**
qed
thus $(\lambda w.\ f\ (c\ \#\#\ w)) \circ pseudo\text{-}proj\text{-}True\ n = (\lambda w.\ f\ (c\ \#\#\ w))$ **by** *auto*
qed
thus $(\lambda w.\ f\ (c\ \#\#\ w)) \in borel\text{-}measurable\ (F\ n)$ **using** *natural-filtration* **by** *simp*
qed

lemma (**in** *infinite-cts-filtration*) *F-n-nn-integral-pos*:
fixes f::*bool stream*⇒*real*
shows $\bigwedge f.\ (\forall x.\ 0 \le f\ x) \Longrightarrow f \in borel\text{-}measurable\ (F\ n) \Longrightarrow integral^N\ M\ f =$
$(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ (emeasure\ M\ ((pseudo\text{-}proj\text{-}True\ n) -` \{w\} \cap space\ M)) *\ ennreal\ (f\ w))$
proof (*induct n*)
case *0*
have *range* (*pseudo-proj-True 0*) = {*sconst True*}
proof
have $\bigwedge w.\ pseudo\text{-}proj\text{-}True\ 0\ w = sconst\ True$
proof −
fix *w*
show *pseudo-proj-True 0 w* = *sconst True* **unfolding** *pseudo-proj-True-def* **by** *simp*
qed
thus *range* (*pseudo-proj-True 0*) ⊆ {*sconst True*} **by** *auto*
show {*sconst True*} ⊆ *range* (*pseudo-proj-True 0*)
using ‹*range* (*pseudo-proj-True 0*) ⊆ {*sconst True*}› *subset-singletonD* **by** *fastforce*
qed
hence (*emeasure M* ((*pseudo-proj-True 0*) −‘{*sconst True*} ∩ *space M*)) = *ennreal 1*
by (*metis Int-absorb1 UNIV-I emeasure-eq-measure image-eqI prob-space subsetI vimage-eq*)
have $(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ 0).\ f\ w) = (\sum\ w \in \{sconst\ True\}.\ f\ w)$ **using** ‹*range* (*pseudo-proj-True 0*) = {*sconst True*}›
sum.cong[*of range* (*pseudo-proj-True n*) {*sconst True*} *f f*] **by** *simp*
also have ... = *f* (*sconst True*) **by** *simp*
finally have $(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ 0).\ f\ w) = f\ (sconst\ True)$ **.**
hence $(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ 0).\ (emeasure\ M\ ((pseudo\text{-}proj\text{-}True\ 0) -` \{w\} \cap space\ M)) * f\ w) = f\ (sconst\ True)$
using ‹(*emeasure M* ((*pseudo-proj-True 0*) −‘{*sconst True*} ∩ *space M*)) = *ennreal 1*›
by (*simp add*: ‹*range* (*pseudo-proj-True 0*) = {*sconst True*}›)
thus $integral^N\ M\ f = (\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ 0).\ (emeasure\ M\ ((pseudo\text{-}proj\text{-}True\ 0) -` \{w\} \cap space\ M)) * f\ w)$
using *0* **by** (*simp add*:*cst-nn-integral*)
next
case (*Suc n*)

define BP **where** $BP = measure\text{-}pmf\ (bernoulli\text{-}pmf\ p)$
have $integral^N\ M\ f = integral^N\ (stream\text{-}space\ BP)\ f$ **using** *bernoulli*
unfolding *bernoulli-stream-def BP-def* **by** *simp*
also have $... = \int^+ x.\ \int^+ X.\ f\ (x\ \#\#\ X)\ \partial stream\text{-}space\ BP\ \partial BP$
proof (*rule prob-space.nn-integral-stream-space*)
show *prob-space BP* **unfolding** *BP-def* **by** (*simp add*: *bernoulli bernoulli-stream-def prob-space.prob-space-stream-space prob-space-measure-pmf*)
have $f\in$ *borel-measurable* (*stream-space BP*) **using** *bernoulli Suc* **unfolding** *bernoulli-stream-def BP-def*
using *measurable-from-subalg nat-filtration-subalgebra natural-filtration* **by** *blast*
thus $(\lambda X.\ ennreal\ (f\ X)) \in borel\text{-}measurable\ (stream\text{-}space\ BP)$ **by** *simp*
qed
also have $... = (\lambda x.\ (\int^+ X.\ f\ (x\ \#\#\ X)\ \partial stream\text{-}space\ BP))\ True * ennreal\ p$ +
$(\lambda x.\ (\int^+ X.\ f\ (x\ \#\#\ X)\ \partial stream\text{-}space\ BP))\ False * ennreal\ (1\ -p)$
using *p-gt-0 p-lt-1* **unfolding** *BP-def* **by** *simp*
also have $... = (\int^+ X.\ f\ (True\ \#\#\ X)\ \partial stream\text{-}space\ BP) * p +$
$(\sum w{\in}range\ (pseudo\text{-}proj\text{-}True\ n).\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n\ -`\ \{w\} \cap space\ M) *\ (f\ (False\ \#\#\ w))) * (1-p)$
proof −
define *ff* **where** $ff = (\lambda w.\ f\ (False\ \#\#\ w))$
have $\bigwedge x.\ 0 \le ff\ x$ **using** *Suc* **unfolding** *ff-def* **by** *simp*
moreover have $ff\in$ *borel-measurable* $(F\ n)$ **using** *Suc* **unfolding** *ff-def* **by** (*simp add:suc-measurable*)
ultimately have $(\int^+ x.\ ennreal\ (ff\ x)\ \partial M) =$
$(\sum w{\in}range\ (pseudo\text{-}proj\text{-}True\ n).\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n\ -`\ \{w\} \cap space\ M) * ennreal\ (ff\ w))$
using *Suc* **by** *simp*
thus *?thesis* **unfolding** *ff-def* **by** (*simp add*: *BP-def bernoulli bernoulli-stream-def*)
qed
also have $... = (\sum w{\in}range\ (pseudo\text{-}proj\text{-}True\ n).\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n\ -`\ \{w\} \cap space\ M) *\ (f\ (True\ \#\#\ w))) * p +$
$(\sum w{\in}range\ (pseudo\text{-}proj\text{-}True\ n).\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n\ -`\ \{w\} \cap space\ M) *\ (f\ (False\ \#\#\ w))) * (1-p)$
proof −
define *ft* **where** $ft = (\lambda w.\ f\ (True\ \#\#\ w))$
have $\bigwedge x.\ 0 \le ft\ x$ **using** *Suc* **unfolding** *ft-def* **by** *simp*
moreover have $ft\in$ *borel-measurable* $(F\ n)$ **using** *Suc* **unfolding** *ft-def* **by** (*simp add:suc-measurable*)
ultimately have $(\int^+ x.\ ennreal\ (ft\ x)\ \partial M) =$
$(\sum w{\in}range\ (pseudo\text{-}proj\text{-}True\ n).\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n\ -`\ \{w\} \cap space\ M) * ennreal\ (ft\ w))$
using *Suc* **by** *simp*
thus *?thesis* **unfolding** *ft-def* **by** (*simp add*: *BP-def bernoulli bernoulli-stream-def*)
qed
also have $... = (\sum w{\in}range\ (pseudo\text{-}proj\text{-}True\ n).\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n\ -`\ \{w\} \cap space\ M) * p *\ (f\ (True\ \#\#\ w))) +$
$(\sum w{\in}range\ (pseudo\text{-}proj\text{-}True\ n).\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n\ -`\ \{w\} \cap$

space M) ∗ (*f* (*False* ## *w*)))∗ (*1*−*p*)
proof −
have (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (*f* (*True* ## *w*))) ∗ *p* =
(∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (*f* (*True* ## *w*)) ∗ *p*)
by (*rule sum-distrib-right*)
also have ... = (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ *p* ∗ (*f* (*True* ## *w*)))
proof (*rule sum.cong*, *simp*)
fix *w*
assume *w*∈ *range* (*pseudo-proj-True n*)
show *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ *ennreal* (*f* (*True* ## *w*)) ∗ *ennreal p* =
emeasure M (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ *ennreal p* ∗ *ennreal* (*f* (*True* ## *w*))
proof −
have *ennreal* (*f* (*True* ## *w*)) ∗ *ennreal p* = *ennreal p* ∗ *ennreal* (*f* (*True* ## *w*)) **by** (*simp add*:*mult.commute*)
hence ⋀*x*. *x* ∗ *ennreal* (*f* (*True* ## *w*)) ∗ *ennreal p* = *x* ∗ *ennreal p* ∗ *ennreal* (*f* (*True* ## *w*))
by (*simp add*: *semiring-normalization-rules*(*16*))
thus *?thesis* **by** *simp*
qed
qed
finally have (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (*f* (*True* ## *w*))) ∗ *p* =
(∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ *p* ∗ (*f* (*True* ## *w*))) .
thus *?thesis* **by** *simp*
qed
also have ... = (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ *p* ∗ (*f* (*True* ## *w*))) +
(∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (*1*−*p*) ∗ (*f* (*False* ## *w*)))
proof −
have (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (*f* (*False* ## *w*))) ∗ (*1*−*p*) =
(∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (*f* (*False* ## *w*)) ∗ (*1*−*p*))
by (*rule sum-distrib-right*)
also have ... = (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (*1*−*p*) ∗ (*f* (*False* ## *w*)))
proof (*rule sum.cong*, *simp*)
fix *w*
assume *w*∈ *range* (*pseudo-proj-True n*)
show *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ *ennreal* (*f* (*False* ## *w*)) ∗ *ennreal* (*1*−*p*) =
emeasure M (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ *ennreal* (*1*−*p*) ∗

ennreal (*f* (*False* ## *w*))
proof −
have *ennreal* (*f* (*False* ## *w*)) ∗ *ennreal* (1−*p*) = *ennreal* (1−*p*) ∗ *ennreal* (*f* (*False* ## *w*)) **by** (*simp add*:*mult.commute*)
hence ⋀*x*. *x* ∗ *ennreal* (*f* (*False* ## *w*)) ∗ *ennreal* (1−*p*) = *x* ∗ *ennreal* (1−*p*) ∗ *ennreal* (*f* (*False* ## *w*))
by (*simp add*: *semiring-normalization-rules*(16))
thus *?thesis* **by** *simp*
qed
qed
finally have (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (*f* (*False* ## *w*))) ∗ (1−*p*) =
(∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (1−*p*) ∗ (*f* (*False* ## *w*))) .
thus *?thesis* **by** *simp*
qed
also have ... = (∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *True* ## *w*}. *emeasure M* (*pseudo-proj-True n* −‘ {*stl y*} ∩ *space M*) ∗ *p* ∗ (*f* (*y*))) +
(∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (1−*p*) ∗ (*f* (*False* ## *w*)))
proof −
have (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ *p* ∗ (*f* (*True* ## *w*))) =
(∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*stl* (*True*##*w*)} ∩ *space M*) ∗ *p* ∗ (*f* (*True* ## *w*))) **by** *simp*
also have ... =
(∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *True* ## *w*}. *emeasure M* (*pseudo-proj-True n* −‘ {*stl y*} ∩ *space M*) ∗ *p* ∗ (*f* (*y*)))
by (*rule reindex-pseudo-proj*)
finally have (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ *p* ∗ (*f* (*True* ## *w*))) =
(∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *True* ## *w*}. *emeasure M* (*pseudo-proj-True n* −‘ {*stl y*} ∩ *space M*) ∗ *p* ∗ (*f* (*y*))) .
thus *?thesis* **by** *simp*
qed
also have ... = (∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *True* ## *w*}. *emeasure M* (*pseudo-proj-True n* −‘ {*stl y*} ∩ *space M*) ∗ *p* ∗ (*f* (*y*))) +
(∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *False* ## *w*}. *emeasure M* (*pseudo-proj-True n* −‘ {*stl y*} ∩ *space M*) ∗ (1−*p*) ∗ (*f* (*y*)))
proof −
have (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ (1−*p*) ∗ (*f* (*False* ## *w*))) =
(∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True n* −‘ {*stl* (*False*##*w*)} ∩ *space M*) ∗ (1−*p*) ∗ (*f* (*False* ## *w*))) **by** *simp*
also have ... =
(∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *False* ## *w*}. *emeasure M* (*pseudo-proj-True n* −‘ {*stl y*} ∩ *space M*) ∗ (1−*p*) ∗ (*f* (*y*)))
by (*rule reindex-pseudo-proj*)
finally have (∑ *w*∈*range* (*pseudo-proj-True n*). *emeasure M* (*pseudo-proj-True*

$n -` \{w\} \cap space\ M) * (1-p) * (f\ (False\ \#\#\ w))) =$
$(\sum y \in \{y.\ \exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = False\ \#\#\ w\}.\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M) * (1-p) * (f\ (y)))$.
thus *?thesis* **by** *simp*
qed
also have ... $= (\sum y \in \{y.\ \exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = True\ \#\#\ w\}.\ (prob\text{-}component\ p\ y\ 0) * emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M) * (f\ (y))) +$
$(\sum y \in \{y.\ \exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = False\ \#\#\ w\}.\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M) * (1-p) * (f\ (y)))$
proof −
have $\forall y \in \{y.\ \exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = True\ \#\#\ w\}.\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M) * p =$
$prob\text{-}component\ p\ y\ 0 * emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M)$
proof
fix *y*
assume $y \in \{y.\ \exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = True\ \#\#\ w\}$
hence $\exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = True\ \#\#\ w$ **by** *simp*
from *this* **obtain** *w* **where** $w \in range\ (pseudo\text{-}proj\text{-}True\ n)$ **and** $y = True\ \#\#\ w$ **by** *auto*
have $emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M) * p = p * emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M)$
by (*simp add:mult.commute*)
also have ... $= prob\text{-}component\ p\ y\ 0 * emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M)$ **using** ‹$y = True\ \#\#\ w$›
unfolding *prob-component-def* **by** *simp*
finally show $emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M) * p = prob\text{-}component\ p\ y\ 0 * emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M)$.
qed
thus *?thesis* **by** *auto*
qed
also have ... $= (\sum y \in \{y.\ \exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = True\ \#\#\ w\}.\ (prob\text{-}component\ p\ y\ 0) * emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M) * (f\ (y))) +$
$(\sum y \in \{y.\ \exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = False\ \#\#\ w\}.\ (prob\text{-}component\ p\ y\ 0) * emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M) * (f\ (y)))$
proof −
have $\forall y \in \{y.\ \exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = False\ \#\#\ w\}.\ emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M) * (1-p) =$
$prob\text{-}component\ p\ y\ 0 * emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -` \{stl\ y\} \cap space\ M)$
proof
fix *y*
assume $y \in \{y.\ \exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = False\ \#\#\ w\}$
hence $\exists w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ y = False\ \#\#\ w$ **by** *simp*
from *this* **obtain** *w* **where** $w \in range\ (pseudo\text{-}proj\text{-}True\ n)$ **and** $y = False\ \#\#\ w$ **by** *auto*

have *emeasure M (pseudo-proj-True n −‘ {stl y} ∩ space M) ∗ (1−p) = (1−p) ∗emeasure M (pseudo-proj-True n −‘ {stl y} ∩ space M)*
by (*simp add:mult.commute*)
also have *... = prob-component p y 0 ∗ emeasure M (pseudo-proj-True n −‘ {stl y} ∩ space M)* **using** ‹*y = False ## w*›
unfolding *prob-component-def* **by** *simp*
finally show *emeasure M (pseudo-proj-True n −‘ {stl y} ∩ space M) ∗ (1−p) =*
prob-component p y 0 ∗ emeasure M (pseudo-proj-True n −‘ {stl y} ∩ space M) **.**
qed
thus *?thesis* **by** *auto*
qed
also have *... = (∑ y∈{y. ∃ w ∈ range (pseudo-proj-True n). y = True ## w}. emeasure M {z ∈ space M. ∃ x∈pseudo-proj-True n −‘ {stl y}. z = True ## x} ∗ (f y)) +*
(∑ y∈{y. ∃ w ∈ range (pseudo-proj-True n). y = False ## w}. (prob-component p y 0) ∗ emeasure M (pseudo-proj-True n −‘ {stl y} ∩ space M) ∗ (f (y)))
proof −
have *(∑ y | ∃ w∈range (pseudo-proj-True n). y = True ## w.*
ennreal (prob-component p y 0) ∗ emeasure M (pseudo-proj-True n −‘ {stl y} ∩ space M) ∗ (f y)) =
(∑ y∈{y. ∃ w ∈ range (pseudo-proj-True n). y = True ## w}. emeasure M {z ∈ space M. ∃ x∈pseudo-proj-True n −‘ {stl y}. z = True ## x} ∗ (f y))
proof (*rule sum.cong*, *simp*)
fix *xx*
assume *xx ∈ {y. ∃ w∈range (pseudo-proj-True n). y = True ## w}*
hence *∃ w∈range (pseudo-proj-True n). xx = True ## w* **by** *simp*
from *this* **obtain** *ww* **where** *ww∈range (pseudo-proj-True n)* **and** *xx = True## ww* **by** *auto*
have *ennreal (prob-component p (True##ww) 0) ∗ emeasure M (pseudo-proj-True n −‘ {ww} ∩ space M) =*
emeasure M {z ∈ space M. ∃ x∈pseudo-proj-True n −‘ {ww}. z = True ## x} **using** ‹*ww∈range (pseudo-proj-True n)*›
by (*rule pseudo-proj-element-prob-pref*[*symmetric*])
thus *ennreal (prob-component p xx 0) ∗ emeasure M (pseudo-proj-True n −‘ {stl xx} ∩ space M) ∗ (f xx) =*
emeasure M {z ∈ space M. ∃ x∈pseudo-proj-True n −‘ {stl xx}. z = True ## x} ∗ (f xx) **using** ‹*xx = True##ww*› **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have *... = (∑ y∈{y. ∃ w ∈ range (pseudo-proj-True n). y = True ## w}. emeasure M {z ∈ space M. ∃ x∈pseudo-proj-True n −‘ {stl y}. z = True ## x} ∗ (f y)) +*
(∑ y∈{y. ∃ w ∈ range (pseudo-proj-True n). y = False ## w}. emeasure M {z ∈ space M. ∃ x∈pseudo-proj-True n −‘ {stl y}. z = False ## x} ∗ (f y))
proof −
have *(∑ y | ∃ w∈range (pseudo-proj-True n). y = False ## w.*

ennreal (*prob-component p y 0*) ∗ *emeasure M* (*pseudo-proj-True n −‘* {*stl y*} ∩ *space M*) ∗ (*f y*)) =
(∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *False ## w*}. *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*stl y*}. *z* = *False ## x*} ∗ (*f y*))
proof (*rule sum.cong*, *simp*)
fix *xx*
assume *xx* ∈ {*y*. ∃ *w*∈*range* (*pseudo-proj-True n*). *y* = *False ## w*}
hence ∃ *w*∈*range* (*pseudo-proj-True n*). *xx* = *False ## w* **by** *simp*
from *this* **obtain** *ww* **where** *ww*∈*range* (*pseudo-proj-True n*) **and** *xx* = *False## ww* **by** *auto*
have *ennreal* (*prob-component p* (*False##ww*) *0*) ∗ *emeasure M* (*pseudo-proj-True n −‘* {*ww*} ∩ *space M*) =
emeasure M {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*ww*}. *z* = *False ## x*} **using** ‹*ww*∈*range* (*pseudo-proj-True n*)›
by (*rule pseudo-proj-element-prob-pref*[*symmetric*])
thus *ennreal* (*prob-component p xx 0*) ∗ *emeasure M* (*pseudo-proj-True n −‘* {*stl xx*} ∩ *space M*) ∗ (*f xx*) =
emeasure M {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*stl xx*}. *z* = *False ## x*} ∗ (*f xx*) **using** ‹*xx* = *False##ww*› **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... = (∑ *w*∈ *range* (*pseudo-proj-True n*). *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*w*}. *z* = *True ## x*} ∗ (*f* (*True##w*))) +
(∑ *w* ∈ *range* (*pseudo-proj-True n*). *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*w*}. *z* = *False ## x*} ∗ (*f* (*False##w*)))
proof −
have ⋀*c*. (∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *c ## w*}. *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*stl y*}. *z* = *c ## x*} ∗ (*f y*)) =
(∑ *w*∈ *range* (*pseudo-proj-True n*). *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*w*}. *z* = *c ## x*} ∗ (*f* (*c##w*)))
proof −
fix *c*
have (∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *c ## w*}. *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*stl y*}. *z* = *c ## x*} ∗ (*f y*)) =
(∑ *w*∈ *range* (*pseudo-proj-True n*). *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*stl* (*c##w*)}. *z* = *c ## x*} ∗ (*f* (*c##w*)))
by (*rule reindex-pseudo-proj*[*symmetric*])
also have ... = (∑ *w*∈ *range* (*pseudo-proj-True n*). *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*w*}. *z* = *c ## x*} ∗ (*f* (*c##w*)))
by *simp*
finally show (∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *c ## w*}. *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*stl y*}. *z* = *c ## x*} ∗ (*f y*)) =
(∑ *w*∈ *range* (*pseudo-proj-True n*). *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n −‘* {*w*}. *z* = *c ## x*} ∗ (*f* (*c##w*))) .
qed
thus *?thesis* **by** *auto*
qed

also have ... = $(\sum w\in \{w.\ w\in range\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)) \wedge w!!0 = True\}.$ *emeasure M* (*pseudo-proj-True* (*Suc n*) −‘ {*w*} ∩ (*space M*)) ∗ (*f w*)) +
($\sum w\in \{w.\ w\in range\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)) \wedge w!!0 = False\}.$ *emeasure M* (*pseudo-proj-True* (*Suc n*) −‘ {*w*} ∩ (*space M*)) ∗ (*f w*))
proof −
have $\bigwedge c.$ ($\sum w\in$ *range* (*pseudo-proj-True n*). *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n* −‘ {*w*}. *z* = *c* ## *x*} ∗ (*f* (*c*##*w*))) =
($\sum w\in \{w.\ w\in range\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)) \wedge w!!0 = c\}.$ *emeasure M* (*pseudo-proj-True* (*Suc n*) −‘ {*w*} ∩ (*space M*)) ∗ (*f w*))
proof −
fix *c*
show ($\sum w\in$ *range* (*pseudo-proj-True n*). *emeasure M* {*z* ∈ *space M*. ∃ *x*∈*pseudo-proj-True n* −‘ {*w*}. *z* = *c* ## *x*} ∗ (*f* (*c*##*w*))) =
($\sum w\in \{w.\ w\in range\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)) \wedge w!!0 = c\}.$ *emeasure M* (*pseudo-proj-True* (*Suc n*) −‘ {*w*} ∩ (*space M*)) ∗ (*f w*))
proof (*rule sum.reindex-cong*)
show *inj-on stl* {*w* ∈ *range* (*pseudo-proj-True* (*Suc n*)). *w* !! *0* = *c*}
proof
fix *x y*
assume *x* ∈ {*w* ∈ *range* (*pseudo-proj-True* (*Suc n*)). *w* !! *0* = *c*}
and *y* ∈ {*w* ∈ *range* (*pseudo-proj-True* (*Suc n*)). *w* !! *0* = *c*}
and *stl x* = *stl y*
have *x*!!*0* = *c* **using** ‹*x* ∈ {*w* ∈ *range* (*pseudo-proj-True* (*Suc n*)). *w* !! *0* = *c*}› **by** *simp*
moreover have *y*!!*0* = *c* **using** ‹*y* ∈ {*w* ∈ *range* (*pseudo-proj-True* (*Suc n*)). *w* !! *0* = *c*}› **by** *simp*
ultimately show *x* = *y* **using** ‹*stl x*= *stl y*›
by (*smt snth.simps*(*1*) *stream-eq-Stream-iff*)

qed
show *range* (*pseudo-proj-True n*) = *stl* ‘ {*w* ∈ *range* (*pseudo-proj-True* (*Suc n*)). *w* !! *0* = *c*}
proof
show *range* (*pseudo-proj-True n*) ⊆ *stl* ‘ {*w* ∈ *range* (*pseudo-proj-True* (*Suc n*)). *w* !! *0* = *c*}
proof
fix *x*
assume *x*∈ *range* (*pseudo-proj-True n*)
hence *pseudo-proj-True n x* = *x* **using** *pseudo-proj-True-proj* **by** *auto*
have *pseudo-proj-True* (*Suc n*) (*c*##*x*) = *c*##*x*
proof −
have *pseudo-proj-True* (*Suc n*) (*c*##*x*) = *c* ## *pseudo-proj-True n x* **using** *pseudo-proj-True-Suc-prefix*[*of n c*##*x*]
by *simp*
also have ... = *c*## *x* **using** ‹*pseudo-proj-True n x* = *x*› **by** *simp*
finally show *?thesis* **.**
qed
hence *c*##*x*∈ *range* (*pseudo-proj-True* (*Suc n*)) **by** (*simp add*: *pseudo-proj-True-img*)

```
          thus x∈ stl‘{w ∈ range (pseudo-proj-True (Suc n)). w !! 0 = c}
          proof −
            have ∃ s. (s ∈ range (pseudo-proj-True (Suc n)) ∧ s !! 0 = c) ∧ stl s
= x
              by (metis (no-types) ‹c ## x ∈ range (pseudo-proj-True (Suc n))›
snth.simps(1) stream.sel(1) stream.sel(2))
           then show ?thesis
             by force
          qed
        qed
         show stl ‘ {w ∈ range (pseudo-proj-True (Suc n)). w !! 0 = c} ⊆ range
(pseudo-proj-True n)
        proof
          fix x
          assume x∈ stl ‘ {w ∈ range (pseudo-proj-True (Suc n)). w !! 0 = c}
          hence ∃ w∈ {w ∈ range (pseudo-proj-True (Suc n)). w !! 0 = c}. x =
stl w by auto
         from this obtain w where w ∈ {w ∈ range (pseudo-proj-True (Suc n)).
w !! 0 = c} and x = stl w by auto
          have w∈ range (pseudo-proj-True (Suc n)) and w!!0 = c using ‹w ∈
{w ∈ range (pseudo-proj-True (Suc n)). w !! 0 = c}›
            by auto
          have c##x = w using ‹x = stl w› ‹w!!0 = c› by force
       also have ... = pseudo-proj-True (Suc n) w using ‹w∈ range (pseudo-proj-True
(Suc n))›
            using pseudo-proj-True-proj by auto
          also have ... = c ## pseudo-proj-True n x using ‹x = stl w› ‹w!!0 =
c› by (simp add:pseudo-proj-True-Suc-prefix)
          finally have c##x = c## pseudo-proj-True n x .
          hence x = pseudo-proj-True n x by simp
          thus x∈ range (pseudo-proj-True n) by auto
        qed
      qed
      show ⋀x. x ∈ {w ∈ range (pseudo-proj-True (Suc n)). w !! 0 = c} ⟹
       emeasure M {z ∈ space M. ∃ x∈pseudo-proj-True n −‘ {stl x}. z = c ##
x} ∗ ennreal (f (c ## stl x)) =
       emeasure M (pseudo-proj-True (Suc n) −‘ {x} ∩ space M) ∗ ennreal (f x)
      proof −
        fix w
        assume w ∈ {w ∈ range (pseudo-proj-True (Suc n)). w !! 0 = c}
        hence w ∈ range (pseudo-proj-True (Suc n)) and w !! 0 = c by auto
        have {z ∈ space M. ∃ x∈pseudo-proj-True n −‘ {stl w}. z = c ## x} =
(pseudo-proj-True (Suc n) −‘ {w} ∩ space M)
        proof
          show {z ∈ space M. ∃ x∈pseudo-proj-True n −‘ {stl w}. z = c ## x}
⊆ pseudo-proj-True (Suc n) −‘ {w} ∩ space M
          proof
            fix z
            assume z ∈ {z ∈ space M. ∃ x∈pseudo-proj-True n −‘ {stl w}. z = c
```

```
## x}
          hence ∃x∈pseudo-proj-True n −‘ {stl w}. z = c ## x and z∈ space
M by auto
           from ‹∃x∈pseudo-proj-True n −‘ {stl w}. z = c ## x› obtain x
where x∈pseudo-proj-True n −‘ {stl w}
            and z = c##x by auto
          have pseudo-proj-True (Suc n) z = c ## pseudo-proj-True n x using
‹z = c##x›
            by (simp add:pseudo-proj-True-Suc-prefix)
           also have ... = c## stl w using ‹x∈pseudo-proj-True n −‘ {stl w}›
by simp
          also have ... = w using ‹w !! 0 = c› by force
          finally have pseudo-proj-True (Suc n) z = w .
          thus z∈ pseudo-proj-True (Suc n) −‘ {w} ∩ space M using ‹z∈ space
M› by auto
        qed
         show pseudo-proj-True (Suc n) −‘ {w} ∩ space M ⊆ {z ∈ space M.
∃x∈pseudo-proj-True n −‘ {stl w}. z = c ## x}
        proof
         fix z
         assume z∈ pseudo-proj-True (Suc n) −‘ {w} ∩ space M
         hence z∈ space M and pseudo-proj-True (Suc n) z = w by auto
         hence stl w = stl (pseudo-proj-True (Suc n) z) by simp
       also have ... = pseudo-proj-True n (stl z) by (simp add: pseudo-proj-True-Suc-prefix)
         finally have stl w = pseudo-proj-True n (stl z) .
         hence stl z ∈ pseudo-proj-True n −‘ {stl w} by simp
       have z!!0 ## pseudo-proj-True n (stl z) = w using pseudo-proj-True-Suc-prefix
           ‹pseudo-proj-True (Suc n) z = w› by simp
         also have ... = c## (stl w) using ‹w!!0 = c› by force
         finally have z!!0 ## pseudo-proj-True n (stl z) = c## (stl w) .
         hence z!!0 = c by simp
         hence z =c## (stl z) by force
         thus z∈ {z ∈ space M. ∃x∈pseudo-proj-True n −‘ {stl w}. z = c ##
x} using ‹z∈ space M›
           ‹stl z ∈ pseudo-proj-True n −‘ {stl w}› by auto
        qed
       qed
        hence emeasure M {z ∈ space M. ∃x∈pseudo-proj-True n −‘ {stl w}. z
= c ## x} * ennreal (f (c ## stl w)) =
         emeasure M (pseudo-proj-True (Suc n) −‘ {w} ∩ space M) * ennreal (f
(c ## stl w)) by simp
        also have ... = emeasure M (pseudo-proj-True (Suc n) −‘ {w} ∩ space
M) * ennreal (f w) using ‹w!!0 = c› by force
       finally show emeasure M {z ∈ space M. ∃x∈pseudo-proj-True n −‘ {stl
w}. z = c ## x} * ennreal (f (c ## stl w)) =
         emeasure M (pseudo-proj-True (Suc n) −‘ {w} ∩ space M) * ennreal (f
w) .
      qed
    qed
```

qed
thus *?thesis* **by** *simp*
qed
also have ... = ($\sum$ w∈ {w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *True*}
∪
{w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *False*}.
emeasure M (*pseudo-proj-True* (*Suc n*) −‘ {w} ∩ *space M*) ∗ *ennreal* (*f w*))
proof (*rule sum.union-disjoint*[*symmetric*])
show *finite* {w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *True*} **by** (*simp add*: *pseudo-proj-True-finite-image*)
show *finite* {w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *False*} **by** (*simp add*: *pseudo-proj-True-finite-image*)
show {w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *True*} ∩ {w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *False*} = {}
by *auto*
qed
also have ... = ($\sum$ w∈ *range* (*pseudo-proj-True* (*Suc n*)).*emeasure M* (*pseudo-proj-True* (*Suc n*) −‘ {w} ∩ *space M*) ∗ *ennreal* (*f w*))
proof (*rule sum.cong*)
show {w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *True*} ∪ {w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *False*} =
range (*pseudo-proj-True* (*Suc n*))
proof
show {w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *True*} ∪ {w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *False*}
⊆ *range* (*pseudo-proj-True* (*Suc n*)) **by** *auto*
show *range* (*pseudo-proj-True* (*Suc n*))
⊆ {w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *True*} ∪
{w ∈ *range* (*pseudo-proj-True* (*Suc n*)). w !! 0 = *False*}
by (*simp add*: *subsetI*)
qed
qed *simp*
finally show $integral^N$ *M f* =
($\sum$ w∈ *range* (*pseudo-proj-True* (*Suc n*)). *emeasure M* (*pseudo-proj-True* (*Suc n*) −‘ {w} ∩ *space M*) ∗ *ennreal* (*f w*)) **.**
qed

lemma (**in** *infinite-cts-filtration*) *F-n-integral-pos*:
fixes *f*::*bool stream*⇒*real*
assumes *f*∈ *borel-measurable* (*F n*)
and ∀ *w*. *0* ≤ *f w*
shows *has-bochner-integral M f*
($\sum$ w∈ *range* (*pseudo-proj-True n*). (*measure M* ((*pseudo-proj-True n*) −‘{w} ∩ *space M*)) ∗ (*f w*))
proof −
have $integral^N$ *M f* = ($\sum$ w∈ *range* (*pseudo-proj-True n*). (*emeasure M* ((*pseudo-proj-True n*) −‘{w} ∩ *space M*)) ∗ (*f w*))
using *assms* **by** (*simp add*: *F-n-nn-integral-pos*)

have $integral^L\ M\ f = enn2real\ (integral^N\ M\ f)$
proof (*rule integral-eq-nn-integral*)
show *AE x in M. 0≤ f x* **using** *assms* **by** *simp*
show *random-variable borel f* **using** *assms*
using *measurable-from-subalg nat-filtration-subalgebra natural-filtration* **by** *blast*
qed
also have ... = *enn2real* ($\sum$ *w∈ range* (*pseudo-proj-True n*). (*emeasure M* ((*pseudo-proj-True n*) −‘{*w*} ∩ *space M*)) ∗ (*f w*))
using *assms* **by** (*simp add*: *F-n-nn-integral-pos*)
also have ... = ($\sum$ *w∈ range* (*pseudo-proj-True n*). *enn2real* ((*emeasure M* ((*pseudo-proj-True n*) −‘{*w*} ∩ *space M*)) ∗ (*f w*)))
proof (*rule enn2real-sum*)
show *finite* (*range* (*pseudo-proj-True n*)) **by** (*simp add*: *pseudo-proj-True-finite-image*)
show $\bigwedge w.\ w \in range\ (pseudo\text{-}proj\text{-}True\ n) \Longrightarrow emeasure\ M\ (pseudo\text{-}proj\text{-}True\ n -‘ \{w\} \cap space\ M) * ennreal\ (f\ w) < \top$
proof −
fix *w*
assume *w∈ range* (*pseudo-proj-True n*)
show *emeasure M* (*pseudo-proj-True n* −‘ {*w*} ∩ *space M*) ∗ *ennreal* (*f w*) < ⊤
by (*simp add*: *emeasure-eq-measure ennreal-mult-less-top*)
qed
qed
also have ... = ($\sum$ *w∈ range* (*pseudo-proj-True n*). ((*measure M* ((*pseudo-proj-True n*) −‘{*w*} ∩ *space M*)) ∗ (*f w*)))
by (*simp add*: *Sigma-Algebra.measure-def assms*(*2*) *enn2real-mult*)
finally have $integral^L\ M\ f$ =($\sum$ *w∈ range* (*pseudo-proj-True n*). ((*measure M* ((*pseudo-proj-True n*) −‘{*w*} ∩ *space M*)) ∗ (*f w*))) **.**
moreover have *integrable M f*
proof (*rule integrableI-nn-integral-finite*)
show *random-variable borel f* **using** *assms*
using *measurable-from-subalg nat-filtration-subalgebra natural-filtration* **by** *blast*
show *AE x in M. 0 ≤ f x* **using** *assms* **by** *simp*
have ($\int^+ x.\ ennreal\ (f\ x)\ \partial M$) = ($\sum$ *w∈ range* (*pseudo-proj-True n*). (*emeasure M* ((*pseudo-proj-True n*) −‘{*w*} ∩ *space M*)) ∗ (*f w*))
using *assms* **by** (*simp add*: *F-n-nn-integral-pos*)
also have ... = ($\sum$ *w∈ range* (*pseudo-proj-True n*). *ennreal* (*measure M* ((*pseudo-proj-True n*) −‘{*w*} ∩ *space M*) ∗ (*f w*)))
proof (*rule sum.cong*, *simp*)
fix *x*
assume *x∈ range* (*pseudo-proj-True n*)
thus *emeasure M* (*pseudo-proj-True n* −‘ {*x*} ∩ *space M*) ∗ *ennreal* (*f x*) = *ennreal* (*prob* (*pseudo-proj-True n* −‘ {*x*} ∩ *space M*) ∗ *f x*)
using *assms*(*2*) *emeasure-eq-measure ennreal-mult''* **by** *auto*
qed
also have ... = *ennreal* ($\sum$ *w∈ range* (*pseudo-proj-True n*). (*measure M* ((*pseudo-proj-True n*) −‘{*w*} ∩ *space M*) ∗ (*f w*)))

proof (*rule ennreal-sum*)
show *finite* (*range* (*pseudo-proj-True n*)) **by** (*simp add*: *pseudo-proj-True-finite-image*)
show $\bigwedge w.\ 0 \leq prob\ (pseudo\text{-}proj\text{-}True\ n\ -`\ \{w\} \cap space\ M) * f\ w$
using *assms*(*2*) *measure-nonneg zero-le-mult-iff* **by** *blast*
qed
finally show $(\int^+ x.\ ennreal\ (f\ x)\ \partial M) =$
ennreal $(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ (measure\ M\ ((pseudo\text{-}proj\text{-}True\ n)\ -`\{w\} \cap space\ M) *\ \ (f\ w)))$ **.**
qed
ultimately show *?thesis* **using** *has-bochner-integral-iff* **by** *blast*
qed

lemma (**in** *infinite-cts-filtration*) *F-n-integral*:
fixes f::*bool stream*⇒*real*
assumes $f \in$ *borel-measurable* $(F\ n)$
shows *has-bochner-integral M f*
$(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ (measure\ M\ ((pseudo\text{-}proj\text{-}True\ n)\ -`\{w\} \cap space\ M)) * (f\ w))$
proof −
define *fpos* **where** $fpos = (\lambda w.\ max\ 0\ (f\ w))$
define *fneg* **where** $fneg = (\lambda w.\ max\ 0\ (-f\ w))$
have $\forall\, w.\ 0 \leq fpos\ w$ **unfolding** *fpos-def* **by** *simp*
have $\forall\, w.\ 0 \leq fneg\ w$ **unfolding** *fneg-def* **by** *simp*
have *fpos* ∈ *borel-measurable* $(F\ n)$ **using** *assms* **unfolding** *fpos-def* **by** *simp*
have *fneg* ∈ *borel-measurable* $(F\ n)$ **using** *assms* **unfolding** *fneg-def* **by** *simp*
have *has-bochner-integral M fpos*
$(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ (measure\ M\ ((pseudo\text{-}proj\text{-}True\ n)\ -`\{w\} \cap space\ M)) *\ \ (fpos\ w))$
using ‹*fpos*∈ *borel-measurable* $(F\ n)$› ‹$\forall\, w.\ 0 \leq fpos\ w$› **by** (*simp add*: *F-n-integral-pos*)
moreover have *has-bochner-integral M fneg*
$(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ (measure\ M\ ((pseudo\text{-}proj\text{-}True\ n)\ -`\{w\} \cap space\ M)) *\ \ (fneg\ w))$
using ‹*fneg*∈ *borel-measurable* $(F\ n)$› ‹$\forall\, w.\ 0 \leq fneg\ w$› **by** (*simp add*: *F-n-integral-pos*)
ultimately have *posd*: *has-bochner-integral M* $(\lambda w.\ fpos\ w - fneg\ w)$
$((\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ (measure\ M\ ((pseudo\text{-}proj\text{-}True\ n)\ -`\{w\} \cap space\ M)) *\ \ (fpos\ w)) -$
$(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ (measure\ M\ ((pseudo\text{-}proj\text{-}True\ n)\ -`\{w\} \cap space\ M)) *\ \ (fneg\ w)))$
by (*simp add*:*has-bochner-integral-diff*)
have $((\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ (measure\ M\ ((pseudo\text{-}proj\text{-}True\ n)\ -`\{w\} \cap space\ M)) *\ \ (fpos\ w)) -$
$(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ (measure\ M\ ((pseudo\text{-}proj\text{-}True\ n)\ -`\{w\} \cap space\ M)) *\ \ (fneg\ w))) =$
$(\sum\ w \in range\ (pseudo\text{-}proj\text{-}True\ n).\ (measure\ M\ ((pseudo\text{-}proj\text{-}True\ n)\ -`\{w\} \cap space\ M) * fpos\ w\ -$
$(measure\ M\ ((pseudo\text{-}proj\text{-}True\ n)\ -`\{w\} \cap space\ M)) * fneg\ w))$
by (*rule sum-subtractf*[*symmetric*])
also have ... =

(∑ w∈ range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) -‘{w} ∩ space M) * (fpos w - fneg w)))
proof (rule sum.cong, simp)
fix x
assume x∈ range (pseudo-proj-True n)
show prob (pseudo-proj-True n -‘ {x} ∩ space M) * fpos x - prob (pseudo-proj-True n -‘ {x} ∩ space M) * fneg x =
prob (pseudo-proj-True n -‘ {x} ∩ space M) * (fpos x - fneg x)
by (rule right-diff-distrib[symmetric])
qed
also have ... =
(∑ w∈ range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) -‘{w} ∩ space M) * f w))
proof (rule sum.cong, simp)
fix x
assume x∈ range (pseudo-proj-True n)
show prob (pseudo-proj-True n -‘ {x} ∩ space M) * (fpos x - fneg x) = prob (pseudo-proj-True n -‘ {x} ∩ space M) * f x
unfolding fpos-def fneg-def by auto
qed
finally have ((∑ w∈ range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) -‘{w} ∩ space M)) * (fpos w)) -
(∑ w∈ range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) -‘{w} ∩ space M)) * (fneg w))) =
(∑ w∈ range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) -‘{w} ∩ space M) * f w)) .
hence has-bochner-integral M (λw. fpos w - fneg w) (∑ w∈ range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) -‘{w} ∩ space M) * f w))
using posd by simp
moreover have ⋀w. fpos w - fneg w = f w unfolding fpos-def fneg-def by auto
ultimately show ?thesis using has-bochner-integral-diff by simp
qed

lemma (in infinite-cts-filtration) F-n-integral-prob-comp:
fixes f::bool stream⇒real
assumes f∈ borel-measurable (F n)
shows has-bochner-integral M f
(∑ w∈ range (pseudo-proj-True n). (prod (prob-component p w) {0..<n}) * (f w))
proof -
have ∀ w∈ range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) -‘{w} ∩ space M)) * f w =
(prod (prob-component p w) {0..<n}) * (f w)
proof
fix w
assume w∈ range (pseudo-proj-True n)
thus prob (pseudo-proj-True n -‘ {w} ∩ space M) * f w = prod (prob-component p w) {0..<n} * f w

```
    using bernoulli-stream-pseudo-prob bernoulli p-lt-1 p-gt-0 by simp
  qed
  thus ?thesis using F-n-integral assms by (metis (no-types, lifting) sum.cong)
qed

lemma (in infinite-cts-filtration) expect-prob-comp:
fixes f::bool stream⇒real
  assumes f∈ borel-measurable (F n)
  shows expectation f =
    (∑ w∈ range (pseudo-proj-True n). (prod (prob-component p w) {0..<n}) * (f
w))
  using assms F-n-integral-prob-comp has-bochner-integral-iff by blast

lemma sum-union-disjoint':
  assumes finite A
    and finite B
    and A ∩ B = {}
    and A ∪ B = C
  shows sum g C = sum g A + sum g B
  using sum.union-disjoint[OF assms(1−3)] and assms(4) by auto

lemma (in infinite-cts-filtration) borel-Suc-expectation:
  fixes f::bool stream⇒ real
  assumes f∈ borel-measurable (F (Suc n))
  and g∈ measurable (F n) N
  and set-discriminating n g N
  and g −‘ {g z} ∈ sets (F n)
  and ∀ y z. (g y = g z ∧ snth y n = snth z n) ⟶ f y = f z
  shows expectation (λx. f x * indicator (g −‘ {g z}) x) =
    prob (g −‘ {g z}) * (p * f (pseudo-proj-True n z) +
     (1 −p) * f (pseudo-proj-False n z))
proof −
  define expind where expind = (λx. f x * indicator (g −‘ {g z}) x)
  have expind∈ borel-measurable (F (Suc n)) unfolding expind-def
  proof (rule borel-measurable-times, (simp add:assms(1,2)))
    show indicator (g −‘ {g z}) ∈ borel-measurable (F (Suc n))
    proof (rule borel-measurable-indicator)
      have g −‘ {g z} ∈ sets (nat-filtration n)
        using assms nat-filtration-borel-measurable-singleton natural-filtration by
simp
      hence g −‘ {g z} ∈ sets (F n) using natural-filtration by simp
      thus g −‘ {g z} ∈ sets (F (Suc n))
        using nat-filtration-Suc-sets natural-filtration by blast
    qed
  qed
  hence expectation expind =
    (∑ w∈ range (pseudo-proj-True (Suc n)). (measure M ((pseudo-proj-True (Suc
n)) −‘{w} ∩ space M)) * (expind w))
    by (simp add:F-n-integral has-bochner-integral-integral-eq)
```

```
  also have ... = (∑ w∈ range (pseudo-proj-True (Suc n)) ∩ g −‘ {g z}.
    (measure M ((pseudo-proj-True (Suc n)) −‘{w} ∩ space M)) ∗ (expind w)) +
    (∑ w∈ range (pseudo-proj-True (Suc n)) − g −‘ {g z}.
    (measure M ((pseudo-proj-True (Suc n)) −‘{w} ∩ space M)) ∗ (expind w))
   by (simp add: Int-Diff-Un Int-Diff-disjoint assms sum-union-disjoint′ pseudo-proj-True-finite-image)
  also have ... = (∑ w∈ range (pseudo-proj-True (Suc n)) ∩ g −‘ {g z}.
    (measure M ((pseudo-proj-True (Suc n)) −‘{w} ∩ space M)) ∗ (expind w))
  proof −
    have ∀ w∈ range (pseudo-proj-True (Suc n)) − g −‘ {g z}. expind w = 0
    proof
      fix w
      assume w∈ range (pseudo-proj-True (Suc n)) − g −‘ {g z}
      thus expind w = 0 unfolding expind-def by simp
    qed
    thus ?thesis by simp
  qed
  also have ... = (∑ w∈ range (pseudo-proj-True (Suc n)) ∩ g −‘ {g z}.
    (measure M ((pseudo-proj-True (Suc n)) −‘{w} ∩ space M)) ∗ f w)
  proof −
    have ∀ w∈ range (pseudo-proj-True (Suc n)) ∩ g −‘ {g z}. expind w = f w
    proof
      fix w
      assume w∈ range (pseudo-proj-True (Suc n)) ∩ g −‘ {g z}
      hence w∈ g −‘{g z} by simp
      thus expind w = f w unfolding expind-def by simp
    qed
    thus ?thesis by simp
  qed
  also have ... = (∑ w∈ (pseudo-proj-True n) ‘ (g −‘ {g z}) ∪ (pseudo-proj-False
n) ‘ (g −‘ {g z}).
     (measure M ((pseudo-proj-True (Suc n)) −‘{w} ∩ space M)) ∗ f w) using
f-borel-Suc-preimage[of g] assms(1,2, 3) by auto
  also have ... = (∑ w∈ (pseudo-proj-True n) ‘ (g −‘ {g z}).
    (measure M ((pseudo-proj-True (Suc n)) −‘{w} ∩ space M)) ∗ f w) +
    (∑ w∈ (pseudo-proj-False n) ‘ (g −‘ {g z}).
    (measure M ((pseudo-proj-True (Suc n)) −‘{w} ∩ space M)) ∗ f w)
  proof (rule sum-union-disjoint′)
    show finite (pseudo-proj-True n ‘ g −‘ {g z})
    proof −
    have finite (range (pseudo-proj-True n)) by (simp add: pseudo-proj-True-finite-image)
      moreover have pseudo-proj-True n ‘ g −‘ {g z} ⊆ range (pseudo-proj-True
n)
        by (simp add: image-mono)
      ultimately show ?thesis by (simp add:finite-subset)
    qed
    show finite (pseudo-proj-False n ‘ g −‘ {g z})
    proof −
      have finite (range (pseudo-proj-False n))
      by (metis image-subsetI infinite-super proj-rep-set proj-rep-set-finite pseudo-proj-True-Suc-False-proj
```

rangeI)
moreover have *pseudo-proj-False n ‘ g −‘ {g z} ⊆ range* (*pseudo-proj-False n*)
by (*simp add*: *image-mono*)
ultimately show *?thesis* **by** (*simp add*:*finite-subset*)
qed
show *pseudo-proj-True n ‘ g −‘ {g z} ∩ pseudo-proj-False n ‘ g −‘ {g z} = {}*
proof (*rule ccontr*)
assume *pseudo-proj-True n ‘ g −‘ {g z} ∩ pseudo-proj-False n ‘ g −‘ {g z} ≠ {}*
hence ∃ *y*. *y*∈ *pseudo-proj-True n ‘ g −‘ {g z} ∩ pseudo-proj-False n ‘ g −‘ {g z}* **by** *auto*
from *this* **obtain** *y* **where** *y*∈ *pseudo-proj-True n ‘ g −‘ {g z}* **and** *y*∈ *pseudo-proj-False n ‘ g −‘ {g z}* **by** *auto*
have ∃ *yt*. *yt*∈ *g −‘{g z}* ∧ *y* = *pseudo-proj-True n yt* **using** ‹*y*∈ *pseudo-proj-True n ‘ g −‘ {g z}*› **by** *auto*
from *this* **obtain** *yt* **where** *y* = *pseudo-proj-True n yt* **by** *auto*
have ∃ *yf*. *yf*∈ *g −‘{g z}* ∧ *y* = *pseudo-proj-False n yf* **using** ‹*y*∈ *pseudo-proj-False n ‘ g −‘ {g z}*› **by** *auto*
from *this* **obtain** *yf* **where** *y* = *pseudo-proj-False n yf* **by** *auto*
have *snth y n* = *True* **using** ‹*y* = *pseudo-proj-True n yt*› **unfolding** *pseudo-proj-True-def* **by** *simp*
moreover have *snth y n* = *False* **using** ‹*y* = *pseudo-proj-False n yf*› **unfolding** *pseudo-proj-False-def* **by** *simp*
ultimately show *False* **by** *simp*
qed
qed *simp*
also have ... = (∑ *w*∈*pseudo-proj-True n ‘ g −‘ {g z}*. *prob* (*pseudo-proj-True* (*Suc n*) *−‘* {*w*} ∩ *space M*) ∗ *f* (*pseudo-proj-True n z*)) +
(∑ *w*∈*pseudo-proj-False n ‘ g −‘ {g z}*. *prob* (*pseudo-proj-True* (*Suc n*) *−‘* {*w*} ∩ *space M*) ∗ *f w*)
proof −
define *zt* **where** *zt* = *pseudo-proj-True n z*
have *eqw*: ⋀*w*. *w*∈*pseudo-proj-True n ‘ g −‘ {g z}* ⟹ (*g w* = *g zt* ∧ *snth w n* = *snth zt n*)
proof
fix *w*
assume *w*∈ *pseudo-proj-True n ‘ g −‘ {g z}*
hence ∃ *y*. *w* = *pseudo-proj-True n y* ∧ *g y* = *g z* **by** *auto*
from *this* **obtain** *yt* **where** *w* = *pseudo-proj-True n yt* **and** *g yt* = *g z* **by** *auto*
have *g w*= *g yt* **using** ‹*w* = *pseudo-proj-True n yt*› *nat-filtration-not-borel-info*[*of g*] *natural-filtration*
assms **by** (*metis comp-apply*)
also have ... = *g zt* **using** *assms* **using** *nat-filtration-not-borel-info*[*of g*] *natural-filtration* ‹*g yt* = *g z*›
unfolding *zt-def* **by** (*metis comp-apply*)
finally show *g w* = *g zt* **.**
show *w* !! *n* = *zt* !! *n* **using** ‹*w* = *pseudo-proj-True n yt*› **unfolding** *zt-def*

pseudo-proj-True-def **by** *simp*
qed
hence $\bigwedge w.\ w \in pseudo\text{-}proj\text{-}True\ n\ `\ g\ -`\ \{g\ z\} \Longrightarrow f\ w = f\ zt$
proof
fix *w*
assume $w \in pseudo\text{-}proj\text{-}True\ n\ `\ g\ -`\ \{g\ z\}$
hence $g\ w = g\ zt \wedge snth\ w\ n = snth\ zt\ n$ **using** *eqw* [*of w*] **by** *simp*
thus $f\ w = f\ zt$ **using** *assms*(5) **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
also have ... = $(\sum w \in pseudo\text{-}proj\text{-}True\ n\ `\ g\ -`\ \{g\ z\}.\ prob\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)\ -`\ \{w\} \cap space\ M) * f\ (pseudo\text{-}proj\text{-}True\ n\ z)) +$
$(\sum w \in pseudo\text{-}proj\text{-}False\ n\ `\ g\ -`\ \{g\ z\}.\ prob\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)\ -`\ \{w\} \cap space\ M) * f\ (pseudo\text{-}proj\text{-}False\ n\ z))$
proof −
define *zf* **where** $zf = pseudo\text{-}proj\text{-}False\ n\ z$
have *eqw*: $\bigwedge w.\ w \in pseudo\text{-}proj\text{-}False\ n\ `\ g\ -`\ \{g\ z\} \Longrightarrow (g\ w = g\ zf \wedge snth\ w\ n = snth\ zf\ n)$
proof
fix *w*
assume $w \in pseudo\text{-}proj\text{-}False\ n\ `\ g\ -`\ \{g\ z\}$
hence $\exists y.\ w = pseudo\text{-}proj\text{-}False\ n\ y \wedge g\ y = g\ z$ **by** *auto*
from *this* **obtain** *yf* **where** $w = pseudo\text{-}proj\text{-}False\ n\ yf$ **and** $g\ yf = g\ z$ **by** *auto*
have $g\ w = g\ yf$ **using** ‹$w = pseudo\text{-}proj\text{-}False\ n\ yf$› *nat-filtration-not-borel-info'*[*of g*] *natural-filtration*
assms **by** (*metis comp-apply*)
also have ... = $g\ zf$ **using** *assms* **using** *nat-filtration-not-borel-info'*[*of g*] *natural-filtration* ‹$g\ yf = g\ z$›
unfolding *zf-def* **by** (*metis comp-apply*)
finally show $g\ w = g\ zf$ **.**
show $w\ !!\ n = zf\ !!\ n$ **using** ‹$w = pseudo\text{-}proj\text{-}False\ n\ yf$› **unfolding** *zf-def pseudo-proj-False-def* **by** *simp*
qed
hence $\bigwedge w.\ w \in pseudo\text{-}proj\text{-}False\ n\ `\ g\ -`\ \{g\ z\} \Longrightarrow f\ w = f\ zf$
proof
fix *w*
assume $w \in pseudo\text{-}proj\text{-}False\ n\ `\ g\ -`\ \{g\ z\}$
hence $g\ w = g\ zf \wedge snth\ w\ n = snth\ zf\ n$ **using** *eqw* [*of w*] **by** *simp*
thus $f\ w = f\ zf$ **using** *assms* **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
also have ... = $(\sum w \in pseudo\text{-}proj\text{-}True\ n\ `\ g\ -`\ \{g\ z\}.\ prob\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)\ -`\ \{w\} \cap space\ M)) * f\ (pseudo\text{-}proj\text{-}True\ n\ z) +$
$(\sum w \in pseudo\text{-}proj\text{-}False\ n\ `\ g\ -`\ \{g\ z\}.\ prob\ (pseudo\text{-}proj\text{-}True\ (Suc\ n)\ -`\ \{w\} \cap space\ M)) * f\ (pseudo\text{-}proj\text{-}False\ n\ z)$
by (*simp add*: *sum-distrib-right*)

also have ... = $(\sum w{\in}$*pseudo-proj-True n ‘ g −‘ {g z}*$.$ *prob* $(\{x.$ *stake n x = stake n w*$\}) * p) * f$ *(pseudo-proj-True n z)* +
$(\sum w{\in}$*pseudo-proj-False n ‘ g −‘ {g z}*$.$ *prob (pseudo-proj-True (Suc n) −‘ {w}* $\cap$ *space M)) * f (pseudo-proj-False n z)*
proof −
have $\bigwedge w.$ *w∈pseudo-proj-True n ‘ g −‘ {g z}* $\Longrightarrow$ *(prob (pseudo-proj-True (Suc n) −‘ {w}) =*
(prob $(\{x.$ *stake n x = stake n w*$\})) * p)$
proof −
fix *w*
assume *w∈pseudo-proj-True n ‘ g −‘ {g z}*
hence $\exists y.$ *w = pseudo-proj-True n y* $\wedge$ *g y = g z* **by** *auto*
from *this* **obtain** *yt* **where** *w = pseudo-proj-True n yt* **and** *g yt = g z* **by** *auto*
hence *snth w n* **unfolding** *pseudo-proj-True-def* **by** *simp*
have *pseudo-proj-True (Suc n) w = w* **using** ‹*w = pseudo-proj-True n yt*›
by (*simp add*: *pseudo-proj-True-Suc-proj*)
hence *pseudo-proj-True (Suc n) −‘ {w} =* $\{x.$ *stake (Suc n) x = stake (Suc n) w*$\}$ **using** *pseudo-proj-True-preimage-stake*
by *simp*
hence *prob (pseudo-proj-True (Suc n) −‘ {w}) = prob* $\{x.$ *stake n x = stake n w*$\} *$ *prob-component p w n*
using *bernoulli-stream-element-prob-rec′ bernoulli bernoulli-stream-space p-lt-1 p-gt-0* **by** *simp*
also have ... = *prob* $\{x.$ *stake n x = stake n w*$\} * p$ **using** ‹*snth w n*› **unfolding** *prob-component-def* **by** *simp*
finally show *prob (pseudo-proj-True (Suc n) −‘ {w}) = prob* $\{x.$ *stake n x = stake n w*$\} * p$ **.**
qed
thus *?thesis* **using** *bernoulli bernoulli-stream-space* **by** *simp*
qed
also have ... = $(\sum w{\in}$*pseudo-proj-True n ‘ g −‘ {g z}*$.$ *prob* $(\{x.$ *stake n x = stake n w*$\}) * p) * f$ *(pseudo-proj-True n z)* +
$(\sum w{\in}$*pseudo-proj-False n ‘ g −‘ {g z}*$.$ *prob* $\{x.$ *stake n x = stake n w*$\} * (1 - p)) * f$ *(pseudo-proj-False n z)*
proof −
have $\bigwedge w.$ *w∈pseudo-proj-False n ‘ g −‘ {g z}* $\Longrightarrow$ *(prob (pseudo-proj-True (Suc n) −‘ {w}* $\cap$ *space M) =*
(prob $\{x.$ *stake n x = stake n w*$\}) * (1-p))$
proof −
fix *w*
assume *w∈pseudo-proj-False n ‘ g −‘ {g z}*
hence $\exists y.$ *w = pseudo-proj-False n y* $\wedge$ *g y = g z* **by** *auto*
from *this* **obtain** *yt* **where** *w = pseudo-proj-False n yt* **and** *g yt = g z* **by** *auto*
hence $\neg$*snth w n* **unfolding** *pseudo-proj-False-def* **by** *simp*
have *pseudo-proj-True (Suc n) w = w* **using** ‹*w = pseudo-proj-False n yt*›
by (*simp add*: *pseudo-proj-True-Suc-False-proj*)
hence *pseudo-proj-True (Suc n) −‘{w} =* $\{x.$ *stake (Suc n) x = stake (Suc*

n) *w*} **using** *pseudo-proj-True-preimage-stake*
by *simp*
hence *prob* (*pseudo-proj-True* (*Suc n*) −‘{*w*}) = *prob* {*x*. *stake n x* = *stake n w*} ∗ *prob-component p w n*
using *bernoulli-stream-element-prob-rec′ bernoulli bernoulli-stream-space p-lt-1 p-gt-0* **by** *simp*
also have ... = *prob* {*x*. *stake n x* = *stake n w*} ∗ (*1*−*p*) **using** ‹¬*snth w n*› **unfolding** *prob-component-def* **by** *simp*
finally show *prob* (*pseudo-proj-True* (*Suc n*) −‘{*w*} ∩ *space M*) = *prob* {*x*. *stake n x* = *stake n w*} ∗ (*1*−*p*) **using** *bernoulli*
bernoulli-stream-space **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... = (∑ *w*∈*pseudo-proj-True n* ‘ *g* −‘ {*g z*}. *prob* ({*x*. *stake n x* = *stake n w*})) ∗ *p* ∗ *f* (*pseudo-proj-True n z*) +
(∑ *w*∈*pseudo-proj-False n* ‘ *g* −‘ {*g z*}. *prob* {*x*. *stake n x* = *stake n w*}) ∗ (*1* −*p*) ∗ *f* (*pseudo-proj-False n z*)
by (*simp add*:*sum-distrib-right*)
also have ... = *prob* (*g* −‘ {*g z*}) ∗ *p* ∗ *f* (*pseudo-proj-True n z*) +
(∑ *w*∈*pseudo-proj-False n* ‘ *g* −‘ {*g z*}. *prob* {*x*. *stake n x* = *stake n w*}) ∗ (*1* −*p*) ∗ *f* (*pseudo-proj-False n z*)
proof −
have *projset*: ⋀*w*. *w*∈*pseudo-proj-True n* ‘ *g* −‘ {*g z*} ⟹ {*x*. *stake n x* = *stake n w*} ∈ *sets M*
proof −
fix *w*
assume *w*∈ *pseudo-proj-True n* ‘ *g* −‘ {*g z*}
hence ∃ *y*. *w* = *pseudo-proj-True n y* **by** *auto*
from *this* **obtain** *y* **where** *w* = *pseudo-proj-True n y* **by** *auto*
hence *w* = *pseudo-proj-True n w* **by** (*simp add*: *pseudo-proj-True-proj*)
hence *pseudo-proj-True n* −‘{*w*} = {*x*. *stake n x* = *stake n w*} **using** *pseudo-proj-True-preimage-stake* **by** *simp*
moreover have *pseudo-proj-True n* −‘{*w*} ∈ *sets M*
using ‹*w* = *pseudo-proj-True n w*› *bernoulli bernoulli-stream-space pseudo-proj-True-singleton* **by** *auto*
ultimately show {*x*. *stake n x* = *stake n w*} ∈ *sets M* **by** *simp*
qed
have (∑ *w*∈*pseudo-proj-True n* ‘ *g* −‘ {*g z*}. *prob* ({*x*. *stake n x* = *stake n w*})) =
prob (⋃ *w*∈*pseudo-proj-True n* ‘ *g* −‘ {*g z*}. {*x*. *stake n x* = *stake n w*})
proof (*rule finite-measure-finite-Union*[*symmetric*])
show *finite* (*pseudo-proj-True n* ‘ *g* −‘ {*g z*})
by (*meson finite-subset image-mono pseudo-proj-True-finite-image subset-UNIV*)
show (*λi*. {*x*. *stake n x* = *stake n i*}) ‘ *pseudo-proj-True n* ‘ *g* −‘ {*g z*} ⊆ *events* **using** *projset* **by** *auto*
show *disjoint-family-on* (*λi*. {*x*. *stake n x* = *stake n i*}) (*pseudo-proj-True n* ‘ *g* −‘ {*g z*})

unfolding *disjoint-family-on-def*
proof (*intro ballI impI*)
fix *u v*
assume *u* ∈ *pseudo-proj-True n ‘ g −‘ {g z}* **and** *v*∈ *pseudo-proj-True n ‘ g −‘ {g z}* **and** *u* ≠ *v* **note** *uvprops* = *this*
show {*x. stake n x* = *stake n u*} ∩ {*x. stake n x* = *stake n v*} = {}
proof (*rule ccontr*)
assume {*x. stake n x* = *stake n u*} ∩ {*x. stake n x* = *stake n v*} ≠ {}
hence ∃ *uu. uu*∈ {*x. stake n x* = *stake n u*} ∩ {*x. stake n x* = *stake n v*} **by** *auto*
from *this* **obtain** *uu* **where** *uu*∈ {*x. stake n x* = *stake n u*} ∩ {*x. stake n x* = *stake n v*} **by** *auto*
hence *stake n uu* = *stake n u* **and** *stake n uu* = *stake n v* **by** *auto*
moreover have *stake n u* ≠ *stake n v* **by** (*metis uvprops imageE pseudo-proj-True-proj pseudo-proj-True-stake-image*)
ultimately show *False* **by** *simp*
qed
qed
qed
also have ... = *prob* (⋃ *w*∈*pseudo-proj-True n ‘ g −‘ {g z}*. *pseudo-proj-True n −‘*{*w*})
proof −
have ⋀*w*. *w*∈*pseudo-proj-True n ‘ g −‘ {g z}* ⟹ {*x. stake n x* = *stake n w*} = *pseudo-proj-True n −‘*{*w*}
using *pseudo-proj-True-preimage-stake pseudo-proj-True-proj* **by** *force*
hence (⋃ *w*∈*pseudo-proj-True n ‘ g −‘ {g z}*. {*x. stake n x* = *stake n w*}) = (⋃ *w*∈*pseudo-proj-True n ‘ g −‘ {g z}*. *pseudo-proj-True n −‘*{*w*}) **by** *auto*
thus *?thesis* **by** *simp*
qed
also have ... = *prob* (*pseudo-proj-True n −‘*(*pseudo-proj-True n ‘ g −‘ {g z}*)) **by** (*metis vimage-eq-UN*)
also have ... = *prob* (*g −‘ {g z}*) **using** *pseudo-proj-preimage*[*symmetric*, *of g n N z*]
pseudo-proj-preimage'[*of g n*] *assms* **by** *simp*
finally have (∑ *w*∈*pseudo-proj-True n ‘ g −‘ {g z}*. *prob* ({*x. stake n x* = *stake n w*})) = *prob* (*g −‘ {g z}*) **.**
thus *?thesis* **by** *simp*
qed
also have ... = *prob* (*g −‘ {g z}*) ∗ *p* ∗ *f* (*pseudo-proj-True n z*) +
prob (*g −‘*{*g z*}) ∗ (*1* −*p*) ∗ *f* (*pseudo-proj-False n z*)
proof −
have *projset*: ⋀*w*. *w*∈*pseudo-proj-False n ‘ g −‘ {g z}* ⟹ {*x. stake n x* = *stake n w*} ∈ *sets M*
proof −
fix *w*
assume *w*∈ *pseudo-proj-False n ‘ g −‘ {g z}*
hence ∃ *y*. *w* = *pseudo-proj-False n y* **by** *auto*
from *this* **obtain** *y* **where** *w* = *pseudo-proj-False n y* **by** *auto*
hence *w* = *pseudo-proj-False n w* **using** *pseudo-proj-False-def pseudo-proj-False-stake*

by *auto*
hence *pseudo-proj-False n −‘{w} = {x. stake n x = stake n w}* **using** *pseudo-proj-False-preimage-stake* **by** *simp*
moreover have *pseudo-proj-False n −‘{w} ∈ sets M*
using ‹*w = pseudo-proj-False n w*› *bernoulli bernoulli-stream-space pseudo-proj-False-singleton* **by** *auto*
ultimately show *{x. stake n x = stake n w} ∈ sets M* **by** *simp*
qed
have *(∑ w∈pseudo-proj-False n ‘ g −‘ {g z}. prob ({x. stake n x = stake n w})) =*
prob (⋃ w∈pseudo-proj-False n ‘ g −‘ {g z}. {x. stake n x = stake n w})
proof (*rule finite-measure-finite-Union*[*symmetric*])
show *finite* (*pseudo-proj-False n ‘ g −‘ {g z}*)
by (*meson finite-subset image-mono pseudo-proj-False-finite-image subset-UNIV*)
show *(λi. {x. stake n x = stake n i}) ‘ pseudo-proj-False n ‘ g −‘ {g z} ⊆ events* **using** *projset* **by** *auto*
show *disjoint-family-on (λi. {x. stake n x = stake n i}) (pseudo-proj-False n ‘ g −‘ {g z})*
unfolding *disjoint-family-on-def*
proof (*intro ballI impI*)
fix *u v*
assume *u ∈ pseudo-proj-False n ‘ g −‘ {g z}* **and** *v∈ pseudo-proj-False n ‘ g −‘ {g z}* **and** *u ≠ v* **note** *uvprops = this*
show *{x. stake n x = stake n u} ∩ {x. stake n x = stake n v} = {}*
proof (*rule ccontr*)
assume *{x. stake n x = stake n u} ∩ {x. stake n x = stake n v} ≠ {}*
hence *∃ uu. uu∈ {x. stake n x = stake n u} ∩ {x. stake n x = stake n v}* **by** *auto*
from *this* **obtain** *uu* **where** *uu∈ {x. stake n x = stake n u} ∩ {x. stake n x = stake n v}* **by** *auto*
hence *stake n uu = stake n u* **and** *stake n uu = stake n v* **by** *auto*
moreover have *stake n u ≠ stake n v*
using *pseudo-proj-False-def pseudo-proj-False-stake uvprops* **by** *auto*
ultimately show *False* **by** *simp*
qed
qed
qed
also have *... = prob (⋃ w∈pseudo-proj-False n ‘ g −‘ {g z}. pseudo-proj-False n −‘{w})*
proof −
have *⋀w. w∈pseudo-proj-False n ‘ g −‘ {g z} ⟹ {x. stake n x = stake n w} = pseudo-proj-False n −‘{w}*
using *pseudo-proj-False-preimage-stake pseudo-proj-False-def pseudo-proj-False-stake* **by** *force*
hence *(⋃ w∈pseudo-proj-False n ‘ g −‘ {g z}. {x. stake n x = stake n w}) = (⋃ w∈pseudo-proj-False n ‘ g −‘ {g z}. pseudo-proj-False n −‘{w})* **by** *auto*
thus *?thesis* **by** *simp*
qed

also have ... = *prob* (*pseudo-proj-False n −‘*(*pseudo-proj-False n ‘ g −‘* {*g z*})) **by** (*metis vimage-eq-UN*)
also have ... = *prob* (*g −‘* {*g z*}) **using** *pseudo-proj-False-preimage*[*symmetric*, *of g n N z*]
pseudo-proj-False-preimage′[*of g n*] *assms* **by** *simp*
finally have (∑ *w*∈*pseudo-proj-False n ‘ g −‘* {*g z*}. *prob* ({*x*. *stake n x* = *stake n w*})) = *prob* (*g −‘* {*g z*}) **.**
thus *?thesis* **by** *simp*
qed
also have ... = *prob* (*g −‘* {*g z*}) ∗ (*p* ∗ *f* (*pseudo-proj-True n z*) +
(1 −*p*) ∗ *f* (*pseudo-proj-False n z*))
using *distrib-left*[*symmetric*, *of prob* (*g −‘* {*g z*}) *p* ∗ *f* (*pseudo-proj-True n z*) (1 − *p*) ∗ *f* (*pseudo-proj-False n z*)]
by *simp*
finally show *expectation* (λ*x*. *f x* ∗ *indicator* (*g −‘* {*g z*}) *x*) =
prob (*g −‘* {*g z*}) ∗ (*p* ∗ *f* (*pseudo-proj-True n z*) +
(1 −*p*) ∗ *f* (*pseudo-proj-False n z*)) **unfolding** *expind-def* **.**
qed

lemma (**in** *infinite-cts-filtration*) *borel-Suc-expectation-pseudo-proj*:
fixes *f*::*bool stream*⇒ *real*
assumes *f*∈ *borel-measurable* (*F* (*Suc n*))
shows *expectation* (λ*x*. *f x* ∗ *indicator* (*pseudo-proj-True n −‘* {*pseudo-proj-True n z*}) *x*) =
prob (*pseudo-proj-True n −‘* {*pseudo-proj-True n z*}) ∗
(*p* ∗ (*f* (*pseudo-proj-True n z*)) + (1−*p*) ∗ (*f* (*pseudo-proj-False n z*)))
proof (*rule borel-Suc-expectation*)
show *f* ∈ *borel-measurable* (*F* (*Suc n*)) **using** *assms* **by** *simp*
show *pseudo-proj-True n* ∈ *F n* →*M M*
by (*simp add*: *nat-filtration-pseudo-proj-True-measurable natural-filtration*)
show *pseudo-proj-True n −‘* {*pseudo-proj-True n z*} ∈ *sets* (*F n*)
by (*simp add*: *nat-filtration-singleton natural-filtration pseudo-proj-True-proj*)
show ∀ *y z*. (*pseudo-proj-True n y* = *pseudo-proj-True n z* ∧ *snth y n* = *snth z n*) −→ *f y* = *f z*
proof (*intro allI impI conjI*)
fix *y z*
assume *pseudo-proj-True n y* = *pseudo-proj-True n z* ∧ *y* !! *n* = *z* !! *n*
hence *pseudo-proj-True n y* = *pseudo-proj-True n z* **and** *snth y n* = *snth z n* **by** *auto*
hence *pseudo-proj-True* (*Suc n*) *y* = *pseudo-proj-True* (*Suc n*) *z* **unfolding** *pseudo-proj-True-def*
by (*metis* (*full-types*) ‹*pseudo-proj-True n y* = *pseudo-proj-True n z*› *pseudo-proj-True-same-img stake-Suc*)
thus *f y* = *f z* **using** *nat-filtration-info assms natural-filtration* **by** (*metis comp-apply*)
qed
show *set-discriminating n* (*pseudo-proj-True n*) *M* **unfolding** *set-discriminating-def* **using** *pseudo-proj-True-proj* **by** *simp*

qed

lemma (**in** *infinite-cts-filtration*) *f-borel-Suc-expl-cond-expect*:
assumes $f \in$ *borel-measurable* $(F\ (Suc\ n))$
and $g \in$ *measurable* $(F\ n)\ N$
and *set-discriminating* $n\ g\ N$
and $g\ -`\ \{g\ w\} \in sets\ (F\ n)$
and $\forall\, y\ z.\ (g\ y = g\ z \wedge snth\ y\ n = snth\ z\ n) \longrightarrow f\ y = f\ z$
and $0 < p$
and $p < 1$
shows *expl-cond-expect* $M\ g\ f\ w = p * f\ (pseudo\text{-}proj\text{-}True\ n\ w) + (1 - p) * f$
$(pseudo\text{-}proj\text{-}False\ n\ w)$
proof −
have *nz:prob* $(g\ -`\{g\ w\}) \neq 0$
proof −
have *pseudo-proj-True* $n\ -`\{pseudo\text{-}proj\text{-}True\ n\ w\} \subseteq g\ -`\ \{g\ w\}$
proof −
have $\forall f\ n\ m\ s.\ f \notin F\ n \rightarrow_M m \vee \neg$ *set-discriminating* $n\ f\ m \vee pseudo\text{-}proj\text{-}True$
$n\ -`\ f\ -`\ \{f\ s::'a\} = f\ -`\ \{f\ s\}$
by (*meson pseudo-proj-preimage'*)
then show *?thesis* **using** *assms* **by** *blast*
qed
moreover have *prob* $(pseudo\text{-}proj\text{-}True\ n\ -`\{pseudo\text{-}proj\text{-}True\ n\ w\}) > 0$
using *bernoulli-stream-pref-prob-pos*
pseudo-proj-True-preimage-stake bernoulli bernoulli-stream-space emeasure-eq-measure
pseudo-proj-True-proj assms **by** *auto*
moreover have *pseudo-proj-True* $n\ -`\{pseudo\text{-}proj\text{-}True\ n\ w\} \in sets\ M$
using *bernoulli bernoulli-stream-space pseudo-proj-True-proj pseudo-proj-True-singleton*
by *auto*
moreover have $g\ -`\{g\ w\} \in events$ **using** *assms natural-filtration nat-filtration-subalgebra*
unfolding *subalgebra-def* **by** *blast*
ultimately show *?thesis* **using** *measure-increasing increasingD*
proof −
have $g\ -`\ \{g\ w\} \notin events \vee pseudo\text{-}proj\text{-}True\ n\ -`\ \{pseudo\text{-}proj\text{-}True\ n\ w\}$
$\notin events \vee prob\ (pseudo\text{-}proj\text{-}True\ n\ -`\ \{pseudo\text{-}proj\text{-}True\ n\ w\}) \leq prob\ (g\ -`\ \{g$
$w\})$
using ‹*pseudo-proj-True* $n\ -`\ \{pseudo\text{-}proj\text{-}True\ n\ w\} \subseteq g\ -`\ \{g\ w\}$›
increasingD measure-increasing **by** *blast*
then show *?thesis*
using ‹$0 < prob\ (pseudo\text{-}proj\text{-}True\ n\ -`\ \{pseudo\text{-}proj\text{-}True\ n\ w\})$› ‹$g\ -`\ \{g$
$w\} \in events$› ‹$pseudo\text{-}proj\text{-}True\ n\ -`\ \{pseudo\text{-}proj\text{-}True\ n\ w\} \in events$› **by** *linarith*
qed
qed
hence *expl-cond-expect* $M\ g\ f\ w =$
expectation $(\lambda x.\ f\ x * indicator\ (g\ -`\ \{g\ w\} \cap space\ M)\ x)\ /$
$prob\ (g\ -`\ \{g\ w\} \cap space\ M)$ **unfolding** *expl-cond-expect-def img-dce-def*
by *simp*

also have ... = *expectation* ($\lambda x.\ f\ x * indicator\ (g -` \{g\ w\})\ x$) / *prob* ($g -` \{g\ w\}$)
using *bernoulli* **by** (*simp add*:*bernoulli-stream-space*)
also have ... = *prob* ($g -` \{g\ w\}$) * ($p * f$ (*pseudo-proj-True n w*) +
($1 - p$) * f (*pseudo-proj-False n w*)) / *prob* ($g -` \{g\ w\}$)
proof −
have *expectation* ($\lambda x.\ f\ x * indicator\ (g -` \{g\ w\})\ x$) = *prob* ($g -` \{g\ w\}$) * ($p * f$ (*pseudo-proj-True n w*) +
($1 - p$) * f (*pseudo-proj-False n w*))
proof (*rule borel-Suc-expectation*)
show $f \in$ *borel-measurable* (*F* (*Suc n*)) **using** *assms* **by** *simp*
show $g \in F\ n \to_M N$ **using** *assms* **by** *simp*
show *set-discriminating n g N* **using** *assms* **by** *simp*
show $g -` \{g\ w\} \in sets\ (F\ n)$ **using** *assms* **by** *simp*
show $\forall\, y\ z.\ g\ y = g\ z \wedge y\ !!\ n = z\ !!\ n \longrightarrow f\ y = f\ z$ **using** *assms*(*5*) **by** *blast*
qed
thus *?thesis* **by** *simp*
qed
also have ... = $p * f$ (*pseudo-proj-True n w*) + ($1 - p$) * f (*pseudo-proj-False n w*) **using** *nz* **by** *simp*
finally show *?thesis* **.**
qed

lemma (**in** *infinite-cts-filtration*) *f-borel-Suc-real-cond-exp*:
assumes $f \in$ *borel-measurable* (*F* (*Suc n*))
and $g \in$ *measurable* (*F n*) *N*
and *set-discriminating n g N*
and $\forall\, w.\ g -` \{g\ w\} \in sets\ (F\ n)$
and $\forall\, r \in range\ g \cap space\ N.\ \exists\, A \in sets\ N.\ range\ g \cap A = \{r\}$
and $\forall\, y\ z.\ (g\ y = g\ z \wedge snth\ y\ n = snth\ z\ n) \longrightarrow f\ y = f\ z$
and $0 < p$
and $p < 1$
shows *AE w in M. real-cond-exp M* (*fct-gen-subalgebra M N g*) $f\ w = p * f$ (*pseudo-proj-True n w*) + ($1 - p$) * f (*pseudo-proj-False n w*)
proof −
have *AE w in M. real-cond-exp M* (*fct-gen-subalgebra M N g*) *f w = expl-cond-expect M g f w*
proof (*rule charact-cond-exp'*)
show *disc-fct g*
proof −
have *g* = *g* ∘ (*pseudo-proj-True n*) **using** *nat-filtration-not-borel-info*[*of g n*] *assms natural-filtration* **by** *simp*
have *disc-fct* (*pseudo-proj-True n*) **unfolding** *disc-fct-def* **using** *pseudo-proj-True-finite-image*
by (*simp add*: *countable-finite*)
hence *disc-fct* (*g* ∘ (*pseudo-proj-True n*)) **unfolding** *disc-fct-def*
by (*metis countable-image image-comp*)
thus *?thesis* **using** ‹*g* = *g* ∘ (*pseudo-proj-True n*)› **by** *simp*
qed

show *integrable M f* **using** *assms nat-filtration-borel-measurable-integrable natural-filtration* **by** *simp*
show *random-variable N g* **using** *assms filtration-measurable natural-filtration nat-filtration-subalgebra*
using *nat-discrete-filtration* **by** *blast*
show $\forall r{\in}range\ g \cap space\ N.\ \exists A{\in}sets\ N.\ range\ g \cap A = \{r\}$ **using** *assms* **by** *simp*
qed
moreover have $\bigwedge w.$ *expl-cond-expect M g f w* $= p * f$ (*pseudo-proj-True n w*) $+ (1 - p) * f$ (*pseudo-proj-False n w*)
using *assms f-borel-Suc-expl-cond-expect* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *infinite-cts-filtration*) *f-borel-Suc-real-cond-exp-proj*:
assumes $f{\in}$ *borel-measurable* (*F* (*Suc n*))
and $0 < p$
and $p < 1$
shows *AE w in M. real-cond-exp M* (*fct-gen-subalgebra M M* (*pseudo-proj-True n*)) *f w* =
$p * f$ (*pseudo-proj-True n w*) $+ (1 - p) * f$ (*pseudo-proj-False n w*)
proof (*rule f-borel-Suc-real-cond-exp*)
show $f \in$ *borel-measurable* (*F* (*Suc n*)) **using** *assms* **by** *simp*
show *pseudo-proj-True n* $\in$ *F n* $\rightarrow_M$ *M*
by (*simp add*: *nat-filtration-pseudo-proj-True-measurable natural-filtration*)
show $\forall w.$ *pseudo-proj-True n* $-`$ {*pseudo-proj-True n w*} $\in$ *sets* (*F n*)
proof
fix *w*
show *pseudo-proj-True n* $-`$ {*pseudo-proj-True n w*} $\in$ *sets* (*F n*)
by (*simp add*: *nat-filtration-singleton natural-filtration pseudo-proj-True-proj*)
qed
show $\forall r{\in}range$ (*pseudo-proj-True n*) $\cap$ *space M*. $\exists A{\in}events.$ *range* (*pseudo-proj-True n*) $\cap A = \{r\}$
proof (*intro ballI*)
fix *r*
assume $r \in$ *range* (*pseudo-proj-True n*) $\cap$ *space M*
hence $r{\in}$ *range* (*pseudo-proj-True n*) **and** $r{\in}$ *space M* **by** *auto*
hence *pseudo-proj-True n r* $= r$ **using** *pseudo-proj-True-proj* **by** *auto*
hence (*pseudo-proj-True n*) $-`\{r\} \cap$ *space M* $\in$ *sets M* **using** *pseudo-proj-True-singleton bernoulli* **by** *simp*
moreover have *range* (*pseudo-proj-True n*) $\cap$ ((*pseudo-proj-True n*) $-`\{r\} \cap$ *space M*) $= \{r\}$
proof
have $r{\in}$ *range* (*pseudo-proj-True n*) $\cap$ (*pseudo-proj-True n* $-`$ {*r*} $\cap$ *space M*)
using ‹*pseudo-proj-True n r* $= r$› ‹$r \in$ *range* (*pseudo-proj-True n*)› ‹$r \in$ *space M*› **by** *blast*
thus {*r*} $\subseteq$ *range* (*pseudo-proj-True n*) $\cap$ (*pseudo-proj-True n* $-`$ {*r*} $\cap$ *space M*) **by** *auto*

show *range* (*pseudo-proj-True n*) ∩ (*pseudo-proj-True n* −‘ {*r*} ∩ *space M*) ⊆ {*r*}
proof
fix *x*
assume *x* ∈ *range* (*pseudo-proj-True n*) ∩ (*pseudo-proj-True n* −‘ {*r*} ∩ *space M*)
hence *x*∈ *range* (*pseudo-proj-True n*) **and** *x*∈ (*pseudo-proj-True n* −‘ {*r*}) **by** *auto*
have *x* = *pseudo-proj-True n x* **using** ‹*x*∈ *range* (*pseudo-proj-True n*)› *pseudo-proj-True-proj* **by** *auto*
also have ... = *r* **using** ‹*x*∈ (*pseudo-proj-True n* −‘ {*r*})› **by** *simp*
finally have *x* = *r* **.**
thus *x*∈ {*r*} **by** *simp*
qed
qed
ultimately show ∃ *A*∈*events*. *range* (*pseudo-proj-True n*) ∩ *A* = {*r*} **by** *auto*
qed
show ∀ *y z*. *pseudo-proj-True n y* = *pseudo-proj-True n z* ∧ *y* !! *n* = *z* !! *n* ⟶ *f y* = *f z*
proof (*intro allI impI conjI*)
fix *y z*
assume *pseudo-proj-True n y* = *pseudo-proj-True n z* ∧ *y* !! *n* = *z* !! *n*
hence *pseudo-proj-True n y* = *pseudo-proj-True n z* **and** *snth y n* = *snth z n* **by** *auto*
hence *pseudo-proj-True* (*Suc n*) *y* = *pseudo-proj-True* (*Suc n*) *z* **unfolding** *pseudo-proj-True-def*
by (*metis* (*full-types*) ‹*pseudo-proj-True n y* = *pseudo-proj-True n z*› *pseudo-proj-True-same-img stake-Suc*)
thus *f y* = *f z* **using** *nat-filtration-info assms natural-filtration* **by** (*metis comp-apply*)
qed
show *set-discriminating n* (*pseudo-proj-True n*) *M* **unfolding** *set-discriminating-def* **using** *pseudo-proj-True-proj* **by** *simp*
show *0* < *p* **and** *p* < *1* **using** *assms* **by** *auto*
qed

5.4 Images of stochastic processes by prefixes of streams

We define a function that, given a stream of coin tosses and a stochastic process, returns a stream of the values of the stochastic process up to a given time. This function will be used to characterize the smallest filtration that, at any time n, makes each random variable of a given stochastic process measurable up to time n.

