# Pricing in discrete financial models 

Mnacho Echenim

September 13, 2023

## Contents

1 Generated subalgebras ..... 1
1.1 Independence between a random variable and a subalgebra. ..... 7
2 Filtrations ..... 9
2.1 Basic definitions ..... 9
2.2 Stochastic processes ..... 11
2.2.1 Adapted stochastic processes ..... 11
2.2.2 Predictable stochastic processes ..... 13
2.3 Initially trivial filtrations ..... 19
2.4 Filtration-equivalent measure spaces ..... 21
3 Martingales ..... 24
4 Discrete Conditional Expectation ..... 27
4.1 Preliminary measurability results ..... 27
4.2 Definition of explicit conditional expectation ..... 32
5 Infinite coin toss space ..... 60
5.1 Preliminary results ..... 60
5.2 Bernoulli streams ..... 66
5.3 Natural filtration on the infinite coin toss space ..... 69
5.3.1 The projection function ..... 69
5.3.2 Natural filtration locale ..... 78
5.3.3 Probability component ..... 90
5.3.4 Filtration equivalence for the natural filtration ..... 101
5.3.5 More results on the projection function ..... 104
5.3.6 Integrals and conditional expectations on the natural filtration ..... 113
5.4 Images of stochastic processes by prefixes of streams ..... 142
5.4.1 Definitions ..... 142
5.4.2 Induced filtration, relationship with filtration gener- ated by underlying stochastic process ..... 151
6 Geometric random walk ..... 171
7 Fair Prices ..... 176
7.1 Preliminary results ..... 176
7.1.1 On the almost everywhere filter ..... 177
7.1.2 On conditional expectations ..... 178
7.2 Financial formalizations ..... 181
7.2.1 Markets ..... 181
7.2.2 Quantity processes and portfolios ..... 182
7.2.3 Trading strategies ..... 191
7.2.4 Self-financing portfolios ..... 199
7.2.5 Replicating portfolios ..... 215
7.2.6 Arbitrages ..... 216
7.2.7 Fair prices ..... 222
7.3 Risk-neutral probability space ..... 240
7.3.1 risk-free rate and discount factor processes ..... 240
7.3.2 Discounted value of a stochastic process ..... 243
7.3.3 Results on risk-neutral probability spaces ..... 246
8 The Cox Ross Rubinstein model ..... 262
8.1 Preliminary results on the market ..... 262
8.2 Risk-neutral probability space for the geometric random walk ..... 281
8.3 Existence of a replicating portfolio ..... 299
9 Effective computation definitions and results ..... 343
9.1 Generation of lists of boolean elements ..... 343
9.2 Probability components for lists ..... 345
9.3 Geometric process applied to lists ..... 346
9.4 Effective computation of discounted values ..... 347
10 Pricing results on options ..... 348
10.1 Call option ..... 348
10.2 Put option ..... 350
10.3 Lookback option ..... 351
10.4 Asian option ..... 355

## 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 $M G=\operatorname{sigma}($ space $M) G$

## lemma gen-subalgebra-space:

shows space (gen-subalgebra $M$ ) $=$ space $M$
by (simp add: gen-subalgebra-def space-measure-of-conv)
lemma gen-subalgebra-sets:
assumes $G \subseteq$ sets $M$
and $A \in G$
shows $A \in$ sets (gen-subalgebra $M$ )
by (metis assms gen-subalgebra-def sets.space-closed sets-measure-of sigma-sets.Basic subset-trans)
lemma gen-subalgebra-sig-sets:
assumes $G \subseteq$ 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 $G \subseteq$ 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: $G \subseteq$ 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 ? $N \subseteq$ sets $M$ using assms by simp
qed
definition fct-gen-subalgebra $::$ 'a measure $\Rightarrow$ 'b measure $\Rightarrow\left({ }^{\prime} a \Rightarrow\right.$ ' $\left.b\right) \Rightarrow^{\prime}$ a measure where
fct-gen-subalgebra $M N X=$ gen-subalgebra $M$ (sigma-sets (space $M$ ) $\left\{X-{ }^{\prime} B \cap\right.$ $($ space $M) \mid B . B \in$ sets $N\})$

```
lemma fct-gen-subalgebra-sets:
    shows sets (fct-gen-subalgebra M N X) = sigma-sets (space M) {X -' B\cap space
M|B.B \in sets N}
unfolding fct-gen-subalgebra-def gen-subalgebra-def
proof -
    have {X -' B\cap space M | B. B\in sets N}\subseteq Pow (space M)
        by blast
    then show sets (sigma (space M) (sigma-sets (space M) {X -' B \cap space M
```



```
    by (meson sigma-algebra.sets-measure-of-eq sigma-algebra-sigma-sets)
qed
lemma fct-gen-subalgebra-space:
    shows space (fct-gen-subalgebra M NX) = 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\in sets N
    shows }X-\mp@subsup{}{}{`}B\cap(\mathrm{ space M)}\in\mathrm{ sets (fct-gen-subalgebra M NX) unfolding
fct-gen-subalgebra-def
proof -
    have f1: {X -` A\cap space M |A.A\in sets N}\subseteqPow (space M)
            by blast
    have }\existsA.X -` B\cap\mathrm{ space }M=X-\mp@subsup{}{}{`}A\cap\mathrm{ space }M\wedgeA\in\mathrm{ sets }
    by (metis assms)
    then show X -' B\cap space M sets (gen-subalgebra M (sigma-sets (space M)
{X -` A\cap space M |A.A E 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\in 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\cap space M | B. B\in sets N}\subseteq sets M (is
?L\subseteq?R)
    proof (rule sigma-algebra.sigma-sets-subset)
```

```
    show {X-' B\cap space M |B.B\in sets N}\subseteq sets M
    proof
        fix }
        assume }a\in{X-\mp@subsup{|}{}{\prime}B\cap(\mathrm{ space M)| B. B 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)
```



```
        show ?preimages \leq Pow (space M) using assms by auto
    qed
qed
lemma fct-gen-subalgebra-fct-measurable:
    assumes }X\in\mathrm{ space }M->\mathrm{ 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 MNX)
            by (simp add: fct-gen-subalgebra-space)
    show }X\in\mathrm{ space (fct-gen-subalgebra MNX)}->\mathrm{ space N
    proof -
        have }X\in\mathrm{ space M }->\mathrm{ space N using assms by simp
        thus ?thesis using speq by simp
    qed
    show }\forally\in\mathrm{ sets }N\mathrm{ .
        X -' y \cap space (fct-gen-subalgebra M NX) \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 MP
and $f \in$ measurable $P N$
shows subalgebra $P$ (fct-gen-subalgebra $M N f$ )
unfolding subalgebra-def
proof (intro conjI)
let $? M f=f c t$-gen-subalgebra $M N f$
show space ? $M f=$ 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\in 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 }\forallA\in\mathrm{ sets N.f-'A \ space P 斻s P using assms by simp
    hence {f -` A\cap (space M)|A.A\in sets N}\subseteq sets P using<space M=
space P> by auto
    hence sigma-sets (space M) {f -' A\cap (space M)|A.A\in sets N}\subseteq sets P
            by (simp add: <sigma-algebra (space P) (sets P)\rangle\langlespace M = space P>
sigma-algebra.sigma-sets-subset)
    thus ?thesis using fct-gen-subalgebra-sets }\langlef\inM \mp@subsup{->}{M}{}N\rangle\langlespace M = spac
P> assms(2)
            measurable-sets mem-Collect-eq sets.sigma-sets-subset subsetI by blast
    qed
qed
lemma fct-preimage-sigma-sets:
    assumes }X\in\mathrm{ space }M->\mathrm{ space N
    shows sigma-sets (space M) {X -' B\cap space M | B. B\in sets N } = {X -'B\cap
space M|B.B\in sets N} (is ?L = ?R)
proof
    show ?R\subseteq?L by blast
    show ?L\subseteq?R
    proof
        fix }
    assume A\in?L
    thus A\in?R
    proof (induct rule:sigma-sets.induct, auto)
        {
            fix }
            assume B\in sets N
            let ?cB = space N-B
            have ?cB \in sets N by (simp add: <B\in sets N\rangle sets.compl-sets)
            have space M - X -' B\cap space M = X -' ?c B \cap space M
            proof
                show space }M-X-\mp@subsup{|}{}{\prime}B\cap\mathrm{ space }M\subseteqX-`(\mathrm{ space }N-B)\cap\mathrm{ space M
                proof
                    fix w
                    assume w\in space M - X -` B\cap space M
                    hence }Xw\in(\mathrm{ space N-B) using assms by blast
                    thus w\inX -'(space N-B)\cap space M using }<w\in\mathrm{ space M - X -`
B\cap space M> by blast
                qed
                show }X-`(\mathrm{ space }N-B)\cap\mathrm{ space }M\subseteq\mathrm{ space }M-X-`B\cap\mathrm{ space }
                    proof
                    fix w
                    assume w\inX -' (space N-B)\cap space M
                    thus w\in space M-X -' B\cap space }M\mathrm{ by blast
                    qed
```

```
            qed
            thus \existsBa. space }M-X-`B\cap\mathrm{ space }M=X-\mp@subsup{}{}{`}Ba\cap\mathrm{ space }M\wedgeBa
sets N using <?cB \in sets N\rangle by auto
    }
        fix S::nat }=>\mathrm{ 'a set
            assume (\bigwedgei. \existsB.S i=X -' B\cap space M ^B\in sets N)
            hence ( }\foralli.\existsB.Si=X-'B\cap\mathrm{ space }M\wedgeB\in\mathrm{ sets }N\mathrm{ ) by auto
            hence }\existsf.\forallx.Sx=X-`(fx)\cap\mathrm{ space M ^(fx) E sets N
            using choice[of \lambdai B.S i=X -' B\cap space M}^BE\mathrm{ sets N] by simp
            from this obtain rep where }\foralli.Si=X -'(rep i)\cap space M ^(rep i
sets N by auto note rProp = this
            let ?uB=\bigcup i\inUNIV. rep i
            have ?uB\in sets N
                by (simp add: <\forall i.S i=X -` rep i \cap space M ^ rep i\in sets N\rangle
countable-Un-Int(1))
            have (\bigcupx.S x) = X -` ?uB\cap space M
            proof
            show }(\bigcupx.Sx)\subseteqX -`(\bigcupi. rep i)\cap space M
            proof
                    fix w
                    assume w\in(\bigcupx.S x)
                    hence }\existsx.w\inSx\mathrm{ by auto
                    from this obtain x where w\inSx}\mathrm{ by auto
                    hence w\in X -' rep x \cap space M using rProp by simp
                    hence w\in(\bigcupi.(X - '(rep i)\cap space M)) by blast
                    also have ... = X -`}(\bigcupi\mathrm{ . rep i) }\cap\mathrm{ space }M\mathrm{ by auto
                    finally show w\inX -'(\i. rep i)\cap space M .
            qed
            show X -`(\bigcupi. rep i)\cap space M\subseteq(\bigcupx.S x)
            proof
                    fix w
                    assume w\in X -`(\i. rep i) \cap space M
                    hence \existsx.w\in X -` (rep x) \cap space M by auto
                    from this obtain x where w\inX -` (rep x) \cap space M by auto
                    hence w\inSx using rProp by simp
                    thus w\in(\bigcupx.S x) by blast
            qed
        qed
```



```
N> by auto
        }
        qed
    qed
qed
lemma fct-gen-subalgebra-sigma-sets:
assumes X\in space M }->\mathrm{ space N
```


by (simp add: assms fct-gen-subalgebra-sets fct-preimage-sigma-sets)
lemma fct-gen-subalgebra-info:
assumes $f \in$ space $M \rightarrow$ space $N$
and $x \in$ space $M$
and $w \in$ space $M$
and $f x=f w$
shows $\bigwedge A$. $A \in$ sets $(f c t-g e n$-subalgebra $M N f) \Longrightarrow(x \in A)=(w \in A)$
proof -
$\{$ fix $A$
assume $A \in$ sigma-sets $($ space $M)\left\{f-{ }^{\prime} B \cap(\right.$ space $M) \mid B . B \in$ sets $\left.N\right\}$
from this have $(x \in A)=(w \in A)$
proof (induct rule:sigma-sets.induct) \{
fix $a$
assume $a \in\left\{f-{ }^{\prime} B \cap\right.$ space $M \mid B . B \in$ sets $\left.N\right\}$
hence $\exists B \in$ sets $N . a=f-{ }^{\prime} B \cap$ space $M$ by auto
from this obtain $B$ where $B \in$ sets $N$ and $a=f-{ }^{\prime} B \cap$ 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 $M$ ) $\left\{f-{ }^{\prime} B \cap\right.$ space $M \mid B . B \in$ sets $\left.N\right\}$
and $(x \in a)=(w \in a)$ note $x h=$ this
show $(x \in \operatorname{space} M-a)=(w \in \operatorname{space} M-a)$ by (simp add: assms(2)
$\operatorname{assms}(3) x h(2))$
\}
\{
fix $a$ ::nat $\Rightarrow$ 'a set
assume ( $\bigwedge i . a i \in$ sigma-sets (space $M)\left\{f-{ }^{\prime} B \cap\right.$ space $M \mid B . B \in$ sets N\})
and $(\bigwedge i .(x \in a i)=(w \in a i))$
show $\left(x \in \bigcup\left(a^{\prime} U N I V\right)\right)=\left(w \in \bigcup\left(a^{\prime} U N I V\right)\right)$ by (simp add: $\backslash \backslash i .(x \in a$
$i)=(w \in a i)\rangle)$
\}
\{show $(x \in\})=(w \in\{ \})$ by $\operatorname{simp}\}$
qed $\}$ note eqsig $=$ this
fix $A$
assume $A \in$ sets (fct-gen-subalgebra $M N f$ )
hence $A \in$ sigma-sets $($ space $M)\left\{f-{ }^{\prime} B \cap(\right.$ space $M) \mid B . B \in$ sets $\left.N\right\}$ 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 subal－ gebra．

definition（in prob－space）subalgebra－indep－var ：：（ ${ }^{\prime} a \Rightarrow$ real $) \Rightarrow{ }^{\prime}$ a measure $\Rightarrow$ bool where
subalgebra－indep－var $X N \longleftrightarrow$
X $\in$ borel－measurable $M$ \＆
（subalgebra M N）\＆
（indep－set（sigma－sets（space $M)\{X-‘ A \cap$ space $M \mid A . A \in$ sets borel $\}$ ） （ sets $N$ ））
lemma（in prob－space）indep－set－mono：
assumes indep－set $A B$
assumes $A^{\prime} \subseteq A$
assumes $B^{\prime} \subseteq B$
shows indep－set $A^{\prime} B^{\prime}$
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 \in borel-measurable M
    and }A\in\mathrm{ sets }
    shows indep-var borel X borel (indicator A)
proof ((rule indep-var-eq[THEN iffD2]), (intro conjI))
    let ?IA = (indicator A)::' }a=>\mathrm{ real
    show bm:random-variable borel X by (simp add: assms(2))
    show random-variable borel ?IA using assms indep-setD-ev2 unfolding subal-
gebra-indep-var-def by auto
    show indep-set (sigma-sets (space M) {X -' A\cap space M |A.A 的新 borel})
        (sigma-sets (space M) {?IA -` Aa \cap space M |Aa.Aa\in sets borel})
    proof (rule indep-set-mono)
        show sigma-sets (space M) {X -' }A\cap\mathrm{ space M |A.A 的ts borel }}\subseteq\mathrm{ sigma-sets
(space M) {X -' A\cap space M }|A.A\in\mathrm{ sets borel } by simp
    show sigma-sets (space M) {?IA -' B\cap space M | B. B\in sets borel }}\subseteq\mathrm{ sets }
        proof -
            have sigma-algebra (space M) (sets N) using assms
            by (metis subalgebra-indep-var-def sets.sigma-algebra-axioms subalgebra-def)
```



```
sigma-sets (space M) (sets N)
            proof (rule sigma-sets-subseteq)
                show {?IA -' B\cap space M | B. B \in sets borel }}\subseteq\mathrm{ sets N
                proof
                    fix }
                        assume }x\in{?IA -` B\cap space M |B. B 盾ets borel
                    then obtain B where B\in sets borel and x=?IA - ' B\cap space M by
auto
                thus }x\in\mathrm{ 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\cap space M | B. B\in sets borel}
\subseteq \text { sets N .}
    qed
    show indep-set (sigma-sets (space M) {X -' }A\cap\mathrm{ space M |A.A 的ts 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 MNX=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 <br> begin

### 2.1 Basic definitions

class linorder-bot $=$ linorder + bot
instantiation nat::linorder-bot
begin
instance proof qed
end
definition filtration :: 'a measure $\Rightarrow$ ('i::linorder-bot $\Rightarrow{ }^{\prime}$ 'a measure $) \Rightarrow$ bool where filtration $M F \longleftrightarrow$

```
    (\forallt. subalgebra M (F t)) ^
    (\forallst.s\leqt\longrightarrow subalgebra (Ft)(Fs))
lemma filtrationI:
    assumes }\forallt\mathrm{ . subalgebra M(Ft)
    and }\forallst.s\leqt\longrightarrow\mathrm{ subalgebra (Ft) (Fs)
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\leqt\Longrightarrow\mathrm{ subalgebra (Ft) (Fs) using assms unfolding filtration-def by
simp
locale filtrated-prob-space = prob-space +
    fixes }
    assumes filtration: filtration M F
lemma (in filtrated-prob-space) filtration-space:
    assumes }s\leq
    shows space (Fs)= space (Ft) 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}->\mathrm{ space }N\wedge(\forally\in\mathrm{ sets N.f -` y \ space }M\in\mathrm{ 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 -> space N using assms unfolding measurable-def by simp
        fix }
        assume y\in sets N
        hence f-'}\\cap\mathrm{ space M sets (F t) using assms unfolding measurable-def
            using <space (F t) = space M> by auto
        thus f-`}y\cap\mathrm{ space }M\in\mathrm{ sets }M\mathrm{ 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)\subseteq sets (F t)
            using assms(2) filtration by (simp add: filtration-def subalgebra-def)
    have sp: space (Fs)= space (Ft) by (metis filtration filtration-def subalge-
bra-def)
    thus }\x.x\in\operatorname{space (Ft)\Longrightarrowfx\in space N using assms by (simp add: mea-
surable-space)
    show }\A.A\in\mathrm{ sets }N\Longrightarrowf-`A\cap\mathrm{ space (F t) E sets (F t)
    proof -
        fix }
        assume A\in sets N
        hence f -' }A\cap\mathrm{ space (Fs) E sets (F s) using assms using measurable-sets
by blast
    hence f-' A\cap space (Fs)\in sets (F t) using subsetD[of F s F t] inc by blast
    thus f-`}A\cap\mathrm{ space (F ) ) sets (F t) using sp by simp
    qed
qed
definition disc-filtr :: 'a measure }=>\mathrm{ (nat }=>\mp@subsup{}{}{\prime}\mathrm{ 'a measure) }=>\mathrm{ bool where
    disc-filtr M F \longleftrightarrow
        (\foralln. subalgebra M (F n)) ^
        (\foralln m.n \leqm\longrightarrow subalgebra (Fm) (Fn))
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 }\forallst.s\leqt\longrightarrow\mathrm{ subalgebra (Ft)(Fs) using assms unfolding filtration-def
by simp
    show }\forallt\mathrm{ . subalgebra N (F t)
    proof
        fix }
        have subalgebra M (F t) using assms unfolding filtration-def by auto
        thus subalgebra N(Ft) using assms by (metis subalgebra-def subsetCE subsetI)
    qed
qed
```

sublocale disc-filtr-prob-space $\subseteq$ 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 $M N=\{X . \forall t .(X t) \in$ measurable $M N\}$

### 2.2.1 Adapted stochastic processes

```
definition adapt-stoch-proc where
    (adapt-stoch-proc F X N)\longleftrightarrow \longleftrightarrow (\forallt.(Xt)\in measurable (F t)N)
```

abbreviation borel-adapt-stoch-proc F X $\equiv$ adapt-stoch-proc F X borel
lemma (in filtrated-prob-space) adapted-is-dsp:
assumes adapt-stoch-proc F $X N$
shows $X \in$ stoch-procs $M N$
unfolding stoch-procs-def
by (intro CollectI, (meson adapt-stoch-proc-def assms filtration filtration-def mea-
surable-from-subalg))
lemma (in filtrated-prob-space) adapt-stoch-proc-borel-measurable:
assumes adapt-stoch-proc F $X$ N
shows $\forall n .(X n) \in$ measurable $M N$
proof
fix $n$
have $X n \in$ measurable ( $F n$ ) $N$ using assms unfolding adapt-stoch-proc-def
by $\operatorname{simp}$
moreover have subalgebra $M$ ( $F n$ ) using filtration unfolding filtration-def by
simp
ultimately show $X n \in$ 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 $\forall n$. $(X n) \in$ borel-measurable $M$
proof
fix $n$
have $X n \in$ borel-measurable ( $F n$ ) using assms unfolding adapt-stoch-proc-def
by $\operatorname{simp}$
moreover have subalgebra $M$ ( $F n$ ) using filtration unfolding filtration-def by
simp
ultimately show $X n \in$ 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 (\lambda n w.c)
unfolding adapt-stoch-proc-def
proof
    fix }
    show (\lambdaw.c) \in borel-measurable (F t) using borel-measurable-const by blast
qed
```

lemma (in filtrated-prob-space) borel-adapt-stoch-proc-add:
fixes $X::^{\prime} b \Rightarrow{ }^{\prime} a \Rightarrow\left({ }^{\prime} c::\{\right.$ second-countable-topology, topological-monoid-add $\left.\}\right)$
assumes borel-adapt-stoch-proc F X
and borel-adapt-stoch-proc F Y
shows borel-adapt-stoch-proc $F(\lambda t w . X t w+Y t w)$ unfolding adapt-stoch-proc-def
proof
fix $t$
have $X t \in$ borel-measurable ( $F t$ ) using assms unfolding adapt-stoch-proc-def
by $\operatorname{simp}$
moreover have $Y t \in$ borel-measurable ( $F t$ ) using assms unfolding adapt-stoch-proc-def
by $\operatorname{simp}$
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-sum:
fixes $A::^{\prime} d \Rightarrow{ }^{\prime} b{ }^{\prime} a \Rightarrow(' c::\{$ second-countable-topology, topological-comm-monoid-add $\}$ )
assumes $\bigwedge i . i \in S \Longrightarrow$ borel-adapt-stoch-proc $F(A i)$
shows borel-adapt-stoch-proc $F\left(\lambda t w .\left(\sum i \in S\right.\right.$. A it w) ) unfolding adapt-stoch-proc-def proof
fix $t$
have $\bigwedge i$. $i \in S \Longrightarrow A$ it borel-measurable ( $F$ t) using assms unfolding adapt-stoch-proc-def by simp
thus $\left(\lambda w .\left(\sum i \in S . A\right.\right.$ it $\left.\left.w\right)\right) \in$ borel-measurable $(F t)$ by (simp add:borel-measurable-sum) qed
lemma (in filtrated-prob-space) borel-adapt-stoch-proc-times:
fixes $X:::^{\prime} b \Rightarrow{ }^{\prime} a \Rightarrow\left({ }^{\prime} c::\{\right.$ second-countable-topology, real-normed-algebra $\left.\}\right)$
assumes borel-adapt-stoch-proc FX
and borel-adapt-stoch-proc F Y
shows borel-adapt-stoch-proc $F(\lambda t w . X t w * Y t w)$ unfolding adapt-stoch-proc-def proof
fix $t$
have $X t \in$ borel-measurable ( $F t$ ) using assms unfolding adapt-stoch-proc-def by $\operatorname{simp}$
moreover have $Y t \in$ borel-measurable ( $F t$ ) using assms unfolding adapt-stoch-proc-def by $\operatorname{simp}$

```
    ultimately show (\lambdaw.Xtw*Ytw)\inborel-measurable (F t) by simp
qed
lemma (in filtrated-prob-space) borel-adapt-stoch-proc-prod:
    fixes }A::'d d 'b ⿰' 'a ('c::{second-countable-topology, real-normed-field}
    assumes }\i.i\inS\Longrightarrow\mathrm{ borel-adapt-stoch-proc F (A i)
shows borel-adapt-stoch-proc F (\lambdatw. (\prod i\inS.A itw)) unfolding adapt-stoch-proc-def
proof
    fix }
        have }\i.i\inS\LongrightarrowA it\in\mathrm{ borel-measurable (F t) using assms unfolding
adapt-stoch-proc-def by simp
    thus (\lambdaw. (\prod i\inS.A itw))\in borel-measurable (F t) by simp
qed
```


### 2.2.2 Predictable stochastic processes

definition predict-stoch-proc where
(predict-stoch-proc $F X N) \longleftrightarrow\left(\begin{array}{ll}X & 0 \in \text { measurable }(F 0) N \wedge(\forall n .(X(S u c ~ n))\end{array}\right.$ $\in$ 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 $F(\lambda n w . c)$
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 $X:: n a t \Rightarrow{ }^{\prime} a \Rightarrow\left({ }^{\prime} c::\{\right.$ 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 $n) w) \in \operatorname{borel-measurable~}(F n)$
proof
fix $n$
have $X(S u c n) \in$ borel-measurable $(F n)$ using assms unfolding predict-stoch-proc-def
by $\operatorname{simp}$
moreover have $Y(S u c n) \in$ 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 $\operatorname{simp}$
qed
qed
lemma (in disc-filtr-prob-space) borel-predict-stoch-proc-sum:
fixes $A::^{\prime} d \Rightarrow$ nat $\Rightarrow{ }^{\prime} a \Rightarrow\left({ }^{\prime} c::\{\right.$ second-countable-topology, topological-comm-monoid-add $\left.\}\right)$
assumes $\bigwedge i . i \in S \Longrightarrow$ borel-predict-stoch-proc $F\left(\begin{array}{ll}A & i\end{array}\right)$
shows borel-predict-stoch-proc $F\left(\lambda t w .\left(\sum i \in S . A\right.\right.$ itw)) unfolding pre-
dict-stoch-proc-def
proof
show $\left(\lambda w . \sum i \in S . A i 0 w\right) \in$ 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 $\left(\lambda w .\left(\sum i \in S . A i 0 w\right)\right) \in$ borel-measurable (F 0) by (simp add:borel-measurable-sum)
qed simp
next
show $\forall n .\left(\lambda w . \sum i \in S . A i(S u c n) w\right) \in$ borel-measurable $(F n)$
proof
fix $n$
have $\bigwedge i . i \in S \Longrightarrow A i(S u c n) \in$ borel-measurable (F $n$ ) using assms unfolding
predict-stoch-proc-def by simp
thus $\left(\lambda w .\left(\sum i \in S . A i(S u c n) w\right)\right) \in$ 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 $\Rightarrow{ }^{\prime} a \Rightarrow\left({ }^{\prime} c::\{\right.$ second-countable-topology, real-normed-algebra $\left.\}\right)$
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 ( $\left.\begin{array}{ll}F & 0\end{array}\right)$
proof -
have $X 0 \in$ borel-measurable (F 0) using assms unfolding predict-stoch-proc-def
by $\operatorname{simp}$
moreover have $Y 0 \in$ borel-measurable ( $F 0$ ) using assms unfolding pre-dict-stoch-proc-def by simp
ultimately show $(\lambda w . X 0 w * Y 0 w) \in$ borel-measurable (F0) by simp
qed
next
show $\forall n .(\lambda w . X(S u c n) w * Y(S u c n) w) \in$ borel-measurable $(F n)$
proof
fix $n$
have $X(S u c n) \in$ borel-measurable $(F n)$ using assms unfolding predict-stoch-proc-def by $\operatorname{simp}$
moreover have $Y(S u c n) \in$ borel-measurable ( $F n$ ) using assms unfolding predict-stoch-proc-def by simp
ultimately show $(\lambda w . X($ Suc $n) w * Y(S u c n) w) \in$ borel-measurable $(F n)$ by $\operatorname{simp}$
qed
qed
lemma (in disc-filtr-prob-space) borel-predict-stoch-proc-prod:
fixes $A::^{\prime} d \Rightarrow$ nat $\Rightarrow{ }^{\prime} a \Rightarrow\left({ }^{\prime} c::\{\right.$ second-countable-topology, real-normed-field $\left.\}\right)$
assumes $\bigwedge i . i \in S \Longrightarrow$ borel-predict-stoch-proc $F\left(\begin{array}{ll}A & i\end{array}\right)$
shows borel-predict-stoch-proc $F\left(\lambda t w .\left(\prod i \in S . A\right.\right.$ itw)) unfolding pre-dict-stoch-proc-def
proof
show $\left(\lambda w . \prod i \in S . A i 0 w\right) \in$ borel-measurable ( $F$ O 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 $\left(\lambda w .\left(\prod i \in S . A i 0 w\right)\right) \in$ borel-measurable $\left(\begin{array}{ll}F & 0) \text { by simp }\end{array}\right.$
qed
next
show $\forall n .\left(\lambda w . \prod i \in S . A i(S u c n) w\right) \in$ borel-measurable $(F n)$
proof
fix $n$
have $\bigwedge i . i \in S \Longrightarrow A i(S u c n) \in$ borel-measurable (F $n$ ) using assms unfolding predict-stoch-proc-def by simp
thus $\left(\lambda w .\left(\prod i \in S . A i(\right.\right.$ Suc $\left.\left.n) w\right)\right) \in$ borel-measurable $(F n)$ by simp qed
qed
definition (in prob-space) constant-image where
constant-image $f=\left(\right.$ if $\exists c:::^{\prime} b::\{t 2-$ 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 $\exists c:: ' b::\{t 2$-space $\} . \forall x \in$ space $M . f x=c$
shows $\forall x \in$ space $M$. $f x=($ constant-image $f)$
proof
fix $x$
assume $x \in$ space $M$
let $? c=S O M E$ c. $\forall x \in$ space $M . f x=c$
have $f x=$ ?c using $\langle x \in$ space $M$ s someI-ex[of $\lambda c$. $\forall x \in$ space $M$. $f x=c]$ assms by blast
thus $f x=($ constant-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 \existsc::real. }\forallx\in\mathrm{ space M. fx=c
shows 0< (constant-image f)
proof -
    {
        fix }
        assume x\in space M
    hence 0<fx using assms by simp
    also have ... = constant-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-except x y = (if x=y then {} else SOME A. open A ^ x\inA A y\not\inA)
lemma open-exceptI:
    assumes (x::'b::{t1-space}) = y
    shows open (open-except x y) and x\in open-except x y and y}\not=\mathrm{ open-except }x\mathrm{ y
proof-
    have ex:\existsU. open }U\wedgex\inU\wedgey\not\inU\mathrm{ using 〈x}\not=y>\mathrm{ by (simp add:t1-space)
    let ? V = SOME A. open }A\wedgex\inA\wedgey\not\in
    have vprop: open ? V \wedge x\in?V }\\y\not\in?V\mathrm{ using someI-ex[of }\lambdaU\mathrm{ . open }U\wedge
\inU\wedgey\not\inU] ex by blast
    show open (open-except x y) by (simp add: open-except-def vprop)
    show }x\in\mathrm{ open-except x y by (metis (full-types) open-except-def vprop)
    show y\not\in open-except x y by (metis (full-types) open-except-def vprop)
qed
lemma open-except-set:
    assumes finite A
    and (x::'b::{t1-space}) &A
shows }\existsU\mathrm{ . open }U\wedgex\inU\wedgeU\capA={
proof(intro exI conjI)
    have }\forally\inA.x\not=y\mathrm{ using assms by auto
    let ? U =\bigcap y G A. open-except x y
    show open?U
    proof (intro open-INT ballI, (simp add: assms))
        fix }
        assume }y\in
        show open (open-except x y) using }\langley\inA.x\not=y\rangle\mathrm{ by (simp add: }\langley\inA
open-exceptI)
    qed
    show }x\in(\bigcapy\inA.\mathrm{ open-except }x\mathrm{ y)
    proof
        fix }
        assume }y\in
```

```
            show x\inopen-except x y using <\forally\inA. x\not= y〉 by (simp add: <y \in A〉
open-exceptI)
    qed
    have }\forally\inA.y\not\in?U\mathrm{ using }\forally\inA.x\not=y\rangle\mathrm{ open-exceptI(3) by auto
    thus (\bigcapy\inA. open-except x y) \capA={} by auto
qed
definition open-exclude-set where
open-exclude-set x A = (if ( }\exists\textrm{U}\mathrm{ . open }U\wedgeU\capA={x})\mathrm{ then SOME U. open U
\wedge U\capA={x} else {})
lemma open-exclude-setI:
    assumes }\existsU\mathrm{ . open }U\wedgeU\capA={x
shows open (open-exclude-set x A) and (open-exclude-set x A) \capA={x}
proof -
    let ?V = SOME U. open }U\wedgeU\capA={x
    have vprop: open ? V ^ ? V \capA={x} using someI-ex[of \lambdaU. open }U\wedgeU
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\capA={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 }\existsU.\mathrm{ open }U\wedgeU\capA={x
    proof -
        have }\existsU\mathrm{ . open }U\wedgex\inU\wedgeU\cap(A-{x})={
        proof (rule open-except-set)
            show finite ( }A-{x})\mathrm{ using assms by auto
            show }x\not\inA-{x}\mathrm{ by simp
            qed
            thus ?thesis using assms by auto
    qed
    thus open (open-exclude-set x A) and (open-exclude-set x A) \capA={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 $\Rightarrow\left(' i:\right.$ :linorder-bot $\Rightarrow{ }^{\prime}$ 'a measure $) \Rightarrow$ bool where init-triv-filt $M F \longleftrightarrow$ filtration $M F \wedge$ sets $(F$ bot $)=\{\{ \}$, space $M\}$

## lemma triv－measurable－cst：

fixes $f::^{\prime} a \Rightarrow^{\prime} b::\{t 2$－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 $\exists c$ ．$c \in f^{\prime}($ space $N)$ by auto
from this obtain $c$ where $c \in f^{\prime}($ 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 $\operatorname{simp}$
show $(f-‘ V) \cap$ space $N=$ space $N$
proof（rule ccontr）
assume $(f-' V) \cap$ space $N \neq$ space $N$
hence $\left(f-{ }^{`} V\right) \cap$ space $N=\{ \}$ using assms $\prec\left(f-{ }^{`} V\right) \cap$ space $N \in$ sets
$N>$ by simp
thus False using $\langle c \in V\rangle$ using $\left\langle c \in f^{‘}\right.$ space $\left.N\right\rangle$ by blast
qed
qed
have $\left(\left(f-{ }^{\prime} U\right) \cap\right.$ space $\left.N\right) \cap((f-‘ V) \cap$ space $N)=\{ \}$ using〈 $U \cap V=\{ \}$ 〉 by auto
moreover have $\left(f-{ }^{\prime} U\right) \cap$ space $N \in$ sets $N$ using assms 〈open $U$ 〉 unfolding measurable－def by simp
ultimately have $\left(f-{ }^{`} U\right) \cap$ space $N=\{ \}$ using assms $\prec\left(f-{ }^{`} V\right) \cap$ space $N$ $=$ space $N>$ by simp
thus False using $\langle f x \in U\rangle\langle x \in$ space $N\rangle$ 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-fil M F
sublocale trivial-init-filtrated-prob-space \subseteq filtrated-prob-space
    using info-filtration unfolding init-triv-fll-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 \subseteq trivial-init-filtrated-prob-space
proof unfold-locales
    show init-triv-filt M F using info-disc-filtr bot-nat-def unfolding init-triv-fil-def
disc-filtr-def
    by (simp add: filtrationI)
qed
sublocale triv-init-disc-filtr-prob-space \subseteq 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 \existsc.\forallw\in space M. ((x 0 w)::real) =c
proof -
    have space M = space (F 0) using filtration
        by (simp add: flltration-def subalgebra-def)
    moreover have \existsc.\forallw\in 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-fil-def bot-nat-def)
        show x 0 forel-measurable (F 0) using assms by (simp add: adapt-stoch-proc-def)
            show space M\not={} 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 FMN
    shows filtration NF using assms unfolding filt-equiv-def filtration-def subalge-
bra-def
    by (metis sets-eq-imp-space-eq)
```

lemma (in filtrated-prob-space) AE-borel-eq:
fixes $f::^{\prime} a \Rightarrow$ real
assumes $f \in$ borel-measurable ( $F t$ )
and $g \in$ borel-measurable ( $F t$ )
and $A E w$ 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 $M . f w \neq g w\} \in \operatorname{sets}(F t)$
proof -
define minus where minus $=(\lambda w .(f w)-(g w))$
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
$\langle$ minus $\in$ borel-measurable $(F$ t) > by auto
ultimately have $\{w \in \operatorname{space}(F t) .0<$ minus $w\} \cup\{w \in \operatorname{space}(F t)$. minus $w$
$<0\} \in \operatorname{sets}(F t)$ by simp
moreover have $\{w \in \operatorname{space}(F t) . f w \neq g w\}=\{w \in \operatorname{space}(F t) .0<$ minus
$w\} \cup\{w \in$ space $(F t)$. minus $w<0\}$
proof
show $\{w \in \operatorname{space}(F t) . f w \neq g w\} \subseteq\{w \in \operatorname{space}(F t) .0<$ minus $w\} \cup\{w$
$\in \operatorname{space}(F t)$. minus $w<0\}$

## proof

fix $w$
assume $w \in\{w \in \operatorname{space}(F t) . f w \neq g w\}$
hence $w \in \operatorname{space}(F t)$ and $f w \neq g w$ by auto
thus $w \in\{w \in \operatorname{space}(F t)$. $0<$ minus $w\} \cup\{w \in \operatorname{space}(F t)$. minus $w<$
0\} unfolding minus-def
by (cases $f w<g w$ ) auto
qed
have $\{w \in \operatorname{space}(F t) .0<$ minus $w\} \subseteq\{w \in \operatorname{space}(F t) . f w \neq g w\}$
unfolding minus-def by auto
moreover have $\{w \in \operatorname{space}(F t)$. minus $w<0\} \subseteq\{w \in \operatorname{space}(F t)$. $f w \neq$ $g w\}$ unfolding minus-def by auto
ultimately show $\{w \in \operatorname{space}(F t) .0<$ minus $w\} \cup\{w \in$ space $(F t)$. minus $w<0\} \subseteq\{w \in \operatorname{space}(F t) . f w \neq g w\}$
by $\operatorname{simp}$
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 $f::^{\prime} a \Rightarrow$ real
assumes filt-equiv $F M N$
and $f \in$ borel-measurable ( $F t$ )
and $g \in$ borel-measurable ( $F t$ )
and $A E w$ in $M . f w=g w$
shows $A E w$ in $N . f w=g w$
proof -
have set0: $\{w \in$ space $M . f w \neq g w\} \in \operatorname{sets}(F t) \wedge$ 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 MF 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.fa\not=ga}
        using AE-iff-measurable «emeasure N {w\in space M.fw\not=gw}=0\rangle\langle{w
< space M. fw\not=gw}\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 FM N
    and sigma-finite-subalgebra MG
shows sigma-finite-subalgebra NG unfolding sigma-finite-subalgebra-def
proof
    show subalgebra NG
    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 NG) 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\in measurable M P
shows f\in measurable N P using assms unfolding filt-equiv-def measurable-def
proof -
    assume a1: sets M = sets N^ Filtration.filtration M F ^(\forallt A. A \in sets (F
t)\longrightarrow(emeasure MA=0)=(emeasure NA=0))
    assume a2: f}\in{f\in\mathrm{ space }M->\mathrm{ space P.}\forally\in\mathrm{ sets P. f-' }y\cap\mathrm{ space }M\in\mathrm{ sets
M}
    have space N = space M
        using a1 by (metis (lifting) sets-eq-imp-space-eq)
    then show f}\in{f\in\mathrm{ space N}->\mathrm{ space P.}\forallC\in\mathrm{ sets P. f-''C \ space N }\in\mathrm{ 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 fil-equiv-def by blast
```

end

## 3 Martingales

```
theory Martingale imports Filtration
begin
definition martingale where
    martingale M F X \longleftrightarrow
        (filtration M F) ^(\forallt. integrable M (Xt)) ^(borel-adapt-stoch-proc F X) ^
        (\forallts.t\leqs\longrightarrow(AE w in M.real-cond-exp M (Ft) (X s) w = X tw))
```

lemma martingale $A E$ :
assumes martingale M FX
and $t \leq s$
shows $A E w$ in $M$. real-cond-exp $M(F t)(X s) w=(X t) w$ using assms un-
folding martingale-def by simp
lemma martingale-add:
assumes martingale MFX
and martingale M F Y
and $\forall m$. sigma-finite-subalgebra $M(F m)$
shows martingale $M F(\lambda n w . X n w+Y n w)$ unfolding martingale-def
proof (intro conjI)
let ?sum $=\lambda n w . X n w+Y n w$
show $\forall n$. integrable $M(\lambda w . X n w+Y n w)$
proof
fix $n$
show integrable $M(\lambda w . X n w+Y n w)$
by (metis Bochner-Integration.integrable-add assms(1) assms(2) martin-
gale-def)
qed
show $\forall n m . n \leq m \longrightarrow(A E w$ in $M$. real-cond-exp $M(F n)(\lambda w . X m w+Y$ $m w) w=X n w+Y n w)$
proof (intro allI impI)
fix $n:: / b$
fix $m$
assume $n \leq m$
show $A E w$ in $M$. real-cond- $\exp M(F n)(\lambda w . X m w+Y m w) w=X n w$
$+Y n w$
proof -
have integrable $M(X \mathrm{~m})$ using assms unfolding martingale-def by simp
moreover have integrable $M$ ( $Y$ m) using assms unfolding martingale-def
by $\operatorname{simp}$
moreover have sigma-finite-subalgebra $M$ (F n) using assms by simp ultimately have $A E w$ in $M$. real-cond-exp $M(F n)(\lambda w . X m w+Y m w)$

```
w =
        real-cond-exp M (F n) (X m)w + real-cond-exp M (F n) (Ym)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 (Fn) (X m)w=X n w using
\n\leqm> assms
            unfolding martingale-def by simp
    moreover have AE w in M. real-cond-exp M (Fn) (Ym)w=Ynwusing
<n\leqm> 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 (\lambdanw. X nw+Ynw) unfolding adapt-stoch-proc-def
    proof
        fix n
    show (\lambdaw.X nw+Ynw)\in 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 (\foralln. integrable M (X n))
    and filtration M F
    and }\forallm\mathrm{ . sigma-finite-subalgebra M (F m)
    and \forallm.X m}\in\mathrm{ borel-measurable (F m)
    and (\foralln.AE w in M. real-cond-exp M (Fn) (X (Suc n)) w = (X n)w)
shows martingale M F X unfolding martingale-def
proof (intro conjI)
    have }\forallkm.k\leqm\longrightarrow(AE w in M. real-cond-exp M (F (m-k)) (Xm) w
X(m-k)w)
    proof (intro allI impI)
    fix m
    fix k::nat
    show }k\leqm\LongrightarrowAEw in M. real-cond-exp M (F (m-k)) (X m)w=X (m-k
w
    proof (induct k)
            case 0
            have Xm}\in\mathrm{ borel-measurable ( }Fm\mathrm{ ) 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 (Fm) (X m)w=Xmw
                    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)) (Xm)) 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)) (Xm)) 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)) (Xm)X(m-k)]
        borel-measurable-cond-exp[of MF (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 \foralln m. n\leqm\longrightarrow(AE w in M. real-cond-exp M (Fn) (Xm)w=X nw)
    by (metis diff-diff-cancel diff-le-self)
    show }\forallt\mathrm{ . 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 }\forallt\mathrm{ . sigma-finite-subalgebra M (Ft)
and filtration M F
shows martingale M F (\lambdanw.c) unfolding martingale-def
proof (intro allI conjI impI)
    show filtration M F using assms by simp
    {
        fix }
        show integrable M (\lambdaw.c) by simp
    }
    fix t::'b
    fix }
    assume t\leqs
    show AE w in M. real-cond-exp M (Ft) (\lambdaw.c) w=c
    by (intro sigma-finite-subalgebra.real-cond-exp-F-meas, (auto simp add: assms))
}
```

show borel-adapt-stoch-proc $F$ ( $\lambda n$ 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}\equiv\mathrm{ countable (range f)
definition point-measurable where
    point-measurable MS S 三(f`(space M)\subseteqS)^(\forallr\in(range f) \capS.f-`{r}\cap
(space M) \in sets M)
lemma singl-meas-if:
    assumes f}\in\mathrm{ space }M->\mathrm{ space N
    and }\forallr\in\mathrm{ range }f\cap\mathrm{ space N. ヨAє sets N. range f}\capA={r
shows point-measurable (fct-gen-subalgebra MNf) (space N) f}\mathrm{ unfolding point-measurable-def
proof
    show f'space (fct-gen-subalgebra MNf)\subseteq space N using assms
        by (simp add: Pi-iff fct-gen-subalgebra-space image-subsetI)
    show (\forallr\inrange f\cap space N.f-`}{r}\cap\mathrm{ space (fct-gen-subalgebra MNf)}
sets (fct-gen-subalgebra M Nf))
    proof
        fix r
        assume r\in range f \cap space N
        hence }\existsA\in\mathrm{ sets N. range f}\capA={r} using assms by blas
        from this obtain }A\mathrm{ where AG sets N and range f }\capA={r} by auto not
Aprops = this
            hence }f-'A=f-'{r} by aut
            hence f-'A\cap space M=f-'{r}\cap space (fct-gen-subalgebra M Nf) by (simp
add: fct-gen-subalgebra-space)
    thus f-'{r}\cap space (fct-gen-subalgebra MNf)\in sets (fct-gen-subalgebra M
Nf)
            using Aprops fct-gen-subalgebra-sets-mem[of A NfM] by simp
    qed
qed
```

```
lemma meas-single-meas:
    assumes f\in measurable MN
    and }\forallr\in\mathrm{ range }f\cap\mathrm{ space N. }\existsA\in\mathrm{ sets N. range f}\capA={r
shows point-measurable M (space N) f
proof -
    have subalgebra M(fct-gen-subalgebra M Nf) using assms fct-gen-subalgebra-is-subalgebra
by blast
    hence sets (fct-gen-subalgebra MNf)\subseteq sets M by (simp add: subalgebra-def)
    moreover have point-measurable (fct-gen-subalgebra M Nf) (space N)f}\mathbf{|
assms singl-meas-if
    by (metis (no-types, lifting) Pi-iff measurable-space)
    ultimately show ?thesis
    proof -
    obtain bb :: 'a measure }=>\mp@subsup{}{}{\prime}b\mathrm{ set }=>('a=>'b)=> 'b where
        f1: \forallm Bf. (\neg point-measurable m Bf\veef'space m\subseteqB\wedge(\forallb.b\not\in range f
\capB\veef-'{b}\cap space m\in sets m))}\wedge(\negf`\mathrm{ space m}\subseteqB\veebbmBf\inrange 
\capB\wedgef-'{bbmBf}\cap space m\not\in sets m\vee point-measurable m Bf)
            by (metis (no-types) point-measurable-def)
    moreover
    { assume f-`{bb M (space N) f} \cap space (fct-gen-subalgebra MNf)\in sets
(fct-gen-subalgebra M Nf)
            then have f-'{bb M (space N)f}\cap space M \in sets(fct-gen-subalgebra M
Nf
            by (metis <subalgebra M (fct-gen-subalgebra MNf)> subalgebra-def)
        then have f-`{bb M (space N)f}\cap space M \in sets M
            using <sets (fct-gen-subalgebra MNf)\subseteq sets M> by blast
        then have f`}\mathrm{ space }M\subseteq\mathrm{ space }N\wedgef-`{bbM(\mathrm{ space N)f} }\cap\mathrm{ space }M
sets M
            using f1 by (metis <point-measurable (fct-gen-subalgebra M Nf) (space N)
f\rangle\langlesubalgebra M (fct-gen-subalgebra M N f)\rangle subalgebra-def)
        then have ?thesis
            using f1 by metis }
    ultimately show ?thesis
        by (metis (no-types)<point-measurable (fct-gen-subalgebra MNf) (space N)
f\rangle\langlesubalgebra M (fct-gen-subalgebra M Nf)\rangle subalgebra-def)
    qed
qed
```

definition countable-preimages where
countable-preimages $B Y=(\lambda n$. if $(($ infinite $B) \vee($ finite $B \wedge n<\operatorname{card} B))$ then $Y-‘\{($ from-nat-into $B) n\}$ else $\})$
lemma count-pre-disj:
fixes $i:: n a t$
assumes countable $B$
and $i \neq j$
shows (countable-preimages BY) $i \cap($ countable-preimages $B Y) j=\{ \}$
proof (cases (countable-preimages $B Y) i=\{ \} \vee($ countable-preimages $B Y) j$ $=\{ \}$ )
case True
thus ?thesis by auto
next
case False
hence $Y-‘\{($ from-nat-into $B) i\} \neq\{ \} \wedge Y-‘\{($ from-nat-into $B) j\} \neq\{ \}$
unfolding countable-preimages-def by meson
have (infinite $B) \vee($ finite $B \wedge i<\operatorname{card} B \wedge j<\operatorname{card} B)$ using False unfolding
countable-preimages-def
by meson
have (from-nat-into B) $i \neq($ 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 $f 1: \forall A f n$. if infinite $A \vee$ finite $A \wedge n<$ card $A$ then countable-preimages
Afn=f-'\{from-nat-into $\left.A n::^{\prime} a\right\}$ else countable-preimages $A f n=\left(\{ \}::^{\prime} b\right.$ set $)$ by (meson countable-preimages-def)
then have f2: infinite $B \vee$ finite $B \wedge i<\operatorname{card} B$
by (metis (no-types) False)
have infinite $B \vee$ finite $B \wedge j<\operatorname{card} B$
using $f 1$ by (meson False)
then show ?thesis
using f2 f1〈from-nat-into $B i \neq$ from-nat-into $B j$ by fastforce
qed
qed
lemma count-pre-surj:
assumes countable $B$
and $w \in Y-{ }^{‘} B$
shows $\exists i . w \in($ countable-preimages $B Y) i$
proof (cases finite B)
case True
have $\exists i<\operatorname{card} B$. (from-nat-into $B) i=Y w$
by (metis True assms(1) assms(2) bij-betw-def from-nat-into-to-nat-on im-
age-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 $w \in$ (countable-preimages $B Y$ ) $i$
by (simp add: countable-preimages-def)
thus $\exists i . w \in($ countable-preimages $B Y) i$ by auto
next
case False
hence $\exists i$. (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 $w \in$ (countable-preimages $B Y$ ) $i$

```
    by (simp add: False countable-preimages-def)
    thus }\existsi.w\in(countable-preimages B Y) i by aut
qed
lemma count-pre-img:
    assumes }x\in(\mathrm{ countable-preimages }BY)
    shows Yx=(from-nat-into B) n
proof -
    have x\in 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=(\bigcup i.(countable-preimages B Y) i)
proof (cases B={})
    case False
    have Y-'B\subseteq(\bigcup i. (countable-preimages B Y) i)
        by (simp add: assms count-pre-surj subset-eq)
    moreover have (U i. (countable-preimages B Y) i)\subseteqY - 'B
    proof -
    have f1:\forallb A f n. (b::'b) & countable-preimages A f n\vee (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 }\foralli\mathrm{ . (countable-preimages B Y) i={} unfolding countable-preimages-def
by simp
    hence (U i. (countable-preimages B Y) i)={} by auto
    moreover have Y-'}B={}\mathrm{ using True by simp
    ultimately show ?thesis by simp
qed
lemma count-pre-meas:
    assumes point-measurable M (space N) Y
    and B\subseteq space N
    and countable B
    shows }\foralli\mathrm{ . (countable-preimages }BY)i\cap\mathrm{ space M sets M
proof
    fix }
```

```
have \(Y-{ }^{`} B=(\bigcup i\). (countable-preimages \(\left.B Y) i\right)\) 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 \(\langle y \in\) countable-preimages \(B Y\) i〉count-pre-img)
    hence \(Y y \in\) space \(N\)
        by (metis UNIV-I UN-I \(\langle y \in\) countable-preimages \(B \quad Y\) i〉 \(\langle Y-' B=(\bigcup\)
i. (countable-preimages \(B Y\) ) i)〉 assms(2) empty-iff from-nat-into subsetCE vim-
age-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 \(M \rightarrow\) space \(N \wedge\left(\forall y \in\right.\) sets \(N . f-{ }^{`} y \cap\) space \(M \in\) sets \(\left.M\right)\)
    proof
        show \(f \in\) space \(M \rightarrow\) 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 ? im \(Y=\) range \(f \cap y\)
        have \(f-' y=f-\) '? im \(Y\) by auto
        moreover have countable ?im \(Y\) using assms unfolding disc-fct-def by auto
        ultimately have \(f-{ }^{\prime} y=(\bigcup i\). (countable-preimages ? im \(Y f\) ) \(i\) ) using assms
count-pre-union-img by metis
    hence yeq: \(f-{ }^{`} y \cap\) space \(M=(\bigcup\). \(((\) countable-preimages ? \(\operatorname{im} Y f) i) \cap\)
space \(M\) ) by auto
    have \(\forall i\). countable-preimages ? im \(Y f i \cap\) space \(M \in\) sets \(M\)
            by (metis \(\langle\) countable (range \(f \cap y\) ) 〉〈y sets \(N\rangle\) assms(2) inf-le2 le-inf-iff
count-pre-meas sets.Int-space-eq1)
    hence \((\bigcup\) i. \(((\) countable-preimages ?im \(Y f) i) \cap\) 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-preimages $\left.B Y) i\right)$ using assms
by (simp add: count-pre-union-img)
hence $Y-‘ B \cap$ space $M=(\bigcup i$. ((countable-preimages $B Y) i \cap$ space $M))$
by auto
have $\forall i$. (countable-preimages $B Y$ ) $i \cap$ space $M \in$ sets $M$ using assms by (simp add: count-pre-meas)
hence $(\bigcup i$. ((countable-preimages $B Y) i \cap$ space $M)) \in$ sets $M$ by blast
show ?thesis
using 〈( $\bigcup$ i. countable-preimages $B Y i \cap$ space $M) \in$ sets $M\rangle\left\langle Y-{ }^{\prime} B \cap\right.$ space $M=(\bigcup i$. countable-preimages $B Y i \cap$ space $M)\rangle$ 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 = (\lambda y. if measure M ((Y-'{y}) \cap space M) = 0 then 0 else
        ((integral L}M(\lambdaw.((Xw)*(\mathrm{ indicator }((Y-`{y})\cap\mathrm{ space M) w))))/(measure
M((Y-`{y})\cap 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 }\forallw\in\mathrm{ space M. 0 \ X w
shows }\forallw\in\mathrm{ space M. O < (expl-cond-expect MYX)w
proof
    fix }
    assume space: w\in space M
    show 0\leq(expl-cond-expect M Y X)w
    proof (cases measure M ((Y-'{{Yw})\cap space M)=0)
        case True
        thus 0\leq(expl-cond-expect M Y X) w unfolding expl-cond-expect-def img-dce-def
```

        hence \(Y-‘\{Y w\} \cap\) space \(M \in\) sets \(M\) using measure-notin-sets by blast
    let ? ind \(A=((\lambda x\). indicator \(((Y-‘\{Y w\}) \cap\) space \(M) x))\)
    have \(\forall w \in\) space \(M .0 \leq(X w) *(? i n d A w)\) by (simp add: assms)
    hence \(0 \leq\left(\right.\) integral \(^{L} M(\lambda w .((X w) *(? i n d A w)))\) by simp
    moreover have (expl-cond-expect \(M Y X) w=\left(\right.\) integral \(^{L} M(\lambda w .((X w) *\)
    $(?$ ?ind $A w))) /($ 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 MYXw=expl-cond-expect MYXy
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 \(\operatorname{pr} Y\) where \(p r Y=\) 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 MYZw
    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 ?ind $A=$ indicator $((Y-‘\{Y w\}) \cap$ space $M)::\left({ }^{\prime} a \Rightarrow\right.$ real $)$
have integrable $M(\lambda x . X x *$ ? indA $x)$
using $\langle Y-‘\{Y w\} \cap$ space $M \in$ sets $M\rangle$ assms(1) integrable-real-mult-indicator by blast
moreover have integrable $M(\lambda x . Z x *$ ? ind $A x)$
using $\langle Y-‘\{Y w\} \cap$ space $M \in$ sets $M\rangle \operatorname{assms}(2)$ integrable-real-mult-indicator by blast
ultimately have integral ${ }^{L} M(\lambda x . X x *$ ? ind $A x+Z x *$ ? ind $A x)=$ integral $^{L}$ $M(\lambda x . X x *$ ? ind $A x)+$ integral $^{L} M(\lambda x . Z x *$ ? ind $A x)$
using Bochner-Integration.integral-add by blast
moreover have $\forall x \in$ space $M . X x *$ ? indA $x+Z x *$ ? indA $x=(X x+Z$ $x) *$ ? ind $A x$ by (simp add: indicator-def)
ultimately have fsteq: integral ${ }^{L} M(\lambda x .(X x+Z x) *$ ? indA $x)=$ integral $^{L}$ $M(\lambda x . X x *$ ? ind $A x)+$ integral $^{L} M(\lambda x . Z x *$ ? ind $A x)$
by (metis (no-types, lifting) Bochner-Integration.integral-cong)
have integral ${ }^{L} M(\lambda x .(X x+Z x) *$ ? ind $A x / p r Y)=$ integral $^{L} M(\lambda x .(X x$ $+Z x) *$ ? ind $A x) / p r Y$
by $\operatorname{simp}$
also have $\ldots=$ integral $^{L} M(\lambda x . X x *$ ? indA $x) / p r Y+$ integral $^{L} M(\lambda x . Z x$ * ? indA $x) / p r Y$ using fsteq
by (simp add: add-divide-distrib)
also have $\ldots=$ integral $^{L} M(\lambda x . X x *$ ?indA $x / p r Y)+$ integral $^{L} M(\lambda x . Z x$ *?ind $A x / p r Y$ ) by auto
finally have integral ${ }^{L} M(\lambda x .(X x+Z x) *$ ? indA $x / p r Y)=$ integral $^{L} M$ $(\lambda x . X x *$ ? ind $A x / p r Y)+$ integral $^{L} M(\lambda x . Z x *$ ? indA $x / p r Y)$.
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 $\forall w \in$ space $M$. expl-cond-expect $M Y(\lambda x . X x-Z x) w=$ expl-cond-expect
MYXw-expl-cond-expect MYZw
proof
fix $w$
assume $w \in$ space $M$
define $p r Y$ where $p r Y=$ 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 MYZw
proof (cases pr $Y=0$ )
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 $p r Y$-def using mea-sure-notin-sets by blast
let ? indA $=$ indicator $\left(\left(Y-{ }^{\prime}\{Y w\}\right) \cap\right.$ space $\left.M\right)::\left({ }^{\prime} a \Rightarrow\right.$ real $)$
have integrable $M(\lambda x . X x *$ ? indA $x)$
using $\langle Y-‘\{Y w\} \cap$ space $M \in$ sets $M\rangle$ assms(1) integrable-real-mult-indicator by blast
moreover have integrable $M(\lambda x . Z x *$ ? ind $A x)$
using $\langle Y-‘\{Y w\} \cap$ space $M \in$ sets $M\rangle$ assms(2) integrable-real-mult-indicator by blast
ultimately have integral ${ }^{L} M(\lambda x . X x *$ ? indA $x-Z x *$ ? ind $A x)=$ integral $^{L}$ $M(\lambda x . X x *$ ? ind $A x)-$ 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 $x=(X x-Z$ $x) *$ ? ind $A x$
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 $x / p r Y)=$ integral $^{L} M(\lambda x .(X x$ $-Z x) *$ ? ind $A x) / p r Y$
by $\operatorname{simp}$
also have $\ldots=$ integral $^{L} M(\lambda x . X x *$ ? indA $x) / p r Y-$ integral $^{L} M(\lambda x . Z x$ * ? ind $A x) / p r Y$ using fsteq
by (simp add: diff-divide-distrib)
also have $\ldots=$ integral $^{L} M(\lambda x . X x *$ ? indA $x / p r Y)-$ integral $^{L} M(\lambda x . Z x$ * ? ind $A x / p r Y)$ by auto
finally have integral ${ }^{L} M(\lambda x .(X x-Z x) *$ ? ind $A x / p r Y)=$ integral $^{L} M$
( $\lambda x . X x *$ ? ind $A x / p r Y)-$ integral $^{L} M(\lambda x . Z x *$ ? indA $x / p r Y)$.
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 $? C=\left\{y \in\left(Y^{‘}(\right.\right.$ space $\left.M)\right) \cap($ space $N) . P($ img-dce $\left.M Y X y)\right\}$
have space $M \subseteq U N I V$ by simp
hence $Y^{\prime}($ space $M) \subseteq$ range $Y$ by auto
hence countable ( $Y^{\prime}($ space $\left.M)\right)$ 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\inD
    hence w\in space M}\wedgeYw\in(\mathrm{ space N)}\wedge(P(\mathrm{ 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 }Yw\in?C\mathrm{ using }<w\in\mathrm{ space }M\wedgeYw\in\mathrm{ space N}\wedge \P(expl-cond-expec
M Y X w)> by blast
    thus w\in(\bigcup b\in?C. Y-‘{b})\cap space M
        using }\langlew\in\mathrm{ space }M\wedgeYw\in\mathrm{ space }N\wedgeP(\mathrm{ expl-cond-expect M Y X w)>
by blast
    qed
    show (\bigcupb\in?C. Y-`{b})\cap space M\subseteqD
    proof
        fix }
        assume w\in(\bigcup b\in?C. Y-`{b})\cap space M
        from this obtain b}\mathrm{ where b}b\inC^^w\inY-‘{b} by auto note bprops = thi
        hence Yw=b by auto
        hence Y w\in space N using bprops by simp
        show }w\in
            by (metis (mono-tags,lifting) IntE <Y w = b\rangle\langlew\in(\bigcupb\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 }\forallb\in?C.Y-`{b}\cap\mathrm{ space M sets M using assms unfolding point-measurable-def
by auto
    hence ( U b\in ?C. Y-‘{b}\cap space M) \in sets M using <countable ?C` by blast
    thus ?thesis
        using }\langleD=(\bigcupb\in?C.Y -`{b}\cap space M)> by blas
qed
lemma expl-cond-expect-prop-sets2:
    assumes disc-fct Y
    and point-measurable (fct-gen-subalgebra MNY) (space N) Y
    and D={w\in space M.Yw\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 YX y)}
    have space M\subseteqUNIV by simp
    hence }\mp@subsup{Y}{}{\prime}(\mathrm{ space }M)\subseteq\mathrm{ range Y by auto
    hence countable ( }\mp@subsup{Y}{}{\prime}(\mathrm{ 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=(\bigcupb\in?C.Y-'{b})\cap space 
    proof
    show }D\subseteq(\bigcupb\in?C.Y-'{b})\cap space 
    proof
```

```
    fix w
    assume w\inD
    hence w\in space M}\wedgeYw\in(\mathrm{ space N)}\wedge(P(\mathrm{ 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 }Yw\in?C\mathrm{ using }<w\in\mathrm{ space }M\wedgeYw\in\mathrm{ space N}\wedge \P(expl-cond-expec
M Y X w)> by blast
    thus w\in(\bigcup b\in?C. Y-‘{b})\cap space M
        using }\langlew\in\mathrm{ space }M\wedgeYw\in\mathrm{ space }N\wedgeP(\mathrm{ expl-cond-expect M Y X w)>
by blast
    qed
    show (\bigcupb\in?C. Y-`{b})\cap space M\subseteqD
    proof
        fix }
        assume w\in(\bigcup b\in?C. Y-`{b})\cap space M
        from this obtain b}\mathrm{ where b}b\inC^^w\inY-‘{b} by auto note bprops = thi
        hence Yw=b by auto
        hence Y w\in space N using bprops by simp
        show }w\in
            by (metis (mono-tags,lifting) IntE <Y w = b\rangle\langlew\in(\bigcupb\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 }\forallb\in\mathrm{ ?C. Y-'{b} 
assms unfolding point-measurable-def by auto
    hence ( }\bigcupb\in?C.Y-{ {b}\cap space M)\in sets (fct-gen-subalgebra M N Y) usin
countable ?C` by blast
    thus ?thesis
        using <D = (\bigcupb\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\subseteqUNIV by simp
    show }\forallr\in\mathrm{ range (expl-cond-expect M Y X) }\cap\mathrm{ UNIV. expl-cond-expect M Y X
_'{r}\cap space M Sets M
        proof
        fix r
        assume r\in range (expl-cond-expect M Y X) \cap UNIV
        let ?D = {w\in space M. Yw\in space N}\wedge(expl-cond-expect MYXw)=r
        have ?D }\in\mathrm{ sets M using expl-cond-expect-prop-sets[of Y M N ?D dx. x = r
X] using assms by simp
            moreover have expl-cond-expect M Y X -`{r}\cap space M = ?D
            proof
                show expl-cond-expect M Y X - {r}\cap space M\subseteq?D
                proof
                    fix }
                    assume w\in expl-cond-expect M Y X -'{r}\cap space M
                hence Yw\in space N
                    by (meson IntD2 assms(1) assms(2) disct-fct-point-measurable measur-
able-space)
                thus w\in?D
                    using }\langlew\in\mathrm{ expl-cond-expect M Y X -`{r} }\cap\mathrm{ space M> by blast
                qed
                show ?D\subseteq expl-cond-expect M Y X -'{r}\cap space M
                proof
                    fix w
                    assume w\in?D
                thus w\in expl-cond-expect M Y X -'{r}\cap space M by blast
                qed
            qed
            ultimately show expl-cond-expect M Y X -`{r}\cap space M 的的 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) \in borel-measurable M using expl-cond-expect-point-meas[of
```

by (simp add: expl-cond-expect-disc-fct)