5.4.1 Definitions

primrec *smap-stoch-proc* **where**
smap-stoch-proc 0 f k w = []

```
| smap-stoch-proc (Suc n) f k w = (f k w) # (smap-stoch-proc n f (Suc k) w)


lemma smap-stoch-proc-length:
  shows length (smap-stoch-proc n f k w) = n
  by (induction n arbitrary:k) auto


lemma smap-stoch-proc-nth:
  shows Suc p ≤ Suc n ⟹ nth (smap-stoch-proc (Suc n) f k w) p = f (k+p) w
proof (induction n arbitrary:p k)
  case 0
  hence p = 0 by simp
  hence (smap-stoch-proc (Suc 0) f k w) ! p = ((f k w) # (smap-stoch-proc 0 f
(Suc k) w))!0 by simp
  also have ... = f (k+p) w using ‹p=0› by simp
  finally show ?case .
next
  case (Suc n)
  show ?case
  proof (cases ∃ m. p = Suc m)
    case True
    from this obtain m where p = Suc m by auto
    hence smap-stoch-proc (Suc (Suc n)) f k w ! p = (smap-stoch-proc (Suc n) f
(Suc k) w) ! m by simp
   also have ... = f ((Suc k) + m) w using Suc(1)[of m] Suc.prems ‹p = Suc m›
by blast
    also have ... = f (k + (Suc m)) w by simp
    finally show smap-stoch-proc (Suc (Suc n)) f k w ! p = f (k + p) w using ‹p
= Suc m› by simp
  next
    case False
    hence p = 0 using less-Suc-eq-0-disj by blast
    thus smap-stoch-proc (Suc (Suc n)) f k w ! p =  f (k+p) w by simp
  qed
qed


definition proj-stoch-proc where
proj-stoch-proc f n = (λw. shift (smap-stoch-proc n f 0 w) (sconst (f n w)))


lemma proj-stoch-proc-component:
  shows k < n ⟹ (snth (proj-stoch-proc f n w) k) = f k w
    and n ≤ k ⟹ (snth (proj-stoch-proc f n w) k) = f n w
proof −
  show k < n ⟹ proj-stoch-proc f n w !! k = f k w
  proof −
    assume k < n
```

```
    hence ∃ m. n = Suc m using less-imp-Suc-add by blast
    from this obtain m where n = Suc m by auto
     have proj-stoch-proc f n w !! k = (smap-stoch-proc n f 0 w) ! k unfolding
proj-stoch-proc-def
      using ‹k<n› by (simp add: smap-stoch-proc-length)
    also have ... = f k w using ‹n = Suc m› ‹k < n› smap-stoch-proc-nth
      by (metis Suc-leI add.left-neutral)
    finally show ?thesis .
  qed
  show n ≤ k ⟹ (snth (proj-stoch-proc f n w) k) = f n w
  proof −
    assume n ≤ k
    hence proj-stoch-proc f n w !! k = (sconst (f n w)) !! (k − n)
      by (simp add: proj-stoch-proc-def smap-stoch-proc-length)
    also have ... = f n w by simp
    finally show ?thesis .
  qed
qed

lemma proj-stoch-proc-component':
  assumes k ≤ n
  shows f k x = snth (proj-stoch-proc f n x) k
  proof (cases k < n)
    case True
    thus ?thesis using proj-stoch-proc-component[of k n f x] assms by simp
  next
    case False
    hence k = n using assms by simp
    thus ?thesis using proj-stoch-proc-component[of k n f x] assms by simp
  qed

lemma proj-stoch-proc-eq-snth:
  assumes proj-stoch-proc f n x = proj-stoch-proc f n y
and k ≤ n
shows f k x = f k y
proof −
 have f k x = snth (proj-stoch-proc f n x) k using assms proj-stoch-proc-component'[of
k n f] by simp
  also have ... = snth (proj-stoch-proc f n y) k using assms by simp
  also have ... = f k y using assms proj-stoch-proc-component'[of k n f] by simp
  finally show ?thesis .
qed

lemma proj-stoch-measurable-if-adapted:
  assumes filtration M F
  and adapt-stoch-proc F f N
  shows proj-stoch-proc f n ∈ measurable M (stream-space N)
proof (rule measurable-stream-space2)
  fix m
```

```
  show (λx. proj-stoch-proc f n x !! m) ∈ M →M N
  proof (cases m < n)
    case True
   hence ∀ x. proj-stoch-proc f n x !! m = f m x by (simp add: proj-stoch-proc-component)
    hence (λx. proj-stoch-proc f n x !! m) = f m by simp
    thus ?thesis using assms unfolding adapt-stoch-proc-def filtration-def
      by (metis measurable-from-subalg)
  next
    case False
   hence ∀ x. proj-stoch-proc f n x !! m = f n x by (simp add: proj-stoch-proc-component)
    hence (λx. proj-stoch-proc f n x !! m) = f n by simp
    thus ?thesis using assms unfolding adapt-stoch-proc-def filtration-def
      by (metis measurable-from-subalg)
  qed
qed

lemma proj-stoch-adapted-if-adapted:
  assumes filtration M F
  and adapt-stoch-proc F f N
  shows proj-stoch-proc f n ∈ measurable (F n) (stream-space N)
proof (rule measurable-stream-space2)
  fix m
  show (λx. proj-stoch-proc f n x !! m) ∈ measurable (F n) N
  proof (cases m < n)
    case True
   hence ∀ x. proj-stoch-proc f n x !! m = f m x by (simp add: proj-stoch-proc-component)
    hence (λx. proj-stoch-proc f n x !! m) = f m by simp
    thus ?thesis using assms unfolding adapt-stoch-proc-def filtration-def
      by (metis True measurable-from-subalg not-less order.asym)
  next
    case False
   hence ∀ x. proj-stoch-proc f n x !! m = f n x by (simp add: proj-stoch-proc-component)
    hence (λx. proj-stoch-proc f n x !! m) = f n by simp
    thus ?thesis using assms unfolding adapt-stoch-proc-def by metis
  qed
qed

lemma proj-stoch-adapted-if-adapted':
  assumes filtration M F
  and adapt-stoch-proc F f N
shows adapt-stoch-proc F (proj-stoch-proc f) (stream-space N) unfolding adapt-stoch-proc-def
proof
  fix n
  show proj-stoch-proc f n ∈ F n →M stream-space N using assms by (simp add:
proj-stoch-adapted-if-adapted)
qed


lemma (in infinite-cts-filtration) proj-stoch-proj-invariant:
```

fixes X::$nat \Rightarrow bool\ stream \Rightarrow 'b$::{$t0$-$space$}
assumes *borel-adapt-stoch-proc F X*
shows *proj-stoch-proc X n w = proj-stoch-proc X n (pseudo-proj-True n w)*
proof −
have $\bigwedge m$. *snth (proj-stoch-proc X n w) m = snth (proj-stoch-proc X n (pseudo-proj-True n w)) m*
proof −
fix *m*
show *snth (proj-stoch-proc X n w) m = snth (proj-stoch-proc X n (pseudo-proj-True n w)) m*
proof (*cases m < n*)
case *True*
hence *snth (proj-stoch-proc X n w) m = X m w* **by** (*simp add*: *proj-stoch-proc-component*)
also have *... = X m (pseudo-proj-True n w)*
proof (*rule borel-adapt-nat-filtration-info*[*symmetric*], (*simp add*:*assms*))
show *m ≤ n* **using** *True* **by** *linarith*
qed
also have *... = snth (proj-stoch-proc X n (pseudo-proj-True n w)) m* **using** *True*
by (*simp add*: *proj-stoch-proc-component*)
finally show *?thesis* **.**
next
case *False*
hence *snth (proj-stoch-proc X n w) m = X n w* **by** (*simp add*: *proj-stoch-proc-component*)
also have *... = X n (pseudo-proj-True n w)*
by (*rule borel-adapt-nat-filtration-info*[*symmetric*]) (*auto simp add*:*assms*)
also have *... = snth (proj-stoch-proc X n (pseudo-proj-True n w)) m* **using** *False*
by (*simp add*: *proj-stoch-proc-component*)
finally show *?thesis* **.**
qed
qed
thus *proj-stoch-proc X n w = proj-stoch-proc X n (pseudo-proj-True n w)*
using *diff-streams-only-if* **by** *auto*
qed

lemma (**in** *infinite-cts-filtration*) *proj-stoch-set-finite-range*:
fixes X::$nat \Rightarrow bool\ stream \Rightarrow 'b$::{$t0$-$space$}
assumes *borel-adapt-stoch-proc F X*
shows *finite (range (proj-stoch-proc X n))*
proof −
have *finite (range (pseudo-proj-True n))* **using** *pseudo-proj-True-finite-image* **by** *simp*
moreover have *proj-stoch-proc X n = (proj-stoch-proc X n) ∘ (pseudo-proj-True n)*
proof
fix *x*
show *proj-stoch-proc X n x = (proj-stoch-proc X n ∘ pseudo-proj-True n) x*
using *assms proj-stoch-proj-invariant* **by** (*metis comp-apply*)

qed
ultimately show *?thesis*
by (*metis finite-imageI fun.set-map*)
qed

lemma (**in** *infinite-cts-filtration*) *proj-stoch-set-discriminating*:
fixes *X*::*nat* ⇒ *bool stream* ⇒ *'b*::{*t0-space*}
assumes *borel-adapt-stoch-proc F X*
shows *set-discriminating n* (*proj-stoch-proc X n*) *N*
proof −
have ∀ *w*. *proj-stoch-proc X n w* = *proj-stoch-proc X n* (*pseudo-proj-True n w*)
using *assms proj-stoch-proj-invariant* **by** *auto*
thus *?thesis* **unfolding** *set-discriminating-def* **by** *simp*
qed

lemma (**in** *infinite-cts-filtration*) *proj-stoch-preimage*:
assumes *borel-adapt-stoch-proc F X*
shows (*proj-stoch-proc X n*) −‘ {*proj-stoch-proc X n w*} = (⋂ *i*∈ {*m*. *m* ≤ *n*}.
(*X i*) −‘ {*X i w*})
proof
define *psX* **where** *psX* = *proj-stoch-proc X n*
show *proj-stoch-proc X n* −‘ {*proj-stoch-proc X n w*} ⊆ (⋂ *i*∈{*m*. *m* ≤ *n*}. *X i*
−‘ {*X i w*})
proof
fix *x*
assume *x* ∈ *proj-stoch-proc X n* −‘ {*proj-stoch-proc X n w*}
hence *psX x* = *psX w* **unfolding** *psX-def* **using** *assms* **by** *simp*
hence ⋀*i*. *i* ∈{*m*. *m* ≤ *n*} ⟹ *x* ∈ (*X i*) −‘{*X i w*}
proof −
fix *i*
assume *i*∈ {*m*. *m*≤*n*}
hence *i* ≤ *n* **by** *auto*
have *X i x* = *snth* (*psX x*) *i* **unfolding** *psX-def*
by (*metis Suc-le-eq Suc-le-mono* ‹*i* ≤ *n*› *le-Suc-eq nat.simps*(*1*) *proj-stoch-proc-component*(*1*)
proj-stoch-proc-component(*2*))
also have ... = *snth* (*psX w*) *i* **using** ‹*psX x* = *psX w*› **by** *simp*
also have ... = *X i w* **unfolding** *psX-def*
by (*metis Suc-le-eq Suc-le-mono* ‹*i* ≤ *n*› *le-Suc-eq nat.simps*(*1*) *proj-stoch-proc-component*(*1*)
proj-stoch-proc-component(*2*))
finally have *X i x* = *X i w* **.**
thus *x* ∈ (*X i*) −‘{*X i w*} **by** *simp*
qed
thus *x* ∈ (⋂ *i*∈{*m*. *m* ≤ *n*}. (*X i*) −‘ {*X i w*}) **by** *auto*
qed
show (⋂ *i*∈{*m*. *m* ≤ *n*}. (*X i*) −‘ {*X i w*}) ⊆ (*proj-stoch-proc X n*) −‘ {*proj-stoch-proc*
X n w}
proof
fix *x*
assume *x*∈ (⋂ *i*∈{*m*. *m* ≤ *n*}. (*X i*) −‘ {*X i w*})

hence $\bigwedge i.\ i \in \{m.\ m \leq n\} \Longrightarrow x\ \in (X\ i) -` \{X\ i\ w\}$ **by** *simp*
hence $\bigwedge i.\ i \in \{m.\ m \leq n\} \Longrightarrow X\ i\ x = X\ i\ w$ **by** *simp*
hence $\bigwedge i.\ i \leq n \Longrightarrow X\ i\ x = X\ i\ w$ **by** *auto*
hence $psX\ x = psX\ w$ **unfolding** *psX-def*
by (*metis* (*mono-tags*, *opaque-lifting*) *diff-streams-only-if linear not-le order-refl proj-stoch-proc-component*(*1*) *proj-stoch-proc-component*(*2*))
thus $x \in (proj\text{-}stoch\text{-}proc\ X\ n) -` \{proj\text{-}stoch\text{-}proc\ X\ n\ w\}$ **unfolding** *psX-def* **by** *auto*
qed
qed

lemma (**in** *infinite-cts-filtration*) *proj-stoch-singleton-set*:
fixes X::$nat \Rightarrow bool\ stream \Rightarrow ('b$::*t2-space*$)$
assumes *borel-adapt-stoch-proc F X*
shows $(proj\text{-}stoch\text{-}proc\ X\ n) -` \{proj\text{-}stoch\text{-}proc\ X\ n\ w\} \in sets\ (F\ n)$
proof −
have $\bigwedge i.\ i \leq n \Longrightarrow (X\ i) \in measurable\ (F\ n)\ borel$
by (*meson adapt-stoch-proc-def assms increasing-measurable-info*)
have $(\bigcap i \in \{m.\ m \leq n\}.\ (X\ i) -` \{X\ i\ w\}) \in sets\ (F\ n)$
proof ((*rule sigma-algebra.countable-INT''*), *auto*)
show *sigma-algebra* $(space\ (F\ n))\ (sets\ (F\ n))$
using *measure-space measure-space-def* **by** *auto*
show $UNIV \in sets\ (F\ n)$
using ‹*sigma-algebra* $(space\ (F\ n))\ (sets\ (F\ n))$› *nat-filtration-space natural-filtration*
sigma-algebra.sigma-sets-eq sigma-sets-top **by** *fastforce*
have $\bigwedge i.\ i \leq n \Longrightarrow (X\ i) -` \{X\ i\ w\} \in sets\ (nat\text{-}filtration\ n)$
proof (*rule nat-filtration-borel-measurable-singleton*)
fix *i*
assume $i \leq n$
show $X\ i \in borel\text{-}measurable\ (nat\text{-}filtration\ n)$ **using** *assms natural-filtration* **unfolding** *adapt-stoch-proc-def*
using ‹$i \leq n$› *increasing-measurable-info* **by** *blast*
qed
thus $\bigwedge i.\ i \leq n \Longrightarrow (X\ i) -` \{X\ i\ w\} \in sets\ (F\ n)$ **using** *natural-filtration* **by** *simp*
qed
thus *?thesis* **using** *assms* **by** (*simp add*: *proj-stoch-preimage*)
qed

lemma (**in** *infinite-cts-filtration*) *finite-range-stream-space*:
fixes f::$'a \Rightarrow 'b$::*t1-space*
assumes *finite* $(range\ f)$
shows $(\lambda w.\ snth\ w\ i) -` (open\text{-}exclude\text{-}set\ (f\ x)\ (range\ f)) \in sets\ (stream\text{-}space\ borel)$
proof −
define *opex* **where** $opex = open\text{-}exclude\text{-}set\ (f\ x)\ (range\ f)$

have *open opex* **and** *opex* ∩ (*range f*) = {*f x*} **using** *assms* **unfolding** *opex-def*
by
(*auto simp add:open-exclude-finite*)
hence *opex*∈ *sets borel* **by** *simp*
hence *vim*: (λ*w*. *snth w i*) −‘ *opex* ∈ *sets* (*vimage-algebra* (*streams* (*space borel*))
(λ*w*. *snth w i*) *borel*)
by (*metis in-vimage-algebra inf-top.right-neutral space-borel streams-UNIV*)
have (λ*w*. *snth w i*) −‘ *opex* ∈ *sets* (⨆ *i*. *vimage-algebra* (*streams* (*space borel*))
(λ*w*. *snth w i*) *borel*)
proof (*rule in-sets-SUP*, *simp*)
show ⋀*i*. *i* ∈ *UNIV* ⟹ *space* (*vimage-algebra* (*streams* (*space borel*)) (λ*w*. *w*
!! *i*) *borel*) =
UNIV **by** *simp*
show (λ*w*. *w* !! *i*) −‘ *opex* ∈ *sets* (*vimage-algebra* (*streams* (*space borel*)) (λ*w*.
w !! *i*) *borel*)
using *vim* **by** *simp*
qed
thus *?thesis* **using** *sets-stream-space-eq* **unfolding** *opex-def* **by** *blast*
qed

lemma (**in** *infinite-cts-filtration*) *proj-stoch-range-singleton*:
fixes *X*::*nat* ⇒ *bool stream* ⇒ ('*b*::*t2-space*)
assumes *borel-adapt-stoch-proc F X*
and *r*∈ *range* (*proj-stoch-proc X n*)
shows ∃ *A*∈*sets* (*stream-space borel*). *range* (*proj-stoch-proc X n*) ∩ *A* = {*r*}
proof −
have ∃ *x*. *r* = *proj-stoch-proc X n x* **using** *assms* **by** *auto*
from *this* **obtain** *x* **where** *r* = *proj-stoch-proc X n x* **by** *auto*
have ⋀*i*. *i* ≤ *n* ⟹ (*X i*) ∈ *measurable* (*F n*) *borel*
by (*meson adapt-stoch-proc-def assms increasing-measurable-info*)
hence *fin*: ⋀*i*. *i* ≤ *n* ⟹ *finite* (*range* (*X i*))
by (*metis bernoulli bernoulli-stream-space nat-filtration-vimage-finite natural-filtration streams-UNIV*)
show *?thesis*
proof
define *cand* **where** *cand* = (⋂ *i* ∈ {*m*. *m*≤ *n*}. (λ*w*. *snth w i*) −‘ (*open-exclude-set*
(*X i x*) (*range* (*X i*))))
show *cand* ∈ *sets* (*stream-space borel*) **unfolding** *cand-def*
proof ((*rule sigma-algebra.countable-INT''*), *auto*)
show *UNIV* ∈ *sets* (*stream-space borel*) **by** (*metis space-borel streams-UNIV*
streams-stream-space)
show *sigma-algebra* (*space* (*stream-space borel*)) (*sets* (*stream-space borel*))
by (*simp add*: *sets.sigma-algebra-axioms*)
show ⋀*i*. *i* ≤ *n* ⟹ (λ*w*. *w* !! *i*) −‘ *open-exclude-set* (*X i x*) (*range* (*X i*)) ∈
sets (*stream-space borel*)
proof −
fix *i*
assume *i* ≤ *n*
thus (λ*w*. *w* !! *i*) −‘ *open-exclude-set* (*X i x*) (*range* (*X i*)) ∈ *sets* (*stream-space*

borel)
        using *fin* **by** (*simp add:finite-range-stream-space*)
      qed
    qed
    have *range* (*proj-stoch-proc X n*) ∩ *cand* = {*proj-stoch-proc X n x*}
    proof
      have *proj-stoch-proc X n x* ∈ *range* (*proj-stoch-proc X n*) ∩ *cand*
      proof
        show *proj-stoch-proc X n x* ∈ *range* (*proj-stoch-proc X n*) **by** *simp*
        show *proj-stoch-proc X n x* ∈ *cand* **unfolding** *cand-def*
        proof
          fix *i*
          assume *i*∈ {*m*. *m*≤ *n*}
          hence *i* ≤ *n* **by** *simp*
          hence *snth* (*proj-stoch-proc X n x*) *i* = *X i x*
            by (*metis le-antisym not-less proj-stoch-proc-component*)
             also have ... ∈ *open-exclude-set* (*X i x*) (*range* (*X i*)) **using** *assms open-exclude-finite*(*2*)
            by (*metis IntE* ‹*i* ≤ *n*› *fin insert-iff rangeI*)
           finally have *snth* (*proj-stoch-proc X n x*) *i* ∈ *open-exclude-set* (*X i x*) (*range* (*X i*)) **.**
            thus *proj-stoch-proc X n x* ∈ (λ*w*. *w* !! *i*) −‘ *open-exclude-set* (*X i x*) (*range* (*X i*)) **by** *simp*
        qed
      qed
      thus {*proj-stoch-proc X n x*} ⊆ *range* (*proj-stoch-proc X n*) ∩ *cand* **by** *simp*
      show *range* (*proj-stoch-proc X n*) ∩ *cand* ⊆ {*proj-stoch-proc X n x*}
      proof
        fix *z*
        assume *z*∈ *range* (*proj-stoch-proc X n*) ∩ *cand*
        hence ∃ *y*. *z* = *proj-stoch-proc X n y* **by** *auto*
        from *this* **obtain** *y* **where** *z* = *proj-stoch-proc X n y* **by** *auto*
        hence *proj-stoch-proc X n y* ∈ *cand* **using** ‹*z*∈ *range* (*proj-stoch-proc X n*) ∩ *cand*› **by** *simp*
        have ∀ *i*. *i*≤*n* ⟶ *X i y* = *X i x*
        proof (*intro allI impI*)
          fix *i*
          assume *i* ≤ *n*
          hence *X i y* = *snth* (*proj-stoch-proc X n y*) *i*
            by (*metis le-antisym not-less proj-stoch-proc-component*)
          also have ... ∈ *open-exclude-set* (*X i x*) (*range* (*X i*))
             using ‹*proj-stoch-proc X n y* ∈ *cand*› ‹*i* ≤ *n*› **unfolding** *cand-def* **by** *simp*
          finally have *X i y* ∈ *open-exclude-set* (*X i x*) (*range* (*X i*)) **.**
          thus *X i y* = *X i x* **using** *assms* **using** *assms open-exclude-finite*(*2*)
            by (*metis Int-iff* ‹*i* ≤ *n*› *empty-iff fin insert-iff rangeI*)
        qed
        hence ∀ *i*. *snth* (*proj-stoch-proc X n y*) *i* = *snth* (*proj-stoch-proc X n x*) *i*
          using *proj-stoch-proc-component* **by** (*metis nat-le-linear not-less*)

hence *proj-stoch-proc X n y = proj-stoch-proc X n x*
using *diff-streams-only-if* **by** *auto*
thus *z*∈ {*proj-stoch-proc X n x*} **using** ‹*z = proj-stoch-proc X n y*› **by** *auto*
qed
qed
thus *range* (*proj-stoch-proc X n*) ∩ *cand* = {*r*} **using** ‹*r = proj-stoch-proc X n x*› **by** *simp*
qed
qed

definition (**in** *infinite-cts-filtration*) *stream-space-single* **where**
stream-space-single X r = (*if* (∃ *U*. *U*∈ *sets* (*stream-space borel*) ∧ *U*∩ (*range X*) = {*r*})
then SOME U. *U*∈ *sets* (*stream-space borel*) ∧ *U* ∩ (*range X*) = {*r*} *else* {})

lemma (**in** *infinite-cts-filtration*) *stream-space-singleI*:
assumes ∃ *U*. *U*∈ *sets* (*stream-space borel*) ∧ *U*∩ (*range X*) = {*r*}
shows *stream-space-single X r* ∈ *sets* (*stream-space borel*) ∧ *stream-space-single X r* ∩ (*range X*) = {*r*}
proof −
let *?V* = *SOME U*. *U*∈ *sets* (*stream-space borel*) ∧ *U*∩ (*range X*) = {*r*}
have *vprop*: *?V*∈ *sets* (*stream-space borel*) ∧ *?V* ∩ (*range X*) = {*r*} **using** *someI-ex*[*of* λ*U*. *U*∈ *sets* (*stream-space borel*) ∧ *U*∩ (*range X*) = {*r*}]
assms **by** *blast*
show *?thesis* **by** (*simp add*:*stream-space-single-def vprop assms*)
qed

lemma (**in** *infinite-cts-filtration*)
fixes *X*::*nat* ⇒ *bool stream* ⇒ (*'b*::*t2-space*)
assumes *borel-adapt-stoch-proc F X*
and *r*∈ *range* (*proj-stoch-proc X n*)
shows *stream-space-single-set*: *stream-space-single* (*proj-stoch-proc X n*) *r* ∈ *sets* (*stream-space borel*)
and *stream-space-single-preimage*: *stream-space-single* (*proj-stoch-proc X n*) *r* ∩ *range* (*proj-stoch-proc X n*) = {*r*}
proof −
have ∃ *A*∈*sets* (*stream-space borel*). *range* (*proj-stoch-proc X n*) ∩ *A* = {*r*}
by (*simp add*: *assms proj-stoch-range-singleton*)
hence ∃ *U*. *U* ∈ *sets* (*stream-space borel*) ∧ *U* ∩ *range* (*proj-stoch-proc X n*) = {*r*} **by** *auto*
hence *a*: *stream-space-single* (*proj-stoch-proc X n*) *r* ∈ *sets* (*stream-space borel*) ∧
stream-space-single (*proj-stoch-proc X n*) *r* ∩ (*range* (*proj-stoch-proc X n*)) = {*r*}
using *stream-space-singleI*[*of proj-stoch-proc X n*] **by** *simp*
thus *stream-space-single* (*proj-stoch-proc X n*) *r* ∈ *sets* (*stream-space borel*) **by** *simp*
show *stream-space-single* (*proj-stoch-proc X n*) *r* ∩ *range* (*proj-stoch-proc X n*) = {*r*} **using** *a* **by** *simp*

qed

5.4.2 Induced filtration, relationship with filtration generated by underlying stochastic process

definition *comp-proj-i* **where**
comp-proj-i X n i y = {*z*∈ *range* (*proj-stoch-proc X n*). *snth z i* = *y*}

lemma (**in** *infinite-cts-filtration*) *comp-proj-i-finite*:
fixes *X*::*nat* ⇒ *bool stream* ⇒ *'b*::{*t0-space*}
assumes *borel-adapt-stoch-proc F X*
shows *finite* (*comp-proj-i X n i y*)
proof −
have *finite* (*range* (*proj-stoch-proc X n*))
using *proj-stoch-set-finite-range*[*of X*] *assms* **by** *simp*
thus *?thesis* **unfolding** *comp-proj-i-def* **by** *simp*
qed

lemma *stoch-proc-comp-proj-i-preimage*:
assumes *i* ≤ *n*
shows (*X i*) −‘ {*X i x*} = (⋃ *z*∈ *comp-proj-i X n i* (*X i x*). (*proj-stoch-proc X n*) −‘ {*z*})
proof
show *X i* −‘ {*X i x*} ⊆ (⋃ *z*∈*comp-proj-i X n i* (*X i x*). *proj-stoch-proc X n* −‘ {*z*})
proof
fix *w*
assume *w* ∈ *X i* −‘ {*X i x*}
hence *X i w* = *X i x* **by** *simp*
hence *snth* (*proj-stoch-proc X n w*) *i* = *X i x* **using** *assms*
by (*metis le-neq-implies-less proj-stoch-proc-component*(*1*) *proj-stoch-proc-component*(*2*))
hence (*proj-stoch-proc X n w*) ∈ *comp-proj-i X n i* (*X i x*) **unfolding** *comp-proj-i-def* **by** *simp*
moreover have *w*∈ *proj-stoch-proc X i* −‘ {*proj-stoch-proc X i w*} **by** *simp*
ultimately show *w*∈ (⋃ *z*∈*comp-proj-i X n i* (*X i x*). *proj-stoch-proc X n* −‘ {*z*}) **by** *simp*
qed
show (⋃ *z*∈*comp-proj-i X n i* (*X i x*). *proj-stoch-proc X n* −‘ {*z*}) ⊆ *X i* −‘ {*X i x*}
proof −
have ∀ *z*∈ *comp-proj-i X n i* (*X i x*). *proj-stoch-proc X n* −‘ {*z*} ⊆ *X i* −‘ {*X i x*}
proof
fix *z*
assume *z*∈ *comp-proj-i X n i* (*X i x*)
hence *z*∈ *range* (*proj-stoch-proc X n*) **and** *snth z i* = *X i x* **unfolding** *comp-proj-i-def* **by** *auto*
show *proj-stoch-proc X n* −‘ {*z*} ⊆ *X i* −‘ {*X i x*}
proof

fix w
assume $w \in$ *proj-stoch-proc* $X\ n -`\{z\}$
have $X\ i\ w =$ *snth* (*proj-stoch-proc* $X\ n\ w$) i **using** *assms*
by (*metis le-neq-implies-less proj-stoch-proc-component*(*1*) *proj-stoch-proc-component*(*2*))
also have ... = *snth* $z\ i$ **using** ‹$w \in$ *proj-stoch-proc* $X\ n -`\{z\}$› **by** *auto*
also have ... = $X\ i\ x$ **using** ‹*snth* $z\ i = X\ i\ x$› **by** *simp*
finally have $X\ i\ w = X\ i\ x$ **.**
thus $w \in X\ i -`\{X\ i\ x\}$ **by** *simp*
qed
qed
thus *?thesis* **by** *auto*
qed
qed

definition *comp-proj* **where**
comp-proj $X\ n\ y = \{z \in$ *range* (*proj-stoch-proc* $X\ n$). *snth* $z\ n = y\}$

lemma (**in** *infinite-cts-filtration*) *comp-proj-finite*:
fixes X::*nat* $\Rightarrow$ *bool stream* $\Rightarrow$ *'b*::{*t0-space*}
assumes *borel-adapt-stoch-proc* $F\ X$
shows *finite* (*comp-proj* $X\ n\ y$)
proof −
have *finite* (*range* (*proj-stoch-proc* $X\ n$))
using *proj-stoch-set-finite-range*[*of* X] *assms* **by** *simp*
thus *?thesis* **unfolding** *comp-proj-def* **by** *simp*
qed

lemma *stoch-proc-comp-proj-preimage*:
shows $(X\ n) -`\{X\ n\ x\} = (\bigcup z \in$ *comp-proj* $X\ n\ (X\ n\ x)$. (*proj-stoch-proc* $X\ n$) $-`\{z\})$
proof
show $X\ n -`\{X\ n\ x\} \subseteq (\bigcup z \in$*comp-proj* $X\ n\ (X\ n\ x)$. *proj-stoch-proc* $X\ n -`\{z\})$
proof
fix w
assume $w \in X\ n -`\{X\ n\ x\}$
hence $X\ n\ w = X\ n\ x$ **by** *simp*
hence *snth* (*proj-stoch-proc* $X\ n\ w$) $n = X\ n\ x$ **by** (*simp add*: *proj-stoch-proc-component*(*2*))
hence (*proj-stoch-proc* $X\ n\ w$) $\in$ *comp-proj* $X\ n\ (X\ n\ x)$ **unfolding** *comp-proj-def*
by *simp*
moreover have $w \in$ *proj-stoch-proc* $X\ n -`\{$*proj-stoch-proc* $X\ n\ w\}$ **by** *simp*
ultimately show $w \in (\bigcup z \in$*comp-proj* $X\ n\ (X\ n\ x)$. *proj-stoch-proc* $X\ n -`\{z\})$
by *simp*
qed
show $(\bigcup z \in$*comp-proj* $X\ n\ (X\ n\ x)$. *proj-stoch-proc* $X\ n -`\{z\}) \subseteq X\ n -`\{X\ n$

```
x}
  proof −
    have ∀ z∈ comp-proj X n (X n x). proj-stoch-proc X n −‘ {z} ⊆ X n −‘ {X n
x}
    proof
      fix z
      assume z∈ comp-proj X n (X n x)
       hence z∈ range (proj-stoch-proc X n) and snth z n = X n x unfolding
comp-proj-def by auto
      show proj-stoch-proc X n −‘ {z} ⊆ X n −‘ {X n x}
      proof
        fix w
        assume w∈proj-stoch-proc X n −‘ {z}
      have X n w = snth (proj-stoch-proc X n w) n by (simp add: proj-stoch-proc-component(2))
        also have ... = snth z n using ‹w∈proj-stoch-proc X n −‘ {z}› by auto
        also have ... = X n x using ‹snth z n = X n x› by simp
        finally have X n w = X n x .
        thus w∈ X n −‘ {X n x} by simp
      qed
    qed
    thus ?thesis by auto
  qed
qed


lemma comp-proj-stoch-proc-preimage:
  shows (proj-stoch-proc X n) −‘ {proj-stoch-proc X n x} = (⋂ i∈ {m. m≤n}. (X
i) −‘{X i x})
proof
  show proj-stoch-proc X n −‘ {proj-stoch-proc X n x} ⊆ (⋂i∈{m. m ≤ n}. X i
−‘ {X i x})
  proof
    fix z
    assume z∈ proj-stoch-proc X n −‘ {proj-stoch-proc X n x}
    hence proj-stoch-proc X n z = proj-stoch-proc X n x by simp
   hence ∀ i≤n. X i z = X i x using proj-stoch-proc-component by (metis less-le)
    hence ∀ i≤n. z∈ X i −‘{X i x} by simp
    thus z∈ (⋂ i∈{m. m ≤ n}. X i −‘ {X i x}) by simp
  qed
 show (⋂ i∈{m. m ≤ n}. X i −‘ {X i x}) ⊆ proj-stoch-proc X n −‘ {proj-stoch-proc
X n x}
  proof
    fix z
    assume z∈ (⋂ i∈{m. m ≤ n}. X i −‘ {X i x})
    hence ∀ i≤ n. X i z = X i x by auto
    hence ∀ i. snth (proj-stoch-proc X n z) i = snth (proj-stoch-proc X n x) i
      using proj-stoch-proc-component by (metis nat-le-linear not-less)
    hence proj-stoch-proc X n z = proj-stoch-proc X n x using diff-streams-only-if
by auto
```

thus $z\in$ *proj-stoch-proc X n* $-`$ $\{$*proj-stoch-proc X n x*$\}$ **by** *simp*
qed
qed

definition *stoch-proc-filt* **where**
stoch-proc-filt M X N (*n*::*nat*) = *gen-subalgebra M* (*sigma-sets* (*space M*) ($\bigcup$ *i*$\in$ $\{$*m*. *m*$\leq$ *n*$\}$. $\{$(*X i* $-`$*A*) $\cap$ (*space M*) | *A*. *A*$\in$ *sets N* $\}$))

lemma *stoch-proc-filt-space*:
shows *space* (*stoch-proc-filt M X N n*) = *space M* **unfolding** *stoch-proc-filt-def*
by (*simp add*:*gen-subalgebra-space*)

lemma *stoch-proc-filt-sets*:
assumes $\bigwedge$*i*. *i* $\leq$ *n*$\Longrightarrow$ (*X i*) $\in$ *measurable M N*
shows *sets* (*stoch-proc-filt M X N n*) = (*sigma-sets* (*space M*) ($\bigcup$ *i*$\in$ $\{$*m*. *m*$\leq$ *n*$\}$. $\{$(*X i* $-`$*A*) $\cap$ (*space M*) | *A*. *A*$\in$ *sets N* $\}$))
unfolding *stoch-proc-filt-def*
proof (*rule gen-subalgebra-sigma-sets*)
show *sigma-algebra* (*space M*) (*sigma-sets* (*space M*) ($\bigcup$*i*$\in${*m*. *m* $\leq$ *n*}. {*X i* $-`$ *A* $\cap$ *space M* |*A*. *A* $\in$ *sets N*})) **using** *assms*
by (*simp add*: *adapt-sigma-sets*)
show *sigma-sets* (*space M*) ($\bigcup$*i*$\in${*m*. *m* $\leq$ *n*}. {*X i* $-`$ *A* $\cap$ *space M* |*A*. *A* $\in$ *sets N*}) $\subseteq$ *sets M*
proof (*rule sigma-algebra.sigma-sets-subset*, *rule Sigma-Algebra.sets.sigma-algebra-axioms*, *rule UN-subset-iff*[*THEN iffD2*], *intro ballI*)
fix *i*
assume *i*$\in$ $\{$*m*. *m*$\leq$*n*$\}$
thus {*X i* $-`$ *A* $\cap$ *space M* |*A*. *A* $\in$ *sets N*} $\subseteq$ *sets M* **using** *assms measurable-sets* **by** *blast*
qed
qed

lemma *stoch-proc-filt-adapt*:
assumes $\bigwedge$*n*. *X n* $\in$ *measurable M N*
shows *adapt-stoch-proc* (*stoch-proc-filt M X N*) *X N* **unfolding** *adapt-stoch-proc-def*
proof
fix *m*
show *X m* $\in$ *measurable* (*stoch-proc-filt M X N m*) *N* **unfolding** *measurable-def*
proof ((*intro CollectI*), (*intro conjI*))
have *space* (*stoch-proc-filt M X N m*) = *space M* **by** (*simp add*: *stoch-proc-filt-space*)
thus *X m* $\in$ *space* (*stoch-proc-filt M X N m*) $\rightarrow$ *space N* **using** *assms* **unfolding**

measurable-def **by** *simp*
show $\forall y{\in}sets\ N.\ X\ m\ -`\ y \cap space\ (stoch\text{-}proc\text{-}filt\ M\ X\ N\ m) \in sets\ (stoch\text{-}proc\text{-}filt\ M\ X\ N\ m)$
proof
fix *B*
assume $B{\in}\ sets\ N$
have $X\ m\ -`\ B \cap space\ (stoch\text{-}proc\text{-}filt\ M\ X\ N\ m) = X\ m\ -`B \cap space\ M$
using ‹$space\ (stoch\text{-}proc\text{-}filt\ M\ X\ N\ m) = space\ M$› **by** *simp*
also have ... $\in (\bigcup\ i{\in}\ \{p.\ p{\leq}\ m\}.\ \{(X\ i\ -`A) \cap (space\ M)\ |\ A.\ A{\in}\ sets\ N\ \})$ **using** ‹$B{\in}\ sets\ N$› **by** *auto*
also have ... $\subseteq$ *sigma-sets* $(space\ M)\ (\bigcup\ i{\in}\ \{p.\ p{\leq}\ m\}.\ \{(X\ i\ -`A) \cap (space\ M)\ |\ A.\ A{\in}\ sets\ N\ \})$ **by** *auto*
also have ... $=$ *sets* $(stoch\text{-}proc\text{-}filt\ M\ X\ N\ m)$ **using** *assms stoch-proc-filt-sets* **by** *blast*
finally show $X\ m\ -`\ B \cap space\ (stoch\text{-}proc\text{-}filt\ M\ X\ N\ m) \in sets\ (stoch\text{-}proc\text{-}filt\ M\ X\ N\ m)$ **.**
qed
qed
qed

lemma *stoch-proc-filt-disc-filtr*:
assumes $\bigwedge i.\ (X\ i) \in measurable\ M\ N$
shows *disc-filtr* M $(stoch\text{-}proc\text{-}filt\ M\ X\ N)$ **unfolding** *disc-filtr-def*
proof (*intro conjI allI impI*)
{
fix *n*
show *subalgebra* M $(stoch\text{-}proc\text{-}filt\ M\ X\ N\ n)$ **unfolding** *subalgebra-def*
proof
show $space\ (stoch\text{-}proc\text{-}filt\ M\ X\ N\ n) = space\ M$ **by** (*simp add*:*stoch-proc-filt-space*)
show $sets\ (stoch\text{-}proc\text{-}filt\ M\ X\ N\ n) \subseteq sets\ M$
proof −
have *sigma-sets* $(space\ M)\ (\bigcup i{\in}\{m.\ m \leq n\}.\ \{X\ i\ -`\ A \cap space\ M\ |A.\ A \in sets\ N\}) \subseteq sets\ M$
proof (*rule sigma-algebra.sigma-sets-subset*, *rule Sigma-Algebra.sets.sigma-algebra-axioms*, *rule UN-subset-iff*[*THEN iffD2*], *intro ballI*)
fix *i*
assume $i{\in}\ \{m.\ m{\leq}n\}$
thus $\{X\ i\ -`\ A \cap space\ M\ |A.\ A \in sets\ N\} \subseteq sets\ M$ **using** *assms measurable-sets* **by** *blast*
qed
thus *?thesis* **using** *assms* **by** (*simp add*:*stoch-proc-filt-sets*)
qed
qed
}
{
fix *n*
fix *p*

assume $(n::nat) \leq p$
show *subalgebra* (*stoch-proc-filt M X N p*) (*stoch-proc-filt M X N n*) **unfolding** *subalgebra-def*
proof
have *space* (*stoch-proc-filt M X N n*) = *space M* **by** (*simp add*: *stoch-proc-filt-space*)
also have ... = *space* (*stoch-proc-filt M X N p*) **by** (*simp add*: *stoch-proc-filt-space*)
finally show *space* (*stoch-proc-filt M X N n*) = *space* (*stoch-proc-filt M X N p*)
.
show *sets* (*stoch-proc-filt M X N n*) ⊆ *sets* (*stoch-proc-filt M X N p*)
proof −
have *sigma-sets* (*space M*) (⋃ *i*∈{*m*. *m* ≤ *n*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets N*}) ⊆
sigma-sets (*space M*) (⋃ *i*∈{*m*. *m* ≤ *p*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets N*})
proof (*rule sigma-sets-mono′*)
show (⋃ *i*∈{*m*. *m* ≤ *n*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets N*}) ⊆ (⋃ *i*∈{*m*. *m* ≤ *p*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets N*})
proof (*rule UN-subset-iff*[*THEN iffD2*], *intro ballI*)
fix *i*
assume *i*∈ {*m*. *m*≤*n*}
show {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets N*} ⊆ (⋃ *i*∈{*m*. *m* ≤ *p*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets N*})
using ‹*i* ∈ {*m*. *m* ≤ *n*}› ‹*n* ≤ *p*› *order-trans* **by** *auto*
qed
qed
thus *?thesis* **using** *assms* **by** (*simp add*:*stoch-proc-filt-sets*)
qed
qed
}
qed

lemma *gen-subalgebra-eq-space-sets*:
assumes *space M* = *space N*
and *P* = *Q*
and *P*⊆ *Pow* (*space M*)
shows *sets* (*gen-subalgebra M P*) = *sets* (*gen-subalgebra N Q*) **unfolding** *gen-subalgebra-def*
using *assms* **by** *simp*

lemma *stoch-proc-filt-eq-sets*:
assumes *space M* = *space N*
shows *sets* (*stoch-proc-filt M X P n*) = *sets* (*stoch-proc-filt N X P n*) **unfolding** *stoch-proc-filt-def*
proof (*rule gen-subalgebra-eq-space-sets*, (*simp add*: *assms*)+)
show *sigma-sets* (*space N*) (⋃ *x*∈{*m*. *m* ≤ *n*}. {*X x* −‘ *A* ∩ *space N* |*A*. *A* ∈ *sets P*}) ⊆ *Pow* (*space N*)
proof (*rule sigma-algebra.sigma-sets-subset*)
show *sigma-algebra* (*space N*) (*Pow* (*space N*)) **by** (*simp add*: *sigma-algebra-Pow*)
show (⋃ *x*∈{*m*. *m* ≤ *n*}. {*X x* −‘ *A* ∩ *space N* |*A*. *A* ∈ *sets P*}) ⊆ *Pow* (*space*

```
N) by auto
  qed
qed


lemma (in infinite-cts-filtration) stoch-proc-filt-triv-init:
  fixes X::nat ⇒ bool stream ⇒ real
  assumes borel-adapt-stoch-proc nat-filtration X
  shows init-triv-filt M (stoch-proc-filt M X borel) unfolding init-triv-filt-def
proof
  show filtration M (stoch-proc-filt M X borel) using stoch-proc-filt-disc-filtr un-
folding filtration-def
   by (metis adapt-stoch-proc-def assms disc-filtr-def measurable-from-subalg nat-filtration-subalgebra)
  show sets (stoch-proc-filt M X borel bot) = {{}, space M}
  proof −
    have seteq: sets (stoch-proc-filt M X borel 0) =
      (sigma-sets (space M) (⋃ i∈ {m. m≤ 0}. {(X i −‘A) ∩ (space M) | A. A∈
sets borel}))
    proof (rule stoch-proc-filt-sets)
      show ⋀i. i ≤ 0 ⟹ random-variable borel (X i)
      proof −
        fix i::nat
        assume i ≤ 0
      show random-variable borel (X i) using assms unfolding adapt-stoch-proc-def
          using filtration-measurable nat-discrete-filtration
          using natural-filtration by blast
      qed
    qed
  have triv-init-disc-filtr-prob-space M nat-filtration
    proof (unfold-locales, intro conjI)
      show disc-filtr M nat-filtration unfolding disc-filtr-def
        using filtrationE2 nat-discrete-filtration nat-filtration-subalgebra by auto
       show sets (nat-filtration ⊥) = {{}, space M} using nat-info-filtration un-
folding init-triv-filt-def by simp
    qed
    hence ∃ c. ∀ w ∈ space M. ((X 0 w)::real) = c using assms
       triv-init-disc-filtr-prob-space.adapted-init[of M nat-filtration X] by simp
    from this obtain c where img: ∀ w ∈ space M. (X 0 w) = c by auto
    have (⋃ i∈ {m. m≤ 0}. {(X i −‘A) ∩ (space M) | A. A∈ sets borel}) =
     {(X 0 −‘A) ∩ (space M) | A. A∈ sets borel} by auto
    also have ... = {{}, space M}
    proof
      show {X 0 −‘ A ∩ space M |A. A ∈ sets borel} ⊆ {{}, space M}
      proof −
        have ∀ A ∈ sets borel. (X 0 −‘A) ∩ (space M) ∈ {{}, space M}
        proof
          fix A::real set
          assume A∈ sets borel
          show (X 0 −‘A) ∩ (space M) ∈ {{}, space M}
```

```
        proof (cases c∈ A)
          case True
          hence X 0 −‘ A ∩ space M = space M using img by auto
          thus ?thesis by simp
        next
          case False
          hence X 0 −‘ A ∩ space M = {} using img by auto
          thus ?thesis by simp
        qed
      qed
      thus ?thesis by auto
    qed
    show {{}, space M} ⊆ {X 0 −‘ A ∩ space M |A. A ∈ sets borel}
    proof −
      have {} ∈ {X 0 −‘ A ∩ space M |A. A ∈ sets borel} by blast
      moreover have space M ∈ {X 0 −‘ A ∩ space M |A. A ∈ sets borel}
      proof −
        have UNIV ⊆ X 0 −‘ space borel
          using space-borel by blast
        then show ?thesis
          using bernoulli-stream-space by blast
      qed
      ultimately show ?thesis by auto
    qed
  qed
  finally have (⋃ i∈ {m. m≤ 0}. {(X i −‘A) ∩ (space M) | A. A∈ sets borel})
= {{}, space M} .
  moreover have sigma-sets (space M) {{}, space M} = {{}, space M}
  proof −
    have sigma-sets (space M) {space M} = {{}, space M} by simp
     have sigma-sets (space M) (sigma-sets (space M) {space M}) = sigma-sets
(space M) {space M}
      by (rule sigma-sets-sigma-sets-eq, simp)
    also have ... = {{}, space M} by simp
    finally show ?thesis by simp
  qed
  ultimately show ?thesis using seteq by (simp add: bot-nat-def)
 qed
qed

lemma (in infinite-cts-filtration) stream-space-borel-union:
fixes X::nat ⇒ bool stream ⇒ ('b::t2-space)
 assumes borel-adapt-stoch-proc F X
 and i≤ n
 and A∈ sets borel
shows ∀ y∈ A∩ range (X i). X i −‘{y} = (proj-stoch-proc X n) −‘ (⋃ z∈ comp-proj-i
X n i y.
   (stream-space-single (proj-stoch-proc X n) z))
proof
```

fix y
assume $y \in A \cap$ *range* $(X\ i)$
hence $\exists x.\ y = X\ i\ x$ **by** *auto*
from *this* **obtain** x **where** $y = X\ i\ x$ **by** *auto*
hence $X\ i -` \{y\} = X\ i -` \{X\ i\ x\}$ **by** *simp*
also have ... $= (\bigcup z \in$ *comp-proj-i* $X\ n\ i\ (X\ i\ x).$ (*proj-stoch-proc* $X\ n) -` \{z\})$
using ‹$i \leq n$› **by** (*simp add*: *stoch-proc-comp-proj-i-preimage*)
also have ... $= (\bigcup z \in$ *comp-proj-i* $X\ n\ i\ (X\ i\ x).$ (*proj-stoch-proc* $X\ n) -`$
(*stream-space-single* (*proj-stoch-proc* $X\ n$) z))
proof −
have $\forall z \in$ *comp-proj-i* $X\ n\ i\ (X\ i\ x).$ (*proj-stoch-proc* $X\ n) -` \{z\} =$ (*proj-stoch-proc* $X\ n) -`$
(*stream-space-single* (*proj-stoch-proc* $X\ n$) z)
proof
fix z
assume $z \in$ *comp-proj-i* $X\ n\ i\ (X\ i\ x)$
have *stream-space-single* (*proj-stoch-proc* $X\ n$) $z \cap$ *range* (*proj-stoch-proc* $X\ n) = \{z\}$
using *stream-space-single-preimage assms*
proof −
have $z \in$ *range* (*proj-stoch-proc* $X\ n$)
using ‹$z \in$ *comp-proj-i* $X\ n\ i\ (X\ i\ x)$› *comp-proj-i-def* **by** *force*
then show *?thesis*
by (*meson assms stream-space-single-preimage*)
qed
thus (*proj-stoch-proc* $X\ n) -` \{z\} =$ (*proj-stoch-proc* $X\ n) -`$
(*stream-space-single* (*proj-stoch-proc* $X\ n$) z) **by** *auto*
qed
thus *?thesis* **by** *auto*
qed
also have ... $=$ *proj-stoch-proc* $X\ n -` (\bigcup z \in$ *comp-proj-i* $X\ n\ i\ y.$ (*stream-space-single* (*proj-stoch-proc* $X\ n$) $z))$
by (*simp add*: ‹$y = X\ i\ x$› *vimage-Union*)
finally show $X\ i -` \{y\} =$ (*proj-stoch-proc* $X\ n) -` (\bigcup z \in$ *comp-proj-i* $X\ n\ i\ y.$
(*stream-space-single* (*proj-stoch-proc* $X\ n$) z)) **using** ‹$y = X\ i\ x$› **by** *simp*
qed

lemma (**in** *infinite-cts-filtration*) *proj-stoch-pre-borel*:
fixes X::*nat* $\Rightarrow$ *bool stream* $\Rightarrow$ ('*b*::*t2-space*)
assumes *borel-adapt-stoch-proc* $F\ X$
shows *proj-stoch-proc* $X\ n -` \{$*proj-stoch-proc* $X\ n\ x\} \in$ *sets* (*stoch-proc-filt* M X *borel* n)
proof −
have *proj-stoch-proc* $X\ n -` \{$*proj-stoch-proc* $X\ n\ x\} = (\bigcap\ i \in \{m.\ m \leq n\}.\ (X\ i) -` \{X\ i\ x\})$
by (*simp add*:*comp-proj-stoch-proc-preimage*)
also have ... $\in$ *sigma-sets* (*space* M) $(\bigcup i \in \{m.\ m \leq n\}.\ \{X\ i -` A \cap$ *space* M

```
|A. A ∈ sets borel})
  proof −
    have inset: ∀ i≤n. (X i) −‘{X i x} ∈ {X i −‘ A ∩ space M |A. A ∈ sets borel}
    proof (intro allI impI)
      fix i
      assume i ≤ n
      have ∃ U. open U ∧ U∩ (range (X i)) = {X i x}
      proof −
        have ∃ U. open U ∧ X i x∈ U ∧ U∩ ((range (X i))−{X i x}) = {}
        proof (rule open-except-set)
          have finite (range (X i)) using assms
            by (metis adapt-stoch-proc-def bernoulli bernoulli-stream-space
              nat-filtration-vimage-finite natural-filtration streams-UNIV)
          thus finite (range (X i) −{X i x}) by auto
          show X i x∉ (range (X i)) −{X i x} by simp
        qed
        thus ?thesis using assms by auto
      qed
       from this obtain U where open U and U∩ (range (X i)) = {X i x} by
auto
       have X i −‘ {X i x} = X i −‘U using ‹U∩ (range (X i)) = {X i x}› by
auto
      also have ... = X i −‘ U ∩ space M using bernoulli bernoulli-stream-space
by simp
      finally have X i −‘ {X i x} = X i −‘ U ∩ space M .
      moreover have U ∈ sets borel using ‹open U› by simp
     ultimately show (X i) −‘{X i x} ∈ {X i −‘ A ∩ space M |A. A ∈ sets borel}
by auto
    qed
    show ?thesis
    proof (rule sigma-set-inter-init)
      show (⋃ i∈{m. m ≤ n}. {X i −‘ A ∩ space M |A. A ∈ sets borel}) ⊆ Pow
(space M) by auto
      show ⋀i. i ≤ n ⟹ X i −‘ {X i x} ∈ sigma-sets (space M) (⋃ i∈{m. m ≤
n}. {X i −‘ A ∩ space M |A. A ∈ sets borel})
      using inset by (metis (no-types, lifting) UN-I mem-Collect-eq sigma-sets.Basic)
    qed
  qed
  also have ... = sets (stoch-proc-filt M X borel n)
  proof (rule stoch-proc-filt-sets[symmetric])
    fix i
    assume i ≤ n
   show random-variable borel (X i) using assms borel-adapt-stoch-proc-borel-measurable
by blast
  qed
  finally show proj-stoch-proc X n −‘ {proj-stoch-proc X n x}
    ∈ sets (stoch-proc-filt M X borel n) .
qed
```

lemma (**in** *infinite-cts-filtration*) *stoch-proc-filt-gen*:
fixes *X*::*nat* ⇒ *bool stream* ⇒ ('*b*::*t2-space*)
assumes *borel-adapt-stoch-proc F X*
shows *stoch-proc-filt M X borel n* = *fct-gen-subalgebra M* (*stream-space borel*) (*proj-stoch-proc X n*)
proof −
have (⋃ *i*∈{*m*. *m* ≤ *n*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets borel*})
⊆ {*proj-stoch-proc X n* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets* (*stream-space borel*)}
proof (*rule UN-subset-iff*[*THEN iffD2*], *intro ballI*)
fix *i*
assume *i*∈ {*m*. *m*≤*n*}
hence *i* ≤ *n* **by** *simp*
show {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets borel*} ⊆
{*proj-stoch-proc X n* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets* (*stream-space borel*)}
proof −
have ⋀*A*. *A*∈ *sets borel* ⟹ *X i* −‘ *A* ∩ *space M* ∈ {*proj-stoch-proc X n* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets* (*stream-space borel*)}
proof −
fix *A*::'*b set*
assume *A*∈ *sets borel*
have *X i* −‘*A* ∩ *space M* = *X i* −‘ *A* **using** *bernoulli bernoulli-stream-space* **by** *simp*
also have ... = *X i* −‘(*A*∩ *range* (*X i*)) **by** *auto*
also have ... = (⋃ *y*∈ *A*∩ *range* (*X i*). *X i* −‘{*y*}) **by** *auto*
also have ... = (⋃ *y*∈ *A*∩ *range* (*X i*). (*proj-stoch-proc X n*) −‘ (⋃*z*∈ *comp-proj-i X n i y*.
(*stream-space-single* (*proj-stoch-proc X n*) *z*))) **using** *stream-space-borel-union assms* ‹*i*≤*n*› ‹*A*∈*sets borel*›
by (*metis* (*mono-tags*, *lifting*) *image-cong*)
also have ... = (*proj-stoch-proc X n*) −‘ (⋃ *y*∈ *A*∩ *range* (*X i*). (⋃*z*∈ *comp-proj-i X n i y*.
(*stream-space-single* (*proj-stoch-proc X n*) *z*))) **by** (*simp add*: *vimage-Union*)
finally have *X i* −‘*A* ∩ *space M* = (*proj-stoch-proc X n*) −‘ (⋃ *y*∈ *A*∩ *range* (*X i*). (⋃*z*∈ *comp-proj-i X n i y*.
(*stream-space-single* (*proj-stoch-proc X n*) *z*))) **.**
moreover have (⋃ *y*∈ *A*∩ *range* (*X i*). (⋃*z*∈ *comp-proj-i X n i y*.
(*stream-space-single* (*proj-stoch-proc X n*) *z*))) ∈ *sets* (*stream-space borel*)
proof −
have *finite* (*A*∩ *range* (*X i*))
proof −
have *finite* (*range* (*X i*)) **using** *assms*
by (*metis adapt-stoch-proc-def bernoulli bernoulli-stream-space nat-filtration-vimage-finite natural-filtration streams-UNIV*)
thus *?thesis* **by** *auto*
qed
moreover have ∀ *y*∈ *A*∩ *range* (*X i*). (⋃*z*∈ *comp-proj-i X n i y*.

(*stream-space-single* (*proj-stoch-proc X n*) *z*)) ∈ *sets* (*stream-space borel*)
proof
fix *y*
assume *y*∈ *A*∩ *range* (*X i*)
have *finite* (*comp-proj-i X n i y*) **by** (*simp add*: *assms comp-proj-i-finite*)
moreover have ∀ *z*∈ *comp-proj-i X n i y*. (*stream-space-single* (*proj-stoch-proc X n*) *z*) ∈ *sets* (*stream-space borel*)
proof
fix *z*
assume *z*∈ *comp-proj-i X n i y*
thus (*stream-space-single* (*proj-stoch-proc X n*) *z*) ∈ *sets* (*stream-space borel*) **using** *assms*
stream-space-single-set **unfolding** *comp-proj-i-def* **by** *auto*
qed
ultimately show (⋃ *z*∈ *comp-proj-i X n i y*. (*stream-space-single* (*proj-stoch-proc X n*) *z*)) ∈
sets (*stream-space borel*) **by** *blast*
qed
ultimately show *?thesis* **by** *blast*
qed
ultimately show *X i* −‘ *A* ∩ *space M* ∈ {*proj-stoch-proc X n* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets* (*stream-space borel*)}
by (*metis* (*mono-tags*, *lifting*) ‹*X i* −‘ *A* ∩ *space M* = *X i* −‘ *A*› *mem-Collect-eq*)
qed
thus *?thesis* **by** *auto*
qed
qed
hence *l*: *sigma-sets* (*space M*) (⋃ *i*∈{*m*. *m* ≤ *n*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets borel*}) ⊆
sigma-sets (*space M*) {*proj-stoch-proc X n* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets* (*stream-space borel*)}
by (*rule sigma-sets-mono*′)
have {*proj-stoch-proc X n* −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets* (*stream-space borel*)}
⊆ *sigma-sets* (*space M*) (⋃ *i*∈{*m*. *m* ≤ *n*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets borel*})
proof −
have ∀ *B*∈ *sets* (*stream-space borel*). *proj-stoch-proc X n* −‘ *B* ∩ *space M* ∈
sigma-sets (*space M*) (⋃ *i*∈ {*m*. *m*≤ *n*}. {(*X i* −‘*A*) ∩ (*space M*) | *A*. *A*∈ *sets borel* })
proof
fix *B*::′*b stream set*
assume *B*∈ *sets* (*stream-space borel*)
have *proj-stoch-proc X n* −‘ *B* ∩ *space M* = *proj-stoch-proc X n* −‘*B* **using** *bernoulli bernoulli-stream-space* **by** *simp*
also have ... = *proj-stoch-proc X n* −‘ (*B* ∩ *range* (*proj-stoch-proc X n*)) **by** *auto*
also have ... = *proj-stoch-proc X n* −‘ (⋃ *y*∈ (*B* ∩ *range* (*proj-stoch-proc X n*)). {*y*}) **by** *simp*

also have ... = ($\bigcup$ $y\in$ ($B \cap$ *range* (*proj-stoch-proc X n*)). *proj-stoch-proc X n* $-$‘$\{y\}$) **by** *auto*
finally have *eqB*: *proj-stoch-proc X n* $-$‘ $B \cap$ *space M* =
($\bigcup$ $y\in$ ($B \cap$ *range* (*proj-stoch-proc X n*)). *proj-stoch-proc X n* $-$‘$\{y\}$) **.**
have $\forall\, y\in$ ($B \cap$ *range* (*proj-stoch-proc X n*)). *proj-stoch-proc X n* $-$‘$\{y\} \in$
sigma-sets (*space M*) ($\bigcup$ $i\in \{m.\ m\leq n\}$. $\{(X\ i -‘A) \cap$ (*space M*) $\mid$ *A*. *A*$\in$ *sets borel* $\}$)
proof
fix *y*
assume $y \in B \cap$ *range* (*proj-stoch-proc X n*)
hence $\exists x.\ y =$ *proj-stoch-proc X n x* **by** *auto*
from *this* **obtain** *x* **where** $y =$ *proj-stoch-proc X n x* **by** *auto*
have *proj-stoch-proc X n* $-$‘$\{$*proj-stoch-proc X n x*$\} \in$ *sets* (*stoch-proc-filt M X borel n*)
by (*rule proj-stoch-pre-borel*, *simp add*:*assms*)
also have ... = *sigma-sets* (*space M*) ($\bigcup$ $i\in \{m.\ m\leq n\}$. $\{(X\ i -‘A) \cap$ (*space M*) $\mid$ *A*. *A*$\in$ *sets borel* $\}$)
proof (*rule stoch-proc-filt-sets*)
fix *i*
assume $i\leq n$
show *random-variable borel* (*X i*) **using** *assms borel-adapt-stoch-proc-borel-measurable* **by** *blast*
qed
finally show *proj-stoch-proc X n* $-$‘$\{y\} \in$
sigma-sets (*space M*) ($\bigcup$ $i\in \{m.\ m\leq n\}$. $\{(X\ i -‘A) \cap$ (*space M*) $\mid$ *A*. *A*$\in$ *sets borel* $\}$)
using ‹$y =$ *proj-stoch-proc X n x*› **by** *simp*
qed
hence ($\bigcup$ $y\in$ ($B \cap$ *range* (*proj-stoch-proc X n*)). *proj-stoch-proc X n* $-$‘$\{y\}$) $\in$
sigma-sets (*space M*) ($\bigcup$ $i\in \{m.\ m\leq n\}$. $\{(X\ i -‘A) \cap$ (*space M*) $\mid$ *A*. *A*$\in$ *sets borel* $\}$)
proof (*rule sigma-set-union-count*)
have *finite* (*range* (*proj-stoch-proc X n*))
by (*simp add*: *assms proj-stoch-set-finite-range*)
thus *countable* ($B \cap$ *range* (*proj-stoch-proc X n*))
by (*simp add*: *countable-finite*)
qed
thus *proj-stoch-proc X n* $-$‘ $B \cap$ *space M* $\in$
sigma-sets (*space M*) ($\bigcup i\in\{m.\ m \leq n\}$. $\{X\ i -‘ A \cap$ *space M* $|A.\ A \in$ *sets borel*$\}$) **using** *eqB* **by** *simp*
qed
thus *?thesis* **by** *auto*
qed
hence *sigma-sets* (*space M*) $\{$*proj-stoch-proc X n* $-‘$ $B \cap$ *space M* $|B.\ B \in$ *sets* (*stream-space borel*)$\}$
$\subseteq$ *sigma-sets* (*space M*) ($\bigcup i\in\{m.\ m \leq n\}$. $\{X\ i -‘ A \cap$ *space M* $|A.\ A \in$ *sets borel*$\}$) **by** (*rule sigma-sets-mono*)
hence *sigma-sets* (*space M*) $\{$*proj-stoch-proc X n* $-‘$ $B \cap$ *space M* $|B.\ B \in$ *sets*

(*stream-space borel*)}
= *sigma-sets* (*space M*) (⋃ *i*∈{*m*. *m* ≤ *n*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets borel*}) **using** *l* **by** *simp*
thus *?thesis* **unfolding** *stoch-proc-filt-def fct-gen-subalgebra-def* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *stoch-proc-subalg-nat-filt*:
assumes *borel-adapt-stoch-proc nat-filtration X*
shows *subalgebra* (*nat-filtration n*) (*stoch-proc-filt M X borel n*) **unfolding** *subalgebra-def*
proof
show *space* (*stoch-proc-filt M X borel n*) = *space* (*nat-filtration n*)
by (*simp add*: *fct-gen-subalgebra-space nat-filtration-def stoch-proc-filt-space*)
show *sets* (*stoch-proc-filt M X borel n*) ⊆ *sets* (*nat-filtration n*)
proof −
have ∀ *i* ≤ *n*. (∀ *A*∈ *sets borel*. *X i* −‘*A* ∩ *space M* ∈ *sets* (*nat-filtration n*))
proof (*intro allI impI*)
fix *i*
assume *i* ≤ *n*
have *X i* ∈ *borel-measurable* (*nat-filtration n*)
by (*metis* ‹*i* ≤ *n*› *adapt-stoch-proc-def assms filtrationE2 measurable-from-subalg nat-discrete-filtration*)
show ∀ *A*∈*sets borel*. *X i* −‘ *A* ∩ *space M* ∈ *sets* (*nat-filtration n*)
proof
fix *A*::*'a set*
assume *A*∈ *sets borel*
thus *X i* −‘ *A* ∩ *space M* ∈ *sets* (*nat-filtration n*) **using** ‹*X i* ∈ *borel-measurable* (*nat-filtration n*)›
by (*metis bernoulli bernoulli-stream-space measurable-sets nat-filtration-space streams-UNIV*)
qed
qed
hence (⋃ *i*∈ {*m*. *m*≤ *n*}. {(*X i* −‘*A*) ∩ (*space M*) | *A*. *A*∈ *sets borel* }) ⊆ *sets* (*nat-filtration n*) **by** *auto*
hence *sigma-sets* (*space M*) (⋃ *i*∈ {*m*. *m*≤ *n*}. {(*X i* −‘*A*) ∩ (*space M*) | *A*. *A*∈ *sets borel* }) ⊆ *sets* (*nat-filtration n*)
by (*metis* (*no-types*, *lifting*) *bernoulli bernoulli-stream-space nat-filtration-space sets.sigma-sets-subset streams-UNIV*)
thus *?thesis* **using** *assms stoch-proc-filt-sets* **unfolding** *adapt-stoch-proc-def*
proof −
assume ∀ *t*. *X t* ∈ *borel-measurable* (*nat-filtration t*)
then have *f1*: ∀ *n m*. *X n* ∈ *borel-measurable m* ∨ ¬ *subalgebra m* (*nat-filtration n*)
by (*meson measurable-from-subalg*)
have ∀ *n*. *subalgebra M* (*nat-filtration n*)
by (*metis infinite-coin-toss-space.nat-filtration-subalgebra infinite-coin-toss-space-axioms*)
then show *?thesis*
using *f1* ‹⋀*n X N M*. (⋀*i*. *i* ≤ *n* ⟹ *X i* ∈ *M* →$_M$ *N*) ⟹ *sets* (*stoch-proc-filt*

M X N n) = *sigma-sets* (*space M*) (⋃ *i*∈{*m*. *m* ≤ *n*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets N*})› ‹*sigma-sets* (*space M*) (⋃ *i*∈{*m*. *m* ≤ *n*}. {*X i* −‘ *A* ∩ *space M* |*A*. *A* ∈ *sets borel*}) ⊆ *sets* (*nat-filtration n*)› **by** *blast*
qed
qed
qed

lemma (**in** *infinite-coin-toss-space*)
assumes *N* = *bernoulli-stream q*
and 0 ≤ *q*
and *q* ≤ 1
and 0 < *p*
and *p* < 1
and *filt-equiv nat-filtration M N*
shows *filt-equiv-sgt*: 0 < *q* **and** *filt-equiv-slt*: *q* < 1
proof −
have *space M* = *space N* **using** *assms filt-equiv-space* **by** *simp*
have *eqs*: {*w*∈ *space M*. (*snth w* 0)} = *pseudo-proj-True* (*Suc* 0) −‘{*True* ##*sconst True*}
proof
show {*w* ∈ *space M*. *w* !! 0} ⊆ *pseudo-proj-True* (*Suc* 0) −‘ {*True* ##*sconst True*}
proof
fix *w*
assume *w* ∈ {*w* ∈ *space M*. *w* !! 0}
hence *snth w* 0 **by** *simp*
hence *pseudo-proj-True* (*Suc* 0) *w* = *True*##*sconst True* **by** (*simp add*: *pseudo-proj-True-def*)
thus *w* ∈ *pseudo-proj-True* (*Suc* 0) −‘ {*True*##*sconst True*} **by** *simp*
qed
show *pseudo-proj-True* (*Suc* 0) −‘ {*True*##*sconst True*} ⊆ {*w* ∈ *space M*. *w* !! 0}
proof
fix *w*
assume *w* ∈ *pseudo-proj-True* (*Suc* 0) −‘ {*True*##*sconst True*}
hence *pseudo-proj-True* (*Suc* 0) *w* = *True*##*sconst True* **by** *simp*
hence *snth w* 0
by (*metis pseudo-proj-True-Suc-prefix stream-eq-Stream-iff*)
thus *w*∈ {*w* ∈ *space M*. *w* !! 0} **using** *bernoulli bernoulli-stream-space* **by** *simp*
qed
qed
hence *natset*: {*w*∈ *space M*. (*snth w* 0)} ∈ *sets* (*nat-filtration* (*Suc* 0))
proof −
have *pseudo-proj-True* (*Suc* 0) −‘ {*True*##*sconst True*} ∈ *sets* (*nat-filtration* (*Suc* 0))

```
    proof (rule nat-filtration-singleton)
     show pseudo-proj-True (Suc 0) (True##sconst True) = True## sconst True
unfolding pseudo-proj-True-def by simp
    qed
    thus ?thesis using eqs by simp
  qed
  have eqf: {w∈ space M. ¬(snth w 0)} = pseudo-proj-True (Suc 0) -‘{False
##sconst True}
  proof
     show {w ∈ space M. ¬(snth w 0)} ⊆ pseudo-proj-True (Suc 0) -‘ {False
##sconst True}
    proof
      fix w
      assume w ∈ {w ∈ space M. ¬(snth w 0)}
      hence ¬(snth w 0) by simp
      hence pseudo-proj-True (Suc 0) w = False ##sconst True
        by (simp add: pseudo-proj-True-def)
      thus w ∈ pseudo-proj-True (Suc 0) -‘ {False ## sconst True} by simp
    qed
    show pseudo-proj-True (Suc 0) -‘ {False ## sconst True} ⊆ {w ∈ space M.
¬(snth w 0)}
    proof
      fix w
      assume w ∈ pseudo-proj-True (Suc 0) -‘ {False##sconst True}
      hence pseudo-proj-True (Suc 0) w = False##sconst True by simp
      hence ¬(snth w 0)
        by (metis pseudo-proj-True-Suc-prefix  stream-eq-Stream-iff)
      thus w∈ {w ∈ space M. ¬(snth w 0)} using bernoulli bernoulli-stream-space
by simp
    qed
  qed
  hence natsetf: {w∈ space M. ¬(snth w 0)} ∈ sets (nat-filtration (Suc 0))
  proof -
    have pseudo-proj-True (Suc 0) -‘ {False##sconst True} ∈ sets (nat-filtration
(Suc 0))
    proof (rule nat-filtration-singleton)
     show pseudo-proj-True (Suc 0) (False##sconst True) = False##sconst True
unfolding pseudo-proj-True-def by simp
    qed
    thus ?thesis using eqf by simp
  qed

  show 0 < q
  proof (rule ccontr)
    assume ¬ 0 < q
    hence q = 0 using assms by simp
   hence emeasure N {w∈ space N. (snth w 0)} = q using bernoulli-stream-component-probability[of
N q]
        assms by blast
```

hence *emeasure N* {*w*∈ *space N*. (*snth w 0*)} = *0* **using** ‹*q* = *0*› **by** *simp*
hence *emeasure M* {*w*∈ *space M*. (*snth w 0*)} = *0* **using** *assms natset* **unfolding** *filt-equiv-def*
by (*simp add*: ‹*space M* = *space N*›)
moreover have *emeasure M* {*w*∈ *space M*. (*snth w 0*)} = *p* **using** *bernoulli-stream-component-probability*[*of M p*] *bernoulli*
p-lt-1 p-gt-0 **by** *blast*
ultimately show *False* **using** *assms* **by** *simp*
qed
show *q* < *1*
proof (*rule ccontr*)
assume ¬ *q* < *1*
hence *q* = *1* **using** *assms* **by** *simp*
hence *emeasure N* {*w*∈ *space N*. ¬(*snth w 0*)} = *1* −*q* **using** *bernoulli-stream-component-probability-compl*[*of N q*]
assms **by** *blast*
hence *emeasure N* {*w*∈ *space N*. ¬(*snth w 0*)} = *0* **using** ‹*q* = *1*› **by** *simp*
hence *emeasure M* {*w*∈ *space M*. ¬(*snth w 0*)} = *0* **using** *assms natsetf* **unfolding** *filt-equiv-def*
by (*simp add*: ‹*space M* = *space N*›)
moreover have *emeasure M* {*w*∈ *space M*. ¬(*snth w 0*)} = *1*−*p* **using** *bernoulli-stream-component-probability-compl*[*of M p*] *bernoulli*
p-lt-1 p-gt-0 **by** *blast*
ultimately show *False* **using** *assms* **by** *simp*
qed
qed

lemma *stoch-proc-filt-filt-equiv*:
assumes *filt-equiv F M N*
shows *stoch-proc-filt M f P n* = *stoch-proc-filt N f P n* **using** *assms filt-equiv-space filt-equiv-sets*
unfolding *stoch-proc-filt-def*
proof −
have *space N* = *space M*
by (*metis assms filt-equiv-space*)
then show *gen-subalgebra M* (*sigma-sets* (*space M*) (⋃*n*∈{*na*. *na* ≤ *n*}. {*f n* −‘ *C* ∩ *space M* |*C*. *C* ∈ *sets P*})) =
gen-subalgebra N (*sigma-sets* (*space N*) (⋃*n*∈{*na*. *na* ≤ *n*}. {*f n* −‘ *C* ∩ *space N* |*C*. *C* ∈ *sets P*}))
by (*simp add*: *gen-subalgebra-def*)
qed

lemma *filt-equiv-filt*:
assumes *filt-equiv F M N*
and *filtration M G*
shows *filtration N G* **unfolding** *filtration-def*
proof (*intro allI conjI impI*)
{
fix *t*

show *subalgebra N* (*G t*) **using** *assms* **unfolding** *filtration-def filt-equiv-def*
by (*metis sets-eq-imp-space-eq subalgebra-def*)
}
{
fix *s*::′*c*
fix *t*
assume *s* ≤ *t*
thus *subalgebra* (*G t*) (*G s*) **using** *assms* **unfolding** *filtration-def* **by** *simp*
}
qed

lemma *filt-equiv-borel-AE-eq-iff*:
fixes *f*::′*a*⇒ *real*
assumes *filt-equiv F M N*
and *f*∈ *borel-measurable* (*F t*)
and *g*∈ *borel-measurable* (*F t*)
and *prob-space N*
and *prob-space M*
shows (*AE w in M*. *f w* = *g w*) ⟷ (*AE w in N*. *f w* = *g w*)
proof −
{
assume *fst*: *AE w in M*. *f w* = *g w*
have *set0*: {*w*∈ *space M*. *f w* ≠ *g w*} ∈ *sets* (*F t*) ∧ *emeasure M* {*w*∈ *space M*. *f w* ≠ *g w*} = *0*
proof (*rule filtrated-prob-space.AE-borel-eq*, (*auto simp add*: *assms*))
show *filtrated-prob-space M F* **using** *assms* **unfolding** *filt-equiv-def*
by (*simp add*: *filtrated-prob-space-axioms.intro filtrated-prob-space-def*)
show *AE w in M*. *f w* = *g w* **using** *fst* **.**
qed
hence *emeasure N* {*w*∈ *space M*. *f w* ≠ *g w*} = *0* **using** *assms* **unfolding** *filt-equiv-def* **by** *auto*
moreover have {*w*∈ *space M*. *f w* ≠ *g w*} ∈ *sets N* **using** *set0 assms* **unfolding** *filt-equiv-def*
filtration-def subalgebra-def **by** *auto*
ultimately have *AE w in N*. *f w* = *g w*
proof −
have *space M* = *space N*
by (*metis assms*(*1*) *filt-equiv-space*)
then have ∀ *p*. *almost-everywhere N p* ∨ {*a* ∈ *space N*. ¬ *p a*} ≠ {*a* ∈ *space N*. *f a* ≠ *g a*}
using *AE-iff-measurable* ‹*emeasure N* {*w* ∈ *space M*. *f w* ≠ *g w*} = *0*› ‹{*w* ∈ *space M*. *f w* ≠ *g w*} ∈ *sets N*›
by *auto*
then show *?thesis*
by *metis*
qed
} **note** *a* = *this*
{

assume *scd*: *AE w in N*. *f w* = *g w*
have *space M* = *space N*
by (*metis assms*(*1*) *filt-equiv-space*)
have *set0*: {*w*∈ *space N*. *f w* ≠ *g w*} ∈ *sets* (*F t*) ∧ *emeasure N* {*w*∈ *space N*. *f w* ≠ *g w*} = *0*
proof (*rule filtrated-prob-space.AE-borel-eq*, (*auto simp add*: *assms*))
show *filtrated-prob-space N F* **using** *assms* **unfolding** *filt-equiv-def*
by (*metis* ‹*prob-space N*› *assms*(*1*) *filt-equiv-filtration filtrated-prob-space-axioms.intro filtrated-prob-space-def*)
show *AE w in N*. *f w* = *g w* **using** *scd* **.**
qed
hence *emeasure M* {*w*∈ *space M*. *f w* ≠ *g w*} = *0* **using** *assms* **unfolding** *filt-equiv-def*
by (*metis* (*full-types*) *assms*(*1*) *filt-equiv-space*)
moreover have {*w*∈ *space M*. *f w* ≠ *g w*} ∈ *sets M* **using** *set0 assms* **unfolding** *filt-equiv-def*
filtration-def subalgebra-def
by (*metis* (*mono-tags*) ‹*space M* = *space N*› *contra-subsetD*)
ultimately have *AE w in M*. *f w*= *g w*
proof −
have ∀ *p*. *almost-everywhere M p* ∨ {*a* ∈ *space M*. ¬ *p a*} ≠ {*a* ∈ *space M*. *f a* ≠ *g a*}
using *AE-iff-measurable* ‹*emeasure M* {*w* ∈ *space M*. *f w* ≠ *g w*} = *0*› ‹{*w* ∈ *space M*. *f w* ≠ *g w*} ∈ *sets M*›
by *auto*
then show *?thesis*
by *metis*
qed
}
thus *?thesis* **using** *a* **by** *auto*
qed

lemma (**in** *infinite-coin-toss-space*) *filt-equiv-triv-init*:
assumes *filt-equiv F M N*
and *init-triv-filt M G*
shows *init-triv-filt N G* **unfolding** *init-triv-filt-def*
proof
show *filtration N G* **using** *assms filt-equiv-filt*[*of F M N G*] **unfolding** *init-triv-filt-def*
by *simp*
show *sets* (*G* ⊥) = {{}, *space N*} **using** *assms filt-equiv-space*[*of F M N*] **unfolding** *init-triv-filt-def* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *fct-gen-subalgebra-meas-info*:
assumes ∀ *w*. *f* (*g w*) = *f w*
and *f* ∈ *space M* → *space N*
and *g* ∈ *space M* → *space M*

shows $g \in$ *measurable* (*fct-gen-subalgebra M N f*) (*fct-gen-subalgebra M N f*)
unfolding *measurable-def*
proof (*intro CollectI conjI*)
show $g \in$ *space* (*fct-gen-subalgebra M N f*) $\rightarrow$ *space* (*fct-gen-subalgebra M N f*)
using *assms*
by (*simp add*: *fct-gen-subalgebra-space*)
show $\forall y{\in}$*sets* (*fct-gen-subalgebra M N f*). $g -` y \cap$ *space* (*fct-gen-subalgebra M N f*) $\in$ *sets* (*fct-gen-subalgebra M N f*)
proof
fix *B*
assume *B*$\in$ *sets* (*fct-gen-subalgebra M N f*)
hence $B \in \{f -` B \cap$ *space M* $|B.\ B \in$ *sets N*$\}$ **using** *assms* **by** (*simp add*:*fct-gen-subalgebra-sigma-sets*)
from *this* **obtain** *C* **where** *C*$\in$ *sets N* **and** $B = f -`C \cap$ *space M* **by** *auto*
note *Cprops* = *this*
have $g -` B \cap$ *space* (*fct-gen-subalgebra M N f*) $= g -` B \cap$ *space M* **using** *assms*
by (*simp add*: *fct-gen-subalgebra-space*)
also have ... $= g -` (f -` C \cap$ *space M*$) \cap$ *space M* **using** *Cprops* **by** *simp*
also have ... $= g -` (f -` C)$ **using** *bernoulli bernoulli-stream-space* **by** *simp*
also have ... $= (f \circ g) -` C$ **by** *auto*
also have ... $= f -` C$
proof
show $(f \circ g) -` C \subseteq f -` C$
proof
fix *w*
assume $w \in (f \circ g) -` C$
hence $f\ (g\ w) \in C$ **by** *simp*
hence $f\ w \in C$ **using** *assms* **by** *simp*
thus $w{\in}\ f -`C$ **by** *simp*
qed
show $f -` C \subseteq (f \circ g) -` C$
proof
fix *w*
assume $w{\in}\ f -`C$
hence $f\ w \in C$ **by** *simp*
hence $f\ (g\ w) \in C$ **using** *assms* **by** *simp*
thus $w{\in}\ (f \circ g) -` C$ **by** *simp*
qed
qed
also have ... $\in$ *sets* (*fct-gen-subalgebra M N f*)
using *Cprops*(*2*) ‹*B* $\in$ *sets* (*fct-gen-subalgebra M N f*)› *bernoulli bernoulli-stream-space inf-top.right-neutral* **by** *auto*
finally show $g -` B \cap$ *space* (*fct-gen-subalgebra M N f*) $\in$ *sets* (*fct-gen-subalgebra M N f*) **.**
qed
qed

end
theory *Geometric-Random-Walk* **imports** *Infinite-Coin-Toss-Space*

begin

6 Geometric random walk

A geometric random walk is a stochastic process that can, at each time, move upwards or downwards, depending on the outcome of a coin toss.