```
lemma expl-cond-exp-borel:
    assumes }Y\in\mathrm{ space }M->\mathrm{ space N
    and disc-fct Y
    and }\forallr\in\mathrm{ range }Y\cap\mathrm{ space N. }\existsA\in\mathrm{ sets N. range }Y\capA={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 MNY) (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. Yw \in space N^(expl-cond-expect MYXw)\inS}
    proof
            show expl-cond-expect M Y X -`S \cap space (fct-gen-subalgebra M N Y)\subseteq
{w\in space M.Yw\in space N\wedge expl-cond-expect MYXw\inS}
            proof
                fix }
                assume asm: x expl-cond-expect M Y X -' S \cap space (fct-gen-subalgebra
MNY)
            hence expl-cond-expect MYX x S S by auto
            moreover have x\in space M using asm by (simp add:fct-gen-subalgebra-space)
                ultimately show }x\in{w\in\mathrm{ space M.Yw
YXw\inS} using assms by auto
            qed
            show {w\in space M.Yw\in space N}\wedge\mathrm{ expl-cond-expect M YXw,S}}
expl-cond-expect M Y X -`}S\cap\mathrm{ space (fct-gen-subalgebra M NY)
            proof
            fix }
            assume asm2: }x\in{w\in\mathrm{ space M. Y w space N}\wedge expl-cond-expect M Y
Xw\inS}
            hence x\in space (fct-gen-subalgebra M NY) 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\mathrm{ expl-cond-expect M Y X - 'S }\cap\mathrm{ space (fct-gen-subalgebra
                MNY) 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 $B \subseteq$ space $N$
and countable $B$
shows ( $\lambda w$. expl-cond-expect MYX $w *$ indicator (countable-preimages $B Y n$ $\cap$ space $M) w) \in$ borel-measurable $M$
proof -
have countable-preimages $B Y n \cap$ space $M \in$ sets $M$ using assms by (auto simp add: count-pre-meas)
have (expl-cond-expect $M Y X) \in$ 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 BYn space $M$ ) ) $\in$ borel-measurable M
using «countable-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
lemma (in finite-measure) dce-prod:
assumes point-measurable $M($ space $N) Y$
and integrable $M X$
and $\forall w \in$ space $M .0 \leq X w$
shows $\forall w .(Y w) \in$ space $N \longrightarrow$ (expl-cond-expect $M Y X) w$ * measure $M((Y$ $-'\{Y w\}) \cap$ space $M)=$ integral $^{L} M(\lambda y .(X y) *($ indicator $((Y-‘\{Y w\}) \cap$ space M) $y)$ )
proof (intro allI impI)
fix $w$
assume $Y w \in$ space $N$
let ?ind $Y=(\lambda y$. indicator $((Y-‘\{Y w\}) \cap$ space $M) y)::^{\prime} a \Rightarrow$ real
show expl-cond-expect $M Y X w *$ measure $M((Y-‘\{Y w\}) \cap$ space $M)=$
integral ${ }^{L} M\left(\lambda y .\binom{X}{y} *\right.$ ?ind $\left.Y y\right)$
proof (cases $A E$ y in $M$. ?ind $Y y=0$ )
case True
hence emeasure $M((Y-‘\{Y w\}) \cap$ space $M)=0$
proof -
have $A E y$ in $M . y \notin Y-'\{Y w\} \cap$ space $M$
using True eventually-elim2 by auto
hence $\exists N \in$ null-sets $M .\left\{x \in\right.$ space $M . \neg\left(x \notin Y-{ }^{`}\{Y w\} \cap\right.$ space $\left.\left.M\right)\right\} \subseteq N$ using eventually-ae-filter $[o f \lambda x . x \notin Y-‘\{Y w\} \cap$ 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 $M\left\{x \in\right.$ space $\left.M . x \in \bar{Y}-{ }^{\prime}\{Y w\}\right\} \leq$ emeasure $M N$
by (simp add: emeasure-mono $\operatorname{Nprops(1)~null-setsD2)~}$
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 $M\left(\left(Y-{ }^{\prime}\{Y w\}\right) \cap\right.$ space $\left.M\right)=0$ 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 $M .(X y) *$ ? ind $Y y=(\lambda y .0) y$ using True by auto
hence rhs: integral ${ }^{L} M(\lambda y .(X y) *$ ? ind $Y y)=0$
proof -
have $\forall w \in$ space $M .0 \leq X w *$ ?ind $Y w$ using assms by simp
have integrable $M(\lambda y .(X y) *$ ? ind $Y y)$ using assms
by (metis (mono-tags, lifting) IntI UNIV-I $\langle Y w \in$ space $N\rangle$ image-eqI
Bochner-Integration.integrable-cong integrable-real-mult-indicator point-measurable-def)
hence $(\lambda y .(X y) *$ ? ind $Y y) \in$ borel-measurable $M$ by blast
thus ?thesis using zer integral-cong- $A E[o f(\lambda y .(X y) *$ ?ind $Y$ y) $M \lambda y .0]$
by $\operatorname{simp}$
qed
thus expl-cond-expect $M Y X w *$ measure $M((Y-‘\{Y w\}) \cap$ space $M)=$ integral $^{L} M(\lambda y .(X y) *$ ? ind $Y y)$ using lhs rhs by simp
next
case False
hence $\neg(A E$ 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 $M \in$ sets $M$
by (meson IntI UNIV-I $\langle Y w \in$ space $N\rangle$ assms(1) image-eqI point-measurable-def)
have $(Y-‘\{Y w\}) \cap$ space $M \notin$ null-sets $M$
using $\prec \neg(A E y$ in $M . y \notin Y-‘\{Y w\} \cap$ space $M)$ 〉 eventually-ae-filter by
blast
thus ?thesis
using $\langle Y-‘\{Y w\} \cap$ space $M \in$ sets $M\rangle$ 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 $^{L} M\left(\lambda y .\binom{X}{y} *\right.$ ? ind $\left.Y y\right)$ 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-cond-expect $M Y X w *$ (indicator $(Y-‘\{Y w\} \cap$ space $M$ )) y) $=$
integral ${ }^{L} M(\lambda y$. expl-cond-expect $M Y X y *($ indicator $(Y-‘\{Y w\} \cap$ space M)) y)
proof -
let ?ind $=\left(\right.$ indicator $\left(Y-{ }^{\prime}\{Y w\} \cap\right.$ space $\left.\left.M\right)\right)$
have $\forall y \in$ space $M$. expl-cond-expect $M Y X w *$ ?ind $y=$ expl-cond-expect $M$ $Y X y *$ ? ind $y$
proof
fix $y$
assume $y \in$ space $M$
show expl-cond-expect $M Y X w *$ ? ind $y=$ expl-cond-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 $Y$ 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 $M .0 \leq X w$
shows integral ${ }^{N} M(\lambda y$. expl-cond-expect $M Y X w *($ indicator $(Y-‘\{Y w\} \cap$ space $M$ )) y) =
integral ${ }^{N} M(\lambda y$. expl-cond-expect $M Y X y *($ indicator $(Y-‘\{Y w\} \cap$ space M)) $y$ )
proof -
let ?ind $=\left(\right.$ indicator $\left(Y-{ }^{\prime}\{Y w\} \cap\right.$ space $\left.\left.M\right)\right)$
have forall: $\forall y \in$ space $M$. expl-cond-expect $M Y X w *$ ? ind $y=$ expl-cond-expect $M Y X y *$ ?ind $y$

## proof

fix $y$
assume $y \in$ space $M$
show expl-cond-expect $M Y X w *$ ?ind $y=$ expl-cond-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 Yw=Yy 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 }\forallw\in\mathrm{ space M. 0 }\leqX
    and integrable M X
    and point-measurable M (space N) Y
    and w\in space M
    and Y w\in space N
    shows integral }\mp@subsup{}{}{N}M(\lambday.\mathrm{ expl-cond-expect M Y X y* (indicator (Y -'{Yw} }
space M)) y) < < 
proof -
    let ?ind = (indicator }(Y-`{Yw}\cap\mathrm{ space M))::'a mreal
    have 0\leqexpl-cond-expect M Y X w using assms nn-expl-cond-expect-pos[of M
X Y] by simp
    have integrable M (\lambday. expl-cond-expect MYX w* ?ind y)
    proof -
        have eq: (Y-`{Yw}\cap space M)\cap space M=(Y-`{Yw}\cap space M) by
auto
            have (Y -`{Yw}\cap space M)\in sets M using assms
                by (simp add: point-measurable-def)
        moreover have emeasure M (Y-‘{Yw} \cap space M)<\infty by (simp add:
inf.strict-order-iff)
    ultimately have integrable M ( }\lambday\mathrm{ . ?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 }\forally\in\mathrm{ space M. 0 < expl-cond-expect MYX w* ?ind y
    using <0 \leq expl-cond-expect M Y X w` mult-nonneg-nonneg by blast
    hence }\forally\in\mathrm{ space M. expl-cond-expect MYX w* ?ind y = norm (expl-cond-expect
MYXw* ?ind y) by auto
    hence inf: integral }\mp@subsup{}{}{N}M(\lambday. expl-cond-expect M YX w* ?ind y)<
            using integrable-iff-bounded[of M (\lambday. expl-cond-expect M YXw* ?ind y)]
                    <0 \leq expl-cond-expect M Y X w` real-norm-def nn-integral-cong
        by (metis (no-types, lifting)<integrable M (\lambday. expl-cond-expect M Y X w*
indicator (Y -` {Yw} \cap space M) y)>)
    have integral }\mp@subsup{}{}{N}M(\lambday. expl-cond-expect MYXy* ?ind y)
```

```
        integral N}M(\lambday. expl-cond-expect MYX w * ?ind y) using assm
    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 $Y:: ' a \Rightarrow$ ' $b$
assumes $B \subseteq$ space $N$
and point-measurable $M$ (space $N$ ) $Y$
and integrable $M X$
and $\forall w \in$ space $M .0 \leq X w$
shows $\forall i$. integral ${ }^{L} M(\lambda y .(X y) *($ indicator (countable-preimages $B Y i \cap$ space
M)) $y$ ) =
integral $^{L} M(\lambda y$. (expl-cond-expect $M$ Y X y) * (indicator (countable-preimages
BYi $\cap$ space $M)$ ) y)
proof
fix $i$
show integral ${ }^{L} M(\lambda y . X y *$ indicator (countable-preimages $B Y i \cap$ space $M$ )
$y)=$
integral ${ }^{L} M(\lambda y$. expl-cond-expect $M Y X y *$ indicator (countable-preimages
$B Y i \cap$ space $M) y)$
proof (cases countable-preimages $B Y i \cap$ space $M=\{ \}$ )
case True
thus ?thesis by simp
next
case False
from this obtain $w$ where $w \in$ countable-preimages $B Y i$ by auto
hence $Y w=($ from-nat-into $B) i$ by (meson count-pre-img)
hence $Y w \in B$
proof (cases infinite B)
case True
thus ?thesis
by (simp add: $\langle Y w=$ from-nat-into $B$ i〉 from-nat-into infinite-imp-nonempty)
next
case False
thus ?thesis
by (metis Finite-Set.card- $0-e q\langle Y w=$ from-nat-into $B \quad i\rangle\langle w \in$ count-
able-preimages $B Y$ i> countable-preimages-def equals0D from-nat-into gr-implies-not0)
qed
let ? ind $=($ indicator $(Y-‘\{Y w\} \cap$ space $M)):^{\prime}{ }^{\prime} a \Rightarrow$ real
have integral ${ }^{L} M(\lambda y .(X y) *($ indicator (countable-preimages $B Y i \cap$ space
M)) $y)=$ integral $^{L} M(\lambda y . X y *$ ?ind $y)$
by (metis (no-types, opaque-lifting) $\langle Y w=$ from-nat-into $B i\rangle\langle\bigwedge$ thesis. ( $\bigwedge w$.
$w \in$ countable-preimages $B Y i \Longrightarrow$ thesis $) \Longrightarrow$ thesis〉countable-preimages-def
empty-iff)
also have ... =
expl-cond-expect $M Y X w *$ measure $M(Y-‘\{Y w\} \cap$ space $M)$ using

```
dce-prod[of N Y X]
            by (metis (no-types, lifting)< Y w\inB> assms subsetCE)
    also have ... = expl-cond-expect MYXw*(integral L}M\mathrm{ ' ?ind)
        by auto
    also have ... = integral }\mp@subsup{}{}{L}M(\lambday. expl-cond-expect M YX w*?ind y)
        by auto
    also have ... = integral }\mp@subsup{}{}{L}M(\lambday. expl-cond-expect M YX y * ?ind y)
    proof -
        have }\forally\in\mathrm{ space M. expl-cond-expect MYXw* ?ind y = expl-cond-expect
MYXy*?ind y
        proof
            fix }
            assume y\in space M
            show expl-cond-expect MYXw* ?ind y = expl-cond-expect M Y X y*
?ind y
            proof (cases y\inY -`{Yw}\cap space M)
                case False
                    thus?thesis by simp
            next
                    case True
                    hence Yw = 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 }\mp@subsup{}{}{L}M(\lambday. expl-cond-expect M Y X y* indicator
(countable-preimages B Yi\cap space M) y)
        by (metis (no-types, opaque-lifting) <Y w = from-nat-into B i\rangle<\thesis. ( }\w
w}\mathrm{ countable-preimages B Y i ב thesis) }\Longrightarrow\mathrm{ 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 $B \subseteq$ space $N$
and countable $B$
and integrable $M X$
and $\forall w \in$ space $M .0 \leq X w$
shows integrable $M$ ( $\lambda w$. expl-cond-expect $M Y X w *$ indicator (countable-preimages $B Y n \cap$ space $M) w$ )
proof -
have integral ${ }^{L} M\left(\lambda y .\left(\begin{array}{l}X\end{array}\right) *\right.$ (indicator (countable-preimages $B Y n \cap$ space M)) $y$ ) $=$
integral ${ }^{L} M(\lambda y$. (expl-cond-expect $M Y X y) *($ indicator (countable-preimages
$B Y n \cap$ space $M)$ ) y) using assms count-prod by blast
have $\forall w \in$ space $M .0 \leq($ expl-cond-expect $M Y X w) *$ (indicator (countable-preimages $B Y n \cap$ space $M)) w$ by (simp add: assms nn-expl-cond-expect-pos)
have countable-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-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-cond-expect M Y X w indicator (countable-preimages B $Y n$
$\cap$ space $M) w) \in$ borel-measurable $M$
proof -
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 $B Y n \cap$ space $M)$ ) $\in$ borel-measurable M
using 〈countable-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 $A E x$ in $M .0 \leq$ expl-cond-expect $M Y X x *$ indicator (countable-preimages $B Y \cap \cap$ space $M) x$
by (simp add: $\langle\forall$ wespace M. $0 \leq$ expl-cond-expect $M Y X w *$ indicator (countable-preimages $B Y \cap \cap$ space $M$ ) w )
show ( $\int^{+}$x. ennreal (expl-cond-expect $M Y X x *$ indicator (countable-preimages $B Y n \cap$ space $M) x) \partial M)<\infty$
proof (cases countable-preimages $B Y n \cap$ space $M=\{ \}$ )
case True
thus ?thesis by simp
next
case False
from this obtain $w$ where $w \in$ countable-preimages $B Y \cap$ space $M$ by auto
hence countable-preimages $B Y n=Y-‘\{Y w\}$
by (metis IntD1 count-pre-img countable-preimages-def equals0D)
have $w \in$ space $M$ using False
using $\langle w \in$ countable-preimages $B Y n \cap$ space $M\rangle$ by blast
moreover have $Y w \in$ space $N$
by (meson $\langle w \in$ space $M\rangle \operatorname{assms(1)~assms(2)~disct-fct-point-measurable~}$ 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 $\forall w \in$ space $M .0 \leq X w$
and integrable $M X$
and disc－fct $Y$
and point－measurable $M$（space $M^{\prime}$ ）$Y$
shows $\forall A \in$ sets $M^{\prime}$ ．integrable $M(\lambda w$ ．$(($ expl－cond－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－cond－expect $\left.M Y X) w\right) *($ indicator $((Y-' A) \cap($ space M）））$w$ ）
proof
fix $A$
assume $A \in$ sets $M^{\prime}$
let $\lim A=A \cap($ range $Y)$
have countable ？imA using assms disc－fct－def by blast
have $Y-$＇$A=Y-$＇？im $A$ by auto
define $p r Y$ where $p r Y=$ countable－preimages ？imA $Y$
have un：$Y-$＇？$i m A=(\bigcup$ i．pr $Y i)$ using 〈countable ？imA〉
by（metis count－pre－union－img pr $Y$－def）
have $\left(Y-{ }^{\prime}\right.$ ？imA $) \cap($ space $M)=(\bigcup i$ ．pr $Y i) \cap($ space $M)$ using $\left\langle Y-{ }^{\prime} A=\right.$
$Y$－＇？im $A$ 〉 un by simp
also have $\ldots=(\bigcup i .(p r Y i) \cap($ space $M))$ by blast
finally have eq2：$(Y-‘ ? i m A) \cap($ space $M)=(\bigcup i .(p r Y i) \cap($ space $M))$ ．
define indpre：：nat $\Rightarrow{ }^{\prime} a \Rightarrow$ real where indpre $=(\lambda$ ix．$($ indicator $((p r Y i) \cap$
（space $M)$ ））$x$ ）
have $\forall i$ ．indpre $i \in$ borel－measurable $M$
proof
fix $i$
show indpre $i \in$ borel－measurable $M$ unfolding indpre－def prY－def
proof（rule borel－measurable－indicator，cases countable－preimages（ $A \cap$ range
Y）$Y i \cap$ space $M=\{ \})$
case True
thus countable－preimages $(A \cap$ range $Y) Y i \cap$ space $M \in$ sets $M$ by simp next
case False
from this obtain $x$ where $x \in$ countable－preimages $(A \cap$ range $Y) Y i \cap$ space $M$ by blast
hence $Y x \in$ space $M^{\prime}$
by（metis Int－iff $U N-I\left\langle A \in\right.$ sets $\left.M^{\prime}\right\rangle\langle p r Y \equiv$ countable－preimages $(A$

```
\cap range Y) Y> imageE
            rangeI sets.sets-into-space subset-eq un vimage-eq)
            thus countable-preimages ( }A\cap\mathrm{ range Y) Yi }\cap\mathrm{ space }M\in\mathrm{ sets }
                            by (metis IntE IntI«x \in countable-preimages ( }A\cap\mathrm{ range Y) Y i }
space M> assms(4)
                count-pre-img countable-preimages-def empty-iff point-measurable-def
rangeI)
            qed
    qed
    have \foralli. integrable M (\lambdaw. (X w)* indpre i w)
    proof
        fix }
        show integrable M ( }\lambdaw.(Xw)*\mathrm{ indpre i w) unfolding indpre-def prY-def
        proof (rule integrable-real-mult-indicator)
            show countable-preimages ( }A\cap\mathrm{ range Y) Yi}\cap\mathrm{ space }M\in\mathrm{ sets }
            proof (cases countable-preimages ( }A\cap\mathrm{ range Y)Yi={})
                case True
                thus countable-preimages (A\cap range Y)Yi\cap space M\in sets M by
(simp add: True)
            next
                case False
                hence Y -'{(from-nat-into }(A\cap\mathrm{ range }Y))i}\not={} unfolding count
able-preimages-def by meson
                    have (infinite ( }A\cap\mathrm{ range Y)) }\vee(\mathrm{ finite ( }A\cap\mathrm{ range }Y)\wedgei<card ( A
range Y)) using False unfolding countable-preimages-def
                    by meson
                show ?thesis
                by (metis }\langleA\in\mathrm{ sets M'><countable ( }A\cap\mathrm{ range }Y\mathrm{ )> 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: }\foralli.(\lambdaw.(X w)* indpre i w)\in borel-measurable M
        by (simp add: assms(2) borel-measurable-integrable borel-measurable-times)
    have posprod: }\foralliw.0\leq(Xw)*indpre i w
    proof (intro allI)
        fix i
        fix w
        show 0\leq(Xw)* 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\mathrm{ range }Y))\cap(\mathrm{ space }M))::'a=>\mathrm{ real
    have }\forallij.i\not=j\longrightarrow((prYi)\cap(\mathrm{ space M) ) ค((prYj) }\cap(\mathrm{ space M)) = {}
        by (simp add:<countable ( }A\cap\mathrm{ range Y)><prY 三 countable-preimages }(A
```

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 $p r Y$－def indpre－def by（metis indicator－sums）
hence sumxlim：$\forall x$ ．（ $\lambda i .(X x) *$ indpre $i x::$ real $)$ sums $((X x) *$ indicator $((Y$ $-‘ ? \operatorname{im} A) \cap($ space $M)) x$ ）using 〈countable ？im $A$ 〉 unfolding prY－def using sums－mult by blast
hence sum：$\forall x$ ．（ $\sum i .\left(\left(\begin{array}{ll}X\end{array}\right) *\right.$ indpre $\left.i x\right)::$ real $)=\left(\begin{array}{ll}X & x) * \text { indicator }((Y)\end{array}\right.$ $-\quad$ ？ $\operatorname{im} A) \cap($ space $M)) x$ by（metis sums－unique）
hence $b: \forall w .0 \leq\left(\sum i .((X w) *\right.$ indpre $\left.i w)\right)$ using suminf－nonneg by（metis $\forall x$ ．$(\lambda i . X x *$ indpre $i x)$ sums $\left(X x * \operatorname{indicator~}\left(Y-{ }^{\prime}(A \cap\right.\right.$ range $Y) \cap$ space $M) x)$＞posprod summable－def）
have sumcondlim：$\forall x$ ．（ $\lambda i$ ．（expl－cond－expect $M Y X x) *$ indpre $i x:$ ：real） sums（ $($ expl－cond－expect $M Y X x) *$ ？indA $x)$ using 〈countable ？imA〉 unfold－ ing $p r Y$－def using sums－mult sumind by blast
have integrable $M(\lambda w .(X w) *$ ？indA $w)$
proof（rule integrable－real－mult－indicator） show $Y-{ }^{\prime}(A \cap$ range $Y) \cap$ space $M \in$ sets $M$
using $\left\langle A \in\right.$ sets $\left.M^{\prime}\right\rangle \operatorname{assms}(3)$ assms（4）disct－fct－point－measurable measur－ able－sets
by（metis $\left\langle Y-{ }^{\prime} A=Y-‘(A \cap\right.$ range $\left.\left.Y)\right\rangle\right)$
show integrable $M X$ using assms by simp
qed
hence intsum：integrable $M\left(\lambda w .\left(\sum i .((X w) *\right.\right.$ indpre $\left.\left.i w)\right)\right)$ using sum Bochner－Integration．integrable－cong［of MM $\lambda$ w．$(X$ w）＊（indicator（ $(Y$ $-‘ A) \cap($ space $M)) w) \lambda w .\left(\sum i .((X w) *\right.$ indpre $\left.\left.i w)\right)\right]$ using $\left\langle Y-{ }^{`} A=Y-{ }^{`}(A \cap\right.$ range $\left.Y)\right\rangle$ by presburger
have integral ${ }^{L} M(\lambda w .(X w) *$ ？indA $w)=$ integral $^{L} M\left(\lambda w .\left(\sum i .((X w) *\right.\right.$ indpre $i w)$ ））
using $\left\langle Y-{ }^{\prime} A=Y-{ }^{\prime}(A \cap\right.$ range $\left.Y)\right\rangle$ sum by auto
also have...$=$
$\int{ }^{+} w . \quad\left(\left(\sum i .((X w) *\right.\right.$ indpre $\left.\left.i w)\right)\right) \partial M$ using nn－integral－eq－integral
by（metis（mono－tags，lifting）AE－I2 intsum b nn－integral－cong）
also have $\left(\int^{+} w . \quad\left(\left(\sum i .((X w) *\right.\right.\right.$ indpre $\left.\left.\left.i w)\right)\right) \partial M\right)=\int{ }^{+} w . \quad\left(\left(\sum i\right.\right.$. ennreal $((X w) *$ indpre $i w))) \partial M$ using suminf－ennreal2 summable－def posprod sum sumxlim

## proof－

\｛ fix $a a::{ }^{\prime} a$
have $\forall a$ ．ennreal $\left(\sum n . X a *\right.$ indpre $\left.n a\right)=\left(\sum n\right.$ ．ennreal $(X a *$ indpre $n$ a））
by（metis（full－types）posprod suminf－ennreal2 summable－def sumxlim）
then have $\left(\int^{+}\right.$a．ennreal $\left(\sum n . X a *\right.$ indpre $\left.\left.n a\right) \partial M\right)=\left(\int^{+} a .\left(\sum n\right.\right.$ ． ennreal $(X a *$ indpre $n a)) \partial M) \vee$ ennreal $\left(\sum n . X a a *\right.$ indpre $\left.n a a\right)=\left(\sum n\right.$ ． ennreal（ $X a a *$ indpre $n a a)$ ）

$$
\text { by metis \} }
$$

then show ？thesis
by presburger

```
    qed
    also have ... =(\sumi. integral }\mp@subsup{}{}{N}M((\lambdaiw.(Xw)*\mathrm{ indpre i w) i))
    proof (intro nn-integral-suminf)
        fix }
        show ( }\lambdax\mathrm{ . ennreal (X x* indpre i x) ) b borel-measurable M
        using measurable-compose-rev measurable-ennreal prod-bm by blast
    qed
    also have ... = (\sumi. integral }\mp@subsup{}{}{N}M((\lambdaiw.(expl-cond-expect M Y X w)* indpre
i w) i))
    proof (intro suminf-cong)
    fix n
    have }A\cap\mathrm{ range }Y\subseteq\mathrm{ space M'
        using <A \in sets M'\rangle sets.Int-space-eq1 by auto
        have integral }\mp@subsup{}{}{N}M(\lambdaw.(Xw)* indpre n w)= integral L M (\lambdaw. (X w)
indpre n w)
        using nn-integral-eq-integral[of M \lambdaw.(X w) * indpre n w]
        by (simp add: <\forall i. integrable M (\lambdaw. X w* indpre i w)〉 posprod)
    also have ... = integral }\mp@subsup{}{}{L}M(\lambdaw.(\mathrm{ expl-cond-expect M Y X) w* indpre n w)
    proof -
        have integral L}M(\lambdaw.Xw* indicator (countable-preimages ( A\cap range Y)
Y }\cap\cap\mathrm{ space M) w) =
            integral L}M(\lambdaw. expl-cond-expect MYXw*indicator (countable-preimages
(A\cap range Y) Yn\cap space M) w)
            using count-prod[of A\cap range Y M' Y X] assms <A\cap range Y}\subseteq\mathrm{ space
M'> by blast
            thus ?thesis
                using<indpre \equiv\lambdai. indicator ( prYi\cap space M)>prY-def by presburger
    qed
    also have ... = integral }\mp@subsup{}{}{N}M(\lambdaw.(\mathrm{ expl-cond-expect M Y X) w* indpre n w)
    proof -
    have integrable M ( }\lambdaw\mathrm{ . (expl-cond-expect M YX) 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. O\leq (expl-cond-expect M YX)w*indpre n w
                    by (simp add:<indpre \equiv \i. indicator (prYi\cap space M)〉assms(1)
nn-expl-cond-expect-pos)
            ultimately show ?thesis by (simp add:nn-integral-eq-integral)
            qed
            finally show integral }\mp@subsup{}{}{N}M(\lambdaw.(X w)* indpre n w)= integral N M (\lambdaw
(expl-cond-expect M Y X)w* indpre n w).
    qed
    also have ... = integral }\mp@subsup{}{}{N}M(\lambdaw.\sumi.((expl-cond-expect M Y X w)* indpre i
w))
    proof -
            have ( }\lambdax.(\sumi.\mathrm{ ennreal (expl-cond-expect MYX x*indpre i }x)))
                (\lambdax. ennreal ( }\sumi.(\mathrm{ expl-cond-expect M Y X x * indpre i x)})
        proof
            have posex: }\forall\mathrm{ i x. 0 {(expl-cond-expect MYX x)*(indpre i x)
```

by（metis IntE〈indpre $\equiv \lambda i$ ．indicator $(p r Y i \cap$ space $M)$ 〉assms（1）indica－ tor－pos－le indicator－simps（2）mult－eq－0－iff mult－sign－intros（1）nn－expl－cond－expect－pos）
have $\forall x$ ．（ $\sum$ i．ennreal（expl－cond－expect $M Y X x *$ indpre $\left.\left.i x\right)\right)=($ ennreal （ $\sum$ i．（expl－cond－expect M Y X $x *$ indpre $\left.i x\right)$ ）
proof
fix $x$
show $\left(\sum i\right.$. ennreal $($ expl－cond－expect $M Y X x *$ indpre $\left.i x)\right)=($ ennreal （ $\sum i$ ．（expl－cond－expect $M Y X x *$ indpre $\left.i x\right)$ ）
using suminf－ennreal2［of $\lambda i$ ．（expl－cond－expect $M Y X x *$ indpre $i x)]$ sumcondlim summable－def posex
proof－
have f1：summable（ $\lambda$ n．expl－cond－expect $M Y X x *$ indpre $n x)$
using sumcondlim summable－def by blast
obtain $n n$ ：：nat where
$\neg 0 \leq$ expl－cond－expect $M Y X x *$ indpre $n n x \vee \neg$ summable $(\lambda n$ ． expl－cond－expect MYXx＊indpre $n x) \vee$ ennreal（ $\sum n$ ．expl－cond－expect $M Y X$ $x *$ indpre $n x)=\left(\sum n\right.$ ．ennreal（expl－cond－expect MYXx＊indpre $\left.\left.n x\right)\right)$
by（metis（full－types）« $\lfloor$ 亿 i． $0 \leq$ expl－cond－expect $M Y X x *$ indpre $i x$ ；summable（ $\lambda i$ ．expl－cond－expect $M Y \bar{X} x *$ indpre $i x) \rrbracket \Longrightarrow\left(\sum i\right.$ ．ennreal $($ expl－cond－expect $M Y X x *$ indpre $i x))=$ ennreal $\left(\sum i\right.$ ．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 $\forall i .(\lambda w$ ．（expl－cond－expect $M Y X w) *$ indpre $i w) \in$ borel－measurable $M$
proof－ show ？thesis
using $\langle\forall i$ ．（indpre $i) \in$ borel－measurable $M\rangle \operatorname{assms}(3)$ assms（4）borel－measurable－times expl－cond－expect－borel－measurable by blast
qed
hence $\wedge i$ ．$(\lambda x$ ．ennreal（expl－cond－expect $M Y X x *$ indpre $i x)) \in$ borel－measurable M
using measurable－compose－rev measurable－ennreal by blast
thus ？thesis using nn－integral－suminf $[o f(\lambda i w$ ．（expl－cond－expect $M Y X$ w） ＊indpre $i w) M$ ，symmetric］
using $\left\langle\left(\lambda x . \sum i\right.\right.$. ennreal（expl－cond－expect MYXx＊indpre $\left.\left.i x\right)\right)=(\lambda x$.
ennreal（ $\sum$ i．expl－cond－expect $M Y X x *$ indpre $\left.i x\right)$ ）＞by auto
qed
also have $\ldots=$ integral $^{N} M(\lambda w .($ expl－cond－expect $M Y X w) *$ ？indA w） using sumcondlim
by（metis（no－types，lifting）sums－unique）
also have $\ldots=$ integral $^{L} M(\lambda w .($ expl－cond－expect $M Y X w) *$ ？indA $w)$
proof－
have scdint：integrable $M(\lambda w .($ expl－cond－expect $M Y X w) *$ ？indA $w)$ proof－
have rv：$\left(\lambda w .(\right.$ expl－cond－expect $M Y X w) *$ indicator $\left(\left(Y-{ }^{`} ? i m A\right) \cap(\right.$ space
M)) w) $\in$ borel-measurable $M$
proof -
have expl-cond-expect $M Y X \in$ borel-measurable $M$ using expl-cond-expect-borel-measurable using assms by blast
moreover have $(Y-‘ ? i m A) \cap($ space $M) \in$ sets $M$
by (metis $\left\langle A \in\right.$ sets $\left.M^{\prime}\right\rangle\left\langle Y-{ }^{\prime} A=Y-{ }^{\prime}(A \cap \operatorname{range} Y)\right\rangle \operatorname{assms}(3)$
$\operatorname{assms}(4)$ 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-cond-expect $M$
$Y X w * ? i n d A w)))<\infty$
proof -
have integral ${ }^{N} M(\lambda w$. ennreal (norm (expl-cond-expect $M Y X w *$ ? indA $w))$ ) $=$
integral ${ }^{N} M(\lambda w$. ennreal (expl-cond-expect $M Y X w *$ ?indA $\left.w)\right)$
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 $\operatorname{assms}(1))$
thus ?thesis by (metis (no-types, lifting) nn-integral-cong)
qed
thus ?thesis
by (metis (no-types, lifting)
$\prec\left(\sum i . \int{ }^{+} x\right.$. ennreal $(X x *$ indpre $\left.i x) \partial M\right)=\left(\sum i . \int{ }^{+} x\right.$. ennreal (expl-cond-expect MYXx*indpre $i x) \partial M)$ >
$\left\langle\left(\sum i . \int+{ }^{+}\right.\right.$. ennreal (expl-cond-expect $M Y X x *$ indpre $\left.\left.i x\right) \partial M\right)=$ $\left(\int^{+}\right.$x. ennreal $\left(\sum i\right.$. expl-cond-expect $M Y X x *$ indpre $\left.\left.i x\right) \partial M\right)$ >
$\prec\left(\int+{ }^{+} w\right.$. $\left(\sum i\right.$. ennreal $(X w *$ indpre $\left.\left.i w)\right) \partial M\right)=\left(\sum i . \int+{ }^{+}\right.$. ennreal ( $X x$ * indpre $i x) \partial M$ ) >
$\left\langle\left(\int^{+}\right.\right.$x. ennreal $\left(\sum i . X x *\right.$ indpre $\left.\left.i x\right) \partial M\right)=\left(\int^{+} w .\left(\sum i\right.\right.$. ennreal $(X w *$ indpre $i w)) \partial M)$ >
$\prec\left(\int{ }^{+}{ }^{x}\right.$. ennreal ( $\sum$ i. expl-cond-expect $M Y X x *$ indpre ix) $\left.\partial M\right)$
$=\left(\int{ }^{+}\right.$x. ennreal (expl-cond-expect $M Y X x *$ indicator $(Y-‘(A \cap$ range $Y) \cap$ space $M$ ) x) $\partial M$ )>
<ennreal (integral ${ }^{L} M\left(\lambda w . \sum i . X w *\right.$ indpre $\left.\left.i w\right)\right)=\left(\int+x\right.$. ennreal ( $\sum$ i. $X x$ * indpre $\left.i x\right) \partial M$ ) >
ennreal-less-top infinity-ennreal-def)
qed
show integrable $M(\lambda w$. (expl-cond-expect $M Y X w) *$ ?ind $A w)$
proof (rule iffD2[OF integrable-iff-bounded])
show ( $(\lambda w$. expl-cond-expect $M Y X w *$ indicator $(Y-‘(A \cap$ range $Y)$ $\cap$ space $M) w) \in$ borel-measurable $M) \wedge$
$\left(\left(\int^{+}{ }^{x}\right.\right.$. (ennreal (norm (expl-cond-expect MYXx*indicator $\left(Y-{ }^{\prime}(A\right.$
$\cap$ range $Y) \cap$ space $M) x))$ ) $\partial M)<\infty$ )
proof
show ( $\lambda w$. expl-cond-expect $M Y X w *$ indicator $(Y-‘(A \cap$ range $Y)$ $\cap$ space $M) w) \in$ borel-measurable $M$
using $r v$ by simp
show ( $\int+$ x. ennreal (norm (expl-cond-expect M Y X $x *$ indicator ( $Y$
$-{ }^{\prime}(A \cap$ range $Y) \cap$ space $\left.\left.\left.\left.M\right) x\right)\right) \partial M\right)<\infty$
using born by simp qed
qed
qed
moreover have $\forall w \in$ space $M .0 \leq($ expl-cond-expect $M Y X w) *$ indicator $((Y-` ? i m A) \cap($ 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 $\left.w)\right)=$ integral $^{L}$ $M(\lambda w .($ expl-cond-expect $M Y X w) * ? i n d A w)$.
thus integrable $M\left(\lambda w\right.$. expl-cond-expect $M Y X w *$ indicator $\left(Y-{ }^{\prime} A \cap\right.$ space $M) w) \wedge \operatorname{integral}^{L} M\left(\lambda w . X w *\right.$ indicator $\left.\left(Y-{ }^{\prime} A \cap \operatorname{space} M\right) w\right)=$ integral $^{L} M$ ( $\lambda w$. expl-cond-expect $M Y X w *$ indicator $(Y-‘ A \cap$ space M) $w$ )
proof -
have $0 \leq$ integral $^{L} M\left(\lambda w . X w *\right.$ indicator $\left(Y-{ }^{`} A \cap\right.$ space $\left.\left.M\right) w\right)$
using $\left\langle Y-{ }^{`} A=Y-{ }^{`}(A \cap\right.$ range $\left.Y)\right\rangle b$ sum by fastforce
moreover have $0 \leq$ integral $^{L} M$ ( $\lambda w$. expl-cond-expect $M Y X w *$ indicator $\left(Y-{ }^{`} A \cap\right.$ space $\left.\left.M\right) w\right)$ by (simp add: assms(1) nn-expl-cond-expect-pos)
ultimately have expeq: integral ${ }^{L} M\left(\lambda w . X w *\right.$ indicator $\left(Y-{ }^{\prime} A \cap\right.$ space M) $w)=$
integral ${ }^{L} M(\lambda w$. expl-cond-expect $M Y X w * \operatorname{indicator}(Y-‘ A \cap$ space M) $w$ )
by (metis (mono-tags, lifting) Bochner-Integration.integral-cong $\left\langle Y-{ }^{‘} A=\right.$ $Y-{ }^{\prime}(A \cap$ range $Y)$ ) ennreal-inj myeq $)$
have integrable $M(\lambda w .($ expl-cond-expect $M Y X w) *$ ?indA $w)$
proof -
have rv: ( $\lambda w$. (expl-cond-expect $M Y X w) *$ indicator $((Y-‘ ? i m A) \cap$
$($ space $M)) w) \in$ borel-measurable $M$
proof -
have expl-cond-expect $M Y X \in$ borel-measurable $M$ using expl-cond-expect-borel-measurable
using assms by blast
moreover have $(Y-‘ ? i m A) \cap($ space $M) \in$ sets $M$
by (metis $\left\langle A \in\right.$ sets $\left.M^{\prime}\right\rangle\left\langle Y-{ }^{\prime} A=Y-{ }^{\prime}(A \cap \operatorname{range} Y)\right\rangle \operatorname{assms}(3)$
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-cond-expect
$M Y X w * ? \operatorname{ind} A w)))<\infty$
proof -
have integral ${ }^{N} M(\lambda w$. ennreal (norm (expl-cond-expect $M Y X w *$ ? ind $A$
$w))$ ) $=$

```
            integral \(^{N} M(\lambda w\). ennreal (expl-cond-expect \(M Y X w * ?\) indA \(\left.w)\right)\)
```

        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 $\operatorname{assms}(1))$
thus ?thesis by (metis (no-types, lifting) nn-integral-cong)
qed
thus ?thesis
by (metis (no-types, lifting)
$\prec\left(\sum i . \int{ }^{+} x\right.$. ennreal $(X x *$ indpre $\left.i x) \partial M\right)=\left(\sum i . \int+x\right.$. ennreal (expl-cond-expect $M Y X x *$ indpre $i x) \partial M$ ) >
$\prec\left(\sum i . \int{ }^{+} x\right.$. ennreal (expl-cond-expect $M Y X x *$ indpre $\left.\left.i x\right) \partial M\right)=$ $\left(\int{ }^{+}\right.$x. ennreal $\left(\sum i\right.$. expl-cond-expect $M Y X x *$ indpre ix) $\left.\partial M\right)$ > $\left\langle\left(\int{ }^{+} w .\left(\sum i\right.\right.\right.$. ennreal $(X w *$ indpre $\left.\left.i w)\right) \partial M\right)=\left(\sum i . \int+x\right.$. ennreal ( $X x$ * indpre $i x) \partial M$ ) >
$\left\langle\left(\int^{+}\right.\right.$x. ennreal $\left(\sum i . X x *\right.$ indpre $\left.\left.i x\right) \partial M\right)=\left(\int^{+} w .\left(\sum i\right.\right.$. ennreal $(X w *$ indpre $i w)) \partial M)$ >
$\left\langle\left(\int^{+}\right.\right.$x. ennreal ( $\sum$ i. expl-cond-expect $M Y X x *$ indpre $\left.\left.i x\right) \partial M\right)$ $=\left(\int^{+} x\right.$. ennreal (expl-cond-expect $M Y X x *$ indicator $(Y-‘(A \cap$ range $Y) \cap$ space $M$ ) x) $\partial M)$ 〉
<ennreal (integral ${ }^{L} M\left(\lambda w . \sum i . X w *\right.$ indpre $\left.\left.i w\right)\right)=\left(\int+{ }^{+}\right.$. ennreal ( $\sum$ i. $X x$ * indpre $\left.i x\right) \partial M$ ) > ennreal-less-top infinity-ennreal-def)
qed
show integrable $M(\lambda w$. (expl-cond-expect $M Y X w) *$ ?ind $A w)$
proof (rule iffD2[OF integrable-iff-bounded])
show ( $\left(\lambda w\right.$. expl-cond-expect $M Y X w *$ indicator $\left(Y-{ }^{\prime}(A \cap\right.$ range $Y)$ $\cap$ space $M) w) \in$ borel-measurable $M) \wedge$
$\left(\left(\int^{+}\right.\right.$x. (ennreal (norm (expl-cond-expect MYXX*indicator ( $Y$-‘ $(A \cap$ range $Y) \cap$ space $M) x)) \partial M)<\infty)$
proof
show ( $\lambda w$. expl-cond-expect $M Y X w *$ indicator $\left(Y-{ }^{\prime}(A \cap\right.$ range $Y)$ $\cap$ space $M) w) \in$ borel-measurable $M$
using $r v$ by simp
show ( $\int{ }^{+}$x. ennreal (norm (expl-cond-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
hence integrable $M$ ( $\lambda w$. expl-cond-expect $M Y X w *$ indicator $\left(Y-{ }^{\prime} A \cap\right.$ space $M$ ) w)
using $\left\langle Y-{ }^{\prime} A=Y-{ }^{\prime}(A \cap\right.$ range $\left.Y)\right\rangle$ by $p$ resburger
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 MYX)
proof -
have $Y$ - 'space $N \cap$ space $M=$ space $M$
by (meson assms(3) assms(4) disct-fct-point-measurable inf-absorb2 measur-able-space subsetI vimageI)
let ?indA $=$ indicator $((Y-‘$ space $N) \cap($ space $M)):: ' a \Rightarrow$ real
have $\forall w \in$ space $M$. (?indA $w)=(1::$ real $)$ by (simp add: $\langle Y-‘$ space $N \cap$ space $M=$ space $M>$ )
hence $\forall w \in$ space $M$. ((expl-cond-expect $M Y X) w) *($ ?indA $w)=($ expl-cond-expect
$M Y X) w$ by $\operatorname{simp}$
moreover have integrable $M(\lambda w .(($ expl-cond-expect $M Y X) w) *(? i n d A w))$
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) $n n$-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 $\forall A \in$ sets $N$. integral $^{L} M(\lambda w .(X w) *($ indicator $((Y-' A) \cap($ space $M))$
w)) $=$
integral ${ }^{L} M(\lambda w .(($ expl-cond-expect $M Y X) w) *($ indicator $((Y-' A) \cap($ 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 $=\lambda w .0$
let ? $X p=\lambda w . \max ($ ?zer $w)(X w)$
let ? $X n=\lambda w . \max ($ ?zer 0$)(-X w)$
have $\forall w . X w=$ ? $X p w-$ ? $X n w$ 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 (\lambdaw. (expl-cond-expect M Y ?Xp) w-
(expl-cond-expect M Y ?Xn) w) by auto
    have }\forallw\in 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 }
        assume w\in space M
            hence (expl-cond-expect M Y ?Xp) w - (expl-cond-expect M Y ?Xn) w =
(expl-cond-expect M Y ( }\lambdax.\mathrm{ ?Xp x - ?Xn x)) w
            using ints by (simp add: expl-cond-exp-diff)
    also have ... = expl-cond-expect MY X w using expl-cond-exp-cong «\forallw. 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 Xw.
    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 }\forallA\in\mathrm{ sets N. integral }\mp@subsup{}{}{L}M(\lambdaw.(Xw)*(indicator ((Y - 'A)\cap (space M)
w))=
    integral }\mp@subsup{}{}{L}M(\lambdaw.((\mathrm{ expl-cond-expect M Y X)w)* (indicator }((Y-'A)\cap(\mathrm{ space
M))) w)
proof -
    let ?zer = \lambdaw.0
    let ?Xp = \lambdaw. max (?zer w) (Xw)
    let ?Xn = \w. max (?zer 0) (-Xw)
    have }\forallw.Xw=?Xpw-?Xn w by aut
    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: }\forallA\in\mathrm{ sets N. integral }\mp@subsup{}{}{L}M(\lambdaw.(?Xp w) * (indicator ((Y - 'A)
(space M)) w)) =
        integral }\mp@subsup{}{}{L}M(\lambdaw.((expl-cond-expect M Y ?Xp)w)*(indicator ((Y - `A) \cap
(space M))) w)
        using nn-cond-expl-is-cond-exp[of ?Xp Y N] assms by auto
    have eqn: }\forallA\in\mathrm{ sets N. integral L}M(\lambdaw.(?Xn w)*(indicator ((Y - 'A)
(space M)) w)) =
        integral }\mp@subsup{}{}{L}M(\lambdaw.((expl-cond-expect M Y ?Xn) w)* (indicator ((Y -`A) \cap
```

$($ space $M))) w$ )
using $n n$-cond-expl-is-cond-exp $[o f$ ? $X n Y N]$ assms by auto
show $\forall A \in$ sets $N$. integral $^{L} M\left(\lambda w .(X w) *\left(\right.\right.$ indicator $\left(\left(Y-{ }^{\prime} A\right) \cap(\right.$ space $\left.M)\right)$ w)) $=$
integral ${ }^{L} M\left(\lambda w .((\right.$ expl-cond-expect $M Y X) w) *\left(\right.$ indicator $\left(\left(Y-{ }^{\prime} A\right) \cap(\right.$ space M))) $w$ )
proof
fix $A$
assume $A \in$ sets $N$
let $\lim A=A \cap($ range $Y)$
have countable ? imA using assms disc-fct-def by blast
have $Y-{ }^{\prime} A=Y-{ }^{\prime} ? i m A$ by auto
have yev: $Y-‘(A \cap$ range $Y) \cap$ space $M \in$ sets $M$ using $\langle A \in$ sets $N\rangle$ assms(3) assms(2) disct-fct-point-measurable measur-able-sets

$$
\text { by }(\text { metis }\langle Y-‘ A=Y-‘(A \cap \text { range } Y)\rangle)
$$

let ? ind $A=$ indicator $\left(\left(Y-{ }^{\prime}(A \cap\right.\right.$ range $\left.Y)\right) \cap($ space $\left.M)\right)::^{\prime} a \Rightarrow$ real
have intp: integrable $M(\lambda w$. (?Xp $w) *$ ?indA $w)$
proof (rule integrable-real-mult-indicator)
show $Y$-' $(A \cap$ range $Y) \cap$ space $M \in$ sets $M$ using yev by simp
show integrable $M$ ? Xp using assms by simp
qed
have intn: integrable $M(\lambda w$. $(? X n w) *$ ? indA $w)$
proof (rule integrable-real-mult-indicator)
show $Y-$ ' $(A \cap$ range $Y) \cap$ space $M \in$ sets $M$ using yev by simp
show integrable $M$ ? Xn using assms by simp
qed
have exintp: integrable $M(\lambda w$. (expl-cond-expect $M Y$ ? $X p w) *$ ?indA $w)$
proof (rule integrable-real-mult-indicator)
show $Y-‘(A \cap$ range $Y) \cap$ space $M \in$ sets $M$ using yev by simp
show integrable $M$ (expl-cond-expect $M$ ? ?Xp) using posint by simp
qed
have exintn: integrable $M(\lambda w$. (expl-cond-expect $M Y$ ? $X n w) *$ ?indA $w)$ proof (rule integrable-real-mult-indicator)
show $Y-‘(A \cap$ range $Y) \cap$ space $M \in$ sets $M$ using yev by simp show integrable $M$ (expl-cond-expect $M Y$ ?Xn) using negint by simp
qed
have integral ${ }^{L} M\left(\lambda w . X w *\right.$ indicator $\left(Y-{ }^{\prime} A \cap\right.$ space $\left.\left.M\right) w\right)=$ integral ${ }^{L} M\left(\lambda w .(? X p w-? X n w) *\right.$ indicator $\left(Y-{ }^{‘} A \cap\right.$ space $\left.\left.M\right) w\right)$ using $\langle\forall w . X w=$ ? $X p w-$ ? $X n w\rangle$ by auto
also have $\ldots=$ integral $^{L} M\left(\lambda w\right.$. (?Xp $w * \operatorname{indicator~}\left(Y-{ }^{`} A \cap\right.$ space $\left.M\right)$
$w)-$ ? $X n w$ * indicator $(Y-‘ A \cap$ space $M) w)$
by (simp add: left-diff-distrib)
also have $\ldots=$ integral $^{L} M(\lambda w .($ ?Xp $w * \operatorname{indicator}(Y-‘ A \cap$ space $M) w)$
integral ${ }^{L} M\left(\lambda w\right.$.?Xn $w *$ indicator $\left(Y-{ }^{\prime} A \cap\right.$ space $\left.\left.M\right) w\right)$
using $\left\langle Y-{ }^{\prime} A=Y-{ }^{\prime}(A \cap\right.$ range $\left.Y)\right\rangle$ intp intn by auto
also have $\ldots=$ integral $^{L} M(\lambda w .(($ expl-cond-expect $M Y$ ?Xp $) w) *($ indicator
$((Y-‘ A) \cap($ space $M))) w)-$
integral ${ }^{L} M(\lambda w .(($ expl-cond-expect $M Y$ ?Xn $) w) *($ indicator $((Y-‘ A) \cap$ $($ space $M))) w$ ) using eqp eqn by (simp add: $\langle A \in$ sets $N\rangle)$
also have $\ldots=$ integral $^{L} M(\lambda w .(($ expl-cond-expect $M Y$ ?Xp $) w) *($ indicator $((Y-‘ A) \cap($ space $M))) w-$ $(($ expl-cond-expect $M Y$ ? Xn $) w) *($ indicator $((Y-‘ A) \cap($ space $M))) w)$ using $\left\langle Y-{ }^{\prime} A=Y-{ }^{\prime}(A \cap\right.$ range $\left.Y)\right\rangle$ exintn exintp by auto
also have $\ldots=$ integral $^{L} M(\lambda w .(($ expl-cond-expect $M Y$ ?Xp $) w-($ expl-cond-expect $M Y$ ?Xn $) w) *($ indicator $((Y-‘ A) \cap($ space $M))) w)$ by (simp add: left-diff-distrib)
also have $\ldots=$ integral $^{L} M(\lambda w$. ( expl-cond-expect $M Y(\lambda x$. ? $X p x-$ ? $X n$ 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 $\ldots=$ integral $^{L} M(\lambda w$. ( expl-cond-expect $M Y X w) *($ indicator $((Y$ $-\quad(A) \cap($ space $M))) w))$
using $\langle\forall w . X w=$ ? $X p w-$ ? $X n$ $w\rangle$ expl-cond-exp-cong $[o f M X \lambda x$. ?Xp $x$ - ? Xn x $Y$ ] by presburger
finally show integral ${ }^{L} M\left(\lambda w . X w *\right.$ indicator $\left(Y-{ }^{\prime} A \cap\right.$ space $\left.\left.M\right) w\right)=$ integral ${ }^{L} M(\lambda w$. ( expl-cond-expect $M Y X w) *\left(\right.$ indicator $\left(\left(Y-{ }^{\prime} A\right) \cap(\right.$ 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 $M \rightarrow$ space $N$
and $\forall r \in$ range $Y \cap$ space $N . \exists A \in$ sets $N$. range $Y \cap A=\{r\}$
shows $A E$ w in $M$. real-cond-exp $M$ (fct-gen-subalgebra $M N Y) X w=$ expl-cond-expect
MYXw
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 $\langle Y \in$ measurable $M N\rangle$
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 $\forall a \in\{$ space $M\}$ ．emeasure（restr－to－subalg $M$（fct－gen－subalgebra $M N$ Y））$a \neq \infty$
by（metis «subalgebra $M$（fct－gen－subalgebra M N Y）〉 emeasure－finite emea－ sure－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 ）
using assms expl－cond－exp－integrable by blast
have $\forall A \in$ sets $M$ ．integral ${ }^{L} M(\lambda w .(X w) *($ 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 $\forall A \in$ sets $M$ ．integral ${ }^{L} M(\lambda w .((\operatorname{expl}$－cond－expect $M Y X) w) *($ indicator $A w))=$ set－lebesgue－integral MA（expl－cond－expect M Y $)$
by（metis（no－types，lifting）Bochner－Integration．integral－cong mult－commute－abs real－scaleR－def set－lebesgue－integral－def）
have $\forall A \in$ sets（fct－gen－subalgebra $M N Y$ ）．integral ${ }^{L} M(\lambda w .(X w) *$（indicator $A w))=$
integral $^{L} M(\lambda w .(($ expl－cond－expect $M Y X) w) *($ indicator $A w))$
proof
fix $A$
assume $A \in$ sets（fct－gen－subalgebra $M N Y$ ）
hence $A \in\left\{Y-{ }^{\prime} B \cap\right.$ space $M \mid B . B \in$ sets $\left.N\right\}$ using assms by（simp add：fct－gen－subalgebra－sigma－sets）
hence $\exists B \in$ sets $N . A=Y-{ }^{\prime} B \cap$ space $M$ by auto
from this obtain $B$ where $B \in$ sets $N$ and $A=Y-{ }^{\prime} B \cap$ space $M$ by auto
thus integral ${ }^{L} M(\lambda w .(X w) *($ indicator $A w))=$
integral ${ }^{L} M(\lambda w .(($ expl－cond－expect $M Y X) w) *($ indicator $A w))$ using is－cond－exp
using Bochner－Integration．integral－cong 〈point－measurable $M($ space $N) Y$ 〉 $\operatorname{assms}(1) \operatorname{assms}(2)$ by blast
qed
thus $\bigwedge A . A \in$ sets（fct－gen－subalgebra $M N Y) \Longrightarrow$ set－lebesgue－integral M A X ＝set－lebesgue－integral M（expl－cond－expect MYX）
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 $Y \in$ measurable $M N$
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-cond-exp M (fct-gen-subalgebra M NY)X w = expl-cond-expect
MYXw
proof (rule charact-cond-exp)
    show disc-fct Y using assms by simp
    show integrable M X using assms by simp
    show }\forallr\in\mathrm{ range }Y\cap\mathrm{ space N. ヨAGsets N. range }Y\capA={r}\mathrm{ using assms by
simp
    show }Y\in\mathrm{ space M }->\mathrm{ 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 (\prodi\in{0..n}.fi)=f0*(\prodi\in{1..n}.fi)
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 $\bigcap A \cap B=(\cap C \in A .(C \cap B))$
proof
show $(\bigcap C \in A . C \cap B) \subseteq \bigcap A \cap B$
proof

```
    fix }
    assume mem: }x\in(\bigcapC\inA.C\capB
    from <A\not={}\rangle obtain C where C\inA by blast
    hence }x\inC\capB\mathrm{ using mem by blast
    hence in1: x\in B by auto
    have }\C.C\inA\Longrightarrowx\inC\capB\mathrm{ using mem by blast
    hence }\C.C\inA\Longrightarrowx\inC\mathrm{ by auto
    hence in2: }x\in\bigcapA\mathrm{ by auto
    thus }x\in\bigcapA\capB\mathrm{ using in1 in2 by simp
    qed
qed auto
lemma enn2real-sum: shows finite }A\Longrightarrow(\bigwedgea.a\inA\Longrightarrowfa<top)\Longrightarrowenn2real
(sum f A) = (\suma\in A. enn2real (f a))
proof (induct rule:finite-induct)
    case empty
    thus ?case by simp
next
    case (insert a A)
    have enn2real (sumf(insert a A)) = enn2real (fa+(sumf 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(\a.0\leqfa)\Longrightarrow(\suma\in A. ennreal ( }
a)) = ennreal ( }\suma\inA.fa
proof (induct rule:finite-induct)
    case empty
    thus ?case by simp
next
    case (insert a A)
    have (\suma\in (insert a A). ennreal (f a)) = ennreal (fa) + ( \suma\in A. ennreal ( f
a))
    by (simp add: insert.hyps(1) insert.hyps(2))
    also have ... = ennreal (fa) + ennreal (\suma\inA.fa)
    by (simp add: insert.prems)
also have ... = ennreal (fa+( (\suma\inA.fa))
    by (simp add: insert.prems sum-nonneg)
also have ... = ennreal ( }\suma\in(\mathrm{ insert a A ). (f a))
    using insert.hyps(1) insert.hyps(2) by auto
finally show ?case .
```

```
lemma stake-snth:
    assumes stake \(n w=\) stake \(n x\)
    shows Suc \(i \leq n \Longrightarrow\) 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 \(\forall i<n\). 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 \((\bigwedge i\). Suc \(i \leq n \Longrightarrow\) snth \(w i=\) snth \(x i) \Longrightarrow\) stake \(n w=\) stake \(n x\)
proof (induct \(n\) arbitrary: \(w x\) )
case 0
    then show ?case by auto
next
case (Suc n)
    hence \(m h\) : \(\bigwedge i\). Suc \(i \leq\) Suc \(n \Longrightarrow w!!i=x!!i\) 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 \(0 \leq n\)
    shows \(\{w \in\) space \(M .(\) stake \(n w=\) stake \(n x)\}=(\bigcap i \in\{0 . . n-1\} .\{w \in\) space
\(M .(\) snth \(w i=\operatorname{snth} x i)\})\)
proof
    let \(? S=\{w \in\) space \(M\). (stake \(n w=\) stake \(n x)\}\)
    show ? \(S \subseteq(\bigcap i \in\{0 . . n-1\} .\{w \in\) space \(M . w!!i=x!!i\})\) using stake-snth
assms by fastforce
    show \((\bigcap i \in\{0 . . n-1\} .\{w \in\) space \(M . w!!i=x!!i\}) \subseteq ? S\)
    proof
        fix \(w\)
        assume inter: \(w \in(\bigcap i \in\{0 . . n-1\} .\{w \in\) space \(M . w!!i=x!!i\})\)
    have \(\forall i .0 \leq i \wedge i \leq n-1 \longrightarrow\) snth \(w i=\) snth \(x i\)
    proof (intro allI impI)
                fix \(i\)
                assume \(0 \leq i \wedge i \leq n-1\)
                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\in?S using inter by auto
    qed
qed
lemma streams-stake-set:
    shows pw\in streams A\Longrightarrow set (stake n pw)\subseteqA
proof (induct n arbitrary: pw)
    case (Suc n) note hyp = this
    have set (stake (Suc 0) pw)\subseteqA
    proof -
            have shd pw\inA 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\inA> 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\not={
    shows (stake (Suc n)'(streams A)) ={s#w| s w. s\inA\wedge w\in(stake n'(streams
A))}(is ?L = ?R)
proof
    show ?L \subseteq?R
    proof
        fix l::'a list
        assume l\in?L
        hence }\exists\textrm{pw}.pw\in\mathrm{ streams }A\wedgel=\mathrm{ stake (Suc n) pw by auto
        from this obtain pw where pw\in 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\in?R by (simp add:<pw \in streams A> streams-shd streams-stl)
    qed
    show ?R\subseteq?L
    proof
            fix l::'a list
            assume l\in ?R
            hence \exists}sw.s\inA\wedgew\in(\mathrm{ stake n'(streams A))}\wedgel=s#w\mathrm{ by auto
            from this obtain }s\mathrm{ and }w\mathrm{ where }s\inA\mathrm{ and }w\in(\mathrm{ stake n '(streams A)) and l
=s#w by blast
            note swprop = this
            have }\existspw.pw\in\mathrm{ streams }A\wedgew=\mathrm{ stake n pw using swprop by auto
            from this obtain pw where pw\in streams A and w= stake n pw by blast
note pwprop = this
    have}l\in\mathrm{ lists A
```

```
    proof -
        have s\inA using swprop by simp
        have set w\subseteqA using pwprop streams-stake-set by simp
        have set l= set w\cup{s} using swprop by auto
        thus ?thesis using \langles\inA\rangle\langleset w\subseteqA\rangle by auto
    qed
    have }\existsx.x\inA\mathrm{ using assms by auto
    from this obtain x where x\inA by blast
    let ?sx = sconst }
    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\subseteqA using <x\inA\rangle by simp
    hence ?sx streams A using sset-streams[of sconst x] by simp
    hence ?st \in streams A using <l l lists A` shift-streams[of l A sconst x] by
simp
    thus l\in ?L using «l = stake (Suc n) ?st> by blast
    qed
qed
lemma stake-finite-universe-finite:
    assumes finite A
    and }A\not={
    shows finite (stake n'(streams A))
proof (induction n)
    let ?L = (stake 0'(streams A))
    have streams A\not={}
    proof
        assume streams A={}
        have \existsx. x \in A using assms by auto
        from this obtain x where x\inA by blast
        let ?sx = sconst }
        have sset ?sx = {x} by simp
        hence sset ?sx \subseteqA using <x\in A\rangle by simp
        hence ?sx }\in\mathrm{ streams A using sset-streams[of sconst x] by simp
        thus False using <streams A = {}> by simp
    qed
    have stake 0 = (\lambdas.[]) unfolding stake-def by simp
    hence ? L = {[]} using <streams A}\not={}> by aut
    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\inA\wedge w\in(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: $\operatorname{smap}(\lambda n . x!!(n-S u c 0))($ fromN $($ Suc 0$)) \neq w$ by (metis assms stream-smap-fromN)
obtain $n n::$ ' $a$ stream $\Rightarrow$ nat stream $\Rightarrow(n a t \Rightarrow ' a) \Rightarrow$ nat where $\forall x 0 x 1 x 2 .(\exists v 3 . x 2(x 1$ !! v3) $\neq x 0$ !! v3) $=(x 2(x 1$ !! nn $x 0 x 1 x 2) \neq x 0$ !!
nn x0 x1 x2)
by moura
then have $x!!($ from $N($ Suc 0$)!!$ nn $w($ fromN $($ Suc 0$))(\lambda n . x!!(n-S u c 0))$

- Suc 0$) \neq w!$ ! nn $w($ fromN $($ Suc 0$))(\lambda n . x!!(n-S u c 0))$
using $f 1$ 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-sets $X Y$
and countable $A$
shows $(\bigcup y \in A . B y) \in$ sigma-sets $X Y$
by (metis (mono-tags, lifting) assms countable-image imageE sigma-sets-UNION)
lemma sigma-set-inter-init:
assumes $\bigwedge i$. $i \leq(n:: n a t) \Longrightarrow A i \in$ sigma-sets sp $B$
and $B \subseteq$ Pow sp
shows $(\bigcap i \in\{m . m \leq n\} . A i) \in$ sigma-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-algebra (space M) (sigma-sets (space M) $\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime}\right.\right.$
$A \cap$ space $M \mid A . A \in$ sets $N\})$ )
proof (rule sigma-algebra-sigma-sets)
show $\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime} A \cap\right.\right.$ space $M \mid A . A \in$ sets $\left.\left.N\right\}\right) \subseteq$ 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}\subseteqPow (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 $\Rightarrow$ (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 $f-{ }^{\prime} A \cap($ 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 $\forall n$. emeasure $N\{w \in$ 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 ? $S=\{w \in$ space $N .($ snth $w n)\}$
have $s: ? S \in$ sets $N$
proof -
have $(\lambda w$. snth $w n) \in$ measurable $N$ (measure-pmf (bernoulli-pmf $p)$ ) using assms by (simp add: bernoulli-stream-def)
moreover have $\{$ True $\} \in$ sets (measure-pmf (bernoulli-pmf p)) by simp
ultimately show?thesis by simp
qed
let $? P M=(\lambda i:: n a t .($ measure-pmf $($ bernoulli-pmf $p)))$
have isps: product-prob-space ?PM by unfold-locales
let $? Z=\{X::$ nat $\Rightarrow$ bool. $X n=$ True $\}$
let $? w P M=P i_{M}$ UNIV ?PM
have space ? $w P M=$ UNIV using space-PiM by fastforce
hence (to-stream - '? $S \cap($ space ? $w P M)$ ) $=$ to-stream - ' ? $S$ by simp
also have $\ldots=$ ? $Z$ using assms by (simp add:bernoulli-stream-space to-stream-def)
also have $\ldots=$ prod-emb UNIV ?PM $\{n\}\left(P i_{E}\{n\}(\lambda x:: n a t .\{\right.$ True $\left.\})\right)$
proof
\{