fun (**in** *infinite-coin-toss-space*) *geom-rand-walk*:: *real* ⇒ *real* ⇒ *real* ⇒ (*nat* ⇒ *bool stream* ⇒ *real*) **where**
base: (*geom-rand-walk u d v*) *0* = (λ*w*. *v*)|
step: (*geom-rand-walk u d v*) (*Suc n*) = (λ*w*. ((λ*True* ⇒ *u* | *False* ⇒ *d*) (*snth w n*)) ∗ (*geom-rand-walk u d v*) *n w*)

locale *prob-grw* = *infinite-coin-toss-space* +
fixes *geom-proc*::*nat* ⇒ *bool stream* ⇒ *real* **and** *u*::*real* **and** *d*::*real* **and** *init*::*real*
assumes *geometric-process*:*geom-proc* = *geom-rand-walk u d init*

lemma (**in** *prob-grw*) *geom-rand-walk-borel-measurable*:
shows (*geom-proc n*) ∈ *borel-measurable M*
proof (*induct n*)
case (*Suc n*)
thus *geom-proc* (*Suc n*) ∈ *borel-measurable M*
proof −
have *geom-rand-walk u d init n* ∈ *borel-measurable M* **using** *Suc geometric-process* **by** *simp*
moreover have (λ*w*. ((λ*True* ⇒ *u* | *False* ⇒ *d*) (*snth w n*))) ∈ *borel-measurable M*
proof −
have (λ*w*. *snth w n*) ∈ *measurable M* (*measure-pmf* (*bernoulli-pmf p*)) **by** (*simp add*: *bernoulli measurable-snth-count-space*)
moreover have (λ*True* ⇒ *u* | *False* ⇒ *d*) ∈ *borel-measurable* (*measure-pmf* (*bernoulli-pmf p*)) **by** *simp*
ultimately show *?thesis* **by** (*simp add*: *measurable-comp*)
qed
ultimately show *?thesis* **by** (*simp add*:*borel-measurable-times geometric-process*)
qed
next
show *random-variable borel* (*geom-proc 0*) **using** *geometric-process* **by** *simp*
qed

lemma (**in** *prob-grw*) *geom-rand-walk-pseudo-proj-True*:
shows *geom-proc n* = *geom-proc n* ∘ *pseudo-proj-True n*

proof (*induct n*)
case (*Suc n*)
let *?tf* = (λ*True* ⇒ *u* | *False* ⇒ *d*)
{
fix *w*
have *geom-proc* (*Suc n*) *w* = *?tf* (*snth w n*) ∗ *geom-proc n w*
using *geom-rand-walk.simps*(*2*) *geometric-process* **by** *simp*
also have ... = *?tf* (*snth* (*pseudo-proj-True* (*Suc n*) *w*) *n*) ∗ *geom-proc n w*
by (*metis lessI pseudo-proj-True-stake stake-nth*)
also have ... = *?tf* (*snth* (*pseudo-proj-True* (*Suc n*) *w*) *n*) ∗ *geom-proc n* (*pseudo-proj-True n w*)
using *Suc geometric-process* **by** (*metis comp-apply*)
also have ... = *?tf* (*snth* (*pseudo-proj-True* (*Suc n*) *w*) *n*) ∗ *geom-proc n* (*pseudo-proj-True* (*Suc n*) *w*)
using *geometric-process* **by** (*metis Suc.hyps comp-apply pseudo-proj-True-proj-Suc*)
also have ... = *geom-proc* (*Suc n*) (*pseudo-proj-True* (*Suc n*) *w*) **using** *geometric-process* **by** *simp*
finally have *geom-proc* (*Suc n*) *w* = *geom-proc* (*Suc n*) (*pseudo-proj-True* (*Suc n*) *w*) **.**
}
thus *geom-proc* (*Suc n*) = *geom-proc* (*Suc n*) ∘ (*pseudo-proj-True* (*Suc n*)) **using** *geometric-process* **by** *auto*
next
show *geom-proc 0* = *geom-proc 0* ∘ *pseudo-proj-True 0* **using** *geometric-process* **by** *auto*
qed

lemma (**in** *prob-grw*) *geom-rand-walk-pseudo-proj-False*:
shows *geom-proc n* = *geom-proc n* ∘ *pseudo-proj-False n*
proof (*induct n*)
case (*Suc n*)
let *?tf* = (λ*True* ⇒ *u* | *False* ⇒ *d*)
{
fix *w*
have *geom-proc* (*Suc n*) *w* = *?tf* (*snth w n*) ∗ *geom-proc n w*
using *geom-rand-walk.simps*(*2*) *geometric-process* **by** *simp*
also have ... = *?tf* (*snth* (*pseudo-proj-False* (*Suc n*) *w*) *n*) ∗ *geom-proc n w*
by (*metis lessI pseudo-proj-False-stake stake-nth*)
also have ... = *?tf* (*snth* (*pseudo-proj-False* (*Suc n*) *w*) *n*) ∗ *geom-proc n* (*pseudo-proj-False n w*)
using *Suc geometric-process* **by** (*metis comp-apply*)
also have ... = *?tf* (*snth* (*pseudo-proj-False* (*Suc n*) *w*) *n*) ∗ *geom-proc n* (*pseudo-proj-True n* (*pseudo-proj-False n w*))
using *geometric-process* **by** (*metis geom-rand-walk-pseudo-proj-True o-apply*)
also have ... = *?tf* (*snth* (*pseudo-proj-False* (*Suc n*) *w*) *n*) ∗ *geom-proc n* (*pseudo-proj-True n* (*pseudo-proj-False* (*Suc n*) *w*))
unfolding *pseudo-proj-True-def pseudo-proj-False-def*
by (*metis pseudo-proj-False-def pseudo-proj-False-stake pseudo-proj-True-def pseudo-proj-True-proj-Suc*)

also have ... = *?tf* (*snth* (*pseudo-proj-False* (*Suc n*) *w*) *n*) ∗ *geom-proc n* (*pseudo-proj-False* (*Suc n*) *w*)
using *geometric-process* **by** (*metis geom-rand-walk-pseudo-proj-True o-apply*)
also have ... = *geom-proc* (*Suc n*) (*pseudo-proj-False* (*Suc n*) *w*) **using** *geometric-process* **by** *simp*
finally have *geom-proc* (*Suc n*) *w* = *geom-proc* (*Suc n*) (*pseudo-proj-False* (*Suc n*) *w*) **.**
}
thus *geom-proc* (*Suc n*) = *geom-proc* (*Suc n*) ∘ (*pseudo-proj-False* (*Suc n*))
using *geometric-process* **by** *auto*
next
show *geom-proc 0* = *geom-proc 0* ∘ *pseudo-proj-False 0* **using** *geometric-process* **by** *auto*
qed

lemma (**in** *prob-grw*) *geom-rand-walk-borel-adapted*:
shows *borel-adapt-stoch-proc nat-filtration geom-proc*
unfolding *adapt-stoch-proc-def*
proof (*auto simp add*:*nat-discrete-filtration*)
fix *n*
show *geom-proc n* ∈ *borel-measurable* (*nat-filtration n*)
proof −
have *geom-proc n* ∈ *borel-measurable* (*nat-filtration n*)
proof (*rule nat-filtration-comp-measurable*)
show *geom-proc n* ∈ *borel-measurable M*
by (*simp add*: *geom-rand-walk-borel-measurable*)
show *geom-proc n* ∘ *pseudo-proj-True n* = *geom-proc n*
using *geom-rand-walk-pseudo-proj-True* **by** *simp*
qed
then show *?thesis* **by** *simp*
qed
qed

lemma (**in** *prob-grw*) *grw-succ-img*:
assumes (*geom-proc n*) −‘ {*x*} ≠ {}
shows (*geom-proc* (*Suc n*)) ‘ ((*geom-proc n*) −‘ {*x*}) = {*u*∗*x*, *d*∗*x*}
proof
have ∃ *w*. *geom-proc n w* = *x* **using** *assms* **by** *auto*
from *this* **obtain** *w* **where** *geom-proc n w* = *x* **by** *auto*
let *?wT* = *spick w n True*
let *?wF* = *spick w n False*
have *bel*: (*?wT* ∈ (*geom-proc n*) −‘ {*x*}) ∧ (*?wF* ∈ (*geom-proc n*) −‘ {*x*})
by (*metis* ‹*geom-proc n w* = *x*› *geom-rand-walk-pseudo-proj-True o-def pseudo-proj-True-stake-image spickI vimage-singleton-eq*)
have *geom-proc* (*Suc n*) *?wT* = *u*∗*x*
proof −

```
    have x = geom-rand-walk u d init n (spick w n True)
      by (metis ‹geom-proc n w = x› comp-apply geom-rand-walk-pseudo-proj-True
geometric-process pseudo-proj-True-stake-image spickI)
    then show ?thesis
      by (simp add: geometric-process spickI)
  qed
  moreover have geom-proc (Suc n) ?wF = d∗x
  proof −
    have x = geom-rand-walk u d init n (spick w n False)
      by (metis ‹geom-proc n w = x› comp-apply geom-rand-walk-pseudo-proj-True
geometric-process pseudo-proj-True-stake-image spickI)
    then show ?thesis
      by (simp add: geometric-process spickI)
  qed
  ultimately show {u∗x, d∗x} ⊆ (geom-proc (Suc n)) ‘ ((geom-proc n) −‘ {x})
using bel
    by (metis empty-subsetI insert-subset rev-image-eqI)
  have ∀ w ∈ (geom-proc n) −‘ {x}. geom-proc (Suc n) w ∈ {u∗x, d∗x}
  proof
    fix w
    assume w ∈ (geom-proc n) −‘ {x}
    have dis: ((snth w (Suc n)) = True) ∨ (snth w (Suc n) = False) by simp
    show geom-proc (Suc n) w ∈ {u∗x, d∗x}
    proof −
      have geom-proc n w = x
        by (metis ‹w ∈ geom-proc n −‘ {x}› vimage-singleton-eq)
      then have geom-rand-walk u d init n w = x
        using geometric-process by blast
      then show ?thesis
        by (simp add: case-bool-if geometric-process)
    qed
  qed
  thus (geom-proc (Suc n)) ‘ ((geom-proc n) −‘ {x}) ⊆ {u∗x, d∗x} by auto
qed

lemma (in prob-grw) geom-rand-walk-strictly-positive:
  assumes 0 < init
  and 0 < d
  and d < u
  shows ∀ n w. 0 < geom-proc n w
proof (intro allI)
  fix n
  fix w
  show 0 < geom-proc n w
  proof (induct n)
  case 0 thus ?case using assms geometric-process by simp
  next
  case (Suc n)
    thus ?case
```

proof (*cases snth w n*)
case *True*
hence *geom-proc* (*Suc n*) *w* = *u* ∗ *geom-proc n w* **using** *geom-rand-walk.simps geometric-process* **by** *simp*
also have *... > 0* **using** *Suc assms* **by** *simp*
finally show *?thesis* **.**
next
case *False*
hence *geom-proc* (*Suc n*) *w* = *d* ∗ *geom-proc n w* **using** *geom-rand-walk.simps geometric-process* **by** *simp*
also have *... > 0* **using** *Suc assms* **by** *simp*
finally show *?thesis* **.**
qed
qed
qed

lemma (**in** *prob-grw*) *geom-rand-walk-diff-induct*:
shows ⋀*w*. (*geom-proc* (*Suc n*) (*spick w n True*) − *geom-proc* (*Suc n*) (*spick w n False*)) = (*geom-proc n w* ∗ (*u* − *d*))
proof −
fix *w*
have *geom-proc* (*Suc n*) (*spick w n True*) = *u* ∗ *geom-proc n w*
proof −
have *snth* (*spick w n True*) *n* = *True* **by** (*simp add*: *spickI*)
hence (λ*w*. (*case w* !! *n of True* ⇒ *u* | *False* ⇒ *d*)) (*spick w n True*) = *u* **by** *simp*
thus *?thesis* **using** *geometric-process geom-rand-walk.simps*(*2*)[*of u d init n*]
by (*metis comp-apply geom-rand-walk-pseudo-proj-True pseudo-proj-True-def spickI*)
qed
moreover have *geom-proc* (*Suc n*) (*spick w n False*) = *d* ∗ *geom-proc n w*
proof −
have *snth* (*spick w n False*) *n* = *False* **by** (*simp add*: *spickI*)
hence (λ*w*. (*case w* !! *n of True* ⇒ *u* | *False* ⇒ *d*)) (*spick w n False*) = *d* **by** *simp*
thus *?thesis* **using** *geometric-process geom-rand-walk.simps*(*2*)[*of u d init n*]
by (*metis comp-apply geom-rand-walk-pseudo-proj-True pseudo-proj-True-def spickI*)
qed
ultimately show (*geom-proc* (*Suc n*) (*spick w n True*) − *geom-proc* (*Suc n*) (*spick w n False*)) = (*geom-proc n w* ∗ (*u* − *d*))
by (*simp add*:*field-simps*)
qed

end

7 Fair Prices

This section contains the formalization of financial notions, such as markets, price processes, portfolios, arbitrages, fair prices, etc. It also defines risk-neutral probability spaces, and proves the main result about the fair price of a derivative in a risk-neutral probability space, namely that this fair price is equal to the expectation of the discounted value of the derivative's payoff.

theory *Fair-Price* **imports** *Filtration Martingale Geometric-Random-Walk*
begin

7.1 Preliminary results

7.1.1 On the almost everywhere filter

lemma *AE-eq-trans*[*trans*]:
assumes *AE x in M. A x = B x*
and *AE x in M. B x = C x*
shows *AE x in M. A x = C x*
using *assms* **by** *auto*

abbreviation *AEeq* **where** *AEeq M X Y* ≡ *AE w in M. X w = Y w*

lemma *AE-add*:
assumes *AE w in M. f w = g w*
and *AE w in M. f′ w = g′ w*
shows *AE w in M. f w + f′ w = g w + g′ w* **using** *assms* **by** *auto*

lemma *AE-sum*:
assumes *finite I*
and ∀ *i*∈*I. AE w in M. f i w = g i w*
shows *AE w in M.* (∑ *i*∈ *I. f i w*) = (∑ *i*∈ *I. g i w*) **using** *assms*(*1*) *subset-refl*[*of I*]
proof (*induct rule*: *finite-subset-induct*)
case *empty*
then show *?case* **by** *simp*
next
case (*insert a F*)
have *AEeq M* (*f a*) (*g a*) **using** *assms*(*2*) *insert.hyps*(*2*) **by** *auto*
have *AE w in M.* (∑ *i*∈ *insert a F. f i w*) = *f a w* + (∑ *i*∈ *F. f i w*)
by (*simp add*: *insert.hyps*(*1*) *insert.hyps*(*3*))
also have *AE w in M. f a w* + (∑ *i*∈ *F. f i w*) = *g a w* + (∑ *i*∈ *F. f i w*)
using ‹*AEeq M* (*f a*) (*g a*)› **by** *auto*
also have *AE w in M. g a w* + (∑ *i*∈ *F. f i w*) = *g a w* + (∑ *i*∈ *F. g i w*)
using *insert.hyps*(*4*) **by** *auto*
also have *AE w in M. g a w* + (∑ *i*∈ *F. g i w*) = (∑ *i*∈ *insert a F. g i w*)

by (*simp add*: *insert.hyps*(*1*) *insert.hyps*(*3*))
finally show *?case* **by** *auto*
qed

lemma *AE-eq-cst*:
assumes *AE w in M*. (λ*w*. *c*) *w* = (λ*w*. *d*) *w*
and *emeasure M* (*space M*) ≠ *0*
shows *c* = *d*
proof (*rule ccontr*)
assume *c* ≠ *d*
from ‹*AE w in M*. (λ*w*. *c*) *w* = (λ*w*. *d*) *w*› **obtain** *N* **where** *Nprops*: {*w*∈ *space M*. ¬(λ*w*. *c*) *w* = (λ*w*. *d*) *w*} ⊆ *N N*∈ *sets M emeasure M N* = *0*
by (*force elim*:*AE-E*)
have ∀ *w*∈ *space M*. (λ*w*. *c*) *w* ≠ (λ*w*. *d*) *w* **using** ‹*c*≠ *d*› **by** *simp*
hence {*w*∈ *space M*. (λ*w*. *c*) *w* ≠ (λ*w*. *d*) *w*} = *space M* **by** *auto*
hence *space M*⊆ *N* **using** *Nprops* **by** *auto*
thus *False* **using** ‹*emeasure M N* = *0*› *assms*
by (*meson Nprops*(*2*) ‹*emeasure M* (*space M*) ≠ *0*› ‹*emeasure M N* = *0*› ‹*space M* ⊆ *N*› *emeasure-eq-0*)
qed

7.1.2 On conditional expectations

lemma (**in** *prob-space*) *subalgebra-sigma-finite*:
assumes *subalgebra M N*
shows *sigma-finite-subalgebra M N* **unfolding** *sigma-finite-subalgebra-def* **by** (*simp add*: *assms prob-space-axioms prob-space-imp-sigma-finite prob-space-restr-to-subalg*)

lemma (**in** *prob-space*) *trivial-subalg-cond-expect-AE*:
assumes *subalgebra M N*
and *sets N* = {{}, *space M*}
and *integrable M f*
shows *AE x in M*. *real-cond-exp M N f x* = (λ*x*. *expectation f*) *x*
proof (*intro sigma-finite-subalgebra.real-cond-exp-charact*)
show *sigma-finite-subalgebra M N* **by** (*simp add*: *assms*(*1*) *subalgebra-sigma-finite*)
show *integrable M f* **using** *assms* **by** *simp*
show *integrable M* (λ*x*. *expectation f*) **by** *auto*
show (λ*x*. *expectation f*) ∈ *borel-measurable N* **by** *simp*
show ⋀*A*. *A* ∈ *sets N* ⟹ *set-lebesgue-integral M A f* = (∫ *x*∈*A*. *expectation f*∂*M*)
proof −
fix *A*
assume *A* ∈ *sets N*
show *set-lebesgue-integral M A f* = (∫ *x*∈*A*. *expectation f*∂*M*)
proof (*cases A* = {})
case *True*

thus *?thesis* **by** (*simp add*: *set-lebesgue-integral-def*)
next
case *False*
hence $A = space\ M$ **using** *assms* ‹$A \in sets\ N$› **by** *auto*
have *set-lebesgue-integral M A f* = *expectation f* **using** ‹$A = space\ M$›
by (*metis* (*mono-tags*, *lifting*) *Bochner-Integration.integral-cong indicator-simps*(*1*)
scaleR-one set-lebesgue-integral-def)
also have ... = $(\int x \in A.\ expectation\ f \partial M)$ **using** ‹$A = space\ M$›
by (*auto simp add*:*prob-space set-lebesgue-integral-def*)
finally show *?thesis* **.**
qed
qed
qed

lemma (**in** *prob-space*) *triv-subalg-borel-eq*:
assumes *subalgebra M F*
and *sets F* = {{}, *space M*}
and *AE x in M*. *f x* = (*c*::*'b*::{*t2-space*})
and *f*∈ *borel-measurable F*
shows ∀ *x*∈ *space M*. *f x* = *c*
proof
fix *x*
assume *x*∈ *space M*
have *space M* = *space F* **using** *assms* **by** (*simp add*:*subalgebra-def*)
hence *x*∈ *space F* **using** ‹*x*∈ *space M*› **by** *simp*
have *space M* ≠ {} **by** (*simp add*:*not-empty*)
hence ∃ *d*. ∀ *y*∈ *space F*. *f y* = *d* **by** (*metis assms*(*1*) *assms*(*2*) *assms*(*4*) *subalgebra-def triv-measurable-cst*)
from *this* **obtain** *d* **where** ∀ *y* ∈*space F*. *f y* = *d* **by** *auto*
hence *f x* = *d* **using** ‹*x*∈ *space F*› **by** *simp*
also have ... = *c*
proof (*rule ccontr*)
assume *d*≠ *c*
from ‹*AE x in M*. *f x*= *c*› **obtain** *N* **where** *Nprops*: {*x*∈ *space M*. ¬*f x* = *c*} ⊆ *N N*∈ *sets M emeasure M N* = *0*
by (*force elim*:*AE-E*)
have *space M* ⊆ {*x*∈ *space M*. ¬*f x* = *c*} **using** ‹∀ *y* ∈*space F*. *f y* = *d*› ‹*space M* = *space F*› ‹*d*≠ *c*› **by** *auto*
hence *space M*⊆ *N* **using** *Nprops* **by** *auto*
thus *False* **using** ‹*emeasure M N* = *0*› *emeasure-space-1* *Nprops*(*2*) *emeasure-mono* **by** *fastforce*
qed
finally show *f x* = *c* **.**
qed

lemma (**in** *prob-space*) *trivial-subalg-cond-expect-eq*:

assumes *subalgebra M N*
and *sets N* = {{}, *space M*}
and *integrable M f*
shows ∀ *x*∈ *space M*. *real-cond-exp M N f x* = *expectation f*
proof (*rule triv-subalg-borel-eq*)
show *subalgebra M N sets N* = {{}, *space M*} **using** *assms* **by** *auto*
show *real-cond-exp M N f* ∈ *borel-measurable N* **by** *simp*
show *AE x in M*. *real-cond-exp M N f x* = *expectation f*
by (*rule trivial-subalg-cond-expect-AE*, (*auto simp add*:*assms*))
qed

lemma (**in** *sigma-finite-subalgebra*) *real-cond-exp-cong*′:
assumes ∀ *w* ∈ *space M*. *f w* = *g w*
and *f*∈ *borel-measurable M*
shows *AE w in M*. *real-cond-exp M F f w* = *real-cond-exp M F g w*
proof (*rule real-cond-exp-cong*)
show *AE w in M*. *f w* = *g w* **using** *assms* **by** *simp*
show *f*∈ *borel-measurable M* **using** *assms* **by** *simp*
show *g*∈ *borel-measurable M* **using** *assms* **by** (*metis measurable-cong*)
qed

lemma (**in** *sigma-finite-subalgebra*) *real-cond-exp-bsum* :
fixes *f*::′*b* ⇒ ′*a* ⇒ *real*
assumes [*measurable*]: ⋀*i*. *i*∈*I* ⟹ *integrable M* (*f i*)
shows *AE x in M*. *real-cond-exp M F* (λ*x*. ∑ *i*∈*I*. *f i x*) *x* = (∑ *i*∈*I*. *real-cond-exp M F* (*f i*) *x*)
proof (*rule real-cond-exp-charact*)
fix *A* **assume** [*measurable*]: *A* ∈ *sets F*
then have *A-meas* [*measurable*]: *A* ∈ *sets M* **by** (*meson subsetD subalg subalgebra-def*)

have ∗: ⋀*i*. *i* ∈ *I* ⟹ *integrable M* (λ*x*. *indicator A x* ∗ *f i x*)
using *integrable-mult-indicator*[*OF* ‹*A* ∈ *sets M*› *assms*(*1*)] **by** *auto*
have ∗∗: ⋀*i*. *i* ∈ *I* ⟹ *integrable M* (λ*x*. *indicator A x* ∗ *real-cond-exp M F* (*f i*) *x*)
using *integrable-mult-indicator*[*OF* ‹*A* ∈ *sets M*› *real-cond-exp-int*(*1*)[*OF assms*(*1*)]]
by *auto*
have *inti*: ⋀*i*. *i* ∈ *I* ⟹(∫ *x*. *indicator A x* ∗ *f i x* ∂*M*) = (∫ *x*. *indicator A x* ∗ *real-cond-exp M F* (*f i*) *x* ∂*M*) **using**
real-cond-exp-intg(*2*)[*symmetric*,*of indicator A*]
using ∗ ‹*A* ∈ *sets F*› *assms borel-measurable-indicator* **by** *blast*
have (∫ *x*∈*A*. (∑ *i*∈*I*. *f i x*)∂*M*) = (∫ *x*. (∑ *i*∈*I*. *indicator A x* ∗ *f i x*)∂*M*)
by (*simp add*: *sum-distrib-left set-lebesgue-integral-def*)
also have ... = (∑ *i*∈*I*. (∫ *x*. *indicator A x* ∗ *f i x* ∂*M*)) **using** *Bochner-Integration.integral-sum*[*of I M* λ*i x*. *indicator A x* ∗ *f i x*] ∗
by *simp*
also have ... = (∑ *i*∈*I*. (∫ *x*. *indicator A x* ∗ *real-cond-exp M F* (*f i*) *x* ∂*M*))

using *inti* **by** *auto*
also have ... = $(\int x.\ (\sum i{\in}I.\ \mathit{indicator}\ A\ x * \mathit{real\text{-}cond\text{-}exp}\ M\ F\ (f\ i)\ x)\partial M)$
by (*rule Bochner-Integration.integral-sum*[*symmetric*], *simp add*: **)
also have ... = $(\int x{\in}A.\ (\sum i{\in}I.\ \mathit{real\text{-}cond\text{-}exp}\ M\ F\ (f\ i)\ x)\partial M)$
by (*simp add*: *sum-distrib-left set-lebesgue-integral-def*)
finally show $(\int x{\in}A.\ (\sum i{\in}I.\ f\ i\ x)\partial M) = (\int x{\in}A.\ (\sum i{\in}I.\ \mathit{real\text{-}cond\text{-}exp}\ M\ F\ (f\ i)\ x)\partial M)$ **by** *auto*
qed (*auto simp add*: *assms real-cond-exp-int*(*1*)[*OF assms*(*1*)])

7.2 Financial formalizations

7.2.1 Markets

definition *stk-strict-subs*::*'c set* ⇒ *bool* **where**
stk-strict-subs S ⟷ *S* ≠ *UNIV*

typedef (*'a*,*'c*) *discrete-market* = {(*s*::(*'c set*), *a*::*'c* ⇒ (*nat* ⇒ *'a* ⇒ *real*)). *stk-strict-subs s*} **unfolding** *stk-strict-subs-def* **by** *fastforce*

definition *prices* **where**
prices Mkt = (*snd* (*Rep-discrete-market Mkt*))

definition *assets* **where**

assets Mkt = *UNIV*

definition *stocks* **where**
stocks Mkt = (*fst* (*Rep-discrete-market Mkt*))

definition *discrete-market-of*
where
discrete-market-of S A =
Abs-discrete-market (*if* (*stk-strict-subs S*) *then S else* {}, *A*)

lemma *prices-of*:
shows *prices* (*discrete-market-of S A*) = *A*
proof −
have *stk-strict-subs* (*if* (*stk-strict-subs S*) *then S else* {})
proof (*cases stk-strict-subs S*)
case *True* **thus** *?thesis* **by** *simp*
next
case *False* **thus** *?thesis* **unfolding** *stk-strict-subs-def* **by** *simp*
qed
hence *fct*: ((*if* (*stk-strict-subs S*) *then S else* {}), *A*) ∈ {(*s*, *a*). *stk-strict-subs s*} **by** *simp*
have *discrete-market-of S A* = *Abs-discrete-market* (*if* (*stk-strict-subs S*) *then S else* {}, *A*) **unfolding** *discrete-market-of-def* **by** *simp*
hence *Rep-discrete-market* (*discrete-market-of S A*) = (*if* (*stk-strict-subs S*) *then S else* {},*A*)

using *Abs-discrete-market-inverse*[*of* (*if* (*stk-strict-subs S*) *then S else* {}, *A*)] *fct* **by** *simp*
thus *?thesis* **unfolding** *prices-def* **by** *simp*
qed

lemma *stocks-of*:
assumes *UNIV* ≠ *S*
shows *stocks* (*discrete-market-of S A*) = *S*
proof −
have *stk-strict-subs S* **using** *assms* **unfolding** *stk-strict-subs-def* **by** *simp*
hence *fct*: ((*if* (*stk-strict-subs S*) *then S else* {}), *A*) ∈ {(*s*, *a*). *stk-strict-subs s*} **by** *simp*
have *discrete-market-of S A* = *Abs-discrete-market* (*if* (*stk-strict-subs S*) *then S else* {}, *A*) **unfolding** *discrete-market-of-def* **by** *simp*
hence *Rep-discrete-market* (*discrete-market-of S A*) = (*if* (*stk-strict-subs S*) *then S else* {},*A*)
using *Abs-discrete-market-inverse*[*of* (*if* (*stk-strict-subs S*) *then S else* {}, *A*)] *fct* **by** *simp*
thus *?thesis* **unfolding** *stocks-def* **using** ‹*stk-strict-subs S*› **by** *simp*
qed

lemma *mkt-stocks-assets*:
shows *stk-strict-subs* (*stocks Mkt*) **unfolding** *stocks-def prices-def*
by (*metis Rep-discrete-market mem-Collect-eq split-beta'*)

7.2.2 Quantity processes and portfolios

These are functions that assign quantities to assets; each quantity is a stochastic process. Basic operations are defined on these processes.

Basic operations **definition** *qty-empty* **where**
qty-empty = (λ (*x*::'*a*) (*n*::*nat*) *w*. *0*::*real*)

definition *qty-single* **where**
qty-single asset qt-proc = (*qty-empty*(*asset* := *qt-proc*))

definition *qty-sum*::('*b* ⇒ *nat* ⇒ '*a* ⇒ *real*) ⇒ ('*b* ⇒ *nat* ⇒ '*a* ⇒ *real*) ⇒ ('*b* ⇒ *nat* ⇒ '*a* ⇒ *real*) **where**
qty-sum pf1 pf2 = (λ*x n w*. *pf1 x n w* + *pf2 x n w*)

definition *qty-mult-comp*::('*b* ⇒ *nat* ⇒ '*a* ⇒ *real*) ⇒ (*nat* ⇒ '*a* ⇒ *real*) ⇒ ('*b* ⇒ *nat* ⇒ '*a* ⇒ *real*) **where**
qty-mult-comp pf1 qty = (λ*x n w*. (*pf1 x n w*) ∗ (*qty n w*))

definition *qty-rem-comp*::('*b* ⇒ *nat* ⇒ '*a* ⇒ *real*) ⇒ '*b* ⇒ ('*b* ⇒ *nat* ⇒ '*a* ⇒ *real*) **where**
qty-rem-comp pf1 x = *pf1*(*x*:=(λ*n w*. *0*))

definition *qty-replace-comp* **where**

qty-replace-comp pf1 x pf2 = qty-sum (qty-rem-comp pf1 x) (qty-mult-comp pf2 (pf1 x))

Support sets If p x n w is different from 0, this means that this quantity is held on interval]n-1, n].

definition *support-set*::(*'b ⇒ nat ⇒ 'a ⇒ real*) ⇒ *'b set* **where**
support-set p = {x. ∃ n w. p x n w ≠ 0}

lemma *qty-empty-support-set*:
shows *support-set qty-empty = {}* **unfolding** *support-set-def qty-empty-def* **by** *simp*
lemma *sum-support-set*:
shows *support-set (qty-sum pf1 pf2) ⊆ (support-set pf1) ∪ (support-set pf2)*
proof (*intro subsetI*, *rule ccontr*)
fix *x*
assume *x∈ support-set (qty-sum pf1 pf2)* **and** *x ∉ support-set pf1 ∪ support-set pf2* **note** *xprops = this*
hence *∃ n w. (qty-sum pf1 pf2) x n w ≠ 0* **by** (*simp add*: *support-set-def*)
from *this* **obtain** *n w* **where** *(qty-sum pf1 pf2) x n w ≠ 0* **by** *auto* **note** *nwprops = this*
have *pf1 x n w = 0 pf2 x n w = 0* **using** *xprops* **by** (*auto simp add*:*support-set-def*)
hence *(qty-sum pf1 pf2) x n w = 0* **unfolding** *qty-sum-def* **by** *simp*
thus *False* **using** *nwprops* **by** *simp*
qed

lemma *mult-comp-support-set*:
shows *support-set (qty-mult-comp pf1 qty) ⊆ (support-set pf1)*
proof (*intro subsetI*, *rule ccontr*)
fix *x*
assume *x∈ support-set (qty-mult-comp pf1 qty)* **and** *x ∉ support-set pf1* **note** *xprops = this*
hence *∃ n w. (qty-mult-comp pf1 qty) x n w ≠ 0* **by** (*simp add*: *support-set-def*)
from *this* **obtain** *n w* **where** *qty-mult-comp pf1 qty x n w ≠ 0* **by** *auto* **note** *nwprops = this*
have *pf1 x n w = 0* **using** *xprops* **by** (*simp add*:*support-set-def*)
hence *(qty-mult-comp pf1 qty) x n w = 0* **unfolding** *qty-mult-comp-def* **by** *simp*
thus *False* **using** *nwprops* **by** *simp*
qed

lemma *remove-comp-support-set*:
shows *support-set (qty-rem-comp pf1 x) ⊆ ((support-set pf1) − {x})*
proof (*intro subsetI*, *rule ccontr*)
fix *y*
assume *y∈ support-set (qty-rem-comp pf1 x)* **and** *y ∉ support-set pf1 − {x}* **note** *xprops = this*
hence *y∉ support-set pf1 ∨ y = x* **by** *simp*
have *∃ n w. (qty-rem-comp pf1 x) y n w ≠ 0* **using** *xprops* **by** (*simp add*: *support-set-def*)

from *this* **obtain** *n w* **where** (*qty-rem-comp pf1 x*) *y n w* ≠ *0* **by** *auto* **note** *nwprops* = *this*
show *False*
proof (*cases y*∉ *support-set pf1*)
case *True*
hence *pf1 y n w = 0* **using** *xprops* **by** (*simp add*:*support-set-def*)
hence (*qty-rem-comp pf1 x*) *x n w = 0* **unfolding** *qty-rem-comp-def* **by** *simp*
thus *?thesis* **using** *nwprops* **by** (*metis* ‹*pf1 y n w = 0*› *fun-upd-apply qty-rem-comp-def*)
next
case *False*
hence *y = x* **using** ‹*y*∉ *support-set pf1* ∨ *y = x*› **by** *simp*
hence (*qty-rem-comp pf1 x*) *x n w = 0* **unfolding** *qty-rem-comp-def* **by** *simp*
thus *?thesis* **using** *nwprops* **by** (*simp add*: ‹*y = x*›)
qed
qed

lemma *replace-comp-support-set*:
shows *support-set* (*qty-replace-comp pf1 x pf2*) ⊆ (*support-set pf1* − {*x*}) ∪ *support-set pf2*
proof −
have *support-set* (*qty-replace-comp pf1 x pf2*) ⊆ *support-set* (*qty-rem-comp pf1 x*) ∪ *support-set* (*qty-mult-comp pf2* (*pf1 x*))
unfolding *qty-replace-comp-def* **by** (*simp add*:*sum-support-set*)
also have ... ⊆ (*support-set pf1* − {*x*}) ∪ *support-set pf2* **using** *remove-comp-support-set mult-comp-support-set*
by (*metis sup.mono*)
finally show *?thesis* **.**
qed

lemma *single-comp-support*:
shows *support-set* (*qty-single asset qty*) ⊆ {*asset*}
proof
fix *x*
assume *x*∈ *support-set* (*qty-single asset qty*)
show *x*∈ {*asset*}
proof (*rule ccontr*)
assume *x*∉ {*asset*}
hence *x*≠ *asset* **by** *auto*
have ∃ *n w. qty-single asset qty x n w* ≠ *0* **using** ‹*x*∈ *support-set* (*qty-single asset qty*)›
by (*simp add*:*support-set-def*)
from *this* **obtain** *n w* **where** *qty-single asset qty x n w* ≠ *0* **by** *auto*
thus *False* **using** ‹*x*≠*asset*› **by** (*simp add*: *qty-single-def qty-empty-def*)
qed
qed

lemma *single-comp-nz-support*:
assumes ∃ *n w. qty n w*≠ *0*
shows *support-set* (*qty-single asset qty*) = {*asset*}

proof
show *support-set* (*qty-single asset qty*) ⊆ {*asset*} **by** (*simp add*: *single-comp-support*)
have *asset*∈ *support-set* (*qty-single asset qty*) **using** *assms* **unfolding** *support-set-def qty-single-def* **by** *simp*
thus {*asset*} ⊆ *support-set* (*qty-single asset qty*) **by** *auto*
qed

Portfolios **definition** *portfolio* **where**
portfolio p ⟷ *finite* (*support-set p*)

definition *stock-portfolio* :: ('*a*, '*b*) *discrete-market* ⇒ ('*b* ⇒ *nat* ⇒ '*a* ⇒ *real*) ⇒ *bool* **where**
stock-portfolio Mkt p ⟷ *portfolio p* ∧ *support-set p* ⊆ *stocks Mkt*

lemma *sum-portfolio*:
assumes *portfolio pf1*
and *portfolio pf2*
shows *portfolio* (*qty-sum pf1 pf2*) **unfolding** *portfolio-def*
proof −
have *support-set* (*qty-sum pf1 pf2*) ⊆ (*support-set pf1*) ∪ (*support-set pf2*) **by** (*simp add*: *sum-support-set*)
thus *finite* (*support-set* (*qty-sum pf1 pf2*)) **using** *assms* **unfolding** *portfolio-def* **using** *infinite-super* **by** *auto*
qed

lemma *sum-basic-support-set*:
assumes *stock-portfolio Mkt pf1*
and *stock-portfolio Mkt pf2*
shows *stock-portfolio Mkt* (*qty-sum pf1 pf2*) **using** *assms sum-support-set*[*of pf1 pf2*] **unfolding** *stock-portfolio-def*
by (*metis Diff-subset-conv gfp.leq-trans subset-Un-eq sum-portfolio*)

lemma *mult-comp-portfolio*:
assumes *portfolio pf1*
shows *portfolio* (*qty-mult-comp pf1 qty*) **unfolding** *portfolio-def*
proof −
have *support-set* (*qty-mult-comp pf1 qty*) ⊆ (*support-set pf1*) **by** (*simp add*: *mult-comp-support-set*)
thus *finite* (*support-set* (*qty-mult-comp pf1 qty*)) **using** *assms* **unfolding** *portfolio-def* **using** *infinite-super* **by** *auto*
qed

lemma *mult-comp-basic-support-set*:
assumes *stock-portfolio Mkt pf1*
shows *stock-portfolio Mkt* (*qty-mult-comp pf1 qty*) **using** *assms mult-comp-support-set*[*of pf1*] **unfolding** *stock-portfolio-def*

using *mult-comp-portfolio* **by** *blast*

lemma *remove-comp-portfolio*:
assumes *portfolio pf1*
shows *portfolio* (*qty-rem-comp pf1 x*) **unfolding** *portfolio-def*
proof −
have *support-set* (*qty-rem-comp pf1 x*) ⊆ (*support-set pf1*) **using** *remove-comp-support-set*[*of pf1 x*] **by** *blast*
thus *finite* (*support-set* (*qty-rem-comp pf1 x*)) **using** *assms* **unfolding** *portfolio-def* **using** *infinite-super* **by** *auto*
qed

lemma *remove-comp-basic-support-set*:
assumes *stock-portfolio Mkt pf1*
shows *stock-portfolio Mkt* (*qty-mult-comp pf1 qty*) **using** *assms mult-comp-support-set*[*of pf1*] **unfolding** *stock-portfolio-def*
using *mult-comp-portfolio* **by** *blast*

lemma *replace-comp-portfolio*:
assumes *portfolio pf1*
and *portfolio pf2*
shows *portfolio* (*qty-replace-comp pf1 x pf2*) **unfolding** *portfolio-def*
proof −
have *support-set* (*qty-replace-comp pf1 x pf2*) ⊆ (*support-set pf1*) ∪ (*support-set pf2*) **using** *replace-comp-support-set*[*of pf1 x pf2*] **by** *blast*
thus *finite* (*support-set* (*qty-replace-comp pf1 x pf2*)) **using** *assms* **unfolding** *portfolio-def* **using** *infinite-super* **by** *auto*
qed

lemma *replace-comp-stocks*:
assumes *support-set pf1* ⊆ *stocks Mkt* ∪ {*x*}
and *support-set pf2* ⊆ *stocks Mkt*
shows *support-set* (*qty-replace-comp pf1 x pf2*) ⊆ *stocks Mkt*
proof −
have *support-set* (*qty-rem-comp pf1 x*) ⊆ *stocks Mkt* **using** *assms*(*1*) *remove-comp-support-set* **by** *fastforce*
moreover have *support-set* (*qty-mult-comp pf2* (*pf1 x*)) ⊆ *stocks Mkt* **using** *assms mult-comp-support-set* **by** *fastforce*
ultimately show *?thesis* **unfolding** *qty-replace-comp-def* **using** *sum-support-set* **by** *fastforce*
qed

lemma *single-comp-portfolio*:
shows *portfolio* (*qty-single asset qty*)
by (*meson finite.emptyI finite.insertI finite-subset portfolio-def single-comp-support*)

Value processes **definition** *val-process* **where**
val-process Mkt p = (*if* (¬ (*portfolio p*)) *then* (λ *n w*. *0*)
else (λ *n w* . (*sum* (λ*x*. ((*prices Mkt*) *x n w*) ∗ (*p x* (*Suc n*) *w*)) (*support-set p*))))

lemma *subset-val-process′*:
assumes *finite A*
and *support-set p* ⊆ *A*
shows *val-process Mkt p n w* = (*sum* (λ*x*. ((*prices Mkt*) *x n w*) ∗ (*p x* (*Suc n*) *w*)) *A*)
proof −
have *portfolio p* **using** *assms* **unfolding** *portfolio-def* **using** *finite-subset* **by** *auto*
have ∃ *C*. (*support-set p*) ∩ *C* = {} ∧ (*support-set p*) ∪ *C* = *A* **using** *assms*(*2*) **by** *auto*
from *this* **obtain** *C* **where** (*support-set p*) ∩ *C* = {} **and** (*support-set p*) ∪ *C* = *A* **by** *auto* **note** *Cprops* = *this*
have *finite C* **using** *assms* ‹(*support-set p*) ∪ *C* = *A*› **by** *auto*
have ∀ *x*∈ *C*. *p x* (*Suc n*) *w* = *0* **using** *Cprops*(*1*) *support-set-def* **by** *fastforce*
hence (∑ *x*∈ *C*. ((*prices Mkt*) *x n w*) ∗ (*p x* (*Suc n*) *w*)) = *0* **by** *simp*
hence *val-process Mkt p n w* = (∑ *x*∈ (*support-set p*). ((*prices Mkt*) *x n w*) ∗ (*p x* (*Suc n*) *w*))
+ (∑ *x*∈ *C*. ((*prices Mkt*) *x n w*) ∗ (*p x* (*Suc n*) *w*)) **unfolding** *val-process-def* **using** ‹*portfolio p*› **by** *simp*
also have ... = (∑ *x*∈ *A*. ((*prices Mkt*) *x n w*) ∗ (*p x* (*Suc n*) *w*))
using ‹*portfolio p*› ‹*finite C*› *Cprops portfolio-def sum-union-disjoint′* **by** (*metis* (*no-types*, *lifting*))
finally show *val-process Mkt p n w* = (∑ *x*∈ *A*. ((*prices Mkt*) *x n w*) ∗ (*p x* (*Suc n*) *w*)) **.**
qed

lemma *sum-val-process*:
assumes *portfolio pf1*
and *portfolio pf2*
shows ∀ *n w*. *val-process Mkt* (*qty-sum pf1 pf2*) *n w* = (*val-process Mkt pf1*) *n w* + (*val-process Mkt pf2*) *n w*
proof (*intro allI*)
fix *n w*
have *vp1*: *val-process Mkt pf1 n w* = (∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*). ((*prices Mkt*) *x n w*) ∗ (*pf1 x* (*Suc n*) *w*))
proof −
have *finite* (*support-set pf1* ∪ *support-set pf2*) ∧ *support-set pf1* ⊆ *support-set pf1* ∪ *support-set pf2*
by (*meson assms*(*1*) *assms*(*2*) *finite-Un portfolio-def sup.cobounded1*)
then show *?thesis*
by (*simp add*: *subset-val-process′*)

qed
have *vp2*: *val-process Mkt pf2 n w* = (∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*). ((*prices Mkt*) *x n w*) ∗ (*pf2 x* (*Suc n*) *w*))
proof −
have *finite* (*support-set pf1* ∪ *support-set pf2*) ∧ *support-set pf2* ⊆ *support-set pf2* ∪ *support-set pf1*
by (*meson assms*(*1*) *assms*(*2*) *finite-Un portfolio-def sup.cobounded1*)
then show *?thesis*
by (*simp add*: *subset-val-process′*)
qed
have *pf*:*portfolio* (*qty-sum pf1 pf2*) **using** *assms* **by** (*simp add*:*sum-portfolio*)
have *fin*:*finite* (*support-set pf1* ∪ *support-set pf2*) **using** *assms finite-Un* **unfolding** *portfolio-def* **by** *auto*
have (*val-process Mkt pf1*) *n w* + (*val-process Mkt pf2*) *n w* =
(∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*). ((*prices Mkt*) *x n w*) ∗ (*pf1 x* (*Suc n*) *w*)) +
(∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*). ((*prices Mkt*) *x n w*) ∗ (*pf2 x* (*Suc n*) *w*))
using *vp1 vp2* **by** *simp*
also have ... = (∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*).
(((*prices Mkt*) *x n w*) ∗ (*pf1 x* (*Suc n*) *w*)) + ((*prices Mkt*) *x n w*) ∗ (*pf2 x* (*Suc n*) *w*))
by (*simp add*: *sum.distrib*)
also have ... = (∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*).
((*prices Mkt*) *x n w*) ∗ ((*pf1 x* (*Suc n*) *w*) + (*pf2 x* (*Suc n*) *w*))) **by** (*simp add*: *distrib-left*)
also have ... = (∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*).
((*prices Mkt*) *x n w*) ∗ ((*qty-sum pf1 pf2*) *x* (*Suc n*) *w*)) **by** (*simp add*: *qty-sum-def*)
also have ... = (∑ *x*∈ (*support-set* (*qty-sum pf1 pf2*)).
((*prices Mkt*) *x n w*) ∗ ((*qty-sum pf1 pf2*) *x* (*Suc n*) *w*)) **using** *sum-support-set*[*of pf1 pf2*]
subset-val-process′[*of support-set pf1*∪ *support-set pf2 qty-sum pf1 pf2*] *pf fin*
unfolding *val-process-def* **by** *simp*
also have ... = *val-process Mkt* (*qty-sum pf1 pf2*) *n w* **by** (*metis* (*no-types*, *lifting*) *pf sum.cong val-process-def*)
finally have (*val-process Mkt pf1*) *n w* + (*val-process Mkt pf2*) *n w* = *val-process Mkt* (*qty-sum pf1 pf2*) *n w* **.**
thus *val-process Mkt* (*qty-sum pf1 pf2*) *n w* = (*val-process Mkt pf1*) *n w* + (*val-process Mkt pf2*) *n w* **..**
qed

lemma *mult-comp-val-process*:
assumes *portfolio pf1*
shows ∀ *n w*. *val-process Mkt* (*qty-mult-comp pf1 qty*) *n w* = ((*val-process Mkt pf1*) *n w*) ∗ (*qty* (*Suc n*) *w*)
proof (*intro allI*)
fix *n w*

have *pf*:*portfolio* (*qty-mult-comp pf1 qty*) **using** *assms* **by** (*simp add*:*mult-comp-portfolio*)
have *fin*:*finite* (*support-set pf1*) **using** *assms* **unfolding** *portfolio-def* **by** *auto*
have ((*val-process Mkt pf1*) *n w*) ∗ (*qty* (*Suc n*) *w*) =
(∑ *x*∈ (*support-set pf1*). ((*prices Mkt*) *x n w*) ∗ (*pf1 x* (*Suc n*) *w*))∗(*qty* (*Suc n*) *w*)
unfolding *val-process-def* **using** *assms* **by** *simp*
also have ... = (∑ *x*∈ (*support-set pf1*).
(((*prices Mkt*) *x n w*) ∗ (*pf1 x* (*Suc n*) *w*) ∗ (*qty* (*Suc n*) *w*))) **using** *sum-distrib-right* **by** *auto*
also have ... = (∑ *x*∈ (*support-set pf1*).
((*prices Mkt*) *x n w*) ∗ ((*qty-mult-comp pf1 qty*) *x* (*Suc n*) *w*)) **unfolding** *qty-mult-comp-def*
by (*simp add*: *mult.commute mult.left-commute*)
also have ... = (∑ *x*∈ (*support-set* (*qty-mult-comp pf1 qty*)).
((*prices Mkt*) *x n w*) ∗ ((*qty-mult-comp pf1 qty*) *x* (*Suc n*) *w*)) **using** *mult-comp-support-set*[*of pf1*]
subset-val-process'[*of support-set pf1 qty-mult-comp pf1 qty*] *pf fin* **unfolding** *val-process-def* **by** *simp*
also have ... = *val-process Mkt* (*qty-mult-comp pf1 qty*) *n w* **by** (*metis* (*no-types*, *lifting*) *pf sum.cong val-process-def*)
finally have (*val-process Mkt pf1*) *n w* ∗ (*qty* (*Suc n*) *w*) = *val-process Mkt* (*qty-mult-comp pf1 qty*) *n w* **.**
thus *val-process Mkt* (*qty-mult-comp pf1 qty*) *n w* = (*val-process Mkt pf1*) *n w* ∗ (*qty* (*Suc n*) *w*) **..**
qed

lemma *remove-comp-values*:
assumes *x* ≠ *y*
shows ∀ *n w*. *pf1 x n w* = (*qty-rem-comp pf1 y*) *x n w*
proof (*intro allI*)
fix *n w*
show *pf1 x n w* = (*qty-rem-comp pf1 y*) *x n w* **by** (*simp add*: *assms qty-rem-comp-def*)
qed

lemma *remove-comp-val-process*:
assumes *portfolio pf1*
shows ∀ *n w*. *val-process Mkt* (*qty-rem-comp pf1 y*) *n w* = ((*val-process Mkt pf1*) *n w*) − (*prices Mkt y n w*)∗ (*pf1 y* (*Suc n*) *w*)
proof (*intro allI*)
fix *n w*
have *pf*:*portfolio* (*qty-rem-comp pf1 y*) **using** *assms* **by** (*simp add*:*remove-comp-portfolio*)
have *fin*:*finite* (*support-set pf1*) **using** *assms* **unfolding** *portfolio-def* **by** *auto*

hence *fin2*: *finite* (*support-set pf1* − {*y*}) **by** *simp*
have ((*val-process Mkt pf1*) *n w*) =
(∑ *x*∈ (*support-set pf1*). ((*prices Mkt*) *x n w*) ∗ (*pf1 x* (*Suc n*) *w*))
unfolding *val-process-def* **using** *assms* **by** *simp*
also have ... = (∑ *x*∈ (*support-set pf1* − {*y*}).
(((*prices Mkt*) *x n w*) ∗ (*pf1 x* (*Suc n*) *w*))) + (*prices Mkt y n w*)∗ (*pf1 y* (*Suc n*) *w*)
proof (*cases y*∈ *support-set pf1*)
case *True*
thus *?thesis* **by** (*simp add*: *fin sum-diff1*)
next
case *False*
hence *pf1 y* (*Suc n*) *w* = *0* **unfolding** *support-set-def* **by** *simp*
thus *?thesis* **by** (*simp add*: *fin sum-diff1*)
qed
also have ... = (∑ *x*∈ (*support-set pf1* − {*y*}).
((*prices Mkt*) *x n w*) ∗ ((*qty-rem-comp pf1 y*) *x* (*Suc n*) *w*)) + (*prices Mkt y n w*)∗ (*pf1 y* (*Suc n*) *w*)
proof −
have (∑ *x*∈ (*support-set pf1* − {*y*}). (((*prices Mkt*) *x n w*) ∗ (*pf1 x* (*Suc n*) *w*))) =
(∑ *x*∈ (*support-set pf1* − {*y*}). ((*prices Mkt*) *x n w*) ∗ ((*qty-rem-comp pf1 y*) *x* (*Suc n*) *w*))
proof (*rule sum.cong*,*simp*)
fix *x*
assume *x*∈ *support-set pf1* − {*y*}
show *prices Mkt x n w* ∗ *pf1 x* (*Suc n*) *w* = *prices Mkt x n w* ∗ *qty-rem-comp pf1 y x* (*Suc n*) *w* **using** *remove-comp-values*
by (*metis DiffD2* ‹*x* ∈ *support-set pf1* − {*y*}› *singletonI*)
qed
thus *?thesis* **by** *simp*
qed
also have ... = (*val-process Mkt* (*qty-rem-comp pf1 y*) *n w*) + (*prices Mkt y n w*)∗ (*pf1 y* (*Suc n*) *w*)
using *subset-val-process'*[*of support-set pf1* − {*y*} *qty-rem-comp pf1 y*] *fin2*
by (*simp add*: *remove-comp-support-set*)
finally have (*val-process Mkt pf1*) *n w* =
(*val-process Mkt* (*qty-rem-comp pf1 y*) *n w*) + (*prices Mkt y n w*)∗ (*pf1 y* (*Suc n*) *w*) **.**
thus *val-process Mkt* (*qty-rem-comp pf1 y*) *n w* = ((*val-process Mkt pf1*) *n w*) − (*prices Mkt y n w*)∗ (*pf1 y* (*Suc n*) *w*) **by** *simp*
qed

lemma *replace-comp-val-process*:
assumes ∀ *n w*. *prices Mkt x n w* = *val-process Mkt pf2 n w*
and *portfolio pf1*

and *portfolio pf2*
shows $\forall$ *n w. val-process Mkt* (*qty-replace-comp pf1 x pf2*) *n w* = *val-process Mkt pf1 n w*
proof (*intro allI*)
fix *n w*
have *val-process Mkt* (*qty-replace-comp pf1 x pf2*) *n w* = *val-process Mkt* (*qty-rem-comp pf1 x*) *n w* +
val-process Mkt (*qty-mult-comp pf2* (*pf1 x*)) *n w* **unfolding** *qty-replace-comp-def*
using *assms*
sum-val-process[*of qty-rem-comp pf1 x qty-mult-comp pf2* (*pf1 x*)]
by (*simp add*: *mult-comp-portfolio remove-comp-portfolio*)
also have ... = *val-process Mkt pf1 n w* − (*prices Mkt x n w* ∗ *pf1 x* (*Suc n*) *w*) + *val-process Mkt pf2 n w* ∗ *pf1 x* (*Suc n*) *w*
by (*simp add*: *assms*(*2*) *assms*(*3*) *mult-comp-val-process remove-comp-val-process*)
also have ... = *val-process Mkt pf1 n w* **using** *assms* **by** *simp*
finally show *val-process Mkt* (*qty-replace-comp pf1 x pf2*) *n w* = *val-process Mkt pf1 n w* **.**
qed

lemma *qty-single-val-process*:
shows *val-process Mkt* (*qty-single asset qty*) *n w* =
prices Mkt asset n w ∗ *qty* (*Suc n*) *w*
proof −
have *val-process Mkt* (*qty-single asset qty*) *n w* =
(*sum* (λ*x*. ((*prices Mkt*) *x n w*) ∗ ((*qty-single asset qty*) *x* (*Suc n*) *w*)) {*asset*})
proof (*rule subset-val-process′*)
show *finite* {*asset*} **by** *simp*
show *support-set* (*qty-single asset qty*) ⊆ {*asset*} **by** (*simp add*: *single-comp-support*)
qed
also have ... = *prices Mkt asset n w* ∗ *qty* (*Suc n*) *w* **unfolding** *qty-single-def* **by** *simp*
finally show *?thesis* **.**
qed

7.2.3 Trading strategies

locale *disc-equity-market* = *triv-init-disc-filtr-prob-space* +
fixes *Mkt*::(*′a*,*′b*) *discrete-market*

Discrete predictable processes

Trading strategy **definition** (**in** *disc-filtr-prob-space*) *trading-strategy*
where
trading-strategy p ⟷ *portfolio p* ∧ (∀ *asset* ∈ *support-set p*. *borel-predict-stoch-proc F* (*p asset*))

definition (**in** *disc-filtr-prob-space*) *support-adapt*:: (*′a*, *′b*) *discrete-market* ⇒ (*′b* ⇒ *nat* ⇒ *′a* ⇒ *real*) ⇒ *bool* **where**

support-adapt Mkt pf $\longleftrightarrow$ ($\forall$ *asset* $\in$ *support-set pf*. *borel-adapt-stoch-proc F* (*prices Mkt asset*))

lemma (**in** *disc-filtr-prob-space*) *quantity-adapted*:
assumes $\forall$ *asset* $\in$ *support-set p*. *p asset* (*Suc n*) $\in$ *borel-measurable* (*F n*)
$\forall$ *asset* $\in$ *support-set p*. *prices Mkt asset n* $\in$ *borel-measurable* (*F n*)
shows *val-process Mkt p n* $\in$ *borel-measurable* (*F n*)
proof (*cases portfolio p*)
case *False*
have *val-process Mkt p* = (λ *n w*. *0*) **unfolding** *val-process-def* **using** *False* **by** *simp*
thus *?thesis* **by** *simp*
next
case *True*
hence *val-process Mkt p n* = (λw. $\sum x \in$*support-set p*. *prices Mkt x n w* $*$ *p x* (*Suc n*) *w*)
unfolding *val-process-def* **using** *True* **by** *simp*
moreover have (λw. $\sum x \in$*support-set p*. *prices Mkt x n w* $*$ *p x* (*Suc n*) *w*) $\in$ *borel-measurable* (*F n*)
proof (*rule borel-measurable-sum*)
fix *asset*
assume *asset*$\in$ *support-set p*
hence *p asset* (*Suc n*) $\in$ *borel-measurable* (*F n*) **using** *assms* **unfolding** *trading-strategy-def adapt-stoch-proc-def* **by** *simp*
moreover have *prices Mkt asset n* $\in$ *borel-measurable* (*F n*)
using ‹*asset* $\in$ *support-set p*› *assms*(*2*) **unfolding** *support-adapt-def* **by** (*simp add*: *adapt-stoch-proc-def*)
ultimately show (λx. *prices Mkt asset n x* $*$ *p asset* (*Suc n*) *x*) $\in$ *borel-measurable* (*F n*) **by** *simp*
qed
ultimately show *val-process Mkt p n* $\in$ *borel-measurable* (*F n*) **by** *simp*
qed

lemma (**in** *disc-filtr-prob-space*) *trading-strategy-adapted*:
assumes *trading-strategy p*
and *support-adapt Mkt p*
shows *borel-adapt-stoch-proc F* (*val-process Mkt p*) **unfolding** *support-adapt-def*
proof (*cases portfolio p*)
case *False*
have *val-process Mkt p* = (λ *n w*. *0*) **unfolding** *val-process-def* **using** *False* **by** *simp*
thus *borel-adapt-stoch-proc F* (*val-process Mkt p*)
by (*simp add*: *constant-process-borel-adapted*)
next
case *True*
show *?thesis* **unfolding** *adapt-stoch-proc-def*
proof

fix *n*
have *val-process Mkt p n* = (λ*w*. ∑ *x*∈*support-set p*. *prices Mkt x n w* ∗ *p x* (*Suc n*) *w*)
unfolding *val-process-def* **using** *True* **by** *simp*
moreover have (λ*w*. ∑ *x*∈*support-set p*. *prices Mkt x n w* ∗ *p x* (*Suc n*) *w*) ∈ *borel-measurable* (*F n*)
proof (*rule borel-measurable-sum*)
fix *asset*
assume *asset*∈ *support-set p*
hence *p asset* (*Suc n*) ∈ *borel-measurable* (*F n*) **using** *assms* **unfolding** *trading-strategy-def predict-stoch-proc-def* **by** *simp*
moreover have *prices Mkt asset n* ∈ *borel-measurable* (*F n*)
using ‹*asset* ∈ *support-set p*› *assms*(*2*) **unfolding** *support-adapt-def* **by** (*simp add*:*adapt-stoch-proc-def*)
ultimately show (λ*x*. *prices Mkt asset n x* ∗ *p asset* (*Suc n*) *x*) ∈ *borel-measurable* (*F n*) **by** *simp*
qed
ultimately show *val-process Mkt p n* ∈ *borel-measurable* (*F n*) **by** *simp*
qed
qed

lemma (**in** *disc-equity-market*) *ats-val-process-adapted*:
assumes *trading-strategy p*
and *support-adapt Mkt p*
shows *borel-adapt-stoch-proc F* (*val-process Mkt p*) **unfolding** *support-adapt-def*
by (*meson assms*(*1*) *assms*(*2*) *subsetCE trading-strategy-adapted*)

lemma (**in** *disc-equity-market*) *trading-strategy-init*:
assumes *trading-strategy p*
and *support-adapt Mkt p*
shows ∃ *c*. ∀ *w* ∈ *space M*. *val-process Mkt p 0 w* = *c* **using** *assms adapted-init ats-val-process-adapted* **by** *simp*

definition (**in** *disc-equity-market*) *initial-value* **where**
initial-value pf = *constant-image* (*val-process Mkt pf 0*)

lemma (**in** *disc-equity-market*) *initial-valueI*:
assumes *trading-strategy pf*
and *support-adapt Mkt pf*
shows ∀ *w*∈ *space M*. *val-process Mkt pf 0 w* = *initial-value pf* **unfolding** *initial-value-def*
proof (*rule constant-imageI*)
show ∃ *c*. ∀ *w*∈*space M*. *val-process Mkt pf 0 w* = *c* **using** *trading-strategy-init*

assms **by** *simp*
qed

lemma (**in** *disc-equity-market*) *inc-predict-support-trading-strat*:
assumes *trading-strategy pf1*
shows ∀ *asset* ∈ *support-set pf1* ∪ *B*. *borel-predict-stoch-proc F* (*pf1 asset*)
proof
fix *asset*
assume *asset* ∈ *support-set pf1* ∪ *B*
show *borel-predict-stoch-proc F* (*pf1 asset*)
proof (*cases asset* ∈ *support-set pf1*)
case *True*
thus *?thesis* **using** *assms* **unfolding** *trading-strategy-def* **by** *simp*
next
case *False*
hence ∀ *n w*. *pf1 asset n w* = *0* **unfolding** *support-set-def* **by** *simp*
show *?thesis* **unfolding** *predict-stoch-proc-def*
proof
show *pf1 asset 0* ∈ *measurable* (*F 0*) *borel* **using** ‹∀ *n w*. *pf1 asset n w* = *0*›
by (*simp add*: *borel-measurable-const measurable-cong*)
next
show ∀ *n*. *pf1 asset* (*Suc n*) ∈ *borel-measurable* (*F n*)
proof
fix *n*
have ∀ *w*. *pf1 asset* (*Suc n*) *w* = *0* **using** ‹∀ *n w*. *pf1 asset n w* = *0*› **by** *simp*
have *0*∈ *space borel* **by** *simp*
thus *pf1 asset* (*Suc n*) ∈ *measurable* (*F n*) *borel* **using** *measurable-const*[*of 0 borel F n*]
by (*metis* ‹*0* ∈ *space borel* ⟹ (λ*x*. *0*) ∈ *borel-measurable* (*F n*)› ‹*0* ∈ *space borel*›
‹∀ *n w*. *pf1 asset n w* = *0*› *measurable-cong*)
qed
qed
qed
qed

lemma (**in** *disc-equity-market*) *inc-predict-support-trading-strat′*:
assumes *trading-strategy pf1*
and *asset* ∈ *support-set pf1*∪ *B*
shows *borel-predict-stoch-proc F* (*pf1 asset*)
proof (*cases asset* ∈ *support-set pf1*)
case *True*
thus *?thesis* **using** *assms* **unfolding** *trading-strategy-def* **by** *simp*
next
case *False*
hence ∀ *n w*. *pf1 asset n w* = *0* **unfolding** *support-set-def* **by** *simp*
show *?thesis* **unfolding** *predict-stoch-proc-def*

proof
show *pf1 asset 0* ∈ *measurable* (*F 0*) *borel* **using** ‹∀ *n w*. *pf1 asset n w* = *0*›
by (*simp add*: *borel-measurable-const measurable-cong*)
next
show ∀ *n*. *pf1 asset* (*Suc n*) ∈ *borel-measurable* (*F n*)
proof
fix *n*
have ∀ *w*. *pf1 asset* (*Suc n*) *w* = *0* **using** ‹∀ *n w*. *pf1 asset n w* = *0*› **by** *simp*
have *0*∈ *space borel* **by** *simp*
thus *pf1 asset* (*Suc n*) ∈ *measurable* (*F n*) *borel* **using** *measurable-const*[*of 0 borel F n*]
by (*metis* ‹*0* ∈ *space borel* ⟹ (λ*x*. *0*) ∈ *borel-measurable* (*F n*)› ‹*0* ∈ *space borel*›
‹∀ *n w*. *pf1 asset n w* = *0*› *measurable-cong*)
qed
qed
qed

lemma (**in** *disc-equity-market*) *inc-support-trading-strat*:
assumes *trading-strategy pf1*
shows ∀ *asset* ∈ *support-set pf1* ∪ *B*. *borel-adapt-stoch-proc F* (*pf1 asset*) **using** *assms*
by (*simp add*: *inc-predict-support-trading-strat predict-imp-adapt*)

lemma (**in** *disc-equity-market*) *qty-empty-trading-strat*:
shows *trading-strategy qty-empty* **unfolding** *trading-strategy-def*
proof (*intro conjI ballI*)
show *portfolio qty-empty*
by (*metis fun-upd-triv qty-single-def single-comp-portfolio*)
show ⋀*asset*. *asset* ∈ *support-set qty-empty* ⟹ *borel-predict-stoch-proc F* (*qty-empty asset*)
using *qty-empty-support-set* **by** *auto*
qed

lemma (**in** *disc-equity-market*) *sum-trading-strat*:
assumes *trading-strategy pf1*
and *trading-strategy pf2*
shows *trading-strategy* (*qty-sum pf1 pf2*)
proof −
have ∀ *asset* ∈ *support-set pf1* ∪ *support-set pf2*. *borel-predict-stoch-proc F* (*pf1 asset*)
using *assms* **by** (*simp add*: *inc-predict-support-trading-strat*)
have ∀ *asset* ∈ *support-set pf2* ∪ *support-set pf1*. *borel-predict-stoch-proc F* (*pf2 asset*)
using *assms* **by** (*simp add*: *inc-predict-support-trading-strat*)
have ∀ *asset* ∈ *support-set pf1* ∪ *support-set pf2*. *borel-predict-stoch-proc F* ((*qty-sum pf1 pf2*) *asset*)

proof
fix *asset*
assume *asset* ∈ *support-set pf1* ∪ *support-set pf2*
show *borel-predict-stoch-proc F* (*qty-sum pf1 pf2 asset*) **unfolding** *predict-stoch-proc-def qty-sum-def*
proof
show (λ*w*. *pf1 asset 0 w* + *pf2 asset 0 w*) ∈ *borel-measurable* (*F 0*)
proof −
have (λ*w*. *pf1 asset 0 w*) ∈ *borel-measurable* (*F 0*)
using ‹∀ *asset*∈*support-set pf1* ∪ *support-set pf2*. *borel-predict-stoch-proc F* (*pf1 asset*)›
‹*asset* ∈ *support-set pf1* ∪ *support-set pf2*› *predict-stoch-proc-def* **by** *blast*
moreover have (λ*w*. *pf2 asset 0 w*) ∈ *borel-measurable* (*F 0*)
using ‹∀ *asset*∈*support-set pf2* ∪ *support-set pf1*. *borel-predict-stoch-proc F* (*pf2 asset*)›
‹*asset* ∈ *support-set pf1* ∪ *support-set pf2*› *predict-stoch-proc-def* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed
next
show ∀ *n*. (λ*w*. *pf1 asset* (*Suc n*) *w* + *pf2 asset* (*Suc n*) *w*) ∈ *borel-measurable* (*F n*)
proof
fix *n*
have (λ*w*. *pf1 asset* (*Suc n*) *w*) ∈ *borel-measurable* (*F n*)
using ‹∀ *asset*∈*support-set pf1* ∪ *support-set pf2*. *borel-predict-stoch-proc F* (*pf1 asset*)›
‹*asset* ∈ *support-set pf1* ∪ *support-set pf2*› *predict-stoch-proc-def* **by** *blast*
moreover have (λ*w*. *pf2 asset* (*Suc n*) *w*) ∈ *borel-measurable* (*F n*)
using ‹∀ *asset*∈*support-set pf2* ∪ *support-set pf1*. *borel-predict-stoch-proc F* (*pf2 asset*)›
‹*asset* ∈ *support-set pf1* ∪ *support-set pf2*› *predict-stoch-proc-def* **by** *blast*
ultimately show (λ*w*. *pf1 asset* (*Suc n*) *w* + *pf2 asset* (*Suc n*) *w*) ∈ *borel-measurable* (*F n*) **by** *simp*
qed
qed
qed
thus *?thesis* **unfolding** *trading-strategy-def* **using** *sum-support-set*[*of pf1 pf2*]
by (*meson assms*(*1*) *assms*(*2*) *subsetCE sum-portfolio trading-strategy-def*)
qed

lemma (**in** *disc-equity-market*) *mult-comp-trading-strat*:
assumes *trading-strategy pf1*
and *borel-predict-stoch-proc F qty*
shows *trading-strategy* (*qty-mult-comp pf1 qty*)
proof −
have ∀ *asset* ∈ *support-set pf1*. *borel-predict-stoch-proc F* (*pf1 asset*)
using *assms* **unfolding** *trading-strategy-def* **by** *simp*
have ∀ *asset* ∈ *support-set pf1*. *borel-predict-stoch-proc F* (*qty-mult-comp pf1 qty asset*)

unfolding *predict-stoch-proc-def qty-mult-comp-def*
proof (*intro ballI conjI*)
fix *asset*
assume *asset* ∈ *support-set pf1*
show (λ*w*. *pf1 asset 0 w* ∗ *qty 0 w*) ∈ *borel-measurable* (*F 0*)
proof −
have (λ*w*. *pf1 asset 0 w*) ∈ *borel-measurable* (*F 0*)
using ‹∀ *asset*∈*support-set pf1*. *borel-predict-stoch-proc F* (*pf1 asset*)›
‹*asset* ∈ *support-set pf1*› *predict-stoch-proc-def* **by** *auto*
moreover have (λ*w*. *qty 0 w*) ∈ *borel-measurable* (*F 0*) **using** *assms predict-stoch-proc-def* **by** *auto*
ultimately show (λ*w*. *pf1 asset 0 w* ∗ *qty 0 w*) ∈ *borel-measurable* (*F 0*) **by** *simp*
qed
show ∀ *n*. (λ*w*. *pf1 asset* (*Suc n*) *w* ∗ *qty* (*Suc n*) *w*) ∈ *borel-measurable* (*F n*)
proof
fix *n*
have (λ*w*. *pf1 asset* (*Suc n*) *w*) ∈ *borel-measurable* (*F n*)
using ‹∀ *asset*∈*support-set pf1*. *borel-predict-stoch-proc F* (*pf1 asset*)›
‹*asset* ∈ *support-set pf1*› *predict-stoch-proc-def* **by** *blast*
moreover have (λ*w*. *qty* (*Suc n*) *w*) ∈ *borel-measurable* (*F n*) **using** *assms predict-stoch-proc-def* **by** *blast*
ultimately show (λ*w*. *pf1 asset* (*Suc n*) *w* ∗ *qty* (*Suc n*) *w*) ∈ *borel-measurable* (*F n*) **by** *simp*
qed
qed
thus *?thesis* **unfolding** *trading-strategy-def* **using** *mult-comp-support-set*[*of pf1 qty*]
by (*meson assms*(*1*) *mult-comp-portfolio subsetCE trading-strategy-def*)
qed

lemma (**in** *disc-equity-market*) *remove-comp-trading-strat*:
assumes *trading-strategy pf1*
shows *trading-strategy* (*qty-rem-comp pf1 x*)
proof −
have ∀ *asset* ∈ *support-set pf1*. *borel-predict-stoch-proc F* (*pf1 asset*)
using *assms* **unfolding** *trading-strategy-def* **by** *simp*
have ∀ *asset* ∈ *support-set pf1*. *borel-predict-stoch-proc F* (*qty-rem-comp pf1 x asset*)
unfolding *predict-stoch-proc-def qty-rem-comp-def*
proof (*intro ballI conjI*)
fix *asset*
assume *asset* ∈ *support-set pf1*
show (*pf1*(*x* := λ*n w*. *0*)) *asset 0* ∈ *borel-measurable* (*F 0*)
proof −
show (λ*w*. (*pf1*(*x* := λ*n w*. *0*)) *asset 0 w*) ∈ *borel-measurable* (*F 0*)
proof (*cases asset* = *x*)
case *True*
thus *?thesis* **by** *simp*

```
      next
        case False
        thus (λw. (pf1(x := λn w. 0)) asset 0 w) ∈ borel-measurable (F 0)
          using ‹∀ asset∈support-set pf1. borel-predict-stoch-proc F (pf1 asset)›
          ‹asset ∈ support-set pf1› by (simp add: predict-stoch-proc-def)
      qed
    qed
    show ∀ n. (λw. (pf1(x := λn w. 0)) asset (Suc n) w) ∈ borel-measurable (F n)
    proof
      fix n
      show (λw. (pf1(x := λn w. 0)) asset (Suc n) w) ∈ borel-measurable (F n)
      proof (cases asset = x)
        case True
        thus ?thesis by simp
      next
        case False
        thus (λw. (pf1(x := λn w. 0)) asset (Suc n) w) ∈ borel-measurable (F n)
          using ‹∀ asset∈support-set pf1. borel-predict-stoch-proc F (pf1 asset)›
          ‹asset ∈ support-set pf1› by (simp add: predict-stoch-proc-def)
      qed
    qed
  qed
  thus ?thesis unfolding trading-strategy-def using remove-comp-support-set[of
pf1 x]
    by (metis Diff-empty assms remove-comp-portfolio subsetCE subset-Diff-insert
trading-strategy-def)
qed


lemma (in disc-equity-market) replace-comp-trading-strat:
  assumes trading-strategy pf1
  and trading-strategy pf2
shows trading-strategy (qty-replace-comp pf1 x pf2) unfolding qty-replace-comp-def
proof (rule sum-trading-strat)
  show trading-strategy (qty-rem-comp pf1 x) using assms by (simp add: re-
move-comp-trading-strat)
  show trading-strategy (qty-mult-comp pf2 (pf1 x))
  proof (cases x∈ support-set pf1)
    case True
  hence borel-predict-stoch-proc F (pf1 x) using assms unfolding trading-strategy-def
by simp
    thus ?thesis using assms by (simp add: mult-comp-trading-strat)
  next
    case False
    thus ?thesis
    by (meson UnCI assms(1) assms(2) disc-equity-market.inc-predict-support-trading-strat
        disc-equity-market-axioms insertI1 mult-comp-trading-strat)
  qed
qed
```

7.2.4 Self-financing portfolios

Closing value process **fun** *up-cl-proc* **where**
up-cl-proc Mkt p 0 = val-process Mkt p 0 |
*up-cl-proc Mkt p (Suc n) = (λw. ∑x∈support-set p. prices Mkt x (Suc n) w * p x (Suc n) w)*

definition *cls-val-process* **where**
cls-val-process Mkt p = (if (¬ (portfolio p)) then (λ n w. 0)
else (λ n w . up-cl-proc Mkt p n w))

lemma (**in** *disc-filtr-prob-space*) *quantity-updated-borel*:
assumes ∀ *n*. ∀ *asset* ∈ *support-set p*. *p asset* (*Suc n*) ∈ *borel-measurable* (*F n*)
and ∀ *n*. ∀ *asset* ∈ *support-set p*. *prices Mkt asset n* ∈ *borel-measurable* (*F n*)
shows ∀ *n*. *cls-val-process Mkt p n* ∈ *borel-measurable* (*F n*)
proof (*cases portfolio p*)
case *False*
have *cls-val-process Mkt p* = (λ *n w*. *0*) **unfolding** *cls-val-process-def* **using** *False* **by** *simp*
thus *?thesis* **by** *simp*
next
case *True*
show ∀ *n*. *cls-val-process Mkt p n* ∈ *borel-measurable* (*F n*)
proof
fix *n*
show *cls-val-process Mkt p n* ∈ *borel-measurable* (*F n*)
proof (*cases n = 0*)
case *False*
hence ∃ *m*. *n* = *Suc m* **using** *old.nat.exhaust* **by** *auto*
from *this* **obtain** *m* **where** *n* = *Suc m* **by** *auto*
have *cls-val-process Mkt p* (*Suc m*) = (λ*w*. ∑*x*∈*support-set p*. *prices Mkt x* (*Suc m*) *w* * *p x* (*Suc m*) *w*)
unfolding *cls-val-process-def* **using** *True* **by** *simp*
moreover have (λ*w*. ∑*x*∈*support-set p*. *prices Mkt x* (*Suc m*) *w* * *p x* (*Suc m*) *w*) ∈ *borel-measurable* (*F* (*Suc m*))
proof (*rule borel-measurable-sum*)
fix *asset*
assume *asset*∈ *support-set p*
hence *p asset* (*Suc m*) ∈ *borel-measurable* (*F m*) **using** *assms* **unfolding** *trading-strategy-def adapt-stoch-proc-def* **by** *simp*
hence *p asset* (*Suc m*) ∈ *borel-measurable* (*F* (*Suc m*))
using *Suc-n-not-le-n increasing-measurable-info nat-le-linear* **by** *blast*
moreover have *prices Mkt asset* (*Suc m*) ∈ *borel-measurable* (*F* (*Suc m*))
using ‹*asset* ∈ *support-set p*› *assms*(*2*) **unfolding** *support-adapt-def adapt-stoch-proc-def* **by** *blast*
ultimately show (λ*x*. *prices Mkt asset* (*Suc m*) *x* * *p asset* (*Suc m*) *x*) ∈ *borel-measurable* (*F* (*Suc m*)) **by** *simp*

```
    qed
    ultimately have cls-val-process Mkt p (Suc m) ∈ borel-measurable (F (Suc
m)) by simp
    thus ?thesis using ‹n = Suc m› by simp
   next
    case True
    thus cls-val-process Mkt p n ∈ borel-measurable (F n)
    by (metis (no-types, lifting) assms(1) assms(2) quantity-adapted up-cl-proc.simps(1)
        cls-val-process-def val-process-def)
   qed
  qed
qed


lemma (in disc-equity-market) cls-val-process-adapted:
  assumes trading-strategy p
and support-adapt Mkt p
  shows borel-adapt-stoch-proc F (cls-val-process Mkt p)
proof (cases portfolio p)
  case False
    have cls-val-process Mkt p = (λ n w. 0) unfolding cls-val-process-def using
False by simp
    thus borel-adapt-stoch-proc F (cls-val-process Mkt p)
     by (simp add: constant-process-borel-adapted)
next
  case True
  show ?thesis unfolding adapt-stoch-proc-def
  proof
    fix n
    show cls-val-process Mkt p n ∈ borel-measurable (F n)
    proof (cases n = 0)
    case True
     thus cls-val-process Mkt p n ∈ borel-measurable (F n)
       using up-cl-proc.simps(1) assms
         by (metis (no-types, lifting) adapt-stoch-proc-def ats-val-process-adapted
cls-val-process-def
           val-process-def)
    next
    case False
     hence ∃ m. Suc m = n using not0-implies-Suc by blast
     from this obtain m where Suc m = n by auto
    hence cls-val-process Mkt p n = up-cl-proc Mkt p n unfolding cls-val-process-def
using True by simp
     also have ... = (λw. ∑ x∈support-set p. prices Mkt x n w * p x n w)
       using up-cl-proc.simps(2) ‹Suc m = n› by auto
     finally have cls-val-process Mkt p n = (λw. ∑ x∈support-set p. prices Mkt x
n w * p x n w) .
    moreover have (λw. ∑ x∈support-set p. prices Mkt x n w * p x n w) ∈
borel-measurable (F n)
```

proof (*rule borel-measurable-sum*)
fix *asset*
assume *asset*∈ *support-set p*
hence *p asset n* ∈ *borel-measurable* (*F n*) **using** *assms* **unfolding** *trading-strategy-def predict-stoch-proc-def*
using *Suc-n-not-le-n* ‹*Suc m* = *n*› *increasing-measurable-info nat-le-linear* **by** *blast*
moreover have *prices Mkt asset n* ∈ *borel-measurable* (*F n*) **using** *assms* ‹*asset* ∈ *support-set p*› **unfolding** *support-adapt-def adapt-stoch-proc-def*
using *stock-portfolio-def* **by** *blast*
ultimately show (λ*x*. *prices Mkt asset n x* ∗ *p asset n x*) ∈ *borel-measurable* (*F n*) **by** *simp*
qed
ultimately show *cls-val-process Mkt p n* ∈ *borel-measurable* (*F n*) **by** *simp*
qed
qed
qed

lemma *subset-cls-val-process*:
assumes *finite A*
and *support-set p* ⊆ *A*
shows ∀ *n w*. *cls-val-process Mkt p* (*Suc n*) *w* = (*sum* (λ*x*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*)) *A*)
proof (*intro allI*)
fix *n*::*nat*
fix *w*::*'b*
have *portfolio p* **using** *assms* **unfolding** *portfolio-def* **using** *finite-subset* **by** *auto*
have ∃ *C*. (*support-set p*) ∩ *C* = {} ∧ (*support-set p*) ∪ *C* = *A* **using** *assms*(*2*) **by** *auto*
from *this* **obtain** *C* **where** (*support-set p*) ∩ *C* = {} **and** (*support-set p*) ∪ *C* = *A* **by** *auto* **note** *Cprops* = *this*
have *finite C* **using** *assms* ‹(*support-set p*) ∪ *C* = *A*› **by** *auto*
have ∀ *x*∈ *C*. *p x* (*Suc n*) *w* = *0* **using** *Cprops*(*1*) *support-set-def* **by** *fastforce*
hence (∑ *x*∈ *C*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*)) = *0* **by** *simp*
hence *cls-val-process Mkt p* (*Suc n*) *w* = (∑ *x*∈ (*support-set p*). ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*))
+ (∑ *x*∈ *C*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*)) **unfolding** *cls-val-process-def*
using ‹*portfolio p*› *up-cl-proc.simps*(*2*)[*of Mkt p n*] **by** *simp*
also have ... = (∑ *x*∈ *A*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*))
using ‹*portfolio p*› ‹*finite C*› *Cprops portfolio-def sum-union-disjoint'* **by** (*metis* (*no-types*, *lifting*))
finally show *cls-val-process Mkt p* (*Suc n*) *w* = (∑ *x*∈ *A*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*)) .
qed

lemma *subset-cls-val-process'*:
assumes *finite A*

and *support-set p* $\subseteq$ *A*
shows *cls-val-process Mkt p* (*Suc n*) *w* = (*sum* (λ*x*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*)) *A*)
proof −
have *portfolio p* **using** *assms* **unfolding** *portfolio-def* **using** *finite-subset* **by** *auto*
have ∃ *C*. (*support-set p*) ∩ *C* = {} ∧ (*support-set p*) ∪ *C* = *A* **using** *assms*(*2*) **by** *auto*
from *this* **obtain** *C* **where** (*support-set p*) ∩ *C* = {} **and** (*support-set p*) ∪ *C* = *A* **by** *auto* **note** *Cprops* = *this*
have *finite C* **using** *assms* ‹(*support-set p*) ∪ *C* = *A*› **by** *auto*
have ∀ *x*∈ *C*. *p x* (*Suc n*) *w* = *0* **using** *Cprops*(*1*) *support-set-def* **by** *fastforce*
hence (∑ *x*∈ *C*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*)) = *0* **by** *simp*
hence *cls-val-process Mkt p* (*Suc n*) *w* = (∑ *x*∈ (*support-set p*). ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*))
+ (∑ *x*∈ *C*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*)) **unfolding** *cls-val-process-def*
using ‹*portfolio p*› *up-cl-proc.simps*(*2*)[*of Mkt p n*] **by** *simp*
also have ... = (∑ *x*∈ *A*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*))
using ‹*portfolio p*› ‹*finite C*› *Cprops portfolio-def sum-union-disjoint′* **by** (*metis* (*no-types*, *lifting*))
finally show *cls-val-process Mkt p* (*Suc n*) *w* = (∑ *x*∈ *A*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*)) .
qed

lemma *sum-cls-val-process-Suc*:
assumes *portfolio pf1*
and *portfolio pf2*
shows ∀ *n w*. *cls-val-process Mkt* (*qty-sum pf1 pf2*) (*Suc n*) *w* =
(*cls-val-process Mkt pf1*) (*Suc n*) *w* + (*cls-val-process Mkt pf2*) (*Suc n*) *w*
proof (*intro allI*)
fix *n w*
have *vp1*: *cls-val-process Mkt pf1* (*Suc n*) *w* =
(∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*). ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf1 x* (*Suc n*) *w*))
proof −
have *finite* (*support-set pf1* ∪ *support-set pf2*) ∧ *support-set pf1* ⊆ *support-set pf1* ∪ *support-set pf2*
by (*meson assms*(*1*) *assms*(*2*) *finite-Un portfolio-def sup.cobounded1*)
then show *?thesis*
by (*simp add*: *subset-cls-val-process*)
qed
have *vp2*: *cls-val-process Mkt pf2* (*Suc n*) *w* = (∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*). ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf2 x* (*Suc n*) *w*))
proof −
have *finite* (*support-set pf1* ∪ *support-set pf2*) ∧ *support-set pf2* ⊆ *support-set pf2* ∪ *support-set pf1*

by (*meson assms*(*1*) *assms*(*2*) *finite-Un portfolio-def sup.cobounded1*)
then show *?thesis* **by** (*auto simp add*: *subset-cls-val-process*)
qed
have *pf*:*portfolio* (*qty-sum pf1 pf2*) **using** *assms* **by** (*simp add*:*sum-portfolio*)
have *fin*:*finite* (*support-set pf1* ∪ *support-set pf2*) **using** *assms finite-Un* **unfolding** *portfolio-def* **by** *auto*
have (*cls-val-process Mkt pf1*) (*Suc n*) *w* + (*cls-val-process Mkt pf2*) (*Suc n*) *w* =
(∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*). ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf1 x* (*Suc n*) *w*)) +
(∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*). ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf2 x* (*Suc n*) *w*))
using *vp1 vp2* **by** *simp*
also have ... = (∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*).
(((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf1 x* (*Suc n*) *w*)) + ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf2 x* (*Suc n*) *w*))
by (*simp add*: *sum.distrib*)
also have ... = (∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*).
((*prices Mkt*) *x* (*Suc n*) *w*) ∗ ((*pf1 x* (*Suc n*) *w*) + (*pf2 x* (*Suc n*) *w*))) **by** (*simp add*: *distrib-left*)
also have ... = (∑ *x*∈ (*support-set pf1*)∪ (*support-set pf2*).
((*prices Mkt*) *x* (*Suc n*) *w*) ∗ ((*qty-sum pf1 pf2*) *x* (*Suc n*) *w*)) **by** (*simp add*: *qty-sum-def*)
also have ... = (∑ *x*∈ (*support-set* (*qty-sum pf1 pf2*)).
((*prices Mkt*) *x* (*Suc n*) *w*) ∗ ((*qty-sum pf1 pf2*) *x* (*Suc n*) *w*)) **using** *sum-support-set*[*of pf1 pf2*]
subset-cls-val-process[*of support-set pf1*∪ *support-set pf2 qty-sum pf1 pf2*] *pf fin*
unfolding *cls-val-process-def* **by** *simp*
also have ... = *cls-val-process Mkt* (*qty-sum pf1 pf2*) (*Suc n*) *w*
by (*metis* (*no-types*, *lifting*) *pf sum.cong up-cl-proc.simps*(*2*) *cls-val-process-def*)
finally have (*cls-val-process Mkt pf1*) (*Suc n*) *w* + (*cls-val-process Mkt pf2*) (*Suc n*) *w* =
cls-val-process Mkt (*qty-sum pf1 pf2*) (*Suc n*) *w* **.**
thus *cls-val-process Mkt* (*qty-sum pf1 pf2*) (*Suc n*) *w* =
(*cls-val-process Mkt pf1*) (*Suc n*) *w* + (*cls-val-process Mkt pf2*) (*Suc n*) *w* **..**
qed

lemma *sum-cls-val-process0*:
assumes *portfolio pf1*
and *portfolio pf2*
shows ∀ *w*. *cls-val-process Mkt* (*qty-sum pf1 pf2*) *0 w* =
(*cls-val-process Mkt pf1*) *0 w* + (*cls-val-process Mkt pf2*) *0 w* **unfolding** *cls-val-process-def*
by (*simp add*: *sum-val-process assms*(*1*) *assms*(*2*) *sum-portfolio*)

lemma *sum-cls-val-process*:
assumes *portfolio pf1*
and *portfolio pf2*
shows ∀ *n w*. *cls-val-process Mkt* (*qty-sum pf1 pf2*) *n w* =
(*cls-val-process Mkt pf1*) *n w* + (*cls-val-process Mkt pf2*) *n w*

by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *sum-cls-val-process0 sum-cls-val-process-Suc up-cl-proc.elims*)

lemma *mult-comp-cls-val-process0*:
assumes *portfolio pf1*
shows ∀ *w*. *cls-val-process Mkt* (*qty-mult-comp pf1 qty*) *0 w* =
((*cls-val-process Mkt pf1*) *0 w*) ∗ (*qty* (*Suc 0*) *w*) **unfolding** *cls-val-process-def*
by (*simp add*: *assms mult-comp-portfolio mult-comp-val-process*)

lemma *mult-comp-cls-val-process-Suc*:
assumes *portfolio pf1*
shows ∀ *n w*. *cls-val-process Mkt* (*qty-mult-comp pf1 qty*) (*Suc n*) *w* =
((*cls-val-process Mkt pf1*) (*Suc n*) *w*) ∗ (*qty* (*Suc n*) *w*)
proof (*intro allI*)
fix *n w*
have *pf*:*portfolio* (*qty-mult-comp pf1 qty*) **using** *assms* **by** (*simp add*:*mult-comp-portfolio*)
have *fin*:*finite* (*support-set pf1*) **using** *assms* **unfolding** *portfolio-def* **by** *auto*
have ((*cls-val-process Mkt pf1*) (*Suc n*) *w*) ∗ (*qty* (*Suc n*) *w*) =
(∑ *x*∈ (*support-set pf1*). ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf1 x* (*Suc n*) *w*))∗(*qty* (*Suc n*) *w*)
unfolding *cls-val-process-def* **using** *assms* **by** *simp*
also have ... = (∑ *x*∈ (*support-set pf1*).
(((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf1 x* (*Suc n*) *w*) ∗ (*qty* (*Suc n*) *w*))) **using** *sum-distrib-right* **by** *auto*
also have ... = (∑ *x*∈ (*support-set pf1*).
((*prices Mkt*) *x* (*Suc n*) *w*) ∗ ((*qty-mult-comp pf1 qty*) *x* (*Suc n*) *w*)) **unfolding** *qty-mult-comp-def*
by (*simp add*: *mult.commute mult.left-commute*)
also have ... = (∑ *x*∈ (*support-set* (*qty-mult-comp pf1 qty*)).
((*prices Mkt*) *x* (*Suc n*) *w*) ∗ ((*qty-mult-comp pf1 qty*) *x* (*Suc n*) *w*)) **using** *mult-comp-support-set*[*of pf1 qty*]
subset-cls-val-process[*of support-set pf1 qty-mult-comp pf1 qty*] *pf fin up-cl-proc.simps*(*2*)
unfolding *cls-val-process-def* **by** *simp*
also have ... = *cls-val-process Mkt* (*qty-mult-comp pf1 qty*) (*Suc n*) *w* **by** (*metis* (*no-types*, *lifting*) *pf sum.cong cls-val-process-def up-cl-proc.simps*(*2*))
finally have (*cls-val-process Mkt pf1*) (*Suc n*) *w* ∗ (*qty* (*Suc n*) *w*) = *cls-val-process Mkt* (*qty-mult-comp pf1 qty*) (*Suc n*) *w* **.**
thus *cls-val-process Mkt* (*qty-mult-comp pf1 qty*) (*Suc n*) *w* = (*cls-val-process Mkt pf1*) (*Suc n*) *w* ∗ (*qty* (*Suc n*) *w*) **..**
qed

lemma *remove-comp-cls-val-process0*:
assumes *portfolio pf1*
shows ∀ *w*. *cls-val-process Mkt* (*qty-rem-comp pf1 y*) *0 w* =
((*cls-val-process Mkt pf1*) *0 w*) − (*prices Mkt y 0 w*)∗ (*pf1 y* (*Suc 0*) *w*) **unfolding** *cls-val-process-def*

by (*simp add*: *assms remove-comp-portfolio remove-comp-val-process*)

lemma *remove-comp-cls-val-process-Suc*:
assumes *portfolio pf1*
shows ∀ *n w*. *cls-val-process Mkt* (*qty-rem-comp pf1 y*) (*Suc n*) *w* =
((*cls-val-process Mkt pf1*) (*Suc n*) *w*) − (*prices Mkt y* (*Suc n*) *w*)∗ (*pf1 y* (*Suc n*) *w*)
proof (*intro allI*)
fix *n w*
have *pf*:*portfolio* (*qty-rem-comp pf1 y*) **using** *assms* **by** (*simp add*:*remove-comp-portfolio*)
have *fin*:*finite* (*support-set pf1*) **using** *assms* **unfolding** *portfolio-def* **by** *auto*
hence *fin2*: *finite* (*support-set pf1* − {*y*}) **by** *simp*
have ((*cls-val-process Mkt pf1*) (*Suc n*) *w*) =
(∑ *x*∈ (*support-set pf1*). ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf1 x* (*Suc n*) *w*))
unfolding *cls-val-process-def* **using** *assms* **by** *simp*
also have ... = (∑ *x*∈ (*support-set pf1* − {*y*}).
(((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf1 x* (*Suc n*) *w*))) + (*prices Mkt y* (*Suc n*) *w*)∗ (*pf1 y* (*Suc n*) *w*)
proof (*cases y*∈ *support-set pf1*)
case *True*
thus *?thesis* **by** (*simp add*: *fin sum-diff1*)
next
case *False*
hence *pf1 y* (*Suc n*) *w* = *0* **unfolding** *support-set-def* **by** *simp*
thus *?thesis* **by** (*simp add*: *fin sum-diff1*)
qed
also have ... = (∑ *x*∈ (*support-set pf1* − {*y*}).
((*prices Mkt*) *x* (*Suc n*) *w*) ∗ ((*qty-rem-comp pf1 y*) *x* (*Suc n*) *w*)) + (*prices Mkt y* (*Suc n*) *w*)∗ (*pf1 y* (*Suc n*) *w*)
proof −
have (∑ *x*∈ (*support-set pf1* − {*y*}). (((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*pf1 x* (*Suc n*) *w*))) =
(∑ *x*∈ (*support-set pf1* − {*y*}). ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ ((*qty-rem-comp pf1 y*) *x* (*Suc n*) *w*))
proof (*rule sum.cong*,*simp*)
fix *x*
assume *x*∈ *support-set pf1* − {*y*}
show *prices Mkt x* (*Suc n*) *w* ∗ *pf1 x* (*Suc n*) *w* = *prices Mkt x* (*Suc n*) *w* ∗ *qty-rem-comp pf1 y x* (*Suc n*) *w* **using** *remove-comp-values*
by (*metis DiffD2* ‹*x* ∈ *support-set pf1* − {*y*}› *singletonI*)
qed
thus *?thesis* **by** *simp*
qed
also have ... = (*cls-val-process Mkt* (*qty-rem-comp pf1 y*) (*Suc n*) *w*) + (*prices Mkt y* (*Suc n*) *w*)∗ (*pf1 y* (*Suc n*) *w*)
using *subset-cls-val-process*[*of support-set pf1* − {*y*} *qty-rem-comp pf1 y*] *fin2*
by (*simp add*: *remove-comp-support-set*)
finally have (*cls-val-process Mkt pf1*) (*Suc n*) *w* =