```
    fix }
    assume X \in prod-emb UNIV ?PM {n} (P\mp@subsup{i}{E}{}{n}(\lambdax::nat. {True}))
    hence restrict X {n}\in(P\mp@subsup{i}{E}{}{n}(\lambdax::nat. {True})) using prod-emb-iff[of
X] by simp
    hence Xn= True
        unfolding PiE-iff by auto
    hence }X\in?Z\mathrm{ by simp
    }
    thus prod-emb UNIV ?PM {n} (Pi\mp@subsup{i}{E}{}{n}(\lambdax::nat. {True}))\subseteq?Z by auto
    {
        fix }
        assume X \in?Z
    hence X n= True by simp
    hence restrict X {n}\in(P\mp@subsup{i}{E}{}{n}(\lambdax::nat. {True}))
        unfolding PiE-iff by auto
    moreover have X \in extensional UNIV by simp
    moreover have }\foralli\inUNIV.X i\in space (?PM i) by aut
    ultimately have X prod-emb UNIV ?PM {n} (P\mp@subsup{i}{E}{}{n}(\lambdax::nat. {True}))
using prod-emb-iff [of X] by simp
    }
    thus ?Z \subseteq prod-emb UNIV ?PM {n} (Pi ( 
    qed
    finally have inteq: (to-stream - '?S \cap (space ?wPM)) = prod-emb UNIV ?PM
{n} (P\mp@subsup{i}{E}{}{n}(\lambdax::nat. {True})).
    have emeasure N ?S = emeasure ?wPM (to-stream - ' ?S \cap (space ?wPM))
        using assms emeasure-distr[of to-stream ?wPM (vimage-algebra (streams (space
(measure-pmf (bernoulli-pmf p)))) (!!)
            (PiM UNIV (\lambdai.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\mp@subsup{i}{E}{}{n} (\lambdax::nat.
{True}))) using inteq by simp
    also have ... =
    (\prodi\in{n}. emeasure (?PM i) ((\lambdax::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\leqp and p\leq1
    shows }\foralln\mathrm{ . emeasure N {w, space N. ᄀ(snth wn)}=1-p
proof
    fix n
    let ?A}={w\in\mathrm{ space N. ᄀw!! n}
    let ?B = {w \in space N.w !! n}
    have ?A\cup?B = space N by auto
```

have $? A \cap ? B=\{ \}$ by auto
hence eqA: ? $A=(? A \cup ? B)-? B$ using Diff-cancel by blast
moreover have ? $A \in$ sets $N$

## proof -

have ( $\lambda w$. snth $w n) \in$ measurable $N$ (measure-pmf (bernoulli-pmf $p$ )) using assms by (simp add: bernoulli-stream-def)
moreover have $\{$ True $\} \in$ sets (measure-pmf (bernoulli-pmf p)) by simp
ultimately show ?thesis by simp
qed
moreover have ? $B \in$ sets $N$
proof -
have $(\lambda w$. snth $w n) \in$ measurable $N$ (measure-pmf (bernoulli-pmf $p)$ ) using assms by (simp add: bernoulli-stream-def)
moreover have $\{$ True $\} \in$ sets (measure-pmf (bernoulli-pmf p)) by simp
ultimately show? ?thesis by simp
qed
ultimately have emeasure $N((? A \cup ? B)-? B)=$ emeasure $N(? A \cup ? B)-$ emeasure $N$ ? $B$
proof -
have $f 1: \wedge S m$. (S::bool stream set $) \notin$ sets $m \vee$ emeasure $m S=\top \vee$ emeasure $m$ (space $m$ ) - emeasure $m S=$ emeasure $m($ space $m-S)$
by (metis emeasure-compl infinity-ennreal-def)
have emeasure $N\{s \in$ space $N . s!!n\} \neq \top$
using assms(1) assms(2) assms(3) ennreal-neq-top bernoulli-stream-component-probability by presburger
then have emeasure $N($ space $N)$ - emeasure $N\{s \in$ space $N . s!!n\}=$ emeasure $N$ (space $N-\{s \in$ space $N . s!!n\})$
using $f 1\langle\{w \in$ space $N . w!!n\} \in$ sets $N\rangle$ by blast
then show? ?thesis
using $\langle\{w \in$ space $N . \neg w!!n\} \cup\{w \in$ space $N . w!!n\}=$ space $N\rangle$ by presburger
qed
hence emeasure $N ? A=$ emeasure $N(? A \cup ? B)-$ emeasure $N ? B$ using eq $A$ by $\operatorname{simp}$
also have $\ldots=1$ - emeasure $N$ ? $B$
by (metis (no-types, lifting) $\langle ? A \cup ? B=$ space $N\rangle$ assms(1) bernoulli-stream-def
prob-space.emeasure-space-1 prob-space.prob-space-stream-space prob-space-measure-pmf)
also have $\ldots=1-p$ 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: $0 \leq p$
and $p-l t-1: p \leq 1$
and bernoulli: $M=$ bernoulli-stream $p$
sublocale infinite-coin-toss-space $\subseteq$ 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 $\Rightarrow$ bool stream $\Rightarrow$ bool stream where
pseudo-proj-True $n=(\lambda w$. shift $($ stake $n w)($ sconst True $))$
definition (in infinite-coin-toss-space) pseudo-proj-False:: nat $\Rightarrow$ bool stream $\Rightarrow$ bool stream where

```
    pseudo-proj-False n = (\lambdaw. shift (append (stake n w)[False]) (sconst True))
```

lemma (in infinite-coin-toss-space) pseudo-proj-False-neq-True:
shows pseudo-proj-False $n w \neq$ 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 $\operatorname{simp}$
moreover have snth (pseudo-proj-True $n w$ ) $n=$ True unfolding pseudo-proj-True-def
by $\operatorname{simp}$
ultimately show snth (pseudo-proj-False $n w) n \neq \operatorname{snth}$ (pseudo-proj-True $n w$ )
$n$ by $\operatorname{simp}$
qed
lemma (in infinite-coin-toss-space) pseudo-proj-False-measurable:

```
    shows pseudo-proj-False n \in measurable (bernoulli-stream p) (bernoulli-stream
p)
proof -
    let ?N = bernoulli-stream p
    have id\in measurable ?N ?N by simp
    moreover have (\lambdaw. (sconst True)) \in 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\leqm
    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 \in measurable (bernoulli-stream p) (bernoulli-stream
p)
proof -
    let ? N = bernoulli-stream p
    have id \in measurable ?N ?N by simp
    moreover have (\lambdaw. (sconst True)) \in 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 ? $U \neq\{ \}$ by auto
ultimately have $f$ : finite (stake n'streams ? U) using stake-finite-universe-finite[of ? $U]$ by simp
let ?sh= ( $\lambda l$. shift $l$ (sconst True))
have finite $\{$ ?sh $l \mid l . l \in($ stake $n$ 'streams ? $U)\}$ using $f i$ by simp
moreover have $\{$ ?sh $l \mid l$. $l \in($ stake $n$ 'streams ? $U$ ) $\}=$ range ( $p$ seudo-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 ? $U=U N I V::$ bool set
have ? $U=\{$ True, False $\}$ by auto
hence finite ? U by simp
moreover have ? $U \neq\{ \}$ by auto
ultimately have $f$ : finite (stake $n$ 'streams ? U) using stake-finite-universe-finite[of ?U] by simp
let $? \operatorname{sh}=(\lambda l$. shift ( $l$ @ [False]) (sconst True))
have finite $\{$ ?sh $l \mid l$. $l \in($ stake $n$ 'streams ? $U)\}$ using $f$ by simp
moreover have $\{$ ?sh l|l. $l \in($ 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 $l=n \Longrightarrow$ 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) $w \in\{$ 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 ( $\lambda w$. snth $w n$ ) $\in$ 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 $\bigwedge i$. Suc $i \leq n \Longrightarrow$ 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 $i \leq n$
thus snth $x i=$ snth $w i$ using «stake $n w=$ stake $n x\rangle$ stake-snth by auto
qed
lemma (in infinite-coin-toss-space) pseudo-proj-True-snth ${ }^{\prime}$ :
assumes $(\bigwedge i$. Suc $i \leq n \Longrightarrow$ 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 $n x \Longrightarrow$ pseudo-proj-True $n w=$ pseudo-proj-True $n x$ using pseudo-proj-True-stake-image $[o f ~ n w x]$ by simp
ultimately show ?thesis by auto
qed

```
lemma (in infinite-coin-toss-space) pseudo-proj-True-preimage:
    assumes w= pseudo-proj-True n w
    shows (pseudo-proj-True n) -` {w} =(\bigcapi\in{m.Suc m\leqn}.(\lambdaw. snth wi)
-'{snth wi})
proof
    show (pseudo-proj-True n) -' {w}\subseteq(\bigcapi\in{m. Suc m\leqn}.(\lambdaw. snth wi)-`
{snth wi})
    proof
        fix }
        assume }x\in(pseudo-proj-True n)-`{w
        hence pseudo-proj-True n x = pseudo-proj-True n w using assms by auto
        hence \i. i }\{m.Suc m\leqn}\Longrightarrowx\in(\lambdax. snth x i) - '{ snth wi
            by (metis (mono-tags) Suc-le-eq mem-Collect-eq
            pseudo-proj-True-stake stake-nth vimage-singleton-eq)
        thus }x\in(\bigcapi\in{m.Suc m\leqn}.(\lambdaw. snth wi)-'{snth wi}) by aut
    qed
    show }(\bigcapi\in{m.Suc m\leqn}.(\lambdaw. snth wi)-`{snth wi})\subseteq(pseudo-proj-True
n) -'{ {w}
    proof
        fix }
        assume }x\in(\bigcapi\in{m.Suc m\leqn}.(\lambdaw. snth wi) -'{snth wi}
```



```
        hence \bigwedgei. i\in{m.Suc m\leqn}\Longrightarrow snth x i= snth wi by simp
        hence }\i\mathrm{ . Suc i}\leqn\Longrightarrow\mathrm{ snth x i= snth wi by auto
    hence pseudo-proj-True n x = pseudo-proj-True n w using pseudo-proj-True-snth'[of
nxw] by simp
    also have ... = w using assms by simp
    finally have pseudo-proj-True n x =w.
    thus }x\in(pseudo-proj-True n) - '{w} by auto
    qed
qed
```

lemma (in infinite-coin-toss-space) pseudo-proj-False-preimage:
assumes $w=$ pseudo-proj-False $n w$
shows (pseudo-proj-False $n)-{ }^{\prime}\{w\}=(\bigcap i \in\{m$. Suc $m \leq n\}$. $(\lambda w$. snth $w i)$
-' $\{$ snth $w i\}$ )
proof
show (pseudo-proj-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-proj-False $n)-‘\{w\}$
hence pseudo-proj-False $n x=$ pseudo-proj-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-proj-False $n)-{ }^{\prime}\{w\}$

## proof

fix $x$
assume $x \in(\bigcap i \in\{m$. Suc $m \leq n\}$. ( $\lambda w$. snth $w i)-{ }^{-}\{$snth $\left.w i\}\right)$
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 $\wedge i$. Suc $i \leq n \Longrightarrow$ snth $x i=$ snth $w i$ by auto
hence pseudo-proj-False $n x=$ pseudo-proj-False $n w$
by (metis (full-types) pseudo-proj-False-def stake-snth')
also have $\ldots=w$ using assms by simp
finally have pseudo-proj-False $n x=w$.
thus $x \in($ pseudo-proj-False $n)-‘\{w\}$ by auto
qed
qed
lemma (in infinite-coin-toss-space) pseudo-proj-True-preimage-stake: assumes $w=$ pseudo-proj-True $n w$ shows (pseudo-proj-True $n)-{ }^{\prime}\{w\}=\{$ x. stake n $x=$ stake $n w\}$ proof
show $\{x$. stake $n x=$ stake $n w\} \subseteq($ pseudo-proj-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-proj-True $n x=w$ using assms pseudo-proj-True-def by auto
thus $x \in$ (pseudo-proj-True $n)-‘\{w\}$ by auto
qed
show (pseudo-proj-True $n)-{ }^{\prime}\{w\} \subseteq\{x$. stake $n x=$ stake $n w\}$
proof
fix $x$
assume $x \in$ pseudo-proj-True $n-‘\{w\}$
hence pseudo-proj-True $n x=$ pseudo-proj-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 $\operatorname{simp}$
qed
qed
lemma (in infinite-coin-toss-space) pseudo-proj-False-preimage-stake:
assumes $w=$ pseudo-proj-False $n w$
shows (pseudo-proj-False $n)-‘\{w\}=\{x$. stake $n x=$ stake $n w\}$
proof
show $\{x$. stake $n x=$ stake $n w\} \subseteq($ pseudo-proj-False $n)-{ }^{\prime}\{w\}$
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\in(\mathrm{ pseudo-proj-False n) -'{w} by auto
    qed
    show (pseudo-proj-False n) -' {w}\subseteq{x. stake n x = stake n w}
    proof
    fix }
    assume x\in 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\in{x\mathrm{ . 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}\cap space M = {x\in 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}\cap space M = {x. stake n x = stake n w}\cap
space M
            by simp
    also have ... ={x\in space M. stake nx= 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} \cap(space (bernoulli-stream p)) \in sets(bernoulli-stream
p)
proof (cases {m.(Suc m)\leqn}={})
case False
    have \i. (\lambdax. snth x i) \in measurable (bernoulli-stream p) (count-space UNIV)
by (simp add: measurable-snth-count-space)
    have fi: \bigwedgei. Suc i\leqn\Longrightarrow(\lambdaw. snth wi)-`{snth wi}\cap (space (bernoulli-stream
p)) \in sets (bernoulli-stream p)
    proof -
        fix }
        assume Suc i\leqn
        have (\lambdax. snth x i) \in measurable (bernoulli-stream p) (count-space UNIV) by
(simp add: measurable-snth-count-space)
    moreover have {snth w i} \in sets (count-space UNIV) by auto
    ultimately show ( }\lambda\mathrm{ x. snth x i) -'{snth w i} }\cap(\mathrm{ space (bernoulli-stream p)) }
sets (bernoulli-stream p)
    unfolding measurable-def by (simp add: measurable-snth-count-space)
    qed
```

have $(\bigcap i \in\{m .(S u c m) \leq n\} .(\lambda w$. snth $w i)-‘\{$ snth $w i\} \cap$ (space (bernoulli-stream $p))) \in$ sets (bernoulli-stream $p$ )
proof ((rule sigma-algebra.countable-INT'$\left.{ }^{\prime \prime}\right)$, auto)
show sigma-algebra (space (bernoulli-stream p)) (sets (bernoulli-stream p))
using measure-space measure-space-def by auto
show $U N I V \in$ sets (bernoulli-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-stream $p$ ) $\in$ sets (bernoulli-stream $p$ ) using $f i$ by simp qed
moreover have $(\bigcap i \in\{m .($ Suc $m) \leq n\}$. ( $\lambda$ w. snth $w i)-{ }^{\prime}\{$ snth $w i\} \cap($ space $($ bernoulli-stream $p)))=$
$(\bigcap i \in\{m .($ Suc $m) \leq n\} .(\lambda w$. snth $w i)-‘\{$ snth $w i\}) \cap($ space (bernoulli-stream p))
using False Inter-nonempty-distrib by auto
ultimately show ?thesis using assms pseudo-proj-True-preimage $[o f$ w $n$ ] by simp
next
case True
hence $n=0$ using less-eq-Suc-le by auto
hence $p$ seudo-proj-True $n=(\lambda w$. sconst True) by (simp add: pseudo-proj-True-def)
hence $w=$ sconst True using assms by simp
hence (pseudo-proj-True $n$ ) - ‘ $\{w\} \cap($ space $($ bernoulli-stream $p))=($ space (bernoulli-stream
p)) by (simp add: <pseudo-proj-True $n=(\lambda w$. sconst True) $\rangle)$
thus (pseudo-proj-True $n)-‘\{w\} \cap($ space $($ bernoulli-stream $p)) \in$ sets (bernoulli-stream
p) by $\operatorname{simp}$
qed
lemma (in infinite-coin-toss-space) pseudo-proj-False-singleton:
assumes $w=$ pseudo-proj-False $n w$
shows (pseudo-proj-False $n$ ) - ' $\{w\} \cap($ space (bernoulli-stream $p)) \in$ sets (bernoulli-stream p)
proof (cases $\{m .($ Suc $m) \leq n\}=\{ \})$
case False
have $\bigwedge i$. $(\lambda x$. snth $x i) \in$ measurable (bernoulli-stream p) (count-space UNIV) by (simp add: measurable-snth-count-space)
have $f: \bigwedge i$. Suc $i \leq n \Longrightarrow(\lambda w$. snth $w i)-‘\{$ snth $w i\} \cap$ (space (bernoulli-stream
$p)) \in$ sets (bernoulli-stream $p$ )
proof -
fix $i$
assume Suc $i \leq n$
have $(\lambda x$. snth $x i) \in$ measurable (bernoulli-stream p) (count-space UNIV) by
(simp add: measurable-snth-count-space)
moreover have $\{$ snth $w i\} \in$ sets (count-space UNIV) by auto
ultimately show $(\lambda x$. snth $x i)-‘\{$ snth $w i\} \cap($ space $($ bernoulli-stream $p)) \in$
sets (bernoulli-stream p)
unfolding measurable-def by (simp add: measurable-snth-count-space)
qed
have $(\bigcap i \in\{m .(S u c m) \leq n\} .(\lambda w$. snth $w i)-‘\{$ snth $w i\} \cap$ (space (bernoulli-stream $p))) \in$ sets (bernoulli-stream $p)$
proof ((rule sigma-algebra.countable-INT'$\left.{ }^{\prime \prime}\right)$, auto)
show sigma-algebra (space (bernoulli-stream p)) (sets (bernoulli-stream p))
using measure-space measure-space-def by auto
show $U N I V \in$ sets (bernoulli-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-stream $p$ ) $\in$ sets (bernoulli-stream $p$ ) using $f i$ by simp
qed
moreover have $\left(\bigcap i \in\{m .(\right.$ Suc $m) \leq n\}$. $(\lambda w$. snth $w i)-{ }^{\prime}\{$ snth $w i\} \cap($ space $($ bernoulli-stream $p)))=$
$(\bigcap i \in\{m .($ Suc $m) \leq n\} .(\lambda w$. snth $w i)-‘\{$ snth $w i\}) \cap($ space (bernoulli-stream p))
using False Inter-nonempty-distrib by auto
ultimately show ?thesis using assms pseudo-proj-False-preimage $[o f$ w $n$ ] by $\operatorname{simp}$
next
case True
hence $n=0$ using less-eq-Suc-le by auto
hence pseudo-proj-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-proj-False $n)-‘\{w\} \cap($ space (bernoulli-stream $p))=($ space (bernoulli-stream $p$ ))
by (simp add: <pseudo-proj-False $n=(\lambda w$. False\#\#sconst True) $)$ )
thus (pseudo-proj-False $n$ ) - $\{w\} \cap($ space (bernoulli-stream $p$ ) $) \in$ sets (bernoulli-stream
p) by $\operatorname{simp}$
qed
lemma (in infinite-coin-toss-space) pseudo-proj-True-inverse-induct:
assumes $w \in$ range (pseudo-proj-True $n$ )
shows (pseudo-proj-True $n$ ) $-‘\{w\}=$
(pseudo-proj-True $($ Suc $n))-‘\{w\} \cup($ pseudo-proj-True $($ Suc $n))-‘$ pseudo-proj-False
$n w\}$
proof
let $? y=$ pseudo-proj-False $n w$
show (pseudo-proj-True $n)-‘\{w\} \subseteq($ pseudo-proj-True (Suc $n))-‘\{w\} \cup$
(pseudo-proj-True (Suc n)) - '\{? $y\}$
proof
fix $z$
assume $z \in$ pseudo-proj-True $n-‘\{w\}$
thus $z \in$ pseudo-proj-True (Suc $n)-‘\{w\} \cup$ pseudo-proj-True (Suc n) - $\{? ?\}$
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-proj-True (Suc $n$ ) -' $\{w\}$
hence pseudo-proj-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 nz=w.
    }
    hence fst: pseudo-proj-True (Suc n) -` {w}\subseteq(pseudo-proj-True n) -'{ {w} by
blast
    {
        fix z
        assume z f pseudo-proj-True (Suc n) -' {?y}
        hence pseudo-proj-True nz=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 nz=w.
        }
    hence scd: pseudo-proj-True (Suc n) -` {?y} \subseteq(pseudo-proj-True n) -' {w}
by blast
    show (pseudo-proj-True (Suc n)) -` {w}\cup(pseudo-proj-True (Suc n)) -`{?y}
\subseteq ( p s e u d o - p r o j - T r u e ~ 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 $\Rightarrow$ 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\cap space M |B.B sets(bernoulli-stream p)}
proof -
    have sigma-sets (space M) {pseudo-proj-True n -`}S\cap\mathrm{ 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}\in sets (nat-filtration n)
proof -
    let ?pw = pseudo-proj-True n - {{ w}
    have memset:?pw \in 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 \in sets (nat-filtration n)
    proof -
        have pseudo-proj-True n -'?pw \cap (space M) \in sets (nat-filtration n) using
memset
            by (metis fct-gen-subalgebra-sets-mem nat-filtration-def)
    moreover have pseudo-proj-True n-'?pw \cap (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 \in 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 $n \in$ 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 $\Rightarrow{ }^{\prime} b::\{t 0$-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 $A h=$ this
have $A \in$ sets borel using $A h$ by simp
hence $\exists$ A 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 $w h$ $=$ 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 $f n: f-{ }^{\prime} A \in$ sets (nat-filtration $n$ ) using nat-filtration-space by simp
have ? $x \in f-‘ A=(w \in f-‘ A)$ using $\prec p s e u d o-p r o j-T r u e ~ n ? x=$ 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 ... = (fw\inA) by simp
    also have ... = (f ?x\not\inA) using Ah by simp
    also have ... = (?x\not\inf-'A) by simp
    finally have ?}x\inf-'A=(?x\not\inf-'A)
    thus False by simp
qed
```

lemma (in infinite-coin-toss-space) nat-filtration-info:
fixes $f::$ bool stream $\Rightarrow{ }^{\prime} b::\{t 0$-space $\}$
assumes $f \in$ borel-measurable (nat-filtration $n$ )
shows $f \circ$ pseudo-proj-True $n=f$
proof (rule nat-filtration-not-borel-info)
show $f \in$ 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 $f \in$ measurable (nat-filtration $n$ ) $N$
and set-discriminating $n f N$
shows $f \circ$ pseudo-proj-False $n=f$
proof
fix $x$
have ( $f \circ$ pseudo-proj-False $n$ ) $x=f$ (pseudo-proj-False $n x$ ) by simp
also have $\ldots=f$ (pseudo-proj-True $n$ (pseudo-proj-False $n x$ )) using assms nat-filtration-not-borel-info
by (metis comp-apply)
also have $\ldots=f$ (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 $\ldots=f x$ using assms nat-filtration-not-borel-info by (metis comp-apply)
finally show $(f \circ$ pseudo-proj-False $n) x=f x$.
qed
lemma (in infinite-coin-toss-space) nat-filtration-info':
fixes $f::$ bool stream $\Rightarrow{ }^{\prime} b::\{t 0$-space $\}$
assumes $f \in$ borel-measurable (nat-filtration $n$ )

```
    shows fo pseudo-proj-False n =f
proof
    fix }
    have (f\circ 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 ... = fx using assms nat-filtration-info by (metis comp-apply)
    finally show ( }f\circ\mathrm{ р peudo-proj-False n) }x=fx\mathrm{ .
qed
```

lemma (in infinite-coin-toss-space) nat-filtration-borel-measurable-characterization:
fixes $f::$ bool stream $\Rightarrow{ }^{\prime} b::\{t 0$-space $\}$
assumes $f \in$ borel-measurable $M$
shows $f \in$ borel-measurable (nat-filtration $n$ ) $\longleftrightarrow f \circ$ 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\in borel-measurable (nat-filtration 0)
    shows f}=(\lambdaw.f\mathrm{ (sconst True))
proof
    fix w
    have fw=f((pseudo-proj-True 0)w) using assms nat-filtration-info[off 0] by
(metis comp-apply)
    also have ... =f (sconst True) by (simp add: pseudo-proj-True-def)
    finally show fw=f(sconst True).
qed
```

lemma (in infinite-coin-toss-space) nat-filtration-Suc-sets:
shows sets (nat-filtration $n) \subseteq$ sets (nat-filtration (Suc n))
proof -
\{
fix $x$
assume $x \in\left\{\right.$ pseudo-proj-True $n-{ }^{\prime} B \cap$ space $M \mid B . B \in$ sets $\left.M\right\}$
hence $\exists B . B \in$ sets $M \wedge x=$ pseudo－proj－True $n-{ }^{\prime} B \cap$ space $M$ by auto
from this obtain $B$ where $B \in$ sets $M$ and $x=$ pseudo－proj－True $n-{ }^{‘} B \cap$ space $M$
by blast note xhyps $=$ this
let ？Bim $=B \cap($ range（pseudo－proj－True $n))$
let ？preT $=(\lambda n w .($ pseudo－proj－True $n)-‘\{w\})$
have finite？Bim using pseudo－proj－True－finite－image by simp
have pseudo－proj－True $n-' B \cap($ space $M)=$ pseudo－proj－True $n-' B$
using bernoulli bernoulli－stream－preimage［of－－pseudo－proj－True n］by simp
also have $\ldots=$ pseudo－proj－True $n-$＇？Bim by auto
also have $\ldots=(\bigcup w \in$ ？Bim．？preT $n w)$ by auto
also have $\ldots=(\bigcup w \in$ ？Bim．（？preT（Suc $n) w \cup$ ？preT（Suc $n$ ）（pseudo－proj－False $n w)$ ）
by（simp add：pseudo－proj－True－inverse－induct）
also have $\ldots=(\bigcup w \in$ ？Bim．？preT $($ Suc $n) w) \cup(\bigcup w \in$ ？Bim．？preT （Suc $n$ ）（pseudo－proj－False $n w$ ））by auto
finally have tmpeq：pseudo－proj－True $n-' B \cap($ space $M)=$
$(\bigcup w \in$ ？Bim．？preT $($ Suc $n) w) \cup(\bigcup w \in$ ？Bim．？preT（Suc $n)$ （pseudo－proj－False $n w)$ ）．
have $(\bigcup w \in$ ？Bim．？preT（Suc n）w）$\in$ sets（nat－filtration（Suc n））
using〈finite ？Bim〉nat－filtration－singleton pseudo－proj－True－Suc－proj by auto
moreover have $(\bigcup w \in$ ？Bim．？pre $T$（Suc n）（pseudo－proj－False $n w)) \in$ sets（nat－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－filtration（Suc n））
using tmpeq xhyps by simp
\} note $x m e m=$ this
have sets $($ nat－filtration $n)=$ sigma－sets $($ space $M)\{p s e u d o-p r o j-T r u e ~ n-' B \cap$ space $M \mid B . B \in$ sets $M\}$
using bernoulli nat－filtration－sets by blast
also have $\ldots \subseteq($ nat－filtration（Suc n））
proof（rule sigma－algebra．sigma－sets－subset）
show $\{$ pseudo－proj－True $n-‘ B \cap$ space $M \mid B . B \in$ sets $M\}$
$\subseteq$ sets（nat－filtration（Suc n））using xmem by auto
show sigma－algebra（space M）（sets（nat－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－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 \foralln m.n\leqm\longrightarrow subalgebra (nat-filtration m)(nat-filtration n)
    proof (intro allI impI)
        let ?F = nat-filtration
        fix n::nat
        fix m
        show }n\leqm\Longrightarrow\mathrm{ 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)\subseteq sets (?F (Suc m)) using nat-filtration-Suc-sets by
simp
            qed
            thus n\leqSuc m\Longrightarrow 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-fil M nat-filtration unfolding init-triv-filt-def
proof
    show filtration M nat-filtration by (simp add:nat-discrete-filtration)
    have img: }\forall\mathrm{ wG 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}\subseteq 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) \subseteq{{}, space M}
            proof -
            have }\forallB\in\mathrm{ sets (bernoulli-stream p). pseudo-proj-True 0 -' B }\cap\mathrm{ space }M
{{}, space M}
            proof
```

```
    fix }
    assume B\in sets (bernoulli-stream p)
    show pseudo-proj-True 0-'}B\cap\mathrm{ space }M\in{{}\mathrm{ , space M}
    proof (cases sconst True \inB)
        case True
        hence pseudo-proj-True 0 - ' B \cap space M = space M using img by auto
        thus ?thesis by auto
        next
            case False
            hence pseudo-proj-True 0 -' B\cap space M = {} using img by auto
            thus ?thesis by auto
        qed
    qed
    hence {pseudo-proj-True 0-'}B\cap\mathrm{ space M|B. B 的新(bernoulli-stream
p)}\subseteq{{}, space M} by auto
    hence sigma-sets (space (bernoulli-stream p))
                {pseudo-proj-True 0 -' B\cap space M | B. B\in sets(bernoulli-stream p)}
\subseteq \{ \{ \} , ~ s p a c e ~ 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 $\subseteq$ 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 $f:: b o o l$ stream $\Rightarrow{ }^{\prime} b::\{t 2-$ space $\}$
assumes $f \in$ borel-measurable (nat-filtration n)
shows finite ( $f^{\prime}($ 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 $\Rightarrow ' b::\{t 2$-space $\}$
assumes $f \in$ borel-measurable (nat-filtration $n$ )
shows simple-function $M f$
proof -
have f1: $\forall m$ ma. ( $m::$ bool stream measure $) \rightarrow_{M}$ (ma::'b measure $)=\{f \in$ space $m \rightarrow$ space ma. $\forall B . B \in$ sets $m a \longrightarrow f-{ }^{\prime} B \cap$ space $m \in$ sets $\left.m\right\}$
by (metis measurable-def)
then have $f \in$ space (nat-filtration $n) \rightarrow$ space borel $\wedge(\forall B . B \in$ sets borel $\longrightarrow$ $f-' B \cap$ space (nat-filtration $n) \in$ sets (nat-filtration $n)$ )
using assms by blast
then have $f \in$ space $M \rightarrow$ space borel $\wedge\left(\forall B . B \in\right.$ sets borel $\longrightarrow f-{ }^{\prime} B \cap$ space $M \in$ events) by (metis (no-types) contra-subsetD nat-filtration-subalgebra subalgebra-def) then have random-variable borel $f$ using $f 1$ 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 $\Rightarrow{ }^{\prime} b::\{t 2-$ space $\}$
assumes $f \in$ borel-measurable (nat-filtration $n$ )
shows $\exists A \in$ sets borel. range $f \cap A=\{f x\}$
proof -
let $? A x=$ 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 $\exists U$. open $U \wedge f x \in U \wedge U \cap$ ?Ax $=\{ \}$ by (simp add:open-except-set)
then obtain $U$ where open $U$ and $f x \in U$ and $U \cap ? A x=\{ \}$ by auto
have $U \in$ sets borel using <open $U$ 〉 by simp
have range $f \cap U=\{f x\}$ using $\langle f x \in U\rangle\langle U \cap ? A x=\{ \}\rangle$ by blast
thus $\exists A \in$ sets borel. range $f \cap A=\{f x\}$ using $\langle U \in$ sets borel $\rangle$ by auto qed
lemma (in infinite-coin-toss-space) nat-filtration-borel-measurable-singleton:
fixes $f::$ bool stream $\Rightarrow$ ' $b::\{t 2-$ space $\}$
assumes $f \in$ borel-measurable (nat-filtration $n$ )
shows $f-\{f x\} \in$ sets (nat-filtration $n$ )
proof -
let ? $A x=f^{\prime}$ space $M-\{f x\}$
have finite ?Ax
using assms nat-filtration-vimage-finite by blast
hence $\exists U$. open $U \wedge f x \in U \wedge U \cap$ ?Ax $=\{ \}$ by (simp add:open-except-set)
then obtain $U$ where open $U$ and $f x \in U$ and $U \cap$ ? $A x=\{ \}$ by auto
have $f x \in f$ ' space $M$ using bernoulli-stream-space bernoulli by simp
hence $f$ 'space $M \cap U=\{f x\}$ using $\langle f x \in U\rangle\langle U \cap$ ?Ax $=\{ \}\rangle$ by blast
hence $\exists A$. open $A \wedge f^{\prime}$ space $M \cap A=\{f x\}$ using <open $\left.U\right\rangle$ 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 $A$ b by simp
hence $f-{ }^{\prime} A \cap$ 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 $f-{ }^{\prime}\{f x\} \cap$ space $M \subseteq f-{ }^{\prime} A \cap$ 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 $f y \in A \cap f^{\prime}$ space $M$ by simp
hence $f y=f x$ using inter by auto
thus $y \in f-‘\{f x\} \cap$ space $M$ using $\langle y \in f-‘ A \cap$ space $M>$ by auto
qed
qed
moreover have $f-' A \cap$ space $M \in($ nat-filtration $n)$ using assms $\langle A \in$ sets borels
using $\langle f-‘ A \cap$ 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 $\Rightarrow$ ' $b::\{t 0$-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 Mf
using Bochner-Integration.simple-bochner-integrable.simps by blast
hence has-bochner-integral Mf(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 $\Rightarrow$ nat $\Rightarrow$ bool $\Rightarrow$ 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 $n w \wedge$ 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 $\operatorname{simp}$
lemma (in infinite-coin-toss-space) spick-pseudo-proj:
shows spick (pseudo-proj-True (Suc n) w) nv=spick wnv
by (metis pseudo-proj-True-proj-Suc pseudo-proj-True-stake spick-def)
lemma (in infinite-coin-toss-space) spick-pseudo-proj-gen:
shows $m<n \Longrightarrow$ spick (pseudo-proj-True $n w$ ) mv=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 (\lambdaw. spick wnv)\in measurable (nat-filtration n) M
proof (rule nat-filtration-comp-measurable)
    show ( }\lambdaw\mathrm{ . spick wnv) € measurable M M
    proof -
        let ? N = bernoulli-stream p
        have id \in measurable ?N ?N by simp
        moreover have ( }\lambdaw.v## (sconst True)) \in 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) nv = spick w n v
```

```
        by (simp add: pseudo-proj-True-stake spick-def)
    }
    thus (\lambdaw. spick wnv) ○ pseudo-proj-True n=(\lambdaw. 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-fil-contain:
    assumes A\in sets (nat-filtration n)
and w\inA
shows pseudo-proj-True n -' {pseudo-proj-True n w}\subseteqA
proof
    define indA where indA = (( indicator A)::bool stream }=>\mathrm{ real }
    have indA \in borel-measurable (nat-filtration n) unfolding indA-def
        by (simp add: assms(1) borel-measurable-indicator)
    fix }
    assume x f 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 \in borel-measurable
(nat-filtration n)>
    unfolding indA-def by (metis comp-apply)
    also have ... = indA (pseudo-proj-True n w) using <x \in pseudo-proj-True n -`
{pseudo-proj-True n w}
    by simp
    also have ... = indA w using nat-filtration-info[of indicator A n]
            <indA \in borel-measurable (nat-filtration n)> unfolding indA-def by (metis
comp-apply)
    also have ... = 1 using assms unfolding indA-def by simp
    finally have }\operatorname{indA}x=1
    thus }x\inA\mathrm{ unfolding indA-def by (simp add: indicator-eq-1-iff)
qed
```

lemma (in infinite-cts-filtration) measurable-range-rep:
fixes $f::$ bool stream $\Rightarrow{ }^{\prime} b::\{t 0$-space $\}$
assumes $f \in$ borel-measurable (nat-filtration $n$ )
shows range $f=(\bigcup r \in($ proj-rep-set $n) .\{f(r)\})$
proof -
have $f=f \circ$ (pseudo-proj-True $n$ ) using assms nat-filtration-info[off $n$ ] by simp
hence range $f=f$ '(proj-rep-set $n$ ) by (metis fun.set-map proj-rep-set)

```
    also have ... = (\bigcupr\inproj-rep-set n. {f r}) by blast
    finally show range f}=(\bigcupr\inproj-rep-set n. {fr})
qed
lemma (in infinite-coin-toss-space) borel-measurable-stake:
    fixes f::bool stream = 'b::{t0-space}
    assumes f\in borel-measurable (nat-filtration n)
    and stake n w = stake n y
shows fw =fy
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 \(p w n \neq 0\) using assms prob-component-def by auto
lemma prob-component-measure:
    fixes \(x\) ::bool stream
assumes \(0 \leq p\)
and \(p \leq 1\)
    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 \(0 \leq n\) and \(M=\) bernoulli-stream \(p\)
    shows \(\{w \in\) space \(M\). (stake \(n w=\) stake \(n x)\} \in\) sets \(M\)
proof -
    let \(? S=\{w \in\) space \(M\). (stake \(n w=\) stake \(n x)\}\)
    have \(? S=(\bigcap i \in\{0 . . n-1\}\). \(\{w \in\) space \(M\). (snth \(w i=\) snth \(x i)\})\) using
stake-inter-snth assms by simp
    moreover have \((\bigcap i \in\{0 . . n-1\} .\{w \in \operatorname{space} M .(\operatorname{snth} w i=\operatorname{snth} x i)\}) \in\) sets
M
    proof -
```

```
    have }\foralli\leqn-1.{w\in space M. (snth wi= snth xi)}\in sets 
    proof (intro allI impI)
    fix }
    assume i\leqn-1
    thus {w\in space M.w !! i=x \!! i}\in sets M
    proof -
    have (\lambdaw. snth wi)\in 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\in space M. snth wi=b} ={X::nat=>bool. Xi=b}
proof -
    let ?S = {w\in space M. snth wi=b}
    let ?PM = (\lambdai::nat. (measure-pmf (bernoulli-pmf p)))
    have isps: product-prob-space ?PM by unfold-locales
    let ?Z = {X::nat }=>\mathrm{ bool. }Xi=b
    show to-stream - '?S = ?Z by (simp add: assms bernoulli-stream-space to-stream-def)
qed
lemma stake-as-fct:
    assumes Suc 0 \leqn and M= bernoulli-stream p
    shows to-stream - '{w\in space M. (stake n w = stake n x) }={X::nat=>bool. \foralli.
0\leqi^i\leqn-1 \longrightarrowXi= snth x i}
proof -
    let ?S = {w\in space M. (stake n w = stake n x) }
    let ?Z ={X::nat=>bool. }\foralli.0\leqi\wedgei\leqn-1\longrightarrowXi=\mathrm{ snth x i }
    have to-stream -'?}S=\mathrm{ to-stream -' (\i ( {0..n-1}. {w space M. (snth w
i= snth xi)})
            using <Suc 0 \leqn> stake-inter-snth by blast
    also have ... = (\bigcapi\in{0..n-1}. to-stream - '{w\in space M. (snth wi= snth
x i)}) by auto
    also have ... =(\bigcap i\in{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 }
    assumes M = bernoulli-stream p
```

```
    shows emeasure M {w\in space M. (stake 0 w = stake 0x)}=1
proof -
    define S where S={w\in space M. (stake 0 w = stake 0 }0\mathrm{ ) }
    have}S=\mathrm{ 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:
```

lemma bernoulli-stream-pref-prob:
fixes $x$
fixes $x$
assumes $M=$ bernoulli-stream $p$
assumes $M=$ bernoulli-stream $p$
and $0 \leq p$ and $p \leq 1$
and $0 \leq p$ and $p \leq 1$
shows $n \geq$ Suc $0 \Longrightarrow$ emeasure $M\{w \in$ space $M$. (stake $n w=$ stake $n x)\}=$
shows $n \geq$ Suc $0 \Longrightarrow$ emeasure $M\{w \in$ space $M$. (stake $n w=$ stake $n x)\}=$
( $\prod_{i \in\{0 . . n-1\} .}$ prob-component $\left.p x i\right)$
( $\prod_{i \in\{0 . . n-1\} .}$ prob-component $\left.p x i\right)$
proof -
proof -
have prob-space $M$
have prob-space $M$
by (simp add: assms bernoulli-stream-def prob-space.prob-space-stream-space
by (simp add: assms bernoulli-stream-def prob-space.prob-space-stream-space
prob-space-measure-pmf)
prob-space-measure-pmf)
fix $n:: n a t$
fix $n:: n a t$
assume $n \geq$ Suc 0
assume $n \geq$ Suc 0
define $S$ where $S=\{w \in$ space $M$. (stake $n w=$ stake $n x)\}$
define $S$ where $S=\{w \in$ space $M$. (stake $n w=$ stake $n x)\}$
have $s: S \in$ sets $M$ unfolding $S$-def by (simp add: assms stake-preimage-measurable
have $s: S \in$ sets $M$ unfolding $S$-def by (simp add: assms stake-preimage-measurable
Suc $0 \leq n\rangle$ )
Suc $0 \leq n\rangle$ )
define $P M$ where $P M=(\lambda i:: n a t$. $($ measure-pmf $($ bernoulli-pmf $p)))$
define $P M$ where $P M=(\lambda i:: n a t$. $($ measure-pmf $($ bernoulli-pmf $p)))$
have isps: product-prob-space PM unfolding PM-def by unfold-locales
have isps: product-prob-space PM unfolding PM-def by unfold-locales
define $Z$ where $Z=\{X::$ nat $\Rightarrow$ bool. $\forall i .0 \leq i \wedge i \leq n-1 \longrightarrow X i=$ snth $x i\}$
define $Z$ where $Z=\{X::$ nat $\Rightarrow$ bool. $\forall i .0 \leq i \wedge i \leq n-1 \longrightarrow X i=$ snth $x i\}$
let ? $w P M=P i_{M}$ UNIV PM
let ? $w P M=P i_{M}$ UNIV PM
define $i m g S b s$ where $i m g S b s=$ prod-emb UNIV PM $\{0 . . n-1\}\left(P i_{E}\{0 . . n-1\}\right.$
define $i m g S b s$ where $i m g S b s=$ prod-emb UNIV PM $\{0 . . n-1\}\left(P i_{E}\{0 . . n-1\}\right.$
( $\lambda i::$ nat. $\{$ snth $x i\})$ )
( $\lambda i::$ nat. $\{$ snth $x i\})$ )
have space ?wPM = UNIV using space-PiM unfolding PM-def by fastforce
have space ?wPM = UNIV using space-PiM unfolding PM-def by fastforce
hence (to-stream -' $S \cap($ space ? $w P M)$ ) $=$ to-stream $-{ }^{\prime} S$ by simp
hence (to-stream -' $S \cap($ space ? $w P M)$ ) $=$ to-stream $-{ }^{\prime} S$ by simp
also have $\ldots=Z$ using stake-as-fct 〈Suc $0 \leq n\rangle$ assms unfolding $Z$-def $S$-def
also have $\ldots=Z$ using stake-as-fct 〈Suc $0 \leq n\rangle$ assms unfolding $Z$-def $S$-def
by $\operatorname{simp}$
by $\operatorname{simp}$
also have $\ldots=i m g S b s$
also have $\ldots=i m g S b s$
proof
proof
\{
\{
fix $X$
fix $X$
assume $X \in i m g S b s$
assume $X \in i m g S b s$
hence restrict $X\{0 . . n-1\} \in\left(P i_{E}\{0 . . n-1\}\right.$ ( $\lambda i:: n a t$. $\{$ snth $\left.x i\}\right)$ ) using
hence restrict $X\{0 . . n-1\} \in\left(P i_{E}\{0 . . n-1\}\right.$ ( $\lambda i:: n a t$. $\{$ snth $\left.x i\}\right)$ ) using
prod-emb-iff $[$ of $X$ ] unfolding imgSbs -def by simp
prod-emb-iff $[$ of $X$ ] unfolding imgSbs -def by simp
hence $\forall i .0 \leq i \wedge i \leq n-1 \longrightarrow X i=$ snth $x i$ by auto
hence $\forall i .0 \leq i \wedge i \leq n-1 \longrightarrow X i=$ snth $x i$ by auto
hence $X \in Z$ unfolding $Z$-def by simp
hence $X \in Z$ unfolding $Z$-def by simp
\}
\}
thus $i m g S b s \subseteq Z$ by blast
thus $i m g S b s \subseteq Z$ by blast
\{
\{
fix $X$

```
        fix \(X\)
```

```
assume X \inZ
    hence }\foralli.0\leqi\wedgei\leqn-1\longrightarrowXi= snth x i unfolding Z-def by sim
    hence restrict X {0..n-1}\in(Pi\mp@subsup{i}{E}{}{0..n-1} (\lambdai::nat. {snth x i})) by simp
    moreover have X extensional UNIV by simp
    moreover have }\foralli\inUNIV.X i\in space (PM i) unfolding PM-def by aut
    ultimately have }X\inimgSb
    using prod-emb-iff[of X] unfolding imgSbs-def by simp
```

    \}
    thus \(Z \subseteq\) imgSbs by auto
    qed
    finally have inteq: (to-stream -' \(S \cap(\) space ? \(w P M))=i m g S b s\).
    have emeasure \(M S=\) emeasure ? \(w P M(\) to-stream \(-‘ S \cap(\) space ? \(w P M))\)
    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 $\ldots=$ emeasure ?wPM imgSbs using inteq by simp
also have $\ldots=\left(\prod i \in\{0 . . n-1\}\right.$. emeasure (PM i) $((\lambda m:: n a t$. $\{$ snth $\left.x m\}) i)\right)$
using isps unfolding imgSbs-def PM-def by (auto simp add:product-prob-space.emeasure-PiM-emb)
also have $\ldots=\left(\prod i \in\{0 . . n-1\}\right.$. prob-component $\left.p x i\right)$ using prob-component-measure
unfolding $P M$-def
proof -
have f1: $\forall N f .(\exists n .(n:: n a t) \in N \wedge \neg 0 \leq f n) \vee\left(\prod n \in N\right.$. ennreal $\left.(f n)\right)=$
ennreal $(\operatorname{prod} f N)$
by (metis (no-types) prod-ennreal)
obtain $n n::($ nat $\Rightarrow$ real $) \Rightarrow$ nat set $\Rightarrow$ nat where
f2: $\forall x 0 x 1 .(\exists v 2 . v 2 \in x 1 \wedge \neg 0 \leq x 0 v 2)=(n n x 0 x 1 \in x 1 \wedge \neg 0 \leq$
$x 0(n n x 0 x 1))$
by moura
have f3: $\forall s n$. if $s$ !! $n$ then prob-component $p s=p$ else $p+$ prob-component ps $n=1$
by (simp add: prob-component-def)
$\{$ assume prob-component $p x(n n($ prob-component $p x)\{0 . . n-1\}) \neq p$
then have $p+$ prob-component $p x(n n($ prob-component $p x)\{0 . . n-1\})$
$=1$
using f3 by metis
then have $n n($ prob-component $p x)\{0 . . n-1\} \notin\{0 . . n-1\} \vee 0 \leq$ prob-component $p x$ (nn (prob-component $p x)\{0 . . n-1\})$
using assms by linarith \}
then have $n n$ (prob-component $p x)\{0 . . n-1\} \notin\{0 . . n-1\} \vee 0 \leq$ prob-component $p x(n n$ (prob-component $p x)\{0 . . n-1\})$ using assms by linarith
then have $\left(\prod n=0 . . n-1\right.$. ennreal $($ prob-component $\left.p x n)\right)=$ ennreal $(p r o d$ (prob-component $p x)\{0 . . n-1\}$ )
using f2 f1 by meson
moreover have $\left(\prod n=0 . . n-1\right.$. ennreal $($ prob-component $\left.p x n)\right)=$
$\left(\prod n=0 . . n-1\right.$. emeasure (measure-pmf (bernoulli-pmf $p$ )) $\{x!!n\}$ ) using prob-component-measure[of $p x]$
assms by simp
ultimately show $\left(\prod n=0 . . n-1\right.$. emeasure (measure-pmf (bernoulli-pmf $\left.p\right)$ ) $\{x!!n\})=$ ennreal $($ prod $($ prob-component $p x)\{0 . . n-1\})$
using prob-component-measure $[$ of $p x]$ by simp

## qed

finally show emeasure $M S=\left(\prod i \in\{0 . . n-1\}\right.$. prob-component $\left.p x i\right)$.
qed
lemma bernoulli-stream-pref-prob':
fixes $x$
assumes $M=$ bernoulli-stream $p$
and $p \leq 1$ and $0 \leq p$
shows emeasure $M\{w \in$ space $M .($ stake $n w=$ stake $n x)\}=\left(\prod i \in\{0 . .<n\}\right.$.
prob-component $p x i)$
proof (cases Suc $0 \leq n$ )
case True
hence emeasure $M\{w \in$ space $M$. (stake $n w=$ stake $n x)\}=\left(\prod i \in\{0 . . n-1\}\right.$.
prob-component $p x i$ ) using assms
by (simp add: bernoulli-stream-pref-prob)
moreover have $\left(\prod i \in\{0 . . n-1\}\right.$. prob-component p $\left.x i\right)=\left(\prod i \in\{0 . .<n\}\right.$. prob-component p $x i)$
proof (rule prod.cong)
show $\{0 . . n-1\}=\{0 . .<n\}$ using True by auto
show $\bigwedge x a . x a \in\{0 . .<n\} \Longrightarrow$ prob-component $p x x a=$ prob-component $p x$ xa
by $\operatorname{simp}$
qed
ultimately show?thesis by simp
next
case False
hence $n=0$ using False by simp
have $\{w \in$ space $M$. (stake $n w=$ stake $n x)\}=$ space $M$
proof
show $\{w \in$ space $M$. stake $n w=$ stake $n x\} \subseteq$ space $M$
proof
fix $w$
assume $w \in\{w \in$ space M. stake $n w=$ stake $n x\}$
thus $w \in$ space $M$ by auto
qed
show space $M \subseteq\{w \in$ space $M$. stake $n w=$ stake $n x\}$
proof
fix $w$
assume $w \in$ space $M$
have stake $0 w=$ stake $0 x$ by simp
hence stake $n w=$ stake $n x$ using $\langle n=0\rangle$ by simp
thus $w \in\{w \in$ space $M$. stake $n w=$ stake $n x\}$ using $\langle w \in$ space $M\rangle$ by auto qed
qed
hence emeasure $M\{w \in$ 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 ... = (\prodi\in{0..<n}. prob-component p x i) using <n=0> by simp
    finally show ?thesis.
qed
lemma bernoulli-stream-stake-prob:
    fixes }
    assumes M= bernoulli-stream p
and p\leq1 and 0\leqp
shows measure M {w\in space M. (stake n w = stake n x)}=(\prodi\in{0..<n}.
prob-component p x i)
proof -
    have measure M {w\in space M. (stake n w = stake n x)}= emeasure M {w\in
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 ... = (\prodi\in{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 }
    assumes M= bernoulli-stream p
and p\leq1 and 0\leqp
and w\in range (pseudo-proj-True n)
shows measure M (pseudo-proj-True n -'{w}\cap space M)=(\prodi\in{0..<n}.prob-component
p wi)
proof -
    have (pseudo-proj-True n-'{w})\cap space M={x\in space M. (stake n w = stake
nx)}
    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 }\langlew\in\mathrm{ 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 }
    assumes M = bernoulli-stream p
and 0\leqp and p\leq1
    shows \bigwedgen. emeasure M {w\in space M. (stake (Suc n) w = stake (Suc n) x)}=
        (emeasure M{w\in space M. (stake n w = stake n x)} * prob-component p x n)
proof -
    fix n
    define S where S = {w\in space M. (stake (Suc n) w = stake (Suc n) x)}
    define precS where precS = {w\in space M. (stake n w = stake n x) }
    show emeasure MS = emeasure M precS * prob-component p x n
    proof (cases n\leq0)
        case True
        hence n=0 by simp
    hence emeasure MS = (\prodi\in{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 MS = emeasure M precS * prob-component p x n.
    next
        case False
    hence n \geqSuc 0 by simp
    hence emeasure MS = (\prodi\in{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 ... = (\prodi\in{0..n-1}. prob-component p x i)* prob-component p x n
using <n \geqSuc 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 \leqn` ennreal-mult" assms prob-component-def by auto
    finally show emeasure MS = emeasure M precS * prob-component p x n.
    qed
qed
lemma bernoulli-stream-element-prob-rec':
    fixes }
    assumes M = bernoulli-stream p
and 0\leqp and p\leq1
    shows \n. measure M {w\in space M. (stake (Suc n) w= stake (Suc n) x)}=
        (measure M {w\in space M. (stake n w = stake n x) } * prob-component p x n)
proof -
    fix n
    have ennreal (measure M {w\in space M. (stake (Suc n) w= stake (Suc n) x)})
=
    emeasure M {w\in 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 $\ldots=($ emeasure $M\{w \in$ space $M .($ stake $n w=$ stake $n x)\} *$ prob-component $p x n$ )
using bernoulli-stream-element-prob-rec assms by simp
also have $\ldots=($ measure $M\{w \in$ 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 $M\{w \in$ space $M$. (stake (Suc n) $w=$ stake (Suc n) $x)\}$ ) $=$
(measure $M\{w \in$ space $M .($ stake $n w=$ stake $n x)\} *$ prob-component $p x n)$.
thus measure $M\{w \in$ space $M$. (stake (Suc n) $w=\operatorname{stake}($ Suc $n) x)\}=$
(measure $M\{w \in$ space $M$. (stake $n w=$ stake $n x)\}$ * prob-component $p \times n$ )
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 ${ }^{\prime}$ 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 $M\{w \in$ space $M .($ stake $n w=$ stake $n x)\}>0$
proof (induct $n$ )
case 0
hence emeasure $M\{w \in$ 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\in space M. stake (Suc n) w = stake (Suc n) x} =
    (emeasure M {w\in 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 }
assumes 0<p
and p<1
    shows emeasure M {w\in space M. (stake n w= stake n x)}\not=0
proof (induct n)
    case 0
    hence emeasure M{w\in space M. (stake 0w= stake 0x)}=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\in space M. stake (Suc n) w= stake (Suc n) x}=
        (emeasure M{w\in 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 $w \in$ range (pseudo-proj-True $n$ )
shows emeasure $M\{y \in$ space $M . \exists x \in(p s e u d o-p r o j-T r u e n-‘\{w\}) . y=c \# \#$
$x\}=$
prob-component $p(c \# \# w) 0 *$ emeasure $M(($ pseudo-proj-True $n)-‘\{w\} \cap$
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 $\{y \in$ space $M . \exists x \in($ pseudo-proj-True $n-‘\{w\}) . y=c \# \# x\}=$ pseudo-proj-True
(Suc n) -‘\{c\#\#w\} $\cap$ space $M$
proof
show $\{y \in$ space $M . \exists x \in$ pseudo-proj-True $n-'\{w\} . y=c \# \# x\} \subseteq$ pseudo-proj-True
(Suc n) -‘ $\{c \# \# w\} \cap$ space $M$

## proof

fix $y$
assume $y \in\{y \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{w\} . y=c \# \# x\}$
hence $y \in$ space $M$ and $\exists x \in$ pseudo-proj-True $n-‘\{w\} . y=c \# \# x$ by auto
from this obtain $x$ where $x \in$ pseudo-proj-True $n-‘\{w\}$ and $y=c \# \# x$ by auto
have pseudo-proj-True (Suc n) $y=c \# \# w$ using $\langle x \in$ pseudo-proj-True $n-‘$ $\{w\}\rangle\langle y=c \# \# x\rangle$
unfolding pseudo-proj-True-def by simp
thus $y \in$ pseudo-proj-True (Suc $n$ ) -‘ $\{c \# \# w\} \cap$ space $M$ using $\prec y \in$ space
$M$ > by auto
qed
show pseudo-proj-True (Suc $n$ ) -‘ $\{c \# \# w\} \cap$ space $M \subseteq\{y \in$ space $M$. $\exists x \in$ pseudo-proj-True $n-‘\{w\} . y=c \# \# x\}$
proof
fix $y$
assume $y \in$ pseudo-proj-True $($ Suc $n)-‘\{c \# \# w\} \cap$ space $M$
hence pseudo-proj-True (Suc n) $y=c \# \# w$ and $y \in$ space $M$ by auto
have pseudo-proj-True $n$ (stl $y$ ) $=$ pseudo-proj-True $n w$
proof (rule pseudo-proj-True-snth')
have pseudo-proj-True (Suc n) $(c \# \# w)=c \# \# w$ using $<p s e u d o-p r o j-T r u e$ (Suc $n)(c \# \# w)=c \# \# w$.
also have $\ldots=$ pseudo-proj-True (Suc n) y using <pseudo-proj-True (Suc
n) $y=c \# \# w\rangle$ by $\operatorname{simp}$
finally have pseudo-proj-True (Suc n) $(c \# \# w)=$ pseudo-proj-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 $\ldots=w$ using assms pseudo-proj-True-def pseudo-proj-True-stake by auto
finally have pseudo-proj-True $n($ stl $y)=w$.
hence stl $y \in$ (pseudo-proj-True $n)-‘\{w\}$ by simp
moreover have $y=c \# \#(s t l y)$
proof -
have stake (Suc n) $y=$ stake (Suc $n$ ) (pseudo-proj-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-proj-True (Suc n) y) by simp
also have $\ldots=\operatorname{shd}(c \# \# w)$ using $\langle p s e u d o-p r o j-T r u e(S u c ~ n) ~ y=c \# \# w\rangle$
by $\operatorname{simp}$
also have $\ldots=c$ by simp
finally have shd $y=c$.
thus ?thesis by (simp add: stream-eq-Stream-iff)
qed
ultimately show $y \in\{y \in$ space $M . \exists x \in p s e u d o-p r o j-T r u e ~ n-'\{w\} . y=c$ $\# \# x\}$ using $\langle y \in$ space $M$ by auto
qed
qed
hence emeasure $M\{y \in$ space $M . \exists x \in($ pseudo-proj-True $n-‘\{w\}) . y=c \# \#$ $x\}=$
emeasure $M$ (pseudo-proj-True (Suc $n$ ) - $\{c \# \# w\} \cap$ space $M$ ) by simp
also have $\ldots=$ emeasure $M\{y \in$ space M. stake (Suc n) $y=\operatorname{stake}$ (Suc n)
( $c \# \# w)\}$

also have $\ldots=\left(\prod i \in\{0 . . n\}\right.$. prob-component $\left.p(c \# \# w) i\right)$
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 $\ldots=$ prob-component $p(c \# \# w) 0 *\left(\prod i \in\{1 . . n\}\right.$. prob-component $p$ $(c \# \# w) i)$
by (simp add: decompose-init-prod)
also have $\ldots=$ prob-component $p(c \# \# w) 0 *\left(\prod i \in\{1 . .<\right.$ Suc $n\}$. prob-component $p(c \# \# w) i)$
proof -
have $\left(\prod i \in\{1 . . n\}\right.$. prob-component $\left.p(c \# \# w) i\right)=\left(\prod i \in\{1 . .<\right.$ Suc $n\}$. prob-component $p(c \# \# w) i)$
proof (rule prod.cong)
show $\{1 . . n\}=\{1 . .<$ Suc $n\}$ by auto
show $\bigwedge x . x \in\{1 . .<$ Suc $n\} \Longrightarrow$ prob-component $p(c \# \# w) x=$ prob-component $p(c \# \# w) x$ by $\operatorname{simp}$
qed
thus ?thesis by simp
qed
also have $\ldots=$ prob-component $p(c \# \# w) 0 *\left(\prod i \in\{0 . .<n\}\right.$. prob-component $p w i)$
proof -
have $\left(\prod i \in\{1 . .<\right.$ Suc $n\}$. prob-component $\left.p(c \# \# w) i\right)=\left(\prod i \in\{0 . .<n\}\right.$. prob-component pwi)
proof (rule prod.reindex-cong)
show inj-on ( $\lambda n$. Suc $n$ ) $\{0 . .<n\}$ by simp
show $\{1 . .<$ Suc $n\}=$ Suc' $\{0 . .<n\}$ by auto
show $\bigwedge x . x \in\{0 . .<n\} \Longrightarrow$ 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 $\ldots=$ prob-component $p(c \# \# w) 0 *$ emeasure $M\{y \in$ space $M$. stake $n y=$ stake $n w\}$
using bernoulli-stream-pref-prob'[symmetric, of $M$ p wn] ennreal-mult' p-gt-0 $p$-lt-1 bernoulli
prob-component-def by auto
also have $\ldots=$ prob-component $p(c \# \# w) 0 *$ emeasure $M$ (pseudo-proj-True $n$

- ' $\{w\} \cap$ 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 $\ldots=$ emeasure $M$ (pseudo-proj-True $n-'\{p s e u d o-p r o j-T r u e ~ n ~ w\}) ~$
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 $A E w$ 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 $\langle A E w$ in $M . f w=0\rangle$ obtain $N^{\prime}$ where Nprops: $\{w \in$ space $M$. $\neg f w=$ $0\} \subseteq N^{\prime} N^{\prime} \in$ sets $M$ emeasure $M N^{\prime}=0$
by (force elim: $A E-E$ )
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.fw>0} by auto
    ultimately have {w\in space M. \negf w=0}\in sets (nat-filtration n) by auto
    hence emeasure M {w\in space M. \negf w=0}=0 using Nprops by (metis
(no-types, lifting) emeasure-eq-0)
    hence {w\in space M. \negf w=0}={} using <{w\in space M. \negf w=0}\in sets
(nat-filtration n)>
    nat-filtration-null-set[of {w\in space M.fw\not=0} n] assms by simp
    hence {w.fw\not=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 }=>\mathrm{ real
    assumes AE w in M.fw=gw
and 0<p
and p<1
and f\in borel-measurable (nat-filtration n)
and g\in borel-measurable (nat-filtration n)
    shows fw=gw
proof -
    define diff where diff =(\lambdaw.fw-gw)
    have AE w in M. diff w=0
    proof (rule AE-mp)
        show AE w in M.fw=gw using assms by simp
        show AE w in M.fw=gw\longrightarrowdiff w=0
            by (rule AE-I2, intro impI, (simp add: diff-def))
    qed
    have }\forallw\mathrm{ . diff }w=
    proof (rule nat-filtration-AE-zero)
        show AE w in M. diff w=0 using <AE w in M. diff w=0〉.
        show diff \in borel-measurable (nat-filtration n) using assms unfolding diff-def
by simp
        show 0<p and p<1 using assms by auto
    qed
    thus fw=gw 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 $\forall t A . A \in$ sets (nat-filtration $t) \longrightarrow($ emeasure $M A=0)=($ emeasure $N$
$A=0$ )
proof (intro allI impI)
fix $n$
fix $A$
assume $A \in$ sets (nat-filtration $n$ )
show (emeasure $M A=0)=($ emeasure $N A=0)$
proof
\{
assume emeasure $M A=0$
hence $A=\{ \}$ using $\langle A \in$ 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 $\langle A \in$ sets (nat-filtration $n$ )〉 infinite-coin-toss-space.nat-filtration-null-set[of
$q$ NAn]
assms
using $\langle$ events $=$ sets $N\rangle$ 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 $0 \leq q$ by simp
have $q<1$ using assms by simp
thus $q \leq 1$ by $\operatorname{simp}$
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 $\langle q<1\rangle\langle 0<q\rangle$
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 }\forallm\mathrm{ . 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 }\langle0\leqq\rangle\langleN=\mathrm{ bernoulli-stream q><q\ 1> infinite-coin-toss-space.intro
        infinite-coin-toss-space.nat-filtration-def by blast
    thus nat-filtration m}=\mathrm{ 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\rangle\langlespace M = space N\rangle
            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) un-
folding pseudo-proj-True-def by simp
    also have \(\ldots=\operatorname{shift}(w!!0 \#(\) stake \(n(\) stl \(w)))(\) sconst True \()\) by simp
    also have \(\ldots=w!!0 \# \#\) shift (stake \(n(\) stl \(w)\) ) (sconst True) by simp
    also have \(\ldots=w!!0 \# \#\) pseudo-proj-True \(n(\) stl \(w\) ) unfolding pseudo-proj-True-def
by \(\operatorname{simp}\)
    finally show ?thesis.
qed
lemma (in infinite-coin-toss-space) pseudo-proj-True-img:
    assumes pseudo-proj-True \(n w=w\)
    shows \(w \in\) 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
proof -
    obtain \(n n::(\) bool \(\Rightarrow\) bool \() \Rightarrow\) bool stream \(\Rightarrow\) bool stream \(\Rightarrow\) nat where
    \(\bigwedge p s n\) sa sb na pa sc pb sd se. \((\neg p(s!!n:: b o o l) \vee \operatorname{smap} p s \neq s a \vee\) sa !! n)
\(\wedge(\neg s b!!n a \vee\) smap pa sc \(\neq s b \vee p a(s c!!n a:: b o o l)) \wedge(\neg p b(s d!!n n ~ p b\) sd se)
\(\vee \neg\) se !! nn pb sd se \(\vee\) smap pb sd \(=s e\) )
            using smap-alt by moura
    then show?thesis
        by (metis (no-types) assms eq-id-iff id-funpow snth-siterate)
```


## qed

lemma (in infinite-coin-toss-space) pseudo-proj-True-suc-img-pref:
shows range (pseudo-proj-True $($ Suc $n))=\{y . \exists w \in$ range (pseudo-proj-True n). $y=$ True $\# \# w\} \cup$
$\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ False $\# \# w\}$
proof
show range (pseudo-proj-True (Suc n))
$\subseteq\{y . \exists w \in$ range (pseudo-proj-True n). $y=$ True $\# \# w\} \cup\{y . \exists w \in$ range (pseudo-proj-True n). $y=$ False $\# \# w\}$
proof
fix $x$
assume $x \in$ range (pseudo-proj-True (Suc n))
hence $x=$ pseudo-proj-True (Suc n) x using pseudo-proj-True-proj by auto
define $x p$ where $x p=s t l x$
have $x p=\operatorname{stl}$ (shift (stake (Suc n) $x$ ) (sconst True)) using $\langle x=$ pseudo-proj-True (Suc n) $x>$
unfolding xp-def pseudo-proj-True-def by simp
also have $\ldots=$ shift $(($ stake $n($ stl $x)))$ (sconst True) by simp
finally have $x p=\operatorname{shift}(($ stake $n($ stl $x)))$ (sconst True) .
hence $x p \in$ range ( $p$ seudo-proj-True $n$ ) using pseudo-proj-True-def by auto
show $x \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ) . $y=$ True $\# \# w\} \cup\{y . \exists w \in$
range (pseudo-proj-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-proj-True n). $y=$ True $\# \# w\}$ using $\langle x p \in$ range (pseudo-proj-True $n$ ) 〉 by auto
thus ?thesis by auto
next
case False
have $x=$ False \#\# xp unfolding $x p$-def using False by (simp add: stream-eq-Stream-iff)
hence $x \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ False $\# \# w\}$ using $\langle x p \in$ range ( $p$ seudo-proj-True $n$ ) $\rangle$ by auto
thus ?thesis by auto
qed
qed
have $\{y . \exists w \in$ range ( $p$ seudo-proj-True $n$ ) $\cdot y=\operatorname{True} \# \# w\} \subseteq$ range (pseudo-proj-True
(Suc n))
proof
fix $y$
assume $y \in\{y . \exists w \in$ range (pseudo-proj-True n) . $y=$ True $\# \# w\}$
hence $\exists w . w \in$ range (pseudo-proj-True $n) \wedge y=$ True $\# \# w$ by auto
from this obtain $w$ where $w \in$ range (pseudo-proj-True $n$ ) and $y=$ True $\# \#$ $w$ by auto
have $w=$ pseudo-proj-True $n w$ using pseudo-proj-True-proj $\prec w \in$ range ( $p$ seudo-proj-True $n$ ) > by auto

```
    hence }y=\mathrm{ 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-proj-True (Suc n) (True ## w) unfolding pseudo-proj-True-def
by simp
            finally have y=pseudo-proj-True (Suc n) (True##w).
            thus }y\in\mathrm{ range (pseudo-proj-True (Suc n)) by simp
    qed
    moreover have {y.\existsw\inrange (pseudo-proj-True n) . y= False ## w}\subseteq
range (pseudo-proj-True (Suc n))
    proof
            fix y
            assume }y\in{y.\existsw\in\mathrm{ range (pseudo-proj-True n). y= False ## w}
            hence }\existsw.w\in\mathrm{ range (pseudo-proj-True n)}\wedgey=False ## w by aut
            from this obtain w where w\in range (pseudo-proj-True n) and y=False ##
w by auto
    have w= pseudo-proj-True n w using pseudo-proj-True-proj }<w\in range (pseudo-proj-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-proj-True (Suc n) (False ## w) unfolding pseudo-proj-True-def
by simp
            finally have }y=\mathrm{ pseudo-proj-True (Suc n) (False##w).
            thus }y\in\mathrm{ range (pseudo-proj-True (Suc n)) by simp
    qed
    ultimately show {y.\existsw\in range (pseudo-proj-True n) . y = True ##w}
    {y.\existsw\in range (pseudo-proj-True n). y = False ##w}\subseteq range (pseudo-proj-True
(Suc n)) by simp
qed
lemma (in infinite-coin-toss-space) reindex-pseudo-proj:
    shows (\sumw\inrange (pseudo-proj-True n).f(c## w))=
            (\sumy\in{y.\existsw\in range (pseudo-proj-True n). y=c##w}.f y)
proof (rule sum.reindex-cong[symmetric],auto)
    define ccons where ccons = ( }\lambdaw.c##w
    show inj-on ccons (range (pseudo-proj-True n))
    proof
        fix }x
            assume x\in range (pseudo-proj-True n) and y\in range (pseudo-proj-True n)
and ccons }x=\mathrm{ ccons }
    hence c### = c##y unfolding ccons-def by simp
    thus }x=y\mathrm{ by simp
    qed
qed
```

lemma (in infinite-coin-toss-space) pseudo-proj-True-imp-False:
assumes pseudo-proj-True $n w=$ pseudo-proj-True $n x$
shows pseudo-proj-False $n w=$ pseudo-proj-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-proj-True n w = pseudo-proj-True n x
    shows pseudo-proj-True (Suc n) w\in{pseudo-proj-True n x, pseudo-proj-False
nx}
proof -
    have pseudo-proj-False n w = pseudo-proj-False n x using assms pseudo-proj-True-imp-False[of
nwx] 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 $n$ ) ) $\cap$ (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 $n)) \cap$ pseudo-proj-True $n-‘\{p s e u d o-p r o j-T r u e ~$
$n x\}$
$\subseteq\{$ pseudo-proj-True $n x$, pseudo-proj-False $n x\}$
proof
fix $w$
assume $w \in$ range (pseudo-proj-True $($ Suc $n)) \cap$ pseudo-proj-True $n-‘\{$ pseudo-proj-True
$n x\}$
hence $w \in$ range (pseudo-proj-True (Suc n)) and $w \in$ pseudo-proj-True $n-‘$
$\{p s e u d o-p r o j-T r u e ~ n x\}$ by auto
hence pseudo-proj-True $n w=$ pseudo-proj-True $n x$ by simp
have $w=$ pseudo-proj-True (Suc n) w using $\prec w \in$ range (pseudo-proj-True (Suc
$n)$ ) >
using pseudo-proj-True-proj by auto
also have $\ldots \in\{$ pseudo-proj-True $n x$, pseudo-proj-False $n x\}$ using $«$ pseudo-proj-True
$n w=$ pseudo-proj-True $n$ x $\rangle$
pseudo-proj-Suc-prefix by simp
finally show $w \in\{$ pseudo-proj-True $n x$, pseudo-proj-False $n x\}$.
qed
show \{pseudo-proj-True $n x$, pseudo-proj-False $n x\}$
$\subseteq$ range (pseudo-proj-True (Suc $n$ )) $\cap$ pseudo-proj-True $n-‘\{p s e u d o-p r o j-T r u e$
$n x\}$
proof -
have pseudo-proj-True $n x \in$ range (pseudo-proj-True (Suc $n$ )) $\cap$ pseudo-proj-True
$n-‘\{p s e u d o-p r o j-T r u e ~ n x\}$
by (simp add: pseudo-proj-True-Suc-proj pseudo-proj-True-img pseudo-proj-True-proj)
moreover have pseudo-proj-False $n x \in$ range (pseudo-proj-True (Suc n) ) $\cap$
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 $n)^{\prime}(f-‘\{f x\}) \cup($ 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)) $\cap f-^{\prime}\{f x\} \subseteq(\bigcup w \in\{y . f y=f x\}$.
\{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 $\operatorname{simp}$
have $w=$ pseudo-proj-True (Suc n) w using $\langle w \in$ range (pseudo-proj-True (Suc n)) >
using pseudo-proj-True-proj by auto
also have $\ldots \in\{$ pseudo-proj-True $n w$, pseudo-proj-False $n w\}$
using pseudo-proj-Suc-prefix by auto
also have $\ldots \subseteq(\bigcup w \in\{y . f y=f x\}$. \{pseudo-proj-True $n w$, pseudo-proj-False $n w\})$ using $\langle w \in\{y . f y=f x\}\rangle$
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 fw}=fy\mathrm{ 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 fw =fy using assms nat-filtration-not-borel-info natural-filtration
        by (metis Int-iff }\langlew\in{\mathrm{ 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.fy=fx}.{pseudo-proj-True n w})\cup(\bigcupw\in{y.fy=fx}.{pseudo-proj-False
n w}) by auto
    also have ... =(pseudo-proj-True n)' {y.fy=fx}\cup(pseudo-proj-False n)
'{y.fy=fx} 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
$\left.n^{\prime}(g-‘\{g z\})\right)$
proof
show pseudo-proj-True $n-' g-'\{g z\} \subseteq$ pseudo-proj-True $n-'$ pseudo-proj-True
$n^{\prime} g-‘\{g z\}$
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 $\ldots \in$ pseudo-proj-True $n ‘(g-‘\{g z\})$ using $\langle w \in$ pseudo-proj-True
$n-‘ g-‘\{g z\}>$
by $\operatorname{simp}$
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-`{gz}
    proof
    fix w
    assume w f pseudo-proj-True n -' pseudo-proj-True n'g -'{gz}
    hence \existsy.pseudo-proj-True n w = pseudo-proj-True n y^gy=gz by auto
    from this obtain y where pseudo-proj-True n w = pseudo-proj-True n y and
gy=gz 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 ... = gz using <g y=gz\rangle.
            finally have g(pseudo-proj-True n w)=gz.
            thus w\in pseudo-proj-True n -' g-'`ggz} by simp
    qed
qed
lemma (in infinite-cts-filtration) borel-pseudo-proj-preimage:
    fixes g::bool stream = 'b::{t0-space }
    assumes g\in borel-measurable (F n)
    shows pseudo-proj-True n-` (g -'{gz}) = pseudo-proj-True n -'(pseudo-proj-True
n'(g-'{gz}))
    using pseudo-proj-preimage[of g n borel z] set-discriminating-if[of g n] natu-
ral-filtration assms by simp
lemma (in infinite-cts-filtration) pseudo-proj-False-preimage:
    assumes g\in measurable (F n) N
    and set-discriminating ng N
    shows pseudo-proj-False n-' (g-'{g z}) = pseudo-proj-False n-'(pseudo-proj-False
n'(g-'{gz}))
proof
    show pseudo-proj-False n -' g-'{g z}\subseteq pseudo-proj-False n -'pseudo-proj-False
n'g-`{gz}
    proof
        fix w
        assume w\in pseudo-proj-False n -' g -` {gz}
        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 ... \in pseudo-proj-False n'(g-'{g}z})\mathrm{ using <w pseudo-proj-False
n-'g -'{ggz}>
            by simp
            finally have pseudo-proj-False n w \in pseudo-proj-False n'(g -'{gz}).
            thus w\in pseudo-proj-False n -'(pseudo-proj-False n' (g-'{gz})) by simp
    qed
    show pseudo-proj-False n -' pseudo-proj-False n' g-'{g z}\subseteq pseudo-proj-False
n-'g-'{gz}
    proof
```

fix $w$
assume $w \in$ pseudo-proj-False $n-$ 'pseudo-proj-False $n ' g-‘\{g z\}$
hence $\exists y$. pseudo-proj-False $n w=$ pseudo-proj-False $n y \wedge g y=g z$ 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 $« p s e u d o-p r o j-F a l s e$ $n w=$ pseudo-proj-False $n y$ y
by simp
also have $\ldots=g y$ using assms nat-filtration-not-borel-info' natural-filtration by (metis comp-apply)
also have $\ldots=g z$ using $\langle g y=g z\rangle$.
finally have $g$ (pseudo-proj-False $n w)=g z$.
thus $w \in$ 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 $\Rightarrow ' b::\{t 0$-space $\}$
assumes $g \in$ borel-measurable ( $F n$ )
shows pseudo-proj-False $n-‘(g-'\{g z\})=$ pseudo-proj-False $n-'($ pseudo-proj-False $\left.n^{\prime}(g-‘\{g z\})\right)$
using pseudo-proj-False-preimage[of $g$ n borel $z]$ set-discriminating-if[of $g n]$ nat-ural-filtration assms by simp
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 $n-'(g-'\{g z\})=g-‘\{g z\}$
proof
show pseudo-proj-True $n-‘ g-‘\{g z\} \subseteq g-'\{g z\}$
proof
fix $w$
assume $w \in$ 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 $\ldots=g z$ using $\langle w \in$ pseudo-proj-True $n-‘ g-‘\{g z\}\rangle$ by simp
finally have $g w=g z$.
thus $w \in g-‘\{g z\}$ by $\operatorname{simp}$
qed
show $g-‘\{g z\} \subseteq$ pseudo-proj-True $n-‘ g-‘\{g z\}$
proof
fix $w$
assume $w \in g-{ }^{\prime}\{g z\}$
have $g$ (pseudo-proj-True $n w)=g w \mathbf{u s i n g}$ assms nat-filtration-not-borel-info natural-filtration
by (metis comp-apply)
also have $\ldots=g z$ using $\langle w \in g-‘\{g z\}\rangle$ by $\operatorname{simp}$

```
    finally have \(g\) (pseudo-proj-True \(n w)=g z\).
    thus \(w \in\) pseudo-proj-True \(n-' g-'\{g z\}\) by simp
    qed
qed
lemma (in infinite-cts-filtration) borel-pseudo-proj-preimage':
    fixes \(g::\) bool stream \(\Rightarrow ' b::\{t 0\)-space \(\}\)
    assumes \(g \in\) borel-measurable ( \(F n\) )
    shows pseudo-proj-True \(n-{ }^{\prime}(g-‘\{g z\})=g-‘\{g z\}\)
    using assms natural-filtration by (simp add: set-discriminating-if pseudo-proj-preimage')
lemma (in infinite-cts-filtration) pseudo-proj-False-preimage':
    assumes \(g \in\) 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 \(n-' g-'\{g z\} \subseteq g-'\{g z\}\)
    proof
    fix \(w\)
    assume \(w \in\) 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 \(\ldots=g z\) using \(\langle w \in\) pseudo-proj-False \(n-‘ g-‘\{g z\}\rangle\) by simp
    finally have \(g w=g z\).
    thus \(w \in g-‘\{g z\}\) by simp
    qed
    show \(g-‘\{g z\} \subseteq\) pseudo-proj-False \(n-‘ g-‘\{g z\}\)
    proof
    fix \(w\)
    assume \(w \in g-‘\{g z\}\)
    have \(g\) (pseudo-proj-False \(n w)=g\) using assms nat-filtration-not-borel-info'
natural-filtration
            by (metis comp-apply)
    also have \(\ldots=g z\) using \(\langle w \in g-‘\{g z\}\rangle\) by simp
    finally have \(g\) (pseudo-proj-False \(n w)=g z\).
    thus \(w \in\) pseudo-proj-False \(n-{ }^{\prime} g-{ }^{\prime}\{g z\}\) by simp
    qed
qed
```

lemma (in infinite-cts-filtration) borel-pseudo-proj-False-preimage':
fixes $g::$ bool stream $\Rightarrow{ }^{\prime} b::\{t 0$-space $\}$
assumes $g \in$ 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 $\Rightarrow$ real
assumes $f \in$ borel-measurable ( $F$ 0)
and $f$ (sconst True) $=c$
shows has-bochner-integral Mfc
proof -
have space $M=$ space ( $F 0$ ) using filtration by (simp add: filtration-def subal-gebra-def)
have $f \in$ borel-measurable $M$
using assms(1) nat-filtration-borel-measurable-integrable natural-filtration by blast
have $\exists d . \forall x \in$ space (F O). $f x=d$
proof (rule triv-measurable-cst)
show space ( $\left.\begin{array}{ll}F & 0\end{array}\right)=$ space $M$ using «space $M=\operatorname{space}\left(\begin{array}{ll}F & 0\end{array}\right)$ 〉..
show sets $\left(\begin{array}{ll}F & 0)\end{array}\right)=\{\{ \}$, space $M\}$ using info-disc-filtr
by (simp add: init-triv-filt-def bot-nat-def)
show $f \in$ borel-measurable ( $\left.\begin{array}{l}F \\ 0\end{array}\right)$ using assms by simp
show space $M \neq\{ \}$ by (simp add:not-empty)
qed
from this obtain $d$ where $\forall x \in$ space ( $F 0$ ). $f x=d$ by auto
hence $\forall x \in$ space $M$. $f x=d$ using «space $M=\operatorname{space}\left(\begin{array}{ll}F & 0) \text { ) by simp }\end{array}\right.$
hence $f$ (sconst True) $=d$ using bernoulli-stream-space bernoulli by simp
hence $c=d$ using assms by simp
hence $\forall x \in$ space $M . f x=c$ using $\langle\forall x \in$ space $M . f x=d\rangle\langle c=d\rangle$ by simp have $f \in$ borel-measurable $M$
using assms(1) nat-filtration-borel-measurable-integrable natural-filtration by blast
have integral ${ }^{N} M f=$ integral $^{N} M(\lambda w . c)$
proof (rule nn-integral-cong)
fix $x$
assume $x \in$ space $M$
thus ennreal $(f x)=$ ennreal $c$ using $\langle\forall x \in$ space $M . f x=d\rangle\langle c=d\rangle$ by auto
qed
also have $\ldots=$ integral $^{N} M(\lambda w . c *($ indicator $($ space $M)) w)$
by (simp add: nn-integral-cong)
also have $\ldots=$ ennreal $c *$ emeasure $M$ (space $M$ ) using $n n$-integral-cmult-indicator $[$ of
space $M M c]$
by (simp add: nn-integral-cong)
also have $\ldots=$ ennreal $c$ by (simp add: emeasure-space-1)
finally have integral ${ }^{N} M f=$ ennreal $c$.
hence integral ${ }^{N} M(\lambda x .-f x)=$ ennreal $(-c)$
by (simp add: $\langle\forall x \in$ space M. $f x=d\rangle\langle c=d\rangle$ emeasure-space-1 nn-integral-cong)
show has-bochner-integral Mfc
proof (cases $0 \leq c$ )
case True
hence $A E x$ in $M .0 \leq f x$ using $\langle\forall x \in$ space $M . f x=c\rangle$ by simp
thus ?thesis using 〈random-variable borel $f\rangle$ True
$\left\langle\right.$ integral $^{N} M f=$ ennreal $\left.c\right\rangle$ by (simp add: has-bochner-integral-nn-integral)
next
case False
let $? m f=\lambda w .-f w$
have $A E x$ in $M .0 \leq$ ? mf $x$ using $\langle\forall x \in$ space $M . f x=c\rangle$ False by simp
hence has-bochner-integral M?mf ( $-c$ ) using 〈random-variable borel $f\rangle$ False $\left\langle\right.$ integral $^{N} M(\lambda x .-f x)=$ ennreal $\left.(-c)\right\rangle$ 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 $\Rightarrow$ real
assumes $f \in$ borel-measurable ( $\left.\begin{array}{ll}F & 0\end{array}\right)$
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 $\Rightarrow{ }^{\prime} b::\{t 0$-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 $p)$ ) $\rightarrow_{M}$ mea-sure-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 $\langle f \in$ 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 $\operatorname{simp}$
thus random-variable borel $(\lambda w . f(c \# \# w))$ using bernoulli unfolding bernoulli-stream-def by simp
have $\forall w . f(c \# \#($ pseudo-proj-True $n w))=f(c \# \# w)$
proof
fix $w$
have $c \# \#$（pseudo－proj－True $n w)=$ pseudo－proj－True（Suc n）$(c \# \# w)$ unfolding pseudo－proj－True－def by simp
hence $f(c \# \#($ pseudo－proj－True $n w))=f$（pseudo－proj－True（Suc $n$ ） $(c \# \# w))$ by $\operatorname{simp}$
also have $\ldots=f(c \# \# w)$ using assms nat－filtration－info［of $f$ Suc n］natu－ ral－filtration
by（metis comp－apply）
finally show $f(c \# \#(p s e u d o-p r o j-T r u e ~ n w))=f(c \# \# w)$ ． qed
thus $(\lambda w \cdot f(c \# \# w)) \circ$ pseudo－proj－True $n=(\lambda w \cdot f(c \# \# w))$ by auto qed
thus $(\lambda w . f(c \# \# w)) \in$ borel－measurable $(F n)$ using natural－filtration by simp qed
lemma（in infinite－cts－filtration）$F$－n－nn－integral－pos：
fixes $f:$ ：bool stream $\Rightarrow$ real
shows $\bigwedge f .(\forall x .0 \leq f x) \Longrightarrow f \in$ borel－measurable $(F n) \Longrightarrow$ integral $^{N} M f=$ （ $\sum w \in$ range（pseudo－proj－True $n$ ）．（emeasure $M$（ $($ pseudo－proj－True $n)-\{\{w\}$
$\cap$ space $M)$ ）＊ennreal $(f w))$
proof（induct $n$ ）
case 0
have range（pseudo－proj－True 0）$=\{$ sconst True $\}$
proof
have $\wedge w$ ．pseudo－proj－True $0 w=$ sconst True
proof－
fix $w$
show pseudo－proj－True $0 w=$ sconst True unfolding pseudo－proj－True－def by $\operatorname{simp}$
qed
thus range（pseudo－proj－True 0 ）$\subseteq\{$ sconst True $\}$ by auto
show $\{$ sconst True $\} \subseteq$ range（pseudo－proj－True 0）
using 〈range（pseudo－proj－True 0）$\subseteq\{$ sconst True\} $\rangle$ subset－singletonD by fastforce
qed
hence $($ emeasure $M(($ pseudo－proj－True 0$)-‘\{$ sconst True $\} \cap$ space $M))=e n$－ nreal 1
by（metis Int－absorb1 UNIV－I emeasure－eq－measure image－eqI prob－space subsetI vimage－eq）
have $\left(\sum w \in\right.$ range（pseudo－proj－True 0）．$\left.f w\right)=\left(\sum w \in\{\right.$ sconst True $\left.\} . f w\right)$ using 〈range（pseudo－proj－True 0）$=\{$ sconst True $\} 〉$
sum．cong［of range（pseudo－proj－True $n$ ）\｛sconst True\} ff] by simp
also have $\ldots=f$（sconst True）by simp
finally have $\left(\sum w \in\right.$ range（pseudo－proj－True 0$\left.) . f w\right)=f$（sconst True）．
hence $\left(\sum w \in\right.$ range（pseudo－proj－True 0）．（emeasure $M$（（pseudo－proj－True 0） $-‘\{w\} \cap$ space $M)) * f w)=f$（sconst True）
using «(emeasure $M(($ pseudo-proj-True 0$)-‘\{$ sconst True $\} \cap$ space $M))=$ ennreal 1)
by (simp add: 〈range (pseudo-proj-True 0) $=\{$ sconst True $\}\rangle)$
thus integral ${ }^{N} M f=\left(\sum w \in\right.$ range (pseudo-proj-True 0). (emeasure $M$ ( $($ pseudo-proj-True $0)-\{\{w\} \cap$ space $M)) * f w)$
using 0 by (simp add:cst-nn-integral)

## next

case (Suc n)
define $B P$ where $B P=$ measure-pmf (bernoulli-pmf p)
have integral ${ }^{N} M f=$ integral $^{N}$ (stream-space BP) $f$ using bernoulli
unfolding bernoulli-stream-def BP-def by simp
also have $\ldots=\int{ }^{+} x . \int{ }^{+} X . f(x \# \# X)$ dstream-space BP $\partial B P$
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-measurable (stream-space BP) by simp
qed
also have $\ldots=\left(\lambda x .\left(\int^{+} X \cdot f(x \# \# X)\right.\right.$ dstream-space BP) $)$ True $*$ ennreal $p$

$$
+
$$

$\left(\lambda x .\left(\int{ }^{+} X . f(x \# \# X)\right.\right.$ Dstream-space BP)) False $*$ ennreal ( $\left.1-p\right)$
using $p$-gt-0 p-lt-1 unfolding $B P$-def by simp
also have $\ldots=\left(\int^{+} X . f(\right.$ True $\# \# X)$ дstream-space BP) $* p+$
( $\sum w \in$ range (pseudo-proj-True n). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$
space $M) *(f($ False $\# \# w))) *(1-p)$
proof -
define $f f$ where $f f=(\lambda w . f($ False \#\# w) $)$
have $\bigwedge x .0 \leq f f x$ 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 $\left(\int^{+}\right.$x. ennreal $\left.(f f x) \partial M\right)=$
( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$
space $M)$ * ennreal (ff $w)$ )
using Suc by simp
thus ?thesis unfolding $f f$-def by (simp add: BP-def bernoulli bernoulli-stream-def)
qed
also have $\ldots=\left(\sum w \in\right.$ range ( $p$ seudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) *(f($ True \#\# w) $)) * p+$
( $\sum w \in$ range ( $p$ seudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) *(f($ False $\# \# w))) *(1-p)$
proof -
define $f t$ where $f t=(\lambda w . f($ True $\# \# w))$
have $\bigwedge x .0 \leq f t x$ using Suc unfolding ft-def by simp
moreover have ft borel-measurable ( $F n$ ) using Suc unfolding ft-def by (simp add:suc-measurable)
ultimately have $\left(\int^{+}\right.$x. ennreal $\left.(f t x) \partial M\right)=$
( $\sum w \in$ range (pseudo-proj-True n). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M)$ * ennreal (ft $w)$ )
using Suc by simp
thus ?thesis unfolding $f t$-def by (simp add: BP-def bernoulli bernoulli-stream-def) qed
also have $\ldots=\left(\sum w \in\right.$ range ( $p$ seudo-proj-True n). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) * p *(f($ True \#\# w $)))+$
( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) *(f($ False \#\#w) $)) *(1-p)$
proof -
have ( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n$-‘ $\{w\} \cap$ space $M) *(f($ True \#\# w) $)) * p=$
( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-‘\{w\}$
$\cap$ space $M) *(f($ True $\# \# w)) * p)$
by (rule sum-distrib-right)
also have $\ldots=\left(\sum w \in\right.$ range (pseudo-proj-True n). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) * p *(f($ True \#\#w) $))$
proof (rule sum.cong, simp)
fix $w$
assume $w \in$ range (pseudo-proj-True $n$ )
show emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) *$ ennreal $(f$ (True $\# \# w)) *$ ennreal $p=$
emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ 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 $\bigwedge x . x * \operatorname{ennreal}(f($ 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 ( $\sum w \in$ range ( $p$ seudo-proj-True $n$ ). emeasure $M$ ( $p s e u d o-p r o j-T r u e$ $n-‘\{w\} \cap$ space $M) *(f($ True \#\# w) $)) * p=$ ( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-‘\{w\}$ $\cap$ space $M) * p *(f(\operatorname{True} \# \# w)))$.
thus ?thesis by simp
qed
also have $\ldots=\left(\sum w \in\right.$ range (pseudo-proj-True n). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) * p *(f($ True \#\# w $)))+$
( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-{ }^{`}\{w\} \cap$ space $M) *(1-p) *(f($ False \#\# w $)))$
proof -
have ( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n$-‘ $\{w\} \cap$ space $M) *(f($ False \#\# w) $)) *(1-p)=$
( $\sum w \in$ range ( $p$ seudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-‘\{w\}$
$\cap$ space $M) *(f($ False \#\# w $)) *(1-p))$
by (rule sum-distrib-right)
also have $\ldots=\left(\sum w \in\right.$ range (pseudo-proj-True $n$ ). emeasure $M$ ( $p$ seudo-proj-True $n-‘\{w\} \cap$ space $M) *(1-p) *(f($ False \#\# w) $))$
proof (rule sum.cong, simp)
fix $w$
assume $w \in$ range (pseudo-proj-True $n$ )
show emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) *$ ennreal $(f$ (False $\# \# w)) * \operatorname{ennreal}(1-p)=$ emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ 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 $\bigwedge x . x * \operatorname{ennreal}(f($ False $\# \# w)) *$ ennreal $(1-p)=x *$ ennreal $(1-p) * \operatorname{ennreal}(f($ False \#\# w) )
by (simp add: semiring-normalization-rules(16))
thus? thesis by simp

## qed

qed
finally have ( $\sum w \in$ range ( $p s e u d o-p r o j-T r u e ~ n$ ). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) *(f($ False $\# \# w))) *(1-p)=$
( $\sum w \in$ range ( $p$ seudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-'\{w\}$ $\cap$ space $M) *(1-p) *(f($ False \#\# w $)))$.
thus ?thesis by simp
qed
also have $\ldots=\left(\sum y \in\{y . \exists w \in\right.$ range (pseudo-proj-True $\left.n) . y=\operatorname{True} \# \# w\right\}$. emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) * p *(f(y)))+$
( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) *(1-p) *(f($ False \#\# w) $))$

## proof -

have ( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-$ $\{w\} \cap$ space $M) * p *(f($ True $\# \# w)))=$
( $\sum w \in$ range (pseudo-proj-True n). emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $($ True\#\#w) $\} \cap$ space $M) * p *(f($ True \#\# w) ) ) by simp
also have ... $=$
( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=\operatorname{True} \# \# w\}$. emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) * p *(f(y)))$
by (rule reindex-pseudo-proj)
finally have ( $\sum w \in$ range ( $\left.p s e u d o-p r o j-T r u e ~ n\right)$. emeasure $M$ ( $p s e u d o-p r o j-T r u e$ $n-‘\{w\} \cap$ space $M) * p *(f($ True \#\#w $)))=$
( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ True $\# \# w\}$. emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) * p *(f(y)))$.
thus ?thesis by simp
qed
also have $\ldots=\left(\sum y \in\{y . \exists w \in\right.$ range (pseudo-proj-True $n) . y=$ True $\left.\# \# w\right\}$. emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) * p *(f(y)))+$
( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ False $\# \# w\}$. emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) *(1-p) *(f(y)))$
proof -
have ( $\sum$ werange (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n$-‘
$\{w\} \cap$ space $M) *(1-p) *(f($ False $\# \# w)))=$
( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $($ False $\# \# w)\} \cap$ space $M) *(1-p) *(f($ False \#\# w) $)$ ) by simp
also have ... $=$
( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ False $\# \# w\}$. emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) *(1-p) *(f(y)))$
by (rule reindex-pseudo-proj)
finally have ( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M$ ( $p s e u d o-p r o j-T r u e$ $n-‘\{w\} \cap$ space $M) *(1-p) *(f($ False \#\# w) $))=$
( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ False $\# \# w\}$. emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) *(1-p) *(f(y)))$.
thus ?thesis by simp
qed
also have $\ldots=\left(\sum y \in\{y . \exists w \in\right.$ range (pseudo-proj-True $\left.n) . y=\operatorname{True} \# \# w\right\}$. (prob-component p y 0$) *$ emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M)$ * $(f(y)))+$
$\left(\sum y \in\{y . \exists w \in\right.$ range (pseudo-proj-True $n) . y=$ False $\left.\# \# w\right\}$. emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) *(1-p) *(f(y)))$
proof -
have $\forall y \in\{y . \exists w \in$ range (pseudo-proj-True n). $y=$ True $\# \# w\}$. emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) * p=$
prob-component py $0 *$ emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space M)
proof
fix $y$
assume $y \in\{y . \exists w \in$ range (pseudo-proj-True n). $y=$ True $\# \# w\}$
hence $\exists w \in$ range (pseudo-proj-True n). $y=$ True \#\# w by simp
from this obtain $w$ where $w \in$ range (pseudo-proj-True $n$ ) and $y=\operatorname{True}$ $\# \# w$ by auto
have emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) * p=p *$ emeasure $M$ ( pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M)$
by (simp add:mult.commute)
also have...$=$ prob-component p y 0 * emeasure $M$ (pseudo-proj-True $n$-‘ $\{$ stl $y\} \cap$ space $M)$ using $\langle y=$ True \#\# $w\rangle$
unfolding prob-component-def by simp
finally show emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) * p=$
prob-component p y 0 * emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space M)
qed
thus ?thesis by auto
qed
also have $\ldots=\left(\sum y \in\{y . \exists w \in\right.$ range (pseudo-proj-True $\left.n) . y=\operatorname{True} \# \# w\right\}$. (prob-component py 0)* emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M)$ * $(f(y)))+$
( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ False $\# \# w\}$. (prob-component $\begin{array}{ll}\text { p } & 0) * \text { emeasure } M(\text { pseudo-proj-True } n-‘\{\text { stl } y\} \cap \text { space } M) *(f(y))) ~\end{array}$ proof -
have $\forall y \in\{y . \exists w \in$ range (pseudo-proj-True n). $y=$ False $\# \# w\}$. emeasure $M($ pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) *(1-p)=$
prob-component py 0 emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space M)
proof
fix $y$
assume $y \in\{y . \exists w \in$ range (pseudo-proj-True n). $y=$ False $\# \# w\}$
hence $\exists w \in$ range (pseudo-proj-True n). $y=$ False $\# \# w$ by simp
from this obtain $w$ where $w \in$ range (pseudo-proj-True $n$ ) and $y=$ False \#\# w by auto
have emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) *(1-p)=$ $(1-p)$ *emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M)$
by (simp add:mult.commute)
also have $\ldots=$ prob-component py $0 *$ emeasure $M$ (pseudo-proj-True $n-$ $\{$ stl $y\} \cap$ space $M)$ using $\langle y=$ False \#\# $w\rangle$
unfolding prob-component-def by simp
finally show emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) *(1-p)$ $=$
prob-component p y $0 *$ emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space M) .
qed
thus ?thesis by auto
qed
also have $\ldots=\left(\sum y \in\{y . \exists w \in\right.$ range (pseudo-proj-True $n$ ). $y=$ True $\# \# w\}$. emeasure $M\left\{z \in\right.$ space $M . \exists x \in$ pseudo-proj-True $n-{ }^{`}\{$ stl $y\} . z=$ True $\left.\# \# x\right\} *$ $(f y))+$
( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ False $\# \# w\}$. (prob-component p y 0) * emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) *(f(y)))$ proof -
have ( $\sum y \mid \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ True $\# \# w$.
ennreal (prob-component pyll) * emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) *(f y))=$
( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ True $\# \# w\}$. emeasure $M$ $\{z \in \operatorname{space}$ M. $\exists x \in$ pseudo-proj-True $n-‘\{$ stl $y\} . z=$ True $\# \# x\} *(f y))$
proof (rule sum.cong, simp)
fix $x x$
assume $x x \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ True $\# \# w\}$
hence $\exists$ werange (pseudo-proj-True $n$ ). $x x=$ True $\# \# w$ by simp
from this obtain $w w$ where $w w \in$ range (pseudo-proj-True $n$ ) and $x x=$ True\#\# ww by auto
have ennreal (prob-component $p($ True\#\#ww) 0$) *$ emeasure $M$ (pseudo-proj-True $n-‘\{w w\} \cap$ space $M)=$
emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-'\{w w\} . z=$ True $\# \#$ $x\}$ using $\langle w w \in$ range (pseudo-proj-True n)〉 by (rule pseudo-proj-element-prob-pref[symmetric])
thus ennreal (prob-component pxx 0) * emeasure M (pseudo-proj-True n-‘ $\{$ stl $x x\} \cap$ space $M) *(f x x)=$ emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{$ stl $x x\} . z=$ True $\# \# x\} *(f x x)$ using $\langle x x=$ True\#\#ww〉 by simp

## qed

thus?thesis by simp

## qed

also have $\ldots=\left(\sum y \in\{y . \exists w \in\right.$ range (pseudo-proj-True $n) . y=$ True $\left.\# \# w\right\}$. emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-'\{$ stl $y\} . z=$ True $\# \# x\} *$ (fy)) +
( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=$ False $\# \# w\}$. emeasure $M\{z$ $\in$ space $M . \exists x \in$ pseudo-proj-True $n-'\{$ stl $y\} . z=$ False $\# \# x\} *(f y))$

## proof -

have $\left(\sum y \mid \exists w \in\right.$ range (pseudo-proj-True $n$ ). $y=$ False $\# \# w$.
ennreal (prob-component py 0) * emeasure $M$ (pseudo-proj-True $n-‘\{$ stl $y\} \cap$ space $M) *(f y))=$
( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True n). $y=$ False $\# \# w\}$. emeasure $M$ $\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{$ stl $y\} . z=$ False $\# \# x\} *(f y))$
proof (rule sum.cong, simp)
fix $x x$
assume $x x \in\{y . \exists w \in$ range (pseudo-proj-True n). $y=$ False $\# \# w\}$
hence $\exists w \in$ range ( $p$ seudo-proj-True $n$ ). $x x=$ False $\# \#$ w bimp
from this obtain $w w$ where $w w \in$ range (pseudo-proj-True $n$ ) and $x x=$ False\#\# ww by auto
have ennreal (prob-component $p($ False\#\#ww) 0$)$ * emeasure $M$ (pseudo-proj-True $n-‘\{w w\} \cap$ space $M)=$
emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{w w\} . z=$ False $\# \# x\}$ using $\langle w w \in$ range (pseudo-proj-True n)〉
by (rule pseudo-proj-element-prob-pref[symmetric])
thus ennreal (prob-component pxx 0 ) * emeasure $M$ (pseudo-proj-True $n-$ $\{$ stl $x x\} \cap$ space $M) *(f x x)=$
emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-'\{$ stl $x x\} . z=$ False $\# \# x\} *(f x x)$ using $\langle x x=$ False\#\#ww〉 by simp
qed
thus ?thesis by simp
qed
also have $\ldots=\left(\sum w \in\right.$ range (pseudo-proj-True $n$ ). emeasure $M\{z \in$ space $M$. $\exists x \in$ pseudo-proj-True $n-‘\{w\} . z=\operatorname{True} \# \# x\} *(f($ True $\# \# w)))+$
( $\sum w \in$ range ( $p$ seudo-proj-True $n$ ). emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{w\} . z=$ False \#\# $x\} *(f($ False $\# \# w)))$
proof -
have $\bigwedge c .\left(\sum y \in\{y . \exists w \in\right.$ range (pseudo-proj-True $\left.n) . y=c \# \# w\right\}$. emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{$ stl $y\} . z=c \# \# x\} *(f y))=$
( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{w\} . z=c \# \# x\} *(f(c \# \# w)))$

## proof -

fix $c$
have $\left(\sum y \in\{y . \exists w \in\right.$ range (pseudo-proj-True n). $y=c \# \# w\}$. emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{$ stl $y\} . z=c \# \# x\} *(f y))=$ ( $\sum w \in$ range ( $p$ seudo-proj-True $n$ ). emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{s t l(c \# \# w)\} \cdot z=c \# \# x\} *(f(c \# \# w)))$
by (rule reindex-pseudo-proj[symmetric])
also have $\ldots=\left(\sum w \in\right.$ range ( pseudo -proj-True $n$ ). emeasure $M\{z \in$ space M. $\exists x \in$ pseudo-proj-True $n-‘\{w\} . z=c \# \# x\} *(f(c \# \# w)))$
by $\operatorname{simp}$
finally show ( $\sum y \in\{y . \exists w \in$ range (pseudo-proj-True $n$ ). $y=c \# \# w\}$. emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{$ stl $y\} . z=c \# \# x\} *(f$ $y))=$
( $\sum w \in$ range ( $p$ seudo-proj-True $n$ ). emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{w\} \cdot z=c \# \# x\} *(f(c \# \# w))) \cdot$
qed
thus ?thesis by auto
qed
also have $\ldots=\left(\sum w \in\{w . w \in\right.$ range (pseudo-proj-True $($ Suc $n)) \wedge w!!0=$ True $\}$. emeasure $M$ (pseudo-proj-True (Suc $n)-‘\{w\} \cap($ space $M)) *(f w))+$
$\left(\sum w \in\{w . w \in\right.$ range (pseudo-proj-True $($ Suc $n)) \wedge w!!0=$ False $\}$. emeasure $M$ (pseudo-proj-True (Suc $n)-‘\{w\} \cap($ space $M)) *(f w))$
proof -
have $\Lambda c .\left(\sum w \in\right.$ range (pseudo-proj-True $\left.n\right)$. emeasure $M\{z \in$ space $M$. $\exists x \in$ pseudo-proj-True $n-‘\{w\} . z=c \# \# x\} *(f(c \# \# w)))=$
$\left(\sum w \in\{w . w \in\right.$ range (pseudo-proj-True $($ Suc $\left.n)) \wedge w!!0=c\right\}$. emeasure $M$ $($ pseudo-proj-True (Suc $n)-‘\{w\} \cap($ space $M)) *(f w))$

## proof -

fix $c$
show ( $\sum w \in$ range (pseudo-proj-True $n$ ). emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{w\} \cdot z=c \# \# x\} *(f(c \# \# w)))=$
$\left(\sum w \in\{w . w \in\right.$ range (pseudo-proj-True $($ Suc $\left.n)) \wedge w!!0=c\right\}$. emeasure $M$
(pseudo-proj-True (Suc n) -‘\{w\} $\cap($ space $M)) *(f w))$
proof (rule sum.reindex-cong)
show inj-on stl $\{w \in$ range (pseudo-proj-True (Suc n)). w !! $0=c\}$
proof
fix $x y$
assume $x \in\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\}$
and $y \in\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\}$
and $s t l x=$ stl $y$
have $x!!0=c$ using $\langle x \in\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0$
$=c\}$ > by $\operatorname{simp}$
moreover have $y!!0=c$ using $\langle y \in\{w \in$ range (pseudo-proj-True (Suc
$n)$ ). $w!!0=c\}$ 〉 by $\operatorname{simp}$
ultimately show $x=y$ using $\langle s t l x=$ stl $y\rangle$
by (smt snth. $\operatorname{simps}(1)$ stream-eq-Stream-iff)
qed
show range ( $p$ seudo-proj-True $n$ ) $=$ stl' $\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\}$
proof
show range (pseudo-proj-True $n$ ) $\subseteq$ stl ' $\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\}$
proof
fix $x$
assume $x \in$ range ( $p$ seudo-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 $\ldots=c \# \# x$ using <pseudo-proj-True $n x=x\rangle$ by simp
finally show ?thesis.
qed
hence $c \# \# x \in$ range (pseudo-proj-True (Suc n)) by (simp add: pseudo-proj-True-img)
thus $x \in \operatorname{stl}^{l}\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\}$
proof -
have $\exists s .(s \in$ range (pseudo-proj-True $(S u c n)) \wedge s!!0=c) \wedge s t l s$ $=x$
by (metis (no-types) $\langle c \# \# x \in$ range (pseudo-proj-True (Suc n)) $\rangle$ snth.simps(1) stream.sel(1) stream.sel(2))
then show? thesis
by force
qed
qed
show stl' $\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\} \subseteq$ range (pseudo-proj-True n)
proof
fix $x$
assume $x \in$ stl' $\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\}$
hence $\exists w \in\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\} . x=$ stl $w$ by auto
from this obtain $w$ where $w \in\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\}$ and $x=$ stl $w$ by auto
have $w \in$ range (pseudo-proj-True (Suc $n)$ ) and $w!!0=c$ using $\langle w \in$ $\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\}$ >
by auto
have $c \# \# x=w$ using $\langle x=$ stl $w\rangle\langle w!!0=c\rangle$ by force
also have $\ldots=$ pseudo-proj-True (Suc n) w using $\langle w \in$ range (pseudo-proj-True (Suc n)) >
using pseudo-proj-True-proj by auto
also have $\ldots=c \# \#$ pseudo-proj-True $n x$ using $\langle x=$ stl $w\rangle\langle w!!0=$ 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 $x \in$ range ( $p$ seudo-proj-True $n$ ) by auto
qed
qed
show $\bigwedge x . x \in\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=c\} \Longrightarrow$
emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{s t l x\} . z=c \# \#$
$x\} * \operatorname{ennreal}(f(c \# \#$ stl $x))=$
emeasure $M$ (pseudo-proj-True (Suc $n)-‘\{x\} \cap$ space $M) *$ ennreal $(f x)$
proof -
fix $w$
assume $w \in\{w \in$ range (pseudo-proj-True (Suc n)). w !! $0=c\}$
hence $w \in$ range (pseudo-proj-True (Suc n)) and $w!!0=c$ by auto
have $\{z \in$ space $M . \exists x \in$ pseudo－proj－True $n-‘\{$ stl $w\} . z=c \# \# x\}=$ （pseudo－proj－True（Suc $n$ ）－‘\｛w\} $\cap$ space $M$ ）
proof
show $\{z \in$ space $M . \exists x \in$ pseudo－proj－True $n-‘\{$ stl $w\} . z=c \# \# x\}$ $\subseteq$ pseudo－proj－True（Suc $n)-{ }^{\prime}\{w\} \cap$ space $M$

## proof

fix $z$
assume $z \in\{z \in$ space $M . \exists x \in$ pseudo－proj－True $n-'\{$ stl $w\} . z=c$
\＃\＃$x$ \}
hence $\exists x \in$ pseudo－proj－True $n-{ }^{`}\{$ stl $w\} . z=c \# \# x$ and $z \in$ space
$M$ by auto
from $\langle\exists x \in$ pseudo－proj－True $n-‘\{$ stl $w\} . z=c \# \# x\rangle$ obtain $x$ where $x \in$ 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 $\langle z=c \# \# x\rangle$
by（simp add：pseudo－proj－True－Suc－prefix）
also have $\ldots=c \# \#$ stl $w$ using 〈x $\quad$ pseudo－proj－True $n-‘\{$ stl $w\}$ 〉 by $\operatorname{simp}$
also have $\ldots=w$ using $\langle w!!0=c\rangle$ by force
finally have pseudo－proj－True（Suc n）$z=w$ ．
thus $z \in$ pseudo－proj－True（Suc $n$ ）－＇$\{w\} \cap$ space $M$ using $\prec z \in$ space
$M$＞by auto
qed
show pseudo－proj－True $($ Suc $n)-{ }^{‘}\{w\} \cap$ space $M \subseteq\{z \in$ space $M$ ． $\exists x \in$ pseudo－proj－True $n-‘\{$ stl $w\} . z=c \# \# x\}$
proof
fix $z$
assume $z \in$ pseudo－proj－True（Suc $n$ ）$-‘\{w\} \cap$ space $M$
hence $z \in$ 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 $\ldots=$ 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 $z \in$ pseudo－proj－True $n-‘\{$ stl $w\}$ by simp
have $z!!0$ \＃\＃pseudo－proj－True $n($ stl $z)=w$ using pseudo－proj－True－Suc－prefix $\langle p s e u d o-p r o j-T r u e ~(S u c ~ n) ~ z=w\rangle$ by $\operatorname{simp}$
also have $\ldots=c \# \#(s t l w)$ using $\langle w!!0=c\rangle$ by force
finally have $z!!0 \# \#$ pseudo－proj－True $n(s t l z)=c \# \#(s t l w)$ ．
hence $z!!0=c$ by $\operatorname{simp}$
hence $z=c \# \#(s t l z)$ by force
thus $z \in\{z \in$ space $M . \exists x \in$ pseudo－proj－True $n-‘\{$ stl $w\} . z=c \# \#$
$x\}$ using $\langle z \in$ space $M$ 〉
$\langle s t l z \in$ pseudo－proj－True $n-‘\{$ stl $w\}>$ by auto
qed
qed
hence emeasure $M\{z \in$ space $M . \exists x \in$ pseudo－proj－True $n-'\{$ stl $w\} . z$ $=c \# \# x\} * \operatorname{ennreal}(f(c \# \#$ stl $w))=$
emeasure $M$（pseudo－proj－True $($ Suc $n)-{ }^{\prime}\{w\} \cap$ space $\left.M\right) *$ ennreal $(f$ （ $c$ \＃\＃stl $w)$ ）by $\operatorname{simp}$
also have $\ldots=$ emeasure $M$ (pseudo-proj-True (Suc $n$ ) $-{ }^{‘}\{w\} \cap$ space $M) *$ ennreal $(f w)$ using $\langle w!!0=c\rangle$ by force
finally show emeasure $M\{z \in$ space $M . \exists x \in$ pseudo-proj-True $n-‘\{$ stl $w\} . z=c \# \# x\} * \operatorname{ennreal}(f(c \# \#$ stl $w))=$
emeasure $M$ (pseudo-proj-True $(S u c n)-‘\{w\} \cap$ space $M) *$ ennreal $(f$
w) .
qed
qed
qed
thus ?thesis by simp
qed
also have $\ldots=\left(\sum w \in\{w \in\right.$ range (pseudo-proj-True (Suc n)). w !! $0=$ True $\}$
$\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=$ False $\}$.
emeasure $M$ (pseudo-proj-True (Suc $n)-‘\{w\} \cap$ space $M) *$ ennreal $(f w))$
proof (rule sum.union-disjoint[symmetric])
show finite $\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=$ True $\}$ by (simp add: pseudo-proj-True-finite-image)
show finite $\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=$ False $\}$ by (simp add: pseudo-proj-True-finite-image)
show $\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=$ True $\} \cap\{w \in$ range (pseudo-proj-True (Suc n)). w !! $0=$ False $\}=\{ \}$
by auto
qed
also have $\ldots=\left(\sum w \in\right.$ range ( pseudo -proj-True (Suc $n$ )).emeasure $M$ (pseudo-proj-True (Suc $n)-‘\{w\} \cap$ space $M) *$ ennreal $(f w))$
proof (rule sum.cong)
show $\{w \in$ range (pseudo-proj-True $(S u c ~ n)) . w!!0=$ True $\} \cup\{w \in$ range (pseudo-proj-True (Suc n)). w !! $0=$ False $\}=$
range (pseudo-proj-True (Suc n))
proof
show $\{w \in$ range (pseudo-proj-True $($ Suc $n)) . w!!0=$ True $\} \cup\{w \in$ range (pseudo-proj-True (Suc n)). w !! $0=$ False $\}$

$$
\subseteq \text { range }(\text { pseudo-proj-True }(\text { Suc } n)) \text { by auto }
$$

show range (pseudo-proj-True (Suc n))
$\subseteq\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=\operatorname{Tr} u e\} \cup$
$\{w \in$ range (pseudo-proj-True (Suc n)). $w!!0=$ False $\}$
by (simp add: subsetI)
qed
qed $\operatorname{simp}$
finally show integral ${ }^{N}$ Mf =
( $\sum w \in$ range (pseudo-proj-True (Suc n)). emeasure $M$ (pseudo-proj-True (Suc $n)-‘\{w\} \cap$ space $M) *$ ennreal $(f w))$.
qed
lemma (in infinite-cts-filtration) F-n-integral-pos:
fixes $f:$ :bool stream $\Rightarrow$ real
assumes $f \in$ borel-measurable ( $F n$ )
and $\forall w .0 \leq f w$
shows has-bochner-integral Mf
( $\sum w \in$ range ( $p$ seudo-proj-True $n$ ). (measure $M$ ( $($ pseudo-proj-True $n)-‘\{w\}$
$\cap$ space $M)) *(f w))$
proof -
have integral ${ }^{N} M f=\left(\sum w \in\right.$ 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 $A E x$ in $M .0 \leq f x$ 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 $\ldots=$ enn2real $\left(\sum w \in\right.$ range (pseudo-proj-True $n$ ). (emeasure $M$
$(($ pseudo-proj-True $n)-‘\{w\} \cap$ space $M)) *(f w))$
using assms by (simp add: F-n-nn-integral-pos)
also have $\ldots=\left(\sum w \in\right.$ range ( pseudo -proj-True $n$ ). enn2real ( $($ emeasure $M$
$(($ pseudo-proj-True $n)-‘\{w\} \cap$ space $M)) *(f w)))$
proof (rule enn2real-sum)
show finite (range (pseudo-proj-True $n$ )) by (simp add: pseudo-proj-True-finite-image)
show $\wedge w . w \in$ range ( $p$ seudo-proj-True $n$ ) $\Longrightarrow$ emeasure $M$ (pseudo-proj-True
$n-‘\{w\} \cap$ space $M) *$ ennreal $(f w)<\top$
proof -
fix $w$
assume $w \in$ range (pseudo-proj-True $n$ )
show emeasure $M$ (pseudo-proj-True $n-‘\{w\} \cap$ space $M) *$ ennreal $(f w)<$
T
by (simp add: emeasure-eq-measure ennreal-mult-less-top)
qed
qed
also have $\ldots=\left(\sum w \in\right.$ range ( $p$ seudo-proj-True $n$ ). ( $($ measure $M$ ( $($ pseudo-proj-True
$n)-\{\{w\} \cap$ space $M)) *(f w)))$
by (simp add: Sigma-Algebra.measure-def assms(2) enn2real-mult)
finally have integral ${ }^{L} M f=\left(\sum w \in\right.$ range (pseudo-proj-True $n$ ). ( $($ measure $M$
$(($ pseudo-proj-True $n)-‘\{w\} \cap$ 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 $A E x$ in $M .0 \leq f x$ using assms by simp
have $\left(\int+\right.$ x. ennreal $\left.(f x) \partial M\right)=\left(\sum w \in\right.$ range (pseudo-proj-True $\left.n\right)$. (emeasure
$M(($ pseudo-proj-True $n)-‘\{w\} \cap$ space $M)) *(f w))$
using assms by (simp add: F-n-nn-integral-pos)
also have $\ldots=\left(\sum w \in\right.$ range (pseudo-proj-True $n$ ). ennreal (measure $M$ $(($ pseudo-proj-True $n)-‘\{w\} \cap$ space $M) *(f w)))$
proof (rule sum.cong, simp)

```
    fix }
    assume x\in range (pseudo-proj-True n)
    thus emeasure M (pseudo-proj-True n -' {x}\cap space M)* ennreal (fx)=
        ennreal (prob (pseudo-proj-True n -'{x}\cap space M)*fx)
        using assms(2) emeasure-eq-measure ennreal-mult'/ by auto
    qed
    also have ... = ennreal ( }\sumw\in\mathrm{ range (pseudo-proj-True n). (measure M
((pseudo-proj-True n) -`{w}\cap space M)* (fw)))
    proof (rule ennreal-sum)
    show finite (range (pseudo-proj-True n)) by (simp add: pseudo-proj-True-finite-image)
            show }\w.0\leq\operatorname{prob}(pseudo-proj-True n -'{w}\cap space M)*f
            using assms(2) measure-nonneg zero-le-mult-iff by blast
    qed
    finally show ( }\mp@subsup{\int}{}{+}\mathrm{ x. ennreal (fx) }\textrm{f}M)
        ennreal (\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True
n) -`{w}\cap space M)* (fw))).
    qed
    ultimately show ?thesis using has-bochner-integral-iff by blast
qed
lemma (in infinite-cts-filtration) F-n-integral:
    fixes f::bool stream }=>\mathrm{ real
    assumes f\in borel-measurable (F n)
    shows has-bochner-integral Mf
    (\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - '{w}
\cap space M))*(fw))
proof -
    define fpos where fpos=( }\lambdaw\mathrm{ w. max 0(f w))
    define fneg where fneg = ( \lambdaw. max 0 (-fw))
    have }\forallw.0\leqfpos w unfolding fpos-def by sim
    have }\forallw.0\leqfneg w unfolding fneg-def by sim
    have fpos \in borel-measurable ( Fn) using assms unfolding fpos-def by simp
    have fneg \in borel-measurable (F n) using assms unfolding fneg-def by simp
    have has-bochner-integral M fpos
    (\sum w\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - '{w}
space M))* (fpos w))
    using <fpos\in borel-measurable (F n)\rangle\langle\forall w. 0 \leqfpos w> by (simp add: F-n-integral-pos)
    moreover have has-bochner-integral M fneg
    (\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - '{w}
\space M))* (fneg w)
    using <fneg\in borel-measurable (F n)\rangle\langle\forall w.0 \leq fneg w〉 by (simp add: F-n-integral-pos)
    ultimately have posd: has-bochner-integral M (\lambdaw. fpos w - fneg w)
    ((\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - `{w}
\cap space M))* (fpos w)) -
    (\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - {{w}
\cap space M))* (fneg w)))
    by (simp add:has-bochner-integral-diff)
    have ((\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n)
```

```
-`{w}\cap space M))* (fpos w)) -
    (\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - '{w}
\cap space M))* (fneg w)))}
    (\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - '{w}
\cap space M)* fpos w -
            (measure M ((pseudo-proj-True n) -`{w} \cap space M)) * fneg w))
    by (rule sum-subtractf[symmetric])
    also have ... =
        (\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - '{w}
    \mathrm{ space M)*(fpos w - fneg w)))}
    proof (rule sum.cong, simp)
        fix }
        assume }x\in\mathrm{ range (pseudo-proj-True n)
    show prob (pseudo-proj-True n - '{x} \cap space M)* fpos x - prob (pseudo-proj-True
n-`{x}\cap space M)* fneg }x
                prob (pseudo-proj-True n -` {x} \cap space M)*(fpos x - fneg x)
            by (rule right-diff-distrib[symmetric])
    qed
    also have ... =
        (\sum w\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - '{w}
    space M)*fw))
    proof (rule sum.cong, simp)
            fix }
    assume x\in range (pseudo-proj-True n)
    show prob (pseudo-proj-True n -'{x} \cap space M)*(fpos x - fneg x) = prob
(pseudo-proj-True n -'{x}\cap space M)*fx
            unfolding fpos-def fneg-def by auto
    qed
    finally have ((\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True
n) - {{w}\cap space M)) * (fpos w)) -
    (\sumw\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - '{w}
\cap space M)) * (fneg w)))=
    (\sum w\in range (pseudo-proj-True n). (measure M ((pseudo-proj-True n) - '{w}
\cap space M)*fw)).
    hence has-bochner-integral M (\lambdaw. fpos w - fneg w) (\sumw\in range (pseudo-proj-True
n). (measure M ((pseudo-proj-True n) -`{w}\cap space M)*fw))
            using posd by simp
            moreover have }\w\mathrm{ . fpos w - fneg w =fw 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 }=>\mathrm{ real
    assumes f\in borel-measurable (F n)
    shows has-bochner-integral Mf
    (\sumw\in range (pseudo-proj-True n). (prod (prob-component p w) {0..<n})* (f
w))
proof -
```

have $\forall w \in$ range (pseudo-proj-True $n)$. (measure $M$ ( $($ pseudo-proj-True $n)-‘\{w\}$ $\cap$ space $M)) * f w=$
$(\operatorname{prod}($ prob-component $p w)\{0 . .<n\}) *(f w)$
proof
fix $w$
assume $w \in$ range ( $p s e u d o-p r o j-T r u e ~ n$ )
thus prob (pseudo-proj-True $n-‘\{w\} \cap$ 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 $\Rightarrow$ real
assumes $f \in$ borel-measurable ( $F n$ )
shows expectation $f=$
$\left(\sum w \in\right.$ range (pseudo-proj-True $\left.n\right) .($ 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 $A \cap B=\{ \}$
and $A \cup B=C$
shows sum $g C=\operatorname{sum} g A+\operatorname{sum} g B$
using sum.union-disjoint $[O F \operatorname{assms}(1-3)]$ and $\operatorname{assms}(4)$ by auto
lemma (in infinite-cts-filtration) borel-Suc-expectation:
fixes $f::$ bool stream $\Rightarrow$ real
assumes $f \in$ borel-measurable ( $F$ (Suc $n$ ))
and $g \in$ measurable ( $F n$ ) $N$
and set-discriminating $n g N$
and $g-'\{g z\} \in \operatorname{sets}(F n)$
and $\forall y z .(g y=g z \wedge$ snth $y n=$ snth $z n) \longrightarrow f y=f z$
shows expectation $\left(\lambda x . f x *\right.$ indicator $\left.\left(g-{ }^{\prime}\{g z\}\right) x\right)=$
prob $\left(g-{ }^{\prime}\{g z\}\right) *(p * f($ pseudo-proj-True $n z)+$ $(1-p) * f($ pseudo-proj-False $n z))$
proof -
define expind where expind $=(\lambda x . f x *$ indicator $(g-'\{g z\}) x)$
have expind $\in$ borel-measurable ( $F$ (Suc $n$ )) unfolding expind-def
proof (rule borel-measurable-times, (simp add:assms(1,2)))
show indicator $(g-‘\{g z\}) \in$ borel-measurable $(F(S u c ~ n))$
proof (rule borel-measurable-indicator)
have $g-'\{g z\} \in$ sets (nat-filtration $n$ )
using assms nat-filtration-borel-measurable-singleton natural-filtration by
simp
hence $g-‘\{g z\} \in$ sets $(F n)$ using natural-filtration by simp

```
        thus g-'{gz}\in sets (F (Suc n))
            using nat-filtration-Suc-sets natural-filtration by blast
        qed
    qed
    hence expectation expind =
    (\sumw\in range (pseudo-proj-True (Suc n)). (measure M ((pseudo-proj-True (Suc
n)) -'{w} \cap space M)) *(expind w))
    by (simp add:F-n-integral has-bochner-integral-integral-eq)
    also have ... = (\sumw\in range (pseudo-proj-True (Suc n)) \capg-`{gz}.
    (measure M ((pseudo-proj-True (Suc n)) -`{w}\cap space M)) * (expind w)) +
    (\sumw\in range (pseudo-proj-True (Suc n)) - g-`{gz}.
    (measure M ((pseudo-proj-True (Suc n)) - {{w}\cap space M)) * (expind w))
    by (simp add: Int-Diff-Un Int-Diff-disjoint assms sum-union-disjoint' pseudo-proj-True-finite-image)
also have ... = (\sumw\in range (pseudo-proj-True (Suc n)) \capg-`{{gz}.
    (measure M ((pseudo-proj-True (Suc n)) - {{w}\cap space M)) *(expind w))
proof -
    have }\forallw\in\mathrm{ range (pseudo-proj-True (Suc n)) - g-`{g z}. expind w=0
    proof
        fix w
        assume w\in range (pseudo-proj-True (Suc n)) - g-' {gz}
        thus expind w=0 unfolding expind-def by simp
    qed
    thus ?thesis by simp
qed
also have ... = (\sumw\in range (pseudo-proj-True (Suc n)) \capg-`{gz}.
    (measure M ((pseudo-proj-True (Suc n)) - `{w}\cap space M))*fw)
proof -
    have }\forallw\in\mathrm{ range (pseudo-proj-True (Suc n)) \g-`{gz}. expind w=fw
    proof
        fix w
        assume w\in range (pseudo-proj-True (Suc n)) \capg-`{gz}
        hence w\ing-'{gz} by simp
        thus expind w =f w unfolding expind-def by simp
    qed
    thus?thesis by simp
qed
also have ... = (\sumw\in(pseudo-proj-True n)'(g-`{gz})\cup(pseudo-proj-False
n)'(g-'{gz}).
    (measure M ((pseudo-proj-True (Suc n)) - {{w}\cap space M)) *fw) using
f-borel-Suc-preimage[of g] assms(1,2,3) by auto
    also have }\ldots=(\sumw\in(pseudo-proj-True n)'(g-'{gz})
        (measure M ((pseudo-proj-True (Suc n)) - `{w}\cap space M))*fw)+
        (\sumw\in(pseudo-proj-False n)' (g-'{gz}).
        (measure M ((pseudo-proj-True (Suc n)) - {{w}\cap space M))*fw)
    proof (rule sum-union-disjoint')
        show finite (pseudo-proj-True n' g-`{gz})
        proof -
        have finite (range (pseudo-proj-True n)) by (simp add: pseudo-proj-True-finite-image)
            moreover have pseudo-proj-True n' g -' {gz}\subseteq 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 $n ' g-‘\{g z\} \subseteq$ range (pseudo-proj-False
n)
by (simp add: image-mono)
ultimately show? ?thesis by (simp add:finite-subset)
qed
show pseudo-proj-True $n ' g-‘\{g z\} \cap$ pseudo-proj-False $n ' g-‘\{g z\}=\{ \}$ proof (rule ccontr)
assume pseudo-proj-True $n ' g-'\{g z\} \cap$ pseudo-proj-False $n ' g-'\{g z\}$ $\neq\{ \}$
hence $\exists y$. $y \in$ pseudo-proj-True $n ' g-‘\{g z\} \cap$ pseudo-proj-False $n ‘ g-‘$ $\{g z\}$ by auto
from this obtain $y$ where $y \in$ pseudo-proj-True $n ' g-{ }^{\prime}\{g z\}$ and $y \in$ pseudo-proj-False $n$ ' $g-‘\{g z\}$ by auto
have $\exists y t . y t \in g-‘\{g z\} \wedge y=$ pseudo-proj-True $n$ yt using $\prec y \in$ pseudo-proj-True $n^{\prime} g-‘\{g z\}$ ’ by auto
from this obtain $y t$ where $y=$ pseudo-proj-True $n$ yt by auto
have $\exists y f . y f \in g-\{g z\} \wedge y=$ pseudo-proj-False $n$ yf using $\langle y \in$ pseudo-proj-False $n$ ' $g-‘\{g z\}$ > by auto
from this obtain $y f$ where $y=$ pseudo-proj-False $n y f$ by auto
have snth $y n=$ True using $\langle y=$ pseudo-proj-True $n$ yt〉unfolding pseudo-proj-True-def by simp
moreover have snth y $n=$ False using $\langle y=$ pseudo-proj-False $n y f\rangle$ un-
folding pseudo-proj-False-def by simp
ultimately show False by simp
qed
qed $\operatorname{simp}$
also have $\ldots=\left(\sum w \in\right.$ pseudo-proj-True $n ' g-‘\{g z\}$. prob (pseudo-proj-True $($ Suc $n)-‘\{w\} \cap$ space $M) * f($ pseudo-proj-True $n z))+$
( $\sum w \in$ pseudo-proj-False $n^{\prime} g-'\{g z\}$. prob (pseudo-proj-True (Suc $n$ ) $-‘\{w\}$
$\cap$ space $M) * f w)$
proof -
define $z t$ where $z t=$ pseudo-proj-True $n z$
have eqw: $\bigwedge w . w \in p s e u d o-p r o j-T r u e ~ n ' g-'\{g z\} \Longrightarrow(g w=g z t \wedge$ snth $w n$ $=$ snth zt n)
proof
fix $w$
assume $w \in$ 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 $n y t$ and $g y t=g z$ by auto
have $g w=g$ yt using $\langle w=$ pseudo-proj-True $n$ yt〉 nat-filtration-not-borel-info[of g] natural-filtration
assms by (metis comp-apply)
also have $\ldots=g$ zt using assms using nat-filtration-not-borel-info[of $g]$ natural-filtration $\langle g y t=g z\rangle$
unfolding $z t$-def by (metis comp-apply)
finally show $g w=g z t$.
show $w!!n=z t!!n$ using $\langle w=$ pseudo-proj-True $n y t\rangle$ unfolding $z t$-def pseudo-proj-True-def by simp
qed
hence $\bigwedge w . w \in$ pseudo-proj-True $n^{\prime} g-'\{g z\} \Longrightarrow f w=f z t$

## proof

fix $w$
assume $w \in$ pseudo-proj-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 $\ldots=\left(\sum w \in p s e u d o-p r o j-T r u e n ' g-‘\{g z\}\right.$. prob (pseudo-proj-True $($ Suc $n)-‘\{w\} \cap$ space $M) * f($ pseudo-proj-True $n z))+$
( $\sum w \in$ pseudo-proj-False $n ' g-‘\{g z\}$. prob (pseudo-proj-True (Suc n) $-{ }^{\prime}\{w\}$ $\cap$ space $M) * f($ pseudo-proj-False $n z))$
proof -
define $z f$ where $z f=$ pseudo-proj-False $n z$
have eqw: $\bigwedge w . w \in$ pseudo-proj-False $n ' g-'\{g z\} \Longrightarrow(g w=g z f \wedge$ snth $w$ $n=$ snth zf $n$ )
proof
fix $w$
assume $w \in$ 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 yf where $w=$ pseudo-proj-False $n y f$ and $g y f=g z$ by auto
have $g w=g$ yf using $\langle w=$ pseudo-proj-False $n y f\rangle$ nat-filtration-not-borel-info'[of g] natural-filtration
assms by (metis comp-apply)
also have $\ldots=g z f$ using assms using nat-filtration-not-borel-info' $[o f g]$ natural-filtration $\langle g y f=g z\rangle$
unfolding $z f$-def by (metis comp-apply)
finally show $g w=g z f$.
show $w!!n=z f!!n$ using $\langle w=$ pseudo-proj-False $n y f\rangle$ unfolding $z f$-def pseudo-proj-False-def by simp
qed
hence $\wedge w . w \in$ pseudo-proj-False $n ' g-‘\{g z\} \Longrightarrow f w=f z f$

## proof

fix $w$
assume $w \in$ pseudo-proj-False $n ' g-‘\{g z\}$
hence $g w=g$ zf $\wedge$ snth $w n=$ snth zf $n$ using eqw [of $w]$ by simp
thus $f w=f z f$ using assms by blast