(*cls-val-process Mkt* (*qty-rem-comp pf1 y*) (*Suc n*) *w*) + (*prices Mkt y* (*Suc n*) *w*)∗ (*pf1 y* (*Suc n*) *w*) **.**
thus *cls-val-process Mkt* (*qty-rem-comp pf1 y*) (*Suc n*) *w* =
((*cls-val-process Mkt pf1*) (*Suc n*) *w*) − (*prices Mkt y* (*Suc n*) *w*)∗ (*pf1 y* (*Suc n*) *w*) **by** *simp*
qed

lemma *replace-comp-cls-val-process0*:
assumes ∀ *w*. *prices Mkt x 0 w* = *cls-val-process Mkt pf2 0 w*
and *portfolio pf1*
and *portfolio pf2*
shows ∀ *w*. *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) *0 w* = *cls-val-process Mkt pf1 0 w*
proof
fix *w*
have *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) *0 w* = *cls-val-process Mkt* (*qty-rem-comp pf1 x*) *0 w* +
cls-val-process Mkt (*qty-mult-comp pf2* (*pf1 x*)) *0 w* **unfolding** *qty-replace-comp-def*
using *assms*
sum-cls-val-process0[*of qty-rem-comp pf1 x qty-mult-comp pf2* (*pf1 x*)]
by (*simp add*: *mult-comp-portfolio remove-comp-portfolio*)
also have ... = *cls-val-process Mkt pf1 0 w* − (*prices Mkt x 0 w* ∗ *pf1 x* (*Suc 0*) *w*) +
cls-val-process Mkt pf2 0 w ∗ *pf1 x* (*Suc 0*) *w*
by (*simp add*: *assms*(*2*) *assms*(*3*) *mult-comp-cls-val-process0 remove-comp-cls-val-process0*)
also have ... = *cls-val-process Mkt pf1 0 w* **using** *assms* **by** *simp*
finally show *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) *0 w* = *cls-val-process Mkt pf1 0 w* **.**
qed

lemma *replace-comp-cls-val-process-Suc*:
assumes ∀ *n w*. *prices Mkt x* (*Suc n*) *w* = *cls-val-process Mkt pf2* (*Suc n*) *w*
and *portfolio pf1*
and *portfolio pf2*
shows ∀ *n w*. *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) (*Suc n*) *w* = *cls-val-process Mkt pf1* (*Suc n*) *w*
proof (*intro allI*)
fix *n w*
have *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) (*Suc n*) *w* = *cls-val-process Mkt* (*qty-rem-comp pf1 x*) (*Suc n*) *w* +
cls-val-process Mkt (*qty-mult-comp pf2* (*pf1 x*)) (*Suc n*) *w* **unfolding** *qty-replace-comp-def*
using *assms*
sum-cls-val-process-Suc[*of qty-rem-comp pf1 x qty-mult-comp pf2* (*pf1 x*)]
by (*simp add*: *mult-comp-portfolio remove-comp-portfolio*)
also have ... = *cls-val-process Mkt pf1* (*Suc n*) *w* − (*prices Mkt x* (*Suc n*) *w* ∗ *pf1 x* (*Suc n*) *w*) +

*cls-val-process Mkt pf2 (Suc n) w * pf1 x (Suc n) w*
by (*simp add*: *assms(2) assms(3) mult-comp-cls-val-process-Suc remove-comp-cls-val-process-Suc*)
also have ... = *cls-val-process Mkt pf1 (Suc n) w* **using** *assms* **by** *simp*
finally show *cls-val-process Mkt (qty-replace-comp pf1 x pf2) (Suc n) w = cls-val-process Mkt pf1 (Suc n) w* **.**
qed

lemma *replace-comp-cls-val-process*:
assumes ∀ *n w*. *prices Mkt x n w = cls-val-process Mkt pf2 n w*
and *portfolio pf1*
and *portfolio pf2*
shows ∀ *n w*. *cls-val-process Mkt (qty-replace-comp pf1 x pf2) n w = cls-val-process Mkt pf1 n w*
by (*metis* (*no-types*, *lifting*) *assms replace-comp-cls-val-process0 replace-comp-cls-val-process-Suc up-cl-proc.elims*)

lemma *qty-single-updated*:
shows *cls-val-process Mkt (qty-single asset qty) (Suc n) w = prices Mkt asset (Suc n) w * qty (Suc n) w*
proof −
have *cls-val-process Mkt (qty-single asset qty) (Suc n) w = (sum (λx. ((prices Mkt) x (Suc n) w) * ((qty-single asset qty) x (Suc n) w)) {asset})*
proof (*rule subset-cls-val-process′*)
show *finite* {*asset*} **by** *simp*
show *support-set (qty-single asset qty)* ⊆ {*asset*} **by** (*simp add*: *single-comp-support*)
qed
also have ... = *prices Mkt asset (Suc n) w * qty (Suc n) w* **unfolding** *qty-single-def* **by** *simp*
finally show *?thesis* **.**
qed

Self-financing **definition** *self-financing* **where**
self-financing Mkt p ⟷ (∀ *n*. *val-process Mkt p (Suc n) = cls-val-process Mkt p (Suc n)*)

lemma *self-financingE*:
assumes *self-financing Mkt p*
shows ∀ *n*. *val-process Mkt p n = cls-val-process Mkt p n*
proof
fix *n*
show *val-process Mkt p n = cls-val-process Mkt p n*
proof (*cases n = 0*)
case *False*
thus *?thesis* **using** *assms* **unfolding** *self-financing-def*
by (*metis up-cl-proc.elims*)

```
  next
    case True
    thus ?thesis by (simp add: cls-val-process-def val-process-def)
  qed
qed



lemma static-portfolio-self-financing:
  assumes ∀ x ∈ support-set p. (∀ w i. p x i w = p x (Suc i) w)
  shows self-financing Mkt p
unfolding self-financing-def
proof (intro allI impI)
  fix n
  show val-process Mkt p (Suc n) = cls-val-process Mkt p (Suc n)
  proof (cases portfolio p)
    case False
    thus ?thesis unfolding val-process-def cls-val-process-def by simp
  next
    case True
    have ∀ w. (∑ x∈ support-set p. prices Mkt x (Suc n) w * p x (Suc (Suc n)) w)
=
        cls-val-process Mkt p (Suc n) w
    proof
      fix w
      show (∑ x∈ support-set p. prices Mkt x (Suc n) w * p x (Suc (Suc n)) w) =
          cls-val-process Mkt p (Suc n) w
      proof −
        have (∑ x∈ support-set p. prices Mkt x (Suc n) w * p x (Suc (Suc n)) w) =
            (∑ x∈ support-set p. prices Mkt x (Suc n) w * p x (Suc n) w)
        proof (rule sum.cong, simp)
          fix x
          assume x∈ support-set p
          hence p x (Suc n) w = p x (Suc (Suc n)) w using assms by blast
          thus prices Mkt x (Suc n) w * p x (Suc (Suc n)) w = prices Mkt x (Suc
n) w * p x (Suc n) w by simp
        qed
        also have ... = cls-val-process Mkt p (Suc n) w
          using up-cl-proc.simps(2)[of Mkt p n] by (metis True cls-val-process-def)
        finally show ?thesis .
      qed
    qed
    moreover have ∀ w. val-process Mkt p (Suc n) w = (∑ x∈ support-set p. prices
Mkt x (Suc n) w * p x (Suc (Suc n)) w)
      unfolding val-process-def using True by simp
    ultimately show ?thesis by auto
  qed
qed
```

lemma *sum-self-financing*:
assumes *portfolio pf1*
and *portfolio pf2*
and *self-financing Mkt pf1*
and *self-financing Mkt pf2*
shows *self-financing Mkt* (*qty-sum pf1 pf2*)
proof −
have ∀ *n w*. *val-process Mkt* (*qty-sum pf1 pf2*) (*Suc n*) *w* =
cls-val-process Mkt (*qty-sum pf1 pf2*) (*Suc n*) *w*
proof (*intro allI*)
fix *n w*
have *val-process Mkt* (*qty-sum pf1 pf2*) (*Suc n*) *w* = *val-process Mkt pf1* (*Suc n*) *w* + *val-process Mkt pf2* (*Suc n*) *w*
using *assms* **by** (*simp add*:*sum-val-process*)
also have ... = *cls-val-process Mkt pf1* (*Suc n*) *w* + *val-process Mkt pf2* (*Suc n*) *w* **using** *assms*
unfolding *self-financing-def* **by** *simp*
also have ... = *cls-val-process Mkt pf1* (*Suc n*) *w* + *cls-val-process Mkt pf2* (*Suc n*) *w*
using *assms* **unfolding** *self-financing-def* **by** *simp*
also have ... = *cls-val-process Mkt* (*qty-sum pf1 pf2*) (*Suc n*) *w* **using** *assms* **by** (*simp add*: *sum-cls-val-process*)
finally show *val-process Mkt* (*qty-sum pf1 pf2*) (*Suc n*) *w* =
cls-val-process Mkt (*qty-sum pf1 pf2*) (*Suc n*) *w* **.**
qed
thus *?thesis* **unfolding** *self-financing-def* **by** *auto*
qed

lemma *mult-time-constant-self-financing*:
assumes *portfolio pf1*
and *self-financing Mkt pf1*
and ∀ *n w*. *qty n w* = *qty* (*Suc n*) *w*
shows *self-financing Mkt* (*qty-mult-comp pf1 qty*)
proof −
have ∀ *n w*. *val-process Mkt* (*qty-mult-comp pf1 qty*) (*Suc n*) *w* =
cls-val-process Mkt (*qty-mult-comp pf1 qty*) (*Suc n*) *w*
proof (*intro allI*)
fix *n w*
have *val-process Mkt* (*qty-mult-comp pf1 qty*) (*Suc n*) *w* = *val-process Mkt pf1* (*Suc n*) *w* ∗ *qty* (*Suc n*) *w*
using *assms* **by** (*simp add*:*mult-comp-val-process*)
also have ... = *cls-val-process Mkt pf1* (*Suc n*) *w* ∗ *qty* (*Suc n*) *w* **using** *assms*
unfolding *self-financing-def* **by** *simp*
also have ... = *cls-val-process Mkt* (*qty-mult-comp pf1 qty*) (*Suc n*) *w* **using** *assms*
by (*auto simp add*: *mult-comp-cls-val-process-Suc*)

finally show *val-process Mkt* (*qty-mult-comp pf1 qty*) (*Suc n*) *w* =
cls-val-process Mkt (*qty-mult-comp pf1 qty*) (*Suc n*) *w* .
qed
thus *?thesis* **unfolding** *self-financing-def* **by** *auto*
qed

lemma *replace-comp-self-financing*:
assumes ∀ *n w*. *prices Mkt x n w* = *cls-val-process Mkt pf2 n w*
and *portfolio pf1*
and *portfolio pf2*
and *self-financing Mkt pf1*
and *self-financing Mkt pf2*
shows *self-financing Mkt* (*qty-replace-comp pf1 x pf2*)
proof −
have *sfeq*: ∀ *n w*. *prices Mkt x n w* = *val-process Mkt pf2 n w* **using** *assms* **by** (*simp add*: *self-financingE*)
have ∀ *n w*. *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) (*Suc n*) *w* =
val-process Mkt (*qty-replace-comp pf1 x pf2*) (*Suc n*) *w*
proof (*intro allI*)
fix *n w*
have *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) (*Suc n*) *w* = *cls-val-process Mkt pf1* (*Suc n*) *w*
using *assms* **by** (*simp add*:*replace-comp-cls-val-process*)
also have ... = *val-process Mkt pf1* (*Suc n*) *w* **using** *assms* **unfolding** *self-financing-def* **by** *simp*
also have ... = *val-process Mkt* (*qty-replace-comp pf1 x pf2*) (*Suc n*) *w*
using *assms sfeq* **by** (*simp add*: *replace-comp-val-process self-financing-def*)
finally show *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) (*Suc n*) *w* =
val-process Mkt (*qty-replace-comp pf1 x pf2*) (*Suc n*) *w* .
qed
thus *?thesis* **unfolding** *self-financing-def* **by** *auto*
qed

Make a portfolio self-financing **fun** *remaining-qty* **where**
init: *remaining-qty Mkt v pf asset 0* = (λ*w*. *0*) |
first: *remaining-qty Mkt v pf asset* (*Suc 0*) = (λ*w*. (*v* − *val-process Mkt pf 0 w*)/(*prices Mkt asset 0 w*)) |
step: *remaining-qty Mkt v pf asset* (*Suc* (*Suc n*)) = (λ*w*. (*remaining-qty Mkt v pf asset* (*Suc n*) *w*) +
(*cls-val-process Mkt pf* (*Suc n*) *w* − *val-process Mkt pf* (*Suc n*) *w*)/(*prices Mkt asset* (*Suc n*) *w*))

lemma (**in** *disc-equity-market*) *remaining-qty-predict′*:
assumes *borel-adapt-stoch-proc F* (*prices Mkt asset*)
and *trading-strategy pf*
and *support-adapt Mkt pf*
shows *remaining-qty Mkt v pf asset* (*Suc n*) ∈ *borel-measurable* (*F n*)

proof (*induct n*)
case *0*
have (λ*w*. (*v* − *val-process Mkt pf 0 w*)/(*prices Mkt asset 0 w*))∈ *borel-measurable* (*F 0*)
proof (*rule borel-measurable-divide*)
have *val-process Mkt pf 0* ∈ *borel-measurable* (*F 0*) **using** *assms ats-val-process-adapted* **by** (*simp add*:*adapt-stoch-proc-def*)
thus (λ*x*. *v* − *val-process Mkt pf 0 x*) ∈ *borel-measurable* (*F 0*) **by** *simp*
show *prices Mkt asset 0* ∈ *borel-measurable* (*F 0*) **using** *assms* **unfolding** *adapt-stoch-proc-def* **by** *simp*
qed
thus *?case* **by** *simp*
next
case (*Suc n*)
have (λ*w*. (*cls-val-process Mkt pf* (*Suc n*) *w* − *val-process Mkt pf* (*Suc n*) *w*)/ (*prices Mkt asset* (*Suc n*) *w*)) ∈ *borel-measurable* (*F* (*Suc n*))
proof (*rule borel-measurable-divide*)
show (λ*w*. (*cls-val-process Mkt pf* (*Suc n*) *w* − *val-process Mkt pf* (*Suc n*) *w*)) ∈ *borel-measurable* (*F* (*Suc n*))
proof (*rule borel-measurable-diff*)
show (λ*w*. (*cls-val-process Mkt pf* (*Suc n*) *w*)) ∈ *borel-measurable* (*F* (*Suc n*))
using *assms cls-val-process-adapted* **unfolding** *adapt-stoch-proc-def* **by** *auto*
show (λ*w*. (*val-process Mkt pf* (*Suc n*) *w*)) ∈ *borel-measurable* (*F* (*Suc n*))
using *assms ats-val-process-adapted* **by** (*simp add*:*adapt-stoch-proc-def*)
qed
show *prices Mkt asset* (*Suc n*) ∈ *borel-measurable* (*F* (*Suc n*)) **using** *assms* **unfolding** *adapt-stoch-proc-def* **by** *simp*
qed
moreover have *remaining-qty Mkt v pf asset* (*Suc n*) ∈ *borel-measurable* (*F* (*Suc n*)) **using** *Suc*
Suc-n-not-le-n increasing-measurable-info nat-le-linear **by** *blast*
ultimately show *?case* **using** *Suc remaining-qty.simps*(*3*)[*of Mkt v pf asset n*] **by** *simp*
qed

lemma (**in** *disc-equity-market*) *remaining-qty-predict*:
assumes *borel-adapt-stoch-proc F* (*prices Mkt asset*)
and *trading-strategy pf*
and *support-adapt Mkt pf*
shows *borel-predict-stoch-proc F* (*remaining-qty Mkt v pf asset*) **unfolding** *predict-stoch-proc-def*
proof (*intro conjI allI*)
show *remaining-qty Mkt v pf asset 0* ∈ *borel-measurable* (*F 0*) **using** *init* **by** *simp*
fix *n*
show *remaining-qty Mkt v pf asset* (*Suc n*) ∈ *borel-measurable* (*F n*) **using** *assms* **by** (*simp add*: *remaining-qty-predict*′)
qed

lemma (**in** *disc-equity-market*) *remaining-qty-adapt*:
assumes *borel-adapt-stoch-proc F* (*prices Mkt asset*)
and *trading-strategy pf*
and *support-adapt Mkt pf*
shows *remaining-qty Mkt v pf asset n* ∈ *borel-measurable* (*F n*)
using *adapt-stoch-proc-def assms*(*1*) *assms*(*2*) *predict-imp-adapt remaining-qty-predict*
by (*metis assms*(*3*))

lemma (**in** *disc-equity-market*) *remaining-qty-adapted*:
assumes *borel-adapt-stoch-proc F* (*prices Mkt asset*)
and *trading-strategy pf*
and *support-adapt Mkt pf*
shows *borel-adapt-stoch-proc F* (*remaining-qty Mkt v pf asset*) **using** *assms* **unfolding** *adapt-stoch-proc-def*
using *assms remaining-qty-adapt* **by** *blast*

definition *self-finance* **where**
self-finance Mkt v pf (*asset*::′*a*) = *qty-sum pf* (*qty-single asset* (*remaining-qty Mkt v pf asset*))

lemma *self-finance-portfolio*:
assumes *portfolio pf*
shows *portfolio* (*self-finance Mkt v pf asset*) **unfolding** *self-finance-def*
by (*simp add*: *assms single-comp-portfolio sum-portfolio*)

lemma *self-finance-init*:
assumes ∀ *w*. *prices Mkt asset 0 w* ≠ *0*
and *portfolio pf*
shows *val-process Mkt* (*self-finance Mkt v pf asset*) *0 w* = *v*
proof −
define *scp* **where** *scp* = *qty-single asset* (*remaining-qty Mkt v pf asset*)
have *val-process Mkt* (*self-finance Mkt v pf asset*) *0 w* =
val-process Mkt pf 0 w +
val-process Mkt scp 0 w **unfolding** *scp-def* **using** *assms single-comp-portfolio*[*of asset*]
sum-val-process[*of pf qty-single asset* (*remaining-qty Mkt v pf asset*) *Mkt*]
by (*simp add*: ‹⋀*qty*. *portfolio* (*qty-single asset qty*)› *self-finance-def*)
also have ... = *val-process Mkt pf 0 w* +
(*sum* (λ*x*. ((*prices Mkt*) *x 0 w*) ∗ (*scp x* (*Suc 0*) *w*)) {*asset*})
using *subset-val-process*′[*of* {*asset*} *scp*] **unfolding** *scp-def* **by** (*auto simp add*: *single-comp-support*)
also have ... = *val-process Mkt pf 0 w* + *prices Mkt asset 0 w* ∗ *scp asset* (*Suc 0*) *w* **by** *auto*
also have ... = *val-process Mkt pf 0 w* + *prices Mkt asset 0 w* ∗ (*remaining-qty*

Mkt v pf asset) (*Suc 0*) *w*
unfolding *scp-def qty-single-def* **by** *simp*
also have ... = *val-process Mkt pf 0 w* + *prices Mkt asset 0 w* ∗ (*v* − *val-process Mkt pf 0 w*)/(*prices Mkt asset 0 w*)
by *simp*
also have ... = *val-process Mkt pf 0 w* + (*v* − *val-process Mkt pf 0 w*) **using** *assms* **by** *simp*
also have ... = *v* **by** *simp*
finally show *?thesis* **.**
qed

lemma *self-finance-succ*:
assumes *prices Mkt asset* (*Suc n*) *w* ≠ *0*
and *portfolio pf*
shows *val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* = *prices Mkt asset* (*Suc n*) *w* ∗ *remaining-qty Mkt v pf asset* (*Suc n*) *w* +
cls-val-process Mkt pf (*Suc n*) *w*
proof −
define *scp* **where** *scp* = *qty-single asset* (*remaining-qty Mkt v pf asset*)
have *val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* =
val-process Mkt pf (*Suc n*) *w* +
val-process Mkt scp (*Suc n*) *w* **unfolding** *scp-def* **using** *assms single-comp-portfolio*[*of asset*]
sum-val-process[*of pf qty-single asset* (*remaining-qty Mkt v pf asset*) *Mkt*]
by (*simp add*: ‹⋀*qty. portfolio* (*qty-single asset qty*)› *self-finance-def*)
also have ... = *val-process Mkt pf* (*Suc n*) *w* +
(*sum* (λ*x*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*scp x* (*Suc* (*Suc n*)) *w*)) {*asset*})
using *subset-val-process′*[*of* {*asset*} *scp*] **unfolding** *scp-def* **by** (*auto simp add*: *single-comp-support*)
also have ... = *val-process Mkt pf* (*Suc n*) *w* + *prices Mkt asset* (*Suc n*) *w* ∗ *scp asset* (*Suc* (*Suc n*)) *w* **by** *auto*
also have ... = *val-process Mkt pf* (*Suc n*) *w* + *prices Mkt asset* (*Suc n*) *w* ∗ (*remaining-qty Mkt v pf asset*) (*Suc* (*Suc n*)) *w*
unfolding *scp-def qty-single-def* **by** *simp*
also have ... = *val-process Mkt pf* (*Suc n*) *w* +
prices Mkt asset (*Suc n*) *w* ∗
(*remaining-qty Mkt v pf asset* (*Suc n*) *w* + (*cls-val-process Mkt pf* (*Suc n*) *w* − *val-process Mkt pf* (*Suc n*) *w*)/(*prices Mkt asset* (*Suc n*) *w*))
by *simp*
also have ... = *val-process Mkt pf* (*Suc n*) *w* +
prices Mkt asset (*Suc n*) *w* ∗ *remaining-qty Mkt v pf asset* (*Suc n*) *w* +
prices Mkt asset (*Suc n*) *w* ∗ (*cls-val-process Mkt pf* (*Suc n*) *w* − *val-process Mkt pf* (*Suc n*) *w*)/(*prices Mkt asset* (*Suc n*) *w*)
by (*simp add*: *distrib-left*)
also have ... = *val-process Mkt pf* (*Suc n*) *w* +
prices Mkt asset (*Suc n*) *w* ∗ *remaining-qty Mkt v pf asset* (*Suc n*) *w* + (*cls-val-process Mkt pf* (*Suc n*) *w* − *val-process Mkt pf* (*Suc n*) *w*)
using *assms* **by** *simp*

also have ... = *prices Mkt asset* (*Suc n*) *w* ∗ *remaining-qty Mkt v pf asset* (*Suc n*) *w* + *cls-val-process Mkt pf* (*Suc n*) *w* **by** *simp*
finally show *?thesis* **.**
qed

lemma *self-finance-updated*:
assumes *prices Mkt asset* (*Suc n*) *w* ≠ *0*
and *portfolio pf*
shows *cls-val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* =
cls-val-process Mkt pf (*Suc n*) *w* + *prices Mkt asset* (*Suc n*) *w* ∗ (*remaining-qty Mkt v pf asset*) (*Suc n*) *w*
proof −
define *scp* **where** *scp* = *qty-single asset* (*remaining-qty Mkt v pf asset*)
have *cls-val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* =
cls-val-process Mkt pf (*Suc n*) *w* +
cls-val-process Mkt scp (*Suc n*) *w* **unfolding** *scp-def* **using** *assms single-comp-portfolio*[*of asset*]
sum-cls-val-process[*of pf qty-single asset* (*remaining-qty Mkt v pf asset*) *Mkt*]
by (*simp add*: ‹⋀*qty. portfolio* (*qty-single asset qty*)› *self-finance-def*)
also have ... = *cls-val-process Mkt pf* (*Suc n*) *w* +
(*sum* (λ*x*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*scp x* (*Suc n*) *w*)) {*asset*})
using *subset-cls-val-process*[*of* {*asset*} *scp*] **unfolding** *scp-def* **by** (*auto simp add*: *single-comp-support*)
also have ... = *cls-val-process Mkt pf* (*Suc n*) *w* + *prices Mkt asset* (*Suc n*) *w* ∗ *scp asset* (*Suc n*) *w* **by** *auto*
also have ... = *cls-val-process Mkt pf* (*Suc n*) *w* + *prices Mkt asset* (*Suc n*) *w* ∗ (*remaining-qty Mkt v pf asset*) (*Suc n*) *w*
unfolding *scp-def qty-single-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *self-finance-charact*:
assumes ∀ *n w. prices Mkt asset* (*Suc n*) *w* ≠ *0*
and *portfolio pf*
shows *self-financing Mkt* (*self-finance Mkt v pf asset*)
proof−
have ∀ *n w. val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* =
cls-val-process Mkt (*self-finance Mkt v pf asset*) (*Suc n*) *w*
proof (*intro allI*)
fix *n w*
show *val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* =
cls-val-process Mkt (*self-finance Mkt v pf asset*) (*Suc n*) *w* **using** *assms self-finance-succ*[*of Mkt asset*]
by (*simp add*: *self-finance-updated*)
qed
thus *?thesis* **unfolding** *self-financing-def* **by** *auto*
qed

7.2.5 Replicating portfolios

definition (**in** *disc-filtr-prob-space*) *price-structure*::('*a* ⇒ *real*) ⇒ *nat* ⇒ *real* ⇒ (*nat* ⇒ '*a* ⇒ *real*) ⇒ *bool* **where**
price-structure pyf T π pr ⟷ ((∀ *w*∈ *space M*. *pr 0 w* = *π*) ∧ (*AE w in M*. *pr T w* = *pyf w*) ∧ (*pr T* ∈ *borel-measurable* (*F T*)))

lemma (**in** *disc-filtr-prob-space*) *price-structure-init*:
assumes *price-structure pyf T π pr*
shows ∀ *w*∈ *space M*. *pr 0 w* = *π* **using** *assms* **unfolding** *price-structure-def* **by** *simp*

lemma (**in** *disc-filtr-prob-space*) *price-structure-borel-measurable*:
assumes *price-structure pyf T π pr*
shows *pr T* ∈ *borel-measurable* (*F T*) **using** *assms* **unfolding** *price-structure-def* **by** *simp*

lemma (**in** *disc-filtr-prob-space*) *price-structure-maturity*:
assumes *price-structure pyf T π pr*
shows *AE w in M*. *pr T w* = *pyf w* **using** *assms* **unfolding** *price-structure-def* **by** *simp*

definition (**in** *disc-equity-market*) *replicating-portfolio* **where**
replicating-portfolio pf der matur ⟷ (*stock-portfolio Mkt pf*) ∧ (*trading-strategy pf*) ∧ (*self-financing Mkt pf*) ∧
(*AE w in M*. *cls-val-process Mkt pf matur w* = *der w*)

definition (**in** *disc-equity-market*) *is-attainable* **where**
is-attainable der matur ⟷ (∃ *pf*. *replicating-portfolio pf der matur*)

lemma (**in** *disc-equity-market*) *replicating-price-process*:
assumes *replicating-portfolio pf der matur*
and *support-adapt Mkt pf*
shows *price-structure der matur* (*initial-value pf*) (*cls-val-process Mkt pf*)
unfolding *price-structure-def*
proof (*intro conjI*)
show *AE w in M*. *cls-val-process Mkt pf matur w* = *der w* **using** *assms* **unfolding** *replicating-portfolio-def* **by** *simp*
show ∀ *w*∈*space M*. *cls-val-process Mkt pf 0 w* = *initial-value pf*
proof
fix *w*
assume *w*∈ *space M*
thus *cls-val-process Mkt pf 0 w* = *initial-value pf* **unfolding** *initial-value-def* **using** *constant-imageI*[*of cls-val-process Mkt pf 0*]
trading-strategy-init[*of pf*] *assms replicating-portfolio-def* [*of pf der matur*]
by (*simp add*: *stock-portfolio-def cls-val-process-def*)
qed
show *cls-val-process Mkt pf matur* ∈ *borel-measurable* (*F matur*) **using** *assms* **unfolding** *replicating-portfolio-def*

using *ats-val-process-adapted*[*of pf*]
cls-val-process-adapted **by** (*simp add*:*adapt-stoch-proc-def*)
qed

7.2.6 Arbitrages

definition (**in** *disc-filtr-prob-space*) *arbitrage-process*
where
arbitrage-process Mkt p ⟷ (∃ *m*. (*self-financing Mkt p*) ∧ (*trading-strategy p*)
∧
(∀ *w* ∈ *space M*. *val-process Mkt p 0 w* = *0*) ∧
(*AE w in M*. *0* ≤ *cls-val-process Mkt p m w*) ∧
0 < *P*(*w in M*. *cls-val-process Mkt p m w* > *0*))

lemma (**in** *disc-filtr-prob-space*) *arbitrage-processE*:
assumes *arbitrage-process Mkt p*
shows (∃ *m*. (*self-financing Mkt p*) ∧ (*trading-strategy p*) ∧
(∀ *w* ∈ *space M*. *cls-val-process Mkt p 0 w* = *0*) ∧
(*AE w in M*. *0* ≤ *cls-val-process Mkt p m w*) ∧
0 < *P*(*w in M*. *cls-val-process Mkt p m w* > *0*))
using *assms disc-filtr-prob-space.arbitrage-process-def disc-filtr-prob-space-axioms self-financingE* **by** *fastforce*

lemma (**in** *disc-filtr-prob-space*) *arbitrage-processI*:
assumes (∃ *m*. (*self-financing Mkt p*) ∧ (*trading-strategy p*) ∧
(∀ *w* ∈ *space M*. *cls-val-process Mkt p 0 w* = *0*) ∧
(*AE w in M*. *0* ≤ *cls-val-process Mkt p m w*) ∧
0 < *P*(*w in M*. *cls-val-process Mkt p m w* > *0*))
shows *arbitrage-process Mkt p* **unfolding** *arbitrage-process-def* **using** *assms*
by (*simp add*: *self-financingE cls-val-process-def*)

definition (**in** *disc-filtr-prob-space*) *viable-market*
where
viable-market Mkt ⟷ (∀ *p*. *stock-portfolio Mkt p* ⟶ ¬ *arbitrage-process Mkt p*)

lemma (**in** *disc-filtr-prob-space*) *arbitrage-val-process*:
assumes *arbitrage-process Mkt pf1*
and *self-financing Mkt pf2*
and *trading-strategy pf2*
and ∀ *n w*. *cls-val-process Mkt pf1 n w* = *cls-val-process Mkt pf2 n w*
shows *arbitrage-process Mkt pf2*
proof −
have (∃ *m*. (*self-financing Mkt pf1*) ∧ (*trading-strategy pf1*) ∧
(∀ *w* ∈ *space M*. *cls-val-process Mkt pf1 0 w* = *0*) ∧
(*AE w in M*. *0* ≤ *cls-val-process Mkt pf1 m w*) ∧
0 < *P*(*w in M*. *cls-val-process Mkt pf1 m w* > *0*)) **using** *assms arbitrage-processE*[*of*

Mkt pf1] **by** *simp*
from *this* **obtain** *m* **where** (*self-financing Mkt pf1*) **and** (*trading-strategy pf1*) **and**
(∀ *w* ∈ *space M*. *cls-val-process Mkt pf1 0 w* = *0*) **and**
(*AE w in M*. *0* ≤ *cls-val-process Mkt pf1 m w*)
0 < *P*(*w in M*. *cls-val-process Mkt pf1 m w* > *0*) **by** *auto*
have *ae-eq*:∀ *w*∈ *space M*. (*cls-val-process Mkt pf1 0 w* = *0*) = (*cls-val-process Mkt pf2 0 w* = *0*)
proof
fix *w*
assume *w*∈ *space M*
show (*cls-val-process Mkt pf1 0 w* = *0*) = (*cls-val-process Mkt pf2 0 w* = *0*)
using *assms* **by** *simp*
qed
have *ae-geq*:*almost-everywhere M* (λ*w*. *cls-val-process Mkt pf1 m w* ≥ *0*) = *almost-everywhere M* (λ*w*. *cls-val-process Mkt pf2 m w* ≥ *0*)
proof (*rule AE-cong*)
fix *w*
assume *w*∈ *space M*
show (*cls-val-process Mkt pf1 m w* ≥ *0*) = (*cls-val-process Mkt pf2 m w* ≥ *0*)
using *assms* **by** *simp*
qed
have *self-financing Mkt pf2* **using** *assms* **by** *simp*
moreover have *trading-strategy pf2* **using** *assms* **by** *simp*
moreover have (∀ *w* ∈ *space M*. *cls-val-process Mkt pf2 0 w* = *0*) **using** ‹(∀ *w* ∈ *space M*. *cls-val-process Mkt pf1 0 w* = *0*)› *ae-eq* **by** *simp*
moreover have *AE w in M*. *0* ≤ *cls-val-process Mkt pf2 m w* **using** ‹*AE w in M*. *0* ≤ *cls-val-process Mkt pf1 m w*› *ae-geq* **by** *simp*
moreover have *0* < *prob* {*w* ∈ *space M*. *0* < *cls-val-process Mkt pf2 m w*}
proof −
have {*w* ∈ *space M*. *0* < *cls-val-process Mkt pf2 m w*} = {*w* ∈ *space M*. *0* < *cls-val-process Mkt pf1 m w*}
by (*simp add*: *assms*(*4*))
thus *?thesis* **by** (*simp add*: ‹*0* < *prob* {*w* ∈ *space M*. *0* < *cls-val-process Mkt pf1 m w*}›)
qed
ultimately show *?thesis* **using** *arbitrage-processI* **by** *blast*
qed

definition *coincides-on* **where**
coincides-on Mkt Mkt2 A ⟷ (*stocks Mkt* = *stocks Mkt2* ∧ (∀ *x*. *x*∈ *A* ⟶ *prices Mkt x* = *prices Mkt2 x*))

lemma *coincides-val-process*:
assumes *coincides-on Mkt Mkt2 A*
and *support-set pf* ⊆ *A*
shows ∀ *n w*. *val-process Mkt pf n w* = *val-process Mkt2 pf n w*
proof (*intro allI*)

fix *n w*
show *val-process Mkt pf n w = val-process Mkt2 pf n w*
proof (*cases portfolio pf*)
case *False*
thus *?thesis* **unfolding** *val-process-def* **by** *simp*
next
case *True*
hence *val-process Mkt pf n w* = (∑ *x*∈ *support-set pf*. *prices Mkt x n w* ∗ *pf x* (*Suc n*) *w*) **using** *assms*
unfolding *val-process-def* **by** *simp*
also have ... = (∑ *x*∈ *support-set pf*. *prices Mkt2 x n w* ∗ *pf x* (*Suc n*) *w*)
proof (*rule sum.cong*, *simp*)
fix *y*
assume *y*∈ *support-set pf*
hence *y*∈ *A* **using** *assms* **unfolding** *stock-portfolio-def* **by** *auto*
hence *prices Mkt y n w = prices Mkt2 y n w* **using** *assms*
unfolding *coincides-on-def* **by** *auto*
thus *prices Mkt y n w* ∗ *pf y* (*Suc n*) *w = prices Mkt2 y n w* ∗ *pf y* (*Suc n*) *w* **by** *simp*
qed
also have ... = *val-process Mkt2 pf n w*
by (*metis* (*mono-tags*, *lifting*) *calculation val-process-def*)
finally show *val-process Mkt pf n w = val-process Mkt2 pf n w* **.**
qed
qed

lemma *coincides-cls-val-process′*:
assumes *coincides-on Mkt Mkt2 A*
and *support-set pf* ⊆ *A*
shows ∀ *n w*. *cls-val-process Mkt pf* (*Suc n*) *w = cls-val-process Mkt2 pf* (*Suc n*) *w*
proof (*intro allI*)
fix *n w*
show *cls-val-process Mkt pf* (*Suc n*) *w = cls-val-process Mkt2 pf* (*Suc n*) *w*
proof (*cases portfolio pf*)
case *False*
thus *?thesis* **unfolding** *cls-val-process-def* **by** *simp*
next
case *True*
hence *cls-val-process Mkt pf* (*Suc n*) *w* = (∑ *x*∈ *support-set pf*. *prices Mkt x* (*Suc n*) *w* ∗ *pf x* (*Suc n*) *w*) **using** *assms*
unfolding *cls-val-process-def* **by** *simp*
also have ... = (∑ *x*∈ *support-set pf*. *prices Mkt2 x* (*Suc n*) *w* ∗ *pf x* (*Suc n*) *w*)
proof (*rule sum.cong*, *simp*)
fix *y*
assume *y*∈ *support-set pf*
hence *y*∈ *A* **using** *assms* **unfolding** *stock-portfolio-def* **by** *auto*
hence *prices Mkt y* (*Suc n*) *w = prices Mkt2 y* (*Suc n*) *w* **using** *assms*

```
      unfolding coincides-on-def by auto
    thus prices Mkt y (Suc n) w * pf y (Suc n) w = prices Mkt2 y (Suc n) w *
pf y (Suc n) w by simp
  qed
  also have ... = cls-val-process Mkt2 pf (Suc n) w
    by (metis (mono-tags, lifting) True up-cl-proc.simps(2) cls-val-process-def)
  finally show cls-val-process Mkt pf (Suc n) w = cls-val-process Mkt2 pf (Suc
n) w .
 qed
qed

lemma coincides-cls-val-process:
 assumes coincides-on Mkt Mkt2 A
 and support-set pf ⊆ A
 shows ∀ n w. cls-val-process Mkt pf n w = cls-val-process Mkt2 pf n w
proof (intro allI)
 fix n w
 show cls-val-process Mkt pf n w = cls-val-process Mkt2 pf n w
 proof (cases portfolio pf)
  case False
  thus ?thesis unfolding cls-val-process-def by simp
 next
  case True
  show cls-val-process Mkt pf n w = cls-val-process Mkt2 pf n w
  proof (induct n)
   case 0
   with assms show ?case
    by (simp add: cls-val-process-def coincides-val-process)
  next
   case Suc
   thus ?case using coincides-cls-val-process' assms by blast
  qed
 qed
qed


lemma (in disc-filtr-prob-space) coincides-on-self-financing:
 assumes coincides-on Mkt Mkt2 A
 and support-set p ⊆ A
 and self-financing Mkt p
shows self-financing Mkt2 p
proof −
 have ∀ n w. val-process Mkt2 p (Suc n) w = cls-val-process Mkt2 p (Suc n) w
 proof (intro allI)
  fix n w
  have val-process Mkt2 p (Suc n) w = val-process Mkt p (Suc n) w
   using assms(1) assms(2) coincides-val-process stock-portfolio-def by fastforce
  also have ... = cls-val-process Mkt p (Suc n) w by (metis ‹self-financing Mkt
p› self-financing-def)
```

```
    also have ... = cls-val-process Mkt2 p (Suc n) w
     using assms(1) assms(2) coincides-cls-val-process stock-portfolio-def by blast
    finally show val-process Mkt2 p (Suc n) w = cls-val-process Mkt2 p (Suc n) w
.
  qed
  thus self-financing Mkt2 p unfolding self-financing-def by auto
qed


lemma (in disc-filtr-prob-space) coincides-on-arbitrage:
  assumes coincides-on Mkt Mkt2 A
  and support-set p ⊆ A
  and arbitrage-process Mkt p
shows arbitrage-process Mkt2 p
proof −
  have (∃ m. (self-financing Mkt p) ∧ (trading-strategy p) ∧
    (∀ w∈ space M. cls-val-process Mkt p 0 w = 0) ∧
    (AE w in M. 0 ≤ cls-val-process Mkt p m w) ∧
     0 < 𝒫(w in M. cls-val-process Mkt p m w > 0)) using assms using arbi-
trage-processE by simp
  from this obtain m where (self-financing Mkt p) and (trading-strategy p) and
    (∀ w∈ space M. cls-val-process Mkt p 0 w = 0) and
    (AE w in M. 0 ≤ cls-val-process Mkt p m w)
    0 < 𝒫(w in M. cls-val-process Mkt p m w > 0) by auto
  have ae-eq:∀ w∈ space M. (cls-val-process Mkt2 p 0 w = 0) = (cls-val-process
Mkt p 0 w = 0)
  proof
    fix w
    assume w∈ space M
    show (cls-val-process Mkt2 p 0 w = 0) = (cls-val-process Mkt p 0 w = 0)
      using assms coincides-cls-val-process by (metis)
  qed
  have ae-geq:almost-everywhere M (λw. cls-val-process Mkt2 p m w ≥ 0) = al-
most-everywhere M (λw. cls-val-process Mkt p m w ≥ 0)
  proof (rule AE-cong)
    fix w
    assume w∈ space M
    show (cls-val-process Mkt2 p m w ≥ 0) = (cls-val-process Mkt p m w ≥ 0)
      using assms coincides-cls-val-process by (metis)
  qed
  have self-financing Mkt2 p using assms coincides-on-self-financing
    using ‹self-financing Mkt p› by blast
  moreover have trading-strategy p using ‹trading-strategy p› .
  moreover have (∀ w∈ space M. cls-val-process Mkt2 p 0 w = 0) using ‹(∀ w∈
space M. cls-val-process Mkt p 0 w = 0)› ae-eq by simp
  moreover have AE w in M. 0 ≤ cls-val-process Mkt2 p m w using ‹AE w in
M. 0 ≤ cls-val-process Mkt p m w› ae-geq by simp
  moreover have 0 < prob {w ∈ space M. 0 < cls-val-process Mkt2 p m w}
  proof −
```

have $\{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt2\ p\ m\ w\} = \{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ p\ m\ w\}$
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *coincides-cls-val-process*)
thus *?thesis* **by** (*simp add*: ‹*0* < *prob* $\{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ p\ m\ w\}$›)
qed
ultimately show *?thesis* **using** *arbitrage-processI* **by** *blast*
qed

lemma (**in** *disc-filtr-prob-space*) *coincides-on-stocks-viable*:
assumes *coincides-on Mkt Mkt2* (*stocks Mkt*)
and *viable-market Mkt*
shows *viable-market Mkt2* **using** *coincides-on-arbitrage*
by (*metis* (*mono-tags*, *opaque-lifting*) *assms*(*1*) *assms*(*2*) *coincides-on-def stock-portfolio-def viable-market-def*)

lemma *coincides-stocks-val-process*:
assumes *stock-portfolio Mkt pf*
and *coincides-on Mkt Mkt2* (*stocks Mkt*)
shows $\forall n\ w.$ *val-process Mkt pf n w* = *val-process Mkt2 pf n w* **using** *assms*
unfolding *stock-portfolio-def*
by (*simp add*: *coincides-val-process*)

lemma *coincides-stocks-cls-val-process*:
assumes *stock-portfolio Mkt pf*
and *coincides-on Mkt Mkt2* (*stocks Mkt*)
shows $\forall n\ w.$ *cls-val-process Mkt pf n w* = *cls-val-process Mkt2 pf n w* **using** *assms*
unfolding *stock-portfolio-def*
by (*simp add*: *coincides-cls-val-process*)

lemma (**in** *disc-filtr-prob-space*) *coincides-on-adapted-val-process*:
assumes *coincides-on Mkt Mkt2 A*
and *support-set p* $\subseteq$ *A*
and *borel-adapt-stoch-proc F* (*val-process Mkt p*)
shows *borel-adapt-stoch-proc F* (*val-process Mkt2 p*) **unfolding** *adapt-stoch-proc-def*
proof
fix *n*
have *val-process Mkt p n* $\in$ *borel-measurable* (*F n*) **using** *assms* **unfolding** *adapt-stoch-proc-def* **by** *simp*
moreover have $\forall w.$ *val-process Mkt p n w* = *val-process Mkt2 p n w* **using** *assms coincides-val-process*[*of Mkt Mkt2 A*]
by *auto*
thus *val-process Mkt2 p n* $\in$ *borel-measurable* (*F n*)
using *calculation* **by** *presburger*
qed

lemma (**in** *disc-filtr-prob-space*) *coincides-on-adapted-cls-val-process*:

assumes *coincides-on Mkt Mkt2 A*
and *support-set p* $\subseteq$ *A*
and *borel-adapt-stoch-proc F* (*cls-val-process Mkt p*)
shows *borel-adapt-stoch-proc F* (*cls-val-process Mkt2 p*) **unfolding** *adapt-stoch-proc-def*
proof
fix *n*
have *cls-val-process Mkt p n* $\in$ *borel-measurable* (*F n*) **using** *assms* **unfolding** *adapt-stoch-proc-def* **by** *simp*
moreover have $\forall$ *w. cls-val-process Mkt p n w = cls-val-process Mkt2 p n w* **using** *assms coincides-cls-val-process*[*of Mkt Mkt2 A*]
by *auto*
thus *cls-val-process Mkt2 p n* $\in$ *borel-measurable* (*F n*)
using *calculation* **by** *presburger*
qed

7.2.7 Fair prices

definition (**in** *disc-filtr-prob-space*) *fair-price* **where**
fair-price Mkt π *pyf matur* $\longleftrightarrow$
($\exists$ *pr. price-structure pyf matur* π *pr* $\wedge$
($\forall$ *x Mkt2 p.* (*x*$\notin$ *stocks Mkt* $\longrightarrow$
((*coincides-on Mkt Mkt2* (*stocks Mkt*)) $\wedge$ (*prices Mkt2 x = pr*) $\wedge$ *portfolio p* $\wedge$ *support-set p* $\subseteq$ *stocks Mkt* $\cup$ {*x*} $\longrightarrow$
$\neg$ *arbitrage-process Mkt2 p*))))

lemma (**in** *disc-filtr-prob-space*) *fair-priceI*:
assumes *fair-price Mkt* π *pyf matur*
shows ($\exists$ *pr. price-structure pyf matur* π *pr* $\wedge$
($\forall$ *x.* (*x*$\notin$ *stocks Mkt* $\longrightarrow$
($\forall$ *Mkt2 p.* (*coincides-on Mkt Mkt2* (*stocks Mkt*)) $\wedge$ (*prices Mkt2 x = pr*) $\wedge$ *portfolio p* $\wedge$ *support-set p* $\subseteq$ *stocks Mkt* $\cup$ {*x*} $\longrightarrow$
$\neg$ *arbitrage-process Mkt2 p*)))) **using** *assms* **unfolding** *fair-price-def* **by** *simp*

Existence when replicating portfolio **lemma** (**in** *disc-equity-market*) *replicating-fair-price*:
assumes *viable-market Mkt*
and *replicating-portfolio pf der matur*
and *support-adapt Mkt pf*
shows *fair-price Mkt* (*initial-value pf*) *der matur*
proof (*rule ccontr*)
define π **where** π = (*initial-value pf*)
assume $\neg$ *fair-price Mkt* π *der matur*
hence *imps*: $\forall$ *pr.* (*price-structure der matur* π *pr*) $\longrightarrow$ ($\exists$ *x Mkt2 p.* (*x*$\notin$ *stocks Mkt* $\wedge$
(*coincides-on Mkt Mkt2* (*stocks Mkt*)) $\wedge$ (*prices Mkt2 x = pr*) $\wedge$ *portfolio p* $\wedge$ *support-set p* $\subseteq$ *stocks Mkt* $\cup$ {*x*} $\wedge$

arbitrage-process Mkt2 p)) **unfolding** *fair-price-def* **by** *simp*
have (*price-structure der matur* π (*cls-val-process Mkt pf*)) **unfolding** π*-def* **using** *replicating-price-process assms* **by** *simp*
hence $\exists$ *x Mkt2 p*. ($x \notin$ *stocks Mkt* $\wedge$
(*coincides-on Mkt Mkt2* (*stocks Mkt*)) $\wedge$ (*prices Mkt2 x* = (*cls-val-process Mkt pf*)) $\wedge$ *portfolio p* $\wedge$ *support-set p* $\subseteq$ *stocks Mkt* $\cup$ $\{x\}$ $\wedge$
arbitrage-process Mkt2 p) **using** *imps* **by** *simp*
from *this* **obtain** *x Mkt2 p* **where** $x \notin$ *stocks Mkt* **and** *coincides-on Mkt Mkt2* (*stocks Mkt*) **and** *prices Mkt2 x* = *cls-val-process Mkt pf*
and *portfolio p* **and** *support-set p*$\subseteq$ *stocks Mkt* $\cup$ $\{x\}$ **and** *arbitrage-process Mkt2 p* **by** *auto*
have $\forall$ *n w*. *cls-val-process Mkt pf n w* = *cls-val-process Mkt2 pf n w*
using *coincides-stocks-cls-val-process*[*of Mkt pf Mkt2*] *assms* ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› **unfolding** *replicating-portfolio-def*
by *simp*
have $\exists$ *m*. *self-financing Mkt2 p* $\wedge$
trading-strategy p $\wedge$
(*AE w in M*. *cls-val-process Mkt2 p 0 w* = *0*) $\wedge$
(*AE w in M*. *0* $\leq$ *cls-val-process Mkt2 p m w*) $\wedge$ *0* $<$ *prob* {*w* $\in$ *space M*. *0* $<$ *cls-val-process Mkt2 p m w*}
using ‹*arbitrage-process Mkt2 p*› **using** *arbitrage-processE*[*of Mkt2*] **by** *simp*
from *this* **obtain** *m* **where** *self-financing Mkt2 p*
trading-strategy p $\wedge$
(*AE w in M*. *cls-val-process Mkt2 p 0 w* = *0*) $\wedge$
(*AE w in M*. *0* $\leq$ *cls-val-process Mkt2 p m w*) $\wedge$ *0* $<$ *prob* {*w* $\in$ *space M*. *0* $<$ *cls-val-process Mkt2 p m w*} **by** *auto* **note** *mprop* = *this*
define *eq-stock* **where** *eq-stock* = *qty-replace-comp p x pf*
have $\forall$ *n w*. *cls-val-process Mkt pf n w* = *cls-val-process Mkt2 pf n w* **using** *assms* **unfolding** *replicating-portfolio-def*
using *coincides-cls-val-process*
using ‹$\forall$ *n w*. *cls-val-process Mkt pf n w* = *cls-val-process Mkt2 pf n w*› **by** *blast*
hence *prx*: $\forall$ *n w*. *prices Mkt2 x n w* = *cls-val-process Mkt2 pf n w* **using** ‹*prices Mkt2 x* = *cls-val-process Mkt pf*› **by** *simp*
have *stock-portfolio Mkt2 eq-stock*
by (*metis* (*no-types*, *lifting*) ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› ‹*portfolio p*› ‹*support-set p* $\subseteq$ *stocks Mkt* $\cup$ $\{x\}$›
assms(*2*) *coincides-on-def disc-equity-market.replicating-portfolio-def disc-equity-market-axioms eq-stock-def*
replace-comp-portfolio replace-comp-stocks stock-portfolio-def)
moreover have *arbitrage-process Mkt2 eq-stock*
proof (*rule arbitrage-val-process*)
show *arbitrage-process Mkt2 p* **using** ‹*arbitrage-process Mkt2 p*› **.**
show *vp*: $\forall$ *n w*. *cls-val-process Mkt2 p n w* = *cls-val-process Mkt2 eq-stock n w* **using** *replace-comp-cls-val-process*
‹*prices Mkt2 x* = *cls-val-process Mkt pf*› **unfolding** *eq-stock-def*
by (*metis* (*no-types*, *lifting*) ‹$\forall$ *n w*. *cls-val-process Mkt pf n w* = *cls-val-process Mkt2 pf n w*› ‹*portfolio p*› *assms*(*2*) *replicating-portfolio-def*
stock-portfolio-def)

show *trading-strategy eq-stock*
by (*metis* ‹*arbitrage-process Mkt2 p*› *arbitrage-process-def assms*(*2*) *eq-stock-def replace-comp-trading-strat replicating-portfolio-def*)
show *self-financing Mkt2 eq-stock* **unfolding** *eq-stock-def*
proof (*rule replace-comp-self-financing*)
show *portfolio pf* **using** *assms* **unfolding** *replicating-portfolio-def stock-portfolio-def* **by** *simp*
show *portfolio p* **using** ‹*portfolio p*› **.**
show ∀ *n w*. *prices Mkt2 x n w* = *cls-val-process Mkt2 pf n w* **using** *prx* **.**
show *self-financing Mkt2 p* **using** ‹*self-financing Mkt2 p*› **.**
show *self-financing Mkt2 pf* **using** *coincides-on-self-financing*[*of Mkt Mkt2 stocks Mkt pf*]
‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› *assms*(*2*) **unfolding** *stock-portfolio-def replicating-portfolio-def* **by** *auto*
qed
qed
moreover have *viable-market Mkt2* **using** *assms coincides-on-stocks-viable*[*of Mkt Mkt2*]
by (*simp add*: ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)›)
ultimately show *False* **unfolding** *viable-market-def* **by** *simp*
qed

Uniqueness when replicating portfolio The proof of uniqueness requires the existence of a stock that always takes strictly positive values.

locale *disc-market-pos-stock* = *disc-equity-market* +
fixes *pos-stock*
assumes *in-stock*: *pos-stock* ∈ *stocks Mkt*
and *positive*: ∀ *n w*. *prices Mkt pos-stock n w* > *0*
and *readable*: ∀ *asset*∈ *stocks Mkt*. *borel-adapt-stoch-proc F* (*prices Mkt asset*)

lemma (**in** *disc-market-pos-stock*) *pos-stock-borel-adapted*:
shows *borel-adapt-stoch-proc F* (*prices Mkt pos-stock*)
using *assets-def readable in-stock* **by** *auto*

definition *static-quantities* **where**
static-quantities p ⟷ (∀ *asset* ∈ *support-set p*. ∃ *c*::*real*. *p asset* = (λ *n w*. *c*))

lemma (**in** *disc-filtr-prob-space*) *static-quantities-trading-strat*:
assumes *static-quantities p*
and *finite* (*support-set p*)
shows *trading-strategy p* **unfolding** *trading-strategy-def*
proof (*intro conjI ballI*)
show *portfolio p* **using** *assms* **unfolding** *portfolio-def* **by** *simp*
fix *asset*

assume *asset* ∈ *support-set p*
hence ∃ *c*. *p asset* = (λ *n w*. *c*) **using** *assms* **unfolding** *static-quantities-def* **by** *simp*
then obtain *c* **where** *p asset* = (λ *n w*. *c*) **by** *auto*
show *borel-predict-stoch-proc F* (*p asset*) **unfolding** *predict-stoch-proc-def*
proof (*intro conjI*)
show *p asset 0* ∈ *borel-measurable* (*F 0*) **using** ‹*p asset* = (λ *n w*. *c*)› **by** *simp*
show ∀ *n*. *p asset* (*Suc n*) ∈ *borel-measurable* (*F n*)
proof
fix *n*
have *p asset* (*Suc n*) = (λ *w*. *c*) **using** ‹*p asset* = (λ *n w*. *c*)› **by** *simp*
thus *p asset* (*Suc n*) ∈ *borel-measurable* (*F n*) **by** *simp*
qed
qed
qed

lemma *two-component-support-set*:
assumes ∃ *n w*. *a n w* ≠ *0*
and ∃ *n w*. *b n w*≠ *0*
and *x* ≠ *y*
shows *support-set* ((λ (*x*::'*b*) (*n*::*nat*) (*w*::'*a*). *0*::*real*)(*x*:= *a*, *y*:= *b*)) = {*x*,*y*}
proof
let *?arb-pf* = (λ (*x*::'*b*) (*n*::*nat*) (*w*::'*a*). *0*::*real*)(*x*:= *a*, *y*:= *b*)
have ∃ *n w*. *?arb-pf x n w* ≠ *0* **using** *assms* **by** *simp*
moreover have ∃ *n w*. *?arb-pf y n w* ≠ *0* **using** *assms* **by** *simp*
ultimately show {*x*, *y*} ⊆ *support-set ?arb-pf* **unfolding** *support-set-def* **by** *simp*
show *support-set ?arb-pf* ⊆ {*x*, *y*}
proof (*rule ccontr*)
assume ¬ *support-set ?arb-pf* ⊆ {*x*, *y*}
hence ∃ *z*. *z*∈ *support-set ?arb-pf* ∧ *z*∉ {*x*, *y*} **by** *auto*
from *this* **obtain** *z* **where** *z*∈ *support-set ?arb-pf* **and** *z*∉ {*x*, *y*} **by** *auto*
have ∃ *n w*. *?arb-pf z n w* ≠ *0* **using** ‹*z*∈ *support-set ?arb-pf*› **unfolding** *support-set-def* **by** *simp*
from *this* **obtain** *n w* **where** *?arb-pf z n w* ≠ *0* **by** *auto*
have *?arb-pf z n w* = *0* **using** ‹*z*∉ {*x*, *y*}› **by** *simp*
thus *False* **using** ‹*?arb-pf z n w* ≠ *0*› **by** *simp*
qed
qed

lemma *two-component-val-process*:
assumes *arb-pf* = ((λ (*x*::'*b*) (*n*::*nat*) (*w*::'*a*). *0*::*real*)(*x*:= *a*, *y*:= *b*))
and *portfolio arb-pf*
and *x* ≠ *y*
and ∃ *n w*. *a n w* ≠ *0*
and ∃ *n w*. *b n w*≠ *0*
shows *val-process Mkt arb-pf n w* =

*prices Mkt y n w * b (Suc n) w + prices Mkt x n w * a (Suc n) w*
proof −
have *support-set arb-pf* = {*x*,*y*} **using** *assms* **by** (*simp add*:*two-component-support-set*)
have *val-process Mkt arb-pf n w* = (∑ *x*∈*support-set arb-pf*. *prices Mkt x n w * arb-pf x (Suc n) w*)
unfolding *val-process-def* **using** ‹*portfolio arb-pf*› **by** *simp*
also have ... = (∑ *x*∈{*x*, *y*}. *prices Mkt x n w * arb-pf x (Suc n) w*) **using** ‹*support-set arb-pf* = {*x*, *y*}›
by *simp*
also have ... = (∑ *x*∈{*y*}. *prices Mkt x n w * arb-pf x (Suc n) w*) + *prices Mkt x n w * arb-pf x (Suc n) w*
using *sum.insert*[*of* {*y*} *x* λ*x*. *prices Mkt x n w * arb-pf x n w*] *assms*(*3*)
by *auto*
also have ... = *prices Mkt y n w * arb-pf y (Suc n) w + prices Mkt x n w * arb-pf x (Suc n) w* **by** *simp*
also have ... = *prices Mkt y n w * b (Suc n) w + prices Mkt x n w * a (Suc n) w* **using** *assms* **by** *auto*
finally show *val-process Mkt arb-pf n w = prices Mkt y n w * b (Suc n) w + prices Mkt x n w * a (Suc n) w* **.**
qed

lemma *quantity-update-support-set*:
assumes ∃ *n w*. *pr n w* ≠ *0*
and *x*∉ *support-set p*
shows *support-set* (*p*(*x*:=*pr*)) = *support-set p* ∪ {*x*}
proof
show *support-set* (*p*(*x* := *pr*)) ⊆ *support-set p* ∪ {*x*}
proof
fix *y*
assume *y*∈ *support-set* (*p*(*x* := *pr*))
show *y* ∈ *support-set p* ∪ {*x*}
proof (*rule ccontr*)
assume ¬*y* ∈ *support-set p* ∪ {*x*}
hence *y* ≠ *x* **by** *simp*
have ∃ *n w*. (*p*(*x* := *pr*)) *y n w* ≠ *0* **using** ‹*y*∈ *support-set* (*p*(*x* := *pr*))›
unfolding *support-set-def* **by** *simp*
then obtain *n w* **where** *nwprop*: (*p*(*x* := *pr*)) *y n w* ≠ *0* **by** *auto*
have *y*∉ *support-set p* **using** ‹¬*y* ∈ *support-set p* ∪ {*x*}› **by** *simp*
hence *y* = *x* **using** *nwprop* **using** *support-set-def* **by** *force*
thus *False* **using** ‹*y*≠ *x*› **by** *simp*
qed
qed
show *support-set p* ∪ {*x*} ⊆ *support-set* (*p*(*x* := *pr*))
proof
fix *y*
assume *y* ∈ *support-set p* ∪ {*x*}
show *y*∈ *support-set* (*p*(*x* := *pr*))
proof (*cases y*∈ *support-set p*)
case *True*

```
    thus ?thesis
    proof −
      have f1: y ∈ {b. ∃ n a. p b n a ≠ 0}
        by (metis True support-set-def)
      then have y ≠ x
        using assms(2) support-set-def by force
      then show ?thesis
        using f1 by (simp add: support-set-def)
    qed
  next
    case False
    hence y = x using ‹y ∈ support-set p ∪ {x}› by auto
    thus ?thesis using assms by (simp add: support-set-def)
  qed
 qed
qed


lemma fix-asset-price:
 shows ∃ x Mkt2. x ∉ stocks Mkt ∧
 coincides-on Mkt Mkt2 (stocks Mkt) ∧
 prices Mkt2 x = pr
proof −
 have ∃ x. x∉ stocks Mkt by (metis UNIV-eq-I stk-strict-subs-def mkt-stocks-assets)
 from this obtain x where x∉ stocks Mkt by auto
 let ?res = discrete-market-of (stocks Mkt) ((prices Mkt)(x:=pr))
 have coincides-on Mkt ?res (stocks Mkt)
 proof −
   have stocks Mkt = stocks (discrete-market-of (stocks Mkt) ((prices Mkt)(x :=
pr)))
     by (metis (no-types) stk-strict-subs-def mkt-stocks-assets stocks-of)
   then show ?thesis
     by (simp add: ‹x ∉ stocks Mkt› coincides-on-def prices-of)
 qed
 have prices ?res x = pr by (simp add: prices-of)
 show ?thesis
   using ‹coincides-on Mkt (discrete-market-of (stocks Mkt) ((prices Mkt)(x :=
pr))) (stocks Mkt)› ‹prices (discrete-market-of (stocks Mkt) ((prices Mkt)(x :=
pr))) x = pr› ‹x ∉ stocks Mkt› by blast
qed


lemma (in disc-market-pos-stock) arbitrage-portfolio-properties:
 assumes price-structure der matur π pr
 and replicating-portfolio pf der matur
 and (coincides-on Mkt Mkt2 (stocks Mkt))
 and (prices Mkt2 x = pr)
 and x∉ stocks Mkt
```

and *diff-inv* = (*π* − *initial-value pf*) / *constant-image* (*prices Mkt pos-stock 0*)
and *diff-inv* ≠ *0*
and *arb-pf* = (λ (*x*::*'b*) (*n*::*nat*) (*w*::*'a*). *0*::*real*)(*x*:= (λ *n w*. −*1*), *pos-stock* := (λ *n w*. *diff-inv*))
and *contr-pf* = *qty-sum arb-pf pf*
shows *self-financing Mkt2 contr-pf*
and *trading-strategy contr-pf*
and ∀ *w*∈ *space M*. *cls-val-process Mkt2 contr-pf 0 w* = *0*
and *0* < *diff-inv* ⟶ (*AE w in M*. *0* < *cls-val-process Mkt2 contr-pf matur w*)
and *diff-inv* < *0* ⟶ (*AE w in M*. *0* > *cls-val-process Mkt2 contr-pf matur w*)
and *support-set arb-pf* = {*x*, *pos-stock*}
and *portfolio contr-pf*
proof −
have *0* < *constant-image* (*prices Mkt pos-stock 0*) **using** *trading-strategy-init*
proof −
have *borel-adapt-stoch-proc F* (*prices Mkt pos-stock*) **using** *pos-stock-borel-adapted* **by** *simp*
hence ∃ *c*. ∀ *w*∈*space M*. *prices Mkt pos-stock 0 w* = *c* **using** *adapted-init*[*of prices Mkt pos-stock*] **by** *simp*
moreover have ∀ *w*∈ *space M*. *0* < *prices Mkt pos-stock 0 w* **using** *positive* **by** *simp*
ultimately show *?thesis* **using** *constant-image-pos* **by** *simp*
qed
show *support-set arb-pf* = {*x*, *pos-stock*}
proof −
have *arb-pf* = (λ (*x*::*'b*) (*n*::*nat*) (*w*::*'a*). *0*::*real*)(*x*:= (λ *n w*. −*1*), *pos-stock* := (λ *n w*. *diff-inv*))
using ‹*arb-pf* = (λ (*x*::*'b*) (*n*::*nat*) (*w*::*'a*). *0*::*real*)(*x*:= (λ *n w*. −*1*), *pos-stock* := (λ *n w*. *diff-inv*))› **.**
moreover have ∃ *n w*. *diff-inv* ≠ *0* **using** *assms* **by** *simp*
moreover have *x*≠ *pos-stock* **using** ‹*x* ∉ *stocks Mkt*› *in-stock* **by** *auto*
ultimately show *?thesis* **by** (*simp add*:*two-component-support-set*)
qed
hence *portfolio arb-pf* **unfolding** *portfolio-def* **by** *simp*
have *arb-vp*:∀ *n w*. *val-process Mkt2 arb-pf n w* = *prices Mkt2 pos-stock n w* ∗ *diff-inv* − *pr n w*
proof (*intro allI*)
fix *n w*
have *val-process Mkt2 arb-pf n w* = *prices Mkt2 pos-stock n w* ∗ (λ *n w*. *diff-inv*) *n w* + *prices Mkt2 x n w* ∗ (λ *n w*. −*1*) *n w*
proof (*rule two-component-val-process*)
show *x*≠ *pos-stock* **using** ‹*x* ∉ *stocks Mkt*› *in-stock* **by** *auto*
show *arb-pf* = (λ*x n w*. *0*)(*x* := λ*a b*. − *1*, *pos-stock* := λ*a b*. *diff-inv*) **using** *assms* **by** *simp*
show *portfolio arb-pf* **using** ‹*portfolio arb-pf*› **by** *simp*
show ∃ *n w*. − (*1*::*real*) ≠ *0* **by** *simp*
show ∃ *n w*. *diff-inv* ≠ *0* **using** *assms* **by** *auto*
qed
also have ... = *prices Mkt2 pos-stock n w* ∗ *diff-inv* − *pr n w* **using** ‹*prices*

Mkt2 x = pr› **by** *simp*
finally show *val-process Mkt2 arb-pf n w = prices Mkt2 pos-stock n w ∗ diff-inv − pr n w* **.**
qed
have *static-quantities arb-pf* **unfolding** *static-quantities-def*
proof
fix *asset*
assume *asset ∈ support-set arb-pf*
thus ∃ *c. arb-pf asset* = (λ*n w. c*)
proof (*cases asset = x*)
case *True*
thus *?thesis* **using** *assms* **by** *auto*
next
case *False*
hence *asset = pos-stock* **using** ‹*support-set arb-pf* = {*x*, *pos-stock*}›
using ‹*asset ∈ support-set arb-pf*› **by** *blast*
thus *?thesis* **using** *assms* **by** *auto*
qed
qed
hence *trading-strategy arb-pf*
using ‹*portfolio arb-pf*› *portfolio-def static-quantities-trading-strat* **by** *blast*
have *self-financing Mkt2 arb-pf*
by (*simp add*: *static-portfolio-self-financing* ‹*arb-pf* = (λ*x n w. 0*) (*x* := λ*n w. −1*, *pos-stock* := λ*n w. diff-inv*)›)
hence *arb-uvp*: ∀ *n w. cls-val-process Mkt2 arb-pf n w = prices Mkt2 pos-stock n w ∗ diff-inv − pr n w* **using** *assms arb-vp*
by (*simp add*:*self-financingE*)
show *portfolio contr-pf* **using** *assms*
by (*metis* ‹*support-set arb-pf* = {*x*, *pos-stock*}› *replicating-portfolio-def finite.emptyI finite.insertI portfolio-def stock-portfolio-def sum-portfolio*)
have *support-set contr-pf* ⊆ *stocks Mkt* ∪ {*x*}
proof −
have *support-set contr-pf* ⊆ *support-set arb-pf* ∪ *support-set pf* **using** *assms*
by (*simp add*:*sum-support-set*)
moreover have *support-set arb-pf* ⊆ *stocks Mkt* ∪ {*x*} **using** ‹*support-set arb-pf* = {*x*, *pos-stock*}› *in-stock* **by** *simp*
moreover have *support-set pf* ⊆ *stocks Mkt* ∪ {*x*} **using** *assms* **unfolding** *replicating-portfolio-def*
stock-portfolio-def **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
show *self-financing Mkt2 contr-pf*
proof −
have *self-financing Mkt2* (*qty-sum arb-pf pf*)
proof (*rule sum-self-financing*)
show *portfolio arb-pf* **using** ‹*support-set arb-pf* = {*x*, *pos-stock*}› **unfolding** *portfolio-def* **by** *auto*
show *portfolio pf* **using** *assms* **unfolding** *replicating-portfolio-def stock-portfolio-def* **by** *auto*

show *self-financing Mkt2 pf* **using** *coincides-on-self-financing*
‹(*coincides-on Mkt Mkt2* (*stocks Mkt*))› ‹(*prices Mkt2 x* = *pr*)› *assms*(*2*)
unfolding *replicating-portfolio-def stock-portfolio-def* **by** *blast*
show *self-financing Mkt2 arb-pf*
by (*simp add*: *static-portfolio-self-financing* ‹*arb-pf* = (λ*x n w*. *0*) (*x* := λ*n w*. −*1*, *pos-stock* := λ*n w*. *diff-inv*)›)
qed
thus *?thesis* **using** *assms* **by** *simp*
qed
show *trading-strategy contr-pf*
proof −
have *trading-strategy* (*qty-sum arb-pf pf*)
proof (*rule sum-trading-strat*)
show *trading-strategy pf* **using** *assms* **unfolding** *replicating-portfolio-def* **by** *simp*
show *trading-strategy arb-pf* **using** ‹*trading-strategy arb-pf*› **.**
qed
thus *?thesis* **using** *assms* **by** *simp*
qed
show ∀ *w*∈ *space M*. *cls-val-process Mkt2 contr-pf 0 w* = *0*
proof
fix *w*
assume *w*∈ *space M*
have *cls-val-process Mkt2 contr-pf 0 w* = *cls-val-process Mkt2 arb-pf 0 w* + *cls-val-process Mkt2 pf 0 w*
using *sum-cls-val-process0*[*of arb-pf pf Mkt2*]
using ‹*portfolio arb-pf*› *assms replicating-portfolio-def stock-portfolio-def* **by** *blast*
also have ... = *prices Mkt2 pos-stock 0 w* ∗ *diff-inv* − *pr 0 w* + *cls-val-process Mkt2 pf 0 w* **using** *arb-uvp* **by** *simp*
also have ... = *constant-image* (*prices Mkt pos-stock 0*) ∗ *diff-inv* − *pr 0 w* + *cls-val-process Mkt2 pf 0 w*
proof −
have *f1*: *prices Mkt pos-stock* = *prices Mkt2 pos-stock*
using ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› *in-stock* **unfolding** *coincides-on-def* **by** *blast*
have *prices Mkt pos-stock 0 w* = *constant-image* (*prices Mkt pos-stock 0*)
using ‹*w* ∈ *space M*› *adapted-init constant-imageI pos-stock-borel-adapted* **by** *blast*
then show *?thesis*
using *f1* **by** *simp*
qed
also have ... = (π − *initial-value pf*) − *pr 0 w* + *cls-val-process Mkt2 pf 0 w*
using ‹*0* < *constant-image* (*prices Mkt pos-stock 0*)› *assms* **by** *simp*
also have ... = (π − *initial-value pf*) − π + *cls-val-process Mkt2 pf 0 w* **using** ‹*price-structure der matur* π *pr*›
price-structure-init[*of der matur* π *pr*] **by** (*simp add*: ‹*w* ∈ *space M*›)
also have ... = (π − *initial-value pf*) − π + (*initial-value pf*) **using** *initial-valueI assms* **unfolding** *replicating-portfolio-def*

using ‹*w* ∈ *space M*› *coincides-stocks-cls-val-process self-financingE readable*
by (*metis* (*no-types*, *opaque-lifting*) *support-adapt-def stock-portfolio-def subsetCE*)
also have ... = *0* **by** *simp*
finally show *cls-val-process Mkt2 contr-pf 0 w = 0* **.**
qed
show *0 < diff-inv* ⟶ (*AE w in M. 0 < cls-val-process Mkt2 contr-pf matur w*)
proof
assume *0 < diff-inv*
show *AE w in M. 0 < cls-val-process Mkt2 contr-pf matur w*
proof (*rule AE-mp*)
have *AE w in M. prices Mkt2 x matur w = der w* **using** ‹*price-structure der matur π pr*› ‹*prices Mkt2 x = pr*›
unfolding *price-structure-def* **by** *auto*
moreover have *AE w in M. cls-val-process Mkt2 pf matur w = der w* **using** *assms coincides-stocks-cls-val-process*[*of Mkt pf Mkt2*]
‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› **unfolding** *replicating-portfolio-def* **by** *auto*
ultimately show *AE w in M. prices Mkt2 x matur w = cls-val-process Mkt2 pf matur w* **by** *auto*
show *AE w in M. prices Mkt2 x matur w = cls-val-process Mkt2 pf matur w* ⟶ *0 < cls-val-process Mkt2 contr-pf matur w*
proof (*rule AE-I2*, *rule impI*)
fix *w*
assume *w*∈ *space M*
and *prices Mkt2 x matur w = cls-val-process Mkt2 pf matur w*
have *cls-val-process Mkt2 contr-pf matur w = cls-val-process Mkt2 arb-pf matur w + cls-val-process Mkt2 pf matur w*
using *sum-cls-val-process*[*of arb-pf pf Mkt2*]
‹*portfolio arb-pf*› *assms replicating-portfolio-def stock-portfolio-def* **by** *blast*
also have ... = *prices Mkt2 pos-stock matur w ∗ diff-inv − pr matur w + cls-val-process Mkt2 pf matur w*
using *arb-uvp* **by** *simp*
also have ... = *prices Mkt2 pos-stock matur w ∗ diff-inv − prices Mkt2 x matur w + cls-val-process Mkt2 pf matur w*
using ‹*prices Mkt2 x = pr*› **by** *simp*
also have ... = *prices Mkt2 pos-stock matur w ∗ diff-inv* **using** ‹*prices Mkt2 x matur w = cls-val-process Mkt2 pf matur w*›
by *simp*
also have ... > *0* **using** *positive* ‹*0 < diff-inv*›
by (*metis* ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› *coincides-on-def in-stock mult-pos-pos*)
finally have *cls-val-process Mkt2 contr-pf matur w > 0***.**
thus *0 < cls-val-process Mkt2 contr-pf matur w* **by** *simp*
qed
qed
qed
show *diff-inv < 0* ⟶ (*AE w in M. 0 > cls-val-process Mkt2 contr-pf matur w*)
proof

assume *diff-inv* < *0*
show *AE w in M. 0 > cls-val-process Mkt2 contr-pf matur w*
proof (*rule AE-mp*)
have *AE w in M. prices Mkt2 x matur w = der w* **using** ‹*price-structure der matur π pr*› ‹*prices Mkt2 x = pr*›
unfolding *price-structure-def* **by** *auto*
moreover have *AE w in M. cls-val-process Mkt2 pf matur w = der w* **using** *assms coincides-stocks-cls-val-process*[*of Mkt pf Mkt2*]
‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› **unfolding** *replicating-portfolio-def* **by** *auto*
ultimately show *AE w in M. prices Mkt2 x matur w = cls-val-process Mkt2 pf matur w* **by** *auto*
show *AE w in M. prices Mkt2 x matur w = cls-val-process Mkt2 pf matur w ⟶ 0 > cls-val-process Mkt2 contr-pf matur w*
proof (*rule AE-I2*, *rule impI*)
fix *w*
assume *w*∈ *space M*
and *prices Mkt2 x matur w = cls-val-process Mkt2 pf matur w*
have *cls-val-process Mkt2 contr-pf matur w = cls-val-process Mkt2 arb-pf matur w + cls-val-process Mkt2 pf matur w*
using *sum-cls-val-process*[*of arb-pf pf Mkt2*]
‹*portfolio arb-pf*› *assms replicating-portfolio-def stock-portfolio-def* **by** *blast*
also have *... = prices Mkt2 pos-stock matur w * diff-inv − pr matur w + cls-val-process Mkt2 pf matur w*
using *arb-uvp* **by** *simp*
also have *... = prices Mkt2 pos-stock matur w * diff-inv − prices Mkt2 x matur w + cls-val-process Mkt2 pf matur w*
using ‹*prices Mkt2 x = pr*› **by** *simp*
also have *... = prices Mkt2 pos-stock matur w * diff-inv* **using** ‹*prices Mkt2 x matur w = cls-val-process Mkt2 pf matur w*›
by *simp*
also have *... < 0* **using** *positive* ‹*diff-inv < 0*›
by (*metis* ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› *coincides-on-def in-stock mult-pos-neg*)
finally have *cls-val-process Mkt2 contr-pf matur w < 0*.
thus *0 > cls-val-process Mkt2 contr-pf matur w* **by** *simp*
qed
qed
qed
qed

lemma (**in** *disc-equity-market*) *mult-comp-cls-val-process-measurable′*:
assumes *cls-val-process Mkt2 pf n* ∈*borel-measurable* (*F n*)
and *portfolio pf*
and *qty n* ∈ *borel-measurable* (*F n*)
and *0* ≠ *n*
shows *cls-val-process Mkt2* (*qty-mult-comp pf qty*) *n* ∈ *borel-measurable* (*F n*)
proof −
have ∃ *m. n = Suc m* **using** *assms* **by** *presburger*

from *this* **obtain** *m* **where** $n = Suc\ m$ **by** *auto*
hence *cls-val-process Mkt2* (*qty-mult-comp pf qty*) (*Suc m*) ∈ *borel-measurable* (*F* (*Suc m*))
using *mult-comp-cls-val-process-Suc*[*of pf Mkt2 qty*] *borel-measurable-times*[*of cls-val-process Mkt2 pf* (*Suc m*) *F* (*Suc m*) *qty* (*Suc m*)]
assms ‹*n*= *Suc m*› **by** *presburger*
thus *?thesis* **using** ‹$n = Suc\ m$› **by** *simp*
qed

lemma (**in** *disc-equity-market*) *mult-comp-cls-val-process-measurable*:
assumes ∀ *n*. *cls-val-process Mkt2 pf n* ∈*borel-measurable* (*F n*)
and *portfolio pf*
and ∀ *n*. *qty* (*Suc n*) ∈ *borel-measurable* (*F n*)
shows ∀ *n*. *cls-val-process Mkt2* (*qty-mult-comp pf qty*) *n* ∈ *borel-measurable* (*F n*)
proof
fix *n*
show *cls-val-process Mkt2* (*qty-mult-comp pf qty*) *n* ∈ *borel-measurable* (*F n*)
proof (*cases n=0*)
case *False*
hence ∃ *m*. $n = Suc\ m$ **by** *presburger*
from *this* **obtain** *m* **where** $n = Suc\ m$ **by** *auto*
have *qty n* ∈ *borel-measurable* (*F n*)
using *Suc-n-not-le-n* ‹$n = Suc\ m$› *assms*(*3*) *increasing-measurable-info nat-le-linear* **by** *blast*
hence *qty* (*Suc m*) ∈ *borel-measurable* (*F* (*Suc m*)) **using** ‹$n = Suc\ m$› **by** *simp*
hence *cls-val-process Mkt2* (*qty-mult-comp pf qty*) (*Suc m*) ∈ *borel-measurable* (*F* (*Suc m*))
using *mult-comp-cls-val-process-Suc*[*of pf Mkt2 qty*] *borel-measurable-times*[*of cls-val-process Mkt2 pf* (*Suc m*) *F* (*Suc m*) *qty* (*Suc m*)]
assms ‹*n*= *Suc m*› **by** *presburger*
thus *?thesis* **using** ‹$n = Suc\ m$› **by** *simp*
next
case *True*
have *qty* (*Suc 0*) ∈ *borel-measurable* (*F 0*) **using** *assms* **by** *simp*
moreover have *cls-val-process Mkt2 pf 0* ∈ *borel-measurable* (*F 0*) **using** *assms* **by** *simp*
ultimately have (λ*w*. *cls-val-process Mkt2 pf 0 w* ∗ *qty* (*Suc 0*) *w*) ∈ *borel-measurable* (*F 0*) **by** *simp*
thus *?thesis* **using** *assms*(*2*) *True mult-comp-cls-val-process0*
by (*simp add*: ‹(λ*w*. *cls-val-process Mkt2 pf 0 w* ∗ *qty* (*Suc 0*) *w*) ∈ *borel-measurable* (*F 0*)› *mult-comp-cls-val-process0 measurable-cong*)
qed
qed

lemma (**in** *disc-equity-market*) *mult-comp-val-process-measurable*:
assumes *val-process Mkt2 pf n* ∈*borel-measurable* (*F n*)
and *portfolio pf*
and *qty* (*Suc n*) ∈ *borel-measurable* (*F n*)
shows *val-process Mkt2* (*qty-mult-comp pf qty*) *n* ∈ *borel-measurable* (*F n*)
using *mult-comp-val-process*[*of pf Mkt2 qty*] *borel-measurable-times*[*of val-process Mkt2 pf n F n qty* (*Suc n*)]
assms **by** *presburger*