```
qed
thus ?thesis by simp
```

qed
also have $\ldots=\left(\sum w \in\right.$ pseudo-proj-True $n$ ' $g-‘\{g z\}$. prob (pseudo-proj-True $($ Suc $n)-‘\{w\} \cap$ space $M)) * f($ 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 $n z)$
by (simp add: sum-distrib-right)
also have $\ldots=\left(\sum w \in\right.$ pseudo-proj-True $n ' g-{ }^{\prime}\{g z\}$. 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 $\bigwedge w . w \in$ pseudo-proj-True $n^{\prime} g-'\{g z\} \Longrightarrow$ (prob (pseudo-proj-True (Suc $n)-(\{w\})=$
$(\operatorname{prob}(\{x$ stake $n x=$ stake $n w\})) * p)$
proof -
fix $w$
assume $w \in$ 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 $n y t$ and $g y t=g z$ by auto
hence snth $w n$ unfolding pseudo-proj-True-def by simp
have pseudo-proj-True (Suc n) $w=w$ using $\langle w=$ pseudo-proj-True $n$ yt $\rangle$
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 $\operatorname{simp}$
hence $\operatorname{prob}($ pseudo-proj-True $($ Suc $n)-‘\{w\})=\operatorname{prob}\{x$. 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 $\ldots=\operatorname{prob}\{x$. stake $n x=$ stake $n w\} * p$ using $\langle$ snth $w n\rangle$ unfolding prob-component-def by simp
finally show prob (pseudo-proj-True $($ Suc $n)-‘\{w\})=\operatorname{prob}\{x$. stake $n x$ $=$ stake $n w\} * p$.
qed
thus ?thesis using bernoulli bernoulli-stream-space by simp
qed
also have $\ldots=\left(\sum w \in\right.$ 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 $\Lambda w . w \in$ 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 $w \in$ 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 $n y t$ and $g y t=g z$ by auto
hence $\neg s n t h$ w unfolding pseudo-proj-False-def by simp
have pseudo-proj-True (Suc n) $w=w$ using $\langle w=$ pseudo-proj-False $n$ yt $\rangle$ 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 $\operatorname{simp}$
hence $\operatorname{prob}($ pseudo-proj-True $(S u c n)-‘\{w\})=\operatorname{prob}\{x$. 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 $\ldots=\operatorname{prob}\{x$. stake $n x=$ stake $n w\} *(1-p)$ using $\langle\neg$ snth $w n\rangle$ unfolding prob-component-def by simp
finally show prob (pseudo-proj-True (Suc $n$ ) -' $\{w\} \cap$ space $M)=\operatorname{prob}\{x$. stake $n x=$ stake $n w\} *(1-p)$ using bernoulli bernoulli-stream-space by simp
qed
thus ?thesis by simp
qed
also have $\ldots=\left(\sum w \in\right.$ pseudo-proj-True $n ' g-{ }^{\prime}\{g z\}$. prob $(\{x$. stake $n x=$ stake $n w\})) * p * f($ pseudo-proj-True $n z)+$
$\left(\sum w \in\right.$ pseudo-proj-False $n ' g-'\{g z\}$. prob $\{x$. stake $n x=$ stake $\left.n w\}\right) *(1$ $-p) * f(p s e u d o-p r o j-F a l s e n z)$
by (simp add:sum-distrib-right)
also have $\ldots=\operatorname{prob}\left(g-{ }^{`}\{g z\}\right) * p * f($ pseudo-proj-True $n z)+$

$$
\left(\sum w \in \text { pseudo-proj-False } n ' g-‘\{g z\} . \text { prob }\{x \text {. stake } n x=\text { stake } n w\}\right) *(1
$$

$-p) * f($ pseudo-proj-False $n z)$
proof -
have projset: $\bigwedge w . w \in$ pseudo-proj-True $n ' g-'\{g z\} \Longrightarrow\{x$. stake $n x=$ stake $n w\} \in$ sets $M$

## proof -

fix $w$
assume $w \in$ pseudo-proj-True $n ' g-‘\{g z\}$
hence $\exists y . w=$ 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 $n w\} \quad$ using pseudo-proj-True-preimage-stake by simp
moreover have pseudo-proj-True $n-'\{w\} \in$ sets $M$
using $\langle w=$ pseudo-proj-True $n w\rangle$ bernoulli bernoulli-stream-space pseudo-proj-True-singleton by auto
ultimately show $\{x$. stake $n x=$ stake $n w\} \in$ sets $M$ by simp
qed
have $\left(\sum w \in\right.$ pseudo-proj-True $n^{\prime} g-‘\{g z\}$. prob $(\{x$. stake $n x=$ stake $\left.n w\})\right)$ $=$
prob $(\bigcup w \in$ 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 sub-set-UNIV)
show ( $\lambda i$. $\{x$. stake $n x=$ stake $n i\}$ )'pseudo-proj-True $n ' g-$ ' $\{g z\} \subseteq$ events using projset by auto
show disjoint-family-on ( $\lambda i$. $\{x$. 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 $u \in$ pseudo-proj-True $n ' g-‘\{g z\}$ and $v \in$ pseudo-proj-True $n '$ $g-‘\{g z\}$ and $u \neq v$ note uvprops $=$ this
show $\{x$. stake $n x=$ stake $n u\} \cap\{x$. stake $n x=$ stake $n v\}=\{ \}$
proof (rule ccontr)
assume $\{x$. stake $n x=$ stake $n u\} \cap\{x$. stake $n x=$ stake $n v\} \neq\{ \}$
hence $\exists u u$. uú $\{x$. stake $n x=$ stake $n u\} \cap\{x$. stake $n x=$ stake $n v\}$ by auto
from this obtain $u u$ where $u u \in\{x$. stake $n x=$ stake $n u\} \cap\{x$. stake $n x=$ stake $n v\}$ by auto
hence stake $n u u=$ stake $n u$ and stake $n u u=$ stake $n v$ by auto
moreover have stake $n u \neq$ 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 $\ldots=\operatorname{prob}(\bigcup w \in p s e u d o-p r o j-T r u e ~ n ' g-'\{g z\}$. pseudo-proj-True $n-‘\{w\})$
proof -
have $\bigwedge w . w \in$ pseudo-proj-True $n^{\prime} g-'\{g z\} \Longrightarrow\{x$. stake $n x=$ stake $n w\}$ $=$ pseudo-proj-True $n-\{w\}$
using pseudo-proj-True-preimage-stake pseudo-proj-True-proj by force
hence $(\bigcup w \in$ pseudo-proj-True $n ' g-‘\{g z\}$. $\{x$. stake $n x=$ stake $n w\})=$
$(\bigcup w \in$ pseudo-proj-True $n ' g-'\{g z\}$. pseudo-proj-True $n-‘\{w\})$ by auto
thus ?thesis by simp
qed
also have $\ldots=\operatorname{prob}($ pseudo-proj-True $n-‘(p s e u d o-p r o j-T r u e ~ n ' g-'\{g z\}))$
by (metis vimage-eq-UN)
also have $\ldots=\operatorname{prob}(g-‘\{g z\})$ using pseudo-proj-preimage[symmetric, of $g$ $n N z]$
pseudo-proj-preimage'[of g n] assms by simp
finally have ( $\sum w \in$ pseudo-proj-True $n ' g-{ }^{\prime}\{g z\}$. prob ( $\{x$. stake $n x=$ stake $n w\}))=\operatorname{prob}\left(g-{ }^{\prime}\{g z\}\right)$.
thus ?thesis by simp
qed
also have $\ldots=\operatorname{prob}(g-‘\{g z\}) * p * f($ pseudo-proj-True $n z)+$ $\operatorname{prob}(g-‘\{g z\}) *(1-p) * f($ pseudo-proj-False $n z)$
proof -
have projset: $\bigwedge w . w \in$ pseudo-proj-False $n ' g-'\{g z\} \Longrightarrow\{x$. stake $n x=$ stake $n w\} \in$ sets $M$

## proof -

fix $w$
assume $w \in$ pseudo-proj-False $n ' g-‘\{g z\}$
hence $\exists y$. $w=$ 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 $n w\} \quad$ using pseudo-proj-False-preimage-stake by simp
moreover have pseudo-proj-False $n-‘\{w\} \in$ sets $M$
using $\langle w=$ pseudo-proj-False $n w\rangle$ bernoulli bernoulli-stream-space pseudo-proj-False-singleton by auto
ultimately show $\{x$. stake $n x=$ stake $n w\} \in$ sets $M$ by simp
qed
have ( $\sum w \in$ pseudo-proj-False $n ' g-‘\{g z\}$. prob $(\{x$. stake $n x=$ stake $n$ $w\}))=$
prob $(\bigcup w \in$ 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 sub-set-UNIV)
show ( $\lambda i$. $\{x$. stake $n x=$ stake $n i\}$ )'pseudo-proj-False $n ' g-$ ' $\{g z\} \subseteq$ events using projset by auto
show disjoint-family-on ( $\lambda i$. $\{x$. 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 $u \in$ pseudo-proj-False $n^{\prime} g-‘\{g z\}$ and $v \in$ pseudo-proj-False $n$
${ }^{\prime} g-'\{g z\}$ and $u \neq v$ note uvprops $=$ this
show $\{x$. stake $n x=$ stake $n u\} \cap\{x$. stake $n x=$ stake $n v\}=\{ \}$
proof (rule ccontr)
assume $\{x$. stake $n x=$ stake $n u\} \cap\{x$. stake $n x=$ stake $n v\} \neq\{ \}$
hence $\exists u u$. uú \{x. stake $n x=$ stake $n u\} \cap\{x$. stake $n x=$ stake $n v\}$ by auto
from this obtain $u u$ where $u u \in\{x$. stake $n x=$ stake $n u\} \cap\{x$. stake $n x=$ stake $n v\}$ by auto
hence stake $n u u=$ stake $n u$ and stake $n u u=$ stake $n v$ by auto
moreover have stake $n u \neq$ 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 $\ldots=$ prob $(\bigcup w \in$ pseudo-proj-False $n ' g-‘\{g z\}$. pseudo-proj-False $n-‘\{w\})$
proof -

```
    have }\bigwedgew.w\in\mathrm{ pseudo-proj-False n' g-'{gz} }\Longrightarrow{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 (Uw\inpseudo-proj-False n'g-'{gzz}.{x. stake n x = stake n w})=
            (U w\inpseudo-proj-False n'g -'{gz}. pseudo-proj-False n -'{w}) by auto
        thus ?thesis by simp
    qed
    also have ... = prob (pseudo-proj-False n - '(pseudo-proj-False n'g -'{gz}))
by (metis vimage-eq-UN)
    also have ... = prob (g-`{g z}) using pseudo-proj-False-preimage[symmetric,
of g n Nz]
        pseudo-proj-False-preimage'[of g n] assms by simp
    finally have (\sumw\inpseudo-proj-False n'g-'{g z}. prob ({x. stake n x =
stake n w}))}=\operatorname{prob}(g-`{gz})
    thus ?thesis by simp
    qed
    also have \ldots}=\operatorname{prob}(g-`{gz})*(p*f(pseudo-proj-True nz)
        (1-p)*f(pseudo-proj-False n z))
    using distrib-left[symmetric, of prob (g-`{gz}) p*f(pseudo-proj-True n z)
(1-p)*f(pseudo-proj-False n z)]
    by simp
    finally show expectation (\lambdax.fx* indicator (g-`{gz}) x)=
        prob (g-`{gz})*(p*f(pseudo-proj-True nz) +
        (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 $\Rightarrow$ real
assumes $f \in$ borel-measurable ( $F($ Suc $n)$ )
shows expectation ( $\lambda x . f x *$ indicator (pseudo-proj-True $n-'\{p s e u d o-p r o j-T r u e$
$n z\}) x$ ) $=$
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 $f \in$ 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 pseudo-proj-True $n-‘\{$ pseudo-proj-True $n z\} \in$ sets $(F n)$
by (simp add: nat-filtration-singleton natural-filtration pseudo-proj-True-proj)
show $\forall y z$. (pseudo-proj-True $n y=$ pseudo-proj-True $n z \wedge$ snth $y n=$ snth $z$
$n) \longrightarrow f y=f z$
proof (intro allI impI conjI)
fix $y z$
assume pseudo-proj-True $n y=$ pseudo-proj-True $n z \wedge y!!n=z!!n$
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 \(n z\rangle\) 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 ( $F(S u c n)$ )
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 Mgfw=p*f(pseudo-proj-True nw) $+(1-p) * f$
(pseudo-proj-False $n$ w)
proof -
have $n z: \operatorname{prob}(g-'\{g w\}) \neq 0$
proof -
have pseudo-proj-True $n-‘\{p s e u d o-p r o j-T r u e ~ n w\} \subseteq g-'\{g w\}$
proof -
have $\forall f n m s . f \notin F n \rightarrow_{M} m \vee \neg$ set-discriminating $n f m \vee$ pseudo-proj-True
$n-‘ f-‘\left\{f s::^{\prime} a\right\}=f-‘\{f s\}$
by (meson pseudo-proj-preimage')
then show ?thesis using assms by blast
qed
moreover have prob (pseudo-proj-True $n-\{$ pseudo-proj-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 $n-‘\{p s e u d o-p r o j-T r u e ~ n ~ w\} \in$ 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-proj-True $n-‘\{p s e u d o-p r o j-T r u e ~ n w\}$
$\notin$ events $\vee$ prob $($ pseudo-proj-True $n-'\{p s e u d o-p r o j-T r u e ~ n ~ w\}) \leq \operatorname{prob}(g-'\{g$
$w\})$
using 〈pseudo-proj-True $n-‘\{p s e u d o-p r o j-T r u e ~ n w\} \subseteq g-‘\{g w\}$ 〉
increasing $D$ measure-increasing by blast
then show ?thesis
using $\langle 0<$ prob (pseudo-proj-True $n-‘\{p s e u d o-p r o j-T r u e ~ n ~ w\})\rangle\langle g-‘\{g$ $w\} \in$ events $\langle p s e u d o-p r o j-T r u e ~ n-'\{p s e u d o-p r o j-T r u e ~ n w\} \in$ events $\rangle$ by linarith qed
qed
hence expl-cond-expect $M g$ f $w=$
expectation $(\lambda x . f x *$ indicator $(g-‘\{g w\} \cap$ space $M) x) /$
$\operatorname{prob}(g-‘\{g w\} \cap$ space $M)$ unfolding expl-cond-expect-def img-dce-def
by $\operatorname{simp}$
also have $\ldots=\operatorname{expectation}(\lambda x . f x *$ indicator $(g-‘\{g w\}) x) / \operatorname{prob}(g-‘\{g$ w\})
using bernoulli by (simp add:bernoulli-stream-space)
also have $\ldots=\operatorname{prob}(g-‘\{g w\}) *(p * f($ pseudo-proj-True $n w)+$ $(1-p) * f($ pseudo-proj-False $n w)) / \operatorname{prob}(g-‘\{g w\})$
proof -
have expectation $(\lambda x . f x *$ indicator $(g-‘\{g w\}) x)=\operatorname{prob}(g-‘\{g w\}) *$ ( $p * f$ (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 $g \in F n \rightarrow_{M} N$ using assms by simp
show set-discriminating $n g N$ using assms by simp
show $g-{ }^{\prime}\{g w\} \in$ 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 $\ldots=p * f(p s e u d o-p r o j-T r u e ~ n w)+(1-p) * f(p s e u d o-p r o j-F a l s e ~ n$
$w)$ using $n z$ by $\operatorname{simp}$
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 $\forall w . g-‘\{g w\} \in \operatorname{sets}(F n)$
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 $A E w$ 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 $A E$ w in $M$. real-cond-exp $M(f c t-g e n-s u b a l g e b r a ~ M N g) f w=$ expl-cond-expect $M g f w$
proof (rule charact-cond-exp ${ }^{\prime}$ )
show disc-fct $g$
proof -
have $g=g \circ($ 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 ( $g \circ(p s e u d o-p r o j-T r u e ~ n))$ unfolding disc-fct-def
by (metis countable-image image-comp)
thus ?thesis using $\langle g=g \circ$ (pseudo-proj-True $n$ ) 〉 by simp
qed
show integrable Mf using assms nat-filtration-borel-measurable-integrable nat-ural-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 $\Lambda w$. 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 $A E$ 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 $n w\} \in \operatorname{sets}(F n)$
proof
fix $w$
show pseudo-proj-True $n-‘\{p s e u d o-p r o j-T r u e ~ n w\} \in \operatorname{sets}(F n)$
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`\langler \in range (pseudo-proj-True n)\rangle\langler\in
space M> by blast
    thus {r}\subseteqrange (pseudo-proj-True n)\cap(pseudo-proj-True n -'{r}\cap space
M) by auto
    show range (pseudo-proj-True n) \cap(pseudo-proj-True n -'{r}\cap space M)
\subseteq \{ r \}
    proof
            fix }
            assume }x\in\mathrm{ range (pseudo-proj-True n) }\cap\mathrm{ (pseudo-proj-True n -`{r} }
space M)
            hence }x\in\mathrm{ range (pseudo-proj-True n) and xG (pseudo-proj-True n -'{r})
by auto
            have x = pseudo-proj-True n x using <x\in range (pseudo-proj-True n)>
pseudo-proj-True-proj by auto
            also have ... =r using <x\in(pseudo-proj-True n -'{r})> by simp
            finally have x=r.
            thus }x\in{r}\mathrm{ by simp
            qed
        qed
    ultimately show }\existsA\in\mathrm{ events. range (pseudo-proj-True n) }\capA={r} by aut
    qed
    show }\forallyz.pseudo-proj-True n y = pseudo-proj-True nz\wedge y!! n=z z!! n
fy=fz
    proof (intro allI impI conjI)
            fix }y
            assume pseudo-proj-True n y= pseudo-proj-True n z\wedge 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 =fz 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\mathrm{ and }p<1\mathrm{ 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) fkw=(fkw)\#(smap-stoch-proc nf(Suck)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 \(p \leq\) Suc \(n \Longrightarrow\) nth (smap-stoch-proc (Suc n) fkw) \(p=f(k+p) w\)
proof (induction \(n\) arbitrary: \(p k\) )
    case 0
    hence \(p=0\) by simp
    hence (smap-stoch-proc (Suc 0) fkw)!p=((fkw)\#(smap-stoch-proc \(0 f\)
(Suc k) w))! 0 by simp
    also have \(\ldots=f(k+p) w\) using \(\langle p=0\rangle\) by \(\operatorname{simp}\)
    finally show? case .
next
    case (Suc n)
    show ?case
    proof (cases \(\exists\) m. \(p=\) Suc \(m\) )
        case True
        from this obtain \(m\) where \(p=\) Suc \(m\) by auto
        hence smap-stoch-proc (Suc (Suc n)) fkw!p=(smap-stoch-proc (Suc n) \(f\)
(Suc \(k\) ) \(w\) )! \(m\) by simp
    also have \(\ldots=f((\) Suc \(k)+m) w\) using Suc(1)[of m] Suc.prems \(\langle p=\) Suc m>
by blast
    also have \(\ldots=f(k+(\) Suc \(m)) w\) by simp
    finally show smap-stoch-proc (Suc (Suc n)) fkw!p=f(k+p)wusing «p
\(=\) Suc \(m>\) by \(\operatorname{simp}\)
    next
        case False
        hence \(p=0\) using less-Suc-eq-0-disj by blast
        thus smap-stoch-proc (Suc (Suc n)) fkw!p=f(k+p)w by simp
    qed
qed
```

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

```
lemma proj-stoch-proc-component:
    shows }k<n\Longrightarrow(snth (proj-stoch-proc f n w)k)=fk
    and n\leqk\Longrightarrow(snth (proj-stoch-proc f nw)k)=fnw
proof -
    show k<n\Longrightarrow proj-stoch-proc f n w !! k=fkw
    proof -
        assume k<n
        hence }\existsm.n=Suc m using less-imp-Suc-add by blas
        from this obtain m}\mathrm{ 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\rangle by (simp add: smap-stoch-proc-length)
        also have ... = fkw using <n=Suc m><k<n` smap-stoch-proc-nth
            by (metis Suc-leI add.left-neutral)
        finally show ?thesis.
    qed
    show n\leqk\Longrightarrow(snth (proj-stoch-proc f n w)k)=fnw
    proof -
        assume n\leqk
        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\leqn
    shows fkx = snth(proj-stoch-proc f n x)k
    proof (cases k<n)
        case True
        thus ?thesis using proj-stoch-proc-component[of kn f x] assms by simp
    next
        case False
    hence k=n using assms by simp
    thus ?thesis using proj-stoch-proc-component[of knfx] 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\leqn
shows fkx = fky
proof -
    have fkx = snth (proj-stoch-proc f n x) k using assms proj-stoch-proc-component'[of
knf] by simp
    also have ... = snth (proj-stoch-proc f n y) k using assms by simp
    also have ... = fk y using assms proj-stoch-proc-component'[of knf] 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 (\lambdax. proj-stoch-proc f n x !! m) \inM 稇 N
    proof (cases m<n)
        case True
    hence }\forallx\mathrm{ . proj-stoch-proc f n x !! m=fmx by (simp add: proj-stoch-proc-component)
        hence ( }\lambdax.\mathrm{ 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 }\forallx\mathrm{ . proj-stoch-proc f nx !! m=fn x by (simp add: proj-stoch-proc-component)
        hence ( }\lambdax.\mathrm{ 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 \in measurable (F n) (stream-space N)
proof (rule measurable-stream-space2)
    fix m
    show ( }\lambdax.\mathrm{ proj-stoch-proc f n x !! m) E measurable (F n) N
    proof (cases m<n)
        case True
    hence }\forallx\mathrm{ . proj-stoch-proc f nx !! m=fmx by (simp add: proj-stoch-proc-component)
        hence ( }\lambdax\mathrm{ . 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 }\forallx.proj-stoch-proc f nx !! m=fnx by (simp add: proj-stoch-proc-component
        hence ( }\lambdax.\mathrm{ 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 $f n \in F n \rightarrow_{M}$ 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 $\Rightarrow{ }^{\prime} b::\{t 0$-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 $X n w) m=\operatorname{snth}$ (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 $\ldots=X m$ (pseudo-proj-True $n w$ )
proof (rule borel-adapt-nat-filtration-info[symmetric], (simp add:assms))
show $m \leq n$ using True by linarith
qed
also have $\ldots=\operatorname{snth}($ 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 $\ldots=X n$ (pseudo-proj-True $n w$ )
by (rule borel-adapt-nat-filtration-info[symmetric]) (auto simp add:assms)
also have $\ldots=$ 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 $\Rightarrow{ }^{\prime} b::\{t 0$-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)\circ(pseudo-proj-True
n)
    proof
        fix }
        show proj-stoch-proc X n x = (proj-stoch-proc X n o 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 }\forallw. 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} = (\bigcapi\in{m.m\leqn}.
(Xi) -'{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}\subseteq(\bigcapi\in{m.m\leqn}. Xi
-'{Xiw})
    proof
    fix }
    assume x f 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\in{m.m\leqn}\Longrightarrowx\in(Xi)-{{Xiw
    proof -
        fix }
        assume i\in{m.m\leqn}
        hence }i\leqn\mathrm{ by auto
        have X ix = snth (psX x) i unfolding psX-def
        by (metis Suc-le-eq Suc-le-mono <i \leqn> 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 \leq n> le-Suc-eq nat.simps(1) proj-stoch-proc-component(1)
                proj-stoch-proc-component(2))
            finally have X ix=X iw.
            thus }x\in(Xi)-'{Xiw} by sim
    qed
```

```
        thus }x\in(\bigcapi\in{m.m\leqn}.(Xi)-`{Xiw})\mathrm{ by auto
    qed
    show (\bigcapi\in{m.m\leqn}. (Xi) -'{Xiw})\subseteq(proj-stoch-proc X n) -'{proj-stoch-proc
X n w}
    proof
        fix }
    assume }x\in(\bigcapi\in{m.m\leqn}.(Xi)-'{Xiw}
    hence }\i.i\in{m.m\leqn}\Longrightarrowx\in(Xi)-{{Xiw} by sim
    hence \i. i }\{m.m\leqn}\LongrightarrowXix=Xiw by sim
    hence }\bigwedgei.i\leqn\LongrightarrowXix=Xiw by aut
    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(\mathrm{ proj-stoch-proc X n) -` {proj-stoch-proc X n w} unfolding psX-def
by auto
    qed
qed
lemma (in infinite-cts-filtration) proj-stoch-singleton-set:
    fixes X::nat }=>\mathrm{ bool stream }=>\mathrm{ ('b::t2-space)
    assumes borel-adapt-stoch-proc F X
    shows (proj-stoch-proc X n) -'{proj-stoch-proc X nw} \in sets (F n)
proof -
    have }\i.i\leqn\Longrightarrow(X i)\in measurable (F n) borel
    by (meson adapt-stoch-proc-def assms increasing-measurable-info)
    have (\bigcapi\in{m.m\leqn}. (X i) -'{Xiw}) \in sets (F n)
    proof ((rule sigma-algebra.countable-INT'\prime), 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 natu-
ral-filtration
                sigma-algebra.sigma-sets-eq sigma-sets-top by fastforce
    have }\bigwedgei.i\leqn\Longrightarrow(Xi)-'{Xiw}\in sets(nat-filtration n)
    proof (rule nat-filtration-borel-measurable-singleton)
            fix }
            assume i\leqn
            show X i borel-measurable (nat-filtration n) using assms natural-filtration
unfolding adapt-stoch-proc-def
                using \langlei}\leqn\rangle\mathrm{ increasing-measurable-info by blast
    qed
    thus }\bigwedgei.i\leqn\Longrightarrow(Xi)-`{Xiw}\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 $f::^{\prime} a \Rightarrow$ ' $b::$ t1-space
assumes finite (range f)
shows $(\lambda w$. snth $w i)-$ (open-exclude-set $(f x)($ range $f)) \in$ sets $($ stream-space borel)
proof -
define opex where opex $=$ open-exclude-set $(f x)$ (range $f$ )
have open opex and opex $\cap$ (range $f)=\{f x\}$ using assms unfolding opex-def by (auto simp add:open-exclude-finite)
hence opex $\in$ sets borel by simp
hence vim: $(\lambda w$. snth $w i)-$ ' opex $\in$ sets (vimage-algebra (streams (space borel))
( $\lambda w$. snth $w i$ ) borel)
by (metis in-vimage-algebra inf-top.right-neutral space-borel streams-UNIV)
have $(\lambda w$. snth $w i)-$ ' opex $\in$ sets $(\square i$. vimage-algebra (streams (space borel))
( $\lambda w$. snth $w i$ ) borel)
proof (rule in-sets-SUP, simp)
show $\bigwedge i . i \in U N I V \Longrightarrow$ space (vimage-algebra (streams (space borel)) ( $\lambda w . w$
!! i) borel) =
UNIV by $\operatorname{simp}$
show $(\lambda w . w!!i)-$ ' opex $\in$ sets (vimage-algebra (streams (space borel)) $(\lambda w$.
$w!!i)$ borel)
using vim by $\operatorname{simp}$
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 $\Rightarrow$ bool stream $\Rightarrow$ ('b::t2-space)
assumes borel-adapt-stoch-proc F X
and $r \in$ range (proj-stoch-proc $X n$ )
shows $\exists A \in$ sets (stream-space borel). range (proj-stoch-proc $X n$ ) $\cap A=\{r\}$
proof -
have $\exists x . r=$ 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 $\bigwedge i . i \leq n \Longrightarrow(X i) \in$ measurable $(F n)$ borel
by (meson adapt-stoch-proc-def assms increasing-measurable-info)
hence fin: $\bigwedge i . i \leq n \Longrightarrow$ finite (range $(X i)$ )
by (metis bernoulli bernoulli-stream-space nat-filtration-vimage-finite natu-ral-filtration streams-UNIV)
show ?thesis
proof
define cand where cand $=\left(\bigcap i \in\{m . m \leq n\}\right.$. $(\lambda w$. snth $w i)-{ }^{\prime}($ open-exclude-set ( $X$ i $x$ ) (range $(X i)))$ )
show cand $\in$ sets (stream-space borel) unfolding cand-def
proof ((rule sigma-algebra.countable-INT ${ }^{\prime \prime}$ ), auto)
show UNIV $\in$ 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\leqn\Longrightarrow(\lambdaw.w!!i) -' open-exclude-set (X i x) (range (X i))
sets (stream-space borel)
    proof -
            fix i
            assume i\leqn
    thus (\lambdaw.w!! i) -' open-exclude-set (X ix) (range (X i)) \in sets (stream-space
borel)
            using fin by (simp add:finite-range-stream-space)
        qed
    qed
    have range (proj-stoch-proc X n) \cap cand ={proj-stoch-proc X n x }
    proof
    have proj-stoch-proc X n x frange (proj-stoch-proc X n) \cap cand
    proof
        show proj-stoch-proc X n x f range (proj-stoch-proc X n) by simp
        show proj-stoch-proc X n x cand unfolding cand-def
        proof
            fix }
            assume i\in{m.m\leqn}
            hence }i\leqn\mathrm{ 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 ... \in open-exclude-set (X i x) (range (X i)) using assms
open-exclude-finite(2)
            by (metis IntE <i\leqn` fin insert-iff rangeI)
            finally have snth (proj-stoch-proc X n x) i \in open-exclude-set (X i x)
(range (X i)) .
            thus proj-stoch-proc X n x \in (\lambdaw.w !! i) -` open-exclude-set (X i x)
(range (X i)) by simp
            qed
            qed
            thus {proj-stoch-proc X n x}\subseteq range (proj-stoch-proc X n) \cap cand by simp
            show range (proj-stoch-proc X n) \cap cand \subseteq{proj-stoch-proc X n x }
            proof
                fix z
            assume z\in range (proj-stoch-proc X n) \cap cand
            hence }\existsy.z=proj-stoch-proc X n y by aut
            from this obtain }y\mathrm{ where z= proj-stoch-proc X n y by auto
        hence proj-stoch-proc X n y \in cand using }<z\in\mathrm{ range (proj-stoch-proc X n)
Cand> by simp
            have }\foralli.i\leqn\longrightarrowXiy=X ix
            proof (intro allI impI)
            fix i
            assume }i\leq
            hence X i y = snth (proj-stoch-proc X n y) i
                by (metis le-antisym not-less proj-stoch-proc-component)
            also have ... \in open-exclude-set (X i x) (range (X i))
                    using <proj-stoch-proc X n y cand〉〈i\leqn` unfolding cand-def by
```

```
simp
            finally have \(X\) i \(y \in\) 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 \(\langle i \leq n\rangle\) empty-iff fin insert-iff rangeI)
            qed
            hence \(\forall i\). 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 \(z \in\{\) proj-stoch-proc \(X n x\}\) using \(\langle z=\) proj-stoch-proc \(X n y\rangle\) by auto
            qed
    qed
    thus range (proj-stoch-proc \(X n\) ) \(\cap\) cand \(=\{r\}\) using \(\langle r=\) proj-stoch-proc \(X\)
\(n x>\) by \(\operatorname{simp}\)
    qed
qed
definition (in infinite-cts-filtration) stream-space-single where
stream-space-single \(X r=(\) if \((\exists U . U \in\) sets (stream-space borel) \() \wedge U \cap(\) range \(X)\)
\(=\{r\})\)
    then SOME \(U . U \in\) sets \((\) stream-space borel \() \wedge U \cap(\) range \(X)=\{r\}\) else \(\})\)
lemma (in infinite-cts-filtration) stream-space-singleI:
    assumes \(\exists U . U \in\) sets (stream-space borel) \(\wedge U \cap(\) range \(X)=\{r\}\)
    shows stream-space-single \(X r \in\) sets (stream-space borel) \(\wedge\) stream-space-single
\(X r \cap(\) range \(X)=\{r\}\)
proof -
    let \(? V=S O M E U . U \in\) sets \((\) stream-space borel \() \wedge U \cap(\) range \(X)=\{r\}\)
    have vprop: ? \(V \in\) sets (stream-space borel) \(\wedge\) ? \(V \cap(\) range \(X)=\{r\}\) using
someI-ex \([\) of \(\lambda U . U \in\) sets \((\) stream-space borel \() \wedge U \cap(\) 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 \(\Rightarrow\) bool stream \(\Rightarrow\) ('b::t2-space)
    assumes borel-adapt-stoch-proc \(F X\)
    and \(r \in\) range (proj-stoch-proc \(X n\) )
shows stream-space-single-set: stream-space-single (proj-stoch-proc \(X n\) ) \(r \in\) sets
(stream-space borel)
and stream-space-single-preimage: stream-space-single (proj-stoch-proc \(X n\) ) \(r \cap\)
range \((\) proj-stoch-proc \(X n)=\{r\}\)
proof -
    have \(\exists A \in\) sets (stream-space borel). range (proj-stoch-proc \(X n\) ) \(\cap A=\{r\}\)
        by (simp add: assms proj-stoch-range-singleton)
    hence \(\exists U . U \in\) sets (stream-space borel) \(\wedge U \cap\) range (proj-stoch-proc \(X n\) ) \(=\)
\(\{r\}\) by auto
    hence a: stream-space-single (proj-stoch-proc \(X n\) ) \(r \in\) sets (stream-space borel)
\(\wedge\)
```

stream-space-single (proj-stoch-proc $X n) r \cap($ 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 $X n$ ) $r \in$ sets (stream-space borel) by simp
show stream-space-single (proj-stoch-proc $X n$ ) $r \cap$ 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 \(y=\{z \in\) range (proj-stoch-proc \(X n\) ). snth \(z i=y\}\)
lemma (in infinite-cts-filtration) comp-proj-i-finite:
    fixes \(X::\) nat \(\Rightarrow\) bool stream \(\Rightarrow ' b::\{t 0\)-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 \(i \leq n\)
    shows \((X i)-{ }^{\prime}\{X i x\}=(\bigcup z \in\) comp-proj-i \(X n i(X i x)\). (proj-stoch-proc \(X\)
\(n)-‘\{z\})\)
proof
    show \(X i-‘\{X i x\} \subseteq(\bigcup z \in\) comp-proj-i \(X n i(X i x)\). proj-stoch-proc \(X n-‘\)
\(\{z\}\) )
    proof
        fix \(w\)
        assume \(w \in X i-‘\{X i x\}\)
        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 \(n w) \in\) comp-proj-i X ni \((X i x)\) unfolding comp-proj-i-def
by \(\operatorname{simp}\)
            moreover have \(w \in\) proj-stoch-proc \(X i-{ }^{\prime}\{\) proj-stoch-proc \(X i w\}\) by simp
            ultimately show \(w \in(\bigcup z \in\) comp-proj-i X n \(i(X i x)\). proj-stoch-proc \(X n-\) -
\(\{z\})\) by \(\operatorname{simp}\)
    qed
    show \((\bigcup z \in\) comp-proj-i \(X n i(X i x)\). proj-stoch-proc \(X n-'\{z\}) \subseteq X i-'\{X\)
\(i x\}\)
    proof -
        have \(\forall z \in\) comp-proj-i \(X n i(X i x)\). proj-stoch-proc \(X n-'\{z\} \subseteq X i-'\{X\)
\(i x\}\)
```

```
    proof
        fix z
        assume z\in comp-proj-i X n i (X i x)
            hence z\in range (proj-stoch-proc X n) and snth zi=X i x unfolding
comp-proj-i-def by auto
            show proj-stoch-proc X n-'{z}\subseteqXi-'{Xix}
            proof
                fix w
            assume w\inproj-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 zi using <w\inproj-stoch-proc X n -` {z}> by auto
            also have ... = X i x using <snth zi=X i x\rangle by simp
            finally have }Xiw=Xi
            thus w\inXi-'{Xix} 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 $\Rightarrow ' b::\{t 0$-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 $(X n)-‘\left\{\begin{array}{ll}X & n\end{array}\right\}=(\bigcup z \in$ comp-proj $X n(X n x)$. (proj-stoch-proc $X n)$
-' $\{z\}$ )
proof
show $X n-‘\{X n x\} \subseteq(\bigcup z \in c o m p-p r o j X n(X n x)$. 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 $\operatorname{simp}$
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 $\operatorname{simp}$
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 $\operatorname{simp}$
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 $\forall z \in$ comp-proj $X n(X n x)$. proj-stoch-proc $X n-'\{z\} \subseteq X n-‘\{X n$ $x\}$
proof

## fix $z$

assume $z \in$ comp-proj $X n\binom{X}{n}$
hence $z \in$ range (proj-stoch-proc $X n$ ) and snth $z n=X n x$ unfolding comp-proj-def by auto

```
            show proj-stoch-proc \(X n-‘\{z\} \subseteq X n-‘\{X n x\}\)
```

            proof
                fix \(w\)
                    assume \(w \in\) proj-stoch-proc \(X n-‘\{z\}\)
            have \(X n w=\operatorname{snth}(\) proj-stoch-proc \(X n w) n\) by (simp add: proj-stoch-proc-component(2))
                also have \(\ldots=\) snth \(z n\) using \(\langle w \in\) proj-stoch-proc \(X n-‘\{z\}\) by auto
                    also have \(\ldots=X n x\) using \(\langle\) snth \(z n=X n x\rangle\) by simp
                    finally have \(X n w=X n x\).
                thus \(w \in X n-‘\{X n x\}\) by \(\operatorname{simp}\)
            qed
    qed
    thus ?thesis by auto
    qed
    qed
lemma comp-proj-stoch-proc-preimage:
shows (proj-stoch-proc $X n)-'\{$ proj-stoch-proc $X n x\}=(\bigcap i \in\{m . m \leq n\} .(X$ i) $-\{\{X i x\})$
proof
show proj-stoch-proc $X n-‘\{$ proj-stoch-proc $X n x\} \subseteq(\bigcap i \in\{m . m \leq n\} . X i$
-' $\{X i x\}$ )
proof
fix $z$
assume $z \in$ 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 $\forall i \leq n . X i z=X i x$ using proj-stoch-proc-component by (metis less-le)
hence $\forall i \leq n . z \in X i-‘\{X i x\}$ by simp
thus $z \in\left(\bigcap i \in\{m . m \leq n\} . X i-{ }^{\prime}\{X i x\}\right)$ by $\operatorname{simp}$
qed
show $(\bigcap i \in\{m . m \leq n\} . X i-‘\{X i x\}) \subseteq$ proj-stoch-proc $X n-‘\{p r o j$-stoch-proc
$X \cap x\}$
proof
fix $z$
assume $z \in(\bigcap i \in\{m . m \leq n\} . X i-'\{X i x\})$
hence $\forall i \leq n$. $X i z=X i x$ by auto
hence $\forall i$. snth (proj-stoch-proc $X n z) i=\operatorname{snth}($ 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 $M) \mid A . A \in$ 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 MXNn)=(sigma-sets (space $M)(\bigcup i \in\{m . m \leq$ $n\} .\{(X i-' A) \cap($ space $M) \mid A . A \in$ 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 $M \mid A . A \in$ sets $N\})$ ) using assms
by (simp add: adapt-sigma-sets)
show sigma-sets (space $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime} A \cap\right.\right.$ space $M \mid A . A \in$ 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 $\left\{X i-{ }^{\prime} A \cap\right.$ space $M \mid A . A \in$ sets $\left.N\right\} \subseteq$ sets $M$ using assms measur-able-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 MXN)XNunfolding 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 $N . X m-{ }^{\prime} y \cap$ space (stoch-proc-filt $M X N m$ ) $\operatorname{sets}$ (stoch-proc-filt MXNm)
proof
fix $B$
assume $B \in$ sets $N$
have $X m-{ }^{\prime} B \cap$ space (stoch-proc-filt $M X N m$ ) $=X m-{ }^{\prime} B \cap$ space $M$ using «space (stoch-proc-filt MXNm)=space $M$ 〉 by simp
also have $\ldots \in(\bigcup i \in\{p . p \leq m\} .\{(X i-' A) \cap($ space $M) \mid A . A \in$ sets $N$ \}) using $\langle B \in$ sets $N\rangle$ by auto
also have $\ldots \subseteq$ sigma-sets (space $M)\left(\bigcup i \in\{p . p \leq m\} .\left\{\left(X i-{ }^{\prime} A\right) \cap(\right.\right.$ space M) | A. $A \in$ sets $N\}$ ) by auto
also have $\ldots=$ sets (stoch-proc-filt $M X N m$ ) using assms stoch-proc-filt-sets by blast
finally show $X m-{ }^{\prime} B \cap$ space (stoch-proc-filt $M X N m$ ) $\operatorname{sets}$ (stoch-proc-filt $M X N m)$.
qed
qed
qed
lemma stoch-proc-filt-disc-filtr:
assumes $\bigwedge i$. $\left(\begin{array}{ll}X & i) \in \text { measurable } M N\end{array}\right.$
shows disc-filtr $M$ (stoch-proc-filt $M X N$ ) unfolding disc-filtr-def
proof (intro conjI allI impI)
\{
fix $n$
show subalgebra $M$ (stoch-proc-filt $M X N n$ ) unfolding subalgebra-def
proof
show space (stoch-proc-filt $M X N n$ ) $=$ space $M$ by (simp add:stoch-proc-filt-space)
show sets (stoch-proc-filt MXNn) $\subseteq$ sets $M$
proof -
have sigma-sets (space $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime} A \cap\right.\right.$ space $M \mid A . A \in$ 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 $\left\{X i-{ }^{\prime} A \cap\right.$ space $M \mid A . A \in$ sets $\left.N\right\} \subseteq$ 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 }
    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 Nn)= 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 MXNn)= space (stoch-proc-filt M X Np)
    show sets (stoch-proc-filt M X N n)\subseteq sets (stoch-proc-filt M X N p)
    proof -
            have sigma-sets (space M)(\bigcupi\in{m.m\leqn}.{Xi-'` A\cap space M |A.A\in
sets N})\subseteq
            sigma-sets (space M)(\bigcupi\in{m.m\leqp}.{Xi-`}A\cap\mathrm{ space M |A.A sets
N})
            proof (rule sigma-sets-mono')
```



```
m\leqp}.{Xi-'A\cap space M |A.A\in sets N})
            proof (rule UN-subset-iff[THEN iffD2], intro ballI)
                fix i
                    assume i\in{m.m\leqn}
                    show {Xi -' A\cap space M |A.A sets N}\subseteq(\bigcupi\in{m.m\leqp}. {Xi
-' }A\cap\mathrm{ space }M|A.A\in\mathrm{ sets N})
                using <i\in{m.m\leqn}>\langlen\leqp\rangle 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=
    and P\subseteqPow (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)(\bigcupx\in{m.m\leqn}. {X x -` A\cap space N |A.A\in
sets P})\subseteqPow (space N)
    proof (rule sigma-algebra.sigma-sets-subset)
    show sigma-algebra (space N) (Pow (space N)) by (simp add: sigma-algebra-Pow)
    show (\bigcupx\in{m.m\leqn}.{Xx-'A\cap space N|A.A\in sets P})\subseteqPow(space
N) by auto
    qed
qed
```

lemma (in infinite-cts-filtration) stoch-proc-filt-triv-init:
fixes $X::$ nat $\Rightarrow$ bool stream $\Rightarrow$ 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 $M)\left(\bigcup i \in\{m . m \leq 0\} .\left\{\left(X^{\prime} i-‘ A\right) \cap(\right.\right.$ space $M) \mid A . A \in$
sets borel\}))
proof (rule stoch-proc-filt-sets)
show $\wedge i . i \leq 0 \Longrightarrow$ random-variable borel $\binom{X}{i}$
proof -
fix $i$ ::nat
assume $i \leq 0$
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 $\perp)=\{\{ \}$, space $M\}$ using nat-info-filtration un-
folding init-triv-filt-def by simp
qed
hence $\exists c . \forall w \in$ space $M .\left(\left(\begin{array}{ll}X & 0 \\ w\end{array}\right)::\right.$ 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: $\forall w \in$ space $M .\left(\begin{array}{lll}X & 0 & w\end{array}\right)=c$ by auto
have $\left(\cup i \in\{m . m \leq 0\} .\left\{\left(X i-{ }^{\prime} A\right) \cap(\right.\right.$ space $M) \mid A . A \in$ sets borel $\left.\}\right)=$
$\{(X 0-' A) \cap($ space $M) \mid A . A \in$ sets borel $\}$ by auto
also have $\ldots=\{\{ \}$, space $M\}$
proof

```
        show {X O -` A\cap space M |A.A\in sets borel }}\subseteq{{},\mathrm{ space M }
        proof -
            have }\forallA\in\mathrm{ sets borel. (X O-'A) }\cap(\mathrm{ space M) }M{{{}\mathrm{ , space M}
            proof
            fix A::real set
            assume A\in sets borel
            show (X O - 'A) \cap (space M) \in{{}, space M}
            proof (cases c\inA)
                case True
                hence X 0 - ' A\cap space M = space M using img by auto
                thus ?thesis by simp
            next
                case False
                hence X O-` A\cap space M={} using img by auto
                thus ?thesis by simp
            qed
            qed
            thus ?thesis by auto
        qed
        show {{}, space M}\subseteq{X0-' A\cap space M |A.A\in sets borel }
        proof -
            have {}\in{X 0 -' A\cap space M |A.A\in sets borel } by blast
            moreover have space M \in{X 0-' A\cap space M |A.A\in sets borel }
            proof -
                        have UNIV\subseteqX O-' 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 (U i\in{m.m\leq0}.{(Xi-'A)\cap(space M)|A.A\in 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 }=>\mathrm{ bool stream }=>\mathrm{ ('b::t2-space)
```

```
    assumes borel-adapt-stoch-proc F X
    and}i\leq
    and A\in sets borel
shows \forally\inA\cap range (Xi). Xi-`{y} = (proj-stoch-proc X n) -` (Uz\in comp-proj-i
Xniy.
    (stream-space-single (proj-stoch-proc X n) z))
proof
    fix }
    assume y\inA\cap range ( }Xi\mathrm{ )
    hence }\existsx.y=X ix by aut
    from this obtain x where y=X ix by auto
    hence Xi-`{y}=Xi-`{Xix} by simp
    also have ... =(\bigcupz\in comp-proj-i X ni (X i x).(proj-stoch-proc X n) -`{z})
    using \langlei\leq n\rangle by (simp add: stoch-proc-comp-proj-i-preimage)
    also have .. = (Uz\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 \forallz\in comp-proj-i X ni (X ix). (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 ccomp-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 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 -`(\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) -'(\bigcupz\incomp-proj-i X n i y.
            (stream-space-single (proj-stoch-proc X n) z)) using « }y=X i x by bimp
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 $X n x\}=(\bigcap i \in\{m . m \leq n\} .(X i)$ - ' $\left\{\begin{array}{l}X \\ i\end{array} \quad x\right\}$ )
by (simp add:comp-proj-stoch-proc-preimage)
also have $\ldots \in$ sigma-sets (space $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime} A \cap\right.\right.$ space $M$ $\mid A . A \in$ sets borel\})
proof -
have inset: $\forall i \leq n .(X i)-‘\{X i x\} \in\left\{X i-{ }^{`} A \cap\right.$ space $M \mid A . A \in$ sets borel $\}$
proof (intro allI impI)
fix $i$
assume $i \leq n$
have $\exists U$. open $U \wedge U \cap($ range $(X i))=\{X i x\}$
proof -
have $\exists U$. open $U \wedge X i x \in U \wedge U \cap(($ range $(X i))-\{X i x\})=\{ \}$
proof (rule open-except-set)
have finite (range $\left(\begin{array}{l}(i)) \text { using assms }\end{array}\right.$
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 $X i x \notin($ range $(X i))-\{X i x\}$ by simp
qed
thus ?thesis using assms by auto
qed
from this obtain $U$ where open $U$ and $U \cap($ range $(X i))=\{X i x\}$ by auto
have $X i-{ }^{\prime}\{X i x\}=X i-{ }^{\prime} U$ using $\left\langle U \cap(\right.$ range $(X i))= \begin{cases}X & i x\}\rangle \text { by }\end{cases}$ auto
also have $\ldots=X i-{ }^{\text {' }} U \cap$ space $M$ using bernoulli bernoulli-stream-space by $\operatorname{simp}$
finally have $X i-{ }^{\prime}\{X i x\}=X i-{ }^{\prime} U \cap$ space $M$.
moreover have $U \in$ sets borel using «open $U$ 〉 by simp
ultimately show $(X i)-‘\{X i x\} \in\left\{X i-{ }^{\prime} A \cap\right.$ space $M \mid A . A \in$ sets borel $\}$
by auto
qed
show ?thesis
proof (rule sigma-set-inter-init)
show $\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime} A \cap\right.\right.$ space $M \mid A . A \in$ sets borel $\left.\}\right) \subseteq$ Pow (space $M$ ) by auto
show $\bigwedge i . i \leq n \Longrightarrow X i-'\{X i x\} \in$ sigma-sets (space $M)(\bigcup i \in\{m . m \leq$ $n\} .\left\{X i-{ }^{\prime} A \cap\right.$ space $M \mid A . A \in$ sets borel $\left.\}\right)$
using inset by (metis (no-types, lifting) UN-I mem-Collect-eq sigma-sets.Basic)
qed
qed
also have $\ldots=$ sets (stoch-proc-filt $M X$ borel $n$ )
proof (rule stoch-proc-filt-sets[symmetric])
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 $X n-$ - $\{$ proj-stoch-proc $X n x\}$ $\in$ sets (stoch-proc-filt M X borel n).
qed
lemma (in infinite-cts-filtration) stoch-proc-filt-gen:
fixes $X:: n a t \Rightarrow$ bool stream $\Rightarrow(' b::$ 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 $\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime} A \cap\right.\right.$ space $M \mid A . A \in$ sets borel $\}$ )
$\subseteq\{$ proj-stoch-proc $X n-' B \cap$ space $M \mid B . B \in$ sets (stream-space borel) $\}$
proof (rule UN-subset-iff[THEN iffD2], intro ballI)
fix $i$
assume $i \in\{m . m \leq n\}$
hence $i \leq n$ by simp
show $\left\{X i-{ }^{\prime} A \cap\right.$ space $M \mid A . A \in$ sets borel $\} \subseteq$
$\left\{\right.$ proj-stoch-proc $X n-{ }^{\prime} B \cap$ space $M \mid B . B \in$ sets (stream-space borel) $\}$
proof -
have $\bigwedge A . A \in$ sets borel $\Longrightarrow X i-‘ A \cap$ space $M \in\{$ proj-stoch-proc $X n-‘$
$B \cap$ space $M \mid B . B \in$ sets (stream-space borel) $\}$
proof -
fix $A::^{\prime} b$ set
assume $A \in$ sets borel
have $X i-' A \cap$ space $M=X i-' A$ using bernoulli bernoulli-stream-space
by $\operatorname{simp}$
also have $\ldots=X i-{ }^{\prime}(A \cap$ range $(X i))$ by auto
also have $\ldots=(\bigcup y \in A \cap$ range $(X i) . X i-‘\{y\})$ by auto
also have $\ldots=\left(\bigcup y \in A \cap\right.$ range $\left(\begin{array}{ll} \\ X & i) \text {. (proj-stoch-proc } X n)-‘ \\ (\bigcup z \in\end{array}\right.$ comp-proj-i X $n$ i $y$.
(stream-space-single (proj-stoch-proc $X n) z))$ ) using stream-space-borel-union assms $\langle i \leq n\rangle\langle A \in$ sets borel $\rangle$
by (metis (mono-tags, lifting) image-cong)
also have $\ldots=($ proj-stoch-proc $X n)-{ }^{\prime}(\bigcup y \in A \cap$ range $(X i) .(\bigcup z \in$ comp-proj-i X $n$ i $y$.
(stream-space-single (proj-stoch-proc $X n$ ) z))) by (simp add: vim-age-Union)
finally have $X i-‘ A \cap$ space $M=($ proj-stoch-proc $X n)-'(\bigcup y \in A \cap$
range $(X i) .(\bigcup z \in$ comp-proj-i $X n i y$.
(stream-space-single (proj-stoch-proc $X n) z))$ ).
moreover have $(\bigcup y \in A \cap$ range $(X i)$. $(\bigcup z \in$ comp-proj-i $X$ n i $y$.
(stream-space-single (proj-stoch-proc $X n) z))) \in$ sets (stream-space borel) proof -
have finite $(A \cap$ 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 \(\forall y \in A \cap\) range \((X i)\). \((\bigcup z \in\) comp-proj-i \(X\) n i y.
    (stream-space-single (proj-stoch-proc \(X n) z)) \in\) sets (stream-space borel)
proof
    fix \(y\)
    assume \(y \in A \cap\) range \((X i)\)
    have finite (comp-proj-i \(X\) n i y) by (simp add: assms comp-proj-i-finite)
moreover have \(\forall z \in\) comp-proj- \(i X n i y\). (stream-space-single (proj-stoch-proc
```

$X n) z) \in$ sets (stream-space borel)
proof
fix $z$
assume $z \in$ comp-proj-i X n i y
thus (stream-space-single (proj-stoch-proc $X n$ ) z) $\in$ sets (stream-space
borel) using assms
stream-space-single-set unfolding comp-proj-i-def by auto
qed
ultimately show ( $\bigcup z \in$ comp-proj-i $X n i y$. (stream-space-single
$($ proj-stoch-proc $X n) z)) \in$
sets (stream-space borel) by blast
qed
ultimately show ?thesis by blast
qed
ultimately show $X i-{ }^{\prime} A \cap$ space $M \in\left\{\right.$ proj-stoch-proc $X n-{ }^{\prime} B \cap$ space
$M \mid B . B \in$ sets (stream-space borel) $\}$
by (metis (mono-tags, lifting) $\langle X i-‘ A \cap$ space $M=X i-‘ A\rangle$
mem-Collect-eq)
qed
thus ?thesis by auto
qed
qed
hence $l$ : sigma-sets (space $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime} A \cap\right.\right.$ space $M \mid A . A \in$
sets borel\}) $\subseteq$
sigma-sets (space $M$ ) \{proj-stoch-proc $X n-{ }^{\prime} B \cap$ space $M \mid B . B \in$ sets
(stream-space borel)\}
by (rule sigma-sets-mono')
have $\left\{\right.$ proj-stoch-proc $X n-{ }^{`} B \cap$ space $M \mid B . B \in$ sets (stream-space borel) $\}$
$\subseteq$ sigma-sets (space $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime} A \cap\right.\right.$ space $M \mid A . A \in$ sets
borel\})
proof -
have $\forall B \in$ sets (stream-space borel). proj-stoch-proc $X n-{ }^{\prime} B \cap$ space $M \in$
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 $B:: ' b$ stream set
assume $B \in$ sets (stream-space borel)
have proj-stoch-proc $X n-' B \cap$ space $M=$ proj-stoch-proc $X n-' B$ using bernoulli bernoulli-stream-space by simp
also have $\ldots=$ proj-stoch-proc $X n-‘(B \cap$ range $($ proj-stoch-proc $X n))$ by auto
also have $\ldots=$ proj-stoch-proc $X n-{ }^{\prime}(\bigcup y \in(B \cap$ range (proj-stoch-proc $X$ $n)$ ). $\{y\}$ ) by $\operatorname{simp}$
also have $\ldots=(\bigcup y \in(B \cap$ 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 $\ldots=$ sigma-sets $($ space $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{\left(X i-{ }^{\prime} A\right) \cap\right.\right.$ $($ 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 $\langle y=$ proj-stoch-proc $X n x\rangle$ by simp
qed
hence $(\bigcup y \in(B \cap$ range (proj-stoch-proc $X n)$ ). proj-stoch-proc $X n-‘\{y\})$ $\epsilon$
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 $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime} A \cap\right.\right.$ space $M \mid A . A \in$ sets borel\}) using eqB by simp
qed

> thus ?thesis by auto
qed
hence sigma-sets (space $M$ ) \{proj-stoch-proc $X n-{ }^{\prime} B \cap$ space $M \mid B . B \in$ sets (stream-space borel)\}
$\subseteq$ sigma-sets $($ space $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{`} A \cap\right.\right.$ space $M \mid A . A \in$ sets borel\}) by (rule sigma-sets-mono)
hence sigma-sets (space $M$ ) \{proj-stoch-proc $X n-{ }^{\prime} B \cap$ space $M \mid B . B \in$ sets (stream-space borel) $\}$
$=$ sigma-sets $($ space $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{`} A \cap\right.\right.$ space $M \mid A . A \in$ 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 sub-algebra-def
proof
show space (stoch-proc-filt MX 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 $n$ ) $\subseteq$ sets (nat-filtration $n$ )
proof -
have $\forall i \leq n .(\forall A \in$ sets borel. $X i-' A \cap$ space $M \in$ sets (nat-filtration n))
proof (intro allI impI)
fix $i$
assume $i \leq n$
have $X i \in$ borel-measurable (nat-filtration $n$ )
by (metis $\langle i \leq n\rangle$ adapt-stoch-proc-def assms filtrationE2 measurable-from-subalg nat-discrete-filtration)
show $\forall A \in$ sets borel. $X i-{ }^{\prime} A \cap$ space $M \in$ sets (nat-filtration $n$ )
proof
fix $A$ ::'a set
assume $A \in$ sets borel
thus $X i-‘ A \cap$ space $M \in$ sets (nat-filtration $n$ ) using $\langle X i \in$ borel-measurable (nat-filtration n)>
by (metis bernoulli bernoulli-stream-space measurable-sets nat-filtration-space streams-UNIV)
qed
qed
hence $(\bigcup i \in\{m . m \leq n\} .\{(X i-' A) \cap($ space $M) \mid A$. A sets borel $\}) \subseteq$ sets (nat-filtration $n$ ) by auto
hence sigma-sets (space $M)(\bigcup i \in\{m . m \leq n\} .\{(X i-‘ A) \cap($ space $M) \mid A$. $A \in$ sets borel $\}) \subseteq$ 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 $\forall t . X t \in$ borel-measurable (nat-filtration $t$ )
then have $f 1: \forall n m$. $X n \in$ borel-measurable $m \vee \neg$ subalgebra $m$ (nat-filtration n)
by (meson measurable-from-subalg)
have $\forall n$. subalgebra $M$ (nat-filtration $n$ )
by (metis infinite-coin-toss-space.nat-filtration-subalgebra infinite-coin-toss-space-axioms)
then show ?thesis
using f1 $\left\langle\bigwedge n X N\right.$. ( $\left.\bigwedge i . i \leq n \Longrightarrow X i \in M \rightarrow_{M} N\right) \Longrightarrow$ sets (stoch-proc-filt $M X N n)=$ sigma-sets $($ space $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-^{\prime} A \cap\right.\right.$ space $M \mid A . A$ $\in$ sets $N\})\rangle\left\langle\right.$ sigma-sets $($ space $M)\left(\bigcup i \in\{m . m \leq n\} .\left\{X i-{ }^{\prime} A \cap\right.\right.$ space $M \mid A . A$ $\in$ sets borel $\}$ ) $\subseteq$ sets ( nat-filtration $n$ ) > by blast

## qed

qed
qed

```
lemma (in infinite-coin-toss-space)
    assumes \(N=\) bernoulli-stream \(q\)
    and \(0 \leq q\)
    and \(q \leq 1\)
    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: \(\{w \in\) space \(M .(\) snth \(w 0)\}=\) pseudo-proj-True (Suc 0) -‘\{True
\#\#sconst True\}
    proof
    show \(\{w \in\) space \(M . w!!0\} \subseteq\) pseudo-proj-True (Suc 0) -' \(\{\) True \#\#sconst
True\}
    proof
            fix \(w\)
            assume \(w \in\{w \in\) 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 \(w \in\) pseudo-proj-True (Suc 0) -' \{True\#\#sconst True \(\}\) by simp
            qed
            show pseudo-proj-True (Suc 0)-'\{True\#\#sconst True \(\} \subseteq\{w \in\) space M. \(w\)
!! 0\(\}\)
            proof
            fix \(w\)
            assume \(w \in\) 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 \(w \in\{w \in\) space \(M . w!!0\}\) using bernoulli bernoulli-stream-space by
```

```
simp
    qed
    qed
    hence natset: {w\in space M. (snth w 0)}\in sets (nat-filtration (Suc 0))
    proof -
    have pseudo-proj-True (Suc 0) - '{True##sconst True} \in 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\in space M. \neg(snth w 0)} = pseudo-proj-True (Suc 0) - '{False
##sconst True}
    proof
        show {w\in space M. \neg(snth w 0)}\subseteq pseudo-proj-True (Suc 0) -' {False
##sconst True}
    proof
            fix w
            assume w}\in{w\in\mathrm{ space M. }\neg(\mathrm{ snth w 0)}
            hence }\neg(\mathrm{ snth w 0) by simp
            hence pseudo-proj-True (Suc 0) w = False ##sconst True
            by (simp add: pseudo-proj-True-def)
            thus w\in pseudo-proj-True (Suc 0) -'{ False ## sconst True} by simp
    qed
    show pseudo-proj-True (Suc 0) -'{False ## sconst True }}\subseteq{w\in\mathrm{ space M.
\neg(snth w 0)}
    proof
            fix }
            assume w\in pseudo-proj-True (Suc 0) -'{False##sconst True}
            hence pseudo-proj-True (Suc 0) w = False##sconst True by simp
            hence }\neg(\mathrm{ snth w 0)
                by (metis pseudo-proj-True-Suc-prefix stream-eq-Stream-iff)
            thus}w\in{w\in\mathrm{ space M. }\neg(\mathrm{ snth w 0)} using bernoulli bernoulli-stream-space
by simp
            qed
    qed
    hence natsetf: {w\in space M. \neg(snth w 0)}\insets (nat-filtration (Suc 0))
    proof -
        have pseudo-proj-True (Suc 0) -'{False##sconst True} \in 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 $\neg 0<q$
hence $q=0$ using assms by simp
hence emeasure $N\{w \in$ space $N .($ snth $w 0)\}=q$ using bernoulli-stream-component-probability $[$ of
$N q]$ assms by blast
hence emeasure $N\{w \in$ space $N$. (snth $w 0)\}=0$ using $\langle q=0\rangle$ by simp
hence emeasure $M\{w \in$ space $M$. (snth $w 0)\}=0$ using assms natset un-
folding filt-equiv-def
by (simp add: 〈space $M=$ space $N \succ$ )
moreover have emeasure $M\{w \in$ space $M .($ snth $w 0)\}=p$ using bernoulli-stream-component-probability $[o$. M p] bernoulli $p-l t-1 \quad p-g t-0$ by blast
ultimately show False using assms by simp
qed
show $q<1$
proof (rule ccontr)
assume $\neg q<1$
hence $q=1$ using assms by simp
hence emeasure $N\{w \in$ space $N . \neg($ snth $w 0)\}=1-q$ using bernoulli-stream-component-probability-compl $[$
$N q]$ assms by blast
hence emeasure $N\{w \in$ space $N . \neg($ snth $w 0)\}=0$ using $\langle q=1\rangle$ by simp
hence emeasure $M\{w \in$ space $M . \neg($ snth $w 0)\}=0$ using assms natsetf
unfolding filt-equiv-def
by (simp add: <space $M=$ space $N 〉$ )
moreover have emeasure $M\{w \in$ space $M . \neg($ snth $w 0)\}=1-p$ using
bernoulli-stream-component-probability-compl[of $M$ p] bernoulli $p-l t-1$ 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 MfPn=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 $M)(\bigcup n \in\{n a . n a \leq n\}$. $\{f n$
-' $C \cap$ space $M \mid C . C \in$ sets $P\}))=$
gen-subalgebra $N$ (sigma-sets (space $N)\left(\bigcup n \in\{n a . n a \leq n\} .\left\{f n-{ }^{\prime} C \cap\right.\right.$ space
$N \mid C . C \in$ 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 \(N\) (Gt) using assms unfolding filtration-def filt-equiv-def
            by (metis sets-eq-imp-space-eq subalgebra-def)
    \}
    \{
        fix \(s::^{\prime} c\)
        fix \(t\)
        assume \(s \leq t\)
        thus subalgebra ( \(G t\) ) ( \(G\) s) using assms unfolding filtration-def by simp
    \}
qed
lemma filt-equiv-borel-AE-eq-iff:
    fixes \(f::^{\prime} a \Rightarrow\) real
    assumes filt-equiv \(F M N\)
and \(f \in\) borel-measurable ( \(F t\) )
and \(g \in\) borel-measurable ( \(F t\) )
and prob-space \(N\)
and prob-space \(M\)
shows \((A E w\) in \(M . f w=g w) \longleftrightarrow(A E w\) in \(N . f w=g w)\)
proof -
    \{
    assume \(f s t: A E w\) in \(M . f w=g w\)
    have set \(0:\{w \in\) space \(M . f w \neq g w\} \in \operatorname{sets}(F t) \wedge\) 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 MF using assms unfolding filt-equiv-def
                by (simp add: filtrated-prob-space-axioms.intro filtrated-prob-space-def)
            show \(A E w\) in \(M . f w=g w\) using \(f s t\).
    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 have \(A E w\) in \(N\). \(f w=g w\)
    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 \(M . f w \neq g w\}=0\rangle\langle\{w\)
```

```
\epsilon space M. fw\not=gw}\in sets N>
            by auto
            then show ?thesis
                by metis
    qed
    } note a= this
    {
    assume scd:AE w in N.fw=gw
    have space M = space N
        by (metis assms(1) filt-equiv-space)
    have set0: {w\in space N.fw\not=gw}\in sets (Ft)\wedge emeasure N {w\in space N.
fw\not=gw} = 0
    proof (rule filtrated-prob-space.AE-borel-eq, (auto simp add: assms))
        show filtrated-prob-space NF using assms unfolding fil-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.fw=gw using scd.
    qed
    hence emeasure M {w\in space M.fw\not=g w}=0 using assms unfolding
fil-equiv-def
            by (metis (full-types) assms(1) filt-equiv-space)
    moreover have {w\in space M.fw\not=gw}\in sets M using set0 assms unfolding
fil-equiv-def
            filtration-def subalgebra-def
            by (metis (mono-tags) <space M = space N`contra-subsetD)
    ultimately have AE w in M.fw=gw
    proof -
            have }\forallp\mathrm{ . almost-everywhere M p}\vee{a\in space M.\negpa}\not={a\in space M.
f a\not=ga}
            using AE-iff-measurable <emeasure M {w\in space M.fw\not=gw}=0\rangle\langle{w
\in space M. fw\not=gw}\in 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 FMN
and init-triv-filt M G
shows init-triv-filt N G unfolding init-triv-filt-def
proof
    show filtration NG using assms filt-equiv-filt[of F M NG] unfolding init-triv-filt-def
by simp
    show sets (G\perp) = {{}, space N} using assms filt-equiv-space[of F M N] un-
folding init-triv-filt-def by simp
qed
```

```
lemma (in infinite-coin-toss-space) fct-gen-subalgebra-meas-info:
    assumes \(\forall w . f(g w)=f w\)
    and \(f \in\) space \(M \rightarrow\) space \(N\)
and \(g \in\) space \(M \rightarrow\) 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 \((f c t-g e n-s u b a l g e b r a M N f)\)
using assms
    by (simp add: fct-gen-subalgebra-space)
    show \(\forall y \in\) sets (fct-gen-subalgebra \(M N f\) ). \(g-{ }^{\prime} y \cap\) space (fct-gen-subalgebra \(M\)
\(N f) \in\) sets (fct-gen-subalgebra MNf)
    proof
            fix \(B\)
            assume \(B \in\) sets (fct-gen-subalgebra \(M N f\) )
            hence \(B \in\left\{f-{ }^{\prime} B \cap\right.\) space \(M \mid B\). \(B \in\) sets \(\left.N\right\}\) 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 \(g-{ }^{\prime} B \cap\) space (fct-gen-subalgebra \(M N f\) ) \(=g-{ }^{\prime} B \cap\) space \(M\) using
assms
            by (simp add: fct-gen-subalgebra-space)
            also have \(\ldots=g-‘\left(f-{ }^{\prime} C \cap\right.\) space \(\left.M\right) \cap\) space \(M\) using Cprops by simp
            also have \(\ldots=g-\) ' \((f-‘ C)\) using bernoulli bernoulli-stream-space by simp
            also have \(\ldots=(f \circ g)-{ }^{\prime} C\) by auto
            also have \(\ldots=f-{ }^{\prime} C\)
    proof
            show \((f \circ g)-{ }^{'} C \subseteq f-{ }^{\prime} C\)
            proof
                fix \(w\)
                    assume \(w \in(f \circ g)-{ }^{\prime} C\)
            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 \(f-{ }^{\prime} C \subseteq(f \circ g)-{ }^{\prime} C\)
                proof
                    fix \(w\)
                    assume \(w \in f-' C\)
                    hence \(f w \in C\) by simp
                    hence \(f(g w) \in C\) using assms by \(\operatorname{simp}\)
            thus \(w \in(f \circ g)-{ }^{‘} C\) by simp
                qed
    qed
    also have ... \(\in\) sets (fct-gen-subalgebra \(M N f\) )
    using Cprops(2) \(\left\langle B \in\right.\) sets (fct-gen-subalgebra MNf) \({ }^{(2)}\) bernoulli bernoulli-stream-space
```

inf-top.right-neutral by auto
finally show $g-' B \cap$ space $(f c t-g e n-s u b a l g e b r a M N f) \in$ sets $(f c t-g e n-s u b a l g e b r a$ $M N f)$.
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 $\Rightarrow$ real $\Rightarrow$ real $\Rightarrow$ (nat $\Rightarrow$ bool stream $\Rightarrow$ real) where
base: (geom-rand-walk udv) $0=(\lambda w . v) \mid$
step: (geom-rand-walk udv) (Suc n) $=(\lambda w .((\lambda$ True $\Rightarrow u \mid$ False $\Rightarrow d)($ snth $w$ $n)) *($ geom-rand-walk $u d v) n w)$
locale prob-grw $=$ infinite-coin-toss-space +
fixes geom-proc::nat $\Rightarrow$ bool stream $\Rightarrow$ real and $u::$ real and $d::$ real and init::real
assumes geometric-process:geom-proc $=$ geom-rand-walk ud init
lemma (in prob-grw) geom-rand-walk-borel-measurable:
shows (geom-proc $n$ ) $\in$ borel-measurable $M$
proof (induct $n$ )
case (Suc n)
thus geom-proc (Suc $n$ ) $\in$ borel-measurable $M$
proof -
have geom-rand-walk $u$ d init $n \in$ borel-measurable $M$ using Suc geomet-ric-process by simp
moreover have $(\lambda w .((\lambda$ True $\Rightarrow u \mid$ False $\Rightarrow d)($ snth $w n))) \in$ borel-measurable M
proof -
have $(\lambda w$. snth $w n) \in$ measurable $M$ ( measure-pmf (bernoulli-pmf $p)$ ) by (simp add: bernoulli measurable-snth-count-space)
moreover have $(\lambda$ True $\Rightarrow u \mid$ False $\Rightarrow d) \in$ 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 $n \circ$ pseudo-proj-True $n$
proof (induct $n$ )
case (Suc n)
let ?tf $=(\lambda$ True $\Rightarrow u \mid$ False $\Rightarrow d)$
\{
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 $\ldots=$ ?tf (snth (pseudo-proj-True (Suc n) w) n) * geom-proc $n w$ by (metis lessI pseudo-proj-True-stake stake-nth) also have $\ldots=$ ?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 $\ldots=$ ? 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 $\ldots=$ geom-proc (Suc n) (pseudo-proj-True (Suc n) w) using geo-
metric-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 $n) \circ($ 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 $n \circ$ pseudo-proj-False $n$
proof (induct $n$ )
case (Suc n)
let ?tf $=(\lambda$ True $\Rightarrow u \mid$ False $\Rightarrow d)$
\{
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 $\ldots=$ ? tf (snth (pseudo-proj-False (Suc n) w) n) * geom-proc n w by (metis lessI pseudo-proj-False-stake stake-nth) also have $\ldots=$ ? 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 $\ldots=$ ? 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 geo-
metric-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 $n \in$ borel-measurable (nat-filtration $n$ )
proof -
have geom-proc $n \in$ borel-measurable (nat-filtration $n$ )
proof (rule nat-filtration-comp-measurable)
show geom-proc $n \in$ 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 $n$ ) -' $\{x\} \neq\{ \}$
shows (geom-proc (Suc n))' ((geom-proc $n)-‘\{x\})=\{u * x, d * x\}$
proof
have $\exists w$. 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 wn False
    have bel: (?wT\in(geom-proc n) -'{x})}\wedge(?wF\in(\mathrm{ 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=\mathrm{ geom-rand-walk ud 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 ud 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}\subseteq(geom-proc (Suc n))'((geom-proc n) -'{x})
using bel
    by (metis empty-subsetI insert-subset rev-image-eqI)
    have }\forallw\in(\mathrm{ geom-proc n) -' {x}.geom-proc (Suc n) w 
    proof
        fix w
        assume w\in(geom-proc n) -' {x}
        have dis: ((snth w (Suc n)) = True) \vee (snth w (Suc n)= False) by simp
        show geom-proc (Suc n) w\in{u*x,d*x}
        proof -
            have geom-proc n w = x
                by (metis }\langlew\in\mathrm{ geom-proc n -` {x}> vimage-singleton-eq)
            then have geom-rand-walk ud 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})\subseteq{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 }\foralln\mathrm{ n. O< 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 wn)
    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 $\bigwedge w$. (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 wn True) $=u *$ geom-proc $n w$
proof -
have snth (spick w $n$ True) $n=$ True by (simp add: spickI)
hence $(\lambda w$. (case $w!!n$ of True $\Rightarrow u \mid$ False $\Rightarrow d))($ spick $w n$ True $)=u$ by
simp
thus ?thesis using geometric-process geom-rand-walk.simps(2)[of udinit 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 $(\lambda w$. (case $w!!n$ of True $\Rightarrow u \mid$ False $\Rightarrow d))($ 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 riskneutral 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

```
lemma (in prob-space) finite-borel-measurable-integrable:
    assumes f\in borel-measurable M
    and finite (f`(space M))
    shows integrable Mf
proof -
    have simple-function Mf using assms by (simp add: simple-function-borel-measurable)
    moreover have emeasure M {y\in space M.fy\not=0}\not=\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
```


### 7.1.1 On the almost everywhere filter

lemma $A E$-eq-trans[trans]:
assumes $A E x$ in $M . A x=B x$
and $A E x$ in $M . B x=C x$
shows $A E x$ in $M . A x=C x$
using assms by auto
abbreviation $A E e q$ where $A E e q M X Y A E w$ in $M . X w=Y w$
lemma $A E-a d d$ :
assumes $A E$ win $M . f w=g w$
and $A E w$ in $M . f^{\prime} w=g^{\prime} w$
shows $A E w$ in $M . f w+f^{\prime} w=g w+g^{\prime} w$ using assms by auto

```
lemma \(A E\)-sum:
    assumes finite \(I\)
    and \(\forall i \in I\). AE \(w\) in M. \(f i w=g i w\)
    shows \(A E w\) in \(M .\left(\sum i \in I . f i w\right)=\left(\sum i \in I . g i w\right)\) 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 \(A E w\) in \(M .\left(\sum i \in\right.\) insert a \(\left.F . f i w\right)=f a w+\left(\sum i \in F . f i w\right)\)
        by (simp add: insert.hyps(1) insert.hyps(3))
    also have \(A E w\) in \(M . f a w+\left(\sum i \in F . f i w\right)=g a w+\left(\sum i \in F\right.\). fiw)
        using \(\langle A E e q M(f a)(g a)\rangle\) by auto
    also have \(A E w\) in \(M . g a w+\left(\sum i \in F . f i w\right)=g a w+\left(\sum i \in F . g i w\right)\)
        using insert.hyps(4) by auto
    also have \(A E w\) in \(M . g a w+\left(\sum i \in F . g i w\right)=\left(\sum i \in\right.\) insert a \(\left.F . g i w\right)\)
        by (simp add: insert.hyps(1) insert.hyps(3))
    finally show? ?case by auto
qed
```

lemma $A E$-eq-cst:
assumes $A E w$ in $M .(\lambda w, c) w=(\lambda w, d) w$
and emeasure $M($ space $M) \neq 0$
shows $c=d$
proof (rule ccontr)
assume $c \neq d$
from $\langle A E$ win $M .(\lambda w, c) w=(\lambda w, d) w\rangle$ obtain $N$ where Nprops: $\{w \in$ space
$M . \neg(\lambda w . c) w=(\lambda w . d) w\} \subseteq N N \in$ sets $M$ emeasure $M N=0$
by (force elim: $A E-E$ )
have $\forall w \in$ space $M$. $(\lambda w . c) w \neq(\lambda w . d) w$ using $\langle c \neq d\rangle$ by simp
hence $\{w \in$ space $M .(\lambda w . c) w \neq(\lambda w . d) w\}=$ space $M$ by auto
hence space $M \subseteq N$ using Nprops by auto
thus False using «emeasure $M N=0$ 〉 assms
by (meson $\operatorname{Nprops}(2)$ (emeasure $M($ space $M) \neq 0\rangle\langle e m e a s u r e ~ M N=0\rangle\langle$ space
$M \subseteq N\rangle$ 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 $M$ 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 \(A E x\) in \(M\). real-cond-exp \(M N f x=(\lambda x\). 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 \(M(\lambda x\). expectation \(f)\) by auto
    show \((\lambda\) x. expectation \(f) \in\) borel-measurable \(N\) by simp
    show \(\bigwedge A . A \in\) sets \(N \Longrightarrow\) set-lebesgue-integral \(M A f=\int x \in A\). expectation \(f \partial M\)
    proof -
    fix \(A\)
    assume \(A \in\) sets \(N\)
    show set-lebesgue-integral \(M A f=\int x \in A\). expectation \(f \partial M\)
    proof (cases \(A=\{ \}\) )
        case True
        thus ?thesis by (simp add: set-lebesgue-integral-def)
    next
        case False
        hence \(A=\) space \(M\) using assms \(\langle A \in\) sets \(N\rangle\) by auto
        have set-lebesgue-integral \(M A f=\) expectation \(f\) using \(\langle A=\) space \(M\rangle\)
                by (metis (mono-tags, lifting) Bochner-Integration.integral-cong indica-
tor-simps(1)
                    scaleR-one set-lebesgue-integral-def)
            also have \(\ldots=\int x \in A\). expectation \(f \partial M\) using \(\langle A=\) space \(M\rangle\)
                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 \(A E x\) in \(M . f x=(c:: ' b::\{t 2-s p a c e\})\)
    and \(f \in\) borel-measurable \(F\)
shows \(\forall x \in\) space \(M . f x=c\)
proof
    fix \(x\)
    assume \(x \in\) space \(M\)
    have space \(M=\) space \(F\) using assms by (simp add:subalgebra-def)
    hence \(x \in\) space \(F\) using \(\langle x \in\) space \(M\rangle\) by simp
    have space \(M \neq\{ \}\) by (simp add:not-empty)
```

hence $\exists d . \forall y \in$ space $F . f y=d$ by (metis assms(1) assms(2) assms(4) subal-gebra-def triv-measurable-cst)
from this obtain $d$ where $\forall y \in$ space $F . f y=d$ by auto
hence $f x=d$ using $\langle x \in$ space $F\rangle$ by simp
also have $\ldots=c$
proof (rule ccontr)
assume $d \neq c$
from $\langle A E x$ in $M . f x=c\rangle$ obtain $N$ where Nprops: $\{x \in$ space $M . \neg f x=c\}$ $\subseteq N N \in$ sets $M$ emeasure $M N=0$
by (force elim:AE-E)
have space $M \subseteq\{x \in$ space $M$. $\neg f x=c\}$ using $\langle\forall y \in$ space $F . f y=d\rangle\langle$ space $M=$ space $F\rangle\langle d \neq c\rangle$ by auto
hence space $M \subseteq N$ using Nprops by auto
thus False using <emeasure $M N=0$ 〉emeasure-space-1 $\quad$ Nprops(2) emea-sure-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 $\forall x \in$ space $M$. real-cond-exp $M N f x=\operatorname{expectation~} f$
proof (rule triv-subalg-borel-eq)
show subalgebra $M N$ sets $N=\{\{ \}$, space $M\}$ using assms by auto
show real-cond-exp $M N f \in$ borel-measurable $N$ by simp
show $A E x$ in $M$. real-cond-exp $M N f x=$ expectation $f$
by (rule trivial-subalg-cond-expect- $A E$, (auto simp add:assms))
qed
lemma (in sigma-finite-subalgebra) real-cond-exp-cong':
assumes $\forall w \in$ space M. $f w=g w$
and $f \in$ borel-measurable $M$
shows $A E$ win M. real-cond-exp $M F f w=$ real-cond-exp $M F g w$
proof (rule real-cond-exp-cong)
show $A E w$ in $M$. $f w=g w$ using assms by simp
show $f \in$ borel-measurable $M$ using assms by simp
show $g \in$ borel-measurable $M$ using assms by (metis measurable-cong)
qed
lemma (in sigma-finite-subalgebra) real-cond-exp-bsum :
fixes $f:: ' b \Rightarrow{ }^{\prime} a \Rightarrow$ real
assumes [measurable]: $\bigwedge i . i \in I \Longrightarrow$ integrable $M(f i)$
shows $A E x$ in $M$. real-cond-exp $M F\left(\lambda x . \sum i \in I\right.$. fix) $x=\left(\sum i \in I\right.$. real-cond-exp

MF(fi) $x)$
proof (rule real-cond-exp-charact)
fix $A$ assume [measurable]: $A \in$ sets $F$
then have $A$-meas [measurable]: $A \in$ sets $M$ by (meson subsetD subalg subalge-bra-def)
have $*: ~ \bigwedge i . i \in I \Longrightarrow$ integrable $M(\lambda x$. indicator $A x * f i x)$
using integrable-mult-indicator $[O F\langle A \in$ sets $M\rangle \operatorname{assms}(1)]$ by auto
have $* *: \bigwedge i . i \in I \Longrightarrow$ integrable $M$ ( $\lambda$ x. indicator $A x *$ real-cond-exp $M F(f$
i) $x$ )
using integrable-mult-indicator[OF $\langle A \in$ sets $M\rangle$ real-cond-exp-int(1)[OF assms(1)]]
by auto
have inti: $\bigwedge i . i \in I \Longrightarrow\left(\int x\right.$. indicator $\left.A x * f i x \partial M\right)=\left(\int x\right.$. indicator $A x *$
real-cond-exp MF (fi) x $\partial M$ ) using
real-cond-exp-intg(2)[symmetric,of indicator $A$ ]
using $*\langle A \in$ sets $F\rangle$ assms borel-measurable-indicator by blast
have $\left(\int x \in A .\left(\sum i \in I . f i x\right) \partial M\right)=\left(\int x .\left(\sum i \in I\right.\right.$. indicator $\left.\left.A x * f i x\right) \partial M\right)$
by (simp add: sum-distrib-left set-lebesgue-integral-def)
also have $\ldots=\left(\sum i \in I\right.$. ( $\int x$. indicator $\left.A x * f i x \partial M\right)$ ) using Bochner-Integration.integral-sum [of
$I M \lambda i x$. indicator $A x * f i x] *$
by $\operatorname{simp}$
also have $\ldots=\left(\sum i \in I .\left(\int x\right.\right.$. indicator $A x *$ real-cond-exp $\left.\left.M F(f i) x \partial M\right)\right)$
using inti by auto
also have $\ldots=\left(\int x\right.$. $\left(\sum i \in I\right.$. indicator $A x *$ real-cond-exp $\left.\left.M F(f i) x\right) \partial M\right)$
by (rule Bochner-Integration.integral-sum[symmetric], simp add: **)
also have $\ldots=\left(\int x \in A .\left(\sum i \in I\right.\right.$. real-cond-exp $\left.\left.M F(f i) x\right) \partial M\right)$
by (simp add: sum-distrib-left set-lebesgue-integral-def)
finally show $\left(\int x \in A .\left(\sum i \in I . f i x\right) \partial M\right)=\left(\int x \in A .\left(\sum i \in I\right.\right.$. real-cond-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 $\Rightarrow$ bool where
stk-strict-subs $S \longleftrightarrow S \neq U N I V$
typedef ('a,'c) discrete-market $=\left\{\left(s::\left({ }^{\prime} c\right.\right.\right.$ set $), a::^{\prime} c \Rightarrow\left(n a t \Rightarrow{ }^{\prime} a \Rightarrow\right.$ real $)$ ). stk-strict-subs s\} unfolding stk-strict-subs-def by fastforce
definition prices where
prices $M k t=($ snd $($ Rep-discrete-market $M k t))$
definition assets where

$$
\text { assets } M k t=U N I V
$$

definition stocks where

$$
\text { stocks Mkt }=(\text { fst }(\text { Rep-discrete-market Mkt }))
$$

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 $f c t:(($ if (stk-strict-subs $S)$ then $S$ else $\{ \}), A) \in\{(s, a)$. stk-strict-subs s $\}$ by $\operatorname{simp}$
have discrete-market-of $S A=A b s$-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 $\operatorname{simp}$
thus ?thesis unfolding prices-def by simp
qed
lemma stocks-of:
assumes $U N I V \neq S$
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 $\}), A) \in\{(s, a)$. stk-strict-subs s\} by $\operatorname{simp}$
have discrete-market-of $S A=A b s$-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 $=(\lambda(x:: ' a)(n::$ nat $)$ w. $0::$ real $)$
definition $q t y$-single where
qty-single asset qt-proc $=($ qty-empty $($ asset $:=q t-p r o c))$
definition $q$ ty-sum: : ('b $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a \Rightarrow$ real $) \Rightarrow\left({ }^{\prime} b \Rightarrow\right.$ nat $\Rightarrow{ }^{\prime} a \Rightarrow$ real $) \Rightarrow\left({ }^{\prime} b \Rightarrow\right.$ nat $\Rightarrow$ ' $a \Rightarrow$ real) where
$q t y$-sum pf1 pf2 $=(\lambda x n w . p f 1 x n w+p f 2 x n w)$
definition qty-mult-comp ::('b $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a \Rightarrow$ real $) \Rightarrow\left(n a t \Rightarrow{ }^{\prime} a \Rightarrow\right.$ real $) \Rightarrow\left({ }^{\prime} b \Rightarrow\right.$ nat $\Rightarrow{ }^{\prime} a \Rightarrow$ real) where

$$
q t y-m u l t-c o m p ~ p f 1 ~ q t y=(\lambda x n w \cdot(p f 1 x n w) *(q t y n w))
$$

definition qty-rem-comp $::\left({ }^{\prime} b \Rightarrow\right.$ nat $\Rightarrow{ }^{\prime} a \Rightarrow$ real $) \Rightarrow{ }^{\prime} b \Rightarrow\left(^{\prime} b \Rightarrow\right.$ nat $\Rightarrow{ }^{\prime} a \Rightarrow$ real) where
qty-rem-comp pf1 $x=p f 1(x:=(\lambda n w .0))$
definition $q$ ty-replace-comp where
qty-replace-comp pf1 x pf2 $=$ qty-sum $(q t y-r e m-c o m p ~ p f 1 x)(q t y-m u l t-c o m p ~ p f 2 ~$ (pf1 x))

Support sets If pxnw is different from 0 , this means that this quantity is held on interval $] \mathrm{n}-1, \mathrm{n}]$.

```
definition support-set::( \(b \Rightarrow\) nat \(\Rightarrow{ }^{\prime} a \Rightarrow\) real \() \Rightarrow\) ' \(b\) set where
    support-set \(p=\{x . \exists n w . p x n w \neq 0\}\)
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) \(\subseteq(\) support-set pf1) \(\cup\) (support-set pf2)
proof (intro subsetI, rule ccontr)
    fix \(x\)
    assume \(x \in\) support-set (qty-sum pf1 pf2) and \(x \notin\) support-set pf1 \(\cup\) support-set
pf2 note xprops \(=\) this
    hence \(\exists n w\). (qty-sum pf1 pf2) \(x n w \neq 0\) by (simp add: support-set-def)
    from this obtain \(n w\) where (qty-sum pf1 pf2) \(x n w \neq 0\) by auto note nwprops
\(=\) this
    have \(p f 1 x n w=0\) 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) \subseteq(support-set pf1)
proof (intro subsetI, rule ccontr)
    fix }
    assume x\in support-set (qty-mult-comp pf1 qty) and x & support-set pf1 note
xprops = this
    hence }\existsnw.(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\mathrm{ 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)\subseteq((support-set pf1) - {x})
proof (intro subsetI, rule ccontr)
    fix }
    assume y\in support-set (qty-rem-comp pf1 x) and y & support-set pf1 - {x}
note xprops= this
    hence y\not\in support-set pf1 \vee y = x by simp
    have \exists 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\not=0 by auto note
nwprops = this
    show False
    proof (cases y }\not=\mathrm{ 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\mathrm{ using «y# support-set pf1 }\veey=x\rangle by sim
        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\rangle\mathrm{ )
    qed
qed
lemma replace-comp-support-set:
    shows support-set (qty-replace-comp pf1 x pf2) \subseteq(support-set pf1 - {x}) \cup
support-set pf2
proof -
    have support-set (qty-replace-comp pf1 x pf2) \subseteq support-set (qty-rem-comp pf1
x)\cup support-set (qty-mult-comp pf2 (pf1 x))
    unfolding qty-replace-comp-def by (simp add:sum-support-set)
    also have ...\subseteq(support-set pf1 - {x})\cup 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) \subseteq{asset}
proof
    fix }
    assume x\in support-set (qty-single asset qty)
    show }x\in{\mathrm{ asset}
    proof (rule ccontr)
        assume x\not\in{asset}
    hence x\not= asset by auto
    have \exists n w. qty-single asset qty x n w}=0\mathrm{ 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\mathrm{ by auto
    thus False using <x\not=asset> by (simp add: qty-single-def qty-empty-def)
    qed
qed
lemma single-comp-nz-support:
    assumes \exists n w. qty n w\not=0
    shows support-set (qty-single asset qty) = {asset}
proof
    show support-set (qty-single asset qty)\subseteq {asset} by (simp add: single-comp-support)
    have asset\in support-set (qty-single asset qty) using assms unfolding sup-
port-set-def qty-single-def by simp
    thus {asset}}\subseteq\mathrm{ support-set (qty-single asset qty) by auto
qed
Portfolios definition portfolio where
    portfolio p \longleftrightarrow finite (support-set p)
```

definition stock-portfolio :: ('a, 'b) discrete-market $\Rightarrow\left(' b \Rightarrow n a t \Rightarrow{ }^{\prime} a \Rightarrow\right.$ real $) \Rightarrow$ bool where
stock-portfolio Mkt $p \longleftrightarrow$ portfolio $p \wedge$ support-set $p \subseteq$ 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) $\subseteq($ support-set pf1) $\cup$ (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) $\subseteq($ support-set pf1) by (simp add: mult-comp-support-set)
thus finite (support-set (qty-mult-comp pf1 qty)) using assms unfolding port-
folio-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 $x$ ) $\subseteq$ (support-set pf1) using remove-comp-support-set [of $p f 1 x]$ by blast
thus finite (support-set (qty-rem-comp pf1 x)) using assms unfolding portfo-
lio-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) \subseteq(support-set pf1) \cup (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\subseteq stocks Mkt \cup{x}
    and support-set pf2 \subseteq stocks Mkt
shows support-set (qty-replace-comp pf1 x pf2) \subseteq stocks Mkt
proof -
    have support-set (qty-rem-comp pf1 x)\subseteq stocks Mkt using assms(1) remove-comp-support-set
by fastforce
    moreover have support-set (qty-mult-comp pf2 (pf1 x))\subseteq 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 $(\neg($ portfolio $p))$ then $(\lambda n w .0)$
else $(\lambda n w .(\operatorname{sum}(\lambda x .(($ prices Mkt) $x n w) *(p x(S u c n) w))($ support-set
p))))
lemma subset-val-process':
assumes finite $A$
and support-set $p \subseteq A$
shows val-process Mkt $p n w=(\operatorname{sum}(\lambda x .(($ prices $M k t) x n w) *(p x($ Suc $n) w))$
A)
proof -
have portfolio $p$ using assms unfolding portfolio-def using finite-subset by
auto
have $\exists C$. (support-set $p) \cap C=\{ \} \wedge($ support-set $p) \cup C=A$ using assms(2)
by auto
from this obtain $C$ where (support-set $p) \cap C=\{ \}$ and (support-set $p$ ) $\cup C$
$=A$ by auto note Cprops $=$ this
have finite $C$ using assms «(support-set $p) \cup C=A$ b by auto
have $\forall x \in C$. px (Suc n) $w=0$ using Cprops(1) support-set-def by fastforce

hence val-process Mkt p $n w=\left(\sum x \in(\right.$ support-set $p) .(($ prices Mkt) $x n w) *(p$
$x($ Suc $n) w))$
$+\left(\sum x \in C .((\right.$ prices $\left.M k t) x n w) *(p x(S u c n) w)\right)$ unfolding val－process－def using 〈portfolio $p\rangle$ by simp

using 〈portfolio $p\rangle\langle$ finite $C$ 〉Cprops portfolio－def sum－union－disjoint＇by（metis （no－types，lifting））
finally show val－process Mkt p $n w=\left(\sum x \in A .((\right.$ prices $M k t) x n w) *(p x$ （Suc n）w））．
qed
lemma sum－val－process：
assumes portfolio pf1
and portfolio pf2
shows $\forall n w$ ．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 $n w=\left(\sum x \in(\right.$ support－set pf1 $) \cup$（support－set pf2）． $(($ prices Mkt）$x n w) *(p f 1 x($ Suc $n) w))$
proof－
have finite（support－set pf1 $\cup$ support－set pf2）$\wedge$ support－set pf1 $\subseteq$ support－set pf1 $\cup$ 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 $n w=\left(\sum x \in(\right.$ support－set pf1）$\cup$（support－set pf2）．
$(($ prices Mkt）$x n w) *(p f 2 x($ Suc $n) w))$
proof－
have finite（support－set pf1 $\cup$ support－set pf2）$\wedge$ support－set pf2 $\subseteq$ support－set pf2 $\cup$ 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 $p f$ ：portfolio（qty－sum pf1 pf2）using assms by（simp add：sum－portfolio）
have fin：finite（support－set pf1 $\cup$ support－set pf2）using assms finite－Un un－
folding portfolio－def by auto
have（val－process Mkt pf1）nw（val－process Mkt pf2）$n w=$
（ $\sum x \in($ support－set pf1 $) \cup$（support－set pf2）．（（prices Mkt）x $\left.n w\right) *(p f 1 x$（Suc
n）$w$ ）+
（ $\sum x \in($ support－set pf1 $) \cup($ support－set pf2）．（（prices Mkt）$x n w) *(p f 2 x(S u c$ n）$w$ ）
using $v p 1$ vp2 by simp
also have $\ldots=\left(\sum x \in(\right.$ support－set pf1 $) \cup($ support－set pf2）．
$((($ prices Mkt）x $n w) *(p f 1 x($ Suc $n) w))+(($ prices Mkt）$x n w) *(p f 2 x($ Suc n）$w$ ）
by（simp add：sum．distrib）
also have $\ldots=\left(\sum x \in(\right.$ support－set pf1 $) \cup($ 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 $\ldots=\left(\sum x \in(\right.$ support-set pf1 $) \cup($ support-set pf2 $)$.
$(($ prices $M k t) x n w) *((q t y-s u m p f 1$ pf2) $x($ Suc $n) w))$ by (simp add: qty-sum-def)
also have $\ldots=\left(\sum x \in\right.$ (support-set (qty-sum pf1 pf2)).
$(($ prices Mkt) x $n \mathrm{w}) *((q t y-$ sum pf1 pf2 $) x($ Suc $n) w))$ using sum-support-set $[o f$ pf1 pf2]
subset-val-process'[of support-set pf1 $\cup$ support-set pf2 qty-sum pf1 pf2] pf fin unfolding val-process-def by simp
also have $\ldots=$ val-process $M k t$ (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 $\forall n w$. val-process Mkt (qty-mult-comp pf1 qty) $n w=((v a l-p r o c e s s ~ M k t ~ p f 1)$
$n w) *(q t y(S u c n) w)$
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) $n w) *(q t y(S u c ~ n) w)=$
$\left(\sum x \in(\right.$ support-set pf1 $) .(($ prices Mkt) $x n w) *($ pf1 $x($ Suc $n) w)) *(q t y(S u c$ n) $w$ )
unfolding val-process-def using assms by simp
also have $\ldots=\left(\sum x \in\right.$ (support-set pf1).
$((($ prices Mkt) $x n w) *(p f 1 x($ Suc n) w) $)(q t y(S u c n) w)))$ using sum-distrib-right
by auto
also have $\ldots=\left(\sum x \in\right.$ (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 $\ldots=\left(\sum x \in\right.$ (support-set (qty-mult-comp pf1 qty)).
$(($ prices Mkt) xnw)*((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 $\ldots=$ 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) $n w *(q t y$ (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) ..
lemma remove-comp-values:
assumes $x \neq y$
shows $\forall n w$.pf1 $x n w=(q t y$-rem-comp pf1 y) $x n w$
proof (intro allI)
fix $n w$
show pf1 $x n w=(q t y-r e m-c o m p p f 1 y) x n w$ by (simp add: assms qty-rem-comp-def)
qed
lemma remove-comp-val-process:
assumes portfolio pf1

$n w)-($ prices Mkt y $n w) *(p f 1 y($ Suc $n) w)$
proof (intro allI)
fix $n w$
have $p f$ :portfolio ( $q$ ty-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) $n w)=$
$\left(\sum x \in(\right.$ support-set pf1 $) .(($ prices Mkt) $x n w) *(p f 1 x($ Suc $n) w))$
unfolding val-process-def using assms by simp
also have $\ldots=\left(\sum x \in\right.$ (support-set pf1-\{y\}).
$((($ prices Mkt) $x n w) *(p f 1 x($ Suc $n) w)))+($ prices Mkt $y n w) *(p f 1 y(S u c$ n) $w)$
proof (cases $y \in$ 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 $\ldots=\left(\sum x \in\right.$ (support-set pf1 $\left.-\{y\}\right)$.
$(($ prices Mkt) $x n w) *(($ qty-rem-comp pf1 y) x (Suc $n) w))+($ prices Mkt y $n$ $w) *(p f 1 y(S u c ~ n) w)$
proof -
have $\left(\sum x \in(\right.$ support-set pf1 $-\{y\}) .((($ prices Mkt) $x n w) *(p f 1 x($ Suc $n)$
w))) $=$
$\left(\sum x \in(\right.$ support-set pf1 $-\{y\}) .(($ prices $M k t) x n w) *((q t y-r e m-c o m p ~ p f 1$ y) $x($ Suc $n) w)$
proof (rule sum.cong,simp)
fix $x$
assume $x \in$ support-set pf1 $-\{y\}$
show prices Mkt x $n w * p f 1 x$ (Suc n) $w=$ prices Mkt x $n w * q t y$-rem-comp pf1 $y x($ Suc $n) w$ using remove-comp-values
by (metis DiffD2 $\langle x \in$ support-set pf1 $-\{y\}\rangle$ singletonI)
qed
thus ?thesis by simp
qed
also have $\ldots=($ val-process Mkt $($ qty-rem-comp pf1 y) $n w)+($ prices Mkt y $n$ $w) *(p f 1 y(S u c ~ n) w)$
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) $n \mathrm{w})+($ prices Mkt y $n w) *(p f 1 y(S u c$ n) $w)$.
thus val-process Mkt (qty-rem-comp pf1 y) n w = ((val-process Mkt pf1) n w)
$-($ prices Mkt y $n w) *(p f 1 y(S u c n) w)$ by simp
qed
lemma replace-comp-val-process:
assumes $\forall n w$. 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 $(p f 1 x)) n$ w unfolding qty-replace-comp-def
using assms
sum-val-process[of qty-rem-comp pf1 $x$ qty-mult-comp pf2 ( $p f 1$ x) ]
by (simp add: mult-comp-portfolio remove-comp-portfolio)
also have $\ldots=$ val-process Mkt pf1 $n w-($ prices Mkt x $n w * p f 1 x(S u c n) w)$

+ val-process Mkt pf2 $n w * p f 1 x$ (Suc n) w
by (simp add: assms(2) assms(3) mult-comp-val-process remove-comp-val-process)
also have $\ldots=$ val-process Mkt pf1 $n \mathrm{w}$ using assms by simp
finally show val-process Mkt (qty-replace-comp pf1 x pfQ) $n w=$ val-process $M k t$ $p f 1 n w$.
qed
lemma qty-single-val-process:
shows val-process Mkt (qty-single asset qty) $n w=$
prices Mkt asset $n w * q t y(S u c n) w$
proof -
have val-process Mkt (qty-single asset qty) n $w=$

```
    (sum (\lambdax. ((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)\subseteq {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

 wheretrading-strategy $p \longleftrightarrow$ portfolio $p \wedge(\forall$ asset $\in$ support-set $p$. borel-predict-stoch-proc $F(p$ asset $))$
definition (in disc-filtr-prob-space) support-adapt:: (' $a,{ }^{\prime} b$ ) discrete-market $\Rightarrow$ ('b $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a \Rightarrow$ real $) \Rightarrow$ bool where support-adapt $M k t p f \longleftrightarrow(\forall$ asset $\in$ support-set $p f$. 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 $(S u c n) \in$ 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 $p=(\lambda n w .0)$ unfolding val-process-def using False by simp
thus ?thesis by simp
next
case True
hence val-process Mkt $p n=\left(\lambda w\right.$. $\sum x \in$ support-set $p$. prices Mkt $x n w * p x$ (Suc n) w)
unfolding val-process-def using True by simp
moreover have $\left(\lambda w\right.$. $\sum x \in$ 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 support-set $p$
hence $p$ asset (Suc $n$ ) $\in$ borel-measurable ( $F n$ ) using assms unfolding trad-ing-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（ $\lambda$ x．prices Mkt asset $n x * p$ asset（Suc $n$ ）$x$ ）$\in$ borel－measurable （ $F n$ ）by $\operatorname{simp}$
qed
ultimately show val－process Mkt p $n \in$ borel－measurable（Fn）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 $p=(\lambda n$ 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 $p n=\left(\lambda w . \sum x \in\right.$ support－set $p$ ．prices Mkt $x n w * p x$ （Suc n）w）
unfolding val－process－def using True by simp
moreover have $\left(\lambda w\right.$ ．$\sum x \in$ support－set p．prices Mkt $x n w * p x($ Suc $\left.n) w\right) \in$ 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 predict－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（ $\lambda x$ ．prices Mkt asset $n x * p$ asset（Suc $n$ ）$x) \in$ borel－measurable （F $n$ ）by $\operatorname{simp}$
qed
ultimately show val－process Mkt $p n \in$ 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 $\exists c . \forall w \in$ 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 $p f$
and support-adapt Mkt pf
shows $\forall w \in$ space $M$. val-process Mkt pf $0 w=$ initial-value pf unfolding ini-
tial-value-def
proof (rule constant-imageI)
show $\exists c . \forall w \in$ 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 \(\forall\) asset \(\in\) support-set pf1 \(\cup B\). borel-predict-stoch-proc \(F\) (pf1 asset)
proof
    fix asset
    assume asset \(\in\) support-set pf1 \(\cup B\)
    show borel-predict-stoch-proc \(F\) (pf1 asset)
    proof (cases asset \(\in\) support-set pf1)
        case True
        thus ?thesis using assms unfolding trading-strategy-def by simp
    next
        case False
        hence \(\forall n w\).pf1 asset \(n w=0\) unfolding support-set-def by simp
        show ?thesis unfolding predict-stoch-proc-def
        proof
            show pf1 asset \(0 \in\) measurable ( \(F\) 0) borel using \(\langle\forall n w . p f 1\) asset \(n w=0\rangle\)
                by (simp add: borel-measurable-const measurable-cong)
        next
            show \(\forall n\). pf1 asset \((S u c n) \in\) borel-measurable \((F n)\)
            proof
                fix \(n\)
```

have $\forall w$ ．pf1 asset（Suc n）$w=0$ using 〈 $\forall n w . p f 1$ asset $n w=0\rangle$ by simp
have $0 \in$ space borel by simp
thus pf1 asset（Suc n）$\in$ measurable（ $F$ n）borel using measurable－const［of 0 borel Fn］
by（metis $\langle 0 \in$ space borel $\Longrightarrow(\lambda x .0) \in$ borel－measurable $(F n)\rangle\langle 0 \in$ space borel＞
$\langle\forall n w . p f 1$ 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 $\in$ support－set pf1 $\cup B$
shows borel－predict－stoch－proc $F$（pf1 asset）
proof（cases asset $\in$ support－set pf1）
case True
thus ？thesis using assms unfolding trading－strategy－def by simp
next
case False
hence $\forall n w$ ．pf1 asset $n w=0$ unfolding support－set－def by simp
show ？thesis unfolding predict－stoch－proc－def
proof
show pf1 asset $0 \in$ measurable（ $F 0$ ）borel using 〈 $\forall n w . p f 1$ asset $n w=0$ 〉
by（simp add：borel－measurable－const measurable－cong）
next
show $\forall n$ ．pf1 asset $(S u c n) \in$ borel－measurable $(F n)$
proof
fix $n$
have $\forall w$ ．pf1 asset（Suc n）$w=0$ using $\langle\forall n w$ ．pf1 asset $n w=0\rangle$ by simp have $0 \in$ space borel by simp
thus pf1 asset（Suc $n$ ）$\in$ measurable（ $F$ n）borel using measurable－const $[$ of 0 borel $F n$ ］
by（metis $\langle 0 \in$ space borel $\Longrightarrow(\lambda x .0) \in$ borel－measurable $(F n)\rangle\langle 0 \in$ space
borel＞
〈 $\forall$ 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 $\forall$ asset $\in$ support－set pf1 $\cup B$ ．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 }\in\mathrm{ support-set qty-empty }\Longrightarrow\mathrm{ 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 }\forall\mathrm{ asset }\in\mathrm{ support-set pf1 U support-set pf2. borel-predict-stoch-proc F (pf1
asset)
            using assms by (simp add: inc-predict-support-trading-strat)
    have }\forall\mathrm{ asset }\in\mathrm{ support-set pf2 U support-set pf1. borel-predict-stoch-proc F (pf2
asset)
            using assms by (simp add: inc-predict-support-trading-strat)
            have }\forall\mathrm{ asset }\in\mathrm{ support-set pf1 U support-set pf2. borel-predict-stoch-proc F
((qty-sum pf1 pf2) asset)
    proof
            fix asset
            assume asset \in support-set pf1 \cup support-set pf2
    show borel-predict-stoch-proc F (qty-sum pf1 pf2 asset) unfolding predict-stoch-proc-def
qty-sum-def
            proof
            show (\lambdaw. pf1 asset 0 w + pf2 asset 0 w) \in borel-measurable (F 0)
            proof -
                have ( }\lambdaw.pf1 asset 0 w) \in borel-measurable (F 0)
                    using <\forall asset\insupport-set pf1 \cup support-set pf2. borel-predict-stoch-proc F
(pf1 asset)>
            <asset \in support-set pf1 \cup support-set pf2> predict-stoch-proc-def by blast
            moreover have ( }\lambdaw.pf2 asset 0 w) \in borel-measurable (F 0)
                    using <\forall asset\insupport-set pf2 \cup support-set pf1. borel-predict-stoch-proc
F (pf2 asset)>
                <asset \in support-set pf1 \cup support-set pf2> predict-stoch-proc-def by blast
                ultimately show ?thesis by simp
            qed
            next
            show }\foralln.(\lambdaw.pf1 asset (Suc n)w+ pf2 asset (Suc n) w) \in borel-measurable
(Fn)
            proof
                    fix n
            have ( }\lambdaw.pf1 asset (Suc n) w) \in borel-measurable (F n)
                using «\forall asset\insupport-set pf1 \cup support-set pf2. borel-predict-stoch-proc
```

```
F (pf1 asset)>
            <asset \in support-set pf1 \cup support-set pf2` predict-stoch-proc-def by blast
            moreover have ( }\lambdaw.pf2\mathrm{ asset (Suc n) w) G borel-measurable (F n)
            using «\forall asset\insupport-set pf2 \cup support-set pf1. borel-predict-stoch-proc
F (pf2 asset)>
            <asset \in support-set pf1 U support-set pf2` predict-stoch-proc-def by blast
                    ultimately show (\lambdaw. pf1 asset (Suc n) w + pf2 asset (Suc n) w) \in
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 }\forall\mathrm{ asset }\in\mathrm{ support-set pf1. borel-predict-stoch-proc F (pf1 asset)
        using assms unfolding trading-strategy-def by simp
    have }\forall\mathrm{ asset }\in\mathrm{ 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 \in support-set pf1
    show (\lambdaw.pf1 asset 0 w * qty 0 w) \in borel-measurable (F 0)
    proof -
            have ( }\lambdaw.pf1 asset 0 w) \in borel-measurable (F 0)
            using <\forall asset\insupport-set pf1. borel-predict-stoch-proc F (pf1 asset)>
            <asset \in support-set pf1 > predict-stoch-proc-def by auto
            moreover have ( }\lambdaw.\mathrm{ qty 0 w) E borel-measurable (F 0) using assms pre-
dict-stoch-proc-def by auto
            ultimately show (\lambdaw.pf1 asset 0 w * qty 0 w) \in borel-measurable (F 0) by
simp
    qed
    show }\foralln.(\lambdaw.pf1 asset (Suc n) w* qty (Suc n) w) \in borel-measurable (F n
    proof
        fix n
        have ( }\lambdaw.pf1 asset (Suc n) w) \in borel-measurable (F n
            using <\forall asset\insupport-set pf1. borel-predict-stoch-proc F (pf1 asset)>
            <asset \in support-set pf1> predict-stoch-proc-def by blast
            moreover have (\lambdaw. qty (Suc n) w) \inborel-measurable (F n) using assms
predict-stoch-proc-def by blast
    ultimately show (\lambdaw.pf1 asset (Suc n) w* qty (Suc n) w) \inborel-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 }\forall\mathrm{ asset }\in\mathrm{ support-set pf1. borel-predict-stoch-proc F (pf1 asset)
        using assms unfolding trading-strategy-def by simp
    have }\forall\mathrm{ asset }\in\mathrm{ 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 \in support-set pf1
        show (pf1(x:= \lambdan w. 0)) asset 0 G borel-measurable (F 0)
        proof -
            show (\lambdaw. (pf1(x:= \n w. 0)) asset 0 w) \in borel-measurable (F 0)
            proof (cases asset =x)
                case True
                thus ?thesis by simp
            next
                case False
                thus (\lambdaw. (pf1 (x:= \lambdan w.0)) asset 0 w) \in borel-measurable (F 0)
                        using <\forall asset\insupport-set pf1. borel-predict-stoch-proc F (pf1 asset)>
                    <asset \in support-set pf1> by (simp add: predict-stoch-proc-def)
            qed
        qed
        show }\foralln.(\lambdaw.(pf1(x:=\lambdanw.0)) asset (Suc n) w)\in borel-measurable (F n
        proof
            fix n
            show (\lambdaw. (pf1(x:= \lambdan w.0)) asset (Suc n) w) \in borel-measurable (F n)
            proof (cases asset =x)
                case True
                    thus ?thesis by simp
            next
                    case False
                    thus (\lambdaw. (pf1(x:= \lambdan w. 0)) asset (Suc n) w) \inborel-measurable (F n)
                    using <\forall asset\insupport-set pf1. borel-predict-stoch-proc F (pf1 asset)>
                    <asset \in 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 ( $p f 1 x)$ )
proof (cases $x \in$ support-set pf1)
case True
hence borel-predict-stoch-proc $F$ ( $p f 1 x$ ) using assms unfolding trading-strategy-def by $\operatorname{simp}$
thus ?thesis using assms by (simp add: mult-comp-trading-strat)

## next

case False
thus ?thesis
proof -
obtain $n n::{ }^{\prime} c \Rightarrow\left({ }^{\prime} c \Rightarrow n a t \Rightarrow{ }^{\prime} a \Rightarrow\right.$ real $) \Rightarrow$ nat and $a a::{ }^{\prime} c \Rightarrow\left({ }^{\prime} c \Rightarrow n a t\right.$ $\Rightarrow^{\prime} a \Rightarrow$ real $) \Rightarrow^{\prime} a$ where
$\forall x 0 x 1 .(\exists v 2 v 3 . x 1 x 0 v 2 v 3 \neq 0)=(x 1 x 0($ nn x0 x1 $)($ aa x0 x1 $) \neq 0)$
by moura
then have $\forall f c .(c \notin\{c . \exists n a . f c n a \neq 0\} \vee f c(n n c f)(a a c f) \neq 0) \wedge$ $(c \in\{c . \exists n a . f c n a \neq 0\} \vee(\forall n a . f c n a=0))$
by auto
then show ?thesis
proof -
have $\wedge f c n a$. qty-mult-comp $f(p f 1 x)\left(c::^{\prime} c\right) n a=0$
by (metis False $\langle\forall c .(c \notin\{c . \exists n a . f c n a \neq 0\} \vee f c(n n c f)(a a$ $c f) \neq 0) \wedge(c \in\{c . \exists n a . f c n a \neq 0\} \vee(\forall n a . f c n a=0))\rangle$ mult.commute mult-zero-left qty-mult-comp-def support-set-def)
then show ?thesis
by (metis (no-types) $\langle\forall f c .(c \notin\{c . \exists n a . f c n a \neq 0\} \vee f c(n n c f)$
$(a a c f) \neq 0) \wedge(c \in\{c . \exists n a . f c n a \neq 0\} \vee(\forall n a . f c n a=0))\rangle \operatorname{assms}(2)$
mult-comp-portfolio support-set-def trading-strategy-def)
qed
qed
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 $0 \mid$
up-cl-proc Mkt $p($ Suc $n)=\left(\lambda w . \sum x \in\right.$ 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 $(\neg($ portfolio $p))$ then $(\lambda n w .0)$ else ( $\lambda$ n w . up-cl-proc Mkt $p$ n w) )
lemma (in disc-filtr-prob-space) quantity-updated-borel:
assumes $\forall n . \forall$ asset $\in$ support-set $p . p$ asset $(S u c n) \in$ borel-measurable $(F n)$
and $\forall n$. $\forall$ asset $\in$ support-set p. prices Mkt asset $n \in$ borel-measurable ( $F n$ )
shows $\forall n$. cls-val-process Mkt $p n \in$ borel-measurable (F $n$ )
proof (cases portfolio $p$ )
case False
have cls-val-process Mkt $p=(\lambda n$ w. 0$)$ unfolding cls-val-process-def using
False by simp
thus ?thesis by simp
next
case True
show $\forall n$. cls-val-process Mkt p $n \in$ borel-measurable ( $F n$ )
proof
fix $n$
show cls-val-process Mkt p $n \in$ borel-measurable (F n)
proof (cases $n=0$ )
case False
hence $\exists m . n=$ Suc $m$ using old.nat.exhaust by auto
from this obtain $m$ where $n=S u c m$ by auto
have cls-val-process Mkt $p$ (Suc m) $=\left(\lambda w\right.$. $\sum x \in$ support-set p. prices Mkt $x$ (Suc m) w * px (Suc m) w)
unfolding cls-val-process-def using True by simp
moreover have ( $\lambda w$. $\sum x \in$ support-set p. prices Mkt $x$ (Suc m) w*px (Suc
$m) w) \in$ borel-measurable (F (Suc m))
proof (rule borel-measurable-sum)
fix asset
assume asset support-set $p$
hence $p$ asset (Suc $m$ ) $\in$ 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 $\in$ support-set $p>\operatorname{assms}(2)$ unfolding support-adapt-def
adapt-stoch-proc-def by blast
ultimately show ( $\lambda$ x. prices Mkt asset (Suc m) $x * p$ asset (Suc m) $x) \in$
borel-measurable (F (Suc m)) by simp
qed
ultimately have cls-val-process Mkt $p(S u c m) \in$ borel-measurable (F (Suc
$m)$ ) by $\operatorname{simp}$
thus ?thesis using $\langle n=$ Suc $m 〉$ by simp
next

```
        case True
        thus cls-val-process Mkt p n\in 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 = (\lambda 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 \in borel-measurable (F n)
        proof (cases n=0)
        case True
            thus cls-val-process Mkt p n \in 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 }\existsm\mathrm{ . Suc m=n using not0-implies-Suc by blast
            from this obtain m}\mathrm{ 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 .. = (\lambdaw. \sumx\insupport-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 = ( \lambdaw. \sumx\insupport-set p. prices Mkt x
nw*pxnw).
            moreover have ( }\lambdaw.\sumx\in\mathrm{ support-set p. prices Mkt x n w * px n w) }
borel-measurable (F n)
    proof (rule borel-measurable-sum)
            fix asset
            assume asset\in support-set p
            hence p asset n \in borel-measurable (F n) using assms unfolding trad-
ing-strategy-def predict-stoch-proc-def
```

using Suc－n－not－le－n〈Suc $m=n\rangle$ increasing－measurable－info nat－le－linear by blast
moreover have prices Mkt asset $n \in$ borel－measurable（ $F n$ ）using assms $\langle a s s e t \in$ support－set $p>$ unfolding support－adapt－def adapt－stoch－proc－def
using stock－portfolio－def by blast
ultimately show（ $\lambda x$ ．prices Mkt asset $n x * p$ asset $n x) \in$ borel－measurable （F $n$ ）by $\operatorname{simp}$
qed
ultimately show cls－val－process Mkt $p n \in$ borel－measurable（ $F n$ ）by simp qed
qed
qed
lemma subset－cls－val－process：
assumes finite $A$
and support－set $p \subseteq A$
shows $\forall n$ w．cls－val－process Mkt $p($ Suc $n) w=(\operatorname{sum}(\lambda x$ ．（ $($ prices Mkt）$x$（Suc $n) w) *(p x($ Suc n）w）$) A)$
proof（intro allI）
fix $n:: n a t$
fix $w:: ' b$
have portfolio $p$ using assms unfolding portfolio－def using finite－subset by auto
have $\exists C$ ．（support－set $p) \cap C=\{ \} \wedge($ support－set $p) \cup C=A$ using assms（2） by auto
from this obtain $C$ where（support－set $p) \cap C=\{ \}$ and（support－set $p$ ）$\cup C$ $=A$ by auto note Cprops $=$ this
have finite $C$ using assms 〈（support－set $p) \cup C=A$ 〉 by auto
have $\forall x \in C . p x(S u c n) w=0$ using $\operatorname{Cprops}(1)$ support－set－def by fastforce
hence $\left(\sum x \in C\right.$ ．（（prices Mkt）$x($ Suc $\left.n) w\right) *(p x($ Suc $\left.n) w)\right)=0$ by simp
hence cls－val－process Mkt $p$（Suc $n$ ）$w=\left(\sum x \in(\right.$ support－set $p)$ ．（ $($ prices Mkt）$x$ $($ Suc $n) w) *(p x($ Suc $n) w))$
$+\left(\sum x \in C .\left((\right.\right.$ prices Mkt）$x($ Suc $n) w) *\left(\begin{array}{ll}p x(\text { Suc } n) w)) \text { unfolding }\end{array}\right.$ cls－val－process－def
using 〈portfolio p〉up－cl－proc．simps（2）［of Mkt p n］by simp
also have $\ldots=\left(\sum x \in A\right.$ ．$(($ prices Mkt）$x($ Suc $n) w) *(p x($ Suc $n) w))$
using 〈portfolio $p\rangle\langle$ finite $C$ 〉Cprops portfolio－def sum－union－disjoint＇by（metis （no－types，lifting））
finally show cls－val－process Mkt $p($ Suc $n) w=\left(\sum x \in A\right.$ ．（（prices Mkt）$x$（Suc $n) w) *(p x(S u c n) w))$ ．
qed
lemma subset－cls－val－process＇：
assumes finite $A$
and support－set $p \subseteq A$
shows cls－val－process Mkt $p($ Suc $n) w=(\operatorname{sum}(\lambda x$ ．$(($ prices Mkt）$x($ Suc n）$w) *$ $(p x(S u c n) w)) A)$
proof－
have portfolio $p$ using assms unfolding portfolio－def using finite－subset by
auto
have $\exists C$ ．（support－set $p) \cap C=\{ \} \wedge($ support－set $p) \cup C=A$ using assms（2） by auto
from this obtain $C$ where（support－set $p) \cap C=\{ \}$ and（support－set $p$ ）$\cup C$ $=A$ by auto note Cprops $=$ this have finite $C$ using assms «（support－set $p) \cup C=A$ by auto
have $\forall x \in C . p x(S u c n) w=0$ using $\operatorname{Cprops}(1)$ support－set－def by fastforce
hence $\left(\sum x \in C\right.$ ．（（prices Mkt）$x($ Suc $\left.n) w\right) *(p x($ Suc $\left.n) w)\right)=0$ by simp
hence cls－val－process Mkt $p$（Suc $n$ ）$w=\left(\sum x \in(\right.$ support－set $p)$ ．（ $($ prices Mkt）$x$ $($ Suc $n) w) *(p x($ Suc $n) w))$
$+\left(\sum x \in C .\left((\right.\right.$ prices Mkt）$x($ Suc $n) w) *\left(\begin{array}{ll}p x(S u c ~ n) & w)) \text { unfolding }\end{array}\right.$ cls－val－process－def
using 〈portfolio p〉up－cl－proc．simps（2）［of Mkt p n］by simp
also have $\ldots=\left(\sum x \in A .((\right.$ 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 $p(S u c n) w=\left(\sum x \in A\right.$ ．（（prices Mkt）$x$（Suc $n) w) *(p x(S u c n) w))$ ．
qed
lemma sum－cls－val－process－Suc：
assumes portfolio pf1
and portfolio pf2
shows $\forall n w$ ．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）$w=$
$\left(\sum x \in(\right.$ support－set pf1 $) \cup($ support－set pf2）$)(($ prices Mkt）$x($ Suc $n) w) *(p f 1$ $x($ Suc $n) w)$ ）
proof－
have finite（support－set pf1 $\cup$ support－set pf2）$\wedge$ support－set pf1 $\subseteq$ support－set $p f 1 \cup$ 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）$w=\left(\sum x \in\right.$（support－set pf1）$\cup$ （support－set pf2）．（（prices Mkt）x（Suc n）w）＊（pf2 x（Suc n）w））
proof－
have finite（support－set pf1 $\cup$ support－set pf2）$\wedge$ support－set pf2 $\subseteq$ support－set pf2 $\cup$ 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 $\cup$ support－set pf2）using assms finite－Un un－
folding portfolio-def by auto
have (cls-val-process Mkt pf1) (Suc n) $w+($ cls-val-process Mkt pf2) (Suc n) w $=$
$\left(\sum x \in(\right.$ support-set $p f 1) \cup($ support-set pf2 $) .(($ prices Mkt) $x($ Suc $n) w) *(p f 1$ $x($ Suc $n) w))+$
$\left(\sum x \in(\right.$ support-set $p f 1) \cup($ support-set pf2 $) .(($ prices Mkt) $x($ Suc $n) w) *(p f 2$ $x($ Suc $n) w)$
using vp1 vp2 by simp
also have $\ldots=\left(\sum x \in\right.$ (support-set pf1 $) \cup$ (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 $\ldots=\left(\sum x \in\right.$ (support-set pf1 $) \cup$ (support-set pf2).
$(($ prices Mkt) $x($ Suc $n) w) *(($ pf1 x $($ Suc $n) w)+(p f 2 x($ Suc $n) w))$ by $($ simp add: distrib-left)
also have $\ldots=\left(\sum x \in\right.$ (support-set pf1 $) \cup$ (support-set pf2).
$(($ prices Mkt) $x($ Suc $n) w) *((q t y-s u m p f 1$ pf2) $x($ Suc n) w) by $($ simp add: qty-sum-def)
also have $\ldots=\left(\sum x \in\right.$ (support-set (qty-sum pf1 pf2)).
$(($ prices Mkt) $x($ Suc $n) w) *((q t y-s u m p f 1$ pf2) $x(S u c n) w))$ using sum-support-set $[o f$ pf1 pf2]
subset-cls-val-process[of support-set pf1 $\cup$ support-set pf2 qty-sum pf1 pf2] pf fin
unfolding cls-val-process-def by simp
also have $\ldots=$ 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 $\forall w$. 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 $\forall n$ w. cls-val-process Mkt (qty-sum pf1 pf2) $n w=$
(cls-val-process Mkt pf1) nw + (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 $\forall w$. 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 $\forall n$ 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 $p f$ :portfolio ( $q t y$-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 $n) w) *(q t y(S u c ~ n) w)=$
$\left(\sum x \in(\right.$ support-set pf1 $) .(($ prices Mkt) $x($ Suc n) $w) *($ pf1 $x($ Suc $n) w)) *(q t y$ (Suc n) w)
unfolding cls-val-process-def using assms by simp
also have $\ldots=\left(\sum x \in\right.$ (support-set pf1).
$((($ prices Mkt) $x($ Suc $n) w) *($ pf1 x $($ Suc $n) w) *(q t y(S u c n) w)))$ using sum-distrib-right by auto
also have $\ldots=\left(\sum x \in(\right.$ support-set $p f 1)$.
$(($ prices Mkt) $x(S u c n) w) *((q t y-m u l t-c o m p ~ p f 1 ~ q t y) x(S u c ~ n) w))$ unfolding qty-mult-comp-def
by (simp add: mult.commute mult.left-commute)
also have $\ldots=\left(\sum x \in(\right.$ 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 $\ldots=$ 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) $w *(q t y(S u c ~ n) w)=c l s$-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 $\forall$ w. cls-val-process Mkt (qty-rem-comp pf1 y) $0 w=$
((cls-val-process Mkt pf1) 0 w$)-($ prices Mkt y $0 w) *(p f 1 y$ (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 $\forall n$ 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)* $\begin{aligned} & \text { pf1 y (Suc }\end{aligned}$ n) $w$ )
proof (intro allI)
fix $n w$
have $p f$ :portfolio ( $q$ ty-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) =
$\left(\sum x \in(\right.$ support-set pf1 $) .(($ prices Mkt $) x($ Suc $n) w) *(p f 1 x($ Suc n)w) $)$
unfolding cls-val-process-def using assms by simp
also have $\ldots=\left(\sum x \in(\right.$ 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 $y \in$ 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 $\ldots=\left(\sum x \in(\right.$ 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) * $(p f 1 y$ (Suc $n) w)$


## proof -

have $\left(\sum x \in(\right.$ support-set pf1 $-\{y\}) .((($ prices Mkt) $x($ Suc $n) w) *(p f 1 x($ Suc n) $w)$ ) $=$
$\left(\sum x \in(\right.$ support-set pf1 $-\{y\}) .(($ prices $M k t) x(S u c n) w) *((q t y-r e m-c o m p$ pf1 y) $x($ Suc n) w) $)$
proof (rule sum.cong,simp)
fix $x$
assume $x \in$ support-set pf1 $-\{y\}$
show prices Mkt x (Suc n) w*pf1x (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 $\in$ support-set pf1 $-\{y\}\rangle$ singletonI)
qed
thus ?thesis by simp
qed
also have $\ldots=($ cls-val-process $M k t($ qty-rem-comp pf1 y) (Suc n) $w)+($ prices Mkt $y$ (Suc n) $w) *(p f 1 y($ 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) $w) *(p f 1 y($ 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 Mkty (Suc n) w) ${ }^{(p f 1 y(S u c}$
n) $w$ ) by $\operatorname{simp}$

## qed

lemma replace-comp-cls-val-process0:
assumes $\forall w$. prices Mkt x $0 w=$ cls-val-process Mkt pf2 $0 w$
and portfolio pf1
and portfolio pf2
shows $\forall w$. 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) $0 \mathrm{w}=$ 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 $\ldots=$ cls-val-process Mkt pf1 $0 w-($ prices Mkt x 0 w *pf1 x (Suc 0)
w) +
cls-val-process Mkt pf2 $0 w * p f 1 x(S u c ~ 0) w$
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 pfo) $0 w=$ cls-val-process Mkt pf1 0 w.
qed
lemma replace-comp-cls-val-process-Suc:
assumes $\forall n w$. prices Mkt $x$ (Suc $n$ ) $w=$ cls-val-process Mkt pf2 (Suc n) $w$
and portfolio pf1
and portfolio pf2
shows $\forall n$ 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 $\ldots=$ 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 $\ldots=$ 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 $\forall n$ w. prices Mkt $x n w=$ cls-val-process Mkt pf2 $n w$
and portfolio pf1
and portfolio pf2
shows $\forall n w$. cls-val-process Mkt (qty-replace-comp pf1 xpf2) $n w=$ 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 $(\lambda x$. $(($ prices Mkt) $x($ Suc $n) w) *((q t y-s i n g l e ~ a s s e t ~ q t y) ~ x(S u c ~ n) w))$
\{asset\})
proof (rule subset-cls-val-process')
show finite \{asset \} by simp
show support-set (qty-single asset qty) $\subseteq\{$ asset $\}$ by (simp add: single-comp-support)
qed
also have $\ldots=$ prices $M k t$ asset (Suc n) $w * q t y$ (Suc n) $w$ unfolding $q t y$-single-def by $\operatorname{simp}$
finally show ?thesis.
qed

## Self-financing definition self-financing where

self-financing Mkt $p \longleftrightarrow(\forall n$. val-process Mkt $p$ (Suc $n)=$ cls-val-process Mkt $p$ (Suc n))
lemma self-financingE:
assumes self-financing Mkt $p$
shows $\forall n$. 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 \(\forall x \in\) support-set \(p .(\forall w i . p x i w=p x(\) 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 \(\forall w\). ( \(\sum x \in\) 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 ( \(\sum x \in\) 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 ( \(\sum x \in\) support-set \(p\). prices Mkt \(x\) (Suc n) \(w * p x(\) Suc (Suc n) \(\left.) w\right)=\)
                    ( \(\sum x \in\) support-set \(p\). prices Mkt \(x(\) Suc \(n) w * p x(\) Suc n) \(w)\)
            proof (rule sum.cong, simp)
                fix \(x\)
                    assume \(x \in\) 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 * \(\mathrm{px}(\) Suc (Suc n)) \(w=\) prices Mkt \(x\) (Suc
n) \(w * p x(\) Suc \(n) w\) by \(\operatorname{simp}\)
                    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 \(\forall w\). val-process Mkt \(p\) (Suc \(n\) ) \(w=\left(\sum x \in\right.\) support-set \(p\). prices
Mkt \(x\) (Suc n) w*px (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 $\forall n$ 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 pfo (Suc n) w
using assms by (simp add:sum-val-process)
also have $\ldots=$ 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 $\ldots=$ 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 $\forall n w$. qty $n w=q t y$ (Suc $n$ ) $w$
shows self-financing Mkt (qty-mult-comp pf1 qty)
proof -
have $\forall n$ 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 $\ldots=$ cls-val-process Mkt pf1 (Suc n) $w * q t y$ (Suc n) $w$ using assms
unfolding self-financing-def by simp
also have $\ldots=$ 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 }\forallnw. 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: \foralln w. prices Mkt x n w = val-process Mkt pf2 n w using assms by
(simp add: self-financingE)
    have }\foralln 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 vpf asset $0=(\lambda w .0) \mid$
first: remaining-qty Mkt v pf asset (Suc 0) $=(\lambda w .(v-v a l-p r o c e s s ~ M k t ~ p f ~ 0 ~$ $w) /($ prices Mkt asset $0 w)) \mid$
step: remaining-qty Mkt $v$ pf asset $($ Suc $($ Suc $n))=(\lambda w$. (remaining-qty Mkt $v p f$ 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) $\in$ borel-measurable (F n)
proof (induct $n$ )
case 0
have $(\lambda w .(v-$ val-process Mkt pf $0 w) /($ prices Mkt asset $0 w)) \in$ borel-measurable (F 0 )
proof (rule borel-measurable-divide)