lemma (**in** *disc-market-pos-stock*) *repl-fair-price-unique*:
assumes *replicating-portfolio pf der matur*
and *fair-price Mkt π der matur*
shows *π* = *initial-value pf*
proof −
have *expr*: (∃ *pr*. *price-structure der matur π pr* ∧
(∀ *x*. (*x*∉ *stocks Mkt* ⟶
(∀ *Mkt2 p*. (*coincides-on Mkt Mkt2* (*stocks Mkt*)) ∧ (*prices Mkt2 x* = *pr*) ∧ *portfolio p* ∧ *support-set p* ⊆ *stocks Mkt* ∪ {*x*} ⟶
¬ *arbitrage-process Mkt2 p*)))) **using** *assms fair-priceI* **by** *simp*
then obtain *pr* **where** *price-structure der matur π pr* **and**
xasset: (∀ *x*. (*x*∉ *stocks Mkt* ⟶
(∀ *Mkt2 p*. (*coincides-on Mkt Mkt2* (*stocks Mkt*)) ∧ (*prices Mkt2 x* = *pr*) ∧ *portfolio p* ∧ *support-set p* ⊆ *stocks Mkt* ∪ {*x*} ⟶
¬ *arbitrage-process Mkt2 p*))) **by** *auto*
define *diff-inv* **where** *diff-inv* = (*π* − *initial-value pf*) / *constant-image* (*prices Mkt pos-stock 0*)
{
fix *x*
assume *x*∉ *stocks Mkt*
hence *mkprop*: (∀ *Mkt2 p*. (*coincides-on Mkt Mkt2* (*stocks Mkt*)) ∧ (*prices Mkt2 x* = *pr*) ∧ *portfolio p* ∧ *support-set p* ⊆ *stocks Mkt* ∪ {*x*} ⟶
¬ *arbitrage-process Mkt2 p*) **using** *xasset* **by** *simp*
fix *Mkt2*
assume (*coincides-on Mkt Mkt2* (*stocks Mkt*)) **and** (*prices Mkt2 x* = *pr*)
have *0* < *constant-image* (*prices Mkt pos-stock 0*) **using** *trading-strategy-init*
proof −
have *borel-adapt-stoch-proc F* (*prices Mkt pos-stock*) **using** *pos-stock-borel-adapted* **by** *simp*
hence ∃ *c*. ∀ *w*∈*space M*. *prices Mkt pos-stock 0 w* = *c* **using** *adapted-init*[*of prices Mkt pos-stock*] **by** *simp*
moreover have ∀ *w*∈ *space M*. *0* < *prices Mkt pos-stock 0 w* **using** *positive* **by** *simp*
ultimately show *?thesis* **using** *constant-image-pos* **by** *simp*
qed

define *arb-pf* **where** *arb-pf* = (λ (*x*::′*b*) (*n*::*nat*) (*w*::′*a*). *0*::*real*)(*x*:= (λ *n w*. −*1*), *pos-stock* := (λ *n w*. *diff-inv*))
define *contr-pf* **where** *contr-pf* = *qty-sum arb-pf pf*

have *1*:*0* ≠ *diff-inv* ⟶ *self-financing Mkt2 contr-pf*
using *arbitrage-portfolio-properties*[*of der matur π pr pf Mkt2 x diff-inv arb-pf contr-pf*]
using ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› ‹*price-structure der matur π pr*› ‹*prices Mkt2 x* = *pr*›
‹*x* ∉ *stocks Mkt*› *arb-pf-def assms*(*1*) *contr-pf-def diff-inv-def* **by** *blast*
have *2*:*0* ≠ *diff-inv* ⟶ *trading-strategy contr-pf*
using *arbitrage-portfolio-properties*[*of der matur π pr pf Mkt2 x diff-inv arb-pf contr-pf*]
using ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› ‹*price-structure der matur π pr*› ‹*prices Mkt2 x* = *pr*›
‹*x* ∉ *stocks Mkt*› *arb-pf-def assms*(*1*) *contr-pf-def diff-inv-def* **by** *blast*
have *3*:*0* ≠ *diff-inv* ⟶ (∀ *w*∈ *space M*. *cls-val-process Mkt2 contr-pf 0 w* = *0*)
using *arbitrage-portfolio-properties*[*of der matur π pr pf Mkt2 x diff-inv arb-pf contr-pf*]
using ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› ‹*price-structure der matur π pr*› ‹*prices Mkt2 x* = *pr*›
‹*x* ∉ *stocks Mkt*› *arb-pf-def assms*(*1*) *contr-pf-def diff-inv-def* **by** *blast*
have *4*: *0* < *diff-inv* ⟶ (*AE w in M*. *0* < *cls-val-process Mkt2 contr-pf matur w*)
using *arbitrage-portfolio-properties*[*of der matur π pr pf Mkt2 x diff-inv arb-pf contr-pf*]
using ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› ‹*price-structure der matur π pr*› ‹*prices Mkt2 x* = *pr*›
‹*x* ∉ *stocks Mkt*› *arb-pf-def assms*(*1*) *contr-pf-def diff-inv-def* **by** *blast*
have *5*: *diff-inv* < *0* ⟶ (*AE w in M*. *0* > *cls-val-process Mkt2 contr-pf matur w*)
using *arbitrage-portfolio-properties*[*of der matur π pr pf Mkt2 x diff-inv arb-pf contr-pf*]
using ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› ‹*price-structure der matur π pr*› ‹*prices Mkt2 x* = *pr*›
‹*x* ∉ *stocks Mkt*› *arb-pf-def assms*(*1*) *contr-pf-def diff-inv-def* **by** *blast*
have *6*: *0* ≠ *diff-inv* ⟶ *support-set arb-pf* = {*x*, *pos-stock*}
using *arbitrage-portfolio-properties*[*of der matur π pr pf Mkt2 x diff-inv arb-pf contr-pf*]
using ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› ‹*price-structure der matur π pr*› ‹*prices Mkt2 x* = *pr*›
‹*x* ∉ *stocks Mkt*› *arb-pf-def assms*(*1*) *contr-pf-def diff-inv-def* **by** *blast*
have *7*: *0* ≠ *diff-inv* ⟶*support-set contr-pf* ⊆ *stocks Mkt* ∪ {*x*}
proof −
have *0* ≠ *diff-inv* ⟶ *support-set contr-pf* ⊆ *support-set arb-pf* ∪ *support-set pf* **unfolding** *contr-pf-def*
by (*simp add*:*sum-support-set*)
moreover have *0* ≠ *diff-inv* ⟶*support-set arb-pf* ⊆ *stocks Mkt* ∪ {*x*} **using** ‹*0* ≠ *diff-inv* ⟶ *support-set arb-pf* = {*x*, *pos-stock*}› *in-stock* **by** *simp*
moreover have *0* ≠ *diff-inv* ⟶*support-set pf* ⊆ *stocks Mkt* ∪ {*x*} **using** *assms* **unfolding** *replicating-portfolio-def*
stock-portfolio-def **by** *auto*
ultimately show *?thesis* **by** *auto*

qed
have *8*:*0 ≠ diff-inv ⟶portfolio contr-pf*
using *arbitrage-portfolio-properties*[*of der matur π pr pf Mkt2 x diff-inv arb-pf contr-pf*]
using ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› ‹*price-structure der matur π pr*› ‹*prices Mkt2 x = pr*›
‹*x ∉ stocks Mkt*› *arb-pf-def assms*(*1*) *contr-pf-def diff-inv-def* **by** *blast*
have *9*: *0 ≠ diff-inv ⟶ cls-val-process Mkt2 contr-pf matur ∈ borel-measurable* (*F matur*)
proof
assume *0 ≠ diff-inv*
have *10*:∀ *asset ∈ support-set arb-pf ∪ support-set pf*. *prices Mkt2 asset matur ∈ borel-measurable* (*F matur*)
proof
fix *asset*
assume *asset ∈ support-set arb-pf ∪ support-set pf*
show *prices Mkt2 asset matur ∈ borel-measurable* (*F matur*)
proof (*cases asset ∈ support-set pf*)
case *True*
thus *?thesis* **using** *assms readable*
by (*metis* (*no-types*, *lifting*) ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› *adapt-stoch-proc-def*
coincides-on-def disc-equity-market.replicating-portfolio-def
disc-equity-market-axioms stock-portfolio-def subsetCE)
next
case *False*
hence *asset∈ support-set arb-pf* **using** ‹*asset ∈ support-set arb-pf ∪ support-set pf*› **by** *auto*
show *?thesis*
proof (*cases asset = x*)
case *True*
thus *?thesis*
using ‹*price-structure der matur π pr*› ‹*prices Mkt2 x = pr*› *price-structure-borel-measurable* **by** *blast*
next
case *False*
hence *asset = pos-stock* **using** ‹*asset∈ support-set arb-pf*› ‹*0 ≠ diff-inv ⟶ support-set arb-pf = {x, pos-stock}*›
‹*0 ≠ diff-inv*› **by** *auto*
thus *?thesis*
by (*metis* ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› *adapt-stoch-proc-def coincides-on-def in-stock pos-stock-borel-adapted*)
qed
qed
qed
moreover have ∀ *asset∈support-set contr-pf*. *contr-pf asset matur ∈ borel-measurable* (*F matur*)
using ‹*0 ≠ diff-inv ⟶trading-strategy contr-pf*› ‹*0 ≠ diff-inv*›
by (*metis adapt-stoch-proc-def disc-filtr-prob-space.predict-imp-adapt disc-filtr-prob-space-axioms*

trading-strategy-def)
ultimately show *cls-val-process Mkt2 contr-pf matur* ∈ *borel-measurable* (*F matur*)
proof −
have ∀ *asset*∈*support-set contr-pf*. *contr-pf asset* (*Suc matur*) ∈ *borel-measurable* (*F matur*)
using ‹*0* ≠ *diff-inv* ⟶*trading-strategy contr-pf*› ‹*0* ≠ *diff-inv*›
by (*simp add*: *predict-stoch-proc-def trading-strategy-def*)
moreover have ∀ *asset*∈*support-set contr-pf*. *prices Mkt2 asset matur* ∈ *borel-measurable* (*F matur*) **using** *10* **unfolding** *contr-pf-def*
using *sum-support-set*[*of arb-pf pf*] **by** *auto*
ultimately show *?thesis* **by** (*metis* (*no-types*, *lifting*) *1* ‹*0* ≠ *diff-inv*› *quantity-adapted self-financingE*)
qed
qed
{
assume *0* > *diff-inv*
define *opp-pf* **where** *opp-pf* = *qty-mult-comp contr-pf* (λ *n w*. −*1*)
have *arbitrage-process Mkt2 opp-pf*
proof (*rule arbitrage-processI*, *rule exI*, *intro conjI*)
show *self-financing Mkt2 opp-pf* **using** *1* ‹*0* > *diff-inv*› *mult-time-constant-self-financing*[*of contr-pf*] *8*
unfolding *opp-pf-def* **by** *auto*
show *trading-strategy opp-pf* **unfolding** *opp-pf-def*
proof (*rule mult-comp-trading-strat*)
show *trading-strategy contr-pf* **using** *2* ‹*0* > *diff-inv*› **by** *auto*
show *borel-predict-stoch-proc F* (λ*n w*. − *1*) **by** (*simp add*: *constant-process-borel-predictable*)
qed
show ∀ *w*∈*space M*. *cls-val-process Mkt2 opp-pf 0 w* = *0*
proof
fix *w*
assume *w*∈ *space M*
show *cls-val-process Mkt2 opp-pf 0 w* = *0* **using** *3 8* ‹*0* > *diff-inv*›
using ‹*w* ∈ *space M*› *mult-comp-cls-val-process0 opp-pf-def* **by** *fastforce*
qed
have *AE w in M*. *0* < *cls-val-process Mkt2 opp-pf matur w*
proof (*rule AE-mp*)
show *AE w in M*. *0* > *cls-val-process Mkt2 contr-pf matur w* **using** *5* ‹*0* > *diff-inv*› **by** *auto*
show *AE w in M*. *cls-val-process Mkt2 contr-pf matur w* < *0* ⟶ *0* < *cls-val-process Mkt2 opp-pf matur w*
proof
fix *w*
assume *w*∈ *space M*
show *cls-val-process Mkt2 contr-pf matur w* < *0* ⟶ *0* < *cls-val-process Mkt2 opp-pf matur w*
proof
assume *cls-val-process Mkt2 contr-pf matur w* < *0*
show *0* < *cls-val-process Mkt2 opp-pf matur w*

```
            proof (cases matur = 0)
              case False
              hence ∃ m. Suc m = matur by presburger
              from this obtain m where Suc m = matur by auto
               hence 0 < cls-val-process Mkt2 opp-pf (Suc m) w using 3 8 ‹0 >
diff-inv› ‹w ∈ space M› mult-comp-cls-val-process-Suc  opp-pf-def
                using ‹cls-val-process Mkt2 contr-pf matur w < 0› by fastforce
              thus ?thesis using ‹Suc m = matur› by simp
            next
              case True
          thus ?thesis using 3 8 ‹0 > diff-inv› ‹w ∈ space M› mult-comp-cls-val-process0
opp-pf-def
              using ‹cls-val-process Mkt2 contr-pf matur w < 0› by auto
            qed
          qed
        qed
      qed
      thus AE w in M. 0 ≤ cls-val-process Mkt2 opp-pf matur w by auto
      show 0 < prob {w ∈ space M. 0 < cls-val-process Mkt2 opp-pf matur w}
      proof −
        let ?P = {w∈ space M. 0 < cls-val-process Mkt2 opp-pf matur w}
        have cls-val-process Mkt2 opp-pf matur ∈ borel-measurable (F matur)
        proof −
          have cls-val-process Mkt2 contr-pf matur ∈ borel-measurable (F matur)
using 9 ‹0 > diff-inv› by simp
          moreover have portfolio contr-pf using 8 ‹0 > diff-inv› by simp
            moreover have (λw. − 1) ∈ borel-measurable (F matur) by (simp
add:constant-process-borel-adapted)
          ultimately show ?thesis
            using mult-comp-cls-val-process-measurable
          proof −
            have diff-inv ≠ 0
              using ‹diff-inv < 0› by blast
            then have self-financing Mkt2 contr-pf
              by (metis 1)
            then show ?thesis
                by (metis (no-types) ‹(λw. − 1) ∈ borel-measurable (F matur)›
‹portfolio contr-pf›
                ‹self-financing Mkt2 opp-pf› ‹cls-val-process Mkt2 contr-pf matur
∈ borel-measurable (F matur)›
                mult-comp-val-process-measurable opp-pf-def self-financingE)
          qed
        qed
        moreover have space M = space (F matur)
          using filtration by (simp add: filtration-def subalgebra-def)
       ultimately have ?P ∈ sets (F matur) using borel-measurable-iff-greater[of
val-process Mkt2 contr-pf matur F matur]
          by auto
           hence ?P ∈ sets M by (meson filtration filtration-def subalgebra-def
```

subsetCE)
hence *measure M ?P = 1* **using** *prob-Collect-eq-1*[*of* λ*x. 0 < cls-val-process Mkt2 opp-pf matur x*]
‹*AE w in M. 0 < cls-val-process Mkt2 opp-pf matur w*› ‹*0 > diff-inv*›
by *blast*
thus *?thesis* **by** *simp*
qed
qed
have ∃ *p. portfolio p* ∧ *support-set p* ⊆ *stocks Mkt* ∪ {*x*} ∧ *arbitrage-process Mkt2 p*
proof(*intro exI conjI*)
show *arbitrage-process Mkt2 opp-pf* **using** ‹*arbitrage-process Mkt2 opp-pf*›
.
show *portfolio opp-pf* **unfolding** *opp-pf-def* **using** *8* ‹*0 > diff-inv*› **by** (*auto simp add*: *mult-comp-portfolio*)
show *support-set opp-pf* ⊆ *stocks Mkt* ∪ {*x*} **unfolding** *opp-pf-def* **using** *7* ‹*0 > diff-inv*›
using *mult-comp-support-set* **by** *fastforce*
qed
} **note** *negp = this*
{
assume *0 < diff-inv*
have *arbitrage-process Mkt2 contr-pf*
proof (*rule arbitrage-processI*, *rule exI*, *intro conjI*)
show *self-financing Mkt2 contr-pf* **using** *1* ‹*0 < diff-inv*› **by** *auto*
show *trading-strategy contr-pf* **using** *2* ‹*0 < diff-inv*› **by** *auto*
show ∀ *w*∈*space M. cls-val-process Mkt2 contr-pf 0 w = 0* **using** *3* ‹*0 < diff-inv*› **by** *auto*
show *AE w in M. 0 ≤ cls-val-process Mkt2 contr-pf matur w* **using** *4* ‹*0 < diff-inv*› **by** *auto*
show *0 < prob* {*w* ∈ *space M. 0 < cls-val-process Mkt2 contr-pf matur w*}
proof −
let *?P* = {*w*∈ *space M. 0 < cls-val-process Mkt2 contr-pf matur w*}
have *cls-val-process Mkt2 contr-pf matur* ∈ *borel-measurable* (*F matur*)
using *9* ‹*0 < diff-inv*› **by** *auto*
moreover have *space M = space* (*F matur*)
using *filtration* **by** (*simp add*: *filtration-def subalgebra-def*)
ultimately have *?P* ∈ *sets* (*F matur*) **using** *borel-measurable-iff-greater*[*of val-process Mkt2 contr-pf matur F matur*]
by *auto*
hence *?P* ∈ *sets M* **by** (*meson filtration filtration-def subalgebra-def subsetCE*)
hence *measure M ?P = 1* **using** *prob-Collect-eq-1*[*of* λ*x. 0 < cls-val-process Mkt2 contr-pf matur x*]
4 ‹*0 < diff-inv*› **by** *blast*
thus *?thesis* **by** *simp*
qed
qed
have ∃ *p. portfolio p* ∧ *support-set p* ⊆ *stocks Mkt* ∪ {*x*} ∧ *arbitrage-process*

Mkt2 p
proof(*intro exI conjI*)
show *arbitrage-process Mkt2 contr-pf* **using** ‹*arbitrage-process Mkt2 contr-pf*› **.**
show *portfolio contr-pf* **using** *8* ‹*0 < diff-inv*› **by** *auto*
show *support-set contr-pf* ⊆ *stocks Mkt* ∪ {*x*} **using** *7* ‹*0 < diff-inv*› **by** *auto*
qed
} note *posp = this*
have *diff-inv* ≠ *0* ⟶ ¬(∃ *pr. price-structure der matur π pr* ∧
(∀ *x.* (*x*∉ *stocks Mkt* ⟶
(∀ *Mkt2 p.* (*coincides-on Mkt Mkt2* (*stocks Mkt*)) ∧ (*prices Mkt2 x = pr*) ∧ *portfolio p* ∧ *support-set p* ⊆ *stocks Mkt* ∪ {*x*} ⟶
¬ *arbitrage-process Mkt2 p*))))
using ‹*coincides-on Mkt Mkt2* (*stocks Mkt*)› ‹*prices Mkt2 x = pr*› ‹*x* ∉ *stocks Mkt*› *xasset posp negp* **by** *force*
}
hence *diff-inv = 0* **using** *fix-asset-price expr* **by** *metis*
moreover have *constant-image* (*prices Mkt pos-stock 0*) > *0*
by (*simp add*: *adapted-init constant-image-pos pos-stock-borel-adapted positive*)
ultimately show *?thesis* **unfolding** *diff-inv-def* **by** *auto*
qed

7.3 Risk-neutral probability space

7.3.1 risk-free rate and discount factor processes

fun *disc-rfr-proc*:: *real* ⇒ *nat* ⇒ *'a* ⇒ *real*
where
rfr-base: (*disc-rfr-proc r*) *0 w = 1* |
rfr-step: (*disc-rfr-proc r*) (*Suc n*) *w* = (*1*+*r*) ∗ (*disc-rfr-proc r*) *n w*

lemma *disc-rfr-proc-borel-measurable*:
shows (*disc-rfr-proc r*) *n* ∈ *borel-measurable M*
proof (*induct n*)
case (*Suc n*) **thus** *?case* **by** (*simp add*:*borel-measurable-times*)
qed *auto*

lemma *disc-rfr-proc-nonrandom*:
fixes *r*::*real*
shows ⋀*n. disc-rfr-proc r n* ∈ *borel-measurable* (*F 0*) **using** *disc-rfr-proc-borel-measurable* **by** *auto*

lemma (**in** *disc-equity-market*) *disc-rfr-constant-time*:
shows ∃ *c.* ∀ *w* ∈ *space* (*F 0*). (*disc-rfr-proc r n*) *w* = *c*
proof (*rule triv-measurable-cst*)
show *space* (*F 0*) = *space M* **using** *filtration* **by** (*simp add*:*filtration-def subalgebra-def*)

show *sets* (*F 0*) = {{}, *space M*} **using** *info-disc-filtr* **by** (*simp add*: *bot-nat-def init-triv-filt-def*)
show (*disc-rfr-proc r n*) ∈ *borel-measurable* (*F 0*) **using** *disc-rfr-proc-nonrandom* **by** *blast*
show *space M* ≠ {} **by** (*simp add*:*not-empty*)
qed

lemma (**in** *disc-filtr-prob-space*) *disc-rfr-proc-borel-adapted*:
shows *borel-adapt-stoch-proc F* (*disc-rfr-proc r*)
unfolding *adapt-stoch-proc-def* **using** *disc-rfr-proc-nonrandom*
filtration **unfolding** *filtration-def*
by (*meson increasing-measurable-info le0*)

lemma *disc-rfr-proc-positive*:
assumes −*1* < *r*
shows ⋀*n w* . *0* < *disc-rfr-proc r n w*
proof −
fix *n*
fix *w*::*'a*
show *0* < *disc-rfr-proc r n w*
proof (*induct n*)
case *0* **thus** *?case* **using** *assms disc-rfr-proc.simps* **by** *simp*
next
case (*Suc n*) **thus** *?case* **using** *assms disc-rfr-proc.simps* **by** *simp*
qed
qed

lemma (**in** *prob-space*) *disc-rfr-constant-time-pos*:
assumes −*1* < *r*
shows ∃ *c* > *0*. ∀ *w* ∈ *space M*. (*disc-rfr-proc r n*) *w* = *c*
proof −
let *?F* = *sigma* (*space M*) {{}, *space M*}
have *ex*: ∃ *c*. ∀ *w* ∈ *space ?F*. (*disc-rfr-proc r n*) *w* = *c*
proof (*rule triv-measurable-cst*)
show *space ?F* = *space M* **by** *simp*
show *sets ?F* = {{}, *space M*} **by** (*meson sigma-algebra.sets-measure-of-eq sigma-algebra-trivial*)
show (*disc-rfr-proc r n*) ∈ *borel-measurable ?F* **using** *disc-rfr-proc-borel-measurable* **by** *blast*
show *space M* ≠ {} **by** (*simp add*:*not-empty*)
qed

from *this* **obtain** *c* **where** $\forall w \in$ *space ?F*. (*disc-rfr-proc r n*) *w* = *c* **by** *auto*
note *cprops* = *this*
have *c>0*
proof −
have ∃ *w*. *w*∈ *space M* **using** *subprob-not-empty* **by** *blast*
from *this* **obtain** *w* **where** *w*∈ *space M* **by** *auto*
hence *c* = *disc-rfr-proc r n w* **using** *cprops* **by** *simp*
also have ... > *0* **using** *disc-rfr-proc-positive*[*of r n*] *assms* **by** *simp*
finally show *?thesis* **.**
qed
moreover have *space M* = *space ?F* **by** *simp*
ultimately show *?thesis* **using** *ex* **using** *cprops* **by** *blast*
qed

lemma *disc-rfr-proc-Suc-div*:
assumes −*1* < *r*
shows ⋀*w*. *disc-rfr-proc r* (*Suc n*) *w*/*disc-rfr-proc r n w* = *1*+*r*
proof −
fix *w*::*'a*
show *disc-rfr-proc r* (*Suc n*) *w*/*disc-rfr-proc r n w* = *1*+*r*
using *disc-rfr-proc-positive assms* **by** (*metis rfr-step less-irrefl nonzero-eq-divide-eq*)
qed

definition *discount-factor* **where**
discount-factor r n = (λ*w*. *inverse* (*disc-rfr-proc r n w*))

lemma *discount-factor-times-rfr*:
assumes −*1* < *r*
shows (*1*+*r*) ∗ *discount-factor r* (*Suc n*) *w* = *discount-factor r n w* **unfolding** *discount-factor-def* **using** *assms* **by** *simp*

lemma *discount-factor-borel-measurable*:
shows *discount-factor r n* ∈ *borel-measurable M* **unfolding** *discount-factor-def*
proof (*rule borel-measurable-inverse*)
show *disc-rfr-proc r n* ∈ *borel-measurable M* **by** (*simp add*:*disc-rfr-proc-borel-measurable*)
qed

lemma *discount-factor-init*:
shows *discount-factor r 0* = (λ*w*. *1*) **unfolding** *discount-factor-def* **by** *simp*

lemma *discount-factor-nonrandom*:
shows *discount-factor r n* ∈ *borel-measurable M* **unfolding** *discount-factor-def*
proof (*rule borel-measurable-inverse*)
show *disc-rfr-proc r n* ∈ *borel-measurable M* **by** (*simp add*:*disc-rfr-proc-borel-measurable*)
qed

lemma *discount-factor-positive*:

assumes $-1 < r$
shows $\bigwedge n\ w\ .\ 0 <$ *discount-factor* $r\ n\ w$ **using** *assms disc-rfr-proc-positive*
unfolding *discount-factor-def* **by** *auto*

lemma (**in** *prob-space*) *discount-factor-constant-time-pos*:
assumes $-1 < r$
shows $\exists\, c > 0.\, \forall\, w \in$ *space* $M.$ (*discount-factor* $r\ n$) $w = c$ **using** *disc-rfr-constant-time-pos*
unfolding *discount-factor-def*
by (*metis assms inverse-positive-iff-positive*)

locale *rsk-free-asset* =
fixes *Mkt r risk-free-asset*
assumes *acceptable-rate*: $-1 < r$
and *rf-price*: *prices Mkt risk-free-asset* = *disc-rfr-proc r*
and *rf-stock*: *risk-free-asset* $\in$ *stocks Mkt*

locale *rfr-disc-equity-market* = *disc-equity-market* + *rsk-free-asset* +
assumes *rd*: $\forall$ *asset*$\in$ *stocks Mkt. borel-adapt-stoch-proc F* (*prices Mkt asset*)

sublocale *rfr-disc-equity-market* $\subseteq$ *disc-market-pos-stock - - - risk-free-asset*
by (*unfold-locales*, (*auto simp add*: *rf-stock rd disc-rfr-proc-positive rf-price acceptable-rate*))

7.3.2 Discounted value of a stochastic process

definition *discounted-value* **where**
discounted-value r X = (λ *n w. discount-factor r n w* $*$ *X n w*)

lemma (**in** *rfr-disc-equity-market*) *discounted-rfr*:
shows *discounted-value r* (*prices Mkt risk-free-asset*) *n w* = *1* **unfolding** *discounted-value-def discount-factor-def*
using *rf-price* **by** (*metis less-irrefl mult.commute positive right-inverse*)

lemma *discounted-init*:
shows $\forall\, w.$ *discounted-value r X 0 w* = *X 0 w* **unfolding** *discounted-value-def*
by (*simp add*: *discount-factor-init*)

lemma *discounted-mult*:
shows $\forall\, n\ w.$ *discounted-value r* ($\lambda m\ x.\ X\ m\ x * Y\ m\ x$) *n w* = *X n w* $*$ (*discounted-value r Y*) *n w*
by (*simp add*: *discounted-value-def*)

lemma *discounted-mult'*:

shows *discounted-value r* (λ*m x*. *X m x* ∗ *Y m x*) *n w* = *X n w* ∗ (*discounted-value r Y*) *n w*
by (*simp add*: *discounted-value-def*)

lemma *discounted-mult-times-rfr*:
assumes −*1* < *r*
shows *discounted-value r* (λ*m w*. (*1*+*r*) ∗ *X w*) (*Suc n*) *w* = *discounted-value r* (λ*m w*. *X w*) *n w*
unfolding *discounted-value-def* **using** *assms discount-factor-times-rfr discounted-mult*
by (*simp add*: *discount-factor-times-rfr mult.commute*)

lemma *discounted-cong*:
assumes ∀ *n w*. *X n w* = *Y n w*
shows ∀ *n w*. *discounted-value r X n w* = *discounted-value r Y n w*
by (*simp add*: *assms discounted-value-def*)

lemma *discounted-cong′*:
assumes *X n w* = *Y n w*
shows *discounted-value r X n w* = *discounted-value r Y n w*
by (*simp add*: *assms discounted-value-def*)

lemma *discounted-AE-cong*:
assumes *AE w in N*. *X n w* = *Y n w*
shows *AE w in N*. *discounted-value r X n w* = *discounted-value r Y n w*
proof (*rule AE-mp*)
show *AE w in N*. *X n w* = *Y n w* **using** *assms* **by** *simp*
show *AE w in N*. *X n w* = *Y n w* ⟶ *discounted-value r X n w* = *discounted-value r Y n w*
proof
fix *w*
assume *w*∈ *space N*
thus *X n w* = *Y n w* ⟶ *discounted-value r X n w* = *discounted-value r Y n w* **by** (*simp add*:*discounted-value-def*)
qed
qed

lemma *discounted-sum*:
assumes *finite I*
shows ∀ *n w*. (∑ *i*∈ *I*. (*discounted-value r* (λ*m x*. *f i m x*)) *n w*) = (*discounted-value r* (λ*m x*. (∑ *i*∈ *I*. *f i m x*)) *n w*)
using *assms*(*1*) *subset-refl*[*of I*]
proof (*induct rule*: *finite-subset-induct*)
case *empty*
then show *?case*
by (*simp add*: *discounted-value-def*)
next
case (*insert a F*)

show *?case*
proof (*intro allI*)
fix *n w*
have ($\sum$ *i*∈*insert a F*. *discounted-value r* (*f i*) *n w*) = *discounted-value r* (*f a*) *n w* + ($\sum$ *i*∈*F*. *discounted-value r* (*f i*) *n w*)
by (*simp add*: *insert.hyps*(*1*) *insert.hyps*(*3*))
also have ... = *discounted-value r* (*f a*) *n w* + *discounted-value r* (λ*m x*. $\sum$ *i*∈*F*. *f i m x*) *n w* **using** *insert.hyps*(*4*) **by** *simp*
also have ... = *discounted-value r* (λ*m x*. $\sum$ *i*∈*insert a F*. *f i m x*) *n w*
by (*simp add*: *discounted-value-def insert.hyps*(*1*) *insert.hyps*(*3*) *ring-class.ring-distribs*(*1*))
finally show ($\sum$ *i*∈*insert a F*. *discounted-value r* (*f i*) *n w*) = *discounted-value r* (λ*m x*. $\sum$ *i*∈*insert a F*. *f i m x*) *n w* **.**
qed
qed

lemma *discounted-adapted*:
assumes *borel-adapt-stoch-proc F X*
shows *borel-adapt-stoch-proc F* (*discounted-value r X*) **unfolding** *adapt-stoch-proc-def*
proof
fix *t*
show *discounted-value r X t* ∈ *borel-measurable* (*F t*) **unfolding** *discounted-value-def*
proof (*rule borel-measurable-times*)
show *X t* ∈ *borel-measurable* (*F t*) **using** *assms* **unfolding** *adapt-stoch-proc-def* **by** *simp*
show *discount-factor r t* ∈ *borel-measurable* (*F t*) **using** *discount-factor-borel-measurable* **by** *auto*
qed
qed

lemma *discounted-measurable*:
assumes *X*∈ *borel-measurable N*
shows *discounted-value r* (λ*m*. *X*) *m* ∈ *borel-measurable N* **unfolding** *discounted-value-def*
proof (*rule borel-measurable-times*)
show *X*∈ *borel-measurable N* **using** *assms* **by** *simp*
show *discount-factor r m* ∈ *borel-measurable N* **using** *discount-factor-borel-measurable* **by** *auto*
qed

lemma (**in** *prob-space*) *discounted-integrable*:
assumes *integrable N* (*X n*)
and −*1* < *r*
and *space N* = *space M*
shows *integrable N* (*discounted-value r X n*) **unfolding** *discounted-value-def*
proof −
have ∃ *c*> *0*. ∀ *w* ∈ *space M*. (*discount-factor r n*) *w* = *c* **using** *discount-factor-constant-time-pos assms* **by** *simp*
from *this* **obtain** *c* **where** *c* > *0* **and** ∀ *w* ∈ *space M*. (*discount-factor r n*) *w*

$= c$ **by** *auto* **note** *cprops = this*
hence $\forall\, w \in$ *space M*. *discount-factor r n w* $= c$ **using** *cprops* **by** *simp*
hence $\forall\, w \in$ *space N*. *discount-factor r n w* $= c$ **using** *assms* **by** *simp*
thus *integrable N* (λw. *discount-factor r n w* $*$ *X n w*)
using ‹$\forall\, w \in$ *space N*. *discount-factor r n w* $= c$› *assms*
Bochner-Integration.integrable-cong[*of N N* (λw. *discount-factor r n w* $*$ *X n w*) (λw. *c* $*$ *X n w*)] **by** *simp*
qed

7.3.3 Results on risk-neutral probability spaces

definition (**in** *rfr-disc-equity-market*) *risk-neutral-prob* **where**
risk-neutral-prob N $\longleftrightarrow$ (*prob-space N*) $\wedge$ ($\forall$ *asset* $\in$ *stocks Mkt*. *martingale N F* (*discounted-value r* (*prices Mkt asset*)))

lemma *integrable-val-process*:
assumes $\forall$ *asset* $\in$ *support-set pf*. *integrable M* (λw. *prices Mkt asset n w* $*$ *pf asset* (*Suc n*) *w*)
shows *integrable M* (*val-process Mkt pf n*)
proof (*cases portfolio pf*)
case *False*
thus *?thesis* **unfolding** *val-process-def* **by** *simp*
next
case *True*
hence *val-process Mkt pf n* $= (\lambda w.\ \sum x \in$ *support-set pf*. *prices Mkt x n w* $*$ *pf x* (*Suc n*) *w*)
unfolding *val-process-def* **by** *simp*
moreover have *integrable M* ($\lambda w.\ \sum x \in$ *support-set pf*. *prices Mkt x n w* $*$ *pf x* (*Suc n*) *w*) **using** *assms* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *integrable-self-fin-uvp*:
assumes $\forall$ *asset* $\in$ *support-set pf*. *integrable M* (λw. *prices Mkt asset n w* $*$ *pf asset* (*Suc n*) *w*)
and *self-financing Mkt pf*
shows *integrable M* (*cls-val-process Mkt pf n*)
proof −
have *val-process Mkt pf n* $=$ *cls-val-process Mkt pf n* **using** *assms* **by** (*simp add*:*self-financingE*)
moreover have *integrable M* (*val-process Mkt pf n*) **using** *assms* **by** (*simp add*:*integrable-val-process*)
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *rfr-disc-equity-market*) *stocks-portfolio-risk-neutral*:

assumes *risk-neutral-prob N*
and *trading-strategy pf*
and *subalgebra N M*
and *support-set pf* $\subseteq$ *stocks Mkt*
and $\forall\, n.\ \forall$ *asset* $\in$ *support-set pf. integrable N* (λw. *prices Mkt asset* (*Suc n*) *w* $*$ *pf asset* (*Suc n*) *w*)
shows $\forall\, x \in$ *support-set pf. AE w in N.*
(*real-cond-exp N* (*F n*) (*discounted-value r* ($\lambda m\ y$. *prices Mkt x m y* $*$ *pf x m y*) (*Suc n*))) *w* =
discounted-value r ($\lambda m\ y$. *prices Mkt x m y* $*$ *pf x* (*Suc m*) *y*) *n w*
proof
have *nsigfin*: $\forall\, n$. *sigma-finite-subalgebra N* (*F n*) **using** *assms* **unfolding** *risk-neutral-prob-def martingale-def subalgebra-def*
using *filtration filtration-def risk-neutral-prob-def prob-space.subalgebra-sigma-finite in-stock* **by** *metis*
have *disc-filtr-prob-space N F*
proof −
have *prob-space N* **using** *assms* **unfolding** *risk-neutral-prob-def* **by** *simp*
moreover have *disc-filtr N F* **using** *assms subalgebra-filtration*
by (*metis* (*no-types*, *lifting*) *filtration disc-filtr-def filtration-def*)
ultimately show *?thesis*
by (*simp add*: *disc-filtr-prob-space-axioms-def disc-filtr-prob-space-def*)
qed
fix *asset*
assume *asset* $\in$ *support-set pf*
hence *asset* $\in$ *stocks Mkt* **using** *assms* **by** *auto*
have *discounted-value r* (*prices Mkt asset*) (*Suc n*) $\in$ *borel-measurable M* **using** *assms readable*
by (*meson* ‹*asset* $\in$ *stocks Mkt*› *borel-adapt-stoch-proc-borel-measurable discounted-adapted*
rfr-disc-equity-market.risk-neutral-prob-def rfr-disc-equity-market-axioms)
hence *b*: *discounted-value r* (*prices Mkt asset*) (*Suc n*) $\in$ *borel-measurable N*
using *assms Conditional-Expectation.measurable-from-subalg*[*of N M - borel*] **by** *auto*
show *AEeq N* (*real-cond-exp N* (*F n*) (*discounted-value r* ($\lambda m\ y$. *prices Mkt asset m y* $*$ *pf asset m y*) (*Suc n*)))
(*discounted-value r* ($\lambda m\ y$. *prices Mkt asset m y* $*$ *pf asset* (*Suc m*) *y*) *n*)
proof −
have *AE w in N.* (*real-cond-exp N* (*F n*) (*discounted-value r* ($\lambda m\ y$. *prices Mkt asset m y* $*$ *pf asset m y*) (*Suc n*))) *w* =
(*real-cond-exp N* (*F n*) (λz. *pf asset* (*Suc n*) *z* $*$ *discounted-value r* ($\lambda m\ y$. *prices Mkt asset m y*) (*Suc n*) *z*)) *w*
proof (*rule sigma-finite-subalgebra.real-cond-exp-cong*)
show *sigma-finite-subalgebra N* (*F n*) **using** *nsigfin* **..**
show *AE w in N. discounted-value r* ($\lambda m\ y$. *prices Mkt asset m y* $*$ *pf asset m y*) (*Suc n*) *w* =
pf asset (*Suc n*) *w* $*$ *discounted-value r* ($\lambda m\ y$. *prices Mkt asset m y*) (*Suc n*) *w*
by (*simp add*: *discounted-value-def*)

show *discounted-value r* (λ*m y*. *prices Mkt asset m y* ∗ *pf asset m y*) (*Suc n*) ∈ *borel-measurable N*
proof −
have (λ*y*. *prices Mkt asset* (*Suc n*) *y* ∗ *pf asset* (*Suc n*) *y*) ∈ *borel-measurable N*
using *assms* ‹*asset*∈ *support-set pf*› **by** (*simp add*:*borel-measurable-integrable*)
thus *?thesis* **unfolding** *discounted-value-def* **using** *discount-factor-borel-measurable*[*of r Suc n N*] **by** *simp*
qed
show (λ*z*. *pf asset* (*Suc n*) *z* ∗ *discounted-value r* (*prices Mkt asset*) (*Suc n*) *z*) ∈ *borel-measurable N*
proof −
have *pf asset* (*Suc n*) ∈ *borel-measurable M* **using** *assms* ‹*asset*∈ *support-set pf*› **unfolding** *trading-strategy-def*
using *borel-predict-stoch-proc-borel-measurable*[*of pf asset*] **by** *auto*
hence *a*: *pf asset* (*Suc n*) ∈ *borel-measurable N* **using** *assms Conditional-Expectation.measurable-from-subalg*[*of N M - borel*] **by** *blast*
show *?thesis* **using** *a b* **by** *simp*
qed
qed
also have *AE w in N*. (*real-cond-exp N* (*F n*) (λ*z*. *pf asset* (*Suc n*) *z* ∗ *discounted-value r* (λ*m y*. *prices Mkt asset m y*) (*Suc n*) *z*)) *w* =
pf asset (*Suc n*) *w* ∗ (*real-cond-exp N* (*F n*) (λ*z*. *discounted-value r* (λ*m y*. *prices Mkt asset m y*) (*Suc n*) *z*)) *w*
proof (*rule sigma-finite-subalgebra.real-cond-exp-mult*)
show *discounted-value r* (*prices Mkt asset*) (*Suc n*) ∈ *borel-measurable N* **using** *b* **by** *simp*
show *sigma-finite-subalgebra N* (*F n*) **using** *nsigfin* **..**
show *pf asset* (*Suc n*) ∈ *borel-measurable* (*F n*) **using** *assms* ‹*asset*∈ *support-set pf*› **unfolding** *trading-strategy-def predict-stoch-proc-def* **by** *auto*
show *integrable N* (λ*z*. *pf asset* (*Suc n*) *z* ∗ *discounted-value r* (*prices Mkt asset*) (*Suc n*) *z*)
proof −
have *integrable N* (λ*w*. *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*) **using** *assms* ‹*asset* ∈ *support-set pf*› **by** *auto*
hence *integrable N* (*discounted-value r* (λ*m w*. *prices Mkt asset m w* ∗ *pf asset m w*) (*Suc n*)) **using** *assms*
unfolding *risk-neutral-prob-def* **using** *acceptable-rate* **by** (*auto simp add*:*discounted-integrable subalgebra-def*)
thus *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *Bochner-Integration.integrable-cong discounted-value-def mult.assoc mult.commute*)
qed
qed
also have *AE w in N*. *pf asset* (*Suc n*) *w* ∗ (*real-cond-exp N* (*F n*) (λ*z*. *discounted-value r* (λ*m y*. *prices Mkt asset m y*) (*Suc n*) *z*)) *w* =
pf asset (*Suc n*) *w* ∗ *discounted-value r* (λ*m y*. *prices Mkt asset m y*) *n w*
proof −

have *AEeq N* (*real-cond-exp N* (*F n*) (λ*z*. *discounted-value r* (λ*m y*. *prices Mkt asset m y*) (*Suc n*) *z*))
(λ*z*. *discounted-value r* (λ*m y*. *prices Mkt asset m y*) *n z*)
proof −
have *martingale N F* (*discounted-value r* (*prices Mkt asset*))
using *assms* ‹*asset* ∈ *stocks Mkt*› **unfolding** *risk-neutral-prob-def* **by** *simp*
moreover have *filtrated-prob-space N F* **using** ‹*disc-filtr-prob-space N F*›
using *assms*(*2*) *disc-filtr-prob-space.axioms*(*1*) *filtrated-prob-space.intro filtrated-prob-space-axioms.intro filtration prob-space-axioms*
by (*metis assms*(*3*) *subalgebra-filtration*)
ultimately show *?thesis* **using** *martingaleAE*[*of N F discounted-value r* (*prices Mkt asset*) *n Suc n*] *assms*
by *simp*
qed
thus *?thesis* **by** *auto*
qed
also have *AE w in N*. *pf asset* (*Suc n*) *w* ∗ *discounted-value r* (λ*m y*. *prices Mkt asset m y*) *n w* =
discounted-value r (λ*m y*. *pf asset* (*Suc m*) *y* ∗ *prices Mkt asset m y*) *n w* **by** (*simp add*: *discounted-value-def*)
also have *AE w in N*. *discounted-value r* (λ*m y*. *pf asset* (*Suc m*) *y* ∗ *prices Mkt asset m y*) *n w* =
discounted-value r (λ*m y*. *prices Mkt asset m y* ∗ *pf asset* (*Suc m*) *y*) *n w*
by (*simp add*: *discounted-value-def*)
finally show *AE w in N*.
(*real-cond-exp N* (*F n*) (*discounted-value r* (λ*m y*. *prices Mkt asset m y* ∗ *pf asset m y*) (*Suc n*))) *w* =
(λ*x*. *discounted-value r* (λ*m y*. *prices Mkt asset m y* ∗ *pf asset* (*Suc m*) *y*) *n x*) *w* **.**
qed
qed

lemma (**in** *rfr-disc-equity-market*) *self-fin-trad-strat-mart*:
assumes *risk-neutral-prob N*
and *filt-equiv F M N*
and *trading-strategy pf*
and *self-financing Mkt pf*
and *stock-portfolio Mkt pf*
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *integrable N* (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*)
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *integrable N* (λ*w*. *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*)
shows *martingale N F* (*discounted-value r* (*cls-val-process Mkt pf*))
proof (*rule disc-martingale-charact*)
show *nsigfin*: ∀ *n*. *sigma-finite-subalgebra N* (*F n*) **using** *filt-equiv-prob-space-subalgebra assms*
using *filtration filtration-def risk-neutral-prob-def subalgebra-sigma-finite*

by *fastforce*
show *filtration N F* **using** *assms* **by** (*simp add*:*filt-equiv-filtration*)
have *borel-adapt-stoch-proc F* (*discounted-value r* (*cls-val-process Mkt pf*)) **using** *assms discounted-adapted*
cls-val-process-adapted[*of pf*] *stock-portfolio-def*
by (*metis* (*mono-tags*, *opaque-lifting*) *support-adapt-def readable subsetCE*)
thus ∀ *m*. *discounted-value r* (*cls-val-process Mkt pf*) *m* ∈ *borel-measurable* (*F m*) **unfolding** *adapt-stoch-proc-def* **by** *simp*
show ∀ *t*. *integrable N* (*discounted-value r* (*cls-val-process Mkt pf*) *t*)
proof
fix *t*
have *integrable N* (*cls-val-process Mkt pf t*) **using** *assms* **by** (*simp add*: *integrable-self-fin-uvp*)
thus *integrable N* (*discounted-value r* (*cls-val-process Mkt pf*) *t*) **using** *assms discounted-integrable acceptable-rate*
by (*metis filt-equiv-space*)
qed
show ∀ *n*. *AE w in N*. *real-cond-exp N* (*F n*) (*discounted-value r* (*cls-val-process Mkt pf*) (*Suc n*)) *w* =
discounted-value r (*cls-val-process Mkt pf*) *n w*
proof
fix *n*
show *AE w in N*. *real-cond-exp N* (*F n*) (*discounted-value r* (*cls-val-process Mkt pf*) (*Suc n*)) *w* =
discounted-value r (*cls-val-process Mkt pf*) *n w*
proof −
{
fix *w*
assume *w*∈ *space M*
have *discounted-value r* (*cls-val-process Mkt pf*) (*Suc n*) *w* =
discount-factor r (*Suc n*) *w* ∗ (∑ *x*∈*support-set pf*. *prices Mkt x* (*Suc n*) *w* ∗ *pf x* (*Suc n*) *w*)
unfolding *discounted-value-def cls-val-process-def* **using** *assms* **unfolding** *trading-strategy-def* **by** *simp*
also have ... = (∑ *x*∈*support-set pf*. *discount-factor r* (*Suc n*) *w* ∗ *prices Mkt x* (*Suc n*) *w* ∗ *pf x* (*Suc n*) *w*)
by (*metis* (*no-types*, *lifting*) *mult.assoc sum.cong sum-distrib-left*)
finally have *discounted-value r* (*cls-val-process Mkt pf*) (*Suc n*) *w* =
(∑ *x*∈*support-set pf*. *discount-factor r* (*Suc n*) *w* ∗ *prices Mkt x* (*Suc n*) *w* ∗ *pf x* (*Suc n*) *w*) **.**
}
hence *space*: ∀ *w*∈ *space M*. *discounted-value r* (*cls-val-process Mkt pf*) (*Suc n*) *w* =
(∑ *x*∈*support-set pf*. *discount-factor r* (*Suc n*) *w* ∗ *prices Mkt x* (*Suc n*) *w* ∗ *pf x* (*Suc n*) *w*) **by** *simp*
hence *nspace*: ∀ *w*∈ *space N*. *discounted-value r* (*cls-val-process Mkt pf*) (*Suc n*) *w* =
(∑ *x*∈*support-set pf*. *discount-factor r* (*Suc n*) *w* ∗ *prices Mkt x* (*Suc n*) *w* ∗ *pf x* (*Suc n*) *w*) **using** *assms* **by** (*simp add*:*filt-equiv-space*)

have *sup-disc*: $\forall x \in$ *support-set pf*. *AE w in N*.
(*real-cond-exp N* (*F n*) (*discounted-value r* (λ*m y*. *prices Mkt x m y* $*$ *pf x m y*) (*Suc n*))) *w* =
discounted-value r (λ*m y*. *prices Mkt x m y* $*$ *pf x* (*Suc m*) *y*) *n w* **using** *assms*
by (*simp add*:*stocks-portfolio-risk-neutral filt-equiv-imp-subalgebra stock-portfolio-def*)
have *AE w in N*. *real-cond-exp N* (*F n*) (*discounted-value r* (*cls-val-process Mkt pf*) (*Suc n*)) *w* =
real-cond-exp N (*F n*) (λ*y*. $\sum x \in$*support-set pf*. *discounted-value r* (λ*m y*. *prices Mkt x m y* $*$ *pf x m y*) (*Suc n*) *y*) *w*
proof (*rule sigma-finite-subalgebra.real-cond-exp-cong'*)
show *sigma-finite-subalgebra N* (*F n*) **using** *nsigfin* **..**
show $\forall w \in$*space N*. *discounted-value r* (*cls-val-process Mkt pf*) (*Suc n*) *w* =
(λ*y*. $\sum x \in$*support-set pf*. *discounted-value r* (λ*m y*. *prices Mkt x m y* $*$ *pf x m y*) (*Suc n*) *y*) *w* **using** *nspace*
by (*metis* (*no-types*, *lifting*) *discounted-value-def mult.assoc sum.cong*)
show (*discounted-value r* (*cls-val-process Mkt pf*) (*Suc n*)) $\in$ *borel-measurable N* **using** *assms*
using ‹$\forall$ *t*. *integrable N* (*discounted-value r* (*cls-val-process Mkt pf*) *t*)› **by** *blast*
qed
also have *AE w in N*. *real-cond-exp N* (*F n*)
(λ*y*. $\sum x \in$*support-set pf*. *discounted-value r* (λ*m y*. *prices Mkt x m y* $*$ *pf x m y*) (*Suc n*) *y*) *w* =
($\sum x \in$ *support-set pf*. (*real-cond-exp N* (*F n*) (*discounted-value r* (λ*m y*. *prices Mkt x m y* $*$ *pf x m y*) (*Suc n*))) *w*)
proof (*rule sigma-finite-subalgebra.real-cond-exp-bsum*)
show *sigma-finite-subalgebra N* (*F n*) **using** *filt-equiv-prob-space-subalgebra assms*
using *filtration filtration-def risk-neutral-prob-def subalgebra-sigma-finite* **by** *fastforce*
fix *asset*
assume *asset* $\in$ *support-set pf*
show *integrable N* (*discounted-value r* (λ*m y*. *prices Mkt asset m y* $*$ *pf asset m y*) (*Suc n*))
proof (*rule discounted-integrable*)
show *integrable N* (λ*y*. *prices Mkt asset* (*Suc n*) *y* $*$ *pf asset* (*Suc n*) *y*) **using** *assms* ‹*asset*$\in$ *support-set pf*› **by** *simp*
show *space N* = *space M* **using** *assms* **by** (*metis filt-equiv-space*)
show $-1 < r$ **using** *acceptable-rate* **by** *simp*
qed
qed
also have *AE w in N*.
($\sum x \in$ *support-set pf*. (*real-cond-exp N* (*F n*) (*discounted-value r* (λ*m y*. *prices Mkt x m y* $*$ *pf x m y*) (*Suc n*))) *w*) =
($\sum x \in$ *support-set pf*. *discounted-value r* (λ*m y*. *prices Mkt x m y* $*$ *pf x* (*Suc m*) *y*) *n w*)
proof (*rule AE-sum*)
show *finite* (*support-set pf*) **using** *assms*(*3*) *portfolio-def trading-strategy-def*

by *auto*
show $\forall x \in$ *support-set pf*. *AE w in N*.
(*real-cond-exp N* (*F n*) (*discounted-value r* ($\lambda m\ y$. *prices Mkt x m y* $*$ *pf x m y*) (*Suc n*))) *w* =
discounted-value r ($\lambda m\ y$. *prices Mkt x m y* $*$ *pf x* (*Suc m*) *y*) *n w* **using** *sup-disc* **by** *simp*
qed
also have *AE w in N*.
($\sum x \in$ *support-set pf*. *discounted-value r* ($\lambda m\ y$. *prices Mkt x m y* $*$ *pf x* (*Suc m*) *y*) *n w*) =
discounted-value r (*cls-val-process Mkt pf*) *n w*
proof
fix *w*
assume *w*$\in$ *space N*
have ($\sum x \in$ *support-set pf*. *discounted-value r* ($\lambda m\ y$. *prices Mkt x m y* $*$ *pf x* (*Suc m*) *y*) *n w*) =
discounted-value r ($\lambda m\ y$. ($\sum x \in$ *support-set pf*. *prices Mkt x m y* $*$ *pf x* (*Suc m*) *y*)) *n w* **using** *discounted-sum assms*(*3*) *portfolio-def trading-strategy-def* **by** (*simp add*: *discounted-value-def sum-distrib-left*)
also have ... = *discounted-value r* (*val-process Mkt pf*) *n w* **unfolding** *val-process-def*
by (*simp add*: *portfolio-def*)
also have ... = *discounted-value r* (*cls-val-process Mkt pf*) *n w* **using** *assms*
by (*simp add*:*self-financingE discounted-cong*)
finally show ($\sum x \in$ *support-set pf*. *discounted-value r* ($\lambda m\ y$. *prices Mkt x m y* $*$ *pf x* (*Suc m*) *y*) *n w*) =
discounted-value r (*cls-val-process Mkt pf*) *n w* **.**
qed
finally show *AE w in N*. *real-cond-exp N* (*F n*) (*discounted-value r* (*cls-val-process Mkt pf*) (*Suc n*)) *w* =
discounted-value r (*cls-val-process Mkt pf*) *n w* **.**
qed
qed
qed

lemma (**in** *disc-filtr-prob-space*) *finite-integrable-vp*:
assumes $\forall n.\ \forall$ *asset* $\in$ *support-set pf*. *finite* (*prices Mkt asset n* '(*space M*))
and $\forall n.\ \forall$ *asset* $\in$ *support-set pf*. *finite* (*pf asset n* '(*space M*))
and *prob-space N*
and *filt-equiv F M N*
and *trading-strategy pf*
and $\forall n.\ \forall$ *asset* $\in$ *support-set pf*. *prices Mkt asset n* $\in$ *borel-measurable M*
shows $\forall n.\ \forall$ *asset*$\in$*support-set pf*. *integrable N* (λw. *prices Mkt asset n w* $*$ *pf asset* (*Suc n*) *w*)
proof (*intro allI ballI*)
fix *n*
fix *asset*
assume *asset*$\in$*support-set pf*

show *integrable N* (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*)
proof (*rule prob-space.finite-borel-measurable-integrable*)
show *prob-space N* **using** *assms* **by** *simp*
have *finite* ((λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*)
proof −
have ∀ *y*∈ *prices Mkt asset n* ‘(*space M*). *finite* ((λ *z*. (λ*w*. *z* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) *y*)
by (*metis* ‹*asset* ∈ *support-set pf*› *assms*(2) *finite-imageI image-image*)
hence *finite* (⋃ *y*∈ *prices Mkt asset n* ‘(*space M*). ((λ *z*. (λ*w*. *z* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) *y*))
using ‹*asset* ∈ *support-set pf*› *assms* **by** *blast*
moreover have (⋃ *y*∈ *prices Mkt asset n* ‘(*space M*). ((λ *z*. (λ*w*. *z* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) *y*)) =
(⋃ *y*∈ *prices Mkt asset n* ‘(*space M*). (λ*w*. *y* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) **by** *simp*
moreover have ((λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) ⊆
(⋃ *y*∈ *prices Mkt asset n* ‘(*space M*). (λ*w*. *y* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*)
proof
fix *x*
assume *x* ∈ (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*
show *x* ∈ (⋃ *y*∈*prices Mkt asset n* ‘ *space M*. (λ*w*. *y* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*)
using ‹*x* ∈ (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*› **by** *auto*
qed
ultimately show *?thesis* **by** (*simp add*:*finite-subset*)
qed
thus *finite* ((λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space N*) **using** *assms* **by** (*simp add*:*filt-equiv-space*)
have (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*) ∈ *borel-measurable M*
proof −
have *prices Mkt asset n* ∈ *borel-measurable M* **using** *assms* ‹*asset* ∈ *support-set pf*› **by** *simp*
moreover have *pf asset* (*Suc n*) ∈ *borel-measurable M* **using** *assms* **unfolding** *trading-strategy-def*
using ‹*asset* ∈ *support-set pf*› *borel-predict-stoch-proc-borel-measurable* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed
thus (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*) ∈ *borel-measurable N* **using** *assms* **by** (*simp add*:*filt-equiv-measurable*)
qed
qed

lemma (**in** *disc-filtr-prob-space*) *finite-integrable-uvp*:
assumes ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*prices Mkt asset n* ‘(*space M*))

and $\forall\, n.\ \forall\ asset \in support\text{-}set\ pf.\ finite\ (pf\ asset\ n\ `(space\ M))$
and *prob-space N*
and *filt-equiv F M N*
and *trading-strategy pf*
and $\forall\, n.\ \forall\ asset \in support\text{-}set\ pf.\ prices\ Mkt\ asset\ n \in borel\text{-}measurable\ M$
shows $\forall\, n.\ \forall\, asset{\in}support\text{-}set\ pf.\ integrable\ N\ (\lambda w.\ prices\ Mkt\ asset\ (Suc\ n)\ w * pf\ asset\ (Suc\ n)\ w)$
proof (*intro allI ballI*)
fix *n*
fix *asset*
assume *asset∈support-set pf*
show *integrable N* ($\lambda w.$ *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*)
proof (*rule prob-space.finite-borel-measurable-integrable*)
show *prob-space N* **using** *assms* **by** *simp*
have *finite* (($\lambda w.$ *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*)
proof −
have $\forall\, y{\in}$ *prices Mkt asset* (*Suc n*) ‘(*space M*). *finite* ((λ *z*. ($\lambda w.$ *z* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) *y*)
by (*metis ‹asset ∈ support-set pf› assms*(2) *finite-imageI image-image*)
hence *finite* ($\bigcup$ *y*∈ *prices Mkt asset* (*Suc n*) ‘(*space M*). ((λ *z*. ($\lambda w.$ *z* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) *y*))
using *‹asset ∈ support-set pf› assms* **by** *blast*
moreover have ($\bigcup$ *y*∈ *prices Mkt asset* (*Suc n*) ‘(*space M*). ((λ *z*. ($\lambda w.$ *z* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) *y*)) =
($\bigcup$ *y*∈ *prices Mkt asset* (*Suc n*) ‘(*space M*). ($\lambda w.$ *y* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) **by** *simp*
moreover have (($\lambda w.$ *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) ⊆
($\bigcup$ *y*∈ *prices Mkt asset* (*Suc n*) ‘(*space M*). ($\lambda w.$ *y* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*)
proof
fix *x*
assume *x* ∈ ($\lambda w.$ *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*
show *x* ∈ ($\bigcup$ *y*∈*prices Mkt asset* (*Suc n*) ‘ *space M*. ($\lambda w.$ *y* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*)
using *‹x* ∈ ($\lambda w.$ *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M›* **by** *auto*
qed
ultimately show *?thesis* **by** (*simp add:finite-subset*)
qed
thus *finite* (($\lambda w.$ *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space N*) **using** *assms* **by** (*simp add:filt-equiv-space*)
have ($\lambda w.$ *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*) ∈ *borel-measurable M*
proof −
have *prices Mkt asset* (*Suc n*) ∈ *borel-measurable M* **using** *assms*
using *‹asset ∈ support-set pf› borel-adapt-stoch-proc-borel-measurable* **by** *blast*

moreover have *pf asset* (*Suc n*) ∈ *borel-measurable M* **using** *assms* **unfolding** *trading-strategy-def*
using ‹*asset* ∈ *support-set pf*› *borel-predict-stoch-proc-borel-measurable* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed
thus (λ*w*. *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*) ∈ *borel-measurable N* **using** *assms* **by** (*simp add*:*filt-equiv-measurable*)
qed
qed

lemma (**in** *rfr-disc-equity-market*) *self-fin-trad-strat-mart-finite*:
assumes *risk-neutral-prob N*
and *filt-equiv F M N*
and *trading-strategy pf*
and *self-financing Mkt pf*
and *support-set pf* ⊆ *stocks Mkt*
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*prices Mkt asset n* ‘(*space M*))
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*pf asset n* ‘(*space M*))
and ∀ *asset*∈ *stocks Mkt*. *borel-adapt-stoch-proc F* (*prices Mkt asset*)
shows *martingale N F* (*discounted-value r* (*cls-val-process Mkt pf*))
proof (*rule self-fin-trad-strat-mart*, (*simp add*:*assms*)+)
show ∀ *n*. ∀ *asset*∈*support-set pf*. *integrable N* (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*)
proof (*intro allI ballI*)
fix *n*
fix *asset*
assume *asset*∈*support-set pf*
show *integrable N* (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*)
proof (*rule prob-space.finite-borel-measurable-integrable*)
show *prob-space N* **using** *assms* **unfolding** *risk-neutral-prob-def* **by** *auto*
have *finite* ((λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*)
proof −
have ∀ *y*∈ *prices Mkt asset n* ‘(*space M*). *finite* ((λ *z*. (λ*w*. *z* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) *y*)
by (*metis* ‹*asset* ∈ *support-set pf*› *assms*(*7*) *finite-imageI image-image*)
hence *finite* (⋃ *y*∈ *prices Mkt asset n* ‘(*space M*). ((λ *z*. (λ*w*. *z* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) *y*))
using ‹*asset* ∈ *support-set pf*› *assms*(*6*) **by** *blast*
moreover have (⋃ *y*∈ *prices Mkt asset n* ‘(*space M*). ((λ *z*. (λ*w*. *z* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) *y*)) =
(⋃ *y*∈ *prices Mkt asset n* ‘(*space M*). (λ*w*. *y* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) **by** *simp*
moreover have ((λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*) ⊆
(⋃ *y*∈ *prices Mkt asset n* ‘(*space M*). (λ*w*. *y* ∗ *pf asset* (*Suc n*) *w*) ‘ *space M*)
proof
fix *x*

assume $x \in (\lambda w.\ prices\ Mkt\ asset\ n\ w * pf\ asset\ (Suc\ n)\ w)\ `\ space\ M$
show $x \in (\bigcup y \in prices\ Mkt\ asset\ n\ `\ space\ M.\ (\lambda w.\ y * pf\ asset\ (Suc\ n)\ w)\ `\ space\ M)$
using ‹$x \in (\lambda w.\ prices\ Mkt\ asset\ n\ w * pf\ asset\ (Suc\ n)\ w)\ `\ space\ M$› **by** *auto*
qed
ultimately show *?thesis* **by** (*simp add*:*finite-subset*)
qed
thus *finite* ($(\lambda w.\ prices\ Mkt\ asset\ n\ w * pf\ asset\ (Suc\ n)\ w)\ `\ space\ N$) **using** *assms* **by** (*simp add*:*filt-equiv-space*)
have $(\lambda w.\ prices\ Mkt\ asset\ n\ w * pf\ asset\ (Suc\ n)\ w) \in borel\text{-}measurable\ M$
proof −
have *prices Mkt asset n* ∈ *borel-measurable M* **using** *assms readable*
using ‹*asset* ∈ *support-set pf*› *borel-adapt-stoch-proc-borel-measurable* **by** *blast*
moreover have *pf asset* (*Suc n*) ∈ *borel-measurable M* **using** *assms* **unfolding** *trading-strategy-def*
using ‹*asset* ∈ *support-set pf*› *borel-predict-stoch-proc-borel-measurable* **by** *blast*
ultimately show *?thesis* **by** *simp*
qed
thus $(\lambda w.\ prices\ Mkt\ asset\ n\ w * pf\ asset\ (Suc\ n)\ w) \in borel\text{-}measurable\ N$ **using** *assms* **by** (*simp add*:*filt-equiv-measurable*)
qed
qed
show $\forall n.\ \forall asset \in support\text{-}set\ pf.\ integrable\ N\ (\lambda w.\ prices\ Mkt\ asset\ (Suc\ n)\ w * pf\ asset\ (Suc\ n)\ w)$
proof (*intro allI ballI*)
fix *n*
fix *asset*
assume *asset*∈*support-set pf*
show *integrable N* ($\lambda w.\ prices\ Mkt\ asset\ (Suc\ n)\ w * pf\ asset\ (Suc\ n)\ w$)
proof (*rule prob-space.finite-borel-measurable-integrable*)
show *prob-space N* **using** *assms* **unfolding** *risk-neutral-prob-def* **by** *auto*
have *finite* ($(\lambda w.\ prices\ Mkt\ asset\ (Suc\ n)\ w * pf\ asset\ (Suc\ n)\ w)\ `\ space\ M$)
proof −
have $\forall y \in prices\ Mkt\ asset\ (Suc\ n)\ `(space\ M).\ finite\ ((\lambda\ z.\ (\lambda w.\ z * pf\ asset\ (Suc\ n)\ w)\ `\ space\ M)\ y)$
by (*metis* ‹*asset* ∈ *support-set pf*› *assms*(7) *finite-imageI image-image*)
hence *finite* $(\bigcup\ y \in prices\ Mkt\ asset\ (Suc\ n)\ `(space\ M).\ ((\lambda\ z.\ (\lambda w.\ z * pf\ asset\ (Suc\ n)\ w)\ `\ space\ M)\ y))$
using ‹*asset* ∈ *support-set pf*› *assms*(6) **by** *blast*
moreover have $(\bigcup\ y \in prices\ Mkt\ asset\ (Suc\ n)\ `(space\ M).\ ((\lambda\ z.\ (\lambda w.\ z * pf\ asset\ (Suc\ n)\ w)\ `\ space\ M)\ y)) =$
$(\bigcup\ y \in prices\ Mkt\ asset\ (Suc\ n)\ `(space\ M).\ (\lambda w.\ y * pf\ asset\ (Suc\ n)\ w)\ `\ space\ M)$ **by** *simp*
moreover have $((\lambda w.\ prices\ Mkt\ asset\ (Suc\ n)\ w * pf\ asset\ (Suc\ n)\ w)\ `\ space\ M) \subseteq$
$(\bigcup\ y \in prices\ Mkt\ asset\ (Suc\ n)\ `(space\ M).\ (\lambda w.\ y * pf\ asset\ (Suc\ n)\ w)$

‘ space M)
proof
fix x
assume x ∈ (λw. prices Mkt asset (Suc n) w ∗ pf asset (Suc n) w) ‘ space M
show x ∈ (⋃ y∈prices Mkt asset (Suc n) ‘ space M. (λw. y ∗ pf asset (Suc n) w) ‘ space M)
using ‹x ∈ (λw. prices Mkt asset (Suc n) w ∗ pf asset (Suc n) w) ‘ space M› by auto
qed
ultimately show ?thesis by (simp add:finite-subset)
qed
thus finite ((λw. prices Mkt asset (Suc n) w ∗ pf asset (Suc n) w) ‘ space N) using assms by (simp add:filt-equiv-space)
have (λw. prices Mkt asset (Suc n) w ∗ pf asset (Suc n) w) ∈ borel-measurable M
proof −
have prices Mkt asset (Suc n) ∈ borel-measurable M using assms readable
using ‹asset ∈ support-set pf› borel-adapt-stoch-proc-borel-measurable by blast
moreover have pf asset (Suc n) ∈ borel-measurable M using assms unfolding trading-strategy-def
using ‹asset ∈ support-set pf› borel-predict-stoch-proc-borel-measurable by blast
ultimately show ?thesis by simp
qed
thus (λw. prices Mkt asset (Suc n) w ∗ pf asset (Suc n) w) ∈ borel-measurable N using assms by (simp add:filt-equiv-measurable)
qed
qed
show stock-portfolio Mkt pf using assms stock-portfolio-def
by (simp add: stock-portfolio-def trading-strategy-def)
qed

lemma (in rfr-disc-equity-market) replicating-expectation:
assumes risk-neutral-prob N
and filt-equiv F M N
and replicating-portfolio pf pyf matur
and ∀ n. ∀ asset ∈ support-set pf. integrable N (λw. prices Mkt asset n w ∗ pf asset (Suc n) w)
and ∀ n. ∀ asset ∈ support-set pf. integrable N (λw. prices Mkt asset (Suc n) w ∗ pf asset (Suc n) w)
and viable-market Mkt
and sets (F 0) = {{}, space M}
and pyf ∈ borel-measurable (F matur)
shows fair-price Mkt (prob-space.expectation N (discounted-value r (λm. pyf) matur)) pyf matur
proof −

have *fn*: *filtrated-prob-space N F* **using** *assms*
by (*simp add*: ‹*pyf* ∈ *borel-measurable* (*F matur*)› *filtrated-prob-space-axioms.intro filtrated-prob-space-def risk-neutral-prob-def filt-equiv-filtration*)
have *discounted-value r* (*cls-val-process Mkt pf*) *matur* ∈ *borel-measurable N*
using *assms*(*3*) *disc-equity-market.replicating-portfolio-def disc-equity-market-axioms discounted-adapted*
filtrated-prob-space.borel-adapt-stoch-proc-borel-measurable fn cls-val-process-adapted
by (*metis* (*no-types*, *opaque-lifting*) *support-adapt-def readable stock-portfolio-def subsetCE*)
have *discounted-value r* (λ*m*. *pyf*) *matur* ∈ *borel-measurable N*
proof −
have (λ*m*. *pyf*) *matur* ∈ *borel-measurable* (*F matur*) **using** *assms* **by** *simp*
hence (λ*m*. *pyf*) *matur* ∈ *borel-measurable M* **using** *filtration filtrationE1 measurable-from-subalg* **by** *blast*
hence (λ*m*. *pyf*) *matur* ∈ *borel-measurable N* **using** *assms* **by** (*simp add*:*filt-equiv-measurable*)
thus *?thesis* **by** (*simp add*:*discounted-measurable*)
qed
have *mpyf*: *AE w in M*. *cls-val-process Mkt pf matur w* = *pyf w* **using** *assms* **unfolding** *replicating-portfolio-def* **by** *simp*
have *AE w in N*. *cls-val-process Mkt pf matur w* = *pyf w*
proof (*rule filt-equiv-borel-AE-eq*)
show *filt-equiv F M N* **using** *assms* **by** *simp*
show *pyf* ∈ *borel-measurable* (*F matur*) **using** *assms* **by** *simp*
show *AE w in M*. *cls-val-process Mkt pf matur w* = *pyf w* **using** *mpyf* **by** *simp*
show *cls-val-process Mkt pf matur* ∈ *borel-measurable* (*F matur*)
using *assms*(*3*) *price-structure-def replicating-price-process*
by (*meson support-adapt-def disc-equity-market.replicating-portfolio-def disc-equity-market-axioms readable stock-portfolio-def subsetCE*)
qed
hence *disc*:*AE w in N*. *discounted-value r* (*cls-val-process Mkt pf*) *matur w* = *discounted-value r* (λ*m*. *pyf*) *matur w*
by (*simp add*:*discounted-AE-cong*)
have *AEeq N* (*real-cond-exp N* (*F 0*) (*discounted-value r* (*cls-val-process Mkt pf*) *matur*))
(*real-cond-exp N* (*F 0*) (*discounted-value r* (λ*m*. *pyf*) *matur*))
proof (*rule sigma-finite-subalgebra.real-cond-exp-cong*)
show *sigma-finite-subalgebra N* (*F 0*)
using *filtrated-prob-space.axioms*(*1*) *filtrated-prob-space.filtration fn filtrationE1 prob-space.subalgebra-sigma-finite* **by** *blast*
show *AEeq N* (*discounted-value r* (*cls-val-process Mkt pf*) *matur*) (*discounted-value r* (λ*m*. *pyf*) *matur*) **using** *disc* **by** *simp*
show *discounted-value r* (*cls-val-process Mkt pf*) *matur* ∈ *borel-measurable N*
using ‹*discounted-value r* (*cls-val-process Mkt pf*) *matur* ∈ *borel-measurable N*› **.**
show *discounted-value r* (λ*m*. *pyf*) *matur* ∈ *borel-measurable N*
using ‹*discounted-value r* (λ*m*. *pyf*) *matur* ∈ *borel-measurable N*› **.**
qed
have *martingale N F* (*discounted-value r* (*cls-val-process Mkt pf*)) **using** *assms*

unfolding *replicating-portfolio-def*
using *self-fin-trad-strat-mart*[*of N pf*] **by** (*simp add*: *stock-portfolio-def*)
hence *AEeq N* (*real-cond-exp N* (*F 0*) (*discounted-value r* (*cls-val-process Mkt pf*) *matur*))
(*discounted-value r* (*cls-val-process Mkt pf*) *0*) **using** *martingaleAE*[*of N F discounted-value r* (*cls-val-process Mkt pf*) *0 matur*]
fn **by** *simp*
also have *AE w in N*. (*discounted-value r* (*cls-val-process Mkt pf*) *0 w*) = *initial-value pf*
proof
fix *w*
assume *w*∈ *space N*
have *discounted-value r* (*cls-val-process Mkt pf*) *0 w* = *cls-val-process Mkt pf 0 w* **by** (*simp add*:*discounted-init*)
also have ... = *val-process Mkt pf 0 w* **unfolding** *cls-val-process-def* **using** *assms*
unfolding *replicating-portfolio-def stock-portfolio-def* **by** *simp*
also have ... = *initial-value pf* **using** *assms* **unfolding** *replicating-portfolio-def* **using** ‹*w*∈ *space N*›
by (*metis* (*no-types*, *lifting*) *support-adapt-def filt-equiv-space initial-valueI readable stock-portfolio-def subsetCE*)
finally show *discounted-value r* (*cls-val-process Mkt pf*) *0 w* = *initial-value pf*
.
qed
finally have *AE w in N*. (*real-cond-exp N* (*F 0*) (*discounted-value r* (*cls-val-process Mkt pf*) *matur*)) *w* =
initial-value pf .
moreover have ∀ *w*∈ *space N*. (*real-cond-exp N* (*F 0*) (*discounted-value r* (*cls-val-process Mkt pf*) *matur*)) *w* =
prob-space.expectation N (*discounted-value r* (*cls-val-process Mkt pf*) *matur*)
proof (*rule prob-space.trivial-subalg-cond-expect-eq*)
show *prob-space N* **using** *assms* **unfolding** *risk-neutral-prob-def* **by** *simp*
show *subalgebra N* (*F 0*)
using ‹*prob-space N*› *filtrated-prob-space.filtration fn filtrationE1* **by** *blast*
show *sets* (*F 0*) = {{}, *space N*} **using** *assms* **by** (*simp add*:*filt-equiv-space*)
show *integrable N* (*discounted-value r* (*cls-val-process Mkt pf*) *matur*)
proof (*rule discounted-integrable*)
show *space N* = *space M* **using** *assms* **by** (*simp add*:*filt-equiv-space*)
show *integrable N* (*cls-val-process Mkt pf matur*) **using** *assms* **unfolding** *replicating-portfolio-def*
by (*simp add*: *integrable-self-fin-uvp*)
show −*1* < *r* **using** *acceptable-rate* **by** *simp*
qed
qed
ultimately have *AE w in N*. *prob-space.expectation N* (*discounted-value r* (*cls-val-process Mkt pf*) *matur*) =
initial-value pf **by** *simp*
hence *prob-space.expectation N* (*discounted-value r* (*cls-val-process Mkt pf*) *matur*)
=

initial-value pf **using** *assms* **unfolding** *risk-neutral-prob-def* **using** *prob-space.emeasure-space-1*[*of N*]
AE-eq-cst[*of - - N*] **by** *simp*
moreover have *prob-space.expectation N* (*discounted-value r* (*cls-val-process Mkt pf*) *matur*) =
prob-space.expectation N (*discounted-value r* (λ*m. pyf*) *matur*)
proof (*rule integral-cong-AE*)
show *AEeq N* (*discounted-value r* (*cls-val-process Mkt pf*) *matur*) (*discounted-value r* (λ*m. pyf*) *matur*)
using *disc* **by** *simp*
show *discounted-value r* (λ*m. pyf*) *matur* ∈ *borel-measurable N*
using ‹*discounted-value r* (λ*m. pyf*) *matur* ∈ *borel-measurable N*› **.**
show *discounted-value r* (*cls-val-process Mkt pf*) *matur* ∈ *borel-measurable N*
using ‹*discounted-value r* (*cls-val-process Mkt pf*) *matur* ∈ *borel-measurable N*› **.**
qed
ultimately have *prob-space.expectation N* (*discounted-value r* (λ*m. pyf*) *matur*) = *initial-value pf* **by** *simp*
thus *?thesis* **using** *assms*
by (*metis* (*full-types*) *support-adapt-def disc-equity-market.replicating-portfolio-def disc-equity-market-axioms*
readable replicating-fair-price stock-portfolio-def subsetCE)
qed

lemma (**in** *rfr-disc-equity-market*) *replicating-expectation-finite*:
assumes *risk-neutral-prob N*
and *filt-equiv F M N*
and *replicating-portfolio pf pyf matur*
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*prices Mkt asset n* ‘(*space M*))
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*pf asset n* ‘(*space M*))
and *viable-market Mkt*
and *sets* (*F 0*) = {{}, *space M*}
and *pyf* ∈ *borel-measurable* (*F matur*)
shows *fair-price Mkt* (*prob-space.expectation N* (*discounted-value r* (λ*m. pyf*) *matur*)) *pyf matur*
proof −
have ∀ *n*. ∀ *asset* ∈ *support-set pf*. *integrable N* (λ*w. prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*)
proof (*rule finite-integrable-vp*, (*auto simp add*:*assms*))
show *prob-space N* **using** *assms* **unfolding** *risk-neutral-prob-def* **by** *simp*
show *trading-strategy pf* **using** *assms* **unfolding** *replicating-portfolio-def* **by** *simp*
show ⋀*n asset*. *asset* ∈ *support-set pf* ⟹ *random-variable borel* (*prices Mkt asset n*)
proof−
fix *n*
fix *asset*
assume *asset* ∈ *support-set pf*

show *random-variable borel* (*prices Mkt asset n*)
using *assms* **unfolding** *replicating-portfolio-def stock-portfolio-def adapt-stoch-proc-def*
using *readable*
by (*meson* ‹*asset* ∈ *support-set pf*› *adapt-stoch-proc-borel-measurable subsetCE*)
qed
qed
moreover have ∀ *n*. ∀ *asset* ∈ *support-set pf*. *integrable N* (λ*w*. *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*)
proof (*rule finite-integrable-uvp*, (*auto simp add*:*assms*))
show *prob-space N* **using** *assms* **unfolding** *risk-neutral-prob-def* **by** *simp*
show *trading-strategy pf* **using** *assms* **unfolding** *replicating-portfolio-def* **by** *simp*
show ⋀*n asset*. *asset* ∈ *support-set pf* ⟹ *random-variable borel* (*prices Mkt asset n*)
proof−
fix *n*
fix *asset*
assume *asset* ∈ *support-set pf*
show *random-variable borel* (*prices Mkt asset n*)
using *assms* **unfolding** *replicating-portfolio-def stock-portfolio-def adapt-stoch-proc-def*
using *readable*
by (*meson* ‹*asset* ∈ *support-set pf*› *adapt-stoch-proc-borel-measurable subsetCE*)
qed
qed
ultimately show *?thesis* **using** *assms replicating-expectation* **by** *simp*
qed

end

8 The Cox Ross Rubinstein model

This section defines the Cox-Ross-Rubinstein model of a financial market, and charcterizes a risk-neutral probability space for this market. This, together with the proof that every derivative is attainable, permits to obtain a formula to explicitely compute the fair price of any derivative.

theory *CRR-Model* **imports** *Fair-Price*

begin

locale *CRR-hyps* = *prob-grw* + *rsk-free-asset* +
fixes *stk*
assumes *stocks*: *stocks Mkt* = {*stk*, *risk-free-asset*}
and *stk-price*: *prices Mkt stk* = *geom-proc*
and *S0-positive*: *0* < *init*

and *down-positive*: $0 < d$ **and** *down-lt-up*: $d < u$
and *psgt*: $0 < p$
and *pslt*: $p < 1$

locale *CRR-market* = *CRR-hyps* +
fixes G
assumes *stock-filtration*:G = *stoch-proc-filt M geom-proc borel*

8.1 Preliminary results on the market

lemma (**in** *CRR-market*) *case-asset*:
assumes *asset* ∈ *stocks Mkt*
shows *asset* = *stk* ∨ *asset* = *risk-free-asset*
proof (*rule ccontr*)
assume ¬ (*asset* = *stk* ∨ *asset* = *risk-free-asset*)
hence *asset* ≠ *stk* ∧ *asset* ≠ *risk-free-asset* **by** *simp*
moreover have *asset* ∈ {*stk*, *risk-free-asset*} **using** *assms stocks* **by** *simp*
ultimately show *False* **by** *auto*
qed

lemma (**in** *CRR-market*)
assumes N = *bernoulli-stream q*
and $0 < q$
and $q < 1$
shows *bernoulli-gen-filtration*: *filtration N G*
and *bernoulli-sigma-finite*: ∀ *n*. *sigma-finite-subalgebra N* (*G n*)
proof −
show *filtration N G*
proof −
have *disc-filtr M* (*stoch-proc-filt M geom-proc borel*)
proof (*rule stoch-proc-filt-disc-filtr*)
fix *i*
show *random-variable borel* (*geom-proc i*)
by (*simp add*: *geom-rand-walk-borel-measurable*)
qed
hence *filtration M G* **using** *stock-filtration* **by** (*simp add*: *filtration-def disc-filtr-def*)
have *filt-equiv nat-filtration M N* **using** *pslt psgt* **by** (*simp add*: *assms bernoulli-stream-equiv*)
hence *sets N* = *sets M* **unfolding** *filt-equiv-def* **by** *simp*
thus *?thesis* **unfolding** *filtration-def*
by (*metis filtration-def* ‹*Filtration.filtration M G*› *sets-eq-imp-space-eq subalgebra-def*)
qed
show ∀ *n*. *sigma-finite-subalgebra N* (*G n*) **using** *assms* **unfolding** *subalgebra-def*
using *filtration-def subalgebra-sigma-finite*
by (*metis* ‹*Filtration.filtration N G*› *bernoulli-stream-def prob-space.prob-space-stream-space prob-space.subalgebra-sigma-finite prob-space-measure-pmf*)
qed

sublocale *CRR-market* ⊆ *rfr-disc-equity-market - G*
proof (*unfold-locales*)
show *disc-filtr M G* ∧ *sets* (*G* ⊥) = {{}, *space M*}
proof
show *sets* (*G* ⊥) = {{}, *space M*} **using** *infinite-cts-filtration.stoch-proc-filt-triv-init stock-filtration geometric-process geom-rand-walk-borel-adapted*
by (*meson infinite-coin-toss-space-axioms infinite-cts-filtration-axioms.intro infinite-cts-filtration-def init-triv-filt-def*)
show *disc-filtr M G*
by (*metis Filtration.filtration-def bernoulli bernoulli-gen-filtration disc-filtr-def psgt pslt*)
qed
show ∀ *asset*∈*stocks Mkt*. *borel-adapt-stoch-proc G* (*prices Mkt asset*)
proof −
have *borel-adapt-stoch-proc G* (*prices Mkt stk*) **using** *stk-price stock-filtration stoch-proc-filt-adapt*
by (*simp add*: *stoch-proc-filt-adapt geom-rand-walk-borel-measurable*)
moreover have *borel-adapt-stoch-proc G* (*prices Mkt risk-free-asset*)
using ‹*disc-filtr M G* ∧ *sets* (*G* ⊥) = {{}, *space M*}› *disc-filtr-prob-space.disc-rfr-proc-borel-adapted disc-filtr-prob-space.intro disc-filtr-prob-space-axioms.intro prob-space-axioms rf-price* **by** *fastforce*
moreover have *disc-filtr-prob-space M G* **proof** (*unfold-locales*)
show *disc-filtr M G* **by** (*simp add*: ‹*disc-filtr M G* ∧ *sets* (*G* ⊥) = {{}, *space M*}›)
qed
ultimately show *?thesis* **using** *stocks* **by** *force*
qed
qed

lemma (**in** *CRR-market*) *two-stocks*:
shows *stk* ≠ *risk-free-asset*
proof (*rule ccontr*)
assume ¬*stk* ≠ *risk-free-asset*
hence *disc-rfr-proc r* = *prices Mkt stk* **using** *rf-price* **by** *simp*
also have ... = *geom-proc* **using** *stk-price* **by** *simp*
finally have *eqf*: *disc-rfr-proc r* = *geom-proc* **.**
hence ∀ *w*. *disc-rfr-proc r 0 w* = *geom-proc 0 w* **by** *simp*
hence *1* = *init* **using** *geometric-process* **by** *simp*
have *eqfs*: ∀ *w*. *disc-rfr-proc r* (*Suc 0*) *w* = *geom-proc* (*Suc 0*) *w* **using** *eqf* **by** *simp*
hence *disc-rfr-proc r* (*Suc 0*) (*sconst True*) = *geom-proc* (*Suc 0*) (*sconst True*) **by** *simp*
hence *1*+*r* = *u* **using** *geometric-process* ‹*1* = *init*› **by** *simp*

have *disc-rfr-proc r* (*Suc 0*) (*sconst False*) = *geom-proc* (*Suc 0*) (*sconst False*) **using** *eqfs* **by** *simp*
hence *1+r = d* **using** *geometric-process* ‹*1 = init*› **by** *simp*
show *False* **using** ‹*1+r = u*› ‹*1+r = d*› *down-lt-up* **by** *simp*
qed

lemma (**in** *CRR-market*) *stock-pf-vp-expand*:
assumes *stock-portfolio Mkt pf*
shows *val-process Mkt pf n w = geom-proc n w * pf stk* (*Suc n*) *w* +
*disc-rfr-proc r n w * pf risk-free-asset* (*Suc n*) *w*
proof −
have *val-process Mkt pf n w* =(*sum* (λ*x*. ((*prices Mkt*) *x n w*) * (*pf x* (*Suc n*) *w*)) (*stocks Mkt*))
proof (*rule subset-val-process'*)
show *finite* (*stocks Mkt*) **using** *stocks* **by** *auto*
show *support-set pf* ⊆ *stocks Mkt* **using** *assms* **unfolding** *stock-portfolio-def* **by** *simp*
qed
also have ... = (∑ *x*∈ {*stk*, *risk-free-asset*}. ((*prices Mkt*) *x n w*) * (*pf x* (*Suc n*) *w*)) **using** *stocks* **by** *simp*
also have ... = *prices Mkt stk n w * pf stk* (*Suc n*) *w* +
(∑ *x*∈ {*risk-free-asset*}. ((*prices Mkt*) *x n w*) * (*pf x* (*Suc n*) *w*)) **by** (*simp add*:*two-stocks*)
also have ... = *prices Mkt stk n w * pf stk* (*Suc n*) *w* +
*prices Mkt risk-free-asset n w * pf risk-free-asset* (*Suc n*) *w* **by** *simp*
also have ... = *geom-proc n w * pf stk* (*Suc n*) *w* + *disc-rfr-proc r n w * pf risk-free-asset* (*Suc n*) *w*
using *rf-price stk-price* **by** *simp*
finally show *?thesis* **.**
qed

lemma (**in** *CRR-market*) *stock-pf-uvp-expand*:
assumes *stock-portfolio Mkt pf*
shows *cls-val-process Mkt pf* (*Suc n*) *w = geom-proc* (*Suc n*) *w * pf stk* (*Suc n*) *w* +
disc-rfr-proc r (*Suc n*) *w * pf risk-free-asset* (*Suc n*) *w*
proof −
have *cls-val-process Mkt pf* (*Suc n*) *w* =(*sum* (λ*x*. ((*prices Mkt*) *x* (*Suc n*) *w*) * (*pf x* (*Suc n*) *w*)) (*stocks Mkt*))
proof (*rule subset-cls-val-process'*)
show *finite* (*stocks Mkt*) **using** *stocks* **by** *auto*
show *support-set pf* ⊆ *stocks Mkt* **using** *assms* **unfolding** *stock-portfolio-def* **by** *simp*
qed
also have ... = (∑ *x*∈ {*stk*, *risk-free-asset*}. ((*prices Mkt*) *x* (*Suc n*) *w*) * (*pf x* (*Suc n*) *w*)) **using** *stocks* **by** *simp*
also have ... = *prices Mkt stk* (*Suc n*) *w * pf stk* (*Suc n*) *w* +
(∑ *x*∈ {*risk-free-asset*}. ((*prices Mkt*) *x* (*Suc n*) *w*) * (*pf x* (*Suc n*) *w*)) **by**