```
    have val-process Mkt pf 0 \in borel-measurable (F 0) using assms
            ats-val-process-adapted by (simp add:adapt-stoch-proc-def)
    thus ( }\lambdax.v-val-process Mkt pf 0x) \in borel-measurable (F 0) by sim
            show prices Mkt asset 0 \in borel-measurable (F 0) using assms unfolding
adapt-stoch-proc-def by simp
    qed
    thus?case by simp
next
    case (Suc n)
    have (\lambdaw. (cls-val-process Mkt pf (Suc n) w - val-process Mkt pf (Suc n) w)/
        (prices Mkt asset (Suc n) w)) \in borel-measurable (F (Suc n))
    proof (rule borel-measurable-divide)
        show (\lambdaw. (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 (\lambdaw. (cls-val-process Mkt pf (Suc n) w)) \in borel-measurable (F (Suc n))
            using assms cls-val-process-adapted unfolding adapt-stoch-proc-def by auto
            show (\lambdaw. (val-process Mkt pf (Suc n)w)) \in 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) \in 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) \inborel-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 pre-
dict-stoch-proc-def
proof (intro conjI allI)
    show remaining-qty Mkt v pf asset 0 \in borel-measurable (F 0) using init by
simp
    fix n
    show remaining-qty Mkt v pf asset (Suc n) \inborel-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 $n \in$ 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 $p f$
and support-adapt Mkt pf
shows borel-adapt-stoch-proc $F$ (remaining-qty Mkt v pf asset) using assms un-
folding 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 $\forall w$. prices Mkt asset $0 w \neq 0$
and portfolio pf
shows val-process Mkt (self-finance Mkt vpf asset) $0 w=v$
proof -
define $s c p$ where $s c p=q t y$-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: < $\backslash q t y$. portfolio (qty-single asset qty)〉 self-finance-def)
also have $\ldots=$ val-process Mkt pf $0 \mathrm{w}+$
$(\operatorname{sum}(\lambda x .(($ prices Mkt) x $0 w) *(\operatorname{scp} x($ Suc 0) w) $)\{$ asset $\})$
using subset-val-process' [of \{asset $\}$ scp] unfolding scp-def by (auto simp add:
single-comp-support)
also have $\ldots=$ val-process Mkt pf $0 w+$ prices Mkt asset $0 w$ *scp asset (Suc
0) $w$ by auto
also have $\ldots=$ 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 $\ldots=$ val-process Mkt pf $0 w+$ prices Mkt asset $0 w *(v-$ val-process Mkt pf $0 w) /($ prices Mkt asset $0 w)$ by $\operatorname{simp}$

```
    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\not=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 (\lambdax. ((prices Mkt) x (Suc n) w)*(scpx (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\not=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 (\lambdax. ((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 }\forallnw\mathrm{ . prices Mkt asset (Suc n) w}\not=
    and portfolio pf
shows self-financing Mkt (self-finance Mkt v pf asset)
proof-
    have \foralln 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::(' $a \Rightarrow$ real) $\Rightarrow$ nat $\Rightarrow$ real $\Rightarrow$ ( $n a t \Rightarrow{ }^{\prime} a \Rightarrow$ real $) \Rightarrow$ bool where price-structure pyf $T \pi$ pr $\longleftrightarrow((\forall w \in$ space $M . \operatorname{pr} 0 w=\pi) \wedge(A E w$ in $M . p r$

$$
T w=\text { pyf } w) \wedge(p r T \in \text { borel-measurable }(F T)))
$$

lemma (in disc-filtr-prob-space) price-structure-init:
assumes price-structure pyf $T \pi p r$
shows $\forall w \in$ space $M$. pr $0 w=\pi$ using assms unfolding price-structure-def by $\operatorname{simp}$
lemma (in disc-filtr-prob-space) price-structure-borel-measurable:
assumes price-structure pyf $T \pi$ pr
shows $\operatorname{pr} T \in$ borel-measurable ( $F T$ ) using assms unfolding price-structure-def
by $\operatorname{simp}$
lemma (in disc-filtr-prob-space) price-structure-maturity:
assumes price-structure pyf $T \pi p r$
shows $A E w$ in $M$. pr $T w=$ pyf $w$ using assms unfolding price-structure-def by $\operatorname{simp}$
definition (in disc-equity-market) replicating-portfolio where
replicating-portfolio pf der matur $\longleftrightarrow$ (stock-portfolio Mkt pf) $\wedge$ (trading-strategy $p f) \wedge($ self-financing Mkt pf $) \wedge$
(AE w in M. cls-val-process Mkt pf matur $w=\operatorname{der} w)$
definition (in disc-equity-market) is-attainable where
is-attainable der matur $\longleftrightarrow(\exists \mathrm{pf}$. 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 $w=\operatorname{der} w$ using assms unfolding
replicating-portfolio-def by simp
show $\forall w \in$ space $M$. cls-val-process Mkt pf $0 w=$ initial-value $p f$
proof
fix $w$
assume $w \in$ space $M$
thus cls-val-process Mkt pf $0 \mathrm{w}=$ 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 $\in$ 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 $p \longleftrightarrow(\exists$ m. (self-financing Mkt $p) \wedge($ trading-strategy $p)$ $\wedge$
$(\forall w \in$ space M. val-process Mkt p $0 w=0) \wedge$
(AE w in M. $0 \leq$ cls-val-process Mkt $p m w) \wedge$
$0<\mathcal{P}(w$ in $M$. cls-val-process Mkt pmw>0))
lemma (in disc-filtr-prob-space) arbitrage-processE:
assumes arbitrage-process Mkt p
shows $(\exists \mathrm{m}$. (self-financing Mkt $p) \wedge($ trading-strategy $p) \wedge$
$(\forall w \in$ space $M$. cls-val-process Mkt p $0 w=0) \wedge$
(AE win M. $0 \leq$ cls-val-process Mkt $p m w) \wedge$
$0<\mathcal{P}(w$ 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 $(\exists$ m. (self-financing Mkt $p) \wedge($ trading-strategy $p) \wedge$
$(\forall w \in$ space M. cls-val-process Mkt p $0 w=0) \wedge$
$(A E w$ in $M .0 \leq$ cls-val-process Mkt $p m w) \wedge$
$0<\mathcal{P}(w$ 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 $M k t \longleftrightarrow$ ( $\forall$ p. stock-portfolio Mkt $p \longrightarrow \neg$ 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 $\forall n$ w. cls-val-process Mkt pf1 $n w=$ cls-val-process Mkt pf2 $n w$
shows arbitrage-process Mkt pf2
proof -
have $(\exists$ m. (self-financing Mkt pf1) $\wedge($ trading-strategy pf1 $) \wedge$
$(\forall w \in$ space M. cls-val-process Mkt pf1 $0 w=0) \wedge$
$(A E w$ in M. $0 \leq$ cls-val-process Mkt pf1 mw) $\wedge$
$0<\mathcal{P}(w$ 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
$(\forall w \in$ space M. cls-val-process Mkt pf1 $0 w=0)$ and
(AE w in M. $0 \leq$ cls-val-process Mkt pf1 mw)

```
    0<\mathcal{P}(w in M. cls-val-process Mkt pf1 m w>0) by auto
    have ae-eq:\forallw\in space M. (cls-val-process Mkt pf1 0 w = 0)=(cls-val-process
Mkt pf2 0 w = 0)
    proof
        fix w
        assume w\in 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 (\lambdaw. cls-val-process Mkt pf1 m w \geq 0) =
almost-everywhere M ( }\lambdaw\mathrm{ . cls-val-process Mkt pf2 m w }\geq0
    proof (rule AE-cong)
            fix w
            assume w\in space M
            show (cls-val-process Mkt pf1 m w \geq0) =(cls-val-process Mkt pf2 mw\geq0)
                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 ( }\forall\textrm{w}\in\mathrm{ space M. cls-val-process Mkt pf2 0 w = 0) using <( }\forall\textrm{w
\in space M. cls-val-process Mkt pf1 0 w = 0)> ae-eq by simp
    moreover have AE w in M. 0 \leqcls-val-process Mkt pf2 m w using <AE w in
M. 0 \leq cls-val-process Mkt pf1 m w> ae-geq by simp
    moreover have 0< prob {w\in space M. 0< cls-val-process Mkt pff mw}
    proof -
    have {w\in space M. 0< cls-val-process Mkt pf2 m w} ={w\in space M. 0<
cls-val-process Mkt pf1 m w}
            by (simp add: assms(4))
    thus ?thesis by (simp add:<0< prob {w\in 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 ^ ( }\forallx.x\inA\longrightarrow\mathrm{ prices
Mkt x = prices Mkt2 x))
lemma coincides-val-process:
    assumes coincides-on Mkt Mkt2 A
    and support-set pf \subseteqA
    shows \foralln 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 = (\sumx\in support-set pf.prices Mkt x n w * pf x
(Suc n) w) using assms
            unfolding val-process-def by simp
    also have ... = (\sumx\in support-set pf. prices Mkt2 x n w * pf x (Suc n)w)
    proof (rule sum.cong, simp)
        fix y
        assume }y\in\mathrm{ support-set pf
        hence }y\inA\mathrm{ 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 nw*pfy (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 \subseteqA
    shows \foralln 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 = (\sumx\in support-set pf.prices Mkt x
(Suc n) w*pfx (Suc n)w) using assms
            unfolding cls-val-process-def by simp
    also have ... = (\sumx\in support-set pf. prices Mkt2 x (Suc n) w * pf x (Suc n)
w)
    proof (rule sum.cong, simp)
        fix y
        assume y\in support-set pf
        hence }y\inA\mathrm{ 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*pfy (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 \subseteqA
    shows \foralln 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\subseteqA
    and self-financing Mkt p
shows self-financing Mkt2 p
proof -
    have }\foralln w.val-process Mkt2 p (Suc n) w =cls-val-process Mkt2 p (Suc n)
    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
```

$$
\text { thus self-financing Mkt2 } p \text { unfolding self-financing-def by auto }
$$ qed

lemma（in disc－filtr－prob－space）coincides－on－arbitrage：
assumes coincides－on Mkt Mkt2 A
and support－set $p \subseteq A$
and arbitrage－process Mkt $p$
shows arbitrage－process Mkt2 $p$
proof－
have $(\exists$ m． （self－financing Mkt $p) \wedge($ trading－strategy $p) \wedge$
$(\forall w \in$ space M．cls－val－process Mkt p $0 w=0) \wedge$
$(A E w$ in $M .0 \leq c l s$－val－process Mkt $p m w) \wedge$
$0<\mathcal{P}(w$ 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 （ $\forall w \in$ space $M$ ．cls－val－process Mkt p $0 w=0$ ）and
（ $A E$ win M． $0 \leq$ cls－val－process Mkt p mw）
$0<\mathcal{P}(w$ in $M$ ．cls－val－process Mkt $p m w>0)$ by auto
have ae－eq：$\forall w \in$ space $M$ ．（cls－val－process Mkt2 p $0 w=0)=($ cls－val－process Mkt p $0 w=0$ ）
proof
fix $w$
assume $w \in$ 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 $M(\lambda w$ ．cls－val－process Mkt2 p $m w \geq 0)=a l$－ most－everywhere $M$（ $\lambda w$ ．cls－val－process Mkt $p m w \geq 0$ ）
proof（rule AE－cong）
fix $w$
assume $w \in$ space $M$
show（cls－val－process Mkt2 p m w $\geq 0)=($ cls－val－process Mkt p m w $\geq 0)$
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 $(\forall w \in$ space M．cls－val－process Mkt2 p $0 w=0)$ using $\langle(\forall w \in$ space M．cls－val－process Mkt p $0 w=0$ ）＞ae－eq by simp
moreover have $A E w$ in $M .0 \leq$ cls－val－process Mkt2 $p m w$ using $\langle A E w$ in M． $0 \leq$ cls－val－process Mkt $p$ m w ae－geq by simp
moreover have $0<\operatorname{prob}\{w \in$ space M． $0<$ cls－val－process Mkt2 $p m w\}$
proof－
have $\{w \in$ space M． $0<$ cls－val－process Mkt2 p $m w\}=\{w \in$ space $M .0<$ cls－val－process Mkt p m w\}
by（metis（no－types，lifting）assms（1）assms（2）coincides－cls－val－process）
thus ？thesis by（simp add：$\langle 0<$ prob $\{w \in$ space M． $0<$ cls－val－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 $n w \mathbf{u s i n g}$ 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 }\forallw\mathrm{ . 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

$$
\text { fair-price Mkt } \pi \text { pyf matur } \longleftrightarrow
$$

( $\exists$ pr. price-structure pyf matur $\pi$ pr $\wedge$
( $\forall$ x Mkt2 $p$. $(x \notin$ stocks Mkt $\longrightarrow$
$(($ coincides-on Mkt Mkt2 $($ stocks Mkt) $) \wedge($ prices Mkt2 $x=p r) \wedge$ 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 $\pi$ pyf matur
shows ( $\exists$ pr. price-structure pyf matur $\pi$ pr $\wedge$
( $\forall$ x. $(x \notin$ stocks Mkt $\longrightarrow$
$(\forall$ Mkt2 $p .($ coincides-on Mkt Mkt2 $($ stocks Mkt $)) \wedge($ prices Mkt2 $x=p r) \wedge$ 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) repli-cating-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 $\pi$ where $\pi=($ initial-value $p f)$
assume $\neg$ fair-price Mkt $\pi$ der matur
hence imps: $\forall$ pr. (price-structure der matur $\pi$ pr $) \longrightarrow(\exists x$ Mkt2 $p$. $(x \notin$ stocks Mkt $\wedge$
(coincides-on Mkt Mkt2 (stocks Mkt)) $\wedge($ prices Mkt2 $x=p r) \wedge$ 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 $\pi$ (cls-val-process Mkt pf)) unfolding $\pi$-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 $M k t \cup\{x\}$ and arbitrage－process Mkt2 $p$ by auto
have $\forall n$ w．cls－val－process Mkt pf $n w=$ cls－val－process Mkt2 $p f n w$
using coincides－stocks－cls－val－process［of Mkt pf Mkt2］assms 〈coincides－on Mkt Mkt2（stocks Mkt）＞unfolding replicating－portfolio－def by $\operatorname{simp}$
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 $p m w) \wedge 0<\operatorname{prob}\{w \in$ space $M .0$
$<$ cls－val－process Mkt2 p mw\}
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 $p m w) \wedge 0<\operatorname{prob}\{w \in$ space M． 0
$<$ cls－val－process Mkt2 $p m w\}$ by auto note mprop $=$ this
define eq－stock where eq－stock $=q t y$－replace－comp $\quad$ p $x$ pf
have $\forall n$ w．cls－val－process Mkt pf $n w=$ cls－val－process Mkt2 pf $n w \mathbf{u s i n g}$ 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 $n w\rangle$ 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 $\cup\{x\}\rangle$
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 $M k t p f\rangle$ 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 $\forall n$ 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 re－ quires 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 \in stocks Mkt
    and positive: }\forall n w. prices Mkt pos-stock n w>0
and readable: }\forall\mathrm{ asset }\in\mathrm{ 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 \(p \longleftrightarrow(\forall\) asset \(\in\) support-set \(p . \exists c::\) real. \(p\) asset \(=(\lambda n w, c))\)
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 \(\in\) support-set \(p\)
    hence \(\exists\) c. p asset \(=(\lambda n w . c)\) using assms unfolding static-quantities-def by
simp
    then obtain \(c\) where \(p\) asset \(=(\lambda n w . c)\) by auto
    show borel-predict-stoch-proc \(F\) ( \(p\) asset) unfolding predict-stoch-proc-def
```

```
    proof (intro conjI)
    show p asset 0 G borel-measurable (F 0) using <p asset = ( }\lambdan~w.c)\rangle\mathrm{ by simp
    show }\foralln.p\mathrm{ asset (Suc n) G borel-measurable (F n)
    proof
    fix n
        have p asset (Suc n) = (\lambda w.c) using <p asset = (\lambda n w.c)> by simp
        thus p asset (Suc n) \in borel-measurable (F n) by simp
    qed
    qed
qed
```

lemma two-component-support-set:
assumes $\exists n$. a $n w \neq 0$
and $\exists n w$. $b n w \neq 0$
and $x \neq y$
shows support-set $\left(\left(\lambda\left(x::^{\prime} b\right)(n:: n a t)\left(w::^{\prime} a\right) .0::\right.\right.$ real $\left.)(x:=a, y:=b)\right)=\{x, y\}$
proof
let ?arb-pf $=\left(\lambda\left(x::^{\prime} b\right)(n:: n a t)\left(w::^{\prime} a\right) .0::\right.$ real $)(x:=a, y:=b)$
have $\exists n w$. ?arb-pf $x n w \neq 0$ using assms by simp
moreover have $\exists n w$. ? arb-pf y $n w \neq 0$ using assms by simp
ultimately show $\{x, y\} \subseteq$ support-set ?arb-pf unfolding support-set-def by
simp
show support-set ?arb-pf $\subseteq\{x, y\}$
proof (rule ccontr)
assume $\neg$ support-set ?arb-pf $\subseteq\{x, y\}$
hence $\exists z$. $z \in$ support-set ?arb-pf $\wedge z \notin\{x, y\}$ by auto
from this obtain $z$ where $z \in$ support-set ?arb-pf and $z \notin\{x, y\}$ by auto
have $\exists n w$. ?arb-pf $z n w \neq 0$ using $\langle z \in$ support-set ?arb-pf〉 unfolding
support-set-def by simp
from this obtain $n w$ where ? arb-pf $z n w \neq 0$ by auto
have ?arb-pf zn $w=0$ using $\langle z \notin\{x, y\} 〉$ by simp
thus False using <? arb-pf $z n w \neq 0$ 〉 by simp
qed
qed
lemma two-component-val-process:
assumes $\operatorname{arb}-p f=\left(\left(\lambda\left(x::^{\prime} b\right)(n:: n a t)\left(w::^{\prime} a\right) .0::\right.\right.$ real $\left.)(x:=a, y:=b)\right)$
and portfolio arb-pf
and $x \neq y$
and $\exists n w$. a $n w \neq 0$
and $\exists n w . b n w \neq 0$
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 $n w=\left(\sum x \in\right.$ 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 $\ldots=\left(\sum x \in\{x, y\}\right.$. prices Mkt $x n w * \operatorname{arb-pf} x$ (Suc n) w) using «support-set arb-pf $=\{x, y\}\rangle$
by $\operatorname{simp}$
also have $\ldots=\left(\sum x \in\{y\}\right.$. prices Mkt x $n w * \operatorname{arb}$-pf $x($ Suc $\left.n) w\right)+$ prices Mkt $x n w * \operatorname{arb}-p f x($ Suc $n) w$
using sum.insert $[$ of $\{y\} x \lambda x$. prices Mkt $x n w * \operatorname{arb-pf} x n w] \operatorname{assms}(3)$ by auto
also have $\ldots=$ prices Mkt $y n w * \operatorname{arb}-\mathrm{pf} y($ Suc $n) w+$ prices Mkt $x n w *$ arb-pf $x$ (Suc $n$ ) w by simp
also have $\ldots=$ 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 $\exists n w$. pr $n w \neq 0$
and $x \notin$ support-set $p$
shows support-set $(p(x:=p r))=$ support-set $p \cup\{x\}$
proof
show support-set $(p(x:=p r)) \subseteq$ support-set $p \cup\{x\}$
proof
fix $y$
assume $y \in$ support-set $(p(x:=p r))$
show $y \in$ support-set $p \cup\{x\}$
proof (rule ccontr)
assume $\neg y \in$ support-set $p \cup\{x\}$
hence $y \neq x$ by $\operatorname{simp}$
have $\exists n w .(p(x:=p r))$ y $n w \neq 0$ using $\langle y \in \operatorname{support-set}(p(x:=p r))$ 〉
unfolding support-set-def by simp
then obtain $n w$ where nwprop: $(p(x:=p r))$ y $n w \neq 0$ by auto have $y \notin$ support-set $p$ using $\langle\neg y \in$ support-set $p \cup\{x\}\rangle$ by $\operatorname{simp}$ hence $y=x$ using nwprop using support-set-def by force thus False using $\langle y \neq x\rangle$ by $\operatorname{simp}$
qed
qed
show support-set $p \cup\{x\} \subseteq$ support-set $(p(x:=p r))$
proof
fix $y$
assume $y \in$ support-set $p \cup\{x\}$
show $y \in \operatorname{support-set}(p(x:=p r))$
proof (cases $y \in$ support-set $p$ )
case True
thus ?thesis
proof -
have $f 1: y \in\{b . \exists n a . p b n a \neq 0\}$
by (metis True support-set-def)
then have $y \neq x$

```
                    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\mathrm{ using « }y\in\mathrm{ support-set }p\cup{x}> by aut
            thus ?thesis using assms by (simp add: support-set-def)
        qed
    qed
qed
lemma fix-asset-price:
    shows \existsx Mkt2. x & stocks Mkt ^
    coincides-on Mkt Mkt2 (stocks Mkt) ^
    prices Mkt2 x = pr
proof -
    have \existsx. x\not\in stocks Mkt by (metis UNIV-eq-I stk-strict-subs-def mkt-stocks-assets)
    from this obtain x where x }\not=\mathrm{ 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>\langlex\not\in stocks Mkt\rangle by blast
qed
lemma (in disc-market-pos-stock) arbitrage-portfolio-properties:
assumes price-structure der matur \(\pi\) pr
and replicating-portfolio pf der matur
and (coincides-on Mkt Mkt2 (stocks Mkt))
and (prices Mkt2 \(x=p r\) )
and \(x \notin\) stocks Mkt
and diff-inv \(=(\pi-\) initial-value \(p f) /\) constant-image (prices Mkt pos-stock 0\()\)
and diff-inv \(\neq 0\)
and arb-pf \(=\left(\lambda\left(x::^{\prime} b\right)(n:: n a t)\left(w::^{\prime} a\right)\right.\). \(0::\) real \()(x:=(\lambda n w .-1)\), pos-stock \(:=\) ( \(\lambda\) n w. diff-inv))
and contr-pf \(=q t y\)-sum arb-pf pf
```

shows self－financing Mkt2 contr－pf
and trading－strategy contr－pf
and $\forall w \in$ space $M$ ．cls－val－process Mkt2 contr－pf $0 w=0$
and $0<$ diff－inv $\longrightarrow(A E w$ in M． $0<$ cls－val－process Mkt2 contr－pf matur w）
and diff－inv $<0 \longrightarrow(A E w$ 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 $\operatorname{simp}$
hence $\exists c . \forall w \in$ space $M$ ．prices Mkt pos－stock $0 w=c$ using adapted－init $[$ of prices Mkt pos－stock］by simp
moreover have $\forall w \in$ space M． $0<$ prices Mkt pos－stock $0 w$ using positive by $\operatorname{simp}$
ultimately show ？thesis using constant－image－pos by simp
qed
show support－set arb－pf $=\{x$ ，pos－stock $\}$
proof－
have arb－pf $=\left(\lambda\left(x::^{\prime} b\right)(n:: n a t)\left(w::^{\prime} a\right) .0::\right.$ real $)(x:=(\lambda n w .-1)$ ，pos－stock $:=(\lambda n w . d i f f-i n v))$
using $<a r b-p f=\left(\lambda\left(x::^{\prime} b\right)(n:: n a t)\left(w::^{\prime} a\right) .0:: r e a l\right)(x:=(\lambda n w .-1)$ ，pos－stock $:=(\lambda n w$ ．diff－inv $))$ ）．
moreover have $\exists n w$ ．diff－inv $\neq 0$ using assms by simp
moreover have $x \neq$ pos－stock using $\langle x \notin$ 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：$\forall n$ 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 $n w *(\lambda n$ w．diff－inv）
$n w+$ prices Mkt2 $x n w *(\lambda n w .-1) n w$
proof（rule two－component－val－process）
show $x \neq$ pos－stock using $\langle x \notin$ stocks Mkt〉 in－stock by auto
show arb－pf $=(\lambda x n w .0)(x:=\lambda a b$ ．-1 ，pos－stock $:=\lambda a b$ ．diff－inv）using assms by simp
show portfolio arb－pf using＜portfolio arb－pf〉 by simp
show $\exists n w$ ．$-(1::$ real $) \neq 0$ by simp
show $\exists n w$ ．diff－inv $\neq 0$ using assms by auto
qed
also have $\ldots=$ prices Mkt2 pos－stock $n w *$ diff－inv－pr n w using $\prec p r i c e s$ Mkt2 $x=p r>$ by $\operatorname{simp}$
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 }\in\mathrm{ support-set arb-pf
    thus \existsc. arb-pf asset = ( }\lambdan\mathrm{ 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 \in 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 = (\lambdax n w.0) (x:= \lambdan
w. - 1, pos-stock:= \lambdan w. diff-inv)>)
    hence arb-uvp: \foralln 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\subseteq stocks Mkt \cup{x}
    proof -
    have support-set contr-pf \subseteq support-set arb-pf \cup support-set pf using assms
        by (simp add:sum-support-set)
        moreover have support-set arb-pf \subseteq stocks Mkt \cup{x} using<support-set
arb-pf ={x, pos-stock }> in-stock by simp
    moreover have support-set pf \subseteqstocks Mkt \cup{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))\rangle\langle(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 = (\lambdax n w.0) (x:= \n
```

```
w. -1, pos-stock:= \lambdan 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 }\forallw\in\mathrm{ space M. cls-val-process Mkt2 contr-pf 0 w = 0
    proof
    fix w
    assume w\in 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 coin-
cides-on-def by blast
    have prices Mkt pos-stock 0 w = constant-image (prices Mkt pos-stock 0)
                using }\langlew\in\mathrm{ space M> adapted-init constant-imageI pos-stock-borel-adapted
by blast
        then show ?thesis
            using f1 by simp
    qed
    also have ... = ( }\pi-\mathrm{ 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 ... = ( }\pi-\mathrm{ initial-value pf ) - }\pi+\mathrm{ cls-val-process Mkt2 pf 0 w using
<price-structure der matur \pi pr>
            price-structure-init[of der matur \pi pr] by (simp add: <w \in space M>)
    also have ... = ( }\pi-\mathrm{ initial-value pf) - }\pi+(\mathrm{ initial-value pf) using initial-valueI
assms unfolding replicating-portfolio-def
    using }\langlew\in\mathrm{ space M> coincides-stocks-cls-val-process self-financingE readable
    by (metis (no-types, opaque-lifting) support-adapt-def stock-portfolio-def sub-
setCE)
    also have ... = 0 by simp
    finally show cls-val-process Mkt2 contr-pf 0 w = 0 .
```


## qed

show $0<$ diff－inv $\longrightarrow(A E w$ in M． $0<$ cls－val－process Mkt2 contr－pf matur $w)$ proof
assume $0<$ diff－inv
show $A E$ w in M． $0<$ cls－val－process Mkt2 contr－pf matur $w$
proof（rule AE－mp）
have $A E w$ in M．prices Mkt2 x matur $w=\operatorname{der} w$ using «price－structure der matur $\pi$ pr〉〈prices Mkt2 $x=p r\rangle$
unfolding price－structure－def by auto
moreover have $A E w$ in M．cls－val－process Mkt2 pf matur $w=\operatorname{der} w$ using assms coincides－stocks－cls－val－process［of Mkt pf Mkt2］

## 〈coincides－on Mkt Mkt2（stocks Mkt）〉 unfolding replicating－portfolio－def

 by autoultimately show $A E$ 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 $w$
$\longrightarrow 0<$ cls－val－process Mkt2 contr－pf matur $w$
proof（rule AE－I2，rule impI）
fix $w$
assume $w \in$ 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 $\ldots=$ prices Mkt2 pos－stock matur $w *$ diff－inv－prices Mkt2 x matur $w+$ cls－val－process Mkt2 pf matur $w$
using 〈prices Mkt2 $x=p r\rangle$ by $\operatorname{simp}$
also have ．．．$=$ prices Mkt2 pos－stock matur $w *$ diff－inv using 〈prices Mkt2 x matur $w=$ cls－val－process Mkt2 pf matur w
by $\operatorname{simp}$
also have $\ldots>0$ using positive $\langle 0<$ 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 $<0 \longrightarrow(A E w$ in M． $0>$ cls－val－process Mkt2 contr－pf matur $w)$
proof
assume diff－inv $<0$
show $A E w$ in $M .0>$ cls－val－process Mkt2 contr－pf matur $w$
proof（rule AE－mp）
have $A E w$ in M．prices Mkt2 $x$ matur $w=$ der $w$ using 〈price－structure der matur $\pi$ pr〉 $\langle$ prices Mkt2 $x=p r>$
unfolding price－structure－def by auto
moreover have $A E$ win M．cls－val－process Mkt2 pf matur $w=\operatorname{der} w$ 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 $A E w$ in M．prices Mkt2 x matur $w=$ cls－val－process Mkt2 pf matur $w$ by auto
show $A E$ w in M．prices Mkt2 x matur $w=$ cls－val－process Mkt2 pf matur $w$ $\longrightarrow 0>$ cls－val－process Mkt2 contr－pf matur $w$
proof (rule AE-I2, rule impI)
fix $w$
assume $w \in$ 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 $\ldots=$ 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 $\ldots=$ prices Mkt2 pos－stock matur $w *$ diff－inv－prices Mkt2 x matur $w+$ cls－val－process Mkt2 pf matur $w$
using $\langle p$ rices Mkt2 $x=p r\rangle$ by simp
also have $\ldots=$ prices Mkt2 pos－stock matur $w *$ diff－inv using $\langle$ prices Mkt2 $x$ matur $w=$ cls－val－process Mkt2 pf matur $w$＞
by $\operatorname{simp}$
also have $\ldots<0$ using positive $\langle$ 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 $n \in$ borel－measurable（F n）
and portfolio pf
and qty $n \in$ borel－measurable（ $F n$ ）
and $0 \neq n$
shows cls－val－process Mkt2（qty－mult－comp pf qty）$n \in$ borel－measurable（F $n$ ）
proof－
have $\exists m . n=$ Suc $m$ using assms by presburger
from this obtain $m$ where $n=S u c m$ by auto
hence cls－val－process Mkt2（qty－mult－comp pf qty）（Suc m）$\in$ 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 $\langle n=$ Suc $m\rangle$ by presburger
thus ?thesis using $\langle n=$ Suc $m\rangle$ by simp qed
lemma (in disc-equity-market) mult-comp-cls-val-process-measurable:
assumes $\forall n$. cls-val-process Mkt2 pf $n \in$ borel-measurable ( $F n$ )
and portfolio pf
and $\forall n$. qty $(S u c n) \in$ borel-measurable ( $F n$ )
shows $\forall n$. cls-val-process Mkt2 (qty-mult-comp pf qty) $n \in$ borel-measurable ( $F n$ )
proof
fix $n$
show cls-val-process Mkt2 (qty-mult-comp pf qty) $n \in$ borel-measurable ( $F$ n )
proof (cases $n=0$ )
case False
hence $\exists m$. $n=$ Suc $m$ by presburger
from this obtain $m$ where $n=S u c m$ by auto
have qty $n \in$ borel-measurable ( $F n$ )
using Suc-n-not-le-n $\langle n=$ Suc $m\rangle$ assms(3) increasing-measurable-info nat-le-linear by blast
hence qty (Suc m) $\in$ borel-measurable $(F(S u c m))$ using $\langle n=S u c m\rangle$ by simp
hence cls-val-process Mkt2 (qty-mult-comp pf qty) (Suc m) $\in$ 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 $\langle n=$ Suc $m\rangle$ by presburger
thus ?thesis using $\langle n=$ 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 $0 \in$ borel-measurable (F 0) using assms by $\operatorname{simp}$
ultimately have ( $\lambda w$. cls-val-process Mkt2 pf $0 w * q t y(S u c ~ 0) w) \in$ borel-measurable (F0) by $\operatorname{simp}$
thus ?thesis using assms(2) True mult-comp-cls-val-process0
by (simp add: $\langle(\lambda w$. cls-val-process Mkt2 pf $0 w * q t y($ Suc 0) $w) \in$ 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 $n \in$ borel-measurable ( $F n$ )
and portfolio pf
and qty (Suc n) $\in$ borel-measurable (F n)
shows val－process Mkt2（qty－mult－comp pf qty）$n \in$ 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 $\pi$ der matur
shows $\pi=$ initial－value $p f$
proof－
have expr：（ $\exists$ pr．price－structure der matur $\pi$ pr $\wedge$
（ $\forall$ x．$(x \notin$ stocks Mkt $\longrightarrow$
$(\forall$ Mkt2 $p$ ．（coincides－on Mkt Mkt2 $($ stocks Mkt $)) \wedge($ prices Mkt2 $x=p r) \wedge$
portfolio $p \wedge$ support－set $p \subseteq$ stocks Mkt $\cup\{x\} \longrightarrow$
$\neg$ arbitrage－process Mkt2 p））））using assms fair－priceI by simp
then obtain $p r$ where price－structure der matur $\pi$ pr and
xasset：$(\forall x$ ．$(x \notin$ stocks $M k t \longrightarrow$
$(\forall$ Mkt2 p．（coincides－on Mkt Mkt2 $($ stocks Mkt）$) \wedge($ prices Mkt2 $x=p r) \wedge$
portfolio $p \wedge$ support－set $p \subseteq$ stocks Mkt $\cup\{x\} \longrightarrow$
$\neg$ arbitrage－process Mkt2 $p$ ）））by auto
define diff－inv where diff－inv $=(\pi-$ initial－value $p f) /$ constant－image（prices Mkt pos－stock 0）
\｛
fix $x$
assume $x \notin$ stocks Mkt
hence mkprop：（ $\forall$ Mkt2 $p$ ．（coincides－on Mkt Mkt2（stocks Mkt））$\wedge$（prices Mkt2 $x=p r) \wedge$ portfolio $p \wedge$ support－set $p \subseteq$ stocks Mkt $\cup\{x\} \longrightarrow$
$\neg$ arbitrage－process Mkt2 $p$ ）using asasset by simp
fix Mkt2
assume（coincides－on Mkt Mkt2（stocks Mkt））and（prices Mkt2 $x=p r$ ）
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 $\operatorname{simp}$
hence $\exists c . \forall w \in$ space $M$ ．prices Mkt pos－stock $0 w=c$ using adapted－init［of prices Mkt pos－stock］by simp
moreover have $\forall w \in$ space $M .0<$ prices Mkt pos－stock $0 w$ using positive by $\operatorname{simp}$
ultimately show ？thesis using constant－image－pos by simp qed
define arb－pf where $a r b-p f=\left(\lambda\left(x::^{\prime} b\right)\right.$（ $\left.n:: n a t\right)\left(w::^{\prime} a\right) .0::$ real $)(x:=(\lambda n w$. $-1)$ ，pos－stock $:=(\lambda n$ w．diff－inv $))$
define contr－pf where contr－pf＝qty－sum arb－pf pf
have 1：0 $\neq$ diff－inv $\longrightarrow$ self－financing Mkt2 contr－pf
using arbitrage－portfolio－properties［of der matur $\pi$ pr pf Mkt2 $x$ diff－inv arb－pf contr－pf］
using 〈coincides－on Mkt Mkt2（stocks Mkt）〉〈price－structure der matur $\pi$ pr〉 $\langle$ prices Mkt2 $x=p r\rangle$
$\langle x \notin$ stocks Mkt〉 arb－pf－def assms（1）contr－pf－def diff－inv－def by blast
have 2：0 $=$ diff－inv $\longrightarrow$ trading－strategy contr－pf
using arbitrage－portfolio－properties［of der matur $\pi$ pr pf Mkt2 $x$ diff－inv arb－pf contr－pf］
using 〈coincides－on Mkt Mkt2（stocks Mkt）〉〈price－structure der matur $\pi$ pr〉 〈prices Mkt2 $x=p r\rangle$
$\langle x \notin$ stocks Mkt＞arb－pf－def assms（1）contr－pf－def diff－inv－def by blast
have 3：0 0 diff－inv $\longrightarrow(\forall w \in$ space $M$ ．cls－val－process Mkt2 contr－pf $0 w=0)$
using arbitrage－portfolio－properties［of der matur $\pi$ pr pf Mkt2 $x$ diff－inv arb－pf contr－pf］
using 〈coincides－on Mkt Mkt2（stocks Mkt）〉〈price－structure der matur $\pi$ $p r\rangle\langle p r i c e s$ Mkt2 $x=p r\rangle$
$\langle x \notin$ stocks Mkt＞arb－pf－def assms（1）contr－pf－def diff－inv－def by blast
have $4: 0<$ diff－inv $\longrightarrow(A E w$ in M． $0<$ cls－val－process Mkt2 contr－pf matur w）
using arbitrage－portfolio－properties［of der matur $\pi$ pr pf Mkt2 $x$ diff－inv arb－pf contr－pf］ using 〈coincides－on Mkt Mkt2（stocks Mkt）〉〈price－structure der matur $\pi$ $p r\rangle\langle p r i c e s$ Mkt2 $x=p r\rangle$
$\langle x \notin$ stocks Mkt〉 arb－pf－def assms（1）contr－pf－def diff－inv－def by blast
have 5：diff－inv $<0 \longrightarrow(A E w$ in M． $0>$ cls－val－process Mkt2 contr－pf matur w）
using arbitrage－portfolio－properties［of der matur $\pi$ pr pf Mkt2 $x$ diff－inv arb－pf contr－pf］
using 〈coincides－on Mkt Mkt2（stocks Mkt）〉〈price－structure der matur $\pi$ pr〉〈prices Mkt2 $x=p r\rangle$ $\langle x \notin$ stocks Mkt〉 arb－pf－def assms（1）contr－pf－def diff－inv－def by blast
have $6: 0 \neq$ diff－inv $\longrightarrow$ support－set arb－pf $=\{x$ ，pos－stock $\}$
using arbitrage－portfolio－properties［of der matur $\pi$ pr pf Mkt2 $x$ diff－inv arb－pf contr－pf］
using 〈coincides－on Mkt Mkt2（stocks Mkt）〉〈price－structure der matur $\pi$ pr〉 $\langle$ prices Mkt2 $x=p r\rangle$ $\langle x \notin$ stocks Mkt〉 arb－pf－def assms（1）contr－pf－def diff－inv－def by blast
have 7： $0 \neq$ diff－inv $\longrightarrow$ support－set contr－pf $\subseteq$ stocks Mkt $\cup\{x\}$
proof－
have $0 \neq$ diff－inv $\longrightarrow$ support－set contr－pf $\subseteq$ support－set arb－pf $\cup$ support－set $p f$ unfolding contr－pf－def by（simp add：sum－support－set）
moreover have $0 \neq$ diff－inv $\longrightarrow$ support－set arb－pf $\subseteq$ stocks Mkt $\cup\{x\}$ using $\langle 0 \neq$ diff－inv $\longrightarrow$ support－set arb－pf $=\{x$, pos－stock $\}\rangle$ in－stock by simp moreover have $0 \neq$ diff－inv $\longrightarrow$ support－set $p f \subseteq$ stocks Mkt $\cup\{x\}$ using assms unfolding replicating－portfolio－def stock－portfolio－def by auto ultimately show ？thesis by auto
qed
have $8: 0 \neq$ diff－inv $\longrightarrow$ portfolio contr－pf
using arbitrage－portfolio－properties［of der matur $\pi$ pr pf Mkt2 $x$ diff－inv arb－pf contr－pf］ using 〈coincides－on Mkt Mkt2（stocks Mkt）〉〈price－structure der matur $\pi$
pr〉〈prices Mkt2 $x=p r\rangle$
$\langle x \notin$ stocks Mkt〉 arb－pf－def assms（1）contr－pf－def diff－inv－def by blast
have 9：0 $\neq$ diff－inv $\longrightarrow$ cls－val－process Mkt2 contr－pf matur $\in$ borel－measurable （ $F$ matur）
proof
assume $0 \neq$ diff－inv
have $10: \forall$ asset $\in$ support－set arb－pf $\cup$ support－set pf．prices Mkt2 asset matur $\in$ borel－measurable（ $F$ matur）

## proof

fix asset
assume asset $\in$ support－set arb－pf $\cup$ support－set pf
show prices Mkt2 asset matur $\in$ borel－measurable（ $F$ matur）
proof（cases asset $\in$ 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 $\in$ support－set arb－pf using＜asset $\in$ support－set arb－pf $\cup$ support－set pf＞by auto
show ？thesis
proof（cases asset $=x$ ）
case True
thus ？thesis
using 〈price－structure der matur $\pi$ pr〉〈prices Mkt2 $x=p r\rangle$ price－structure－borel－measurable by blast
next
case False
hence asset $=$ pos－stock using $\langle$ asset $\in$ support－set arb－pf $\rangle\langle 0 \neq$ diff－inv $\longrightarrow$ support－set arb－pf $=\{x$, pos－stock $\}>$

$$
\langle 0 \neq \text { diff-inv> by auto }
$$

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 $\forall$ asset $\in$ support－set contr－pf．contr－pf asset matur $\in$ borel－measurable （ $F$ matur）
using $\langle 0 \neq$ diff－inv $\longrightarrow$ trading－strategy contr－pf〉〈0 $\neq$ diff－inv $\rangle$
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 $\in$ borel－measurable（ $F$ matur）
proof－
have $\forall$ asset $\in$ support－set contr－pf．contr－pf asset（Suc matur $) \in$ borel－measurable
using $\langle 0 \neq$ diff-inv $\longrightarrow$ trading-strategy contr-pf〉〈0 $\neq$ diff-inv>
by (simp add: predict-stoch-proc-def trading-strategy-def)
moreover have $\forall$ asset $\in$ support-set contr-pf. prices Mkt2 asset matur $\in$
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) $1\langle 0 \neq$ diff-inv〉
quantity-adapted self-financingE)
qed
qed
\{
assume $0>$ diff-inv
define opp-pf where opp-pf $=q t y$-mult-comp contr-pf $(\lambda n w .-1)$
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 $2\langle 0>$ diff-inv〉 by auto
show borel-predict-stoch-proc F ( $\lambda n w .-1)$ by (simp add: constant-process-borel-predictable)
qed
show $\forall w \in$ space $M$. cls-val-process Mkt2 opp-pf $0 w=0$
proof
fix $w$
assume $w \in$ space $M$
show cls-val-process Mkt2 opp-pf $0 w=0$ using $38\langle 0>$ diff-inv〉
using $\langle w \in$ space $M\rangle$ 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 $A E$ w in M. $0>$ cls-val-process Mkt2 contr-pf matur $w$ using $5<0$
$>$ diff-inv> by auto
show $A E w$ in M. cls-val-process Mkt2 contr-pf matur $w<0 \longrightarrow 0<$
cls-val-process Mkt2 opp-pf matur w
proof
fix $w$
assume $w \in$ space $M$
show cls-val-process Mkt2 contr-pf matur $w<0 \longrightarrow 0<$ 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 $\exists m$. 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〉 $\langle w \in$ space $M\rangle$ mult－comp－cls－val－process－Suc opp－pf－def using 〈cls－val－process Mkt2 contr－pf matur $w<0\rangle$ by fastforce thus ？thesis using «Suc $m=$ matur〉 by simp next case True
thus ？thesis using $38\langle 0>$ diff－inv〉 $\langle w \in$ space $M\rangle$ 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 $A E w$ in $M .0 \leq$ cls－val－process Mkt2 opp－pf matur $w$ by auto
show $0<\operatorname{prob}\{w \in \operatorname{space} M .0<$ cls－val－process Mkt2 opp－pf matur $w\}$
proof－
let $? P=\{w \in$ space $M .0<$ cls－val－process Mkt2 opp－pf matur $w\}$
have cls－val－process Mkt2 opp－pf matur $\in$ borel－measurable（F matur）
proof－
have cls－val－process Mkt2 contr－pf matur $\in$ borel－measurable（F matur）
using $9\langle 0>$ diff－inv〉 by simp
moreover have portfolio contr－pf using $8\langle 0\rangle$ diff－inv〉 by simp
moreover have $(\lambda w .-1) \in$ 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 $\neq 0$
using 〈diff－inv＜0〉 by blast
then have self－financing Mkt2 contr－pf
by（metis 1）
then show ？thesis
by（metis（no－types）$\langle(\lambda w .-1) \in$ borel－measurable（F matur）$\rangle$
〈portfolio contr－pf〉
〈self－financing Mkt2 opp－pf〉〈cls－val－process Mkt2 contr－pf matur $\in$ 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 ？$P \in$ sets（ $F$ matur）using borel－measurable－iff－greater $[$ of val－process Mkt2 contr－pf matur F matur］
by auto
hence $? P \in$ sets $M$ by（meson filtration filtration－def subalgebra－def subsetCE）
hence measure $M ? P=1$ using prob－Collect－eq－ $1[$ of $\lambda x .0<$ cls－val－process Mkt2 opp－pf matur $x$ ］
$\langle A E w$ in M． $0<$ cls－val－process Mkt2 opp－pf matur $w\rangle\langle 0>$ diff－inv〉 by blast

```
            thus?thesis by simp
            qed
    qed
    have \exists p. portfolio p}\wedge support-set p\subseteq stocks Mkt \cup{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\subseteq stocks Mkt \cup{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 }\forallw\in\mathrm{ space M. cls-val-process Mkt2 contr-pf 0 w = 0 using 3<0<
                diff-inv> by auto
            show AE w in M. 0 \leqcls-val-process Mkt2 contr-pf matur w using 4 <0
        < diff-inv> by auto
            show 0< prob {w\in space M. O< cls-val-process Mkt2 contr-pf matur w}
        proof -
            let ?P = {w\in space M. 0 < cls-val-process Mkt2 contr-pf matur w}
            have cls-val-process Mkt2 contr-pf matur }\in\mathrm{ 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 \in sets (F matur) using borel-measurable-iff-greater[of
                val-process Mkt2 contr-pf matur F matur]
                by auto
                hence ?P \in sets M by (meson filtration filtration-def subalgebra-def
subsetCE)
                            hence measure M ?P = 1 using prob-Collect-eq-1[of \lambdax. 0 <
cls-val-process Mkt2 contr-pf matur x]
                    4<0< diff-inv\rangle by blast
                thus?thesis by simp
            qed
            qed
                            have \exists p. portfolio p}\wedge\mathrm{ support-set p}\subseteq\mathrm{ stocks Mkt U{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 \subseteq stocks Mkt \cup {x} using 7 <0 < diff-inv>
by auto
            qed
    } note posp = this
    have diff-inv }\not=0\longrightarrow\neg(\exists\mathrm{ pr. price-structure der matur }\pi\mathrm{ pr ^ ^
        ( }\forall\mathrm{ x. (x& stocks Mkt }
            (\forall Mkt2 p. (coincides-on Mkt Mkt2 (stocks Mkt)) ^(prices Mkt2 x = pr)
\wedge ~ p o r t f o l i o ~ p ~ \wedge ~ s u p p o r t - s e t ~ p \subseteq ~ s t o c k s ~ M k t ~ \cup \{ x \} \longrightarrow
            \neg arbitrage-process Mkt2 p))))
            using <coincides-on Mkt Mkt2 (stocks Mkt)\rangle\langleprices Mkt2 x = pr><x\not\in
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 $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a \Rightarrow$ real where

```
rfr-base: (disc-rfr-proc r) 0w = 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 $r$ ) $n \in$ 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 $\bigwedge n$. disc-rfr-proc $r n \in$ borel-measurable (F 0) using disc-rfr-proc-borel-measurable by auto
lemma (in disc-equity-market) disc-rfr-constant-time:
shows $\exists c . \forall w \in \operatorname{space}(F 0)$. (disc-rfr-proc rn) $w=c$
proof (rule triv-measurable-cst)
show space ( $\left.\begin{array}{ll} & 0\end{array}\right)=$ space $M$ using filtration by (simp add:filtration-def subal-gebra-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 $n$ ) $\in$ borel-measurable ( $F$ O) using disc-rfr-proc-nonrandom by blast
show space $M \neq\{ \}$ by (simp add:not-empty)
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 \nw.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 $\exists c>0 . \forall w \in$ space $M .($ disc-rfr-proc $r n) w=c$
proof -
let $? F=\operatorname{sigma}($ space $M)\{\{ \}$, space $M\}$
have ex: $\exists c . \forall w \in$ space ? $F$. (disc-rfr-proc rn) $w=c$
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 $r n$ ) $\in$ borel-measurable ?F using disc-rfr-proc-borel-measurable
by blast
show space $M \neq\{ \}$ 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 $\exists w . w \in$ space $M$ using subprob-not-empty by blast

```
    from this obtain w where w\in 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 = (\lambdaw. 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 b borel-measurable M by (simp add:disc-rfr-proc-borel-measurable)
qed
lemma discount-factor-init:
    shows discount-factor r 0 = (\lambdaw.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 b borel-measurable M by (simp add:disc-rfr-proc-borel-measurable)
qed
lemma discount-factor-positive:
    assumes -1<r
    shows \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 $r$ sk-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 $r f r$-disc-equity-market $=$ disc-equity-market $+r s k$-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 accept-able-rate))

### 7.3.2 Discounted value of a stochastic process

definition discounted-value where
discounted-value $r X=(\lambda n w$. 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 dis-counted-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 $r(\lambda m x . X m x * Y m x) n w=X n w *$ (discounted-value $r Y) n w$
by (simp add: discounted-value-def)
lemma discounted-mult-times-rfr:
assumes $-1<r$
shows discounted-value $r(\lambda m w .(1+r) * X w)($ Suc $n) w=$ discounted-value $r$ $(\lambda m w . X w) n w$
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 \(\forall n w . X n w=Y n w\)
    shows \(\forall n w\). 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 \(A E w\) in \(N . X n w=Y n w\)
    shows \(A E w\) in \(N\). discounted-value \(r X n w=\) discounted-value \(r Y n w\)
proof (rule \(A E-m p\) )
    show \(A E w\) in \(N . X n w=Y n w\) using assms by simp
    show \(A E w\) in \(N . X n w=Y n w \longrightarrow\) discounted-value \(r X n w=\) discounted-value
\(r Y n w\)
    proof
        fix \(w\)
        assume \(w \in\) space \(N\)
        thus \(X n w=Y n w \longrightarrow\) 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 $\forall n w$. ( $\sum i \in I$. (discounted-value $r(\lambda m x$.fimx) $n w)=($ discounted-value
$\left.r\left(\lambda m x .\left(\sum i \in I . f i m x\right)\right) n w\right)$
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 $\left(\sum i \in\right.$ insert a $F$. discounted-value $\left.r(f i) n w\right)=$ discounted-value $r(f a)$
$n w+\left(\sum i \in F\right.$. discounted-value $\left.r(f i) n w\right)$
by (simp add: insert.hyps(1) insert.hyps(3))
also have $\ldots=$ discounted-value $r(f a) n w+$ discounted-value $r$ ( $\lambda m x . \sum i \in F$.
fimx) n w using insert.hyps(4) by simp
also have $\ldots=$ discounted-value $r\left(\lambda m x\right.$. $\sum i \in$ insert a F. fimx) n w
by (simp add: discounted-value-def insert.hyps(1) insert.hyps(3) ring-class.ring-distribs(1))
finally show ( $\sum i \in$ insert a $F$. discounted-value $\left.r(f i) n w\right)=$ discounted-value
$r\left(\lambda m x . \sum i \in\right.$ insert $a F$. fimx) $n w$.
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 $r X t \in$ borel-measurable ( $F t$ ) unfolding discounted-value-def proof (rule borel-measurable-times)
show $X t \in$ borel-measurable ( $F t$ ) using assms unfolding adapt-stoch-proc-def by $\operatorname{simp}$
show discount-factor $r t \in$ borel-measurable ( $F t$ ) using discount-factor-borel-measurable by auto
qed
qed
lemma discounted-measurable:
assumes $X \in$ borel-measurable $N$
shows discounted-value $r(\lambda m . X) m \in$ borel-measurable $N$ unfolding dis-
counted-value-def
proof (rule borel-measurable-times)
show $X \in$ borel-measurable $N$ using assms by simp
show discount-factor $r m \in$ borel-measurable $N$ using discount-factor-borel-measurable
by auto
qed
lemma (in prob-space) discounted-integrable:
assumes integrable $N$ ( $\mathrm{X} n$ )
and $-1<r$
and space $N=$ space $M$
shows integrable $N$ (discounted-value $r X n$ ) unfolding discounted-value-def
proof -
have $\exists c>0 . \forall w \in$ 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 $\forall w \in$ 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 $N(\lambda w$. discount-factor $r n w * X n w)$
using $\langle\forall w \in$ space $N$. discount-factor $r n w=c\rangle$ assms

Bochner-Integration.integrable-cong[of $N N(\lambda w$. discount-factor r $n w * X n$ w) $(\lambda w . c * X n w)]$ by $\operatorname{simp}$
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 $M$ ( $\lambda w$. prices Mkt asset $n w * p f$ 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 $n=\left(\lambda w . \sum x \in\right.$ support-set pf. prices Mkt x $n w * p f x$ (Suc n) w)
unfolding val-process-def by simp
moreover have integrable $M\left(\lambda w\right.$. $\sum x \in$ support-set pf. prices Mkt x $n w * p f x$ (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 $M$ ( $\lambda w$. prices Mkt asset $n w * p f$ 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 $N(\lambda w$. prices Mkt asset (Suc n) w