(*simp add:two-stocks*)
also have ... = *prices Mkt stk* (*Suc n*) *w* ∗ *pf stk* (*Suc n*) *w* +
prices Mkt risk-free-asset (*Suc n*) *w* ∗ *pf risk-free-asset* (*Suc n*) *w* **by** *simp*
also have ... = *geom-proc* (*Suc n*) *w* ∗ *pf stk* (*Suc n*) *w* + *disc-rfr-proc r* (*Suc n*) *w* ∗ *pf risk-free-asset* (*Suc n*) *w*
using *rf-price stk-price* **by** *simp*
finally show *?thesis* **.**
qed

lemma (**in** *CRR-market*) *pos-pf-neg-uvp*:
assumes *stock-portfolio Mkt pf*
and *d* < *1+r*
and *0* < *pf stk* (*Suc n*) (*spick w n False*)
and *val-process Mkt pf n* (*spick w n False*) ≤ *0*
shows *cls-val-process Mkt pf* (*Suc n*) (*spick w n False*) < *0*
proof −
define *wnf* **where** *wnf* = *spick w n False*
have *cls-val-process Mkt pf* (*Suc n*) (*spick w n False*) =
geom-proc (*Suc n*) *wnf* ∗ *pf stk* (*Suc n*) *wnf* +
disc-rfr-proc r (*Suc n*) *wnf* ∗ *pf risk-free-asset* (*Suc n*) *wnf* **unfolding** *wnf-def*
using *assms* **by** (*simp add:stock-pf-uvp-expand*)
also have ... = *d* ∗ *geom-proc n wnf* ∗ *pf stk* (*Suc n*) *wnf* + *disc-rfr-proc r* (*Suc n*) *wnf* ∗ *pf risk-free-asset* (*Suc n*) *wnf*
unfolding *wnf-def* **using** *geometric-process spickI*[*of n w False*] **by** *simp*
also have ... = *d* ∗ *geom-proc n wnf* ∗ *pf stk* (*Suc n*) *wnf* + (*1+r*) ∗ *disc-rfr-proc r n wnf* ∗ *pf risk-free-asset* (*Suc n*) *wnf*
by *simp*
also have ... < (*1+r*) ∗ *geom-proc n wnf* ∗ *pf stk* (*Suc n*) *wnf* + (*1+r*) ∗ *disc-rfr-proc r n wnf* ∗ *pf risk-free-asset* (*Suc n*) *wnf*
unfolding *wnf-def* **using** *assms geom-rand-walk-strictly-positive S0-positive down-positive down-lt-up* **by** *simp*
also have ... = (*1+r*) ∗ (*geom-proc n wnf* ∗ *pf stk* (*Suc n*) *wnf* + *disc-rfr-proc r n wnf* ∗ *pf risk-free-asset* (*Suc n*) *wnf*)
by (*simp add*: *distrib-left*)
also have ... = (*1+r*) ∗ *val-process Mkt pf n wnf* **using** *stock-pf-vp-expand assms* **by** *simp*
also have ... ≤ *0*
proof −
have *0* < *1+r* **using** *assms down-positive* **by** *simp*
moreover have *val-process Mkt pf n wnf* ≤ *0* **using** *assms* **unfolding** *wnf-def* **by** *simp*
ultimately show (*1+r*) ∗ (*val-process Mkt pf n wnf*) ≤ *0* **unfolding** *wnf-def*
using *less-eq-real-def*[*of 0 1+r*] *mult-nonneg-nonpos*[*of 1+r val-process Mkt pf n* (*spick w n False*)] **by** *simp*
qed
finally show *?thesis* **.**
qed

lemma (**in** *CRR-market*) *neg-pf-neg-uvp*:
assumes *stock-portfolio Mkt pf*
and $1+r < u$
and *pf stk* (*Suc n*) (*spick w n True*) < 0
and *val-process Mkt pf n* (*spick w n True*) ≤ 0
shows *cls-val-process Mkt pf* (*Suc n*) (*spick w n True*) < 0
proof −
define *wnf* **where** *wnf* = *spick w n True*
have *cls-val-process Mkt pf* (*Suc n*) (*spick w n True*) =
geom-proc (*Suc n*) *wnf* ∗ *pf stk* (*Suc n*) *wnf* +
disc-rfr-proc r (*Suc n*) *wnf* ∗ *pf risk-free-asset* (*Suc n*) *wnf* **unfolding** *wnf-def*
using *assms* **by** (*simp add*:*stock-pf-uvp-expand*)
also have ... = *u* ∗ *geom-proc n wnf* ∗ *pf stk* (*Suc n*) *wnf* + *disc-rfr-proc r* (*Suc n*) *wnf* ∗ *pf risk-free-asset* (*Suc n*) *wnf*
unfolding *wnf-def* **using** *geometric-process spickI*[*of n w True*] **by** *simp*
also have ... = *u* ∗ *geom-proc n wnf* ∗ *pf stk* (*Suc n*) *wnf* + (*1+r*) ∗ *disc-rfr-proc r n wnf* ∗ *pf risk-free-asset* (*Suc n*) *wnf*
by *simp*
also have ... < (*1+r*) ∗ *geom-proc n wnf* ∗ *pf stk* (*Suc n*) *wnf* + (*1+r*) ∗ *disc-rfr-proc r n wnf* ∗ *pf risk-free-asset* (*Suc n*) *wnf*
unfolding *wnf-def* **using** *assms geom-rand-walk-strictly-positive S0-positive down-positive down-lt-up* **by** *simp*
also have ... = (*1+r*) ∗ (*geom-proc n wnf* ∗ *pf stk* (*Suc n*) *wnf* + *disc-rfr-proc r n wnf* ∗ *pf risk-free-asset* (*Suc n*) *wnf*)
by (*simp add*: *distrib-left*)
also have ... = (*1+r*) ∗ *val-process Mkt pf n wnf* **using** *stock-pf-vp-expand assms* **by** *simp*
also have ... ≤ 0
proof −
have $0 < 1+r$ **using** *acceptable-rate* **by** *simp*
moreover have *val-process Mkt pf n wnf* ≤ 0 **using** *assms* **unfolding** *wnf-def* **by** *simp*
ultimately show (*1+r*) ∗ (*val-process Mkt pf n wnf*) $\leq$ *0* **unfolding** *wnf-def*
using *less-eq-real-def*[*of 0 1+r*] *mult-nonneg-nonpos*[*of 1+r val-process Mkt pf n* (*spick w n True*)] **by** *simp*
qed
finally show *?thesis* **.**
qed

lemma (**in** *CRR-market*) *zero-pf-neg-uvp*:
assumes *stock-portfolio Mkt pf*
and *pf stk* (*Suc n*) *w* = *0*
and *pf risk-free-asset* (*Suc n*) $w \neq 0$
and *val-process Mkt pf n w* ≤ 0

shows *cls-val-process Mkt pf* (*Suc n*) *w* < *0*
proof −
have *cls-val-process Mkt pf* (*Suc n*) *w* =
S (*Suc n*) *w* ∗ *pf stk* (*Suc n*) *w* +
disc-rfr-proc r (*Suc n*) *w* ∗ *pf risk-free-asset* (*Suc n*) *w*
using *assms* **by** (*simp add*:*stock-pf-uvp-expand*)
also have ... = *disc-rfr-proc r* (*Suc n*) *w* ∗ *pf risk-free-asset* (*Suc n*) *w* **using** *assms* **by** *simp*
also have ... = (*1*+*r*) ∗ *disc-rfr-proc r n w* ∗ *pf risk-free-asset* (*Suc n*) *w* **by** *simp*
also have ... < *0*
proof −
have *0* < *1*+*r* **using** *acceptable-rate* **by** *simp*
moreover have *0* < *disc-rfr-proc r n w* **using** *acceptable-rate* **by** (*simp add*: *disc-rfr-proc-positive*)
ultimately have *0* < (*1*+*r*) ∗ *disc-rfr-proc r n w* **by** *simp*
have *1*: *0*< *pf risk-free-asset* (*Suc n*) *w* ⟶ *0* <(*1*+*r*) ∗ *disc-rfr-proc r n w* ∗ *pf risk-free-asset* (*Suc n*) *w*
proof (*intro impI*)
assume *0* < *pf risk-free-asset* (*Suc n*) *w*
thus *0* < (*1* + *r*) ∗ *disc-rfr-proc r n w* ∗ *pf risk-free-asset* (*Suc n*) *w* **using** ‹*0* < (*1*+*r*) ∗ *disc-rfr-proc r n w*›
by *simp*
qed
have *2*: *pf risk-free-asset* (*Suc n*) *w* < *0* ⟶ (*1*+*r*) ∗ *disc-rfr-proc r n w* ∗ *pf risk-free-asset* (*Suc n*) *w* < *0*
proof (*intro impI*)
assume *pf risk-free-asset* (*Suc n*) *w* < *0*
thus (*1* + *r*) ∗ *disc-rfr-proc r n w* ∗ *pf risk-free-asset* (*Suc n*) *w* < *0* **using** ‹*0* < (*1*+*r*) ∗ *disc-rfr-proc r n w*›
by (*simp add*:*mult-pos-neg*)
qed
have *0* ≥ *val-process Mkt pf n w* **using** *assms* **by** *simp*
also have *val-process Mkt pf n w* = *geom-proc n w* ∗ *pf stk* (*Suc n*) *w* +
disc-rfr-proc r n w ∗ *pf risk-free-asset* (*Suc n*) *w* **using** *assms* **by** (*simp add*:*stock-pf-vp-expand*)
also have ... = *disc-rfr-proc r n w* ∗ *pf risk-free-asset* (*Suc n*) *w* **using** *assms* **by** *simp*
finally have *0*≥ *disc-rfr-proc r n w* ∗ *pf risk-free-asset* (*Suc n*) *w* **.**
have *0*< *pf risk-free-asset* (*Suc n*) *w* ∨ *pf risk-free-asset* (*Suc n*) *w* < *0* **using** *assms*
by *linarith*
thus *?thesis*
using *2* ‹*0* < *disc-rfr-proc r n w*› ‹*disc-rfr-proc r n w* ∗ *pf risk-free-asset* (*Suc n*) *w* ≤ *0*›
mult-pos-pos **by** *fastforce*
qed
finally show *?thesis* **.**
qed

lemma (**in** *CRR-market*) *neg-pf-exists*:
assumes *stock-portfolio Mkt pf*
and *trading-strategy pf*
and *1+r < u*
and *d < 1+r*
and *val-process Mkt pf n w ≤ 0*
and *pf stk (Suc n) w ≠ 0 ∨ pf risk-free-asset (Suc n) w ≠ 0*
shows ∃ *y*. *cls-val-process Mkt pf (Suc n) y < 0*
proof −
have *borel-predict-stoch-proc G (pf stk)*
proof (*rule inc-predict-support-trading-strat′*)
show *trading-strategy pf* **using** *assms* **by** *simp*
show *stk ∈ support-set pf ∪ {stk}* **by** *simp*
qed
hence *pf stk (Suc n) ∈ borel-measurable (G n)* **unfolding** *predict-stoch-proc-def*
by *simp*
have *val-process Mkt pf n ∈ borel-measurable (G n)*
proof −
have *borel-adapt-stoch-proc G (val-process Mkt pf)* **using** *assms*
using *support-adapt-def ats-val-process-adapted readable* **unfolding** *stock-portfolio-def*
by *blast*
thus *?thesis* **unfolding** *adapt-stoch-proc-def* **by** *simp*
qed
define *wn* **where** *wn = pseudo-proj-True n w*
show *?thesis*
proof (*cases pf stk (Suc n) w ≠ 0*)
case *True*
show *?thesis*
proof (*cases pf stk (Suc n) w > 0*)
case *True*
have *0 <pf stk (Suc n) (spick wn n False)*
proof −
have *0 < pf stk (Suc n) w* **using** ‹*0 < pf stk (Suc n) w*› **by** *simp*
also have *... = pf stk (Suc n) wn* **unfolding** *wn-def*
using ‹*pf stk (Suc n) ∈ borel-measurable (G n)*› *stoch-proc-subalg-nat-filt*[*of geom-proc*] *geometric-process*
nat-filtration-info stock-filtration
by (*metis comp-apply geom-rand-walk-borel-adapted measurable-from-subalg*)
also have *... = pf stk (Suc n) (spick wn n False)* **using** ‹*pf stk (Suc n) ∈ borel-measurable (G n)*› *comp-def nat-filtration-info*
pseudo-proj-True-stake-image spickI stoch-proc-subalg-nat-filt[*of geom-proc*] *geometric-process stock-filtration*
by (*metis geom-rand-walk-borel-adapted measurable-from-subalg*)
finally show *?thesis* **.**
qed
moreover have *0 ≥ val-process Mkt pf n (spick wn n False)*

proof −
have *0* ≥ *val-process Mkt pf n w* **using** *assms* **by** *simp*
also have *val-process Mkt pf n w* = *val-process Mkt pf n wn* **unfolding** *wn-def* **using** ‹*val-process Mkt pf n* ∈ *borel-measurable* (*G n*)›
nat-filtration-info stoch-proc-subalg-nat-filt[*of geom-proc*] *geometric-process stock-filtration* **by** (*metis comp-apply geom-rand-walk-borel-adapted measurable-from-subalg*)
also have ... = *val-process Mkt pf n* (*spick wn n False*) **using** ‹*val-process Mkt pf n* ∈ *borel-measurable* (*G n*)›
comp-def nat-filtration-info
pseudo-proj-True-stake-image spickI stoch-proc-subalg-nat-filt[*of geom-proc*] *geometric-process stock-filtration*
by (*metis geom-rand-walk-borel-adapted measurable-from-subalg*)
finally show *?thesis* **.**
qed
ultimately have *cls-val-process Mkt pf* (*Suc n*) (*spick wn n False*) < *0* **using** *assms*
by (*simp add*:*pos-pf-neg-uvp*)
thus ∃ *y*. *cls-val-process Mkt pf* (*Suc n*) *y* < *0* **by** *auto*
next
case *False*
have *0* >*pf stk* (*Suc n*) (*spick wn n True*)
proof −
have *0* > *pf stk* (*Suc n*) *w* **using** ‹¬ *0* < *pf stk* (*Suc n*) *w*› ‹*pf stk* (*Suc n*) *w* ≠ *0*› **by** *simp*
also have *pf stk* (*Suc n*) *w* = *pf stk* (*Suc n*) *wn* **unfolding** *wn-def* **using** ‹*pf stk* (*Suc n*) ∈ *borel-measurable* (*G n*)›
nat-filtration-info stoch-proc-subalg-nat-filt[*of geom-proc*] *geometric-process stock-filtration* **by** (*metis comp-apply geom-rand-walk-borel-adapted measurable-from-subalg*)
also have ... = *pf stk* (*Suc n*) (*spick wn n True*) **using** ‹*pf stk* (*Suc n*) ∈ *borel-measurable* (*G n*)›
comp-def nat-filtration-info
pseudo-proj-True-stake-image spickI stoch-proc-subalg-nat-filt[*of geom-proc*] *geometric-process stock-filtration*
by (*metis geom-rand-walk-borel-adapted measurable-from-subalg*)
finally show *?thesis* **.**
qed
moreover have *0* ≥ *val-process Mkt pf n* (*spick wn n True*)
proof −
have *0* ≥ *val-process Mkt pf n w* **using** *assms* **by** *simp*
also have *val-process Mkt pf n w* = *val-process Mkt pf n wn* **unfolding** *wn-def* **using** ‹*val-process Mkt pf n* ∈ *borel-measurable* (*G n*)›
comp-def nat-filtration-info
pseudo-proj-True-stake-image spickI stoch-proc-subalg-nat-filt[*of geom-proc*] *geometric-process stock-filtration*
by (*metis geom-rand-walk-borel-adapted measurable-from-subalg*)
also have ... = *val-process Mkt pf n* (*spick wn n True*) **using** ‹*val-process Mkt pf n* ∈ *borel-measurable* (*G n*)›

comp-def nat-filtration-info
pseudo-proj-True-stake-image spickI stoch-proc-subalg-nat-filt[*of geom-proc*]
geometric-process stock-filtration
by (*metis geom-rand-walk-borel-adapted measurable-from-subalg*)
finally show *?thesis* **.**
qed
ultimately have *cls-val-process Mkt pf* (*Suc n*) (*spick wn n True*) $<$ *0* **using** *assms*
by (*simp add*:*neg-pf-neg-uvp*)
thus $\exists$ *y. cls-val-process Mkt pf* (*Suc n*) *y* $<$ *0* **by** *auto*
qed
next
case *False*
hence *pf risk-free-asset* (*Suc n*) *w* $\neq$ *0* **using** *assms* **by** *simp*
hence *cls-val-process Mkt pf* (*Suc n*) *w* $<$ *0* **using** *False assms* **by** (*auto simp add*:*zero-pf-neg-uvp*)
thus $\exists$ *y. cls-val-process Mkt pf* (*Suc n*) *y* $<$ *0* **by** *auto*
qed
qed

lemma (**in** *CRR-market*) *non-zero-components*:
assumes *val-process Mkt pf n y* $\neq$ *0*
and *stock-portfolio Mkt pf*
shows *pf stk* (*Suc n*) *y* $\neq$ *0* $\vee$ *pf risk-free-asset* (*Suc n*) *y* $\neq$ *0*
proof (*rule ccontr*)
assume $\neg$(*pf stk* (*Suc n*) *y* $\neq$ *0* $\vee$ *pf risk-free-asset* (*Suc n*) *y* $\neq$ *0*)
hence *pf stk* (*Suc n*) *y* = *0 pf risk-free-asset* (*Suc n*) *y* = *0* **by** *auto*
have *val-process Mkt pf n y* = *geom-proc n y* $*$ *pf stk* (*Suc n*) *y* +
disc-rfr-proc r n y $*$ *pf risk-free-asset* (*Suc n*) *y* **using** ‹*stock-portfolio Mkt pf*›
stock-pf-vp-expand[*of pf n*] **by** *simp*
also have ... = *0* **using** ‹*pf stk* (*Suc n*) *y* = *0*› ‹*pf risk-free-asset* (*Suc n*) *y* = *0*› **by** *simp*
finally have *val-process Mkt pf n y* = *0* **.**
moreover have *val-process Mkt pf n y* $\neq$ *0* **using** *assms* **by** *simp*
ultimately show *False* **by** *simp*
qed

lemma (**in** *CRR-market*) *neg-pf-Suc*:
assumes *stock-portfolio Mkt pf*
and *trading-strategy pf*
and *self-financing Mkt pf*
and *1*+*r* $<$ *u*
and *d* $<$ *1*+*r*
and *cls-val-process Mkt pf n w* $<$ *0*
shows *n* $\leq$ *m* $\Longrightarrow$ $\exists$ *y. cls-val-process Mkt pf m y* $<$ *0*
proof (*induct m*)
case *0*
assume *n* $\leq$ *0*

hence *n=0* **by** *simp*
thus ∃ *y*. *cls-val-process Mkt pf 0 y < 0* **using** *assms* **by** *auto*
next
case (*Suc m*)
assume *n ≤ Suc m*
thus ∃ *y*. *cls-val-process Mkt pf* (*Suc m*) *y < 0*
proof (*cases n < Suc m*)
case *False*
hence *n = Suc m* **using** ‹*n ≤ Suc m*› **by** *simp*
thus ∃ *y*. *cls-val-process Mkt pf* (*Suc m*) *y < 0* **using** *assms* **by** *auto*
next
case *True*
hence *n ≤ m* **by** *simp*
hence ∃ *y*. *cls-val-process Mkt pf m y < 0* **using** *Suc* **by** *simp*
from *this* **obtain** *y* **where** *cls-val-process Mkt pf m y < 0* **by** *auto*
hence *val-process Mkt pf m y < 0* **using** *assms* **by** (*simp add*:*self-financingE*)
hence *val-process Mkt pf m y ≤ 0* **by** *simp*
have *val-process Mkt pf m y ≠ 0* **using** ‹*val-process Mkt pf m y < 0*› **by** *simp*
hence *pf stk* (*Suc m*) *y ≠ 0 ∨ pf risk-free-asset* (*Suc m*) *y ≠ 0* **using** *assms*
non-zero-components **by** *simp*
thus ∃ *y*. *cls-val-process Mkt pf* (*Suc m*) *y < 0* **using** *neg-pf-exists*[*of pf m y*]
assms
‹*val-process Mkt pf m y ≤ 0*› **by** *simp*
qed
qed

lemma (**in** *CRR-market*) *viable-if*:
assumes *1+r < u*
and *d < 1+r*
shows *viable-market Mkt* **unfolding** *viable-market-def*
proof (*rule ccontr*)
assume ¬(∀ *p*. *stock-portfolio Mkt p* ⟶ ¬ *arbitrage-process Mkt p*)
hence ∃ *p*. *stock-portfolio Mkt p* ∧ *arbitrage-process Mkt p* **by** *simp*
from *this* **obtain** *pf* **where** *stock-portfolio Mkt pf* **and** *arbitrage-process Mkt pf*
by *auto*
have (∃ *m*. (*self-financing Mkt pf*) ∧ (*trading-strategy pf*) ∧
(∀ *w* ∈ *space M*. *cls-val-process Mkt pf 0 w = 0*) ∧
(*AE w in M*. *0 ≤ cls-val-process Mkt pf m w*) ∧
0 < 𝒫(*w in M*. *cls-val-process Mkt pf m w > 0*)) **using** ‹*arbitrage-process Mkt*
pf›
using *arbitrage-processE* **by** *simp*
from *this* **obtain** *m* **where** *self-financing Mkt pf* **and** (*trading-strategy pf*)
and (∀ *w* ∈ *space M*. *cls-val-process Mkt pf 0 w = 0*)
and (*AE w in M*. *0 ≤ cls-val-process Mkt pf m w*)
and *0 < 𝒫*(*w in M*. *cls-val-process Mkt pf m w > 0*) **by** *auto*
have {*w*∈ *space M*. *cls-val-process Mkt pf m w > 0*} ≠ {} **using**

‹0 < $\mathcal{P}$(w in M. cls-val-process Mkt pf m w > 0)› **by** *force*
hence ∃ w∈ space M. cls-val-process Mkt pf m w > 0 **by** *auto*
from *this* **obtain** y **where** y∈ space M **and** cls-val-process Mkt pf m y > 0 **by** *auto*
define A **where** A = {n::nat. n ≤ m ∧ cls-val-process Mkt pf n y > 0}
have *finite A* **unfolding** *A-def* **by** *auto*
have m ∈ A **using** ‹cls-val-process Mkt pf m y > 0› **unfolding** *A-def* **by** *simp*
hence A ≠ {} **by** *auto*
hence Min A ∈ A **using** ‹*finite A*› **by** *simp*
have Min A ≤ m **using** ‹*finite A*› ‹m∈ A› **by** *simp*
have 0 < Min A
proof −
have cls-val-process Mkt pf 0 y = 0 **using** ‹y∈ space M› ‹∀ w ∈ space M. cls-val-process Mkt pf 0 w = 0›
by *simp*
hence 0∉ A **unfolding** *A-def* **by** *simp*
moreover have 0 ≤ Min A **by** *simp*
ultimately show *?thesis* **using** ‹Min A ∈ A› *neq0-conv* **by** *fastforce*
qed
hence ∃ l. Suc l = Min A **using** *Suc-diff-1* **by** *blast*
from *this* **obtain** l **where** Suc l = Min A **by** *auto*
have cls-val-process Mkt pf l y ≤ 0
proof −
have l < Min A **using** ‹Suc l = Min A› **by** *simp*
hence l∉ A **using** ‹*finite A*› ‹A ≠ {}› **by** *auto*
moreover have l ≤ m **using** ‹Suc l = Min A› ‹m∈ A› ‹*finite A*› ‹A ≠ {}› ‹l < Min A› **by** *auto*
ultimately show *?thesis* **unfolding** *A-def* **by** *auto*
qed
hence val-process Mkt pf l y ≤ 0 **using** ‹*self-financing Mkt pf*› **by** (*simp add:self-financingE*)
moreover have pf stk (Suc l) y ≠ 0 ∨ pf risk-free-asset (Suc l) y ≠ 0
proof (*rule ccontr*)
assume ¬(pf stk (Suc l) y ≠ 0 ∨ pf risk-free-asset (Suc l) y ≠ 0)
hence pf stk (Suc l) y = 0 pf risk-free-asset (Suc l) y = 0 **by** *auto*
have cls-val-process Mkt pf (Min A) y = geom-proc (Suc l) y ∗ pf stk (Suc l) y +
disc-rfr-proc r (Suc l) y ∗ pf risk-free-asset (Suc l) y **using** ‹*stock-portfolio Mkt pf*›
‹Suc l = Min A› *stock-pf-uvp-expand*[*of pf l*] **by** *simp*
also have ... = 0 **using** ‹pf stk (Suc l) y = 0› ‹pf risk-free-asset (Suc l) y = 0› **by** *simp*
finally have cls-val-process Mkt pf (Min A) y = 0 **.**
moreover have cls-val-process Mkt pf (Min A) y > 0 **using** ‹Min A ∈ A› **unfolding** *A-def* **by** *simp*
ultimately show *False* **by** *simp*
qed
ultimately have ∃ z. cls-val-process Mkt pf (Suc l) z < 0 **using** *assms* ‹*stock-portfolio Mkt pf*›
‹*trading-strategy pf*› **by** (*simp add:neg-pf-exists*)

from *this* **obtain** *z* **where** *cls-val-process Mkt pf* (*Suc l*) *z* < *0* **by** *auto*
hence ∃ *x'*. *cls-val-process Mkt pf m x'* < *0* **using** *neg-pf-Suc assms* ‹*trading-strategy pf*›
‹*self-financing Mkt pf*› ‹*Suc l* = *Min A*› ‹*Min A* ≤ *m*› ‹*stock-portfolio Mkt pf*› **by** *simp*
from *this* **obtain** *x'* **where** *cls-val-process Mkt pf m x'* < *0* **by** *auto*
have *x'*∈ *space M* **using** *bernoulli-stream-space bernoulli* **by** *auto*
hence *x'*∈ {*w*∈ *space M*. ¬*0* ≤ *cls-val-process Mkt pf m w*} **using** ‹*cls-val-process Mkt pf m x'* < *0*› **by** *auto*
from ‹*AE w in M*. *0* ≤ *cls-val-process Mkt pf m w*› **obtain** *N* **where**
{*w*∈ *space M*. ¬*0* ≤ *cls-val-process Mkt pf m w*} ⊆ *N* **and** *emeasure M N* = *0* **and** *N*∈ *sets M* **using** *AE-E* **by** *auto*
have {*w*∈ *space M*. (*stake m w* = *stake m x'*)} ⊆ *N*
proof
fix *x*
assume *x* ∈ {*w* ∈ *space M*. *stake m w* = *stake m x'*}
hence *x*∈ *space M* **and** *stake m x* = *stake m x'* **by** *auto*
have *cls-val-process Mkt pf m* ∈ *borel-measurable* (*G m*)
proof −
have *borel-adapt-stoch-proc G* (*cls-val-process Mkt pf*) **using** ‹*trading-strategy pf*› ‹*stock-portfolio Mkt pf*›
by (*meson support-adapt-def readable stock-portfolio-def subsetCE cls-val-process-adapted*)
thus *?thesis* **unfolding** *adapt-stoch-proc-def* **by** *simp*
qed
hence *cls-val-process Mkt pf m x'* = *cls-val-process Mkt pf m x*
using ‹*stake m x* = *stake m x'*› *borel-measurable-stake*[*of cls-val-process Mkt pf m m x x'*]
pseudo-proj-True-stake-image spickI stoch-proc-subalg-nat-filt[*of geom-proc*] *geometric-process stock-filtration*
by (*metis geom-rand-walk-borel-adapted measurable-from-subalg*)
hence *cls-val-process Mkt pf m x* < *0* **using** ‹*cls-val-process Mkt pf m x'* < *0*› **by** *simp*
thus *x*∈ *N* **using** ‹{*w*∈ *space M*. ¬*0* ≤ *cls-val-process Mkt pf m w*} ⊆ *N*› ‹*x*∈ *space M*›
‹*cls-val-process Mkt pf* (*Suc l*) *z* < *0*› **by** *auto*
qed
moreover have *emeasure M* {*w*∈ *space M*. (*stake m w* = *stake m x'*)} ≠ *0* **using** *bernoulli-stream-pref-prob-neq-zero psgt pslt* **by** *simp*
ultimately show *False* **using** ‹*emeasure M N* = *0*› ‹*N* ∈ *events*› *emeasure-eq-0* **by** *blast*
qed

lemma (**in** *CRR-market*) *viable-only-if-d*:
assumes *viable-market Mkt*
shows *d* < *1*+*r*
proof (*rule ccontr*)
assume ¬ *d* < *1*+*r*
hence *1*+*r* ≤ *d* **by** *simp*

define *arb-pf* **where** *arb-pf* = (λ (*x*::'*a*) (*n*::*nat*) *w*. *0*::*real*)(*stk*:= (λ *n w*. *1*), *risk-free-asset* := (λ *n w*. − *geom-proc 0 w*))
have *support-set arb-pf* = {*stk*, *risk-free-asset*}
proof
show *support-set arb-pf* ⊆ {*stk*, *risk-free-asset*}
by (*simp add*: *arb-pf-def subset-iff support-set-def*)
have *stk*∈ *support-set arb-pf* **unfolding** *arb-pf-def support-set-def* **using** *two-stocks* **by** *simp*
moreover have *risk-free-asset*∈ *support-set arb-pf* **unfolding** *arb-pf-def support-set-def*
using *two-stocks geometric-process S0-positive* **by** *simp*
ultimately show {*stk*, *risk-free-asset*}⊆ *support-set arb-pf* **by** *simp*
qed
hence *stock-portfolio Mkt arb-pf* **using** *stocks*
by (*simp add*: *portfolio-def stock-portfolio-def*)
have *arbitrage-process Mkt arb-pf*
proof (*rule arbitrage-processI*, *intro exI conjI*)
show *self-financing Mkt arb-pf* **unfolding** *arb-pf-def* **using** ‹*support-set arb-pf* = {*stk*, *risk-free-asset*}›
by (*simp add*: *static-portfolio-self-financing*)
show *trading-strategy arb-pf* **unfolding** *trading-strategy-def*
proof (*intro conjI ballI*)
show *portfolio arb-pf* **unfolding** *portfolio-def* **using** ‹*support-set arb-pf* = {*stk*, *risk-free-asset*}› **by** *simp*
fix *asset*
assume *asset*∈ *support-set arb-pf*
show *borel-predict-stoch-proc G* (*arb-pf asset*)
proof (*cases asset* = *stk*)
case *True*
hence *arb-pf asset* = (λ *n w*. *1*) **unfolding** *arb-pf-def* **by** (*simp add*: *two-stocks*)
show *?thesis* **unfolding** *predict-stoch-proc-def*
proof
show *arb-pf asset 0* ∈ *borel-measurable* (*G 0*) **using** ‹*arb-pf asset* = (λ *n w*. *1*)› **by** *simp*
show ∀ *n*. *arb-pf asset* (*Suc n*) ∈ *borel-measurable* (*G n*)
proof
fix *n*
show *arb-pf asset* (*Suc n*) ∈ *borel-measurable* (*G n*) **using** ‹*arb-pf asset* = (λ *n w*. *1*)› **by** *simp*
qed
qed
next
case *False*
hence *arb-pf asset* = (λ *n w*. − *geom-proc 0 w*) **using** ‹*support-set arb-pf* = {*stk*, *risk-free-asset*}›
‹*asset* ∈ *support-set arb-pf*› **unfolding** *arb-pf-def* **by** *simp*
show *?thesis* **unfolding** *predict-stoch-proc-def*
proof

```
        show arb-pf asset 0 ∈ borel-measurable (G 0) using ‹arb-pf asset = (λ n
w. − geom-proc 0 w)›
          geometric-process by simp
        show ∀ n. arb-pf asset (Suc n) ∈ borel-measurable (G n)
        proof
          fix n
          show arb-pf asset (Suc n) ∈ borel-measurable (G n) using ‹arb-pf asset
= (λ n w. − geom-proc 0 w)›
            geometric-process by simp
        qed
      qed
    qed
    show ∀ w∈space M. cls-val-process Mkt arb-pf 0 w = 0
    proof
      fix w
      assume w∈ space M
      have cls-val-process Mkt arb-pf 0 w = geom-proc 0 w ∗ arb-pf stk (Suc 0) w
+
       disc-rfr-proc r 0 w ∗ arb-pf risk-free-asset (Suc 0) w using stock-pf-vp-expand
        ‹stock-portfolio Mkt arb-pf›
        using ‹self-financing Mkt arb-pf› self-financingE by fastforce
     also have ... = geom-proc 0 w ∗ (1) + disc-rfr-proc r 0 w ∗ arb-pf risk-free-asset
(Suc 0) w
        by (simp add: arb-pf-def two-stocks)
      also have ... = geom-proc 0 w + arb-pf risk-free-asset (Suc 0) w by simp
       also have ... = geom-proc 0 w  − geom-proc 0 w unfolding arb-pf-def by
simp
      also have ... = 0 by simp
      finally show cls-val-process Mkt arb-pf 0 w = 0 .
    qed
     have dev: ∀ w∈ space M. cls-val-process Mkt arb-pf (Suc 0) w = geom-proc
(Suc 0) w − (1+r) ∗ geom-proc 0 w
    proof (intro ballI)
      fix w
      assume w∈ space M
      have cls-val-process Mkt arb-pf (Suc 0) w =  geom-proc (Suc 0) w ∗ arb-pf
stk (Suc 0) w +
      disc-rfr-proc r (Suc 0) w ∗ arb-pf risk-free-asset (Suc 0) w using stock-pf-uvp-expand
        ‹stock-portfolio Mkt arb-pf› by simp
       also have ... = geom-proc (Suc 0) w + disc-rfr-proc r (Suc 0) w ∗ arb-pf
risk-free-asset (Suc 0) w
        by (simp add: arb-pf-def two-stocks)
       also have ... = geom-proc (Suc 0) w + (1+r) ∗ arb-pf risk-free-asset (Suc
0) w by simp
       also have ... = geom-proc (Suc 0) w − (1+r) ∗ geom-proc 0 w by (simp
add:arb-pf-def)
     finally show cls-val-process Mkt arb-pf (Suc 0) w = geom-proc (Suc 0) w −
(1+r) ∗ geom-proc 0 w .
```

qed
have *iniT*: ∀ *w*∈ *space M*. *snth w 0* ⟶ *cls-val-process Mkt arb-pf* (*Suc 0*) *w* > *0*
proof (*intro ballI impI*)
fix *w*
assume *w*∈ *space M* **and** *snth w 0*
have *cls-val-process Mkt arb-pf* (*Suc 0*) *w* = *geom-proc* (*Suc 0*) *w* − (*1*+*r*) ∗ *geom-proc 0 w*
using *dev* ‹*w*∈ *space M*› **by** *simp*
also have ... = *u* ∗ *geom-proc 0 w* − (*1*+*r*) ∗ *geom-proc 0 w* **using** ‹*snth w 0*› *geometric-process* **by** *simp*
also have ... = (*u* − (*1*+*r*)) ∗ *geom-proc 0 w* **by** (*simp add*: *left-diff-distrib*)
also have ... > *0* **using** *S0-positive* ‹*1* + *r* ≤ *d*› *down-lt-up geometric-process* **by** *auto*
finally show *cls-val-process Mkt arb-pf* (*Suc 0*) *w* > *0* **.**
qed
have *iniF*: ∀ *w*∈ *space M*. ¬*snth w 0* ⟶ *cls-val-process Mkt arb-pf* (*Suc 0*) *w* ≥ *0*
proof (*intro ballI impI*)
fix *w*
assume *w*∈ *space M* **and** ¬*snth w 0*
have *cls-val-process Mkt arb-pf* (*Suc 0*) *w* = *geom-proc* (*Suc 0*) *w* − (*1*+*r*) ∗ *geom-proc 0 w*
using *dev* ‹*w*∈ *space M*› **by** *simp*
also have ... = *d* ∗ *geom-proc 0 w* − (*1*+*r*) ∗ *geom-proc 0 w* **using** ‹¬*snth w 0*› *geometric-process* **by** *simp*
also have ... = (*d* − (*1*+*r*)) ∗ *geom-proc 0 w* **by** (*simp add*: *left-diff-distrib*)
also have ... ≥ *0* **using** *S0-positive* ‹*1* + *r* ≤ *d*› *down-lt-up geometric-process* **by** *auto*
finally show *cls-val-process Mkt arb-pf* (*Suc 0*) *w* ≥ *0* **.**
qed
have ∀ *w*∈ *space M*. *cls-val-process Mkt arb-pf* (*Suc 0*) *w* ≥ *0*
proof
fix *w*
assume *w*∈ *space M*
show *cls-val-process Mkt arb-pf* (*Suc 0*) *w* ≥ *0*
proof (*cases snth w 0*)
case *True*
thus *?thesis* **using** ‹*w*∈ *space M*› *iniT* **by** *auto*
next
case *False*
thus *?thesis* **using** ‹*w*∈ *space M*› *iniF* **by** *simp*
qed
qed
thus *AE w in M*. *0* ≤ *cls-val-process Mkt arb-pf* (*Suc 0*) *w* **by** *simp*
show *0* < *prob* {*w* ∈ *space M*. *0* < *cls-val-process Mkt arb-pf* (*Suc 0*) *w*}
proof −
have *cls-val-process Mkt arb-pf* (*Suc 0*) ∈ *borel-measurable M* **using** *borel-adapt-stoch-proc-borel-measurable*
cls-val-process-adapted ‹*trading-strategy arb-pf*› ‹*stock-portfolio Mkt arb-pf*›

using *support-adapt-def readable* **unfolding** *stock-portfolio-def* **by** *blast*
hence *set-event*:$\{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ (Suc\ 0)\ w\} \in sets\ M$
using *borel-measurable-iff-greater* **by** *blast*
have $\forall n.\ emeasure\ M\ \{w \in space\ M.\ w\ !!\ n\} = ennreal\ p$
using *bernoulli p-gt-0 p-lt-1 bernoulli-stream-component-probability*[*of M p*]
by *auto*
hence $emeasure\ M\ \{w \in space\ M.\ w\ !!\ 0\} = ennreal\ p$ **by** *blast*
moreover have $\{w \in space\ M.\ w\ !!\ 0\} \subseteq \{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ 1\ w\}$
proof
fix *w*
assume $w \in \{w \in space\ M.\ w\ !!\ 0\}$
hence $w \in space\ M$ **and** $w\ !!\ 0$ **by** *auto* **note** *wprops* = *this*
hence $0 < cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ 1\ w$ **using** *iniT* **by** *simp*
thus $w \in \{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ 1\ w\}$ **using** *wprops* **by** *simp*
qed
ultimately have $p \leq emeasure\ M\ \{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ 1\ w\}$
using *emeasure-mono set-event* **by** *fastforce*
hence $p \leq prob\ \{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ 1\ w\}$ **by** (*simp add*: *emeasure-eq-measure*)
thus $0 < prob\ \{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ (Suc\ 0)\ w\}$ **using** *psgt* **by** *simp*
qed
qed
thus *False* **using** *assms* **unfolding** *viable-market-def* **using** ‹*stock-portfolio Mkt arb-pf*› **by** *simp*
qed

lemma (**in** *CRR-market*) *viable-only-if-u*:
assumes *viable-market Mkt*
shows $1+r < u$
proof (*rule ccontr*)
assume $\neg\ 1+r < u$
hence $u \leq 1+r$ **by** *simp*
define *arb-pf* **where** *arb-pf* = $(\lambda\ (x::'a)\ (n::nat)\ w.\ 0::real)(stk := (\lambda\ n\ w.\ -1)$, *risk-free-asset* := $(\lambda\ n\ w.\ geom\text{-}proc\ 0\ w))$
have *support-set arb-pf* = {*stk*, *risk-free-asset*}
proof
show *support-set arb-pf* ⊆ {*stk*, *risk-free-asset*}
by (*simp add*: *arb-pf-def subset-iff support-set-def*)
have *stk* ∈ *support-set arb-pf* **unfolding** *arb-pf-def support-set-def* **using** *two-stocks* **by** *simp*
moreover have *risk-free-asset* ∈ *support-set arb-pf* **unfolding** *arb-pf-def support-set-def*
using *two-stocks geometric-process S0-positive* **by** *simp*

```
    ultimately show {stk, risk-free-asset}⊆ support-set arb-pf by simp
  qed
  hence stock-portfolio Mkt arb-pf using stocks
    by (simp add: portfolio-def stock-portfolio-def)
  have arbitrage-process Mkt arb-pf
  proof (rule arbitrage-processI, intro exI conjI)
    show self-financing Mkt arb-pf unfolding arb-pf-def using ‹support-set arb-pf
= {stk, risk-free-asset}›
      by (simp add: static-portfolio-self-financing)
    show trading-strategy arb-pf unfolding trading-strategy-def
    proof (intro conjI allI)
      show portfolio arb-pf unfolding portfolio-def using ‹support-set arb-pf =
{stk, risk-free-asset}› by simp
      fix asset
      assume asset∈ support-set arb-pf
      show borel-predict-stoch-proc G (arb-pf asset)
      proof (cases asset = stk)
        case True
        hence arb-pf asset = (λ n w. −1) unfolding arb-pf-def by (simp add:
two-stocks)
        show ?thesis unfolding predict-stoch-proc-def
        proof
          show arb-pf asset 0 ∈ borel-measurable (G 0) using ‹arb-pf asset = (λ n
w. −1)› by simp
          show ∀ n. arb-pf asset (Suc n) ∈ borel-measurable (G n)
          proof
            fix n
            show arb-pf asset (Suc n) ∈ borel-measurable (G n) using ‹arb-pf asset
= (λ n w. −1)› by simp
          qed
        qed
      next
        case False
        hence arb-pf asset = (λ n w. geom-proc 0 w) using ‹support-set arb-pf =
{stk, risk-free-asset}›
          ‹asset ∈ support-set arb-pf› unfolding arb-pf-def by simp
        show ?thesis unfolding predict-stoch-proc-def
        proof
          show arb-pf asset 0 ∈ borel-measurable (G 0) using ‹arb-pf asset = (λ n
w. geom-proc 0 w)›
            geometric-process by simp
          show ∀ n. arb-pf asset (Suc n) ∈ borel-measurable (G n)
          proof
            fix n
            show arb-pf asset (Suc n) ∈ borel-measurable (G n) using ‹arb-pf asset
= (λ n w. geom-proc 0 w)›
              geometric-process by simp
          qed
        qed
```

```
    qed
  qed
  show ∀ w∈space M. cls-val-process Mkt arb-pf 0 w = 0
  proof
    fix w
    assume w∈ space M
    have cls-val-process Mkt arb-pf 0 w = geom-proc 0 w * arb-pf stk (Suc 0) w
+
     disc-rfr-proc r 0 w * arb-pf risk-free-asset (Suc 0) w using stock-pf-vp-expand
      ‹stock-portfolio Mkt arb-pf›
      using ‹self-financing Mkt arb-pf› self-financingE by fastforce
       also have ... = geom-proc 0 w * (−1) + disc-rfr-proc r 0 w * arb-pf
risk-free-asset (Suc 0) w
      by (simp add: arb-pf-def two-stocks)
    also have ... = −geom-proc 0 w + arb-pf risk-free-asset (Suc 0) w by simp
     also have ... = geom-proc 0 w  − geom-proc 0 w unfolding arb-pf-def by
simp
    also have ... = 0 by simp
    finally show cls-val-process Mkt arb-pf 0 w = 0 .
  qed
  have dev: ∀ w∈ space M. cls-val-process Mkt arb-pf (Suc 0) w = −geom-proc
(Suc 0) w + (1+r) * geom-proc 0 w
  proof (intro ballI)
    fix w
    assume w∈ space M
     have cls-val-process Mkt arb-pf (Suc 0) w =  geom-proc (Suc 0) w * arb-pf
stk (Suc 0) w +
    disc-rfr-proc r (Suc 0) w * arb-pf risk-free-asset (Suc 0) w using stock-pf-uvp-expand
      ‹stock-portfolio Mkt arb-pf› by simp
     also have ... = −geom-proc (Suc 0) w + disc-rfr-proc r (Suc 0) w * arb-pf
risk-free-asset (Suc 0) w
      by (simp add: arb-pf-def two-stocks)
    also have ... = −geom-proc (Suc 0) w + (1+r) * arb-pf risk-free-asset (Suc
0) w by simp
     also have ... = −geom-proc (Suc 0) w + (1+r) * geom-proc 0 w by (simp
add:arb-pf-def)
    finally show cls-val-process Mkt arb-pf (Suc 0) w = −geom-proc (Suc 0) w
+ (1+r) * geom-proc 0 w .
  qed
  have iniT: ∀ w∈ space M. snth w 0 ⟶ cls-val-process Mkt arb-pf (Suc 0) w
≥ 0
  proof (intro ballI impI)
    fix w
    assume w∈ space M and snth w 0
   have cls-val-process Mkt arb-pf (Suc 0) w =  −geom-proc (Suc 0) w + (1+r)
* geom-proc 0 w
      using dev ‹w∈ space M› by simp
    also have ... = − u * geom-proc 0 w + (1+r) * geom-proc 0 w using ‹snth
w 0› geometric-process by simp
```

also have ... = $(-u + (1+r)) * geom\text{-}proc\ 0\ w$ **by** (*simp add*: *left-diff-distrib*)
also have ... ≥ 0 **using** *S0-positive* ‹$u \leq 1 + r$› *down-lt-up geometric-process* **by** *auto*
finally show *cls-val-process Mkt arb-pf* (*Suc 0*) $w \geq 0$ **.**
qed
have *iniF*: $\forall w \in space\ M. \neg snth\ w\ 0 \longrightarrow cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ (Suc\ 0)\ w > 0$
proof (*intro ballI impI*)
fix *w*
assume $w \in space\ M$ **and** $\neg snth\ w\ 0$
have *cls-val-process Mkt arb-pf* (*Suc 0*) $w = -geom\text{-}proc\ (Suc\ 0)\ w + (1+r) * geom\text{-}proc\ 0\ w$
using *dev* ‹$w \in space\ M$› **by** *simp*
also have ... $= -d * geom\text{-}proc\ 0\ w + (1+r) * geom\text{-}proc\ 0\ w$ **using** ‹$\neg snth\ w\ 0$› *geometric-process* **by** *simp*
also have ... $= (-d + (1+r)) * geom\text{-}proc\ 0\ w$ **by** (*simp add*: *left-diff-distrib*)
also have ... > 0 **using** *S0-positive* ‹$u <= 1 + r$› *down-lt-up geometric-process* **by** *auto*
finally show *cls-val-process Mkt arb-pf* (*Suc 0*) $w > 0$ **.**
qed
have $\forall w \in space\ M.\ cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ (Suc\ 0)\ w \geq 0$
proof
fix *w*
assume $w \in space\ M$
show *cls-val-process Mkt arb-pf* (*Suc 0*) $w \geq 0$
proof (*cases snth w 0*)
case *True*
thus *?thesis* **using** ‹$w \in space\ M$› *iniT* **by** *simp*
next
case *False*
thus *?thesis* **using** ‹$w \in space\ M$› *iniF* **by** *auto*
qed
qed
thus *AE w in M*. $0 \leq cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ (Suc\ 0)\ w$ **by** *simp*
show $0 < prob\ \{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ (Suc\ 0)\ w\}$
proof −
have *cls-val-process Mkt arb-pf* (*Suc 0*) ∈ *borel-measurable M* **using** *borel-adapt-stoch-proc-borel-measurable*
cls-val-process-adapted ‹*trading-strategy arb-pf*› ‹*stock-portfolio Mkt arb-pf*›
using *support-adapt-def readable* **unfolding** *stock-portfolio-def* **by** *blast*
hence *set-event*:$\{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ (Suc\ 0)\ w\} \in sets\ M$
using *borel-measurable-iff-greater* **by** *blast*
have $\forall n.\ emeasure\ M\ \{w \in space\ M.\ \neg w\ !!\ n\} = ennreal\ (1-p)$
using *bernoulli p-gt-0 p-lt-1 bernoulli-stream-component-probability-compl*[*of M p*]
by *auto*
hence *emeasure M* $\{w \in space\ M.\ \neg w\ !!\ 0\} = ennreal\ (1-p)$ **by** *blast*
moreover have $\{w \in space\ M.\ \neg w\ !!\ 0\} \subseteq \{w \in space\ M.\ 0 < cls\text{-}val\text{-}process\ Mkt\ arb\text{-}pf\ 1\ w\}$

proof
fix *w*
assume *w*∈ {*w* ∈ *space M*. ¬*w* !! *0*}
hence *w* ∈ *space M* **and** ¬*w* !! *0* **by** *auto* **note** *wprops* = *this*
hence *0* < *cls-val-process Mkt arb-pf 1 w* **using** *iniF* **by** *simp*
thus *w*∈ {*w* ∈ *space M*. *0* < *cls-val-process Mkt arb-pf 1 w*} **using** *wprops* **by** *simp*
qed
ultimately have *1*−*p* ≤ *emeasure M* {*w* ∈ *space M*. *0* < *cls-val-process Mkt arb-pf 1 w*}
using *emeasure-mono set-event* **by** *fastforce*
hence *1*−*p* ≤ *prob* {*w* ∈ *space M*. *0* < *cls-val-process Mkt arb-pf 1 w*} **by** (*simp add*: *emeasure-eq-measure*)
thus *0* < *prob* {*w* ∈ *space M*. *0* < *cls-val-process Mkt arb-pf* (*Suc 0*) *w*} **using** *pslt* **by** *simp*
qed
qed
thus *False* **using** *assms* **unfolding** *viable-market-def* **using** ‹*stock-portfolio Mkt arb-pf*› **by** *simp*
qed

lemma (**in** *CRR-market*) *viable-iff*:
shows *viable-market Mkt* ⟷ (*d* < *1*+*r* ∧ *1*+*r* < *u*) **using** *viable-if viable-only-if-d viable-only-if-u* **by** *auto*

8.2 Risk-neutral probability space for the geometric random walk

lemma (**in** *CRR-market*) *stock-price-borel-measurable*:
shows *borel-adapt-stoch-proc G* (*prices Mkt stk*)
proof −
have *borel-adapt-stoch-proc* (*stoch-proc-filt M geom-proc borel*) (*prices Mkt stk*)
by (*simp add*: *geom-rand-walk-borel-measurable stk-price stoch-proc-filt-adapt*)
thus *?thesis* **by** (*simp add*:*stock-filtration*)
qed

lemma (**in** *CRR-market*) *risk-free-asset-martingale*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *martingale N G* (*discounted-value r* (*prices Mkt risk-free-asset*))
proof −
have *filtration N G* **by** (*simp add*: *assms bernoulli-gen-filtration*)
moreover have ∀ *n*. *sigma-finite-subalgebra N* (*G n*) **by** (*simp add*: *assms bernoulli-sigma-finite*)
moreover have *finite-measure N* **using** *assms bernoulli-stream-def prob-space.prob-space-stream-space prob-space-def prob-space-measure-pmf* **by** *auto*
moreover have *discounted-value r* (*prices Mkt risk-free-asset*) = (λ *n w*. *1*)

using *discounted-rfr* **by** *auto*
ultimately show *?thesis* **using** *finite-measure.constant-martingale* **by** *simp*
qed

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-from-eq-sets*:
assumes N = *bernoulli-stream* q
and $0 < q$
and $q < 1$
shows *sets* (*infinite-coin-toss-space.nat-filtration* N n) = *sets* (*nat-filtration* n)
proof −
have *sigma-sets* (*space* (*bernoulli-stream* q)) {*pseudo-proj-True* n −‘ B ∩ *space* N |B. B ∈ *sets* (*bernoulli-stream* q)} = *sigma-sets* (*space* (*bernoulli-stream* p)) {*pseudo-proj-True* n −‘ B ∩ *space* M |B. B ∈ *sets* (*bernoulli-stream* p)}
proof −
have *sets* N = *events*
by (*metis assms*(1) *bernoulli-stream-def infinite-coin-toss-space-axioms infinite-coin-toss-space-def sets-measure-pmf sets-stream-space-cong*)
then show *?thesis*
using *assms*(1) *bernoulli-stream-space infinite-coin-toss-space-axioms infinite-coin-toss-space-def* **by** *auto*
qed
thus *?thesis* **using** *infinite-coin-toss-space.nat-filtration-sets*
using *assms*(1) *assms*(2) *assms*(3) *infinite-coin-toss-space-axioms infinite-coin-toss-space-def*
by *auto*
qed

lemma (**in** *CRR-market*) *geom-proc-integrable*:
assumes N = *bernoulli-stream* q
and $0 \leq q$
and $q \leq 1$
shows *integrable* N (*geom-proc* n)
proof (*rule infinite-coin-toss-space.nat-filtration-borel-measurable-integrable*)
show *infinite-coin-toss-space* q N **using** *assms* **by** *unfold-locales*
show *geom-proc* n ∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration* N n)
using *geometric-process*
prob-grw.geom-rand-walk-borel-adapted[*of* q N *geom-proc* u d *init*]
by (*metis* ‹*infinite-coin-toss-space* q N› *geom-rand-walk-pseudo-proj-True infinite-coin-toss-space.nat-filtration-borel-measurable-characterization prob-grw.geom-rand-walk-borel-measurable prob-grw-axioms prob-grw-def*)
qed

lemma (**in** *CRR-market*) *CRR-infinite-cts-filtration*:
shows *infinite-cts-filtration* p M *nat-filtration*
by (*unfold-locales*, *simp*)

lemma (**in** *CRR-market*) *proj-stoch-proc-geom-disc-fct*:
shows *disc-fct* (*proj-stoch-proc geom-proc n*) **unfolding** *disc-fct-def* **using** *CRR-infinite-cts-filtration*
by (*simp add*: *countable-finite geom-rand-walk-borel-adapted infinite-cts-filtration.proj-stoch-set-finite-range*)

lemma (**in** *CRR-market*) *proj-stoch-proc-geom-rng*:
assumes N = *bernoulli-stream q*
shows *proj-stoch-proc geom-proc n* $\in$ N $\rightarrow_M$ *stream-space borel*
proof −
have *random-variable* (*stream-space borel*) (*proj-stoch-proc geom-proc n*) **using** *CRR-infinite-cts-filtration*
using *geom-rand-walk-borel-adapted nat-discrete-filtration proj-stoch-measurable-if-adapted*
by *blast*
then show *?thesis*
using *assms*(*1*) *bernoulli bernoulli-stream-def* **by** *auto*
qed

lemma (**in** *CRR-market*) *proj-stoch-proc-geom-open-set*:
shows $\forall$ *r*$\in$*range* (*proj-stoch-proc geom-proc n*) $\cap$ *space* (*stream-space borel*).
$\exists$ *A*$\in$*sets* (*stream-space borel*). *range* (*proj-stoch-proc geom-proc n*) $\cap$ *A* = {*r*}
proof
fix *r*
assume *r*$\in$ *range* (*proj-stoch-proc geom-proc n*) $\cap$ *space* (*stream-space borel*)
show $\exists$ *A*$\in$*sets* (*stream-space borel*). *range* (*proj-stoch-proc geom-proc n*) $\cap$ *A* = {*r*}
proof
show *infinite-cts-filtration.stream-space-single* (*proj-stoch-proc geom-proc n*) *r* $\in$ *sets* (*stream-space borel*)
using *infinite-cts-filtration.stream-space-single-set* ‹*r* $\in$ *range* (*proj-stoch-proc geom-proc n*) $\cap$ *space* (*stream-space borel*)›
geom-rand-walk-borel-adapted CRR-infinite-cts-filtration **by** *blast*
show *range* (*proj-stoch-proc geom-proc n*) $\cap$ *infinite-cts-filtration.stream-space-single* (*proj-stoch-proc geom-proc n*) *r* = {*r*}
using *infinite-cts-filtration.stream-space-single-preimage* ‹*r* $\in$ *range* (*proj-stoch-proc geom-proc n*) $\cap$ *space* (*stream-space borel*)›
geom-rand-walk-borel-adapted CRR-infinite-cts-filtration **by** *blast*
qed
qed

lemma (**in** *CRR-market*) *bernoulli-AE-cond-exp*:
assumes N = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *integrable N X*
shows *AE w in N*. *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)) *X w* =
expl-cond-expect N (*proj-stoch-proc geom-proc n*) *X w*
proof (*rule finite-measure.charact-cond-exp'*)
have *infinite-cts-filtration p M nat-filtration*

by (*unfold-locales*, *simp*)
show *finite-measure N* **using** *assms*
by (*simp add*: *bernoulli-stream-def prob-space.finite-measure prob-space.prob-space-stream-space prob-space-measure-pmf*)
show *disc-fct* (*proj-stoch-proc geom-proc n*) **using** *proj-stoch-proc-geom-disc-fct* **by** *simp*
show *integrable N X* **using** *assms* **by** *simp*
show *proj-stoch-proc geom-proc n* ∈ *N* →$_M$ *stream-space borel* **using** *assms proj-stoch-proc-geom-rng* **by** *simp*
show ∀ *r*∈*range* (*proj-stoch-proc geom-proc n*) ∩ *space* (*stream-space borel*).
∃ *A*∈*sets* (*stream-space borel*). *range* (*proj-stoch-proc geom-proc n*) ∩ *A* = {*r*}
using *proj-stoch-proc-geom-open-set* **by** *simp*
qed

lemma (**in** *CRR-market*) *geom-proc-cond-exp*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *AE w in N*. *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)) (*geom-proc* (*Suc n*)) *w* =
expl-cond-expect N (*proj-stoch-proc geom-proc n*) (*geom-proc* (*Suc n*)) *w*
proof (*rule bernoulli-AE-cond-exp*)
show *integrable N* (*geom-proc* (*Suc n*)) **using** *assms geom-proc-integrable*[*of N q Suc n*] **by** *simp*
qed (*auto simp add*: *assms*)

lemma (**in** *CRR-market*) *expl-cond-eq-sets*:
assumes *N* = *bernoulli-stream q*
shows *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) *X* ∈
borel-measurable (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
proof (*rule expl-cond-exp-borel*)
show *proj-stoch-proc geom-proc n* ∈ *space N* → *space* (*stream-space borel*)
proof −
have *random-variable* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)
using *CRR-infinite-cts-filtration geom-rand-walk-borel-adapted proj-stoch-measurable-if-adapted nat-discrete-filtration* **by** *blast*
then show *?thesis*
by (*simp add*: *assms*(*1*) *bernoulli bernoulli-stream-space measurable-def*)
qed
show *disc-fct* (*proj-stoch-proc geom-proc n*) **unfolding** *disc-fct-def* **using** *CRR-infinite-cts-filtration*
by (*simp add*: *countable-finite geom-rand-walk-borel-adapted infinite-cts-filtration.proj-stoch-set-finite-range*)
show ∀ *r*∈*range* (*proj-stoch-proc geom-proc n*) ∩ *space* (*stream-space borel*).
∃ *A*∈*sets* (*stream-space borel*). *range* (*proj-stoch-proc geom-proc n*) ∩ *A* = {*r*}
proof
fix *r*
assume *r*∈*range* (*proj-stoch-proc geom-proc n*) ∩ *space* (*stream-space borel*)
show ∃ *A*∈*sets* (*stream-space borel*). *range* (*proj-stoch-proc geom-proc n*) ∩ *A*

$= \{r\}$
proof
show *infinite-cts-filtration.stream-space-single* (*proj-stoch-proc geom-proc n*) *r* ∈ *sets* (*stream-space borel*)
using *infinite-cts-filtration.stream-space-single-set* ‹*r* ∈ *range* (*proj-stoch-proc geom-proc n*) ∩ *space* (*stream-space borel*)›
geom-rand-walk-borel-adapted CRR-infinite-cts-filtration **by** *blast*
show *range* (*proj-stoch-proc geom-proc n*) ∩ *infinite-cts-filtration.stream-space-single* (*proj-stoch-proc geom-proc n*) *r* = {*r*}
using *infinite-cts-filtration.stream-space-single-preimage* ‹*r* ∈ *range* (*proj-stoch-proc geom-proc n*) ∩ *space* (*stream-space borel*)›
geom-rand-walk-borel-adapted CRR-infinite-cts-filtration **by** *blast*
qed
qed
qed

lemma (**in** *CRR-market*) *bernoulli-real-cond-exp-AE*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *integrable N X*
shows *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
X w = *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) *X w*
proof −
have *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
X w = *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) *X w*
proof (*rule infinite-coin-toss-space.nat-filtration-AE-eq*)
show *infinite-coin-toss-space q N* **using** *assms*
by (*simp add*: *infinite-coin-toss-space-def*)
show *AE w in N. real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)) *X w* =
expl-cond-expect N (*proj-stoch-proc geom-proc n*) *X w* **using** *assms bernoulli-AE-cond-exp*
by *simp*
show *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)) *X*
∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N n*)
proof −
have *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)) *X*
∈ *borel-measurable* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
by *simp*
moreover have *subalgebra* (*infinite-coin-toss-space.nat-filtration N n*) (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
using *stock-filtration infinite-coin-toss-space.stoch-proc-subalg-nat-filt*[*of q N geom-proc n*]

infinite-cts-filtration.stoch-proc-filt-gen[*of q N*]
by (*metis* ‹*infinite-coin-toss-space q N*› *infinite-cts-filtration-axioms.intro infinite-cts-filtration-def*
prob-grw.geom-rand-walk-borel-adapted prob-grw-axioms prob-grw-def)
ultimately show *?thesis* **using** *measurable-from-subalg* **by** *blast*
qed
show *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) *X* ∈
borel-measurable (*infinite-coin-toss-space.nat-filtration N n*)
proof −
have *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) *X* ∈
borel-measurable (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
by (*simp add*: *expl-cond-eq-sets assms*)
moreover have *subalgebra* (*infinite-coin-toss-space.nat-filtration N n*) (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
using *stock-filtration infinite-coin-toss-space.stoch-proc-subalg-nat-filt*[*of q N geom-proc n*]
infinite-cts-filtration.stoch-proc-filt-gen[*of q N*]
by (*metis* ‹*infinite-coin-toss-space q N*› *infinite-cts-filtration-axioms.intro infinite-cts-filtration-def*
prob-grw.geom-rand-walk-borel-adapted prob-grw-axioms prob-grw-def)
ultimately show *?thesis* **using** *measurable-from-subalg* **by** *blast*
qed
show *0* < *q* **and** *q* < *1* **using** *assms* **by** *auto*
qed
thus *?thesis* **by** *simp*
qed

lemma (**in** *CRR-market*) *geom-proc-real-cond-exp-AE*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
(*geom-proc* (*Suc n*)) *w* = *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) (*geom-proc* (*Suc n*)) *w*
proof (*rule bernoulli-real-cond-exp-AE*)
show *integrable N* (*geom-proc* (*Suc n*)) **using** *assms geom-proc-integrable*[*of N q Suc n*] **by** *simp*
qed (*auto simp add*: *assms*)

lemma (**in** *CRR-market*) *geom-proc-stoch-proc-filt*:
assumes *N*= *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *stoch-proc-filt N geom-proc borel n* = *fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)
proof (*rule infinite-cts-filtration.stoch-proc-filt-gen*)

show *infinite-cts-filtration q N* (*infinite-coin-toss-space.nat-filtration N*) **unfolding** *infinite-cts-filtration-def*
proof
show *infinite-coin-toss-space q N* **using** *assms*
by (*simp add*: *infinite-coin-toss-space-def*)
show *infinite-cts-filtration-axioms N* (*infinite-coin-toss-space.nat-filtration N*)
using *infinite-cts-filtration-axioms-def* **by** *blast*
qed
show *borel-adapt-stoch-proc* (*infinite-coin-toss-space.nat-filtration N*) *geom-proc*
using ‹*infinite-cts-filtration q N* (*infinite-coin-toss-space.nat-filtration N*)›
prob-grw.geom-rand-walk-borel-adapted prob-grw-axioms prob-grw-def
using *infinite-cts-filtration-def* **by** *auto*
qed

lemma (**in** *CRR-market*) *bernoulli-cond-exp*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *integrable N X*
shows *real-cond-exp N* (*stoch-proc-filt N geom-proc borel n*) *X w* = *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) *X w*
proof −
have *aeq*: *AE w in N*. *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)) *X w* =
expl-cond-expect N (*proj-stoch-proc geom-proc n*) *X w* **using** *assms*
bernoulli-AE-cond-exp **by** *simp*
have ∀ *w*. *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
X w = *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) *X w* **using** *assms bernoulli-real-cond-exp-AE* **by** *simp*
moreover have *stoch-proc-filt N geom-proc borel n* = *fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)
using *assms geom-proc-stoch-proc-filt* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *CRR-market*) *stock-cond-exp*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *real-cond-exp N* (*stoch-proc-filt N geom-proc borel n*) (*geom-proc* (*Suc n*)) *w* = *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) (*geom-proc* (*Suc n*)) *w*
proof (*rule bernoulli-cond-exp*)
show *integrable N* (*geom-proc* (*Suc n*)) **using** *assms geom-proc-integrable*[*of N q Suc n*] **by** *simp*
qed (*auto simp add*: *assms*)

lemma (**in** *prob-space*) *discount-factor-real-cond-exp*:
assumes *integrable M X*
and *subalgebra M G*
and $-1 < r$
shows *AE w in M. real-cond-exp M G* (λx. *discount-factor r n x* $*$ *X x*) *w* = *discount-factor r n w* $*$ (*real-cond-exp M G X*) *w*
proof (*rule sigma-finite-subalgebra.real-cond-exp-mult*)
show *sigma-finite-subalgebra M G* **using** *assms subalgebra-sigma-finite* **by** *simp*
show *discount-factor r n* $\in$ *borel-measurable G* **by** (*simp add*: *discount-factor-borel-measurable*)
show *random-variable borel X* **using** *assms* **by** *simp*
show *integrable M* (λx. *discount-factor r n x* $*$ *X x*) **using** *assms discounted-integrable*[*of M* λn. *X*]
unfolding *discounted-value-def* **by** *simp*
qed

lemma (**in** *prob-space*) *discounted-value-real-cond-exp*:
assumes *integrable M X*
and $-1 < r$
and *subalgebra M G*
shows *AE w in M. real-cond-exp M G* ((*discounted-value r* (λ *m. X*)) *n*) *w* =
discounted-value r (λm. (*real-cond-exp M G X*)) *n w* **using** *assms*
unfolding *discounted-value-def init-triv-filt-def filtration-def*
by (*simp add*: *assms discount-factor-real-cond-exp*)

lemma (**in** *CRR-market*)
assumes $q = (1 + r - d)/(u - d)$
and *viable-market Mkt*
shows *gt-param*: $0 < q$
and *lt-param*: $q < 1$
and *risk-neutral-param*: $u * q + d * (1 - q) = 1 + r$
proof −
show $0 < q$ **using** *down-lt-up viable-only-if-d assms* **by** *simp*
show $q < 1$ **using** *down-lt-up viable-only-if-u assms* **by** *simp*
show $u * q + d * (1 - q) = 1 + r$
proof −
have $1 - q = 1 - (1 + r - d) / (u - d)$ **using** *assms* **by** *simp*
also have ... = $(u - d)/(u - d) - (1 + r - d) / (u - d)$ **using** *down-lt-up* **by** *simp*
also have ... = $(u - d - (1 + r - d))/(u-d)$ **using** *diff-divide-distrib*[*of u − d 1 + r −d u −d*] **by** *simp*
also have ... = $(u - 1 - r)/(u-d)$ **by** *simp*
finally have $1 - q = (u - 1 - r)/(u - d)$.
hence $u * q + d * (1 - q) = u * (1 + r - d)/(u - d) + d * (u - 1 - r)/(u - d)$ **using** *assms* **by** *simp*
also have ... = $(u * (1 + r - d) + d * (u - 1 - r))/(u - d)$ **using** *add-divide-distrib*[*of u* $*$ $(1 + r - d)$] **by** *simp*

```
    also have ... = (u * (1 + r) - u * d + d * u - d * (1 + r))/(u - d)
      by (simp add: diff-diff-add right-diff-distrib')
    also have ... = (u * (1+r) - d * (1+r))/(u - d) by simp
    also have ... = ((u - d) * (1+r))/(u - d) by (simp add: left-diff-distrib)
    also have ... = 1 + r using down-lt-up by simp
    finally show ?thesis .
  qed
qed

lemma (in CRR-market) bernoulli-expl-cond-expect-adapt:
  assumes N = bernoulli-stream q
and 0 < q
and q < 1
  shows expl-cond-expect N (proj-stoch-proc geom-proc n) f∈ borel-measurable (G
n)
proof −
  have sets N = sets M using assms by (simp add: bernoulli bernoulli-stream-def
sets-stream-space-cong)
  have icf: infinite-cts-filtration p M nat-filtration by (unfold-locales, simp)
  have G n = stoch-proc-filt M geom-proc borel n using stock-filtration by simp
 also have ... = fct-gen-subalgebra M (stream-space borel) (proj-stoch-proc geom-proc
n)
  proof (rule infinite-cts-filtration.stoch-proc-filt-gen)
    show infinite-cts-filtration p M nat-filtration using icf .
   show borel-adapt-stoch-proc nat-filtration geom-proc using geom-rand-walk-borel-adapted
.
  qed
 also have ... = fct-gen-subalgebra N (stream-space borel) (proj-stoch-proc geom-proc
n)
    by (rule fct-gen-subalgebra-eq-sets, (simp add: ‹sets N = sets M›))
  finally have G n = fct-gen-subalgebra N (stream-space borel) (proj-stoch-proc
geom-proc n) .
  moreover have expl-cond-expect N (proj-stoch-proc geom-proc n) f ∈
      borel-measurable (fct-gen-subalgebra N (stream-space borel) (proj-stoch-proc
geom-proc n))
    by (simp add: expl-cond-eq-sets assms)
  ultimately show ?thesis by simp
qed



lemma (in CRR-market) real-cond-exp-discount-stock:
  assumes N = bernoulli-stream q
and 0 < q
and q < 1
shows AE w in N. real-cond-exp N (G n)
   (discounted-value r (prices Mkt stk) (Suc n)) w =
                discounted-value r (λm w. (q * u + (1 − q) * d) * prices Mkt stk n
w) (Suc n) w
```

proof −
have *qlt*: *0* < *q* **and** *qgt*: *q* < *1* **using** *assms* **by** *auto*
have *G n* = (*fct-gen-subalgebra M* (*stream-space borel*)
(*proj-stoch-proc geom-proc n*))
using *stock-filtration infinite-cts-filtration.stoch-proc-filt-gen*[*of p M nat-filtration geom-proc n*] *geometric-process*
geom-rand-walk-borel-adapted CRR-infinite-cts-filtration **by** *simp*
also have ... = (*fct-gen-subalgebra N* (*stream-space borel*)
(*proj-stoch-proc geom-proc n*))
proof (*rule fct-gen-subalgebra-eq-sets*)
show *events* = *sets N* **using** *assms qlt qgt*
by (*simp add*: *bernoulli bernoulli-stream-def sets-stream-space-cong*)
qed
finally have *G n* = (*fct-gen-subalgebra N* (*stream-space borel*)
(*proj-stoch-proc geom-proc n*)) **.**
hence *AE w in N*. *real-cond-exp N* (*G n*)
(*discounted-value r* (*prices Mkt stk*) (*Suc n*)) *w* = *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*)
(*proj-stoch-proc geom-proc n*))
(*discounted-value r* (*prices Mkt stk*) (*Suc n*)) *w* **by** *simp*
moreover have *AE w in N*. *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*)
(*proj-stoch-proc geom-proc n*))
(*discounted-value r* (*prices Mkt stk*) (*Suc n*)) *w* =
real-cond-exp N (*fct-gen-subalgebra N* (*stream-space borel*)
(*proj-stoch-proc geom-proc n*))
(*discounted-value r* (λ*m*. (*prices Mkt stk*) (*Suc n*)) (*Suc n*)) *w*
proof −
have ∀ *w*. (*discounted-value r* (*prices Mkt stk*) (*Suc n*)) *w* =
(*discounted-value r* (λ*m*. (*prices Mkt stk*) (*Suc n*)) (*Suc n*)) *w*
proof
fix *w*
show *discounted-value r* (*prices Mkt stk*) (*Suc n*) *w* = *discounted-value r* (λ*m*. *prices Mkt stk* (*Suc n*)) (*Suc n*) *w*
by (*simp add*: *discounted-value-def*)
qed
hence (*discounted-value r* (*prices Mkt stk*) (*Suc n*)) =
(*discounted-value r* (λ*m*. (*prices Mkt stk*) (*Suc n*)) (*Suc n*)) **by** *auto*
thus *?thesis* **by** *simp*
qed
moreover have *AE w in N*. (*real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*)
(*proj-stoch-proc geom-proc n*))
(*discounted-value r* (λ*m*. (*prices Mkt stk*) (*Suc n*)) (*Suc n*))) *w* =
discounted-value r (λ*m*. *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*)
(*proj-stoch-proc geom-proc n*))

```
                                        ((prices Mkt stk) (Suc n))) (Suc n) w
  proof (rule prob-space.discounted-value-real-cond-exp)
    show −1 < r using acceptable-rate by simp
   show integrable N (prices Mkt stk (Suc n)) using stk-price geom-proc-integrable
assms qlt qgt by simp
    show subalgebra N (fct-gen-subalgebra N (stream-space borel) (proj-stoch-proc
geom-proc n))
    proof (rule fct-gen-subalgebra-is-subalgebra)
      show proj-stoch-proc geom-proc n ∈ N →M stream-space borel
      proof −
        have proj-stoch-proc geom-proc n ∈ measurable M (stream-space borel)
        proof (rule proj-stoch-measurable-if-adapted)
          show borel-adapt-stoch-proc nat-filtration geom-proc using
            geometric-process
            geom-rand-walk-borel-adapted by simp
          show filtration M nat-filtration using CRR-infinite-cts-filtration
            by (simp add: nat-discrete-filtration)
        qed
        thus ?thesis using assms bernoulli-stream-equiv filt-equiv-measurable qlt qgt
psgt pslt by blast
      qed
    qed
    show prob-space N using assms
     by (simp add: bernoulli bernoulli-stream-def prob-space.prob-space-stream-space
prob-space-measure-pmf)
  qed
  moreover have AE w in N. discounted-value r (λm. real-cond-exp N (fct-gen-subalgebra
N (stream-space borel)
                                                (proj-stoch-proc geom-proc n))
                                               ((prices Mkt stk) (Suc n))) (Suc n) w =
                  discounted-value r (λm w. (q ∗ u + (1 − q) ∗ d) ∗ prices Mkt stk
n w) (Suc n) w
  proof (rule discounted-AE-cong)
   have AEeq N (real-cond-exp N (fct-gen-subalgebra N (stream-space borel)
                                (proj-stoch-proc geom-proc n))
                                ((prices Mkt stk) (Suc n)))
             (λw. q ∗ (prices Mkt stk) (Suc n) (pseudo-proj-True n w) +
              (1 − q) ∗ (prices Mkt stk) (Suc n) (pseudo-proj-False n w))
     proof (rule infinite-cts-filtration.f-borel-Suc-real-cond-exp)
      show icf: infinite-cts-filtration q N (infinite-coin-toss-space.nat-filtration N)
unfolding infinite-cts-filtration-def
      proof
        show infinite-coin-toss-space q N using assms qlt qgt
          by (simp add: infinite-coin-toss-space-def)
         show infinite-cts-filtration-axioms N (infinite-coin-toss-space.nat-filtration
N)
          using infinite-cts-filtration-axioms-def by blast
      qed
      have badapt: borel-adapt-stoch-proc (infinite-coin-toss-space.nat-filtration N)
```

(*prices Mkt stk*)
using *stk-price prob-grw.geom-rand-walk-borel-adapted*[*of q N geom-proc*]
unfolding *adapt-stoch-proc-def*
by (*metis* (*full-types*) *borel-measurable-integrable geom-proc-integrable geom-rand-walk-pseudo-proj-True icf*
infinite-coin-toss-space.nat-filtration-borel-measurable-characterization infinite-coin-toss-space-def
infinite-cts-filtration-def)
show *prices Mkt stk* (*Suc n*) ∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N* (*Suc n*))
using *badapt* **unfolding** *adapt-stoch-proc-def* **by** *simp*
show *proj-stoch-proc geom-proc n* ∈ *infinite-coin-toss-space.nat-filtration N n* →$_M$ *stream-space borel*
proof (*rule proj-stoch-adapted-if-adapted*)
show *filtration N* (*infinite-coin-toss-space.nat-filtration N*) **using** *icf*
using *infinite-coin-toss-space.nat-discrete-filtration infinite-cts-filtration-def* **by** *blast*
show *borel-adapt-stoch-proc* (*infinite-coin-toss-space.nat-filtration N*) *geom-proc* **using** *badapt stk-price* **by** *simp*
qed
show *set-discriminating n* (*proj-stoch-proc geom-proc n*) (*stream-space borel*) **unfolding** *set-discriminating-def*
proof (*intro allI impI*)
fix *w*
assume *proj-stoch-proc geom-proc n w* ≠ *proj-stoch-proc geom-proc n* (*pseudo-proj-True n w*)
hence *False* **using** *CRR-infinite-cts-filtration*
by (*metis* ‹*proj-stoch-proc geom-proc n w* ≠ *proj-stoch-proc geom-proc n* (*pseudo-proj-True n w*)›
geom-rand-walk-borel-adapted infinite-cts-filtration.proj-stoch-proj-invariant)
thus ∃ *A*∈*sets* (*stream-space borel*).
(*proj-stoch-proc geom-proc n w* ∈ *A*) = (*proj-stoch-proc geom-proc n* (*pseudo-proj-True n w*) ∉ *A*) **by** *simp*
qed
show ∀ *w*. *proj-stoch-proc geom-proc n* −‘ {*proj-stoch-proc geom-proc n w*} ∈ *sets* (*infinite-coin-toss-space.nat-filtration N n*)
proof
fix *w*
show *proj-stoch-proc geom-proc n* −‘ {*proj-stoch-proc geom-proc n w*} ∈ *sets* (*infinite-coin-toss-space.nat-filtration N n*)
using ‹*proj-stoch-proc geom-proc n* ∈ *infinite-coin-toss-space.nat-filtration N n* →$_M$ *stream-space borel*›
using *assms geom-rand-walk-borel-adapted nat-filtration-from-eq-sets qlt qgt*
infinite-cts-filtration.proj-stoch-singleton-set CRR-infinite-cts-filtration **by** *blast*
qed
show ∀ *r*∈*range* (*proj-stoch-proc geom-proc n*) ∩ *space* (*stream-space borel*).
∃ *A*∈*sets* (*stream-space borel*). *range* (*proj-stoch-proc geom-proc n*) ∩ *A* =

$\{r\}$
proof
fix r
assume *asm*: $r \in$ *range* (*proj-stoch-proc geom-proc n*) $\cap$ *space* (*stream-space borel*)
define A **where** $A =$ *infinite-cts-filtration.stream-space-single* (*proj-stoch-proc geom-proc n*) r
have $A \in$ *sets* (*stream-space borel*) **using** *infinite-cts-filtration.stream-space-single-set*
unfolding *A-def* **using** *badapt icf stk-price asm* **by** *blast*
moreover have *range* (*proj-stoch-proc geom-proc n*) $\cap$ $A = \{r\}$
unfolding *A-def* **using** *badapt icf stk-price infinite-cts-filtration.stream-space-single-preimage asm* **by** *blast*
ultimately show $\exists A \in$*sets* (*stream-space borel*). *range* (*proj-stoch-proc geom-proc n*) $\cap$ $A = \{r\}$ **by** *auto*
qed
show $\forall y\ z.$ *proj-stoch-proc geom-proc n y* = *proj-stoch-proc geom-proc n z* $\wedge$ *y* !! *n* = *z* !! *n* $\longrightarrow$
prices Mkt stk (*Suc n*) *y* = *prices Mkt stk* (*Suc n*) *z*
proof (*intro allI impI*)
fix *y z*
assume *proj-stoch-proc geom-proc n y* = *proj-stoch-proc geom-proc n z* $\wedge$ *y* !! *n* = *z* !! *n*
hence *geom-proc n y* = *geom-proc n z* **using** *proj-stoch-proc-component*(*2*)[*of n n*]
proof −
show *?thesis*
by (*metis* ‹$\bigwedge w\ f.\ n \leq n \Longrightarrow$ *proj-stoch-proc f n w* !! *n* = *f n w*› ‹*proj-stoch-proc geom-proc n y* = *proj-stoch-proc geom-proc n z* $\wedge$ *y* !! *n* = *z* !! *n*› *order-refl*)
qed
hence *geom-proc* (*Suc n*) *y* = *geom-proc* (*Suc n*) *z* **using** *geometric-process*
by (*simp add*: ‹*proj-stoch-proc geom-proc n y* = *proj-stoch-proc geom-proc n z* $\wedge$ *y* !! *n* = *z* !! *n*›)
thus *prices Mkt stk* (*Suc n*) *y* = *prices Mkt stk* (*Suc n*) *z* **using** *stk-price* **by** *simp*
qed
show *0* < *q* **and** *q* < *1* **using** *assms* **by** *auto*
qed
moreover have $\forall w.$ *q* ∗ *prices Mkt stk* (*Suc n*) (*pseudo-proj-True n w*) + (*1* − *q*) ∗ *prices Mkt stk* (*Suc n*) (*pseudo-proj-False n w*) =
(*q* ∗ *u* + (*1* − *q*) ∗ *d*) ∗ *prices Mkt stk n w*
proof
fix *w*
have *q* ∗ *prices Mkt stk* (*Suc n*) (*pseudo-proj-True n w*) + (*1* − *q*) ∗ *prices Mkt stk* (*Suc n*) (*pseudo-proj-False n w*) =
q ∗ *geom-proc* (*Suc n*) (*pseudo-proj-True n w*) + (*1*−*q*) ∗ *geom-proc* (*Suc n*) (*pseudo-proj-False n w*)
by (*simp add*:*stk-price*)
also have ... = *q* ∗ *u* ∗ *geom-proc n* (*pseudo-proj-True n w*) + (*1*−*q*) ∗

geom-proc (*Suc n*) (*pseudo-proj-False n w*)
using *geometric-process* **unfolding** *pseudo-proj-True-def* **by** *simp*
also have ... = *q* * *u* * *geom-proc n w* + (*1*−*q*) * *geom-proc* (*Suc n*) (*pseudo-proj-False n w*)
by (*metis geom-rand-walk-pseudo-proj-True o-apply*)
also have ... = *q* * *u* * *geom-proc n w* + (*1*−*q*) * *d* * *geom-proc n* (*pseudo-proj-False n w*)
using *geometric-process* **unfolding** *pseudo-proj-False-def* **by** *simp*
also have ... = *q* * *u* * *geom-proc n w* + (*1*−*q*) * *d* * *geom-proc n w*
by (*metis geom-rand-walk-pseudo-proj-False o-apply*)
also have ... = (*q* * *u* + (*1* − *q*) * *d*) * *geom-proc n w* **by** (*simp add*: *distrib-right*)
finally show *q* * *prices Mkt stk* (*Suc n*) (*pseudo-proj-True n w*) + (*1* − *q*) * *prices Mkt stk* (*Suc n*) (*pseudo-proj-False n w*) =
(*q* * *u* + (*1* − *q*) * *d*) * *prices Mkt stk n w* **using** *stk-price* **by** *simp*
qed
ultimately show *AEeq N* (*real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*)
(*proj-stoch-proc geom-proc n*))
((*prices Mkt stk*) (*Suc n*)))
(λ*w*. (*q* * *u* + (*1* − *q*) * *d*) * *prices Mkt stk n w*) **by** *simp*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma (**in** *CRR-market*) *risky-asset-martingale-only-if*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *martingale N G* (*discounted-value r* (*prices Mkt stk*))
shows *q* = (*1* + *r* − *d*) / (*u* − *d*)
proof −
have *AE w in N*. *real-cond-exp N* (*G 0*)
(*discounted-value r* (*prices Mkt stk*) (*Suc 0*)) *w* = *discounted-value r* (*prices Mkt stk*) *0 w* **using** *assms*
unfolding *martingale-def* **by** *simp*
hence *AE w in N*. *real-cond-exp N* (*G 0*)
(*discounted-value r* (*prices Mkt stk*) (*Suc 0*)) *w* = *prices Mkt stk 0 w* **by** (*simp add*: *discounted-init*)
moreover have *AE w in N*. *real-cond-exp N* (*G 0*) (*discounted-value r* (*prices Mkt stk*) (*Suc 0*)) *w* =
discounted-value r (λ*m w*. (*q* * *u* + (*1* − *q*) * *d*) * *prices Mkt stk 0 w*) (*Suc 0*) *w*
using *assms real-cond-exp-discount-stock* **by** *simp*
ultimately have *AE w in N*. *discounted-value r* (λ*m w*. (*q* * *u* + (*1* − *q*) * *d*) * *prices Mkt stk 0 w*) (*Suc 0*) *w* =
prices Mkt stk 0 w **by** *auto*

hence *AE w in N. discounted-value r* (*λm w. (q * u + (1 − q) * d) * init*) (*Suc 0*) *w* =
(*λw. init*) *w* **using** *stk-price geometric-process* **by** *simp*
hence *AE w in N. discount-factor r* (*Suc 0*) *w* * (*q * u + (1 − q) * d*) * *init* =
(*λw. init*) *w* **unfolding** *discounted-value-def* **by** *simp*
hence *AE w in N.* (*1+r*) * *discount-factor r* (*Suc 0*) *w* * (*q * u + (1 − q) * d*) * *init* =
(*1+r*) * (*λw. init*) *w* **by** *auto*
hence *prev*: *AE w in N. discount-factor r 0 w* * (*q * u + (1 − q) * d*) * *init* =
(*1+r*) * (*λw. init*) *w* **using** *discount-factor-times-rfr*[*of r 0*] *acceptable-rate*
proof −
have ∀ *s.* (*1 + r*) * *discount-factor r* (*Suc 0*) (*s*::*bool stream*) = *discount-factor r 0 s*
by (*metis* (*no-types*) ‹⋀*w. − 1 < r* ⟹ (*1 + r*) * *discount-factor r* (*Suc 0*) *w* = *discount-factor r 0 w*› *acceptable-rate*)
then show *?thesis*
using ‹*AEeq N* (*λw.* (*1 + r*) * *discount-factor r* (*Suc 0*) *w* * (*q * u + (1 − q) * d*) * *init*) (*λw.* (*1 + r*) * *init*)› **by** *presburger*
qed
hence ∀ *w.* (*λw. discount-factor r 0 w* * (*q * u + (1 − q) * d*) * *init*) *w* =
(*λw.* (*1+r*) * *init*) *w*
proof −
have (*λw. discount-factor r 0 w* * (*q * u + (1 − q) * d*) * *init*)
∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N 0*)
proof (*rule borel-measurable-times*)+
show (*λx. init*) ∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N 0*) **by** *simp*
show (*λx. q * u + (1 − q) * d*) ∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N 0*) **by** *simp*
show *discount-factor r 0* ∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N 0*)
using *discount-factor-nonrandom*[*of r 0 infinite-coin-toss-space.nat-filtration N 0*] **by** *simp*
qed
moreover have (*λw.* (*1 + r*) * *init*) ∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N 0*) **by** *simp*
moreover have *infinite-coin-toss-space q N* **using** *assms* **by** (*simp add*: *infinite-coin-toss-space-def*)
ultimately show *?thesis*
using *prev infinite-coin-toss-space.nat-filtration-AE-eq*[*of q N*
(*λw. discount-factor r 0 w* * (*q * u + (1 − q) * d*) * *init*) (*λw.* (*1 + r*) * *init*) *0*] *assms*
by (*simp add*: *discount-factor-init*)
qed
hence (*q * u + (1 − q) * d*) * *init* = (*1+r*) * *init* **by** (*simp add*: *discount-factor-init*)
hence *q * u + (1 − q) * d* = *1+r* **using** *S0-positive* **by** *simp*
hence *q * u + d − q * d* = *1+r* **by** (*simp add*: *left-diff-distrib*)
hence *q* * (*u − d*) = *1 + r − d*

by (*metis* (*no-types*, *opaque-lifting*) *add.commute add.left-commute add-diff-cancel-left' add-uminus-conv-diff left-diff-distrib mult.commute*)
thus $q = (1 + r - d) / (u - d)$ **using** *down-lt-up*
by (*metis add.commute add.right-neutral diff-add-cancel nonzero-eq-divide-eq order-less-irrefl*)
qed

locale *CRR-market-viable* = *CRR-market* +
assumes *CRR-viable*: *viable-market Mkt*

lemma (**in** *CRR-market-viable*) *real-cond-exp-discount-stock-q-const*:
assumes N = *bernoulli-stream* q
and $q = (1+r-d) / (u-d)$
shows *AE w in N. real-cond-exp N* (*G n*)
(*discounted-value r* (*prices Mkt stk*) (*Suc n*)) *w* =
discounted-value r (*prices Mkt stk*) *n w*
proof −
have *qlt*: $0 < q$ **and** *qgt*: $q < 1$ **using** *assms gt-param lt-param CRR-viable* **by** *auto*
have *AE w in N. real-cond-exp N* (*G n*) (*discounted-value r* (*prices Mkt stk*) (*Suc n*)) *w* =
discounted-value r (λ*m w*. (*q* ∗ *u* + (1 − *q*) ∗ *d*) ∗ *prices Mkt stk n w*) (*Suc n*) *w*
using *assms real-cond-exp-discount-stock*[*of N q*] *qlt qgt* **by** *simp*
moreover have ∀ *w*. (*q* ∗ *u* + (1 − *q*) ∗ *d*) ∗ *prices Mkt stk n w* =
(1+*r*) ∗ *prices Mkt stk n w* **using** *risk-neutral-param assms CRR-viable*
by (*simp add*: *mult.commute*)
ultimately have *AE w in N. real-cond-exp N* (*G n*) (*discounted-value r* (*prices Mkt stk*) (*Suc n*)) *w* =
discounted-value r (λ*m w*. (1+*r*) ∗ *prices Mkt stk n w*) (*Suc n*) *w*
by *simp*
moreover have ∀ *w*∈ *space N. discounted-value r* (λ*m w*. (1+*r*) ∗ *prices Mkt stk n w*) (*Suc n*) *w* =
discounted-value r (λ*m w. prices Mkt stk n w*) *n w*
using *acceptable-rate* **by** (*simp add*:*discounted-mult-times-rfr*)
moreover hence ∀ *w*∈ *space N. discounted-value r* (λ*m w*. (1+*r*) ∗ *prices Mkt stk n w*) (*Suc n*) *w* =
discounted-value r (*prices Mkt stk*) *n w*
using *acceptable-rate* **by** (*simp add*:*discounted-value-def*)
ultimately show *AE w in N. real-cond-exp N* (*G n*) (*discounted-value r* (*prices Mkt stk*) (*Suc n*)) *w* =
discounted-value r (*prices Mkt stk*) *n w* **by** *simp*
qed

lemma (**in** *CRR-market-viable*) *risky-asset-martingale-if*:

assumes $N = bernoulli\text{-}stream\ q$
and $q = (1 + r - d) / (u - d)$
shows $martingale\ N\ G\ (discounted\text{-}value\ r\ (prices\ Mkt\ stk))$
proof (*rule disc-martingale-charact*)
have *qlt*: $0 < q$ **and** *qgt*: $q < 1$ **using** *assms gt-param lt-param CRR-viable* **by** *auto*
show $\forall n.\ integrable\ N\ (discounted\text{-}value\ r\ (prices\ Mkt\ stk)\ n)$
proof
fix n
show $integrable\ N\ (discounted\text{-}value\ r\ (prices\ Mkt\ stk)\ n)$
proof (*rule discounted-integrable*)
show $space\ N = space\ M$ **using** *assms* **by** (*simp add*: *bernoulli bernoulli-stream-space*)
show $integrable\ N\ (prices\ Mkt\ stk\ n)$
proof (*rule infinite-coin-toss-space.nat-filtration-borel-measurable-integrable*)
show *infinite-coin-toss-space q N* **using** *assms qlt qgt*
by (*simp add*: *infinite-coin-toss-space-def*)
show $prices\ Mkt\ stk\ n \in borel\text{-}measurable\ (infinite\text{-}coin\text{-}toss\text{-}space.nat\text{-}filtration\ N\ n)$
using *geom-rand-walk-borel-adapted stk-price nat-filtration-from-eq-sets*
unfolding *adapt-stoch-proc-def*
by (*metis* ‹*infinite-coin-toss-space q N*› *borel-measurable-integrable geom-proc-integrable geom-rand-walk-pseudo-proj-True infinite-coin-toss-space.nat-filtration-borel-measurable-characterization infinite-coin-toss-space-def*)
qed
show $-1 < r$ **using** *acceptable-rate* **by** *simp*
qed
qed
show *filtration N G* **using** *qlt qgt* **by** (*simp add*: *bernoulli-gen-filtration assms*)
show $\forall n.\ sigma\text{-}finite\text{-}subalgebra\ N\ (G\ n)$ **using** *qlt qgt* **by** (*simp add*: *assms bernoulli-sigma-finite*)
show $\forall m.\ discounted\text{-}value\ r\ (prices\ Mkt\ stk)\ m \in borel\text{-}measurable\ (G\ m)$
proof
fix m
have $discounted\text{-}value\ r\ (\lambda ma.\ prices\ Mkt\ stk\ m)\ m \in borel\text{-}measurable\ (G\ m)$
proof (*rule discounted-measurable*)
show $prices\ Mkt\ stk\ m \in borel\text{-}measurable\ (G\ m)$ **using** *stock-price-borel-measurable*
unfolding *adapt-stoch-proc-def* **by** *simp*
qed
thus $discounted\text{-}value\ r\ (prices\ Mkt\ stk)\ m \in borel\text{-}measurable\ (G\ m)$
by (*metis* (*mono-tags*, *lifting*) *discounted-value-def measurable-cong*)
qed
show $\forall n.\ AE\ w\ in\ N.\ real\text{-}cond\text{-}exp\ N\ (G\ n)\ (discounted\text{-}value\ r\ (prices\ Mkt\ stk)\ (Suc\ n))\ w = discounted\text{-}value\ r\ (prices\ Mkt\ stk)\ n\ w$
proof
fix n
show $AE\ w\ in\ N.\ real\text{-}cond\text{-}exp\ N\ (G\ n)\ (discounted\text{-}value\ r\ (prices\ Mkt\ stk)\ (Suc\ n))\ w = discounted\text{-}value\ r\ (prices$

Mkt stk) *n w*
using *assms real-cond-exp-discount-stock-q-const* **by** *simp*
qed
qed

lemma (**in** *CRR-market-viable*) *risk-neutral-iff'*:
assumes N = *bernoulli-stream q*
and $0 \leq q$
and $q \leq 1$
and *filt-equiv nat-filtration M N*
shows *rfr-disc-equity-market.risk-neutral-prob G Mkt r N* $\longleftrightarrow$ $q = (1 + r - d) / (u - d)$
proof
have $0 < q$ **and** $q < 1$ **using** *assms filt-equiv-sgt filt-equiv-slt psgt pslt* **by** *auto*
note *qprops* = *this*
have *dem*: *rfr-disc-equity-market M G Mkt r risk-free-asset* **by** *unfold-locales*
{
assume *rfr-disc-equity-market.risk-neutral-prob G Mkt r N*
hence (*prob-space N*) $\wedge$ ($\forall$ *asset* $\in$ *stocks Mkt*. *martingale N G* (*discounted-value r* (*prices Mkt asset*)))
using *rfr-disc-equity-market.risk-neutral-prob-def*[*of M G Mkt*] *dem* **by** *simp*
hence *martingale N G* (*discounted-value r* (*prices Mkt stk*)) **using** *stocks* **by** *simp*
thus $q = (1 + r - d) / (u - d)$ **using** *assms risky-asset-martingale-only-if*[*of N q*] *qprops* **by** *simp*
}
{
assume $q = (1 + r - d) / (u - d)$
hence *martingale N G* (*discounted-value r* (*prices Mkt stk*)) **using** *risky-asset-martingale-if*[*of N q*] *assms* **by** *simp*
moreover have *martingale N G* (*discounted-value r* (*prices Mkt risk-free-asset*)) **using** *risk-free-asset-martingale*
assms qprops **by** *simp*
ultimately show *rfr-disc-equity-market.risk-neutral-prob G Mkt r N* **using** *stocks*
using *assms*(*1*) *bernoulli-stream-def dem prob-space.prob-space-stream-space prob-space-measure-pmf*
rfr-disc-equity-market.risk-neutral-prob-def **by** *fastforce*
}
qed

lemma (**in** *CRR-market-viable*) *risk-neutral-iff*:
assumes N = *bernoulli-stream q*
and $0 < q$
and $q < 1$
shows *rfr-disc-equity-market.risk-neutral-prob G Mkt r N* $\longleftrightarrow$ $q = (1 + r - d) / (u - d)$
using *bernoulli-stream-equiv assms risk-neutral-iff' psgt pslt* **by** *auto*

8.3 Existence of a replicating portfolio

fun (**in** *CRR-market*) *rn-rev-price* **where**
rn-rev-price N der matur 0 w = *der w* |
rn-rev-price N der matur (*Suc n*) *w* = *discount-factor r* (*Suc 0*) *w* ∗
expl-cond-expect N (*proj-stoch-proc geom-proc* (*matur* − *Suc n*)) (*rn-rev-price N der matur n*) *w*

lemma (**in** *CRR-market*) *stock-filtration-eq*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *G n* = *stoch-proc-filt N geom-proc borel n*
proof −
have *G n*= *stoch-proc-filt M geom-proc borel n* **using** *stock-filtration* **by** *simp*
also have ... = *stoch-proc-filt N geom-proc borel n*
proof (*rule stoch-proc-filt-filt-equiv*)
show *filt-equiv nat-filtration M N* **using** *assms bernoulli-stream-equiv psgt pslt*
by *simp*
qed
finally show *?thesis* **.**
qed

lemma (**in** *CRR-market*) *real-exp-eq*:
assumes *der*∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *real-cond-exp N* (*stoch-proc-filt N geom-proc borel n*) *der w* =
expl-cond-expect N (*proj-stoch-proc geom-proc n*) *der w*
proof −
have *der* ∈ *borel-measurable* (*nat-filtration matur*) **using** *assms*
using *geom-rand-walk-borel-adapted measurable-from-subalg stoch-proc-subalg-nat-filt stock-filtration* **by** *blast*
have *integrable N der*
proof (*rule infinite-coin-toss-space.nat-filtration-borel-measurable-integrable*)
show *infinite-coin-toss-space q N* **using** *assms*
by (*simp add*: *infinite-coin-toss-space-def*)
show *der* ∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N matur*)
by (*metis* ‹*der* ∈ *borel-measurable* (*nat-filtration matur*)› ‹*infinite-coin-toss-space q N*›
assms(*2*) *assms*(*3*) *assms*(*4*) *infinite-coin-toss-space.nat-filtration-space measurable-from-subalg*

nat-filtration-from-eq-sets nat-filtration-space subalgebra-def subset-eq)
qed
show *real-cond-exp N* (*stoch-proc-filt N geom-proc borel n*) *der w* =
expl-cond-expect N (*proj-stoch-proc geom-proc n*) *der w*
proof (*rule bernoulli-cond-exp*)
show *N* = *bernoulli-stream q 0* < *q q* < *1* **using** *assms* **by** *auto*
show *integrable N der* **using** ‹*integrable N der*› **.**
qed
qed

lemma (**in** *CRR-market*) *rn-rev-price-rev-borel-adapt*:
assumes *cash-flow* ∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows (*n* ≤ *matur*) ⟹ (*rn-rev-price N cash-flow matur n*) ∈ *borel-measurable* (*G* (*matur* − *n*))
proof (*induct n*)
case *0* **thus** *?case* **using** *assms* **by** *simp*
next
case (*Suc n*)
have *rn-rev-price N cash-flow matur* (*Suc n*) =
(λ*w*. *discount-factor r* (*Suc 0*) *w* ∗
(*expl-cond-expect N* (*proj-stoch-proc geom-proc* (*matur* − *Suc n*)) (*rn-rev-price N cash-flow matur n*)) *w*)
using *rn-rev-price.simps*(*2*) **by** *blast*
also have ... ∈ *borel-measurable* (*G* (*matur* − *Suc n*))
proof (*rule borel-measurable-times*)
show *discount-factor r* (*Suc 0*) ∈ *borel-measurable* (*G* (*matur* − *Suc n*)) **by** (*simp add*:*discount-factor-borel-measurable*)
show *expl-cond-expect N* (*proj-stoch-proc geom-proc* (*matur* − *Suc n*)) (*rn-rev-price N cash-flow matur n*)
∈ *borel-measurable* (*G* (*matur* − *Suc n*)) **using** *assms* **by** (*simp add*: *bernoulli-expl-cond-expect-adapt*)
qed
finally show *rn-rev-price N cash-flow matur* (*Suc n*) ∈ *borel-measurable* (*G* (*matur* − *Suc n*)) **.**
qed

lemma (**in** *infinite-coin-toss-space*) *bernoulli-discounted-integrable*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *der* ∈ *borel-measurable* (*nat-filtration n*)
and −*1* < *r*
shows *integrable N* (*discounted-value r* (λ*m*. *der*) *m*)
proof −
have *prob-space N* **using** *assms*
by (*simp add*: *bernoulli bernoulli-stream-def prob-space.prob-space-stream-space prob-space-measure-pmf*)

have *integrable N der*
proof (*rule infinite-coin-toss-space.nat-filtration-borel-measurable-integrable*)
show *infinite-coin-toss-space q N* **using** *assms*
by (*simp add*: *infinite-coin-toss-space-def*)
show *der* ∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N n*)
using *assms filt-equiv-filtration*
by (*simp add*: *assms*(*1*) *measurable-def nat-filtration-from-eq-sets nat-filtration-space*)
qed
thus *?thesis* **using** *discounted-integrable assms*
by (*metis* ‹*prob-space N*› *prob-space.discounted-integrable*)
qed

lemma (**in** *CRR-market*) *rn-rev-expl-cond-expect*:
assumes *der*∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *n* ≤ *matur* ⟹ *rn-rev-price N der matur n w* =
expl-cond-expect N (*proj-stoch-proc geom-proc* (*matur* − *n*)) (*discounted-value r* (λ*m*. *der*) *n*) *w*
proof (*induct n arbitrary*: *w*)
case *0*
have *der* ∈ *borel-measurable* (*nat-filtration matur*) **using** *assms*
using *geom-rand-walk-borel-adapted measurable-from-subalg stock-filtration stoch-proc-subalg-nat-filt* **by** *blast*
have *integrable N der*
proof (*rule infinite-coin-toss-space.nat-filtration-borel-measurable-integrable*)
show *infinite-coin-toss-space q N* **using** *assms*
by (*simp add*: *infinite-coin-toss-space-def*)
show *der* ∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N matur*)
by (*metis* ‹*der* ∈ *borel-measurable* (*nat-filtration matur*)› ‹*infinite-coin-toss-space q N*›
assms(*2*) *assms*(*3*) *assms*(*4*) *infinite-coin-toss-space.nat-filtration-space measurable-from-subalg*
nat-filtration-from-eq-sets nat-filtration-space subalgebra-def subset-eq)
qed
have *rn-rev-price N der matur 0 w* = *der w* **by** *simp*
also have ... = *expl-cond-expect N* (*proj-stoch-proc geom-proc matur*) (*discounted-value r* (λ*m*. *der*) *0*) *w*
proof (*rule nat-filtration-AE-eq*)
show *der* ∈ *borel-measurable* (*nat-filtration matur*) **using** ‹*der* ∈ *borel-measurable* (*nat-filtration matur*)› **.**
have (*discounted-value r* (λ*m*. *der*) *0*) = *der* **unfolding** *discounted-value-def discount-factor-def* **by** *simp*
moreover have *AEeq N* (*real-cond-exp N* (*G matur*) *der*) *der*
proof (*rule sigma-finite-subalgebra.real-cond-exp-F-meas*)
show *der* ∈ *borel-measurable* (*G matur*) **using** *assms* **by** *simp*

show *integrable N der* **using** ‹*integrable N der*› **.**
show *sigma-finite-subalgebra N* (*G matur*) **using** *bernoulli-sigma-finite*
using *assms* **by** *simp*
qed
moreover have ∀ *w*. *real-cond-exp N* (*stoch-proc-filt N geom-proc borel matur*) *der w* =
expl-cond-expect N (*proj-stoch-proc geom-proc matur*) *der w* **using** *assms real-exp-eq* **by** *simp*
ultimately have *eqn*: *AEeq N der* (*expl-cond-expect N* (*proj-stoch-proc geom-proc matur*) (*discounted-value r* (λ*m*. *der*) *0*))
using *stock-filtration-eq assms* **by** *auto*
have *stoch-proc-filt M geom-proc borel matur* = *stoch-proc-filt N geom-proc borel matur*
using *bernoulli-stream-equiv*[*of N q*] *assms psgt pslt* **by** (*simp add*: *stoch-proc-filt-filt-equiv*)
also have *stoch-proc-filt N geom-proc borel matur* =
fct-gen-subalgebra N (*stream-space borel*) (*proj-stoch-proc geom-proc matur*)
using *assms geom-proc-stoch-proc-filt* **by** *simp*
finally have *stoch-proc-filt M geom-proc borel matur* =
fct-gen-subalgebra N (*stream-space borel*) (*proj-stoch-proc geom-proc matur*) **.**
moreover have *expl-cond-expect N* (*proj-stoch-proc geom-proc matur*) (*discounted-value r* (λ*m*. *der*) *0*)
∈ *borel-measurable* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc matur*))
proof (*rule expl-cond-exp-borel*)
show *proj-stoch-proc geom-proc matur* ∈ *space N* → *space* (*stream-space borel*)
using *assms proj-stoch-proc-geom-rng* **by** (*simp add*: *measurable-def*)
show *disc-fct* (*proj-stoch-proc geom-proc matur*) **using** *proj-stoch-proc-geom-disc-fct* **by** *simp*
show ∀ *r*∈*range* (*proj-stoch-proc geom-proc matur*) ∩ *space* (*stream-space borel*).
∃ *A*∈*sets* (*stream-space borel*). *range* (*proj-stoch-proc geom-proc matur*) ∩ *A* = {*r*}
using *proj-stoch-proc-geom-open-set* **by** *simp*
qed
ultimately show *ebm*: *expl-cond-expect N* (*proj-stoch-proc geom-proc matur*) (*discounted-value r* (λ*m*. *der*) *0*)
∈ *borel-measurable* (*nat-filtration matur*)
by (*metis geom-rand-walk-borel-adapted measurable-from-subalg stoch-proc-subalg-nat-filt*)
show *AEeq M der* (*expl-cond-expect N* (*proj-stoch-proc geom-proc matur*) (*discounted-value r* (λ*m*. *der*) *0*))
proof (*rule filt-equiv-borel-AE-eq-iff*[*THEN iffD2*])
show *filt-equiv nat-filtration M N* **using** *assms bernoulli-stream-equiv psgt pslt* **by** *simp*
show *der* ∈ *borel-measurable* (*nat-filtration matur*) **using** ‹*der* ∈ *borel-measurable* (*nat-filtration matur*)› **.**
show *AEeq N der* (*expl-cond-expect N* (*proj-stoch-proc geom-proc matur*) (*discounted-value r* (λ*m*. *der*) *0*))
using *eqn* **.**
show *expl-cond-expect N* (*proj-stoch-proc geom-proc matur*) (*discounted-value*

```
r (λm. der) 0)
        ∈ borel-measurable (nat-filtration matur) using ebm .
      show prob-space N using assms by (simp add: bernoulli-stream-def
          prob-space.prob-space-stream-space prob-space-measure-pmf)
      show prob-space M by (simp add: bernoulli bernoulli-stream-def
          prob-space.prob-space-stream-space prob-space-measure-pmf)
    qed
    show 0 < p p < 1 using psgt pslt by auto
  qed
  also have ... = expl-cond-expect N (proj-stoch-proc geom-proc (matur − 0))
(discounted-value r (λm. der) 0) w
    by simp
  finally show rn-rev-price N der matur 0 w =
    expl-cond-expect N (proj-stoch-proc geom-proc (matur − 0)) (discounted-value
r (λm. der) 0) w .
next
  case (Suc n)
  have rn-rev-price N der matur (Suc n) w = discount-factor r (Suc 0) w ∗
       expl-cond-expect N (proj-stoch-proc geom-proc (matur − Suc n)) (rn-rev-price
N der matur n) w by simp
  also have ... = discount-factor r (Suc 0) w ∗
   real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n)) (rn-rev-price
N der matur n) w
  proof −
   have expl-cond-expect N (proj-stoch-proc geom-proc (matur − Suc n)) (rn-rev-price
N der matur n) w =
    real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n)) (rn-rev-price
N der matur n) w
    proof (rule real-exp-eq[symmetric])
      show rn-rev-price N der matur n ∈ borel-measurable (G (matur − n))
        using assms rn-rev-price-rev-borel-adapt Suc by simp
      show N = bernoulli-stream q 0 < q q < 1 using assms by auto
    qed
    thus ?thesis by simp
  qed
  also have ... = discount-factor r (Suc 0) w ∗
    real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n))
    (expl-cond-expect N (proj-stoch-proc geom-proc (matur − n)) (discounted-value
r (λm. der) n)) w
  proof −
     have real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n))
(rn-rev-price N der matur n) w =
      real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n))
    (expl-cond-expect N (proj-stoch-proc geom-proc (matur − n)) (discounted-value
r (λm. der) n)) w
    proof (rule infinite-coin-toss-space.nat-filtration-AE-eq)
      show AEeq N (real-cond-exp N (stoch-proc-filt N geom-proc borel (matur −
Suc n)) (rn-rev-price N der matur n))
        (real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n))
```

(*expl-cond-expect N (proj-stoch-proc geom-proc (matur − n)) (discounted-value r (λm. der) n)*))
proof (*rule sigma-finite-subalgebra.real-cond-exp-cong*)
show *sigma-finite-subalgebra N (stoch-proc-filt N geom-proc borel (matur − Suc n))*
using *assms(2) assms(3) assms(4) bernoulli-sigma-finite stock-filtration-eq* **by** *auto*
show *rn-rev-price N der matur n ∈ borel-measurable N*
proof −
have *rn-rev-price N der matur n ∈ borel-measurable (G (matur − n))*
by (*metis* (*full-types*) *Suc.prems Suc-leD assms(1) assms(2) assms(3) assms(4) rn-rev-price-rev-borel-adapt*)
then show *?thesis*
by (*metis* (*no-types*) *assms(2) bernoulli bernoulli-stream-def filtration-measurable measurable-cong-sets sets-measure-pmf sets-stream-space-cong*)
qed
show *expl-cond-expect N (proj-stoch-proc geom-proc (matur − n)) (discounted-value r (λm. der) n) ∈ borel-measurable N*
using *Suc.hyps Suc.prems Suc-leD ‹rn-rev-price N der matur n ∈ borel-measurable N›* **by** *presburger*
show *AEeq N (rn-rev-price N der matur n)*
(*expl-cond-expect N (proj-stoch-proc geom-proc (matur − n)) (discounted-value r (λm. der) n)*) **using** *Suc* **by** *auto*
qed
show *real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n)) (rn-rev-price N der matur n)*
∈ borel-measurable (infinite-coin-toss-space.nat-filtration N (matur − Suc n))
by (*metis assms(2) assms(3) assms(4) borel-measurable-cond-exp infinite-coin-toss-space.intro*
infinite-coin-toss-space.stoch-proc-subalg-nat-filt linear measurable-from-subalg not-less
prob-grw.geom-rand-walk-borel-adapted prob-grw-axioms prob-grw-def)
show *real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n)) (expl-cond-expect N (proj-stoch-proc geom-proc (matur − n)) (discounted-value r (λm. der) n))*
∈ borel-measurable (infinite-coin-toss-space.nat-filtration N (matur − Suc n))
by (*metis assms(2) assms(3) assms(4) borel-measurable-cond-exp infinite-coin-toss-space.intro*
infinite-coin-toss-space.stoch-proc-subalg-nat-filt linear measurable-from-subalg not-less
prob-grw.geom-rand-walk-borel-adapted prob-grw-axioms prob-grw-def)
show *0 < q q < 1* **using** *assms* **by** *auto*
show *infinite-coin-toss-space q N* **using** *assms*
by (*simp add*: *infinite-coin-toss-space-def*)
qed
thus *?thesis* **by** *simp*
qed

also have ... = *discount-factor r* (*Suc 0*) *w* ∗
real-cond-exp N (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*)) *w*
proof −
have *real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*expl-cond-expect N* (*proj-stoch-proc geom-proc* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*)) *w* =
real-cond-exp N (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*)) *w*
proof (*rule infinite-coin-toss-space.nat-filtration-AE-eq*)
show *AEeq N* (*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*expl-cond-expect N* (*proj-stoch-proc geom-proc* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*)))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*)))
proof (*rule sigma-finite-subalgebra.real-cond-exp-cong*)
show *sigma-finite-subalgebra N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
using *assms*(*2*) *assms*(*3*) *assms*(*4*) *bernoulli-sigma-finite stock-filtration-eq*
by *auto*
show *real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*) ∈ *borel-measurable N*
by *simp*
show *expl-cond-expect N* (*proj-stoch-proc geom-proc* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*) ∈ *borel-measurable N*
by (*metis assms*(*2*) *assms*(*3*) *assms*(*4*) *bernoulli bernoulli-expl-cond-expect-adapt bernoulli-stream-def filtration-measurable*
measurable-cong-sets sets-measure-pmf sets-stream-space-cong)
show *AEeq N* (*expl-cond-expect N* (*proj-stoch-proc geom-proc* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*))
proof −
have *discounted-value r* (λ*m. der*) *n* ∈ *borel-measurable* (*G matur*) **using** *assms discounted-measurable*[*of der*]
by *simp*
hence ∀ *w*. (*expl-cond-expect N* (*proj-stoch-proc geom-proc* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*)) *w* =
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*)) *w*
using *real-exp-eq*[*of - matur N q matur*−*n*] *assms* **by** *simp*
thus *?thesis* **by** *simp*
qed
qed
show *real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))

(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*))
∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N* (*matur* − *Suc n*))
by (*metis assms*(*2*) *assms*(*3*) *assms*(*4*) *borel-measurable-cond-exp infinite-coin-toss-space.intro*
infinite-coin-toss-space.stoch-proc-subalg-nat-filt linear measurable-from-subalg not-less
prob-grw.geom-rand-walk-borel-adapted prob-grw-axioms prob-grw-def)
show *real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*expl-cond-expect N* (*proj-stoch-proc geom-proc* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*))
∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N* (*matur* − *Suc n*))
by (*metis assms*(*2*) *assms*(*3*) *assms*(*4*) *borel-measurable-cond-exp infinite-coin-toss-space.intro*
infinite-coin-toss-space.stoch-proc-subalg-nat-filt linear measurable-from-subalg not-less
prob-grw.geom-rand-walk-borel-adapted prob-grw-axioms prob-grw-def)
show *0* < *q q* < *1* **using** *assms* **by** *auto*
show *infinite-coin-toss-space q N* **using** *assms*
by (*simp add*: *infinite-coin-toss-space-def*)
qed
thus *?thesis* **by** *simp*
qed
also have ... = *real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*discounted-value r* (λ*m. der*) (*Suc n*)) *w*
proof (*rule infinite-coin-toss-space.nat-filtration-AE-eq*)
show *real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*)) (*discounted-value r* (λ*m. der*) (*Suc n*))
∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N* (*matur* − *Suc n*))
by (*metis assms*(*2*) *assms*(*3*) *assms*(*4*) *borel-measurable-cond-exp infinite-coin-toss-space.intro*
infinite-coin-toss-space.stoch-proc-subalg-nat-filt linear measurable-from-subalg not-less
prob-grw.geom-rand-walk-borel-adapted prob-grw-axioms prob-grw-def)
show (λ*a. discount-factor r* (*Suc 0*) *a* ∗
real-cond-exp N (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*)) *a*)
∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N* (*matur* − *Suc n*))
proof −
have *real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*))
∈ *borel-measurable* (*infinite-coin-toss-space.nat-filtration N* (*matur* − *Suc n*))

by (*metis assms*(2) *assms*(3) *assms*(4) *borel-measurable-cond-exp infinite-coin-toss-space.intro*
infinite-coin-toss-space.stoch-proc-subalg-nat-filt linear measurable-from-subalg not-less
prob-grw.geom-rand-walk-borel-adapted prob-grw-axioms prob-grw-def)
thus *?thesis* **using** *discounted-measurable*[*of real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*))]
unfolding *discounted-value-def* **by** *simp*
qed
show *0* < *q q* < *1* **using** *assms* **by** *auto*
show *infinite-coin-toss-space q N* **using** *assms*
by (*simp add*: *infinite-coin-toss-space-def*)
show *AEeq N* (λ*w. discount-factor r* (*Suc 0*) *w* ∗
real-cond-exp N (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*))
(*discounted-value r* (λ*m. der*) *n*)) *w*)
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*)) (*discounted-value r* (λ*m. der*) (*Suc n*)))
proof−
have *AEeq N*
(λ*w. discount-factor r* (*Suc 0*) *w* ∗
real-cond-exp N (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*))
(*discounted-value r* (λ*m. der*) *n*)) *w*)
(λ*w. discount-factor r* (*Suc 0*) *w* ∗
real-cond-exp N (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*discounted-value r* (λ*m. der*) *n*) *w*)
proof −
have *AEeq N* (*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*))
(*discounted-value r* (λ*m. der*) *n*)))
(*real-cond-exp N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
(*discounted-value r* (λ*m. der*) *n*))
proof (*rule sigma-finite-subalgebra.real-cond-exp-nested-subalg*)
show *sigma-finite-subalgebra N* (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
using *assms*(2) *assms*(3) *assms*(4) *bernoulli-sigma-finite stock-filtration-eq*
by *auto*
show *subalgebra N* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*))
using *assms*(2) *assms*(3) *assms*(4) *bernoulli-sigma-finite sigma-finite-subalgebra.subalg*
stock-filtration-eq **by** *fastforce*
show *subalgebra* (*stoch-proc-filt N geom-proc borel* (*matur* − *n*)) (*stoch-proc-filt N geom-proc borel* (*matur* − *Suc n*))
proof −
have *init-triv-filt M* (*stoch-proc-filt M geom-proc borel*) **using** *infinite-cts-filtration.stoch-proc-filt-triv-init*

using *info-filtration stock-filtration* **by** *auto*
moreover have *matur − (Suc n) ≤ matur − n* **by** *simp*
ultimately show *?thesis* **unfolding** *init-triv-filt-def filtration-def*
using *assms(2) assms(3) assms(4) stock-filtration stock-filtration-eq*
by *auto*
qed
show *integrable N (discounted-value r (λm. der) n)* **using** *bernoulli-discounted-integrable[of N q der matur r n] acceptable-rate assms*
using *geom-rand-walk-borel-adapted measurable-from-subalg stoch-proc-subalg-nat-filt stock-filtration* **by** *blast*
qed
thus *?thesis* **by** *auto*
qed
moreover have *AEeq N*
(λw. discount-factor r (Suc 0) w ∗
real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n)) (discounted-value r (λm. der) n) w)
(λw. discount-factor r (Suc 0) w ∗ (discounted-value r
(λm. real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n)) der) n) w)
proof −
have *AEeq N (real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n)) (discounted-value r (λm. der) n))*
(discounted-value r
(λm. real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n)) der) n)
proof (*rule prob-space.discounted-value-real-cond-exp*)
show *prob-space N* **using** *assms*
by (*simp add*: *bernoulli bernoulli-stream-def prob-space.prob-space-stream-space prob-space-measure-pmf*)
have *der ∈ borel-measurable (nat-filtration matur)* **using** *assms*
using *geom-rand-walk-borel-adapted measurable-from-subalg stoch-proc-subalg-nat-filt stock-filtration* **by** *blast*
show *integrable N der*
proof (*rule infinite-coin-toss-space.nat-filtration-borel-measurable-integrable*)
show *infinite-coin-toss-space q N* **using** *assms*
by (*simp add*: *infinite-coin-toss-space-def*)
show *der ∈ borel-measurable (infinite-coin-toss-space.nat-filtration N matur)*
by (*metis ‹der ∈ borel-measurable (nat-filtration matur)› ‹infinite-coin-toss-space q N› assms(2) assms(3) assms(4) infinite-coin-toss-space.nat-filtration-space measurable-from-subalg*
nat-filtration-from-eq-sets nat-filtration-space subalgebra-def subset-eq)
qed
show *−1 < r* **using** *acceptable-rate* **.**
show *subalgebra N (stoch-proc-filt N geom-proc borel (matur − Suc n))*
using *assms(2) assms(3) assms(4) bernoulli-sigma-finite sigma-finite-subalgebra.subalg stock-filtration-eq* **by** *fastforce*

```
      qed
      thus ?thesis  by auto
    qed
    moreover have ∀ w. (λw. discount-factor r (Suc 0) w * (discounted-value r
      (λm. real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n))
der) n) w) w =
      (discounted-value r
       (λm. real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n))
der) (Suc n)) w
      unfolding discounted-value-def discount-factor-def  by simp
    moreover have AEeq N
      (real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n))
      (discounted-value r (λm. der) (Suc n)))
      (discounted-value r
       (λm. real-cond-exp N (stoch-proc-filt N geom-proc borel (matur − Suc n))
der) (Suc n))
    proof (rule prob-space.discounted-value-real-cond-exp)
      show prob-space N using assms
       by (simp add: bernoulli bernoulli-stream-def prob-space.prob-space-stream-space
prob-space-measure-pmf)
      have der ∈ borel-measurable (nat-filtration matur) using assms
     using geom-rand-walk-borel-adapted measurable-from-subalg stoch-proc-subalg-nat-filt
stock-filtration by blast
      show integrable N der
     proof (rule infinite-coin-toss-space.nat-filtration-borel-measurable-integrable)
       show infinite-coin-toss-space q N using assms
         by (simp add: infinite-coin-toss-space-def)
      show der ∈ borel-measurable (infinite-coin-toss-space.nat-filtration N matur)
       by (metis ‹der ∈ borel-measurable (nat-filtration matur)› ‹infinite-coin-toss-space
q N›
           assms(2) assms(3) assms(4) infinite-coin-toss-space.nat-filtration-space
measurable-from-subalg
             nat-filtration-from-eq-sets nat-filtration-space subalgebra-def subset-eq)
      qed
      show −1 < r using acceptable-rate .
      show subalgebra N (stoch-proc-filt N geom-proc borel (matur − Suc n))
     using assms(2) assms(3) assms(4) bernoulli-sigma-finite sigma-finite-subalgebra.subalg
       stock-filtration-eq by fastforce
    qed
    ultimately show ?thesis by auto
  qed
 qed
 also have ... = expl-cond-expect N (proj-stoch-proc geom-proc (matur − Suc n))
  (discounted-value r (λm. der) (Suc n)) w
 proof (rule real-exp-eq)
  show discounted-value r (λm. der) (Suc n) ∈ borel-measurable (G matur) using
assms discounted-measurable[of der]
    by simp
  show N = bernoulli-stream q 0 < q q < 1 using assms by auto
```

qed
finally show *rn-rev-price N der matur* (*Suc n*) *w* =
expl-cond-expect N (*proj-stoch-proc geom-proc* (*matur* − *Suc n*)) (*discounted-value r* (λ*m. der*) (*Suc n*)) *w* **.**
qed

definition (**in** *CRR-market*) *rn-price* **where**
rn-price N der matur n w = *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) (*discounted-value r* (λ*m. der*) (*matur* − *n*)) *w*

definition (**in** *CRR-market*) *rn-price-ind* **where**
rn-price-ind N der matur n w = *rn-rev-price N der matur* (*matur* − *n*) *w*

lemma (**in** *CRR-market*) *rn-price-eq*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *der* ∈ *borel-measurable* (*G matur*)
and *n* ≤ *matur*
shows *rn-price N der matur n w* = *rn-price-ind N der matur n w* **using** *rn-rev-expl-cond-expect*
unfolding *rn-price-def rn-price-ind-def*
by (*simp add*: *assms*)

lemma (**in** *CRR-market*) *geom-proc-filt-info*:
fixes *f*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *f* ∈ *borel-measurable* (*G n*)
shows *f w* = *f* (*pseudo-proj-True n w*)
proof −
have *subalgebra* (*nat-filtration n*) (*G n*) **using** *stoch-proc-subalg-nat-filt*[*of geom-proc n*] *geometric-process*
stock-filtration geom-rand-walk-borel-adapted **by** *simp*
hence *f*∈ *borel-measurable* (*nat-filtration n*) **using** *assms* **by** (*simp add*: *measurable-from-subalg*)
thus *?thesis* **using** *nat-filtration-info*[*of f n*] **by** (*metis comp-apply*)
qed

lemma (**in** *CRR-market*) *geom-proc-filt-info'*:
fixes *f*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *f* ∈ *borel-measurable* (*G n*)
shows *f w* = *f* (*pseudo-proj-False n w*)
proof −
have *subalgebra* (*nat-filtration n*) (*G n*) **using** *stoch-proc-subalg-nat-filt*[*of geom-proc n*] *geometric-process*
stock-filtration geom-rand-walk-borel-adapted **by** *simp*
hence *f*∈ *borel-measurable* (*nat-filtration n*) **using** *assms* **by** (*simp add*: *measurable-from-subalg*)
thus *?thesis* **using** *nat-filtration-info'*[*of f n*] **by** (*metis comp-apply*)

qed

lemma (**in** *CRR-market*) *rn-price-borel-adapt*:
assumes *cash-flow* ∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *n* ≤ *matur*
shows (*rn-price N cash-flow matur n*) ∈ *borel-measurable* (*G n*)
proof −
show (*rn-price N cash-flow matur n*) ∈ *borel-measurable* (*G n*)
using *assms rn-rev-price-rev-borel-adapt*[*of cash-flow matur N q matur* − *n*]
rn-price-eq rn-price-ind-def
by (*smt add.right-neutral cancel-comm-monoid-add-class.diff-cancel diff-commute*
diff-le-self
increasing-measurable-info measurable-cong nat-le-linear ordered-cancel-comm-monoid-diff-class.add-diff-i
qed

definition (**in** *CRR-market*) *delta-price* **where**
delta-price N cash-flow T =
(λ *n w*. *if* (*Suc n* ≤ *T*)
then (*rn-price N cash-flow T* (*Suc n*) (*pseudo-proj-True n w*) − *rn-price N*
cash-flow T (*Suc n*) (*pseudo-proj-False n w*))/
(*geom-proc* (*Suc n*) (*spick w n True*) − *geom-proc* (*Suc n*) (*spick w n False*))
else 0)

lemma (**in** *CRR-market*) *delta-price-eq*:
assumes *Suc n* ≤ *T*
shows *delta-price N cash-flow T n w* = (*rn-price N cash-flow T* (*Suc n*) (*spick*
w n True) − *rn-price N cash-flow T* (*Suc n*) (*spick w n False*))/
((*geom-proc n w*) ∗ (*u* − *d*))
proof −
have (*geom-proc* (*Suc n*) (*spick w n True*) − *geom-proc* (*Suc n*) (*spick w n False*))
= *geom-proc n w* ∗ (*u* − *d*)
by (*simp add*: *geom-rand-walk-diff-induct*)
then show *?thesis* **unfolding** *delta-price-def* **using** *assms spick-eq-pseudo-proj-True*
spick-eq-pseudo-proj-False **by** *simp*
qed

lemma (**in** *CRR-market*) *geom-proc-spick*:
shows *geom-proc* (*Suc n*) (*spick w n x*) = (*if x then u else d*) ∗ *geom-proc n w*
proof −

have *geom-proc* (*Suc n*) (*spick w n x*) = *geom-rand-walk u d init* (*Suc n*) (*spick w n x*) **using** *geometric-process* **by** *simp*
also have ... = (*case* (*spick w n x*) !! *n of True* ⇒ *u* | *False* ⇒ *d*) ∗ *geom-rand-walk u d init n* (*spick w n x*)
by *simp*
also have ... = (*case x of True* ⇒ *u* | *False* ⇒ *d*) ∗ *geom-rand-walk u d init n* (*spick w n x*)
unfolding *spick-def* **by** *simp*
also have ... = (*if x then u else d*) ∗ *geom-rand-walk u d init n* (*spick w n x*) **by** *simp*
also have ... = (*if x then u else d*) ∗ *geom-rand-walk u d init n w*
by (*metis comp-def geom-rand-walk-pseudo-proj-True geometric-process pseudo-proj-True-stake-image spickI*)
finally show *?thesis* **using** *geometric-process* **by** *simp*
qed

lemma (**in** *CRR-market*) *spick-red-geom*:
shows (λ*w*. *spick w n x*) ∈ *measurable* (*fct-gen-subalgebra M borel* (*geom-proc n*)) (*fct-gen-subalgebra M borel* (*geom-proc* (*Suc n*)))
unfolding *measurable-def*
proof (*intro CollectI conjI*)
show (λ*w*. *spick w n x*)
∈ *space* (*fct-gen-subalgebra M borel* (*geom-proc n*)) → *space* (*fct-gen-subalgebra M borel* (*geom-proc* (*Suc n*)))
by (*simp add*: *bernoulli bernoulli-stream-space fct-gen-subalgebra-space*)
show ∀ *y*∈*sets* (*fct-gen-subalgebra M borel* (*geom-proc* (*Suc n*))).
(λ*w*. *spick w n x*) −‘ *y* ∩ *space* (*fct-gen-subalgebra M borel* (*geom-proc n*))
∈ *sets* (*fct-gen-subalgebra M borel* (*geom-proc n*))
proof
fix *A*
assume *A*: *A* ∈ *sets* (*fct-gen-subalgebra M borel* (*geom-proc* (*Suc n*)))
show (λ*w*. *spick w n x*) −‘ *A* ∩ *space* (*fct-gen-subalgebra M borel* (*geom-proc n*)) ∈
sets (*fct-gen-subalgebra M borel* (*geom-proc n*))
proof −
define *sp* **where** *sp* = (λ*w*. *spick w n x*)
have *A* ∈ {(*geom-proc* (*Suc n*)) −‘ *B* ∩ *space M* |*B*. *B* ∈ *sets borel*} **using** *A*
by (*simp add*:*fct-gen-subalgebra-sigma-sets*)
from *this* **obtain** *C* **where** *C*∈ *sets borel* **and** *A* = (*geom-proc* (*Suc n*)) −‘*C* ∩ *space M* **by** *auto*
hence *A* = (*geom-proc* (*Suc n*)) −‘*C* **using** *bernoulli bernoulli-stream-space* **by** *simp*
hence *sp* −‘*A* = *sp* −‘ (*geom-proc* (*Suc n*)) −‘*C* **by** *simp*
also have ... = (*geom-proc* (*Suc n*) ∘ *sp*) −‘ *C* **by** *auto*
also have ... = (λ*w*. (*if x then u else d*) ∗ *geom-proc n w*) −‘ *C* **using** *geom-proc-spick*
sp-def **by** *auto*
also have ... ∈ *sets* (*fct-gen-subalgebra M borel* (*geom-proc n*))

proof (*cases x*)
case *True*
hence (λ*w*. (*if x then u else d*) ∗ *geom-proc n w*) −‘ *C* = (λ*w*. *u* ∗ *geom-proc n w*) −‘ *C* **by** *simp*
moreover have (λ*w*. *u* ∗ *geom-proc n w*) ∈ *borel-measurable* (*fct-gen-subalgebra M borel* (*geom-proc n*))
proof −
have *geom-proc n* ∈*borel-measurable* (*fct-gen-subalgebra M borel* (*geom-proc n*))
using *fct-gen-subalgebra-fct-measurable*
by (*metis* (*no-types*, *lifting*) *geom-rand-walk-borel-measurable measurable-def mem-Collect-eq*)
thus *?thesis* **by** *simp*
qed
ultimately show *?thesis* **using** ‹*C*∈ *sets borel*›
by (*metis bernoulli bernoulli-stream-preimage fct-gen-subalgebra-space measurable-sets*)
next
case *False*
hence (λ*w*. (*if x then u else d*) ∗ *geom-proc n w*) −‘ *C* = (λ*w*. *d* ∗ *geom-proc n w*) −‘ *C* **by** *simp*
moreover have (λ*w*. *d* ∗ *geom-proc n w*) ∈ *borel-measurable* (*fct-gen-subalgebra M borel* (*geom-proc n*))
proof −
have *geom-proc n* ∈*borel-measurable* (*fct-gen-subalgebra M borel* (*geom-proc n*))
using *fct-gen-subalgebra-fct-measurable*
by (*metis* (*no-types*, *lifting*) *geom-rand-walk-borel-measurable measurable-def mem-Collect-eq*)
thus *?thesis* **by** *simp*
qed
ultimately show *?thesis* **using** ‹*C*∈ *sets borel*›
by (*metis bernoulli bernoulli-stream-preimage fct-gen-subalgebra-space measurable-sets*)
qed
finally show *?thesis* **unfolding** *sp-def* **by** (*simp add*: *bernoulli bernoulli-stream-space fct-gen-subalgebra-space*)
qed
qed
qed

lemma (**in** *CRR-market*) *geom-spick-Suc*:
assumes *A* ∈ {(*geom-proc* (*Suc n*)) −‘ *B* |*B*. *B* ∈ *sets borel*}
shows (λ*w*. *spick w n x*) −‘*A* ∈ {*geom-proc n* −‘*B* | *B*. *B*∈ *sets borel*}
proof −
have *sets* (*fct-gen-subalgebra M borel* (*geom-proc n*)) = {*geom-proc n* −‘ *B* ∩*space M* |*B*. *B* ∈ *sets borel*}
by (*simp add*: *fct-gen-subalgebra-sigma-sets*)
also have ... = {*geom-proc n* −‘ *B* |*B*. *B* ∈ *sets borel*} **using** *bernoulli bernoulli-stream-space*

by *simp*
finally have *sf*: *sets* (*fct-gen-subalgebra M borel* (*geom-proc n*)) = {*geom-proc n* −‘ *B* |*B*. *B* ∈ *sets borel*} **.**
define *sp* **where** *sp* = (λ*w*. *spick w n x*)
from *assms*(*1*) **obtain** *C* **where** *C*∈ *sets borel* **and** *A* = (*geom-proc* (*Suc n*)) −‘*C* **by** *auto*
hence *A* = (*geom-proc* (*Suc n*)) −‘*C* **using** *bernoulli bernoulli-stream-space* **by** *simp*
hence *sp* −‘*A* = *sp* −‘ (*geom-proc* (*Suc n*)) −‘*C* **by** *simp*
also have ... = (*geom-proc* (*Suc n*) ∘ *sp*) −‘ *C* **by** *auto*
also have ... = (λ*w*. (*if x then u else d*) ∗ *geom-proc n w*) −‘ *C* **using** *geom-proc-spick sp-def* **by** *auto*
also have ... ∈ {*geom-proc n* −‘*B* | *B*. *B*∈ *sets borel*}
proof (*cases x*)
case *True*
hence (λ*w*. (*if x then u else d*) ∗ *geom-proc n w*) −‘ *C* = (λ*w*. *u* ∗ *geom-proc n w*) −‘ *C* **by** *simp*
moreover have (λ*w*. *u* ∗ *geom-proc n w*) ∈ *borel-measurable* (*fct-gen-subalgebra M borel* (*geom-proc n*))
proof −
have *geom-proc n* ∈*borel-measurable* (*fct-gen-subalgebra M borel* (*geom-proc n*))
using *fct-gen-subalgebra-fct-measurable*
by (*metis* (*no-types*, *lifting*) *geom-rand-walk-borel-measurable measurable-def mem-Collect-eq*)
thus *?thesis* **by** *simp*
qed
ultimately show *?thesis* **using** ‹*C*∈ *sets borel*› *sf*
by (*simp add*: *bernoulli bernoulli-stream-preimage fct-gen-subalgebra-space in-borel-measurable-borel*)
next
case *False*
hence (λ*w*. (*if x then u else d*) ∗ *geom-proc n w*) −‘ *C* = (λ*w*. *d* ∗ *geom-proc n w*) −‘ *C* **by** *simp*
moreover have (λ*w*. *d* ∗ *geom-proc n w*) ∈ *borel-measurable* (*fct-gen-subalgebra M borel* (*geom-proc n*))
proof −
have *geom-proc n* ∈*borel-measurable* (*fct-gen-subalgebra M borel* (*geom-proc n*))
using *fct-gen-subalgebra-fct-measurable*
by (*metis* (*no-types*, *lifting*) *geom-rand-walk-borel-measurable measurable-def mem-Collect-eq*)
thus *?thesis* **by** *simp*
qed
ultimately show *?thesis* **using** ‹*C*∈ *sets borel*› *sf*
by (*simp add*: *bernoulli bernoulli-stream-preimage fct-gen-subalgebra-space in-borel-measurable-borel*)
qed
finally show *?thesis* **unfolding** *sp-def* **.**

qed

lemma (**in** *CRR-market*) *geom-spick-lt*:
assumes $m< n$
shows *geom-proc m* (*spick w n x*) = *geom-proc m w*
proof −
have *geom-proc m* (*spick w n x*) = *geom-proc m* (*pseudo-proj-True m* (*spick w n x*))
using *geom-rand-walk-pseudo-proj-True* **by** (*metis comp-apply*)
also have ... = *geom-proc m* (*pseudo-proj-True m w*) **using** *assms*
by (*metis less-imp-le-nat pseudo-proj-True-def pseudo-proj-True-prefix spickI*)
also have ... = *geom-proc m w* **using** *geom-rand-walk-pseudo-proj-True* **by** (*metis comp-apply*)
finally show *?thesis* **.**
qed

lemma (**in** *CRR-market*) *geom-spick-eq*:
shows *geom-proc m* (*spick w m x*) = *geom-proc m w*
proof (*cases x*)
case *True*
have *geom-proc m* (*spick w m x*) = *geom-proc m* (*pseudo-proj-True m* (*spick w m x*))
using *geom-rand-walk-pseudo-proj-True* **by** (*metis comp-apply*)
also have ... = *geom-proc m* (*pseudo-proj-True m w*) **using** *True*
by (*metis pseudo-proj-True-def spickI*)
also have ... = *geom-proc m w* **using** *geom-rand-walk-pseudo-proj-True* **by** (*metis comp-apply*)
finally show *?thesis* **.**
next
case *False*
have *geom-proc m* (*spick w m x*) = *geom-proc m* (*pseudo-proj-False m* (*spick w m x*))
using *geom-rand-walk-pseudo-proj-False* **by** (*metis comp-apply*)
also have ... = *geom-proc m* (*pseudo-proj-False m w*) **using** *False*
by (*metis pseudo-proj-False-def spickI*)
also have ... = *geom-proc m w* **using** *geom-rand-walk-pseudo-proj-False* **by** (*metis comp-apply*)
finally show *?thesis* **.**
qed

lemma (**in** *CRR-market*) *spick-red-geom-filt*:
shows (λ*w*. *spick w n x*) ∈ *measurable* (*G n*) (*G* (*Suc n*)) **unfolding** *measurable-def*
proof (*intro CollectI conjI*)
show (λ*w*. *spick w n x*) ∈ *space* (*G n*) → *space* (*G* (*Suc n*)) **using** *stock-filtration*
by (*simp add*: *bernoulli bernoulli-stream-space stoch-proc-filt-space*)
show ∀ *y*∈*sets* (*G* (*Suc n*)). (λ*w*. *spick w n x*) −‘ *y* ∩ *space* (*G n*) ∈ *sets* (*G n*)

proof
fix B
assume $B\in$ *sets* (*G* (*Suc n*))
hence $B\in$ (*sigma-sets* (*space M*) ($\bigcup$ $i\in$ {*m*. *m*$\leq$ (*Suc n*)}. {(*geom-proc i* $-$‘*A*) $\cap$ (*space M*) | *A*. *A*$\in$ *sets borel* }))
using *stock-filtration stoch-proc-filt-sets geometric-process*
proof $-$
have $\forall$ *n*. *sigma-sets* (*space M*) ($\bigcup n\in${*na*. *na* $\leq$ *n*}. {*geom-proc n* $-$‘ *R* $\cap$ *space M* |*R*. *R* $\in$ *sets borel*}) = *sets* (*G n*)
by (*simp add*: *geom-rand-walk-borel-measurable stoch-proc-filt-sets stock-filtration*)
then show *?thesis*
using ‹*B* $\in$ *sets* (*G* (*Suc n*))› **by** *blast*
qed
hence (λ*w*. *spick w n x*) $-$‘ *B* $\in$ *sets* (*G n*)
proof (*induct rule*:*sigma-sets.induct*)
{
fix *C*
assume *C* $\in$ ($\bigcup i\in${*m*. *m* $\leq$ *Suc n*}. {*geom-proc i* $-$‘ *A* $\cap$ *space M* |*A*. *A* $\in$ *sets borel*})
hence $\exists$ *m* $\leq$ *Suc n*. *C*$\in$ {*geom-proc m* $-$‘ *A* $\cap$ *space M* |*A*. *A* $\in$ *sets borel*} **by** *auto*
from *this* **obtain** *m* **where** *m*$\leq$ *Suc n* **and** *C*$\in$ {*geom-proc m* $-$‘ *A* $\cap$ *space M* |*A*. *A* $\in$ *sets borel*} **by** *auto*
note *Cprops* = *this*
from *this* **obtain** *D* **where** *C* = *geom-proc m* $-$‘ *D*$\cap$ *space M* **and** *D*$\in$ *sets borel* **by** *auto*
hence *C* = *geom-proc m* $-$‘*D* **using** *bernoulli bernoulli-stream-space* **by** *simp*
have *C*$\in$ {*geom-proc m* $-$‘ *A* |*A*. *A* $\in$ *sets borel*} **using** *bernoulli bernoulli-stream-space Cprops* **by** *simp*
show (λ*w*. *spick w n x*) $-$‘ *C* $\in$ *sets* (*G n*)
proof (*cases m* $\leq$ *n*)
case *True*
have (λ*w*. *spick w n x*) $-$‘ *C* = (λ*w*. *spick w n x*) $-$‘ *geom-proc m* $-$‘*D* **using** ‹*C* = *geom-proc m* $-$‘*D*› **by** *simp*
also have ... = (*geom-proc m* $\circ$ (λ*w*. *spick w n x*)) $-$‘*D* **by** *auto*
also have ... = *geom-proc m* $-$‘*D* **using** *geom-spick-lt geom-spick-eq* ‹*m*$\leq$*n*›
using *le-eq-less-or-eq* **by** *auto*
also have ... $\in$ *sets* (*G n*) **using** *stock-filtration geometric-process* ‹*D*$\in$ *sets borel*›
by (*metis* (*no-types*, *lifting*) *True adapt-stoch-proc-def bernoulli bernoulli-stream-preimage geom-rand-walk-borel-measurable increasing-measurable-info measurable-sets stoch-proc-filt-adapt stoch-proc-filt-space*)
finally show (λ*w*. *spick w n x*) $-$‘ *C* $\in$ *sets* (*G n*) **.**
next
case *False*
hence *m* = *Suc n* **using** ‹*m* $\leq$ *Suc n*› **by** *simp*
hence (λ*w*. *spick w n x*) $-$‘ *C* $\in$ {*geom-proc n* $-$‘ *B* |*B*. *B* $\in$ *sets borel*}

using ‹$C \in \{$*geom-proc m* $-`$ $A \mid A.\ A \in$ *sets borel*$\}$› *geom-spick-Suc* **by** *simp*
also have ... $\subseteq$ *sets* $(G\ n)$
proof −
have $\{$*geom-proc n* $-`$ $B \mid B.\ B \in$ *sets borel*$\} \subseteq \{$*geom-proc n* $-`$ $B \cap$ *space M* $\mid B.\ B \in$ *sets borel*$\}$
using *bernoulli bernoulli-stream-space* **by** *simp*
also have ... $\subseteq$ $(\bigcup i \in \{m.\ m \leq n\}.\ \{$*geom-proc i* $-`$ $A \cap$ *space M* $\mid A.\ A \in$ *sets borel*$\})$
by *auto*
also have ... $\subseteq$ *sigma-sets* *(space M)* $(\bigcup i \in \{m.\ m \leq n\}.\ \{$*geom-proc i* $-`$ $A \cap$ *space M* $\mid A.\ A \in$ *sets borel*$\})$
by (*rule sigma-sets-superset-generator*)
also have ... = *sets* $(G\ n)$ **using** *stock-filtration geometric-process stoch-proc-filt-sets*[*of n geom-proc M borel*] *geom-rand-walk-borel-measurable* **by** *blast*
finally show *?thesis* **.**
qed
finally show *?thesis* **.**
qed
}
show $(\lambda w.$ *spick w n x*$)$ $-`$ $\{\} \in$ *sets* $(G\ n)$ **by** *simp*
{
fix *C*
assume *C* $\in$ *sigma-sets* *(space M)* $(\bigcup i \in \{m.\ m \leq Suc\ n\}.\ \{$*geom-proc i* $-`$ *A* $\cap$ *space M* $\mid A.\ A \in$ *sets borel*$\})$
and $(\lambda w.$ *spick w n x*$)$ $-`$ *C* $\in$ *sets* $(G\ n)$
hence $(\lambda w.$ *spick w n x*$)$ $-`$ *(space M* $-$ *C)* = $(\lambda w.$ *spick w n x*$)$ $-`$ *(space M)* $-$ $(\lambda w.$ *spick w n x*$)$ $-`$ *C*
by (*simp add*: *vimage-Diff*)
also have ... = *space M* $-$ $(\lambda w.$ *spick w n x*$)$ $-`$ *C* **using** *bernoulli bernoulli-stream-space* **by** *simp*
also have ... $\in$ *sets* $(G\ n)$ **using** ‹$(\lambda w.$ *spick w n x*$)$ $-`$ *C* $\in$ *sets* $(G\ n)$›
by (*metis algebra.compl-sets disc-filtr-def discrete-filtration sets.sigma-algebra-axioms sigma-algebra-def subalgebra-def*)
finally show $(\lambda w.$ *spick w n x*$)$ $-`$ *(space M* $-$ *C)* $\in$ *sets* $(G\ n)$ **.**
}
{
fix *C*::*nat* $\Rightarrow$ *bool stream set*
assume $(\bigwedge i.$ *C i* $\in$ *sigma-sets* *(space M)* $(\bigcup i \in \{m.\ m \leq Suc\ n\}.\ \{$*geom-proc i* $-`$ *A* $\cap$ *space M* $\mid A.\ A \in$ *sets borel*$\}))$
and $(\bigwedge i.\ (\lambda w.$ *spick w n x*$)$ $-`$ *C i* $\in$ *sets* $(G\ n))$
hence $(\lambda w.$ *spick w n x*$)$ $-`$ $\bigcup$ *(C* ‘ *UNIV)* = $(\bigcup\ i \in$ *UNIV*$.\ (\lambda w.$ *spick w n x*$)$ $-`$ *(C i))* **by** *blast*
also have ... $\in$ *sets* $(G\ n)$ **using** ‹$\bigwedge i.\ (\lambda w.$ *spick w n x*$)$ $-`$ *C i* $\in$ *sets* $(G\ n)$› **by** *simp*
finally show $(\lambda w.$ *spick w n x*$)$ $-`$ $\bigcup$ *(C* ‘ *UNIV)* $\in$ *sets* $(G\ n)$ **.**
}
qed

thus ($\lambda w.$ *spick w n x*) $-$‘ $B \cap$ *space* (*G n*) $\in$ *sets* (*G n*) **using** *stock-filtration stoch-proc-filt-space*
bernoulli bernoulli-stream-space **by** *simp*
qed
qed

lemma (**in** *CRR-market*) *delta-price-adapted*:
fixes *cash-flow*::*bool stream* $\Rightarrow$ *real*
assumes *cash-flow* $\in$ *borel-measurable* (*G T*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *borel-adapt-stoch-proc G* (*delta-price N cash-flow T*)
unfolding *adapt-stoch-proc-def*
proof
fix *n*
show *delta-price N cash-flow T n* $\in$ *borel-measurable* (*G n*)
proof (*cases Suc n* $\leq$ *T*)
case *True*
hence *deleq*: $\forall$ *w. delta-price N cash-flow T n w* = (*rn-price N cash-flow T* (*Suc n*) (*spick w n True*) $-$ *rn-price N cash-flow T* (*Suc n*) (*spick w n False*))/
((*geom-proc n w*) $*$ (*u* $-$ *d*)) **using** *delta-price-eq* **by** *simp*
have ($\lambda w.$ *rn-price N cash-flow T* (*Suc n*) (*spick w n True*)) $\in$ *borel-measurable* (*G n*)
proof $-$
have *rn-price N cash-flow T* (*Suc n*) $\in$ *borel-measurable* (*G* (*Suc n*)) **using** *rn-price-borel-adapt assms*
using *True* **by** *blast*
moreover have ($\lambda w.$ *spick w n True*) $\in$ *G n* $\rightarrow_M$ *G* (*Suc n*) **using** *spick-red-geom-filt* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
moreover have ($\lambda w.$ *rn-price N cash-flow T* (*Suc n*) (*spick w n False*)) $\in$ *borel-measurable* (*G n*)
proof $-$
have *rn-price N cash-flow T* (*Suc n*) $\in$ *borel-measurable* (*G* (*Suc n*)) **using** *rn-price-borel-adapt assms*
using *True* **by** *blast*
moreover have ($\lambda w.$ *spick w n False*) $\in$ *G n* $\rightarrow_M$ *G* (*Suc n*) **using** *spick-red-geom-filt* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
ultimately have ($\lambda w.$ *rn-price N cash-flow T* (*Suc n*) (*spick w n True*) $-$ *rn-price N cash-flow T* (*Suc n*) (*spick w n False*))
$\in$ *borel-measurable* (*G n*) **by** *simp*
moreover have ($\lambda w.$ (*geom-proc n w*) $*$ (*u* $-$ *d*)) $\in$ *borel-measurable* (*G n*)
proof $-$
have *geom-proc n* $\in$ *borel-measurable* (*G n*) **using** *stock-filtration*
by (*metis adapt-stoch-proc-def stk-price stock-price-borel-measurable*)

thus *?thesis* **by** *simp*
qed
ultimately have (*λw*. (*rn-price N cash-flow T* (*Suc n*) (*spick w n True*) − *rn-price N cash-flow T* (*Suc n*) (*spick w n False*))/
((*geom-proc n w*) ∗ (*u* − *d*)))∈ *borel-measurable* (*G n*) **by** *simp*
thus *?thesis* **using** *deleq* **by** *presburger*
next
case *False*
thus *?thesis* **unfolding** *delta-price-def* **by** *simp*
qed
qed

fun (**in** *CRR-market*) *delta-predict* **where**
delta-predict N der matur 0 = (*λw*. *delta-price N der matur 0 w*) |
delta-predict N der matur (*Suc n*) = (*λw*. *delta-price N der matur n w*)

lemma (**in** *CRR-market*) *delta-predict-predict*:
assumes *der* ∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *borel-predict-stoch-proc G* (*delta-predict N der matur*) **unfolding** *predict-stoch-proc-def*
proof (*intro conjI*)
show *delta-predict N der matur 0* ∈ *borel-measurable* (*G 0*) **using** *delta-price-adapted*[*of der matur N q*]
assms **unfolding** *adapt-stoch-proc-def* **by** *force*
show ∀ *n*. *delta-predict N der matur* (*Suc n*) ∈ *borel-measurable* (*G n*)
proof
fix *n*
show *delta-predict N der matur* (*Suc n*) ∈ *borel-measurable* (*G n*) **using** *delta-price-adapted*[*of der matur N q*]
assms **unfolding** *adapt-stoch-proc-def* **by** *force*
qed
qed

definition (**in** *CRR-market*) *delta-pf* **where**
delta-pf N der matur = *qty-single stk* (*delta-predict N der matur*)

lemma (**in** *CRR-market*) *delta-pf-support*:
shows *support-set* (*delta-pf N der matur*) ⊆ {*stk*} **unfolding** *delta-pf-def*
using *single-comp-support*[*of stk delta-predict N der matur*] **by** *simp*

definition (**in** *CRR-market*) *self-fin-delta-pf* **where**
self-fin-delta-pf N der matur v0 = *self-finance Mkt v0* (*delta-pf N der matur*) *risk-free-asset*

lemma (**in** *disc-equity-market*) *self-finance-trading-strat*:

assumes *trading-strategy pf*
and *portfolio pf*
and *borel-adapt-stoch-proc F* (*prices Mkt asset*)
and *support-adapt Mkt pf*
shows *trading-strategy* (*self-finance Mkt v pf asset*) **unfolding** *self-finance-def*
proof (*rule sum-trading-strat*)
show *trading-strategy pf* **using** *assms* **by** *simp*
show *trading-strategy* (*qty-single asset* (*remaining-qty Mkt v pf asset*)) **unfolding** *trading-strategy-def*
proof (*intro conjI ballI*)
show *portfolio* (*qty-single asset* (*remaining-qty Mkt v pf asset*))
by (*simp add*: *self-finance-def single-comp-portfolio*)
show ⋀*a*.
a ∈ *support-set* (*qty-single asset* (*remaining-qty Mkt v pf asset*)) ⟹
borel-predict-stoch-proc F (*qty-single asset* (*remaining-qty Mkt v pf asset*) *a*)
proof (*cases support-set* (*qty-single asset* (*remaining-qty Mkt v pf asset*)) = {})
case *False*
hence *eqasset*: *support-set* (*qty-single asset* (*remaining-qty Mkt v pf asset*)) = {*asset*}
using *single-comp-support* **by** *fastforce*
fix *a*
assume *a*∈ *support-set* (*qty-single asset* (*remaining-qty Mkt v pf asset*))
hence *a* = *asset* **using** *eqasset* **by** *simp*
hence *qty-single asset* (*remaining-qty Mkt v pf asset*) *a* = (*remaining-qty Mkt v pf asset*)
unfolding *qty-single-def* **by** *simp*
moreover have *borel-predict-stoch-proc F* (*remaining-qty Mkt v pf asset*)
proof (*rule remaining-qty-predict*)
show *trading-strategy pf* **using** *assms* **by** *simp*
show *borel-adapt-stoch-proc F* (*prices Mkt asset*) **using** *assms* **by** *simp*
show *support-adapt Mkt pf* **using** *assms* **by** *simp*
qed
ultimately show *borel-predict-stoch-proc F* (*qty-single asset* (*remaining-qty Mkt v pf asset*) *a*)
by *simp*
next
case *True*
thus ⋀*a*. *a* ∈ *support-set* (*qty-single asset* (*remaining-qty Mkt v pf asset*)) ⟹
support-set (*qty-single asset* (*remaining-qty Mkt v pf asset*)) = {} ⟹
borel-predict-stoch-proc F (*qty-single asset* (*remaining-qty Mkt v pf asset*) *a*) **by** *simp*
qed
qed
qed

lemma (**in** *CRR-market*) *self-fin-delta-pf-trad-strat*:
assumes *der*∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*

and $q < 1$
shows *trading-strategy* (*self-fin-delta-pf N der matur v0*) **unfolding** *self-fin-delta-pf-def*
proof (*rule self-finance-trading-strat*)
show *trading-strategy* (*delta-pf N der matur*) **unfolding** *trading-strategy-def*
proof (*intro conjI ballI*)
show *portfolio* (*delta-pf N der matur*) **unfolding** *portfolio-def* **using** *delta-pf-support*
by (*meson finite.emptyI finite-insert infinite-super*)
show $\bigwedge$*asset*. *asset* ∈ *support-set* (*delta-pf N der matur*) ⟹ *borel-predict-stoch-proc G* (*delta-pf N der matur asset*)
proof (*cases support-set* (*delta-pf N der matur*) = {})
case *False*
fix *asset*
assume *asset* ∈ *support-set* (*delta-pf N der matur*)
hence *asset* = *stk* **using** *False delta-pf-support* **by** *auto*
hence *delta-pf N der matur asset* = *delta-predict N der matur* **unfolding** *delta-pf-def qty-single-def* **by** *simp*
thus *borel-predict-stoch-proc G* (*delta-pf N der matur asset*) **using** *delta-predict-predict assms* **by** *simp*
next
case *True*
thus $\bigwedge$*asset*. *asset* ∈ *support-set* (*delta-pf N der matur*) ⟹ *support-set* (*delta-pf N der matur*) = {} ⟹ *borel-predict-stoch-proc G* (*delta-pf N der matur asset*) **by** *simp*
qed
qed
show *portfolio* (*delta-pf N der matur*) **using** *delta-pf-support* **unfolding** *portfolio-def*
by (*meson finite.emptyI finite-insert infinite-super*)
show *borel-adapt-stoch-proc G* (*prices Mkt risk-free-asset*) **using** *rf-price disc-rfr-proc-borel-adapted* **by** *simp*
show *support-adapt Mkt* (*delta-pf N der matur*) **unfolding** *support-adapt-def*
proof
show $\bigwedge$*asset*. *asset* ∈ *support-set* (*delta-pf N der matur*) ⟹ *borel-adapt-stoch-proc G* (*prices Mkt asset*)
proof (*cases support-set* (*delta-pf N der matur*) = {})
case *False*
fix *asset*
assume *asset* ∈ *support-set* (*delta-pf N der matur*)
hence *asset* = *stk* **using** *False delta-pf-support* **by** *auto*
hence *prices Mkt asset* = *geom-proc* **using** *stk-price* **by** *simp*
thus *borel-adapt-stoch-proc G* (*prices Mkt asset*)
using ‹*asset* = *stk*› *stock-price-borel-measurable* **by** *auto*
next
case *True*
thus $\bigwedge$*asset*. *asset* ∈ *support-set* (*delta-pf N der matur*) ⟹ *borel-adapt-stoch-proc G* (*prices Mkt asset*)
by *simp*
qed
qed

qed

definition (**in** *CRR-market*) *delta-hedging* **where**
delta-hedging N der matur = *self-fin-delta-pf N der matur*
(*prob-space.expectation N* (*discounted-value r* (λ*m. der*) *matur*))

lemma (**in** *CRR-market*) *geom-proc-eq-snth*:
shows (⋀*m. m* ≤ *Suc n* ⟹ *geom-proc m x* = *geom-proc m y*) ⟹
(⋀*m. m* ≤ *n* ⟹ *snth x m* = *snth y m*)
proof (*induct n*)
case *0*
assume *asm*: (⋀*m. m* ≤*Suc 0* ⟹ *geom-proc m x* = *geom-proc m y*) **and** *m*≤ *0*
hence *m* = *0* **by** *simp*
have *geom-proc* (*Suc 0*) *x* = *geom-proc* (*Suc 0*) *y* **using** *asm* **by** *simp*
have *snth x 0* = *snth y 0*
proof (*rule ccontr*)
assume *snth x 0* ≠ *snth y 0*
show *False*
proof (*cases snth x 0*)
case *True*
hence ¬ *snth y 0* **using** ‹*snth x 0* ≠ *snth y 0*› **by** *simp*
have *geom-proc* (*Suc 0*) *x* = *u* ∗ *init* **using** *geometric-process True* **by** *simp*
moreover have *geom-proc* (*Suc 0*) *y* = *d* ∗ *init* **using** *geometric-process* ‹¬ *snth y 0*› **by** *simp*
ultimately have *geom-proc* (*Suc 0*) *x* ≠ *geom-proc* (*Suc 0*) *y* **using** *S0-positive down-lt-up* **by** *simp*
thus *?thesis* **using** ‹*geom-proc* (*Suc 0*) *x* = *geom-proc* (*Suc 0*) *y*› **by** *simp*
next
case *False*
hence *snth y 0* **using** ‹*snth x 0* ≠ *snth y 0*› **by** *simp*
have *geom-proc* (*Suc 0*) *x* = *d* ∗ *init* **using** *geometric-process False* **by** *simp*
moreover have *geom-proc* (*Suc 0*) *y* = *u* ∗ *init* **using** *geometric-process* ‹*snth y 0*› **by** *simp*
ultimately have *geom-proc* (*Suc 0*) *x* ≠ *geom-proc* (*Suc 0*) *y* **using** *S0-positive down-lt-up* **by** *simp*
thus *?thesis* **using** ‹*geom-proc* (*Suc 0*) *x* = *geom-proc* (*Suc 0*) *y*› **by** *simp*
qed
qed
thus ⋀*m.* (⋀*m. m* ≤ *Suc 0* ⟹ *geom-proc m x* = *geom-proc m y*) ⟹ *m* ≤ *0* ⟹ *x* !! *m* = *y* !! *m* **by** *simp*
next
case (*Suc n*)
assume *fst*: (⋀*m.* (⋀*m. m* ≤ *Suc n* ⟹ *geom-proc m x* = *geom-proc m y*) ⟹ *m* ≤ *n* ⟹ *x* !! *m* = *y* !! *m*)
and *scd*: (⋀*m. m* ≤ *Suc* (*Suc n*) ⟹ *geom-proc m x* = *geom-proc m y*) **and** *m* ≤ *Suc n*
show *x* !! *m* = *y* !! *m*

proof (*cases* $m \leq n$)
case *True*
thus *?thesis* **using** *fst scd* **by** *simp*
next
case *False*
hence $m =$ *Suc n* **using** ‹$m \leq$ *Suc n*› **by** *simp*
have *geom-proc* (*Suc* (*Suc n*)) $x =$ *geom-proc* (*Suc* (*Suc n*)) y **using** *scd* **by** *simp*
show *?thesis*
proof (*rule ccontr*)
assume $x \,!!\, m \neq y \,!!\, m$
thus *False*
proof (*cases* $x \,!!\, m$)
case *True*
hence $\neg\ y \,!!\, m$ **using** ‹$x \,!!\, m \neq y \,!!\, m$› **by** *simp*
have *geom-proc* (*Suc* (*Suc n*)) $x = u *$ *geom-proc* (*Suc n*) x **using** *geometric-process True*
‹$m =$ *Suc n*› **by** *simp*
also have ... $= u *$ *geom-proc* (*Suc n*) y **using** *scd* ‹$m =$ *Suc n*› **by** *simp*
finally have *geom-proc* (*Suc* (*Suc n*)) $x = u *$ *geom-proc* (*Suc n*) y **.**
moreover have *geom-proc* (*Suc* (*Suc n*)) $y = d *$ *geom-proc* (*Suc n*) y **using** *geometric-process*
‹$m =$ *Suc n*› ‹$\neg\ y \,!!\, m$› **by** *simp*
ultimately have *geom-proc* (*Suc* (*Suc n*)) $x \neq$ *geom-proc* (*Suc* (*Suc n*)) y
by (*metis S0-positive down-lt-up down-positive geom-rand-walk-strictly-positive less-irrefl mult-cancel-right*)
thus *?thesis* **using** ‹*geom-proc* (*Suc* (*Suc n*)) $x =$ *geom-proc* (*Suc* (*Suc n*)) y› **by** *simp*
next
case *False*
hence $y \,!!\, m$ **using** ‹$x \,!!\, m \neq y \,!!\, m$› **by** *simp*
have *geom-proc* (*Suc* (*Suc n*)) $x = d *$ *geom-proc* (*Suc n*) x **using** *geometric-process False*
‹$m =$ *Suc n*› **by** *simp*
also have ... $= d *$ *geom-proc* (*Suc n*) y **using** *scd* ‹$m =$ *Suc n*› **by** *simp*
finally have *geom-proc* (*Suc* (*Suc n*)) $x = d *$ *geom-proc* (*Suc n*) y **.**
moreover have *geom-proc* (*Suc* (*Suc n*)) $y = u *$ *geom-proc* (*Suc n*) y **using** *geometric-process*
‹$m =$ *Suc n*› ‹$y \,!!\, m$› **by** *simp*
ultimately have *geom-proc* (*Suc* (*Suc n*)) $x \neq$ *geom-proc* (*Suc* (*Suc n*)) y
by (*metis S0-positive down-lt-up down-positive geom-rand-walk-strictly-positive less-irrefl mult-cancel-right*)
thus *?thesis* **using** ‹*geom-proc* (*Suc* (*Suc n*)) $x =$ *geom-proc* (*Suc* (*Suc n*)) y› **by** *simp*
qed
qed
qed
qed

lemma (**in** *CRR-market*) *geom-proc-eq-pseudo-proj-True*:
shows $(\bigwedge m.\ m \leq\ n \Longrightarrow$ *geom-proc m x* = *geom-proc m y*$) \Longrightarrow$
(*pseudo-proj-True* (*n*) *x* = *pseudo-proj-True* (*n*) *y*)
by (*meson geom-proc-eq-snth le-trans order.refl pseudo-proj-True-snth'*)

lemma (**in** *CRR-market*) *proj-stoch-eq-pseudo-proj-True*:
assumes *proj-stoch-proc geom-proc m x* = *proj-stoch-proc geom-proc m y*
shows *pseudo-proj-True m x* = *pseudo-proj-True m y*
proof −
have $\forall\ k \leq m.$ *geom-proc k x* = *geom-proc k y*
proof (*intro allI impI*)
fix *k*
assume $k \leq m$
thus *geom-proc k x* = *geom-proc k y* **using** *proj-stoch-proc-eq-snth*[*of geom-proc m x y k*] *assms* **by** *simp*
qed
thus *?thesis* **using** *geom-proc-eq-pseudo-proj-True*[*of m x y*] **by** *auto*
qed

lemma (**in** *CRR-market-viable*) *rn-rev-price-cond-expect*:
assumes *N* = *bernoulli-stream q*
and *0* <*q*
and *q* < *1*
and *der* ∈ *borel-measurable* (*G matur*)
and *Suc n* ≤ *matur*
shows *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) (*rn-rev-price N der matur* (*matur* − *Suc n*)) *w*=
(*q* ∗ *rn-rev-price N der matur* (*matur* − *Suc n*) (*pseudo-proj-True n w*) +
(*1* − *q*) ∗ *rn-rev-price N der matur* (*matur* − *Suc n*) (*pseudo-proj-False n w*))
proof (*rule infinite-cts-filtration.f-borel-Suc-expl-cond-expect*)
show *infinite-cts-filtration q N nat-filtration* **using** *assms pslt psgt bernoulli-nat-filtration* **by** *simp*
show *rn-rev-price N der matur* (*matur* − *Suc n*) ∈ *borel-measurable* (*nat-filtration* (*Suc n*))
using *rn-rev-price-rev-borel-adapt*[*of der matur N q Suc n*] *assms stock-filtration stoch-proc-subalg-nat-filt*[*of geom-proc*] *geom-rand-walk-borel-adapted*
by (*metis add-diff-cancel-right' diff-le-self measurable-from-subalg ordered-cancel-comm-monoid-diff-class.add-diff-inverse rn-rev-price-rev-borel-adapt*)
show *proj-stoch-proc geom-proc n* ∈ *nat-filtration n* →$_M$ *stream-space borel*
using *proj-stoch-adapted-if-adapted*[*of M nat-filtration geom-proc borel n*] *pslt psgt bernoulli-nat-filtration*[*of M p*] *bernoulli geom-rand-walk-borel-adapted nat-discrete-filtration* **by** *blast*
show *set-discriminating n* (*proj-stoch-proc geom-proc n*) (*stream-space borel*)
using *infinite-cts-filtration.proj-stoch-set-discriminating pslt psgt bernoulli-nat-filtration*[*of M p*] *bernoulli geom-rand-walk-borel-adapted*
by *simp*

show *proj-stoch-proc geom-proc n −‘ {proj-stoch-proc geom-proc n w} ∈ sets (nat-filtration n)*
using *infinite-cts-filtration.proj-stoch-singleton-set*
pslt psgt bernoulli-nat-filtration[of M p] bernoulli geom-rand-walk-borel-adapted
by *simp*
show ∀ *y z. proj-stoch-proc geom-proc n y = proj-stoch-proc geom-proc n z ∧ y !! n = z !! n ⟶*
rn-rev-price N der matur (matur − Suc n) y = rn-rev-price N der matur (matur − Suc n) z
proof (*intro allI impI*)
fix *y z*
assume *as:proj-stoch-proc geom-proc n y = proj-stoch-proc geom-proc n z ∧ y !! n = z !! n*
hence *pseudo-proj-True n y = pseudo-proj-True n z* **using** *proj-stoch-eq-pseudo-proj-True[of n y z]* **by** *simp*
moreover have *snth y n = snth z n* **using** *as* **by** *simp*
ultimately have *pseudo-proj-True (Suc n) y = pseudo-proj-True (Suc n) z*
by (*metis* (*full-types*) *pseudo-proj-True-def pseudo-proj-True-same-img stake-Suc*)
have *rn-rev-price N der matur (matur − Suc n) y =*
rn-rev-price N der matur (matur − Suc n) (pseudo-proj-True (Suc n) y) **using** *nat-filtration-info[of rn-rev-price N der matur (matur − Suc n) Suc n]*
rn-rev-price-rev-borel-adapt[of der matur N q]
by (*metis ‹rn-rev-price N der matur (matur − Suc n) ∈ borel-measurable (nat-filtration (Suc n))› o-apply*)
also have *... = rn-rev-price N der matur (matur − Suc n) (pseudo-proj-True (Suc n) z)*
using *‹pseudo-proj-True (Suc n) y = pseudo-proj-True (Suc n) z›* **by** *simp*
also have *... = rn-rev-price N der matur (matur − Suc n) z* **using** *nat-filtration-info[of rn-rev-price N der matur (matur − Suc n) Suc n]*
rn-rev-price-rev-borel-adapt[of der matur N q]
by (*metis ‹rn-rev-price N der matur (matur − Suc n) ∈ borel-measurable (nat-filtration (Suc n))› o-apply*)
finally show *rn-rev-price N der matur (matur − Suc n) y = rn-rev-price N der matur (matur − Suc n) z* **.**
qed
show *0 < q* **and** *q < 1* **using** *assms* **by** *auto*
qed

lemma (**in** *CRR-market-viable*) *rn-price-eq-ind*:
assumes *N = bernoulli-stream q*
and *n < matur*
and *0 < q*
and *q < 1*
and *der ∈ borel-measurable (G matur)*
shows *(1+r) ∗ rn-price N der matur n w = q ∗ rn-price N der matur (Suc n) (pseudo-proj-True n w) +*

$(1 - q) * rn\text{-}price\ N\ der\ matur\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)$
proof −
define V **where** $V = rn\text{-}price\ N\ der\ matur$
let $?m = matur - Suc\ n$
have $matur - n = Suc\ ?m$ **by** (*simp add*: *assms Suc-diff-Suc Suc-le-lessD*)
have $(1+r) * V\ n\ w = (1+r) * rn\text{-}price\text{-}ind\ N\ der\ matur\ n\ w$ **using** *rn-price-eq assms* **unfolding** *V-def* **by** *simp*
also have ... $= (1+r) * rn\text{-}rev\text{-}price\ N\ der\ matur\ (Suc\ ?m)\ w$ **using** ‹$matur - n = Suc\ ?m$›
unfolding *rn-price-ind-def* **by** *simp*
also have ... $= (1+r) * discount\text{-}factor\ r\ (Suc\ 0)\ w *$
$expl\text{-}cond\text{-}expect\ N\ (proj\text{-}stoch\text{-}proc\ geom\text{-}proc\ (matur - Suc\ ?m))$
$(rn\text{-}rev\text{-}price\ N\ der\ matur\ ?m)\ w$
by *simp*
also have ... $= expl\text{-}cond\text{-}expect\ N\ (proj\text{-}stoch\text{-}proc\ geom\text{-}proc\ (matur - Suc\ ?m))\ (rn\text{-}rev\text{-}price\ N\ der\ matur\ ?m)\ w$
unfolding *discount-factor-def* **using** *acceptable-rate* **by** *auto*
also have ... $= expl\text{-}cond\text{-}expect\ N\ (proj\text{-}stoch\text{-}proc\ geom\text{-}proc\ n)\ (rn\text{-}rev\text{-}price\ N\ der\ matur\ ?m)\ w$
using ‹$matur - n = Suc\ ?m$› **by** *simp*
also have ... $= (q * rn\text{-}rev\text{-}price\ N\ der\ matur\ ?m\ (pseudo\text{-}proj\text{-}True\ n\ w)\ +$
$(1 - q) * rn\text{-}rev\text{-}price\ N\ der\ matur\ ?m\ (pseudo\text{-}proj\text{-}False\ n\ w))$
using *rn-rev-price-cond-expect*[*of N q der matur n w*] *assms* **by** *simp*
also have ... $= q * rn\text{-}price\text{-}ind\ N\ der\ matur\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ n\ w) +$
$(1 - q) * rn\text{-}price\text{-}ind\ N\ der\ matur\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)$ **unfolding** *rn-price-ind-def* **by** *simp*
also have ... $= q * rn\text{-}price\ N\ der\ matur\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ n\ w) +$
$(1 - q) * rn\text{-}price\ N\ der\ matur\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)$ **using** *rn-price-eq assms* **by** *simp*
also have ... $= q * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ n\ w) + (1 - q) * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)$
unfolding *V-def* **by** *simp*
finally have $(1+r) * V\ n\ w = q * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ n\ w) + (1 - q) * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)$ **.**
thus *?thesis* **unfolding** *V-def* **by** *simp*
qed

lemma *self-finance-updated-suc-suc*:
assumes *portfolio pf*
and $\forall n.\ prices\ Mkt\ asset\ n\ w \neq 0$
shows $cls\text{-}val\text{-}process\ Mkt\ (self\text{-}finance\ Mkt\ v\ pf\ asset)\ (Suc\ (Suc\ n))\ w =$
$cls\text{-}val\text{-}process\ Mkt\ pf\ (Suc\ (Suc\ n))\ w +$
$(prices\ Mkt\ asset\ (Suc\ (Suc\ n))\ w\ /\ (prices\ Mkt\ asset\ (Suc\ n)\ w)) *$
$(cls\text{-}val\text{-}process\ Mkt\ (self\text{-}finance\ Mkt\ v\ pf\ asset)\ (Suc\ n)\ w -$
$val\text{-}process\ Mkt\ pf\ (Suc\ n)\ w)$
proof −
have $cls\text{-}val\text{-}process\ Mkt\ (self\text{-}finance\ Mkt\ v\ pf\ asset)\ (Suc\ (Suc\ n))\ w = cls\text{-}val\text{-}process$

Mkt pf (Suc (Suc n)) w +
*prices Mkt asset (Suc (Suc n)) w * remaining-qty Mkt v pf asset (Suc (Suc n)) w* **using** *assms*
by (*simp add*: *self-finance-updated*)
also have ... = *cls-val-process Mkt pf (Suc (Suc n)) w +*
*prices Mkt asset (Suc (Suc n)) w * ((remaining-qty Mkt v pf asset (Suc n) w) +*
(cls-val-process Mkt pf (Suc n) w − val-process Mkt pf (Suc n) w)/(prices Mkt asset (Suc n) w))
by *simp*
also have ... = *cls-val-process Mkt pf (Suc (Suc n)) w +*
*prices Mkt asset (Suc (Suc n)) w **
*((prices Mkt asset (Suc n) w) * (remaining-qty Mkt v pf asset (Suc n) w) / (prices Mkt asset (Suc n) w) +*
(cls-val-process Mkt pf (Suc n) w − val-process Mkt pf (Suc n) w)/(prices Mkt asset (Suc n) w)) **using** *assms*
by (*metis nonzero-mult-div-cancel-left*)
also have ... = *cls-val-process Mkt pf (Suc (Suc n)) w +*
*prices Mkt asset (Suc (Suc n)) w * ((prices Mkt asset (Suc n) w) * (remaining-qty Mkt v pf asset (Suc n) w) +*
cls-val-process Mkt pf (Suc n) w − val-process Mkt pf (Suc n) w)/(prices Mkt asset (Suc n) w)
using *add-divide-distrib*[*symmetric*, *of prices Mkt asset (Suc n) w * remaining-qty Mkt v pf asset (Suc n) w*
prices Mkt asset (Suc n) w] **by** *simp*
also have ... = *cls-val-process Mkt pf (Suc (Suc n)) w +*
*(prices Mkt asset (Suc (Suc n)) w / (prices Mkt asset (Suc n) w)) **
*((prices Mkt asset (Suc n) w) * (remaining-qty Mkt v pf asset (Suc n) w) +*
cls-val-process Mkt pf (Suc n) w − val-process Mkt pf (Suc n) w **by** *simp*
also have ... = *cls-val-process Mkt pf (Suc (Suc n)) w +*
*(prices Mkt asset (Suc (Suc n)) w / (prices Mkt asset (Suc n) w)) **
(cls-val-process Mkt (self-finance Mkt v pf asset) (Suc n) w −
val-process Mkt pf (Suc n) w)
using *self-finance-updated*[*of Mkt asset n w pf v*] *assms* **by** *auto*
finally show *?thesis* **.**
qed

lemma *self-finance-updated-suc-0*:
assumes *portfolio pf*
and ∀ *n w*. *prices Mkt asset n w* ≠ *0*
shows *cls-val-process Mkt (self-finance Mkt v pf asset) (Suc 0) w = cls-val-process Mkt pf (Suc 0) w +*
*(prices Mkt asset (Suc 0) w / (prices Mkt asset 0 w)) **
(val-process Mkt (self-finance Mkt v pf asset) 0 w −
val-process Mkt pf 0 w)
proof −
have *cls-val-process Mkt (self-finance Mkt v pf asset) (Suc 0) w = cls-val-process Mkt pf (Suc 0) w +*
*prices Mkt asset (Suc 0) w * remaining-qty Mkt v pf asset (Suc 0) w* **using**

assms
by (*simp add*: *self-finance-updated*)
also have ... = *cls-val-process Mkt pf* (*Suc 0*) *w* +
prices Mkt asset (*Suc 0*) *w* ∗ ((*v* − *val-process Mkt pf 0 w*)/(*prices Mkt asset 0 w*))
by *simp*
also have ... = *cls-val-process Mkt pf* (*Suc 0*) *w* +
prices Mkt asset (*Suc 0*) *w* ∗ ((*remaining-qty Mkt v pf asset 0 w*) +
(*v* − *val-process Mkt pf 0 w*)/(*prices Mkt asset 0 w*))
by *simp*
also have ... = *cls-val-process Mkt pf* (*Suc 0*) *w* +
prices Mkt asset (*Suc 0*) *w* ∗
((*prices Mkt asset 0 w*) ∗ (*remaining-qty Mkt v pf asset 0 w*) / (*prices Mkt asset 0 w*) +
(*v* − *val-process Mkt pf 0 w*)/(*prices Mkt asset 0 w*)) **using** *assms*
by (*metis nonzero-mult-div-cancel-left*)
also have ... = *cls-val-process Mkt pf* (*Suc 0*) *w* +
prices Mkt asset (*Suc 0*) *w* ∗ ((*prices Mkt asset 0 w*) ∗ (*remaining-qty Mkt v pf asset 0 w*) +
v − *val-process Mkt pf 0 w*)/(*prices Mkt asset 0 w*)
using *add-divide-distrib*[*symmetric*, *of prices Mkt asset 0 w* ∗ *remaining-qty Mkt v pf asset 0 w*
prices Mkt asset 0 w] **by** *simp*
also have ... = *cls-val-process Mkt pf* (*Suc 0*) *w* +
(*prices Mkt asset* (*Suc 0*) *w* / (*prices Mkt asset 0 w*)) ∗
((*prices Mkt asset 0 w*) ∗ (*remaining-qty Mkt v pf asset 0 w*) +
v − *val-process Mkt pf 0 w*) **by** *simp*
also have ... = *cls-val-process Mkt pf* (*Suc 0*) *w* +
(*prices Mkt asset* (*Suc 0*) *w* / (*prices Mkt asset 0 w*)) ∗
((*prices Mkt asset 0 w*) ∗ (*remaining-qty Mkt v pf asset 0 w*) +
val-process Mkt (*self-finance Mkt v pf asset*) *0 w* − *val-process Mkt pf 0 w*)
using *self-finance-init*[*of Mkt asset pf v w*] *assms* **by** *simp*
also have ... = *cls-val-process Mkt pf* (*Suc 0*) *w* +
(*prices Mkt asset* (*Suc 0*) *w* / (*prices Mkt asset 0 w*)) ∗
(*val-process Mkt* (*self-finance Mkt v pf asset*) *0 w* −
val-process Mkt pf 0 w) **by** *simp*
finally show *?thesis* **.**
qed

lemma *self-finance-updated-ind*:
assumes *portfolio pf*
and ∀ *n w*. *prices Mkt asset n w* ≠ *0*
shows *cls-val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* = *cls-val-process Mkt pf* (*Suc n*) *w* +
(*prices Mkt asset* (*Suc n*) *w* / (*prices Mkt asset n w*)) ∗
(*val-process Mkt* (*self-finance Mkt v pf asset*) *n w* −
val-process Mkt pf n w)
proof (*cases n* = *0*)
case *True*

thus *?thesis* **using** *assms self-finance-updated-suc-0* **by** *simp*
next
case *False*
hence $\exists m.\ n = Suc\ m$ **by** (*simp add*: *not0-implies-Suc*)
from *this* **obtain** *m* **where** $n = Suc\ m$ **by** *auto*
hence *cls-val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* =
cls-val-process Mkt (*self-finance Mkt v pf asset*) (*Suc* (*Suc m*)) *w* **by** *simp*
also have ... = *cls-val-process Mkt pf* (*Suc* (*Suc m*)) *w* +
(*prices Mkt asset* (*Suc* (*Suc m*)) *w* / (*prices Mkt asset* (*Suc m*) *w*)) ∗
(*cls-val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc m*) *w* −
val-process Mkt pf (*Suc m*) *w*) **using** *assms self-finance-updated-suc-suc*[*of pf*]
by *simp*
also have ... = *cls-val-process Mkt pf* (*Suc* (*Suc m*)) *w* +
(*prices Mkt asset* (*Suc* (*Suc m*)) *w* / (*prices Mkt asset* (*Suc m*) *w*)) ∗
(*val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc m*) *w* −
val-process Mkt pf (*Suc m*) *w*) **using** *assms self-finance-charact* **unfolding**
self-financing-def
by (*simp add*: *self-finance-succ self-finance-updated*)
also have ... = *cls-val-process Mkt pf* (*Suc n*) *w* +
(*prices Mkt asset* (*Suc n*) *w* / (*prices Mkt asset n w*)) ∗
(*val-process Mkt* (*self-finance Mkt v pf asset*) *n w* −
val-process Mkt pf n w) **using** ‹*n* = *Suc m*› **by** *simp*
finally show *?thesis* **.**
qed

lemma (**in** *rfr-disc-equity-market*) *self-finance-risk-free-update-ind*:
assumes *portfolio pf*
shows *cls-val-process Mkt* (*self-finance Mkt v pf risk-free-asset*) (*Suc n*) *w* =
cls-val-process Mkt pf (*Suc n*) *w* +
(*1* + *r*) ∗ (*val-process Mkt* (*self-finance Mkt v pf risk-free-asset*) *n w* − *val-process*
Mkt pf n w)
proof −
have *cls-val-process Mkt* (*self-finance Mkt v pf risk-free-asset*) (*Suc n*) *w* =
cls-val-process Mkt pf (*Suc n*) *w* +
(*prices Mkt risk-free-asset* (*Suc n*) *w* / (*prices Mkt risk-free-asset n w*)) ∗
(*val-process Mkt* (*self-finance Mkt v pf risk-free-asset*) *n w* −
val-process Mkt pf n w)
proof (*rule self-finance-updated-ind*, (*simp add*: *assms*), *intro allI*)
fix *n w*
show *prices Mkt risk-free-asset n w* ≠ *0* **using** *positive* **by** (*metis less-irrefl*)
qed
also have ... = *cls-val-process Mkt pf* (*Suc n*) *w* +
(*1*+*r*) ∗ (*val-process Mkt* (*self-finance Mkt v pf risk-free-asset*) *n w* −
val-process Mkt pf n w) **using** *rf-price positive*
by (*metis acceptable-rate disc-rfr-proc-Suc-div*)
finally show *?thesis* **.**
qed

lemma (**in** *CRR-market*) *delta-pf-portfolio*:
shows *portfolio* (*delta-pf N der matur*) **unfolding** *delta-pf-def* **by** (*simp add*: *single-comp-portfolio*)

lemma (**in** *CRR-market*) *delta-pf-updated*:
shows *cls-val-process Mkt* (*delta-pf N der matur*) (*Suc n*) *w* =
geom-proc (*Suc n*) *w* ∗ *delta-price N der matur n w* **unfolding** *delta-pf-def*
using *stk-price qty-single-updated*[*of Mkt*] **by** *simp*

lemma (**in** *CRR-market*) *delta-pf-val-process*:
shows *val-process Mkt* (*delta-pf N der matur*) *n w* =
geom-proc n w ∗ *delta-price N der matur n w* **unfolding** *delta-pf-def*
using *stk-price qty-single-val-process*[*of Mkt*] **by** *simp*

lemma (**in** *CRR-market*) *delta-hedging-cls-val-process*:
shows *cls-val-process Mkt* (*delta-hedging N der matur*) (*Suc n*) *w* =
geom-proc (*Suc n*) *w* ∗ *delta-price N der matur n w* +
(*1* + *r*) ∗ (*val-process Mkt* (*delta-hedging N der matur*) *n w* − *geom-proc n w* ∗ *delta-price N der matur n w*)
proof −
define *X* **where** *X* = *delta-hedging N der matur*
define *init* **where** *init* = *integral*^{L} *N* (*discounted-value r* (λ*m. der*) *matur*)
have *cls-val-process Mkt X* (*Suc n*) *w* = *cls-val-process Mkt* (*delta-pf N der matur*) (*Suc n*) *w* +
(*1* + *r*) ∗ (*val-process Mkt X n w* − *val-process Mkt* (*delta-pf N der matur*) *n w*)
unfolding *X-def delta-hedging-def self-fin-delta-pf-def init-def*
proof (*rule self-finance-risk-free-update-ind*)
show *portfolio* (*delta-pf N der matur*) **unfolding** *portfolio-def* **using** *delta-pf-support*
by (*meson finite.simps infinite-super*)
qed
also have ... = *geom-proc* (*Suc n*) *w* ∗ *delta-price N der matur n w* +
(*1* + *r*) ∗ (*val-process Mkt X n w* − *val-process Mkt* (*delta-pf N der matur*) *n w*)
using *delta-pf-updated* **by** *simp*
also have ... = *geom-proc* (*Suc n*) *w* ∗ *delta-price N der matur n w* +
(*1* + *r*) ∗ (*val-process Mkt X n w* − *geom-proc n w* ∗ *delta-price N der matur n w*)
using *delta-pf-val-process* **by** *simp*
finally show *?thesis* **unfolding** *X-def* **.**
qed

lemma (**in** *CRR-market-viable*) *delta-hedging-eq-derivative-price*:
fixes *der*::*bool stream* ⇒ *real* **and** *matur*::*nat*
assumes *N* = *bernoulli-stream* ((*1* + *r* − *d*) / (*u* − *d*))
and *der*∈ *borel-measurable* (*G matur*)
shows ⋀*n w*. *n*≤ *matur* ⟹
val-process Mkt (*delta-hedging N der matur*) *n w* =
(*rn-price N der matur*) *n w*
unfolding *delta-hedging-def*
proof −
define *q* **where** *q* = (*1* + *r* − *d*) / (*u* − *d*)
have *0* < *q* **and** *q* < *1* **unfolding** *q-def* **using** *assms gt-param lt-param CRR-viable*
by *auto*
note *qprops* = *this*
define *init* **where** *init* = (*prob-space.expectation N* (*discounted-value r* (λ*m*. *der*) *matur*))
define *X* **where** *X* = *val-process Mkt* (*delta-hedging N der matur*)
define *V* **where** *V* = *rn-price N der matur*
define Δ **where** Δ = *delta-price N der matur*
{
fix *n*
fix *w*
have *n* ≤ *matur* ⟹ *X n w* = *V n w*
proof (*induct n*)
case *0*
have *v0*: *V 0* ∈ *borel-measurable* (*G 0*) **using** *assms rn-price-borel-adapt 0.prems qprops*
unfolding *V-def q-def* **by** *auto*
have *X 0 w*= *init* **using** *self-finance-init*[*of Mkt risk-free-asset delta-pf N der matur integral*^{L} *N* (*discounted-value r* (λ*m*. *der*) *matur*)]
delta-pf-support
unfolding *X-def init-def delta-hedging-def self-fin-delta-pf-def init-def*
by (*metis finite-insert infinite-imp-nonempty infinite-super less-irrefl portfolio-def positive*)
also have ... = *V 0 w*
proof −
have ∀ *x*∈*space N*. *real-cond-exp N* (*G 0*) (*discounted-value r* (λ*m*. *der*) *matur*) *x* =
integral^L *N* (*discounted-value r* (λ*m*. *der*) *matur*)
proof (*rule prob-space.trivial-subalg-cond-expect-eq*)
show *prob-space N* **using** *assms qprops* **unfolding** *q-def*
by (*simp add*: *bernoulli bernoulli-stream-def prob-space.prob-space-stream-space prob-space-measure-pmf*)
have *init-triv-filt M* (*stoch-proc-filt M geom-proc borel*)
proof (*rule infinite-cts-filtration.stoch-proc-filt-triv-init*)
show *borel-adapt-stoch-proc nat-filtration geom-proc* **using** *geom-rand-walk-borel-adapted*
by *simp*
show *infinite-cts-filtration p M nat-filtration* **using** *bernoulli-nat-filtration*[*of M p*] *bernoulli psgt pslt*

```
        by simp
      qed
      hence init-triv-filt N (stoch-proc-filt M geom-proc borel) using assms qprops
        filt-equiv-triv-init[of nat-filtration N] stock-filtration
        bernoulli-stream-equiv[of N] psgt pslt unfolding q-def by simp
    thus subalgebra N (G 0) and sets (G 0) = {{}, space N} using stock-filtration
unfolding init-triv-filt-def
        filtration-def bot-nat-def by auto
      show integrable N (discounted-value r (λm. der) matur)
      proof (rule bernoulli-discounted-integrable)
      show der ∈ borel-measurable (nat-filtration matur) using assms geom-rand-walk-borel-adapted
          measurable-from-subalg stoch-proc-subalg-nat-filt stock-filtration by blast
        show N = bernoulli-stream q using assms unfolding q-def by simp
        show 0 < q q < 1 using qprops by auto
      qed (simp add: acceptable-rate)
    qed
    hence integral^L N (discounted-value r (λm. der) matur) =
        real-cond-exp N (G 0) (discounted-value r (λm. der) matur) w using
bernoulli-stream-space[of N q]
      by (simp add: assms(1) q-def)
  also have ... = real-cond-exp N (stoch-proc-filt M geom-proc borel 0) (discounted-value
r (λm. der) matur) w
      using stock-filtration by simp
  also have ... = real-cond-exp N (stoch-proc-filt N geom-proc borel 0) (discounted-value
r (λm. der) matur) w
      using stoch-proc-filt-filt-equiv[of nat-filtration M N geom-proc]
        bernoulli-stream-equiv[of N] q-def qprops assms pslt psgt by auto
  also have ... = expl-cond-expect N (proj-stoch-proc geom-proc 0) (discounted-value
r (λm. der) matur) w
    proof (rule bernoulli-cond-exp)
      show N = bernoulli-stream q using assms unfolding q-def by simp
      show 0 < q q < 1 using qprops by auto
      show integrable N (discounted-value r (λm. der) matur)
      proof (rule bernoulli-discounted-integrable)
      show der ∈ borel-measurable (nat-filtration matur) using assms geom-rand-walk-borel-adapted
          measurable-from-subalg stoch-proc-subalg-nat-filt stock-filtration by blast
        show N = bernoulli-stream q using assms unfolding q-def by simp
        show 0 < q q < 1 using qprops by auto
      qed (simp add: acceptable-rate)
    qed
    finally show init = V 0 w unfolding init-def V-def rn-price-def by simp
  qed
  finally show X 0 w = V 0 w .
  next
    case (Suc n)
    hence n < matur by simp
    show ?case
    proof −
     have X n w = V n w using Suc by (simp add: Suc.hyps Suc.prems Suc-leD)
```

have *0< 1+r* **using** *acceptable-rate* **by** *simp*
let *?m = matur − Suc n*
have *matur −n = Suc ?m* **by** (*simp add*: *Suc.prems Suc-diff-Suc Suc-le-lessD*)
have *(1+r) ∗ V n w = q ∗ V (Suc n) (pseudo-proj-True n w) + (1 − q) ∗V (Suc n) (pseudo-proj-False n w)*
using *rn-price-eq-ind qprops assms Suc q-def V-def* **by** *simp*
show *X (Suc n) w = V (Suc n) w*
proof (*cases snth w n*)
case *True*
hence *pseq*: *pseudo-proj-True (Suc n) w = pseudo-proj-True (Suc n) (spick w n True)*
by (*metis* (*mono-tags*, *lifting*) *pseudo-proj-True-stake-image spickI stake-Suc*)
have *X (Suc n) w = cls-val-process Mkt (delta-hedging N der matur) (Suc n) w*
unfolding *X-def delta-hedging-def self-fin-delta-pf-def* **using** *delta-pf-portfolio*
unfolding *self-financing-def*
by (*metis less-irrefl positive self-finance-charact self-financingE*)
also have *... = geom-proc (Suc n) w ∗ Δ n w + (1 + r) ∗ (X n w − geom-proc n w ∗ Δ n w)*
using *delta-hedging-cls-val-process* **unfolding** *X-def Δ-def* **by** *simp*
also have *... = u ∗ geom-proc n w ∗ Δ n w + (1 + r) ∗ (X n w − geom-proc n w ∗ Δ n w)*
using *True geometric-process* **by** *simp*
also have *... = u ∗ geom-proc n w ∗ Δ n w + (1 + r) ∗ X n w − (1+r) ∗ geom-proc n w ∗ Δ n w*
by (*simp add*: *right-diff-distrib*)
also have *... = (1+r) ∗ X n w + geom-proc n w ∗ Δ n w ∗ u − geom-proc n w ∗ Δ n w ∗ (1 + r)*
by (*simp add*: *mult.commute mult.left-commute*)
also have *... = (1+r)∗ X n w + geom-proc n w ∗ Δ n w ∗ (u − (1 + r))* **by** (*simp add*: *right-diff-distrib*)
also have *... = (1+r) ∗ X n w + geom-proc n w ∗ (V (Suc n) (pseudo-proj-True n w) − V (Suc n) (pseudo-proj-False n w))/*
(geom-proc (Suc n) (spick w n True) − geom-proc (Suc n) (spick w n False)) ∗ (u − (1 + r))
using *Suc V-def* **by** (*simp add*: *Δ-def delta-price-def geom-rand-walk-diff-induct*)
also have *... = (1+r) ∗ X n w + geom-proc n w ∗ ((V (Suc n) (pseudo-proj-True n w) − V (Suc n) (pseudo-proj-False n w))) /*
(geom-proc n w ∗ (u − d)) ∗ (u − (1 + r))
proof −
have *geom-proc (Suc n) (spick w n True) − geom-proc (Suc n) (spick w n False) =*
geom-proc n w ∗ (u − d)
by (*simp add*: *geom-rand-walk-diff-induct*)
then show *?thesis* **by** *simp*
qed
also have *... = (1+r) ∗ X n w + ((V (Suc n) (pseudo-proj-True n w) − V (Suc n) (pseudo-proj-False n w)))∗ (u − (1 + r))/ (u−d)*

proof −
have *geom-proc n w* ≠ *0*
by (*metis S0-positive down-lt-up down-positive geom-rand-walk-strictly-positive less-irrefl*)
then show *?thesis*
by *simp*
qed
also have ... = (*1*+*r*) ∗ *X n w* + ((*V* (*Suc n*) (*pseudo-proj-True n w*) − *V* (*Suc n*) (*pseudo-proj-False n w*))∗ (*1* − *q*))
proof −
have *1* − *q* = *1* − (*1* + *r* − *d*)/(*u* −*d*) **unfolding** *q-def* **by** *simp*
also have ... = (*u* − *d*)/(*u* − *d*) − (*1* + *r* − *d*)/(*u* −*d*) **using** *down-lt-up* **by** *simp*
also have ... = (*u* − *d* − (*1* + *r* − *d*))/(*u* − *d*) **using** *diff-divide-distrib*[*of u* − *d 1* + *r* −*d*] **by** *simp*
also have ... = (*u* − (*1*+*r*))/(*u*−*d*) **by** *simp*
finally have *1* − *q* = (*u* − (*1*+*r*))/(*u*−*d*) **.**
thus *?thesis* **by** *simp*
qed
also have ... = (*1*+*r*) ∗ *X n w* + (*1* − *q*) ∗ *V* (*Suc n*) (*pseudo-proj-True n w*) −
(*1* − *q*) ∗ *V* (*Suc n*) (*pseudo-proj-False n w*)
by (*simp add*: *mult.commute right-diff-distrib*)
also have ... = (*1*+*r*) ∗ *V n w* + (*1* − *q*) ∗ *V* (*Suc n*) (*pseudo-proj-True n w*) −
(*1* − *q*) ∗ *V* (*Suc n*) (*pseudo-proj-False n w*) **using** ‹*X n w* = *V n w*› **by** *simp*
also have ... = *q* ∗ *V* (*Suc n*) (*pseudo-proj-True n w*) + (*1* − *q*) ∗ *V* (*Suc n*) (*pseudo-proj-False n w*) +
(*1* − *q*) ∗ *V* (*Suc n*) (*pseudo-proj-True n w*) − (*1* − *q*) ∗ *V* (*Suc n*) (*pseudo-proj-False n w*)
using *assms Suc rn-price-eq-ind*[*of N q n matur der w*] ‹*n* < *matur*› *qprops* **unfolding** *V-def q-def*
by *simp*
also have ... = *q* ∗ *V* (*Suc n*) (*pseudo-proj-True n w*) + (*1* − *q*) ∗ *V* (*Suc n*) (*pseudo-proj-True n w*) **by** *simp*
also have ... = *V* (*Suc n*) (*pseudo-proj-True n w*)
using *distrib-right*[*of q 1* − *q V* (*Suc n*) (*pseudo-proj-True n w*)] **by** *simp*
also have ... = *V* (*Suc n*) *w*
proof −
have *V* (*Suc n*) ∈ *borel-measurable* (*G* (*Suc n*)) **unfolding** *V-def q-def*
proof (*rule rn-price-borel-adapt*)
show *der* ∈ *borel-measurable* (*G matur*) **using** *assms* **by** *simp*
show *N* = *bernoulli-stream q* **using** *assms* **unfolding** *q-def* **by** *simp*
show *0* < *q* **and** *q* < *1* **using** *qprops* **by** *auto*
show *Suc n* ≤ *matur* **using** *Suc* **by** *simp*
qed
hence *V* (*Suc n*) (*pseudo-proj-True n w*) = *V* (*Suc n*) (*pseudo-proj-True*

(*Suc n*) (*pseudo-proj-True n w*))
using *geom-proc-filt-info*[*of V* (*Suc n*) *Suc n*] **by** *simp*
also have ... = *V* (*Suc n*) (*pseudo-proj-True* (*Suc n*) *w*) **using** *True*
by (*simp add*: *pseq spick-eq-pseudo-proj-True*)
also have ... = *V* (*Suc n*) *w* **using** ‹*V* (*Suc n*) ∈ *borel-measurable* (*G* (*Suc n*))›
geom-proc-filt-info[*of V* (*Suc n*) *Suc n*] **by** *simp*
finally show *?thesis* **.**
qed
finally show *X* (*Suc n*) *w* = *V* (*Suc n*) *w* **.**
next
case *False*
hence *pseq*: *pseudo-proj-True* (*Suc n*) *w* = *pseudo-proj-True* (*Suc n*) (*spick w n False*) **using** *filtration*
by (*metis* (*full-types*) *pseudo-proj-True-def spickI stake-Suc*)
have *X* (*Suc n*) *w* = *cls-val-process Mkt* (*delta-hedging N der matur*) (*Suc n*) *w*
unfolding *X-def delta-hedging-def self-fin-delta-pf-def* **using** *delta-pf-portfolio*
unfolding *self-financing-def*
by (*metis less-irrefl positive self-finance-charact self-financingE*)
also have ... = *geom-proc* (*Suc n*) *w* ∗ Δ *n w* + (*1* + *r*) ∗ (*X n w* − *geom-proc n w* ∗ Δ *n w*)
using *delta-hedging-cls-val-process* **unfolding** *X-def* Δ*-def* **by** *simp*
also have ... = *d* ∗ *geom-proc n w* ∗ Δ *n w* + (*1* + *r*) ∗ (*X n w* − *geom-proc n w* ∗ Δ *n w*)
using *False geometric-process* **by** *simp*
also have ... = *d* ∗ *geom-proc n w* ∗ Δ *n w* + (*1* + *r*) ∗ *X n w* − (*1*+*r*) ∗ *geom-proc n w* ∗ Δ *n w*
by (*simp add*: *right-diff-distrib*)
also have ... = (*1*+*r*) ∗ *X n w* + *geom-proc n w* ∗ Δ *n w* ∗ *d* − *geom-proc n w* ∗ Δ *n w* ∗ (*1* + *r*)
by (*simp add*: *mult.commute mult.left-commute*)
also have ... = (*1*+*r*)∗ *X n w* + *geom-proc n w* ∗ Δ *n w* ∗ (*d* − (*1* + *r*)) **by** (*simp add*: *right-diff-distrib*)
also have ... = (*1*+*r*) ∗ *X n w* + *geom-proc n w* ∗ (*V* (*Suc n*) (*pseudo-proj-True n w*) − *V* (*Suc n*) (*pseudo-proj-False n w*))/
(*geom-proc* (*Suc n*) (*spick w n True*) − *geom-proc* (*Suc n*) (*spick w n False*)) ∗ (*d* − (*1* + *r*))
using *Suc V-def* **by** (*simp add*: Δ*-def delta-price-def geom-rand-walk-diff-induct*)
also have ... = (*1*+*r*) ∗ *X n w* + *geom-proc n w* ∗ ((*V* (*Suc n*) (*pseudo-proj-True n w*) − *V* (*Suc n*) (*pseudo-proj-False n w*))) /
(*geom-proc n w* ∗ (*u* − *d*)) ∗ (*d* − (*1* + *r*))
by (*simp add*: *geom-rand-walk-diff-induct*)
also have ... = (*1*+*r*) ∗ *X n w* + ((*V* (*Suc n*) (*pseudo-proj-True n w*) − *V* (*Suc n*) (*pseudo-proj-False n w*)))∗ (*d* − (*1* + *r*))/ (*u*−*d*)
proof −
have *geom-proc n w* ≠ *0*
by (*metis S0-positive down-lt-up down-positive geom-rand-walk-strictly-positive less-irrefl*)

then show *?thesis*
by *simp*
qed
also have ... = $(1+r) * X\ n\ w + ((V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ n\ w) - V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)) * (-q))$
proof −
have $0-q = 0-(1 + r - d)/(u - d)$ **unfolding** *q-def* **by** *simp*
also have ... = $(d - (1 + r))/(u - d)$ **by** (*simp add*: *minus-divide-left*)
finally have $0 - q = (d - (1+r))/(u-d)$ **.**
thus *?thesis* **by** *simp*
qed
also have ... = $(1+r) * X\ n\ w + (- V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ n\ w) * q + V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w) * q)$
by (*metis* (*no-types*, *opaque-lifting*) *add.inverse-inverse distrib-right minus-mult-commute minus-real-def mult-minus-left*)
also have ... = $(1+r) * X\ n\ w - q * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ n\ w) + q * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)$ **by** *simp*
also have ... = $(1+r) * V\ n\ w - q * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ n\ w) + q * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)$ **using** ‹$X\ n\ w = V\ n\ w$› **by** *simp*
also have ... = $q * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ n\ w) + (1 - q) * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w) - q * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}True\ n\ w) + q * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)$
using *assms Suc rn-price-eq-ind*[*of N q n matur der w*] ‹$n < matur$› *qprops* **unfolding** *V-def q-def*
by *simp*
also have ... = $(1-q) * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w) + q * V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)$ **by** *simp*
also have ... = $V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w)$
using *distrib-right*[*of q 1 − q V* (*Suc n*) (*pseudo-proj-False n w*)] **by** *simp*
also have ... = $V\ (Suc\ n)\ w$
proof −
have $V\ (Suc\ n) \in borel\text{-}measurable\ (G\ (Suc\ n))$ **unfolding** *V-def q-def*
proof (*rule rn-price-borel-adapt*)
show $der \in borel\text{-}measurable\ (G\ matur)$ **using** *assms* **by** *simp*
show $N = bernoulli\text{-}stream\ q$ **using** *assms* **unfolding** *q-def* **by** *simp*
show $0 < q$ **and** $q < 1$ **using** *qprops* **by** *auto*
show $Suc\ n \leq matur$ **using** *Suc* **by** *simp*
qed
hence $V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w) = V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ n\ w))$
using *geom-proc-filt-info′*[*of V* (*Suc n*) *Suc n*] **by** *simp*
also have ... = $V\ (Suc\ n)\ (pseudo\text{-}proj\text{-}False\ (Suc\ n)\ w)$ **using** *False spick-eq-pseudo-proj-False*
by (*metis pseq pseudo-proj-True-imp-False*)
also have ... = $V\ (Suc\ n)\ w$ **using** ‹$V\ (Suc\ n) \in borel\text{-}measurable\ (G\ (Suc\ n))$›
geom-proc-filt-info′[*of V* (*Suc n*) *Suc n*] **by** *simp*

```
        finally show ?thesis .
      qed
      finally show X (Suc n) w = V (Suc n) w .
    qed
  qed
 qed
}
thus ⋀n w. n ≤ matur ⟹
      val-process Mkt (self-fin-delta-pf N der matur (integral^L N (discounted-value
r (λm. der) matur))) n w =
      rn-price N der matur n w by (simp add: X-def init-def V-def delta-hedging-def)
qed


lemma (in CRR-market-viable) delta-hedging-same-cash-flow:
 assumes der ∈ borel-measurable (G matur)
and N = bernoulli-stream ((1 + r − d) / (u − d))
 shows cls-val-process Mkt (delta-hedging N der matur) matur w =
   der w
proof −
 define q where q = (1 + r − d) / (u − d)
 have 0 < q and q < 1 unfolding q-def using assms gt-param lt-param CRR-viable
by auto
 note qprops = this
 have cls-val-process Mkt (delta-hedging N der matur) matur w =
    val-process Mkt (delta-hedging N der matur) matur w using self-financingE
self-finance-charact
   unfolding delta-hedging-def self-fin-delta-pf-def
   by (metis delta-pf-portfolio mult-1s(1) mult-cancel-right not-real-square-gt-zero
positive)
 also have ... = rn-price N der matur matur w using delta-hedging-eq-derivative-price
assms by simp
 also have ... = rn-rev-price N der matur 0 w using rn-price-eq qprops assms
   unfolding rn-price-ind-def q-def by simp
 also have ... = der w by simp
 finally show ?thesis .
qed

lemma (in CRR-market) delta-hedging-trading-strat:
 assumes N = bernoulli-stream q
 and 0 < q
and q < 1
and der ∈ borel-measurable (G matur)
 shows trading-strategy (delta-hedging N der matur) unfolding delta-hedging-def
 by (simp add: assms self-fin-delta-pf-trad-strat)

lemma (in CRR-market) delta-hedging-self-financing:
 shows self-financing Mkt (delta-hedging N der matur) unfolding delta-hedging-def
self-fin-delta-pf-def
```

proof (*rule self-finance-charact*)
show $\forall\, n\ w.$ *prices Mkt risk-free-asset* (*Suc n*) $w \neq 0$ **using** *positive*
by (*metis less-numeral-extra*(*3*))
show *portfolio* (*delta-pf N der matur*) **using** *delta-pf-portfolio* **.**
qed

lemma (**in** *CRR-market-viable*) *delta-hedging-replicating*:
assumes *der* ∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream* ((*1* + *r* − *d*) / (*u* − *d*))
shows *replicating-portfolio* (*delta-hedging N der matur*) *der matur*
unfolding *replicating-portfolio-def*
proof (*intro conjI*)
define *q* **where** *q* = (*1* + *r* − *d*) / (*u* − *d*)
have *0* < *q* **and** *q* < *1* **unfolding** *q-def* **using** *assms gt-param lt-param CRR-viable*
by *auto*
note *qprops* = *this*
let *?X* = (*delta-hedging N der matur*)
show *trading-strategy ?X* **using** *delta-hedging-trading-strat qprops assms* **unfolding** *q-def* **by** *simp*
show *self-financing Mkt ?X* **using** *delta-hedging-self-financing* **.**
show *stock-portfolio Mkt* (*delta-hedging N der matur*) **unfolding** *delta-hedging-def self-fin-delta-pf-def*
stock-portfolio-def portfolio-def **using** *stocks delta-pf-support*
by (*smt Un-insert-right delta-pf-portfolio insert-commute portfolio-def self-finance-def self-finance-portfolio single-comp-support subset-insertI2 subset-singleton-iff sum-support-set sup-bot.right-neutral*)
show *AEeq M* (*cls-val-process Mkt* (*delta-hedging N der matur*) *matur*) *der*
using *delta-hedging-same-cash-flow assms* **by** *simp*
qed

definition (**in** *disc-equity-market*) *complete-market* **where**
complete-market ⟷ (∀ *matur*. ∀ *der*∈ *borel-measurable* (*F matur*). (∃ *p*. *replicating-portfolio p der matur*))

lemma (**in** *CRR-market-viable*) *CRR-market-complete*:
shows *complete-market* **unfolding** *complete-market-def*
proof (*intro allI impI*)
fix *matur*::*nat*
show ∀ *der* ∈ *borel-measurable* (*G matur*). (∃ *p*. *replicating-portfolio p der matur*)
proof
fix *der*::*bool stream*⇒*real*
assume *der* ∈ *borel-measurable* (*G matur*)
define *N* **where** *N* = *bernoulli-stream* ((*1* + *r* − *d*) / (*u* − *d*))
hence *replicating-portfolio* (*delta-hedging N der matur*) *der matur* **using** *delta-hedging-replicating* ‹*der* ∈ *borel-measurable* (*G matur*)› **by** *simp*
thus ∃ *pf*. *replicating-portfolio pf der matur* **by** *auto*
qed
qed

lemma *subalgebras-filtration*:
assumes *filtration M F*
and $\forall\, t.$ *subalgebra* $(F\ t)\ (G\ t)$
and $\forall\ s\ t.\ s \leq t \longrightarrow$ *subalgebra* $(G\ t)\ (G\ s)$
shows *filtration M G* **unfolding** *filtration-def*
proof (*intro conjI allI impI*)
{
fix *t*
have *subalgebra* $(F\ t)\ (G\ t)$ **using** *assms* **by** *simp*
moreover have *subalgebra M* $(F\ t)$ **using** *assms* **unfolding** *filtration-def* **by** *simp*
ultimately show *subalgebra M* $(G\ t)$ **by** (*metis subalgebra-def subsetCE subsetI*)
}
{
fix *s t*::*'b*
assume $s \leq t$
thus *subalgebra* $(G\ t)\ (G\ s)$ **using** *assms* **by** *simp*
}
qed

lemma *subfilt-filt-equiv*:
assumes *filt-equiv F M N*
and $\forall\ t.$ *subalgebra* $(F\ t)\ (G\ t)$
and $\forall\ s\ t.\ s \leq t \longrightarrow$ *subalgebra* $(G\ t)\ (G\ s)$
shows *filt-equiv G M N* **unfolding** *filt-equiv-def*
proof (*intro conjI*)
show *sets M* = *sets N* **using** *assms* **unfolding** *filt-equiv-def* **by** *simp*
show *filtration M G* **using** *assms subalgebras-filtration*[*of M F G*] **unfolding** *filt-equiv-def* **by** *simp*
show $\forall\, t\ A.\ A \in$ *sets* $(G\ t) \longrightarrow$ (*emeasure M A* = *0*) = (*emeasure N A* = *0*)
proof (*intro allI ballI impI*)
fix *t*
fix *A*
assume *A*∈ *sets* $(G\ t)$
hence *A* ∈ *sets* $(F\ t)$ **using** *assms* **unfolding** *subalgebra-def* **by** *auto*
thus (*emeasure M A* = *0*) = (*emeasure N A* = *0*) **using** *assms* **unfolding** *filt-equiv-def* **by** *simp*
qed
qed

lemma (**in** *CRR-market-viable*) *CRR-market-fair-price*:
assumes *pyf*∈ *borel-measurable* $(G\ matur)$
shows *fair-price Mkt*
$(\sum\ w \in$ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component* $((1 + r - d) / (u - d))\ w)\ \{0..<matur\}) *$

((*discounted-value r* (λ*m. pyf*) *matur*) *w*))
pyf matur
proof −
define *dpf* **where** *dpf* = (*discounted-value r* (λ*m. pyf*) *matur*)
define *q* **where** *q* = (*1* + *r* − *d*) / (*u* − *d*)
have ∃ *pf*. *replicating-portfolio pf pyf matur* **using** *CRR-market-complete assms*
unfolding *complete-market-def* **by** *simp*
from *this* **obtain** *pf* **where** *replicating-portfolio pf pyf matur* **by** *auto* **note**
pfprop = *this*
define *N* **where** *N* = *bernoulli-stream* ((*1* + *r* − *d*) / (*u* − *d*))
have *fair-price Mkt* (*integral*^{L} *N dpf*) *pyf matur* **unfolding** *dpf-def*
proof (*rule replicating-expectation-finite*)
show *risk-neutral-prob N* **using** *assms risk-neutral-iff*
using *CRR-viable gt-param lt-param N-def* **by** *blast*
have *filt-equiv nat-filtration M N* **using** *bernoulli-stream-equiv*[*of N* (*1*+*r*−*d*)/(*u*−*d*)]
assms gt-param lt-param CRR-viable psgt pslt N-def **by** *simp*
thus *filt-equiv G M N* **using** *subfilt-filt-equiv*
using *Filtration.filtration-def filtration geom-rand-walk-borel-adapted*
stoch-proc-subalg-nat-filt stock-filtration **by** *blast*
show *pyf* ∈ *borel-measurable* (*G matur*) **using** *assms* **by** *simp*
show *viable-market Mkt* **using** *CRR-viable* **by** *simp*
have *infinite-cts-filtration p M nat-filtration* **using** *bernoulli-nat-filtration*[*of M*
p] *bernoulli psgt pslt*
by *simp*
thus *sets* (*G 0*) = {{}, *space M*} **using** *stock-filtration*
infinite-cts-filtration.stoch-proc-filt-triv-init[*of p M nat-filtration geom-proc*]
geom-rand-walk-borel-adapted bot-nat-def **unfolding** *init-triv-filt-def* **by** *simp*
show *replicating-portfolio pf pyf matur* **using** *pfprop* **.**
show ∀ *n*. ∀ *asset*∈*support-set pf*. *finite* (*prices Mkt asset n* ‘ *space M*)
proof (*intro allI ballI*)
fix *n*
fix *asset*
assume *asset* ∈ *support-set pf*
hence *prices Mkt asset n* ∈ *borel-measurable* (*G n*) **using** *readable pfprop*
unfolding *replicating-portfolio-def stock-portfolio-def adapt-stoch-proc-def*
by *auto*
hence *prices Mkt asset n* ∈ *borel-measurable* (*nat-filtration n*) **using** *stock-filtration*
stoch-proc-subalg-nat-filt geom-rand-walk-borel-adapted
measurable-from-subalg[*of nat-filtration n G n prices Mkt asset n borel*]
unfolding *adapt-stoch-proc-def* **by** *auto*
thus *finite* (*prices Mkt asset n* ‘ *space M*) **using** *nat-filtration-vimage-finite*[*of*
prices Mkt asset n] **by** *simp*
qed
show ∀ *n*. ∀ *asset*∈*support-set pf*. *finite* (*pf asset n* ‘ *space M*)
proof (*intro allI ballI*)
fix *n*
fix *asset*
assume *asset* ∈ *support-set pf*
hence *pf asset n* ∈ *borel-measurable* (*G n*) **using** *pfprop predict-imp-adapt*[*of*

pf asset]
unfolding *replicating-portfolio-def trading-strategy-def adapt-stoch-proc-def*
by *auto*
hence *pf asset n* ∈ *borel-measurable* (*nat-filtration n*) **using** *stock-filtration*
stoch-proc-subalg-nat-filt geom-rand-walk-borel-adapted
measurable-from-subalg[*of nat-filtration n G n pf asset n borel*]
unfolding *adapt-stoch-proc-def* **by** *auto*
thus *finite* (*pf asset n ' space M*) **using** *nat-filtration-vimage-finite*[*of pf asset*
n] **by** *simp*
qed
qed
moreover have *integral*^{L} *N dpf* =
(∑ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component q w*) {*0..<matur*})
∗ (*dpf w*))
proof (*rule infinite-cts-filtration.expect-prob-comp*)
show *infinite-cts-filtration q N nat-filtration* **using** *assms pslt psgt*
bernoulli-nat-filtration **unfolding** *q-def* **using** *gt-param lt-param CRR-viable*
N-def **by** *auto*
have *dpf* ∈ *borel-measurable* (*G matur*) **using** *assms discounted-measurable*[*of*
pyf G matur]
unfolding *dpf-def* **by** *simp*
thus *dpf* ∈ *borel-measurable* (*nat-filtration matur*) **using** *stock-filtration*
stoch-proc-subalg-nat-filt geom-rand-walk-borel-adapted
measurable-from-subalg[*of nat-filtration matur G matur dpf*]
unfolding *adapt-stoch-proc-def* **by** *auto*
qed
ultimately show *?thesis* **unfolding** *dpf-def q-def* **by** *simp*
qed

end
theory *Option-Price-Examples* **imports** *CRR-Model*

begin

This file contains pricing results for four options in the Cox-Ross-Rubinstein model. The first section contains results relating some functions to the more abstract counterparts that were used to prove fairness and completeness results. The second section contains the pricing results for a few options; some path-dependent and others not.

9 Effective computation definitions and results

9.1 Generation of lists of boolean elements

The function gener-bool-list permits to generate lists of boolean elements. It is used to generate a list representative of the range of boolean streams by the function pseudo-proj-True.

fun *gener-bool-list* **where**

gener-bool-list 0 = {[]}
| *gener-bool-list* (*Suc n*) = {*True* # *w*| *w*. *w*∈ *gener-bool-list n*} ∪ {*False* # *w*| *w*. *w*∈ *gener-bool-list n*}

lemma *gener-bool-list-elem-length*:
shows ⋀*x*. *x*∈ *gener-bool-list n* ⟹ *length x* = *n*
proof (*induction n*)
case *0*
fix *x*
assume *x*∈ *gener-bool-list 0*
hence *x* = [] **by** *simp*
thus *length x* = *0* **by** *simp*
next
case (*Suc n*)
fix *x*
assume *x*∈ *gener-bool-list* (*Suc n*)
hence *mem*: *x*∈ {*True* # *w*| *w*. *w*∈ *gener-bool-list n*} ∪ {*False* # *w*| *w*. *w*∈ *gener-bool-list n*} **by** *simp*
show *length x* = *Suc n*
proof (*cases x*∈ {*True* # *w*| *w*. *w*∈ *gener-bool-list n*})
case *True*
hence ∃ *w* ∈ *gener-bool-list n*. *x* = *True* # *w* **by** *auto*
from *this* **obtain** *w* **where** *w*∈ *gener-bool-list n* **and** *x* = *True* # *w* **by** *auto*
hence *length w* = *n* **using** *Suc* **by** *simp*
thus *length x* = *Suc n* **using** ‹*x* = *True* # *w*› **by** *simp*
next
case *False*
hence *x*∈ {*False* # *w*| *w*. *w*∈ *gener-bool-list n*} **using** *mem* **by** *auto*
hence ∃ *w* ∈ *gener-bool-list n*. *x* = *False* # *w* **by** *auto*
from *this* **obtain** *w* **where** *w*∈ *gener-bool-list n* **and** *x* = *False* # *w* **by** *auto*
hence *length w* = *n* **using** *Suc* **by** *simp*
thus *length x* = *Suc n* **using** ‹*x* = *False* # *w*› **by** *simp*
qed
qed

lemma (**in** *infinite-coin-toss-space*) *stake-gener-bool-list*:
shows *stake n*‘*streams* (*UNIV*::*bool set*) = *gener-bool-list n*
proof (*induction n*)
case *0*
thus *stake 0* ‘ *streams UNIV* = *gener-bool-list 0* **by** *auto*
next
case (*Suc n*)
show *stake* (*Suc n*) ‘ *streams UNIV* = *gener-bool-list* (*Suc n*)
proof −
have *stake* (*Suc n*)‘*streams* (*UNIV*::*bool set*) = {*s*#*w*| *s w*. *s*∈ *UNIV* ∧ *w*∈ (*stake n* ‘(*streams UNIV*))}
by (*metis* (*no-types*) *UNIV-bool UNIV-not-empty stake-finite-universe-induct*[*of UNIV n*] *finite.emptyI finite-insert*)
also have ... = {*s*#*w*| *s w*. *s*∈ {*True*, *False*} ∧ *w*∈ (*stake n* ‘(*streams UNIV*))}

by *simp*
also have ... = {*s*#*w*| *s w*. *s*∈ {*True*, *False*} ∧ *w*∈ *gener-bool-list n*} **using** *Suc* **by** *simp*
also have ... = {*s*#*w*| *s w*. *s*∈ {*True*} ∧ *w*∈ *gener-bool-list n*} ∪ {*s*#*w*| *s w*. *s*∈ { *False*} ∧ *w*∈ *gener-bool-list n*} **by** *auto*
also have ... = {*True* # *w* | *w*. *w*∈ *gener-bool-list n*} ∪ {*False*#*w* | *w*. *w*∈ *gener-bool-list n*} **by** *auto*
also have ... = *gener-bool-list* (*Suc n*) **by** *simp*
finally show *?thesis* **.**
qed
qed

lemma (**in** *infinite-coin-toss-space*) *pseudo-range-stake*:
assumes ⋀*w*. *f w* = *g* (*stake n w*)
shows (∑ *w*∈ *range* (*pseudo-proj-True n*). *f w*) = (∑ *y*∈ (*gener-bool-list n*). *g y*)
proof (*rule sum.reindex-cong*)
show *inj-on* (λ *l*. *shift l* (*sconst True*)) (*gener-bool-list n*)
proof
fix *x y*
assume *x*∈ *gener-bool-list n*
and *y*∈ *gener-bool-list n*
and *x* @− *sconst True* = *y* @− *sconst True*
have *length x* = *n* **using** *gener-bool-list-elem-length* ‹*x*∈ *gener-bool-list n*› **by** *simp*
have *length y* = *n* **using** *gener-bool-list-elem-length* ‹*y*∈ *gener-bool-list n*› **by** *simp*
show *x* = *y*
proof −
have ∀ *i* < *n*. *nth x i* = *nth y i*
proof (*intro allI impI*)
fix *i*
assume *i* < *n*
have *xi*: *snth* (*x* @− *sconst True*) *i* = *nth x i* **using** ‹*i* < *n*› ‹*length x* = *n*› **by** *simp*
have *yi*: *snth* (*y* @− *sconst True*) *i* = *nth y i* **using** ‹*i* < *n*› ‹*length y* = *n*› **by** *simp*
have *snth* (*x* @− *sconst True*) *i* = *snth* (*y* @− *sconst True*) *i* **using** ‹*x* @− *sconst True* = *y* @− *sconst True*›
by *simp*
thus *nth x i* = *nth y i* **using** *xi yi* **by** *simp*
qed
thus *?thesis* **using** ‹*length x* = *n*› ‹*length y* = *n*› **by** (*simp add*: *list-eq-iff-nth-eq*)
qed
qed
have *range* (*pseudo-proj-True n*) = {*shift l* (*sconst True*)|*l*. *l*∈(*stake n* ‘*streams* (*UNIV*::*bool set*))}
unfolding *pseudo-proj-True-def* **by** *auto*
also have ... = {*shift l* (*sconst True*)|*l*. *l*∈(*gener-bool-list n*)} **using** *stake-gener-bool-list*

by *simp*
also have ... = (λ*l*. *l* @− *sconst True*) ‘ *gener-bool-list n* **by** *auto*
finally show *range* (*pseudo-proj-True n*) = (λ*l*. *l* @− *sconst True*) ‘ *gener-bool-list n* **.**
fix *x*
assume *x*∈ *gener-bool-list n*
have *length x* = *n* **using** *gener-bool-list-elem-length* ‹*x*∈ *gener-bool-list n*› **by** *simp*
have *f* (*x* @− *sconst True*) = *g* (*stake n* (*x* @− *sconst True*)) **using** *assms* **by** *simp*
also have ... = *g x* **using** ‹*length x* = *n*› **by** (*simp add*: *stake-shift*)
finally show *f* (*x* @− *sconst True*) = *g x* **.**
qed

9.2 Probability components for lists

fun *lprob-comp* **where**
lprob-comp (*p*::*real*) [] = *1*
| *lprob-comp p* (*x* # *xs*) = (*if x then p else* (*1*−*p*)) ∗ *lprob-comp p xs*

lemma *lprob-comp-last*:
shows *lprob-comp p* (*xs* @ [*x*]) = (*lprob-comp p xs*) ∗ (*if x then p else* (*1* − *p*))
proof (*induction xs*)
case *Nil*
have *lprob-comp p* (*Nil* @ [*x*]) = *lprob-comp p* [*x*] **by** *simp*
also have ... = (*if x then p else* (*1* − *p*)) **by** *simp*
also have ... = (*lprob-comp p Nil*) ∗ (*if x then p else* (*1* − *p*)) **by** *simp*
finally show *lprob-comp p* (*Nil* @ [*x*]) = (*lprob-comp p Nil*) ∗ (*if x then p else* (*1* − *p*)) **.**
next
case (*Cons a xs*)
have *lprob-comp p* ((*Cons a xs*) @ [*x*]) = (*if a then p else* (*1* − *p*)) ∗ *lprob-comp p* (*xs* @ [*x*]) **by** *simp*
also have ... = (*if a then p else* (*1* − *p*)) ∗ (*lprob-comp p xs*) ∗ (*if x then p else* (*1*−*p*)) **using** *Cons* **by** *simp*
also have ... = *lprob-comp p* (*Cons a xs*) ∗ (*if x then p else* (*1*−*p*)) **by** *simp*
finally show *lprob-comp p* ((*Cons a xs*) @ [*x*]) = *lprob-comp p* (*Cons a xs*) ∗ (*if x then p else* (*1*−*p*)) **.**
qed

lemma (**in** *infinite-coin-toss-space*) *lprob-comp-stake*:
shows (*prod* (*prob-component pr w*) {*0*..<*matur*}) = *lprob-comp pr* (*stake matur w*)
proof (*induction matur*)
case *0*
have *prod* (*prob-component pr w*) {*0*..<*0*} = *1* **by** *simp*
also have ... = *lprob-comp pr* [] **by** *simp*
also have ... = *lprob-comp pr* (*stake 0 w*) **by** *simp*

finally show *prod (prob-component pr w) {0..<0} = lprob-comp pr (stake 0 w)*
.
next
case *(Suc n)*
have *prod (prob-component pr w) {0..<Suc n} = prod (prob-component pr w) {0..< n} ∗*
(prob-component pr w n) **using** *prod.atLeast0-lessThan-Suc* **by** *blast*
also have *... = lprob-comp pr (stake n w) ∗ (prob-component pr w n)* **using** *Suc* **by** *simp*
also have *... = lprob-comp pr (stake n w) ∗ (if (snth w n) then pr else 1−pr)* **by** *(simp add: prob-component-def)*
also have *... = lprob-comp pr ((stake n w) @ [snth w n])* **by** *(simp add: lprob-comp-last)*
also have *... = lprob-comp pr (stake (Suc n) w)* **by** *(metis Stream.stake-Suc)*
finally show *prod (prob-component pr w) {0..<Suc n} = lprob-comp pr (stake (Suc n) w)* **.**
qed

9.3 Geometric process applied to lists

fun *lrev-geom* **where**
lrev-geom u d v [] = v
| lrev-geom u d v (x#xs) = (if x then u else d) ∗ lrev-geom u d v xs

fun *lgeom-proc* **where** *lgeom-proc u d v l = lrev-geom u d v (rev l)*

lemma (**in** *infinite-coin-toss-space*) *geom-lgeom*:
shows *geom-rand-walk u d v n w = lgeom-proc u d v (stake n w)*
proof *(induction n)*
case *0*
have *geom-rand-walk u d v 0 w = v* **by** *simp*
also have *... = lrev-geom u d v []* **by** *simp*
also have *... = lrev-geom u d v (rev (stake 0 w))* **by** *simp*
also have *... = lgeom-proc u d v (stake 0 w)* **by** *simp*
finally show *geom-rand-walk u d v 0 w = lgeom-proc u d v (stake 0 w)* **.**
next
case *(Suc n)*
have *snth w n = nth (stake (Suc n) w) n* **using** *stake-nth* **by** *blast*
have *(stake n w) @ [nth (stake (Suc n) w) n] = stake (Suc n) w*
by *(metis Stream.stake-Suc lessI stake-nth)*
have *geom-rand-walk u d v (Suc n) w = ((λTrue ⇒ u | False ⇒ d) (snth w n)) ∗ geom-rand-walk u d v n w* **by** *simp*
also have *... = (if (snth w n) then u else d) ∗ geom-rand-walk u d v n w* **by** *simp*
also have *... = (if (snth w n) then u else d) ∗ lgeom-proc u d v (stake n w)* **using** *Suc* **by** *simp*
also have *... = (if (snth w n) then u else d) ∗ lrev-geom u d v (rev (stake n w))* **by** *simp*
also have *... = lrev-geom u d v ((snth w n) # (rev (stake n w)))* **by** *simp*
also have *... = lrev-geom u d v (rev ((stake n w) @ [snth w n]))* **by** *simp*
also have *... = lrev-geom u d v (rev ((stake n w) @ [nth (stake (Suc n) w) n]))*

using ‹*snth w n = nth* (*stake* (*Suc n*) *w*) *n*› **by** *simp*
also have ... = *lrev-geom u d v* (*rev* (*stake* (*Suc n*) *w*))
using ‹(*stake n w*) @ [*nth* (*stake* (*Suc n*) *w*) *n*] = *stake* (*Suc n*) *w*› **by** *simp*
also have ... = *lgeom-proc u d v* (*stake* (*Suc n*) *w*) **by** *simp*
finally show *geom-rand-walk u d v* (*Suc n*) *w* = *lgeom-proc u d v* (*stake* (*Suc n*) *w*) .
qed

lemma *lgeom-proc-take*:
assumes $i \leq n$
shows *lgeom-proc u d init* (*stake i w*) = *lgeom-proc u d init* (*take i* (*stake n w*))
proof −
have *stake i w* = *take i* (*stake n w*) **using** *assms* **by** (*simp add*: *min.absorb1 take-stake*)
thus *?thesis* **by** *simp*
qed

9.4 Effective computation of discounted values

fun *det-discount* **where**
det-discount (*r*::*real*) *0* = *1*
| *det-discount r* (*Suc n*) = (*inverse* (*1*+*r*)) ∗ (*det-discount r n*)

lemma *det-discounted*:
shows *discounted-value r X n w* = (*det-discount r n*) ∗ (*X n w*) **unfolding** *discounted-value-def discount-factor-def*
proof (*induction n arbitrary*: *X*)
case *0*
have *inverse* (*disc-rfr-proc r 0 w*) ∗ *X 0 w* = *X 0 w* **by** *simp*
also have ... = (*det-discount r 0*) ∗ (*X 0 w*) **by** *simp*
finally show *inverse* (*disc-rfr-proc r 0 w*) ∗ *X 0 w* = (*det-discount r 0*) ∗ (*X 0 w*) .
next
case (*Suc n*)
have *inverse* (*disc-rfr-proc r* (*Suc n*) *w*) ∗ *X* (*Suc n*) *w* =
inverse ((*1*+*r*) ∗ (*disc-rfr-proc r*) *n w*) ∗ *X* (*Suc n*) *w* **by** *simp*
also have ... = (*inverse* (*1*+*r*)) ∗ *inverse* ((*disc-rfr-proc r*) *n w*) ∗ *X* (*Suc n*) *w* **by** *simp*
also have ... = (*inverse* (*1*+*r*)) ∗ (*det-discount r n*) ∗ *X* (*Suc n*) *w* **using** *Suc*[*of* λ*n*. *X* (*Suc n*)] **by** *auto*
also have ... = (*det-discount r* (*Suc n*)) ∗ *X* (*Suc n*) *w* **by** *simp*
finally show *inverse* (*disc-rfr-proc r* (*Suc n*) *w*) ∗ *X* (*Suc n*) *w* = (*det-discount r* (*Suc n*)) ∗ *X* (*Suc n*) *w* .
qed

10 Pricing results on options

10.1 Call option

A call option is parameterized by a strike K and maturity T. If S denotes the price of the (unique) risky asset at time T, then the option pays max(S - K, 0) at that time.

definition (**in** *CRR-market*) *call-option* **where**
call-option (*T*::*nat*) (*K*::*real*) = (λ *w*. *max* (*prices Mkt stk T w* − *K*) *0*)

lemma (**in** *CRR-market*) *call-borel*:
shows *call-option T K* ∈ *borel-measurable* (*G T*) **unfolding** *call-option-def*
proof (*rule borel-measurable-max*)
show (λ*x*. *0*) ∈ *borel-measurable* (*G T*) **by** *simp*
show (λ*x*. *prices Mkt stk T x* − *K*) ∈ *borel-measurable* (*G T*)
proof (*rule borel-measurable-diff*)
show *prices Mkt stk T* ∈ *borel-measurable* (*G T*)
by (*metis adapt-stoch-proc-def stock-price-borel-measurable*)
qed *simp*
qed

lemma (**in** *CRR-market-viable*) *call-option-lgeom*:
shows *call-option T K w* = *max* ((*lgeom-proc u d init* (*stake T w*)) − *K*) *0*
using *geom-lgeom stk-price geometric-process* **unfolding** *call-option-def* **by** *simp*

lemma (**in** *CRR-market-viable*) *disc-call-option-lgeom*:
shows (*discounted-value r* (λ*m*. (*call-option T K*)) *T w*) =
(*det-discount r T*) ∗ (*max* ((*lgeom-proc u d init* (*stake T w*)) − *K*) *0*)
using *det-discounted*[*of r* λ*m*. *call-option T K T w*] *call-option-lgeom*[*of T K w*] **by** *simp*

lemma (**in** *CRR-market-viable*) *call-effect-compute*:
shows (∑ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component pr w*) {*0*..<*matur*}) ∗
(*discounted-value r* (λ*m*. (*call-option matur K*)) *matur w*)) =
(∑ *y*∈ (*gener-bool-list matur*). *lprob-comp pr y* ∗ (*det-discount r matur*) ∗
(*max* ((*lgeom-proc u d init* (*take matur y*)) − *K*) *0*))
proof (*rule pseudo-range-stake*)
fix *w*
have *prod* (*prob-component pr w*) {*0*..<*matur*} ∗ *discounted-value r* (λ*m*. *call-option matur K*) *matur w* =
lprob-comp pr (*stake matur w*) ∗ *discounted-value r* (λ*m*. *call-option matur K*) *matur w*
using *lprob-comp-stake* **by** *simp*
also have ... = *lprob-comp pr* (*stake matur w*) ∗
(*det-discount r matur*) ∗ (*max* ((*lgeom-proc u d init* (*take matur* (*stake matur w*))) − *K*) *0*)
using *disc-call-option-lgeom*[*of matur K*] **by** *simp*
finally show *prod* (*prob-component pr w*) {*0*..<*matur*} ∗ *discounted-value r* (λ*m*.

call-option matur K) *matur w* =
lprob-comp pr (*stake matur w*) ∗
(*det-discount r matur*) ∗ (*max* ((*lgeom-proc u d init* (*take matur* (*stake matur w*))) − *K*) *0*) .
qed

fun *call-price* **where**
call-price u d init r matur K = (∑ *y*∈ (*gener-bool-list matur*). *lprob-comp* ((*1* + *r* − *d*) / (*u* − *d*)) *y* ∗ (*det-discount r matur*) ∗
(*max* ((*lgeom-proc u d init* (*take matur* (*take matur y*))) − *K*) *0*))

Evaluating the function above returns the fair price of a call option.

lemma (**in** *CRR-market-viable*) *call-price*:
shows *fair-price Mkt*
(*call-price u d init r matur K*)
(*call-option matur K*) *matur*
proof −
have *fair-price Mkt*
(∑ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component* ((*1* + *r* − *d*) / (*u* − *d*)) *w*) {*0*..<*matur*}) ∗
(*discounted-value r* (λ*m*. (*call-option matur K*)) *matur w*))
(*call-option matur K*) *matur*
by (*rule CRR-market-fair-price*, *rule call-borel*)
thus *?thesis* **using** *call-effect-compute* **by** *simp*
qed

10.2 Put option

A put option is also parameterized by a strike K and maturity T. If S denotes the price of the (unique) risky asset at time T, then the option pays max(K - S, 0) at that time.

definition (**in** *CRR-market*) *put-option* **where**
put-option (*T*::*nat*) (*K*::*real*) = (λ *w*. *max* (*K* − *prices Mkt stk T w*) *0*)

lemma (**in** *CRR-market*) *put-borel*:
shows *put-option T K* ∈ *borel-measurable* (*G T*) **unfolding** *put-option-def*
proof (*rule borel-measurable-max*)
show (λ*x*. *0*) ∈ *borel-measurable* (*G T*) **by** *simp*
show (λ*x*. *K* − *prices Mkt stk T x*) ∈ *borel-measurable* (*G T*)
proof (*rule borel-measurable-diff*)
show *prices Mkt stk T* ∈ *borel-measurable* (*G T*)
by (*metis adapt-stoch-proc-def stock-price-borel-measurable*)
qed *simp*
qed

lemma (**in** *CRR-market-viable*) *put-option-lgeom*:
shows *put-option T K w* = *max* (*K* − (*lgeom-proc u d init* (*stake T w*))) *0*
using *geom-lgeom stk-price geometric-process* **unfolding** *put-option-def* **by** *simp*

lemma (**in** *CRR-market-viable*) *disc-put-option-lgeom*:
shows (*discounted-value r* (λ*m*. (*put-option T K*)) *T w*) =
(*det-discount r T*) ∗ (*max* (*K* − (*lgeom-proc u d init* (*stake T w*))) *0*)
using *det-discounted*[*of r* λ*m*. *put-option T K T w*] *put-option-lgeom*[*of T K w*]
by *simp*

lemma (**in** *CRR-market-viable*) *put-effect-compute*:
shows (∑ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component pr w*) {*0*..<*matur*}) ∗
(*discounted-value r* (λ*m*. (*put-option matur K*)) *matur w*)) =
(∑ *y*∈ (*gener-bool-list matur*). *lprob-comp pr y* ∗ (*det-discount r matur*) ∗
(*max* (*K* − (*lgeom-proc u d init* (*take matur y*))) *0*))
proof (*rule pseudo-range-stake*)
fix *w*
have *prod* (*prob-component pr w*) {*0*..<*matur*} ∗ *discounted-value r* (λ*m*. *put-option matur K*) *matur w* =
lprob-comp pr (*stake matur w*) ∗ *discounted-value r* (λ*m*. *put-option matur K*) *matur w*
using *lprob-comp-stake* **by** *simp*
also have ... = *lprob-comp pr* (*stake matur w*) ∗
(*det-discount r matur*) ∗ (*max* (*K* − (*lgeom-proc u d init* (*take matur* (*stake matur w*)))) *0*)
using *disc-put-option-lgeom*[*of matur K*] **by** *simp*
finally show *prod* (*prob-component pr w*) {*0*..<*matur*} ∗ *discounted-value r* (λ*m*. *put-option matur K*) *matur w* =
lprob-comp pr (*stake matur w*) ∗
(*det-discount r matur*) ∗ (*max* (*K* − (*lgeom-proc u d init* (*take matur* (*stake matur w*)))) *0*) **.**
qed

fun *put-price* **where**
put-price u d init r matur K = (∑ *y*∈ (*gener-bool-list matur*). *lprob-comp* ((*1* + *r* − *d*) / (*u* − *d*)) *y* ∗ (*det-discount r matur*) ∗
(*max* (*K* − (*lgeom-proc u d init* (*take matur* (*take matur y*)))) *0*))

Evaluating the function above returns the fair price of a put option.

lemma (**in** *CRR-market-viable*) *put-price*:
shows *fair-price Mkt*
(*put-price u d init r matur K*)
(*put-option matur K*) *matur*
proof −
have *fair-price Mkt*
(∑ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component* ((*1* + *r* − *d*) / (*u* − *d*)) *w*) {*0*..<*matur*}) ∗
(*discounted-value r* (λ*m*. (*put-option matur K*)) *matur w*))
(*put-option matur K*) *matur*
by (*rule CRR-market-fair-price*, *rule put-borel*)
thus *?thesis* **using** *put-effect-compute* **by** *simp*

qed

10.3 Lookback option

A lookback option is parameterized by a maturity T. If Sn denotes the price of the (unique) risky asset at time n, then the option pays max(Sn. 0 <= n <= T) - ST at that time.

definition (**in** *CRR-market*) *lbk-option* **where**
lbk-option (*T*::*nat*) = (λ *w*. *Max* ((λ*i*. (*prices Mkt stk*) *i w*)‘{*0* .. *T*}) − (*prices Mkt stk T w*))

lemma *borel-measurable-Max-finite*:
fixes *f*::′*a* ⇒ ′*b* ⇒ ′*c*::{*second-countable-topology*, *linorder-topology*}
assumes *0* < (*n*::*nat*)
shows ⋀*A*. *card A* = *n* ⟹ ∀ *a* ∈ *A*. *f a* ∈ *borel-measurable M* ⟹ (λ*w*. *Max* ((λ*a*. *f a w*)‘*A*)) ∈ *borel-measurable M* **using** *assms*
proof (*induct n*)
case *0*
show ⋀*A*. *card A* = *0* ⟹ ∀ *a*∈*A*. *f a* ∈ *borel-measurable M* ⟹ (*0*::*nat*) < *0* ⟹ (λ*w*. *Max* ((λ*a*. *f a w*) ‘ *A*)) ∈ *borel-measurable M*
proof −
fix *A*::′*a set*
assume *card A* = *0* **and** ∀ *a*∈*A*. *f a* ∈ *borel-measurable M* **and** (*0*::*nat*) < *0*
thus (λ*w*. *Max* ((λ*a*. *f a w*) ‘ *A*)) ∈ *borel-measurable M* **by** *simp*
qed
next
case *Suc*
show ⋀*n A*. (⋀*A*. *card A* = *n* ⟹
∀ *a*∈*A*. *f a* ∈ *borel-measurable M* ⟹ *0* < *n* ⟹ (λ*w*. *Max* ((λ*a*. *f a w*) ‘ *A*)) ∈ *borel-measurable M*) ⟹
card A = *Suc n* ⟹
∀ *a*∈*A*. *f a* ∈ *borel-measurable M* ⟹ *0* < *Suc n* ⟹ (λ*w*. *Max* ((λ*a*. *f a w*) ‘ *A*)) ∈ *borel-measurable M*
proof −
fix *n*
fix *A*::′*a set*
assume *ameas*: (⋀*A*. *card A* = *n* ⟹
∀ *a*∈*A*. *f a* ∈ *borel-measurable M* ⟹ *0* < *n* ⟹ (λ*w*. *Max* ((λ*a*. *f a w*) ‘ *A*)) ∈ *borel-measurable M*)
and *card A* = *Suc n*
and ∀ *a*∈*A*. *f a* ∈ *borel-measurable M*
and *0* < *Suc n*
from ‹*card A* = *Suc n*› **have** *aprop*: *A*≠ {} ∧ *finite A* **using** *card-eq-0-iff*[*of A*] **by** *simp*
hence ∃ *x*. *x*∈ *A* **by** *auto*
from *this* **obtain** *a* **where** *a*∈ *A* **by** *auto*
hence *Suc* (*card* (*A* − {*a*})) = *Suc n* **using** *aprop card-Suc-Diff1*[*of A*] ‹*card A* = *Suc n*› **by** *auto*
hence *card* (*A* − {*a*}) = *n* **by** *simp*

show $(\lambda w.\ Max\ ((\lambda a.\ f\ a\ w)\ `\ A)) \in borel\text{-}measurable\ M$
proof $(cases\ n = 0)$
case *False*
hence $0 < n$ **by** *simp*
moreover have $\forall\, a{\in}A - \{a\}.\ f\ a \in borel\text{-}measurable\ M$ **using** ‹$\forall\, a{\in}A.\ f\ a \in borel\text{-}measurable\ M$› **by** *simp*
ultimately have $(\lambda w.\ Max\ ((\lambda a.\ f\ a\ w)\ `\ (A{-}\{a\}))) \in borel\text{-}measurable\ M$ **using** ‹$card\ (A - \{a\}) = n$›
ameas[*of* $A - \{a\}$] **by** *simp*
moreover have $f\ a \in borel\text{-}measurable\ M$ **using** ‹$\forall\, a{\in}A.\ f\ a \in borel\text{-}measurable\ M$› ‹$a{\in}A$› **by** *simp*
ultimately have $(\lambda\ w.\ max\ (f\ a\ w)\ (Max\ ((\lambda a.\ f\ a\ w)\ `\ (A{-}\{a\})))) \in borel\text{-}measurable\ M$
using *borel-measurable-max* **by** *simp*
moreover have $\bigwedge w.\ max\ (f\ a\ w)\ (Max\ ((\lambda a.\ f\ a\ w)\ `\ (A{-}\{a\}))) = Max\ ((\lambda a.\ f\ a\ w)\ `A)$
proof −
fix *w*
define *FA* **where** $FA = ((\lambda a.\ f\ a\ w)\ `\ (A{-}\{a\}))$
have *finite FA* **unfolding** *FA-def* **using** *aprop* **by** *simp*
have $A - \{a\} \neq \{\}$ **using** *aprop False* ‹$card\ (A - \{a\}) = n$› *card-eq-0-iff*[*of* $A - \{a\}$] **by** *simp*
hence $FA \neq \{\}$ **unfolding** *FA-def* **by** *simp*
have $max\ (f\ a\ w)\ (Max\ FA) = Max\ (insert\ (f\ a\ w)\ FA)$ **using** ‹*finite FA*› ‹$FA \neq \{\}$› **by** *simp*
hence $max\ (f\ a\ w)\ (Max\ ((\lambda a.\ f\ a\ w)\ `\ (A{-}\{a\}))) = Max\ (insert\ (f\ a\ w)\ ((\lambda a.\ f\ a\ w)\ `(A{-}\{a\})))$
unfolding *FA-def* **by** *simp*
also have $\ldots = Max\ ((\lambda a.\ f\ a\ w)\ `A)$
proof −
have $insert\ (f\ a\ w)\ ((\lambda a.\ f\ a\ w)\ `(A{-}\{a\})) = (\lambda a.\ f\ a\ w)\ `(insert\ a\ (A - \{a\}))$
by *auto*
also have $\ldots = ((\lambda a.\ f\ a\ w)\ `A)$ **using** ‹$a \in A$› **by** *blast*
finally have $insert\ (f\ a\ w)\ ((\lambda a.\ f\ a\ w)\ `(A{-}\{a\})) = ((\lambda a.\ f\ a\ w)\ `A)$ **.**
thus *?thesis* **by** *simp*
qed
finally show $max\ (f\ a\ w)\ (Max\ ((\lambda a.\ f\ a\ w)\ `\ (A{-}\{a\}))) = Max\ ((\lambda a.\ f\ a\ w)\ `A)$ **.**
qed
ultimately show $(\lambda w.\ Max\ ((\lambda a.\ f\ a\ w)\ `A)) \in borel\text{-}measurable\ M$ **by** *simp*
next
case *True*
hence $A - \{a\} = \{\}$ **using** *aprop card-eq-0-iff*[*of* $A - \{a\}$] ‹$card\ (A - \{a\}) = n$› **by** *simp*
hence $\{a\} = insert\ a\ (A - \{a\})$ **by** *simp*
also have $\ldots = A$ **using** ‹$a{\in}\ A$› **by** *blast*
finally have $\{a\} = A$ **.**
hence $\bigwedge w.\ (\lambda a.\ f\ a\ w)\ `A = \{f\ a\ w\}$ **by** *auto*

hence $\bigwedge w.\ Max\ ((\lambda a.\ f\ a\ w)\ `A) = Max\ \{f\ a\ w\}$ **by** *simp*
hence $\bigwedge w.\ Max\ ((\lambda a.\ f\ a\ w)\ `A) = f\ a\ w$ **by** *simp*
hence $(\lambda w.\ Max\ ((\lambda a.\ f\ a\ w)\ `A)) = f\ a$ **by** *simp*
thus $(\lambda w.\ Max\ ((\lambda a.\ f\ a\ w)\ `A)) \in$ *borel-measurable M* **using** ‹$\forall\, a{\in}A.\ f\ a \in$ *borel-measurable M*›
‹$a{\in}\ A$› **by** *simp*
qed
qed
qed

lemma (**in** *CRR-market*) *lbk-borel*:
shows *lbk-option T* $\in$ *borel-measurable* (*G T*) **unfolding** *lbk-option-def*
proof (*rule borel-measurable-diff*)
show $(\lambda x.\ Max\ ((\lambda i.\ prices\ Mkt\ stk\ i\ x)\ `\ \{0..T\})) \in$ *borel-measurable* (*G T*)
proof (*rule borel-measurable-Max-finite*)
show *card* $\{0..T\} = Suc\ T$ **by** *simp*
show $0 < Suc\ T$ **by** *simp*
show $\forall\, i{\in}\{0..T\}.$ *prices Mkt stk i* $\in$ *borel-measurable* (*G T*)
proof
fix *i*
assume $i{\in}\ \{0..T\}$
show *prices Mkt stk i* $\in$ *borel-measurable* (*G T*)
by (*metis* ‹$i \in \{0..T\}$› *adapt-stoch-proc-def atLeastAtMost-iff increasing-measurable-info*
stock-price-borel-measurable)
qed
qed
show *prices Mkt stk T* $\in$ *borel-measurable* (*G T*) **by** (*metis adapt-stoch-proc-def stock-price-borel-measurable*)
qed

lemma (**in** *CRR-market-viable*) *lbk-option-lgeom*:
shows *lbk-option T w* $= Max\ ((\lambda i.\ (lgeom\text{-}proc\ u\ d\ init\ (stake\ i\ w)))`\{0\ ..\ T\}) - (lgeom\text{-}proc\ u\ d\ init\ (stake\ T\ w))$
using *geom-lgeom stk-price geometric-process* **unfolding** *lbk-option-def* **by** *simp*

lemma (**in** *CRR-market-viable*) *disc-lbk-option-lgeom*:
shows $(discounted\text{-}value\ r\ (\lambda m.\ (lbk\text{-}option\ T))\ T\ w) =$
$(det\text{-}discount\ r\ T) * (Max\ ((\lambda i.\ (lgeom\text{-}proc\ u\ d\ init\ (take\ i\ (stake\ T\ w))))`\{0\ ..\ T\}) - (lgeom\text{-}proc\ u\ d\ init\ (stake\ T\ w)))$
using *det-discounted*[*of r* $\lambda m.$ *lbk-option T T w*] *lbk-option-lgeom*[*of T w*] *lgeom-proc-take*
by (*metis* (*no-types*, *lifting*) *atLeastAtMost-iff image-cong*)

lemma (**in** *CRR-market-viable*) *lbk-effect-compute*:
shows $(\sum\ w{\in}\ range\ (pseudo\text{-}proj\text{-}True\ matur).\ (prod\ (prob\text{-}component\ pr\ w)\ \{0..{<}matur\}) *$

```
    (discounted-value r (λm. (lbk-option matur)) matur w)) =
    (∑ y∈ (gener-bool-list matur). lprob-comp pr y * (det-discount r matur) *
    (Max ((λi. (lgeom-proc u d init (take i y)))`{0 .. matur}) - (lgeom-proc u d
init y)))
proof (rule pseudo-range-stake)
  fix w
  have prod (prob-component pr w) {0..<matur} * discounted-value r (λm. lbk-option
matur) matur w =
     lprob-comp pr (stake matur w) * discounted-value r (λm. lbk-option matur)
matur w
    using lprob-comp-stake by simp
  also have ... = lprob-comp pr (stake matur w) *
    (det-discount r matur) * (Max ((λi. (lgeom-proc u d init (take i (stake matur
w))))`{0 .. matur}) -
     (lgeom-proc u d init (stake matur w))) using disc-lbk-option-lgeom by simp
  finally show prod (prob-component pr w) {0..<matur} * discounted-value r (λm.
lbk-option matur) matur w =
    lprob-comp pr (stake matur w) *
    (det-discount r matur) * (Max ((λi. (lgeom-proc u d init (take i (stake matur
w))))`{0 .. matur}) -
     (lgeom-proc u d init (stake matur w))) .
qed

fun lbk-price where
lbk-price u d init r matur = (∑ y∈ (gener-bool-list matur). lprob-comp ((1 + r -
d) / (u - d)) y * (det-discount r matur) *
    (Max ((λi. (lgeom-proc u d init (take i y)))`{0 .. matur}) - (lgeom-proc u d
init y)))
```

Evaluating the function above returns the fair price of a lookback option.

```
lemma (in CRR-market-viable) lbk-price:
  shows fair-price Mkt
    (lbk-price u d init r matur)
    (lbk-option matur) matur
proof -
  have fair-price Mkt
    (∑ w∈ range (pseudo-proj-True matur). (prod (prob-component ((1 + r - d)
/ (u - d)) w) {0..<matur}) *
     (discounted-value r (λm. (lbk-option matur)) matur w))
    (lbk-option matur) matur
    by (rule CRR-market-fair-price, rule lbk-borel)
  thus ?thesis using lbk-effect-compute by simp
qed

value lbk-price 1.2 0.8 10 0.03 2
```

10.4 Asian option

An asian option is parameterized by a maturity T. This option pays the average price of the risky asset at time T.

definition (**in** *CRR-market*) *asian-option* **where**
asian-option (*T*::*nat*) = (λ *w*. (∑ *i*∈ {*1*.. *T*}. *prices Mkt stk i w*)/ *T*)

lemma (**in** *CRR-market*) *asian-borel*:
shows *asian-option T* ∈ *borel-measurable* (*G T*) **unfolding** *asian-option-def*
proof −
have (λ *w*. (∑ *i*∈ {*1*.. *T*}. *prices Mkt stk i w*)) ∈ *borel-measurable* (*G T*)
proof (*rule borel-measurable-sum*)
fix *i*
assume *i*∈ {*1*..*T*}
show *prices Mkt stk i* ∈ *borel-measurable* (*G T*)
by (*metis* ‹*i* ∈ {*1*..*T*}› *adapt-stoch-proc-def atLeastAtMost-iff increasing-measurable-info*

stock-price-borel-measurable)
qed
from *this* **show** (λ*w*. (∑ *i* = *1*..*T*. *prices Mkt stk i w*) / *real T*) ∈ *borel-measurable* (*G T*) **by** *simp*
qed

lemma (**in** *CRR-market-viable*) *asian-option-lgeom*:
shows *asian-option T w* = (∑ *i*∈ {*1*.. *T*}. *lgeom-proc u d init* (*stake i w*))/ *T*
using *geom-lgeom stk-price geometric-process* **unfolding** *asian-option-def* **by** *simp*

lemma (**in** *CRR-market-viable*) *disc-asian-option-lgeom*:
shows (*discounted-value r* (λ*m*. (*asian-option T*)) *T w*) =
(*det-discount r T*) ∗ (∑ *i*∈ {*1*.. *T*}. *lgeom-proc u d init* (*take i* (*stake T w*)))/ *T*
proof −
have ∀ *i*∈ {*1*..*T*}. *lgeom-proc u d init* (*stake i w*) = *lgeom-proc u d init* (*take i* (*stake T w*))
using *lgeom-proc-take* **by** *auto*
hence (∑ *i*∈ {*1*.. *T*}. *lgeom-proc u d init* (*stake i w*)) =
(∑ *i*∈ {*1*.. *T*}. *lgeom-proc u d init* (*take i* (*stake T w*))) **by** *auto*
thus *?thesis*
using *det-discounted*[*of r* λ*m*. *asian-option T T w*] *asian-option-lgeom*[*of T w*]
by *auto*
qed

lemma (**in** *CRR-market-viable*) *asian-effect-compute*:
shows (∑ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component pr w*) {*0*..<*matur*}) ∗
(*discounted-value r* (λ*m*. (*asian-option matur*)) *matur w*)) =
(∑ *y*∈ (*gener-bool-list matur*). *lprob-comp pr y* ∗ (*det-discount r matur*) ∗

```
    (∑ i∈ {1.. matur}. lgeom-proc u d init (take i y))/ matur)
proof (rule pseudo-range-stake)
  fix w
  have prod (prob-component pr w) {0..<matur} * discounted-value r (λm. asian-option
matur) matur w =
    lprob-comp pr (stake matur w) * discounted-value r (λm. asian-option matur)
matur w
    using lprob-comp-stake by simp
  also have ... = lprob-comp pr (stake matur w) *
    (det-discount r matur) * (∑ i∈ {1.. matur}. lgeom-proc u d init (take i (stake
matur w)))/ matur
    using disc-asian-option-lgeom[of matur w] by simp
  finally show prod (prob-component pr w) {0..<matur} * discounted-value r (λm.
asian-option matur) matur w =
    lprob-comp pr (stake matur w) *
    (det-discount r matur) * (∑ i∈ {1.. matur}. lgeom-proc u d init (take i (stake
matur w)))/ matur .
qed

fun asian-price where
asian-price u d init r matur = (∑ y∈ (gener-bool-list matur). lprob-comp ((1 + r
- d) / (u - d)) y * (det-discount r matur) *
      (∑ i∈ {1.. matur}. lgeom-proc u d init (take i y))/ matur)
```

Evaluating the function above returns the fair price of an asian option.

```
lemma (in CRR-market-viable) asian-price:
  shows fair-price Mkt
    (asian-price u d init r matur)
    (asian-option matur) matur
proof -
  have fair-price Mkt
    (∑ w∈ range (pseudo-proj-True matur). (prod (prob-component ((1 + r - d)
/ (u - d)) w) {0..<matur}) *
      (discounted-value r (λm. (asian-option matur)) matur w))
    (asian-option matur) matur
    by (rule CRR-market-fair-price, rule asian-borel)
  thus ?thesis using asian-effect-compute by simp
qed

end
```