* pf asset (Suc n) w)
shows $\forall x \in$ support-set $p f$. $A E w$ in $N$.
(real-cond-exp $N(F n)$ (discounted-value $r(\lambda m y$. prices Mkt x m y * pf $x$
my) $($ Suc $n))) w=$
discounted-value $r(\lambda m y$. prices Mkt $x m y * p f x($ 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 NF 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 $\langle a s s e t \in$ stocks Mkt〉 borel-adapt-stoch-proc-borel-measurable dis-counted-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 $m y * p f$ asset $m y)($ Suc $n))$ )
(discounted-value $r$ ( $\lambda m$ y. prices Mkt asset $m y * p f$ asset (Suc m) y) n)
proof -
have $A E w$ in $N$. (real-cond-exp $N(F n)$ (discounted-value $r(\lambda m y$. prices Mkt asset $m y * p f$ asset $m y)($ Suc $n))) w=$
(real-cond-exp $N(F n)(\lambda z . p f$ asset $(S u c n) z *$ discounted-value $r(\lambda m y$. prices Mkt asset my) (Suc n) z)) w
proof (rule sigma-finite-subalgebra.real-cond-exp-cong)
show sigma-finite-subalgebra $N(F n)$ using nsigfin ..
show $A E w$ in $N$. discounted-value $r$ ( $\lambda m y$. prices Mkt asset $m y * p f$ asset my) $($ 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 $r$ ( $\lambda m$ y. prices Mkt asset $m y * p f$ asset $m y$ ) (Suc n) $\in$ borel-measurable $N$
proof -
have ( $\lambda$. prices Mkt asset (Suc n) $y * \operatorname{pf}$ asset (Suc $n$ ) $y$ ) $\in$ borel-measurable
using assms 〈asset $\in$ 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 ( $\lambda z$. pf asset (Suc n) $z *$ discounted-value $r$ (prices Mkt asset) (Suc $n$ )
$z) \in$ borel-measurable $N$
proof -
have $p f$ asset (Suc $n$ ) $\in$ borel-measurable $M$ using assms «asset $\in$ support-set $p f\rangle$ unfolding trading-strategy-def
using borel-predict-stoch-proc-borel-measurable[of pf asset] by auto
hence $a$ : pf asset (Suc n) $\in$ borel-measurable $N$ using assms Condi-tional-Expectation.measurable-from-subalg[of $N$ M - borel] by blast
show ?thesis using $a b$ by simp
qed
qed
also have $A E w$ in $N$. (real-cond-exp $N(F n)(\lambda z$. pf asset (Suc n) $z *$ discounted-value $r(\lambda m y$. prices Mkt asset $m y)(S u c n) z)) w=$
$p f$ asset $(S u c n) w *($ real-cond-exp $N(F n)$ ( $\lambda z$. discounted-value $r(\lambda m y$. 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) $(S u c n) \in$ borel-measurable $N$ using $b$ by simp
show sigma-finite-subalgebra $N$ ( $F n$ ) using nsigfin ..
show pf asset (Suc $n$ ) $\in$ borel-measurable $(F n)$ using assms «asset $\in$ sup-port-set pf> unfolding trading-strategy-def
predict-stoch-proc-def by auto
show integrable $N$ ( $\lambda z$. pf asset (Suc n) $z *$ discounted-value $r$ (prices Mkt asset) (Suc n) z)
proof -
have integrable $N(\lambda w$. prices Mkt asset (Suc n) w * pf asset (Suc n) w) using assms <asset $\in$ support-set pf > by auto
hence integrable $N$ (discounted-value $r$ ( $\lambda m$ w. prices Mkt asset $m w * p f$ 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 $A E w$ in $N$. pf asset (Suc $n$ ) $w *($ real-cond-exp $N(F n)(\lambda z$. discounted-value $r$ ( $\lambda m$ y. prices Mkt asset $m$ y) $($ Suc $n) z)$ ) $w=$
$p f$ asset (Suc n) $w$ * discounted-value $r$ ( $\lambda m$ y. prices Mkt asset $m y$ ) $n w$
proof -
have $A E e q N($ real-cond-exp $N(F n)(\lambda z$. discounted-value $r(\lambda m y$. prices Mkt asset $m$ y) (Suc n) z))
( $\lambda z$. discounted-value $r(\lambda m y$. prices Mkt asset $m y) n z)$
proof -
have martingale NF (discounted-value r (prices Mkt asset))
using assms 〈asset $\in$ 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 NF discounted－value $r$ （prices Mkt asset）n Suc n］assms
by $\operatorname{simp}$
qed
thus ？thesis by auto
qed
also have $A E w$ in $N$ ．pf asset（Suc n）$w *$ discounted－value $r$（ $\lambda m$ y．prices Mkt asset $m$ y）$n w=$
discounted－value $r(\lambda m y$ ．pf asset（Suc m）$y *$ prices Mkt asset $m y) n w$ by （simp add：discounted－value－def）
also have $A E w$ in $N$ ．discounted－value $r$（ $\lambda m y$ ．pf asset（Suc m）y＊prices Mkt asset $m y$ ）$n w=$
discounted－value $r(\lambda m y$ ．prices Mkt asset $m y * p f$ asset（Suc m）y）$n w$
by（simp add：discounted－value－def）
finally show $A E$ w in $N$ ．
（real－cond－exp $N(F n)$（discounted－value $r$（ $\lambda m$ y．prices Mkt asset $m y * p f$ asset $m y)($ Suc $n))) w=$
（ $\lambda x$ ．discounted－value $r(\lambda m y$ ．prices Mkt asset $m y * p f$ 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 $\forall n . \forall$ asset $\in$ support－set $p f$ ．integrable $N(\lambda w$ ．prices Mkt asset $n w * p f$ asset（Suc n）w）
and $\forall n . \forall$ asset $\in$ support－set pf．integrable $N(\lambda w$ ．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：$\forall n$ ．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 NF 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 \(\forall m\). discounted-value \(r\) (cls-val-process Mkt pf) \(m \in\) borel-measurable ( \(F\)
\(m\) ) unfolding adapt-stoch-proc-def by simp
    show \(\forall t\). 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: inte-
grable-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 \(\forall n\). AE win \(N\). 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 \(A E w\) 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 \(w \in\) space \(M\)
        have discounted-value \(r\) (cls-val-process Mkt pf) (Suc n) \(w=\)
                                    discount-factor \(r\) (Suc \(n\) ) \(w *\left(\sum x \in\right.\) support-set pf. prices Mkt \(x\)
(Suc n) \(w * p f x(\) Suc \(n) w)\)
            unfolding discounted-value-def cls-val-process-def using assms unfolding
trading-strategy-def by simp
            also have \(\ldots=\left(\sum x \in\right.\) support-set \(p f\). discount-factor \(r\) (Suc \(n\) ) \(w *\) prices
Mkt \(x\) (Suc n) \(w * p f x(\) 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) \(w=\)
```

                            ( \(\sum\) x support-set pf. discount-factor r (Suc n) w * prices Mkt \(x\)
    (Suc n) w*pfx(Suc n)w).
\}
hence space: $\forall w \in$ space M. discounted-value $r$ (cls-val-process Mkt pf) (Suc
n) $w=$
( $\sum x \in$ support-set $p f$. discount-factor $r($ Suc $n$ ) $w *$ prices Mkt $x$ (Suc
n) $w * p f x($ Suc $n) w)$ by simp
hence nspace: $\forall w \in$ space $N$. discounted-value $r$ (cls-val-process Mkt pf) (Suc
n) $w=$
( $\sum x \in$ support-set pf. discount-factor $r$ (Suc n) $w$ * prices Mkt $x$ (Suc
n) $w * p f x(S u c n) w)$ 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 $r(\lambda m y$. prices Mkt x m y * pf $x$
$m y)($ Suc $n))) w=$
discounted-value $r(\lambda m y$. prices Mkt $x m y * p f x(S u c m) y) n w$ using
assms
by (simp add:stocks-portfolio-risk-neutral filt-equiv-imp-subalgebra stock-portfolio-def)
have $A E w$ in $N$. real-cond-exp $N(F n)$ (discounted-value $r$ (cls-val-process Mkt pf) (Suc n)) $w=$
real-cond-exp $N(F n)\left(\lambda y . \sum x \in\right.$ support-set $p f$. discounted-value $r$
( $\lambda m$ y. prices Mkt $x m y * p f x m y)($ 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) $w=$
( $\lambda y$. $\sum x \in$ support-set $p f$. discounted-value $r(\lambda m y$. prices Mkt $x m y * p f$ $x m y)($ 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 $\langle\forall t$. integrable $N$ (discounted-value $r($ cls-val-process $M k t p f) t)\rangle$ by blast
qed
also have $A E w$ in $N$. real-cond- $\exp N(F n)$
( $\lambda y$. $\sum x \in$ support-set $p f$. discounted-value $r$ ( $\lambda m$ y. prices Mkt x my*pfx $m y)($ Suc $n) y) w=$
( $\sum x \in$ support-set $p f$. (real-cond-exp $N(F n)$ (discounted-value $r(\lambda m y$. prices Mkt $x m y * p f x m y)(S u c n))) w$ )
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 $r$ ( $\lambda m$ y. prices Mkt asset $m y * p f$ asset $m$ y) (Suc n))
proof (rule discounted-integrable)
show integrable $N$ ( $\lambda y$. prices Mkt asset (Suc n) $y * p f$ 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 $A E$ win $N$.
( $\sum x \in$ support-set pf. (real-cond-exp $N(F n)$ (discounted-value $r(\lambda m y$. prices Mkt x m $y * p f x m y)(S u c n))) w)=$
( $\sum x \in$ support-set pf. discounted-value $r(\lambda m y$. prices Mkt $x m y * p f x$ (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. $A E w$ in $N$.
(real-cond-exp $N(F n)$ (discounted-value $r(\lambda m y$. prices Mkt x m y *pfx
m y) $($ Suc $n))) w=$
discounted-value $r(\lambda m y$. prices Mkt $x m y * p f x(S u c m) y) n w$ using
qed
also have $A E$ in $N$.
( $\sum x \in$ support-set $p f$. discounted-value $r(\lambda m y$. prices Mkt $x m y * p f x$ (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 $x m y * p f$ $x($ Suc m) y) $n w)=$
discounted-value $r$ ( $\lambda m$ y. ( $\sum x \in$ support-set pf. prices Mkt $x m y * p f x$ (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 $\ldots=$ discounted-value $r$ (val-process Mkt pf) $n$ w unfolding val-process-def
by (simp add: portfolio-def)
also have $\ldots=$ discounted-value $r($ cls-val-process Mkt pf) $n$ wing assms by (simp add:self-financingE discounted-cong)
finally show ( $\sum x \in$ support-set $p f$. discounted-value $r$ ( $\lambda m$ y. prices Mkt $x$ $m y * p f x($ Suc $m) y) n w)=$
discounted-value $r$ (cls-val-process Mkt pf) $n w$.
qed
finally show $A E w$ 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 $p f$. finite ( $p f$ asset $n$ '(space $M)$ )
and prob-space $N$
and filt-equiv $F M N$
and trading-strategy $p f$
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 $N(\lambda w$. prices Mkt asset $n w * p f$ 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 $n w * \operatorname{pf}$ 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 $n w * p f$ asset (Suc n) w)'space M)
proof -
have $\forall y \in$ prices Mkt asset $n$＇（space M）．finite（ $(\lambda z .(\lambda w . z * p f$ asset（Suc n）$w)$＇space $M$ ）$y$ ）
by（metis «asset $\in$ support－set pf〉assms（2）finite－imageI image－image）
hence finite $(\bigcup y \in$ prices Mkt asset $n$＇（space M）．（（ $\lambda z .(\lambda w . z * p f$ asset （Suc n）w）＇space M）y））
using＜asset $\in$ support－set pf〉assms by blast
moreover have $(\bigcup y \in$ prices Mkt asset $n$＇$($ space $M) .((\lambda z .(\lambda w . z * p f$ asset $($ Suc $n) w)$＇space $M) y))=$
$(\bigcup y \in$ prices Mkt asset $n$＇（space M）．（ $\lambda w . y * p f$ asset（Suc n）w）＇space M）by $\operatorname{simp}$
moreover have（（ $\lambda w$ ．prices Mkt asset $n w * p f$ asset（Suc n）w）＇space M） $\subseteq$
$(\bigcup y \in$ prices Mkt asset $n$＇（space $M) .(\lambda w . y * p f$ asset $($ Suc $n) w)$＇space M）
proof
fix $x$
assume $x \in(\lambda w$ ．prices Mkt asset $n w * p f$ asset（Suc n）w）＇space M
show $x \in(\bigcup y \in$ prices Mkt asset $n$＇space M．（ $\lambda w . y * p f$ asset（Suc n）w） ＇space M）
using $\langle x \in(\lambda w$ ．prices Mkt asset $n w * p f$ 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 $n w * p f$ asset（Suc n）$w)$＇space $N$ ）using assms by（simp add：filt－equiv－space）
have（ $\lambda w$ ．prices Mkt asset $n w * p f$ asset $(S u c n) w) \in$ borel－measurable $M$
proof－
have prices Mkt asset $n \in$ borel－measurable $M$ using assms $\langle$ asset $\in$ support－set $p f>$ by simp
moreover have pf asset（Suc $n$ ）$\in$ borel－measurable $M$ using assms unfolding trading－strategy－def
using $\langle a s s e t \in$ support－set pf〉borel－predict－stoch－proc－borel－measurable by blast
ultimately show？？thesis by simp
qed
thus（ $\lambda w$ ．prices Mkt asset $n w * p f$ asset（Suc $n$ ）$w) \in$ borel－measurable $N$ using assms by（simp add：filt－equiv－measurable）
qed
qed
lemma（in disc－filtr－prob－space）finite－integrable－uvp：
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 $N(\lambda w$ ．prices Mkt asset（Suc $n$ ）w ＊pf asset（Suc n）w）
proof（intro allI ballI）
fix $n$
fix asset
assume asset $\in$ 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（（ $\lambda z .(\lambda w . z$＊pf asset （Suc n）w）＇space M）y）
by（metis $\langle$ asset $\in$ support－set $p f\rangle$ assms（2）finite－imageI image－image）
hence finite $(\bigcup y \in$ prices Mkt asset（Suc n）＇＇space M）．（（ $\lambda z .(\lambda w . z * p f$ asset（Suc n）w）＇space M）y））
using＜asset $\in$ support－set $p f\rangle$ assms 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）．（ $\lambda w . y * p f$ 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）．（ $\lambda w . y * \operatorname{pf}$ asset（Suc n）w）＇ space $M$ ）

## proof

fix $x$
assume $x \in(\lambda w$ ．prices Mkt asset（Suc n）w＊pf asset（Suc n）w）＇space
show $x \in(\bigcup y \in$ prices Mkt asset（Suc n）＇space M．（ $\lambda w . y * p f$ asset（Suc n）w）＇space $M$ ）
using $\langle x \in(\lambda w$ ．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 $n$ ）$w$ ）$\in$ borel－measurable M
proof－
have prices Mkt asset（Suc $n$ ）$\in$ borel－measurable $M$ using assms
using «asset $\in$ support－set pf〉borel－adapt－stoch－proc－borel－measurable by blast
moreover have pf asset（Suc $n$ ）$\in$ borel－measurable $M$ using assms unfolding trading－strategy－def
using 〈asset $\in$ support－set pf〉borel－predict－stoch－proc－borel－measurable by blast
ultimately show ？thesis by simp
qed
thus $(\lambda w$. prices Mkt asset $($ Suc $n) w * p f$ asset $(S u c n) w) \in$ 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 $p f$
and self-financing Mkt pf
and support-set pf $\subseteq$ stocks Mkt
and $\forall n . \forall$ asset $\in$ support-set pf. finite (prices Mkt asset $n$ '(space $M)$ )
and $\forall n . \forall$ asset $\in$ support-set $p f$. finite ( $p f$ asset $n$ '(space $M)$ )
and $\forall$ asset $\in$ 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 $\forall n$. $\forall$ asset $\in$ support-set $p f$. integrable $N$ ( $\lambda w$. prices Mkt asset $n w * p f$ 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 $n w * p f$ 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 $n w * p f$ asset (Suc n) w)' space M)
proof -
have $\forall y \in$ prices Mkt asset $n$ ' space M). finite ( $(\lambda z .(\lambda w . z * p f$ asset (Suc n) $w$ ) ' space $M$ ) $y$ )
by (metis $\langle$ asset $\in$ support-set $p f\rangle \operatorname{assms}(7)$ finite-imageI image-image)
hence finite $(\bigcup y \in$ prices Mkt asset $n$ ' $($ space $M) .((\lambda z .(\lambda w . z * p f$ asset
(Suc n)w)' space M) y))
using <asset $\in$ support-set pf>assms(6) by blast
moreover have $(\bigcup y \in$ prices $M k t$ asset $n$ ' (space $M) .((\lambda z .(\lambda w . z * p f$ asset $(S u c n) w) '$ space $M) y))=$
$(\bigcup y \in$ prices Mkt asset $n$ '(space $M) .(\lambda w . y * \operatorname{pf}$ asset (Suc n)w)'space M) by $\operatorname{simp}$
moreover have (( $\lambda w$. prices Mkt asset $n w * p f$ asset (Suc n) w)' space
$M) \subseteq$
$(\bigcup y \in$ prices Mkt asset $n$ '(space $M) .(\lambda w . y * p f$ asset $(S u c n) w)$ 'space M)
proof
fix $x$
assume $x \in(\lambda w$. prices Mkt asset $n w * p f$ asset (Suc n) w)' space $M$
show $x \in(\bigcup y \in$ prices Mkt asset $n$ ' space M. ( $\lambda w . y * p f$ asset (Suc n)
w) 'space $M$ )
using $\langle x \in(\lambda w$. prices Mkt asset $n w * p f$ asset (Suc n) w)'space M>
by auto

```
            qed
            ultimately show ?thesis by (simp add:finite-subset)
    qed
    thus finite ((\lambdaw. prices Mkt asset n w * pf asset (Suc n) w)` space N) using
assms by (simp add:filt-equiv-space)
    have (\lambdaw. prices Mkt asset n w* pf asset (Suc n) w) \in borel-measurable M
    proof -
        have prices Mkt asset n \in borel-measurable M using assms readable
            using <asset \in support-set pf`borel-adapt-stoch-proc-borel-measurable by
blast
            moreover have pf asset (Suc n) \in borel-measurable M using assms
unfolding trading-strategy-def
                using <asset \in support-set pf> borel-predict-stoch-proc-borel-measurable by
blast
            ultimately show ?thesis by simp
        qed
        thus (\lambdaw. prices Mkt asset n w * pf asset (Suc n) w) \in borel-measurable N
using assms by (simp add:fil-equiv-measurable)
    qed
    qed
    show }\foralln.\forall\mathrm{ asset }\in\mathrm{ support-set pf. integrable N (\w. prices Mkt asset (Suc n) w
* pf asset (Suc n) w)
    proof (intro allI ballI)
        fix n
        fix asset
        assume asset\insupport-set pf
    show integrable N (\lambdaw. 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 ((\lambdaw. prices Mkt asset (Suc n)w*pf asset (Suc n)w)'space M)
            proof -
                have }\forally\in\mathrm{ prices Mkt asset (Suc n) '(space M). finite (( }\lambda\mathrm{ z. ( \w.z * pf
asset (Suc n) w)` space M) y)
                by (metis «asset \in support-set pf> assms(7) finite-imageI image-image)
            hence finite (\bigcupy\in prices Mkt asset (Suc n) '(space M). ((\lambdaz. (\lambdaw.z*pf
asset (Suc n) w)' space M) y))
                using <asset \in support-set pf> assms(6) by blast
                moreover have (U y\in prices Mkt asset (Suc n) '(space M). ((\lambdaz. (\lambdaw.z
* pf asset (Suc n) w) ' space M) y)) =
                            (U y\in prices Mkt asset (Suc n)'(space M). (\lambdaw. y* pf asset (Suc n)w)
' space M) by simp
                                    moreover have ((\lambdaw. prices Mkt asset (Suc n) w * pf asset (Suc n)w)`
space M)\subseteq
                (U y\in prices Mkt asset (Suc n)'(space M). (\lambdaw. y * pf asset (Suc n) w)
    ` space M)
            proof
                fix }
                    assume x ( (\lambdaw. prices Mkt asset (Suc n) w* pf asset (Suc n) w)'space
M
```

show $x \in(\bigcup y \in$ prices Mkt asset（Suc n）＇space M．（ $\lambda w . y * p f$ asset（Suc n）w）＇space $M$ ）
using $\langle x \in(\lambda w$ ．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）$\in$ borel－measurable M

## proof－

have prices Mkt asset（Suc n）$\in$ borel－measurable $M$ using assms readable using 〈asset $\in$ support－set $p f$ 〉borel－adapt－stoch－proc－borel－measurable by
blast
moreover have pf asset（Suc $n$ ）$\in$ borel－measurable $M$ using assms unfolding trading－strategy－def
using 〈asset $\in$ support－set pf〉borel－predict－stoch－proc－borel－measurable by blast
ultimately show ？thesis by simp
qed
thus（ $\lambda w$ ．prices Mkt asset（Suc n）$w * p f$ asset（Suc $n$ ）$w) \in$ 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 $\forall n . \forall$ asset $\in$ support－set pf．integrable $N$（ $\lambda w$ ．prices Mkt asset $n w * p f$ asset（Suc n）w）
and $\forall n . \forall$ asset $\in$ support－set pf．integrable $N$（ $\lambda w$ ．prices Mkt asset（Suc $n$ ）$w$
＊pf asset（Suc n）w）
and viable－market Mkt
and sets $(F 0)=\{\{ \}$ ，space $M\}$
and pyf $\in$ borel－measurable（ $F$ matur）
shows fair－price Mkt（prob－space．expectation $N$（discounted－value r（ $\lambda$ m．pyf）matur）） pyf matur
proof－
have fn：filtrated－prob－space NF using assms
by（simp add：$\langle p y f \in$ borel－measurable（ $F$ matur）$\rangle$ 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 $\in$ 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 $r$ ( $\lambda m$. pyf) matur $\in$ borel-measurable $N$
proof -
have ( $\lambda m$. pyf) matur $\in$ borel-measurable ( $F$ matur) using assms by simp
hence ( $\lambda m$. pyf) matur $\in$ borel-measurable $M$ using filtration filtrationE1
measurable-from-subalg by blast
hence ( $\lambda m$. pyf) matur $\in$ 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 $w=p y f w$ using assms
unfolding replicating-portfolio-def by simp
have AE w in N. cls-val-process Mkt pf matur $w=p y f w$
proof (rule filt-equiv-borel-AE-eq)
show filt-equiv $F M N$ using assms by simp
show pyf $\in$ borel-measurable ( $F$ matur) using assms by simp
show $A E$ w in M. cls-val-process Mkt pf matur $w=p y f$ w using mpyf by simp show cls-val-process Mkt pf matur $\in$ 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 $r$ ( $\lambda m$. pyf) matur $w$
by (simp add:discounted-AE-cong)
have AEeq $N$ (real-cond-exp $N$ (F O) (discounted-value r (cls-val-process Mkt pf) matur))
(real-cond-exp $N$ ( $F$ 0) (discounted-value $r(\lambda m$. pyf) matur $)$ )
proof (rule sigma-finite-subalgebra.real-cond-exp-cong)
show sigma-finite-subalgebra $N$ ( $\left.\begin{array}{l}\text { 0 }\end{array}\right)$
using filtrated-prob-space.axioms(1) filtrated-prob-space.filtration fn filtra-
tionE1
prob-space.subalgebra-sigma-finite by blast
show $A$ Eeq $N$ (discounted-value $r$ (cls-val-process Mkt pf) matur) (discounted-value $r$ ( $\lambda m$. pyf) matur) using disc by simp
show discounted-value r (cls-val-process Mkt pf) matur $\in$ borel-measurable $N$ using <discounted-value $r$ (cls-val-process Mkt pf) matur $\in$ borel-measurable
$N$ >.
show discounted-value $r$ ( $\lambda$ m. pyf) matur $\in$ borel-measurable $N$
using 〈discounted-value $r$ ( $\lambda$ m. pyf) matur $\in$ 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 $N p f]$ by (simp add: stock-portfolio-def)
hence AEeq $N$ (real-cond-exp $N$ (F O) (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) O matur]
    fn by simp
    also have AE w in N. (discounted-value r (cls-val-process Mkt pf) 0 w)= ini-
tial-value pf
    proof
    fix }
    assume w\in 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\in 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 \forallw\in 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 (F0)={{}, 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 (\lambdam. pyf) matur)
    proof (rule integral-cong-AE)
    show AEeq N (discounted-value r (cls-val-process Mkt pf) matur) (discounted-value
r (\lambdam. pyf) matur)
        using disc by simp
    show discounted-value r (\lambdam.pyf) matur }\in\mathrm{ borel-measurable N
        using <discounted-value r (\lambdam. pyf) matur }\in\mathrm{ borel-measurable N>.
    show discounted-value r (cls-val-process Mkt pf) matur \in borel-measurable N
        using <discounted-value r (cls-val-process Mkt pf) matur \in borel-measurable
N>.
    qed
    ultimately have prob-space.expectation N (discounted-value r (\lambdam. 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 $\forall n . \forall$ asset $\in$ support-set $p f$. finite (prices Mkt asset $n$ '(space $M)$ )
and $\forall n . \forall$ asset $\in$ support-set $p f$. finite ( $p f$ asset $n$ '(space $M$ ))
and viable-market Mkt
and sets $(F 0)=\{\{ \}$, space $M\}$
and pyf $\in$ borel-measurable ( $F$ matur)
shows fair-price Mkt (prob-space.expectation $N$ (discounted-value r ( $\lambda$ m. pyf) matur))
pyf matur
proof -
have $\forall n . \forall$ asset $\in$ support-set pf. integrable $N(\lambda w$. prices Mkt asset $n w * p f$
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 $\bigwedge n$ asset. asset $\in$ support-set $p f \Longrightarrow$ random-variable borel (prices Mkt
asset $n$ )
proof -
fix $n$
fix asset
assume asset $\in$ 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 $\in$ support-set pf〉adapt-stoch-proc-borel-measurable sub-
setCE)

```
    qed
    qed
    moreover have }\foralln.\forall\mathrm{ asset }\in\mathrm{ support-set pf. integrable N ( }\lambdaw\mathrm{ . 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 \in support-set pf \Longrightarrow random-variable borel (prices Mkt
asset n)
    proof -
            fix n
            fix asset
            assume asset \in 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 }\in\mathrm{ support-set pf` adapt-stoch-proc-borel-measurable sub-
setCE)
    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 $C R R$-hyps $=$ prob-grw $+r s k$-free-asset +
fixes stk
assumes stocks: stocks Mkt $=\{$ stk, risk-free-asset $\}$
and stk-price: prices $M k t$ 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 $\in$ stocks Mkt
shows asset $=$ stk $\vee$ asset $=$ risk-free-asset
proof (rule ccontr)
assume $\neg($ asset $=$ stk $\vee$ asset $=$ risk-free-asset $)$
hence asset $\neq$ stk $\wedge$ asset $\neq$ risk-free-asset by simp
moreover have asset $\in\{$ 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: $\forall n$. 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 subal-
gebra-def)
qed
show $\forall n$. sigma-finite-subalgebra $N(G n)$ using assms unfolding subalgebra-def using filtration-def subalgebra-sigma-finite
by (metis $\langle$ Filtration.filtration $N G\rangle$ bernoulli-stream-def prob-space.prob-space-stream-space prob-space.subalgebra-sigma-finite prob-space-measure-pmf)
qed
sublocale $C R R$-market $\subseteq$ rfr-disc-equity-market - $G$
proof (unfold-locales)
show disc-filtr $M G \wedge$ sets $(G \perp)=\{\{ \}$, space $M\}$
proof
show sets $(G \perp)=\{\{ \}$, 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 $\forall$ asset $\in$ 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 $\langle$ disc-filtr $M G \wedge$ sets $(G \perp)=\{\{ \}$, 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 $M G \wedge$ sets $(G \perp)=\{\{ \}$, space M\}〉)
qed
ultimately show ?thesis using stocks by force
qed
qed

```
lemma (in CRR-market) two-stocks:
shows stk \(\neq\) risk-free-asset
proof (rule ccontr)
    assume \(\neg\) stk \(\neq\) risk-free-asset
    hence disc-rfr-proc \(r=\) prices Mkt stk using rf-price by simp
    also have \(\ldots=\) geom-proc using stk-price by simp
    finally have eqf: disc-rfr-proc \(r=\) geom-proc.
    hence \(\forall w\). disc-rfr-proc r \(0 w=\) geom-proc \(0 w\) by simp
    hence \(1=\) init using geometric-process by simp
    have eqfs: \(\forall w\). 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 \(\operatorname{simp}\)
    hence \(1+r=u\) using geometric-process \(\langle 1=\) init \(\rangle\) 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 \(\langle 1=\) init \(\rangle\) by simp
    show False using \(\langle 1+r=u\rangle\langle 1+r=d\rangle\) 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 $n w * p f$ stk (Suc n) $w+$ disc-rfr-proc rnw*pf risk-free-asset (Suc n) w
proof -
have val-process Mkt pf $n w=(\operatorname{sum}(\lambda x$. ((prices Mkt) x $n w) *(p f x(S u c n)$ w)) ( stocks Mkt))
proof (rule subset-val-process')
show finite (stocks Mkt) using stocks by auto
show support-set pf $\subseteq$ stocks Mkt using assms unfolding stock-portfolio-def by $\operatorname{simp}$
qed
also have $\ldots=\left(\sum x \in\{\right.$ stk, risk-free-asset $\}$. ((prices Mkt) x n w) * (pf x (Suc n) w)) using stocks by simp
also have $\ldots=$ prices Mkt stk $n w * p f$ stk (Suc n) $w+$
$\left(\sum x \in\{\right.$ risk-free-asset $\} .(($ prices Mkt) $x n w) *(p f x($ Suc $n) w))$ by (simp add:two-stocks)
also have $\ldots=$ prices Mkt stk $n w * p f$ stk (Suc n) $w+$
prices Mkt risk-free-asset $n w * p f$ risk-free-asset (Suc n) w by simp
also have $\ldots=$ geom-proc $n w * p f$ stk (Suc n) $w+$ disc-rfr-proc r n w $w$ 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) $w * p f$ 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 $=(\operatorname{sum}(\lambda x$. ((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 $\subseteq$ stocks Mkt using assms unfolding stock-portfolio-def
by $\operatorname{simp}$
qed
also have $\ldots=\left(\sum x \in\{s t k\right.$, risk-free-asset $\} .(($ prices $M k t) x($ Suc $n) w) *(p f x$ (Suc n) w)) using stocks by simp
also have $\ldots=$ prices Mkt stk (Suc n) $w * p f$ stk (Suc n) $w+$
$\left(\sum x \in\{\right.$ risk-free-asset $\} .(($ prices $M k t) x($ Suc n) w) $)(p f x($ Suc n) w) $)$ by (simp add:two-stocks)
also have $\ldots=$ 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 $\ldots=$ geom-proc (Suc $n$ ) $w * p f$ 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 $0<p f$ stk (Suc n) (spick w $n$ False)
and val-process Mkt pf $n$ (spick $w n$ False) $\leq 0$
shows cls-val-process Mkt pf (Suc n) (spick w n False) $<0$
proof -
define $w n f$ where $w n f=$ 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 $\ldots=d *$ 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 $\ldots=d *$ geom-proc $n$ wnf $* p f$ stk (Suc n) wnf $+(1+r) *$ disc-rfr-proc $r n w n f$ * pf risk-free-asset (Suc n) wnf
by $\operatorname{simp}$
also have $\ldots<(1+r) *$ 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 SO-positive
down-positive down-lt-up by simp
also have $\ldots=(1+r) *($ geom-proc $n$ wnf $* p f$ stk (Suc n) wnf + disc-rfr-proc $r n$ wnf * pf risk-free-asset (Suc n) wnf)
by (simp add: distrib-left)
also have $\ldots=(1+r) *$ val-process Mkt pf $n$ wnf using stock-pf-vp-expand assms
by simp
also have $\ldots \leq 0$
proof -
have $0<1+r$ using assms down-positive by simp
moreover have val-process Mkt pf $n$ wnf $\leq 0$ using assms unfolding wnf-def by $\operatorname{simp}$
ultimately show $(1+r) *($ val-process Mkt pf $n$ wnf $) \leq 0$ unfolding wnf-def
using less-eq-real-def[of $01+r$ ] 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 $p f$ stk (Suc n) (spick wn True) $<0$
and val-process Mkt pf $n$ (spick $w n$ True) $\leq 0$
shows cls-val-process Mkt pf (Suc n) (spick w $n$ True) $<0$
proof -
define $w n f$ where $w n f=$ 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 $\ldots=u *$ 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 $\ldots=u *$ 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 $\operatorname{simp}$
also have $\ldots<(1+r) *$ 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 SO-positive down-positive down-lt-up by simp
also have $\ldots=(1+r) *($ geom-proc $n$ wnf $*$ pf stk (Suc n) wnf + disc-rfr-proc $r n w n f * p f$ risk-free-asset (Suc n) wnf)
by (simp add: distrib-left)
also have $\ldots=(1+r) *$ val-process Mkt pf $n$ wnf using stock-pf-vp-expand assms by $\operatorname{simp}$
also have $\ldots \leq 0$
proof -
have $0<1+r$ using acceptable-rate by simp
moreover have val-process Mkt pf $n$ wnf $\leq 0$ using assms unfolding wnf-def by $\operatorname{simp}$
ultimately show $(1+r) *($ val-process Mkt pf $n$ wnf $) \leq 0$ unfolding wnf-def using less-eq-real-def[of $01+r]$ 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 \(n w \leq 0\)
shows cls-val-process Mkt pf (Suc n) \(w<0\)
proof -
    have cls-val-process Mkt pf (Suc n) \(w=\)
        \(S(S u c n) w * p f\) 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 $\ldots=$ disc-rfr-proc $r$ (Suc n) $w * p f$ risk-free-asset (Suc n) w using assms by simp
also have $\ldots=(1+r) *$ disc-rfr-proc $r n w * p f$ risk-free-asset $(S u c n) w$ by simp
also have ... $<0$
proof -
have $0<1+r$ using acceptable-rate by simp
moreover have $0<$ disc-rfr-proc $r n$ wing acceptable-rate by (simp add: disc-rfr-proc-positive)
ultimately have $0<(1+r)$ * disc-rfr-proc r $n$ w by simp
have 1: $0<p f$ risk-free-asset $(S u c n) w \longrightarrow 0<(1+r) *$ disc-rfr-proc r $n w *$ pf risk-free-asset (Suc n) w
proof (intro impI)
assume $0<p f$ risk-free-asset (Suc n) w
thus $0<(1+r) *$ disc-rfr-proc r $n w * p f$ risk-free-asset (Suc n) w using $\langle 0<(1+r) *$ disc-rfr-proc $r n$ w
by $\operatorname{simp}$
qed
have 2: pf risk-free-asset (Suc n) $w<0 \longrightarrow(1+r) *$ disc-rfr-proc r $n w * p f$ risk-free-asset (Suc n) w<0
proof (intro impI)
assume $p f$ 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 $\langle 0<(1+r) *$ disc-rfr-proc $r n w\rangle$
by (simp add:mult-pos-neg)
qed
have $0 \geq$ val-process Mkt pf $n w$ using assms by simp
also have val-process Mkt pf $n w=$ geom-proc $n w * p f$ stk (Suc n) $w+$
disc-rfr-proc $r n w * p f$ risk-free-asset (Suc n) w using assms by (simp add:stock-pf-vp-expand)
also have $\ldots=$ disc-rfr-proc r n w $\quad$ * pf risk-free-asset (Suc n) w using assms by $\operatorname{simp}$
finally have $0 \geq$ disc-rfr-proc r n w $w$ pf risk-free-asset (Suc n) w.
have $0<p f$ risk-free-asset (Suc n) $w \vee p f$ risk-free-asset (Suc n) $w<0$ using assms
by linarith
thus ?thesis
using 2 $\langle 0<$ disc-rfr-proc r $n$ w〉〈disc-rfr-proc r $n w * p f$ risk-free-asset (Suc n) $w \leq 0$ > 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 $p f$
and $1+r<u$
and $d<1+r$
and val-process Mkt pf $n w \leq 0$
and $p f$ stk (Suc n) $w \neq 0 \vee p f$ risk-free-asset (Suc n) $w \neq 0$
shows $\exists y$.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 $\in$ support-set $p f \cup\{$ stk $\}$ by simp
qed
hence pf stk (Suc n) borel-measurable ( $G$ n) unfolding predict-stoch-proc-def
by $\operatorname{simp}$
have val-process Mkt pf $n \in$ 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 $w n$ where $w n=$ pseudo-proj-True $n w$
show ?thesis
proof (cases pf stk (Suc $n$ ) $w \neq 0$ )
case True
show ?thesis
proof (cases pf stk (Suc n) w>0)
case True
have $0<p f$ stk (Suc n) (spick wn n False)
proof -
have $0<p f$ stk (Suc n) w using $\langle 0<p f$ stk (Suc n) w〉 by simp
also have $\ldots=p f$ stk (Suc n) wn unfolding wn-def
using $\langle p f$ stk $(S u c n) \in$ borel-measurable $(G n)\rangle$ 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...$=p f$ stk (Suc n) (spick wn n False) using $\langle p f$ stk (Suc n) $\in$ 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 $0 \geq$ val-process Mkt pf $n$ (spick wn $n$ False)
proof -
have $0 \geq$ 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 $n \in$ 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 mea-surable-from-subalg)
also have $\ldots=$ val-process Mkt pf $n$ (spick wn $n$ False) using «val-process Mkt pf $n \in$ 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 $\exists y$. cls-val-process Mkt pf (Suc n) $y<0$ by auto
next
case False
have $0>p f$ stk (Suc n) (spick wn $n$ True)
proof -
have $0>p f$ stk (Suc n) w using $\prec \neg 0<p f$ stk (Suc n) w〉<pf stk (Suc n) $w \neq 0$ > by $\operatorname{simp}$
also have pf stk (Suc n) w $=p f$ stk (Suc n) wn unfolding wn-def using $\langle p f$ stk $(S u c n) \in$ borel-measurable $(G n)\rangle$
nat-filtration-info stoch-proc-subalg-nat-filt[of geom-proc] geometric-process stock-filtration by (metis comp-apply geom-rand-walk-borel-adapted mea-surable-from-subalg)
also have $\ldots=p f$ stk (Suc $n$ ) (spick wn $n$ True) using $\langle p f$ stk (Suc $n) \in$ 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 $0 \geq$ val-process Mkt pf $n$ (spick wn $n$ True)
proof -
have $0 \geq$ 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 $n \in$ 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 $\ldots=$ val-process Mkt pf $n$ (spick wn $n$ True) using «val-process Mkt pf $n \in$ borel-measurable ( $G n$ 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 \existsy.cls-val-process Mkt pf (Suc n) y<0 by auto
    qed
    next
    case False
    hence pf risk-free-asset (Suc n) w}\not=0\mathrm{ 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 \existsy.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 }=
and stock-portfolio Mkt pf
shows pf stk (Suc n) y}\not=0\veepf risk-free-asset (Suc n) y\not=
proof (rule ccontr)
    assume }\neg(pf stk (Suc n) y = 0 \vee pf risk-free-asset (Suc n) y f=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\not=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+
    and cls-val-process Mkt pf n w<0
shows n\leqm\Longrightarrow\existsy.cls-val-process Mkt pf m y<0
proof (induct m)
    case 0
    assume n\leq0
    hence n=0 by simp
    thus \existsy.cls-val-process Mkt pf 0 y < 0 using assms by auto
next
    case (Suc m)
    assume n\leqSuc m
```

```
thus \existsy.cls-val-process Mkt pf (Suc m) y<0
proof (cases n<Suc m)
    case False
    hence n=Suc m using <n\leqSuc m> by simp
    thus \existsy.cls-val-process Mkt pf (Suc m) y<0 using assms by auto
next
    case True
    hence }n\leqm\mathrm{ by simp
    hence \existsy.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 \leq0 by simp
    have val-process Mkt pf my\not=0 using <val-process Mkt pf m y<0> by simp
    hence pf stk (Suc m) y}\not=0\veepf risk-free-asset (Suc m) y =0 using assms
non-zero-components by simp
    thus \existsy.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 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 $\neg(\forall p$. stock-portfolio Mkt $p \longrightarrow \neg$ arbitrage-process Mkt $p)$
hence $\exists$ p. stock-portfolio Mkt $p \wedge$ arbitrage-process Mkt $p$ by simp
from this obtain pf where stock-portfolio Mkt pf and arbitrage-process Mkt pf
by auto
have $(\exists$ m. (self-financing Mkt pf) $) \wedge($ trading-strategy pf $) \wedge$
$(\forall w \in$ space $M$. cls-val-process Mkt pf $0 w=0) \wedge$
( $A E$ w in $M .0 \leq$ cls-val-process Mkt pf $m w) \wedge$
$0<\mathcal{P}(w$ in $M$. cls-val-process Mkt pf $m w>0))$ using $<$ arbitrage-process $M k t$
$p f>$
using arbitrage-processE by simp
from this obtain $m$ where self-financing Mkt pf and (trading-strategy pf)
and $(\forall w \in$ space $M$. cls-val-process Mkt pf $0 w=0)$
and $(A E w$ in $M .0 \leq c l s$-val-process Mkt pf $m w)$
and $0<\mathcal{P}(w$ in $M$. cls-val-process Mkt pf $m w>0)$ by auto
have $\{w \in$ space M. cls-val-process Mkt pf $m w>0\} \neq\{ \}$ using
$\langle 0<\mathcal{P}(w$ in $M$. cls-val-process Mkt pf $m w>0)\rangle$ by force
hence $\exists w \in$ space $M$. cls-val-process Mkt pf $m w>0$ by auto
from this obtain $y$ where $y \in$ space $M$ and cls-val-process $M k t$ pf $m y>0$ by
auto
define $A$ where $A=\{n:: n a t . n \leq m \wedge$ cls-val-process Mkt pf $n y>0\}$
have finite $A$ unfolding $A$－def by auto
have $m \in A$ using 〈cls－val－process Mkt pf $m y>0\rangle$ unfolding $A$－def by simp
hence $A \neq\{ \}$ by auto
hence $\operatorname{Min} A \in A$ using $\langle$ finite $A\rangle$ by simp
have $\operatorname{Min} A \leq m$ using $\langle$ finite $A\rangle\langle m \in A\rangle$ by simp
have $0<\operatorname{Min} A$
proof－
have cls－val－process Mkt pf $0 y=0$ using $\langle y \in$ space $M\rangle\langle\forall w \in$ space $M$ ． cls－val－process Mkt pf $0 w=0$ 〉
by simp
hence $0 \notin A$ unfolding $A$－def by simp
moreover have $0 \leq \operatorname{Min} A$ by simp
ultimately show ？thesis using $\langle\operatorname{Min} A \in A\rangle$ neq0－conv by fastforce
qed
hence $\exists l$ ．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 $l y \leq 0$
proof－
have $l<\operatorname{Min} A$ using $\langle S u c l=\operatorname{Min} A\rangle$ by simp
hence $l \notin A$ using 〈finite $A\rangle\langle A \neq\{ \}\rangle$ by auto
moreover have $l \leq m$ using $\langle S u c l=\operatorname{Min} A\rangle\langle m \in A\rangle\langle$ finite $A\rangle\langle A \neq\{ \}\rangle\langle l$
$<\operatorname{Min} A$ by auto
ultimately show ？thesis unfolding $A$－def by auto
qed
hence val－process Mkt pf ly $\leq 0$ using 〈self－financing Mkt pf〉 by（simp add：self－financingE）
moreover have pf stk（Suc l）$y \neq 0 \vee p f$ risk－free－asset（Suc l）$y \neq 0$
proof（rule ccontr）
assume $\neg(p f$ stk $(S u c l) y \neq 0 \vee p f$ risk－free－asset $(S u c l) y \neq 0)$
hence pf stk（Suc l）$y=0$ pf risk－free－asset（Suc l）$y=0$ by auto
have cls－val－process Mkt pf（ $\operatorname{Min} A$ ）$y=$ geom－proc（Suc l）$y * p f s t k$（Suc l）
$y+$
disc－rfr－proc $r$（Suc l）$y * p f$ risk－free－asset（Suc l）y using «stock－portfolio Mkt pf＞

〈Suc $l=$ Min A〉stock－pf－uvp－expand $[$ of $p f l]$ by simp
also have $\ldots=0$ using $\langle p f$ stk（Suc l）$y=0\rangle\langle p f$ risk－free－asset（Suc l）$y=$ 0 ）by simp
finally have cls－val－process Mkt pf $(\operatorname{Min} A) y=0$ ．
moreover have cls－val－process Mkt pf $(\operatorname{Min} A) y>0$ using $\langle\operatorname{Min} A \in A\rangle$
unfolding $A$－def by simp
ultimately show False by simp
qed
ultimately have $\exists z$ ．cls－val－process Mkt pf（Suc l）z＜0 using assms $\langle$ 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 $\exists x^{\prime}$ ．cls－val－process Mkt pf $m x^{\prime}<0$ using neg－pf－Suc assms＜trad－ ing－strategy pf〉
$\langle$ self－financing Mkt pf〉〈Suc $l=\operatorname{Min} A\rangle\langle\operatorname{Min} A \leq m\rangle\langle$ stock－portfolio Mkt $p f>$ by $\operatorname{simp}$
from this obtain $x^{\prime}$ where cls－val－process Mkt pf $m x^{\prime}<0$ by auto have $x^{\prime} \in$ space $M$ using bernoulli－stream－space bernoulli by auto hence $x^{\prime} \in\{w \in$ space $M . \neg 0 \leq$ cls－val－process Mkt pf $m w\}$ using＜cls－val－process Mkt pf $\left.m x^{\prime}<0\right\rangle$ by auto
from $\langle A E w$ in $M .0 \leq$ cls－val－process $M k t p f m$ obtain $N$ where
$\{w \in$ space $M . \neg 0 \leq$ cls－val－process Mkt pf $m w\} \subseteq N$ and emeasure $M N=0$
and $N \in$ sets $M$ using $A E-E$ by auto
have $\left\{w \in\right.$ space $M .\left(\right.$ stake $m w=$ stake $\left.\left.m x^{\prime}\right)\right\} \subseteq N$
proof
fix $x$
assume $x \in\left\{w \in\right.$ space $M$ ．stake $m w=$ stake $\left.m x^{\prime}\right\}$
hence $x \in$ space $M$ and stake $m x=$ stake $m x^{\prime}$ by auto
have cls－val－process Mkt pf $m \in$ 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 $x^{\prime}=$ cls－val－process Mkt pf $m x$
using «stake $m x=$ stake $\left.m x^{\prime}\right\rangle$ 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 $m x^{\prime}<0$ 〉 by $\operatorname{simp}$
thus $x \in N$ using $\langle\{w \in$ space $M . \neg 0 \leq$ cls－val－process $M k t$ pf $m w\} \subseteq N\rangle\langle x \in$ space $M>$

〈cls－val－process Mkt pf（Suc l）z＜0〉 by auto
qed
moreover have emeasure $M\left\{w \in\right.$ space $M$ ．（stake $m w=$ stake $\left.\left.m x^{\prime}\right)\right\} \neq 0$ using bernoulli－stream－pref－prob－neq－zero psgt pslt by simp
ultimately show False using＜emeasure $M N=0\rangle\langle N \in$ events $\rangle$ 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 $\neg d<1+r$
hence $1+r \leq d$ by $\operatorname{simp}$
define arb－pf where arb－pf $=\left(\lambda\left(x::^{\prime} a\right)(n:: n a t) w .0:: r e a l\right)(s t k:=(\lambda n w .1)$ ，
risk－free－asset $:=(\lambda n w .-$ geom－proc $0 w))$
have support－set arb－pf $=\{$ stk，risk－free－asset $\}$
proof
show support－set arb－pf $\subseteq\{$ stk，risk－free－asset $\}$
by (simp add: arb-pf-def subset-iff support-set-def)
have stk $\in$ support-set arb-pf unfolding arb-pf-def support-set-def using two-stocks by $\operatorname{simp}$
moreover have risk-free-asset support-set arb-pf unfolding arb-pf-def sup-port-set-def
using two-stocks geometric-process S0-positive by simp
ultimately show $\{$ stk, risk-free-asset $\} \subseteq$ 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 $\}\rangle$
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 $\in$ support-set arb-pf
show borel-predict-stoch-proc G (arb-pf asset)
proof (cases asset $=s t k)$
case True
hence arb-pf asset $=\left(\begin{array}{ll}\lambda & n w\end{array}\right)$ unfolding arb-pf-def by (simp add: two-stocks)
show ?thesis unfolding predict-stoch-proc-def
proof
show arb-pf asset $0 \in$ borel-measurable ( $\begin{aligned} & \text { 0 }\end{aligned}$ ) using $\langle\operatorname{arb}-\mathrm{pf}$ asset $=(\lambda n$
w. 1)> by $\operatorname{simp}$
show $\forall n$. arb-pf asset $(S u c n) \in$ borel-measurable $(G n)$
proof
fix $n$
show arb-pf asset (Suc $n$ ) $\in$ borel-measurable ( $G n$ ) using $<$ arb-pf asset $=(\lambda n w .1)\rangle$ by $\operatorname{simp}$
qed
qed
next
case False
hence arb-pf asset $=(\lambda n w .-$ geom-proc $0 w)$ using <support-set arb-pf $=\{$ stk, risk-free-asset $\}\rangle$

〈asset $\in$ support-set arb-pf〉 unfolding arb-pf-def by simp
show ?thesis unfolding predict-stoch-proc-def
proof
 w. - geom-proc 0 w) >
geometric-process by simp
show $\forall n$. arb-pf asset $(S u c n) \in$ borel-measurable ( $G n$ )
proof

```
            fix n
            show arb-pf asset (Suc n) \in borel-measurable (G n) using <arb-pf asset
=(\lambdan w. - geom-proc 0 w)>
                        geometric-process by simp
            qed
            qed
            qed
    qed
    show }\forallw\in\mathrm{ space M. cls-val-process Mkt arb-pf 0 w = 0
    proof
            fix w
            assume w\in 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 0w = 0.
    qed
    have dev: }\forall\mathrm{ wG 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 }
            assume w\in 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
        <tock-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:}\forallw\in space M. snth w0 < cls-val-process Mkt arb-pf (Suc 0)w
>0
    proof (intro ballI impI)
            fix w
```

```
    assume w\in 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 \langlew\in 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 SO-positive <1 +r\leqd\rangle down-lt-up geometric-process
by auto
                            finally show cls-val-process Mkt arb-pf (Suc 0) w>0 .
    qed
    have iniF: }\forallw\in\mathrm{ space M. ᄀsnth w 0 }\longrightarrow\mathrm{ cls-val-process Mkt arb-pf (Suc 0)w
\geq0
    proof (intro ballI impI)
        fix }
        assume w\in space M and \negsnth 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 }\langlew\in\mathrm{ space M> by simp
            also have ... = d* geom-proc 0 w-(1+r)* geom-proc 0 w using }\neg\mathrm{ snth
w 0> geometric-process by simp
            also have ... = (d - (1+r)) * geom-proc 0 w by (simp add: left-diff-distrib)
            also have ...\geq0 using S0-positive <1 +r\leqd\rangle down-lt-up geometric-process
by auto
            finally show cls-val-process Mkt arb-pf (Suc 0) w \geq 0 .
            qed
            have }\forallw\in space M. cls-val-process Mkt arb-pf (Suc 0) w\geq0
            proof
            fix w
            assume w\in space M
            show cls-val-process Mkt arb-pf (Suc 0) w\geq0
            proof (cases snth w 0)
            case True
            thus ?thesis using <w\in space M> iniT by auto
            next
            case False
            thus ?thesis using <w\in space M> iniF by simp
            qed
qed
thus AE w in M. 0 \leqcls-val-process Mkt arb-pf (Suc 0) w by simp
show 0< prob {w\in space M. 0 < cls-val-process Mkt arb-pf (Suc 0) w}
proof -
have cls-val-process Mkt arb-pf (Suc 0) \in borel-measurable M using borel-adapt-stoch-proc-borel-measurabl
                    cls-val-process-adapted〈trading-strategy arb-pf\rangle\langlestock-portfolio Mkt arb-pf\rangle
                    using support-adapt-def readable unfolding stock-portfolio-def by blast
                            hence set-event:{w\in space M. 0<cls-val-process Mkt arb-pf (Suc 0)w}\in
sets M
            using borel-measurable-iff-greater by blast
            have }\foralln\mathrm{ . emeasure }M{w\in\mathrm{ 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-val-process Mkt arb-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-val-process Mkt arb-pf 1 w using iniT by simp
thus $w \in\{w \in$ space M. $0<$ cls-val-process Mkt arb-pf $1 w\}$ using wprops by $\operatorname{simp}$
qed
ultimately have $p \leq$ emeasure $M\{w \in$ space $M .0<$ cls-val-process $M k t$ arb-pf 1 w$\}$
using emeasure-mono set-event by fastforce
hence $p \leq \operatorname{prob}\{w \in$ space $M .0<$ cls-val-process Mkt arb-pf $1 w\}$ by (simp add: emeasure-eq-measure)
thus $0<\operatorname{prob}\{w \in$ space M. $0<$ cls-val-process Mkt arb-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 $\operatorname{simp}$
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 $=\left(\lambda\left(x::^{\prime} a\right)(n:: n a t) w .0:: r e a l\right)(s t k:=(\lambda n w .-1)$, risk-free-asset $:=(\lambda n w$. geom-proc $0 w))$
have support-set arb-pf $=\{$ stk, risk-free-asset $\}$
proof
show support-set arb-pf $\subseteq\{$ stk, risk-free-asset $\}$
by (simp add: arb-pf-def subset-iff support-set-def)
have stk $\in$ support-set arb-pf unfolding arb-pf-def support-set-def using two-stocks by $\operatorname{simp}$
moreover have risk-free-asset $\in$ support-set arb-pf unfolding arb-pf-def sup-port-set-def
using two-stocks geometric-process S0-positive by simp
ultimately show $\{$ stk, risk-free-asset $\} \subseteq$ 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 $\in$ support－set arb－pf
show borel－predict－stoch－proc G（arb－pf asset）
proof（cases asset $=$ stk）
case True
hence arb－pf asset $=(\lambda n w .-1)$ unfolding arb－pf－def by（simp add： two－stocks）
show ？thesis unfolding predict－stoch－proc－def
proof
 w．-1 ）＞by $\operatorname{simp}$
show $\forall n$ ．arb－pf asset $(S u c n) \in$ borel－measurable $(G n)$
proof
fix $n$
show arb－pf asset（Suc n） borel－measurable（ $G$ n）using 〈arb－pf asset $=(\lambda n w .-1)\rangle$ by $\operatorname{simp}$
qed
qed
next
case False
hence arb－pf asset $=(\lambda n w$ ．geom－proc $0 w)$ using «support－set arb－pf $=$ \｛stk，risk－free－asset $\}$ ，

〈asset $\in$ support－set arb－pf〉 unfolding arb－pf－def by simp
show ？thesis unfolding predict－stoch－proc－def
proof
 w．geom－proc $0 w)$＞
geometric－process by simp
show $\forall$ n．arb－pf asset $($ Suc $n) \in$ borel－measurable（ $G n$ ）
proof
fix $n$
show arb－pf asset（Suc n） borel－measurable（ $G n$ ）using 〈arb－pf asset $=(\lambda n w$. geom－proc $0 w)\rangle$ geometric－process by simp
qed
qed
qed
qed
show $\forall w \in$ space $M$ ．cls－val－process Mkt arb－pf $0 w=0$
proof
fix $w$

```
    assume w\in 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: }\forallw\in space M. cls-val-process Mkt arb-pf (Suc 0) w = - geom-pro
(Suc 0) w+(1+r)* geom-proc 0 w
    proof (intro ballI)
    fix w
    assume w\in 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:}\forallw\in space M. snth w0 < cls-val-process Mkt arb-pf (Suc 0)w
\geq0
    proof (intro ballI impI)
            fix w
            assume w\in 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\in 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 ... \geq0 using SO-positive }\langleu\leq1+r\rangle\mathrm{ down-lt-up geometric-process
by auto
    finally show cls-val-process Mkt arb-pf (Suc 0) w\geq0.
    qed
```

```
    have iniF: }\forall\textrm{w}\in\mathrm{ space M. ᄀsnth w 0 }\longrightarrow\mathrm{ cls-val-process Mkt arb-pf (Suc 0) w
>0
    proof (intro ballI impI)
        fix w
            assume w\in space M and \negsnth 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 }\langlew\in\mathrm{ space M> by simp
            also have ... = -d* geom-proc 0 w + (1+r)* geom-proc 0 w using }\neg\neg\mathrm{ 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 SO-positive }\langleu<=1+r\rangle\mathrm{ down-lt-up geometric-process
by auto
            finally show cls-val-process Mkt arb-pf (Suc 0) w>0.
    qed
    have }\forallw\in space M. cls-val-process Mkt arb-pf (Suc 0) w\geq0
    proof
        fix }
        assume w\in space M
        show cls-val-process Mkt arb-pf (Suc 0) w}\geq
        proof (cases snth w 0)
            case True
            thus ?thesis using \langlew\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 \leqcls-val-process Mkt arb-pf (Suc 0) w by simp
    show 0< prob {w\in space M. 0<cls-val-process Mkt arb-pf (Suc 0)w}
    proof -
    have cls-val-process Mkt arb-pf (Suc 0) \in borel-measurable M using borel-adapt-stoch-proc-borel-measurabl
            cls-val-process-adapted〈trading-strategy arb-pf\rangle\langlestock-portfolio Mkt arb-pf\rangle
            using support-adapt-def readable unfolding stock-portfolio-def by blast
    hence set-event:{w\in space M. 0 < cls-val-process Mkt arb-pf (Suc 0)w}\in
sets M
            using borel-measurable-iff-greater by blast
            have }\foralln\mathrm{ . emeasure }M{w\in\mathrm{ space M. }\negw!! 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. \negw !! 0} = ennreal (1-p) by blast
    moreover have {w\in space M. \negw!! 0} \subseteq{w\in space M. 0< cls-val-process
Mkt arb-pf 1 w}
    proof
        fix w
            assume w\in{w\in space M. \negw !! 0}
            hence }w\in\mathrm{ space M and }\negw!! 0 by auto note wprops = thi
            hence 0<cls-val-process Mkt arb-pf 1 w using iniF by simp
```

thus $w \in\{w \in$ space M. $0<$ cls-val-process Mkt arb-pf $1 w\}$ using wprops by $\operatorname{simp}$
qed
ultimately have $1-p \leq$ emeasure $M\{w \in$ space $M .0<$ cls-val-process $M k t$ arb-pf 1 w\}
using emeasure-mono set-event by fastforce
hence $1-p \leq \operatorname{prob}\{w \in$ space $M .0<$ cls-val-process $M k t$ arb-pf $1 w\}$ by (simp add: emeasure-eq-measure)
thus $0<\operatorname{prob}\{w \in \operatorname{space} M .0<c l s-v a l-p r o c e s s$ 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 $\operatorname{simp}$
qed
lemma (in CRR-market) viable-iff:
shows viable-market $M k t \longleftrightarrow(d<1+r \wedge 1+r<u)$ 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 $\forall n$. 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) $=\left(\begin{array}{ll}\lambda & n \\ w .1\end{array}\right)$
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 \cap space
N|B.B sets (bernoulli-stream q)}= sigma-sets (space (bernoulli-stream p))
                    {pseudo-proj-True n -' B\cap space M | B. B \in sets (bernoulli-stream p)}
    proof -
    have sets N = events
            by (metis assms(1) bernoulli-stream-def infinite-coin-toss-space-axioms infi-
nite-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 infi-
nite-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 $n \in$ borel-measurable (infinite-coin-toss-space.nat-filtration $N n$ )
using geometric-process
prob-grw.geom-rand-walk-borel-adapted $[$ of $q$ geom-proc u d init]
by (metis 〈infinite-coin-toss-space $q$ \ ${ }^{\text {l }}$ geom-rand-walk-pseudo-proj-True infi-
nite-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 $C R R$-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 $\langle r \in$ 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 $\langle r \in$ 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 $C R R$-market) bernoulli-AE-cond-exp:
assumes $N=$ bernoulli-stream $q$
and $0<q$
and $q<1$
and integrable $N X$
shows $A E w$ 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 NX using assms by simp
    show proj-stoch-proc geom-proc n \inN 和 stream-space borel using assms
proj-stoch-proc-geom-rng by simp
    show }\forallr\in\mathrm{ range (proj-stoch-proc geom-proc n) }\cap\mathrm{ space (stream-space borel).
        \existsA\insets (stream-space borel). range (proj-stoch-proc geom-proc n) \capA={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\in
        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 \in space N}->\mathrm{ 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 }\forallr\in\mathrm{ range (proj-stoch-proc geom-proc n) }\cap\mathrm{ space (stream-space borel).
        \existsA\insets (stream-space borel). range (proj-stoch-proc geom-proc n) \capA={r}
    proof
        fix r
        assume r\inrange (proj-stoch-proc geom-proc n) \cap space (stream-space borel)
        show \exists A\insets (stream-space borel). range (proj-stoch-proc geom-proc n) \capA
={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 $\langle r \in$ 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
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 $A E w$ 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 $\operatorname{simp}$
show real-cond-exp $N$ (fct-gen-subalgebra $N$ (stream-space borel) (proj-stoch-proc geom-proc n)) $X$
$\in$ 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)) $X$
$\in$ 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) $X \in$
borel-measurable (infinite-coin-toss-space.nat-filtration $N n$ )
proof -
have expl-cond-expect $N$ (proj-stoch-proc geom-proc n) $X \in$
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 $q \quad N$ 〉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 $C R R$-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 $n=f$ ct-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 NX
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)) Xw=
    expl-cond-expect N (proj-stoch-proc geom-proc n) X w using assms
    bernoulli-AE-cond-exp by simp
    have }\forallw\mathrm{ . real-cond-exp N (fct-gen-subalgebra N (stream-space borel) (proj-stoch-proc
geom-proc n))
    Xw = 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 (\lambdax. discount-factor r n x * X x) w=
discount-factor r n w*(real-cond-exp MGX)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 borel-measurable G by (simp add: discount-factor-borel-measurable)
    show random-variable borel X using assms by simp
    show integrable M ( }\lambdax\mathrm{ . discount-factor r nx* X x) using assms discounted-integrable[of
M \lambdan. 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 $A E w$ in $M$. real-cond-exp $M G(($ discounted-value $r(\lambda m . X)) n) w=$
discounted-value $r$ ( $\lambda m$. (real-cond-exp $M G X)) n w \mathbf{u s i n g}$ 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 $\ldots=(u-d) /(u-d)-(1+r-d) /(u-d)$ using down-lt-up
by $\operatorname{simp}$
also have $\ldots=(u-d-(1+r-d)) /(u-d)$ using diff-divide-distrib[of $u$
$-d 1+r-d u-d]$ by $\operatorname{simp}$
also have $\ldots=(u-1-r) /(u-d)$ by $\operatorname{simp}$
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 $\ldots=(u *(1+r-d)+d *(u-1-r)) /(u-d)$ using
add-divide-distrib[of $u *(1+r-d)]$ by $\operatorname{simp}$
also have $\ldots=(u *(1+r)-u * d+d * u-d *(1+r)) /(u-d)$
by (simp add: diff-diff-add right-diff-distrib)
also have $\ldots=(u *(1+r)-d *(1+r)) /(u-d)$ by simp
also have $\ldots=((u-d) *(1+r)) /(u-d)$ by (simp add: left-diff-distrib)
also have $\ldots=1+r$ 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\in 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 Gn=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\in
        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 (\lambdam 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 Gn = (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 $\ldots=($ 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 $G n=(f c t-g e n$-subalgebra $N$ (stream-space borel)
(proj-stoch-proc geom-proc n)).
hence $A E$ win $N$. real-cond-exp $N(G n)$
(discounted-value r (prices Mkt stk) (Suc $n$ )) $w=$ real-cond-exp $N(f c t-g e n-s u b a l g e b r a$ $N$ (stream-space borel)
(proj-stoch-proc geom-proc n))
(discounted-value $r$ (prices Mkt stk) (Suc n)) w by simp
moreover have $A E w$ 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 ( $\lambda$ m. (prices Mkt stk) (Suc n)) (Suc
n)) $w$
proof -
have $\forall w$. (discounted-value $r$ (prices Mkt stk) (Suc n)) $w=$
(discounted-value r ( $\lambda$ m. (prices Mkt stk) (Suc n)) (Suc n)) w
proof
fix $w$
show discounted-value $r$ (prices Mkt stk) (Suc n) $w=$ discounted-value $r(\lambda m$. 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 $r$ ( $\lambda m$. (prices Mkt stk) (Suc n)) (Suc n)) by auto
thus ?thesis by simp
qed
moreover have $A E w$ in $N$. (real-cond-exp $N$ (fct-gen-subalgebra $N$ (stream-space borel)
(proj-stoch-proc geom-proc n))
(discounted-value $r$ ( $\lambda m$. (prices Mkt stk) (Suc n)) (Suc
n))) $w=$
discounted-value $r$ ( $\lambda m$. 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 \inN 施 stream-space borel
        proof -
            have proj-stoch-proc geom-proc n \in 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 (\lambdam. 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 (\lambdam 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)))
                (\lambdaw.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 $n \in$ infinite-coin-toss-space.nat-filtration $N n$ $\rightarrow_{M}$ 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 $\langle p r o j-$-stoch-proc geom-proc $n w \neq$ proj-stoch-proc geom-proc $n$ (pseudo-proj-True $n$ w) >
geom-rand-walk-borel-adapted infinite-cts-filtration.proj-stoch-proj-invariant)
thus $\exists A \in$ sets (stream-space borel).
(proj-stoch-proc geom-proc $n w \in A)=($ proj-stoch-proc geom-proc $n$ (pseudo-proj-True $n w) \notin A)$ by $\operatorname{simp}$
qed
show $\forall w$. proj-stoch-proc geom-proc $n-‘\{$ proj-stoch-proc geom-proc $n w\} \in$ sets (infinite-coin-toss-space.nat-filtration $N n$ )
proof
fix $w$
show proj-stoch-proc geom-proc $n-'\{$ proj-stoch-proc geom-proc $n w\} \in$ sets (infinite-coin-toss-space.nat-filtration $N n$ )
using $<$ proj-stoch-proc geom-proc $n \in$ infinite-coin-toss-space.nat-filtration $N n \rightarrow{ }_{M}$ stream-space borel>
using assms geom-rand-walk-borel-adapted nat-filtration-from-eq-sets qlt $q g t$
infinite-cts-filtration.proj-stoch-singleton-set CRR-infinite-cts-filtration by blast
qed
show $\forall r \in$ 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 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 $f n w!!n=f n w\rangle$ $\langle$ proj-stoch-proc geom-proc $n y=$ proj-stoch-proc geom-proc $n z \wedge y!!n=z!!n\rangle$ 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 $n z \wedge y!!n=z!!n\rangle)$
thus prices Mkt stk (Suc n) y = prices Mkt stk (Suc n) zusing stk-price by simp
qed
show $0<q$ and $q<1$ using assms by auto

## qed

moreover have $\forall w . q *$ prices Mkt stk $(S u c n)($ 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 $\ldots=q * u *$ 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 $\ldots=q * u *$ 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))
```

                        (proj-stoch-proc geom-proc n))
                        ((prices Mkt stk) (Suc n)))
                        ((prices Mkt stk) (Suc n)))
                (\lambdaw. (q*u+(1-q)*d)* prices Mkt stk n w) by simp
                (\lambdaw. (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 \(A E w\) in \(N\). real-cond-exp \(N\left(\begin{array}{ll}G & 0\end{array}\right)\)
        (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 \(A E\) win \(N\). real-cond-exp \(N\left(\begin{array}{ll}G & 0)\end{array}\right.\)
        (discounted-value \(r\) (prices Mkt stk) (Suc 0)) \(w=\) prices Mkt stk \(0 w\) by
(simp add: discounted-init)
    moreover have \(A E w\) in \(N\). real-cond-exp \(N\) ( \(\begin{aligned} & G\end{aligned} 0\) ) (discounted-value \(r\) (prices
Mkt stk) (Suc 0)) w =
    discounted-value \(r(\lambda m w .(q * u+(1-q) * d) *\) prices Mkt stk \(0 w)(S u c 0)\)
\(w\)
    using assms real-cond-exp-discount-stock by simp
    ultimately have \(A E w\) in \(N\). discounted-value \(r(\lambda m w .(q * u+(1-q) * d)\)
* prices Mkt stk 0 w) (Suc 0) \(w=\)
    prices Mkt stk \(0 w\) by auto
    hence \(A E w\) in \(N\). discounted-value \(r(\lambda m w .(q * u+(1-q) * d) *\) init) (Suc
0) \(w=\)
    ( \(\lambda\) w. init) \(w\) using stk-price geometric-process by simp
    hence \(A E w\) in \(N\). discount-factor \(r(S u c 0) w *(q * u+(1-q) * d) *\) init \(=\)
    ( \(\lambda w\). init) \(w\) unfolding discounted-value-def by simp
hence \(A E w\) in \(N .(1+r) *\) discount-factor \(r(S u c ~ 0) w *(q * u+(1-q) *\) d) \(*\) init \(=\)
\((1+r) *(\lambda w\). init) \(w\) by auto
hence prev: AE win N. discount-factor r \(0 w *(q * u+(1-q) * d) *\) init \(=\) \((1+r) *(\lambda w\). init) \(w\) using discount-factor-times-rfr[of r 0] acceptable-rate
proof -
have \(\forall s .(1+r) *\) discount-factor \(r(\) Suc 0\()(s::\) bool stream \()=\) discount-factor \(r 0 s\)
by (metis (no-types) < \(\bigwedge w .-1<r \Longrightarrow(1+r) *\) discount-factor \(r(S u c 0) w\) \(=\) discount-factor \(r 0\) 〉 acceptable-rate)
then show ?thesis
using \(\langle A E e q N(\lambda w .(1+r) *\) discount-factor \(r(\) Suc 0\() w *(q * u+(1-\) \(q) * d) *\) init \()(\lambda w .(1+r) *\) init \()>\) by presburger
qed
hence \(\forall w\). ( \(\lambda w\). discount-factor r \(0 w *(q * u+(1-q) * d) *\) init \() w=\)
\((\lambda w .(1+r) *\) init \() w\)
proof -
have \((\lambda w\). discount-factor r \(0 w *(q * u+(1-q) * d) *\) init \()\)
\(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration N 0 )
proof (rule borel-measurable-times)+
show \((\lambda x\). init \() \in\) borel-measurable (infinite-coin-toss-space.nat-filtration \(N 0\) )
by \(\operatorname{simp}\)
show \((\lambda x . q * u+(1-q) * d) \in\) borel-measurable (infinite-coin-toss-space.nat-filtration
\(N 0)\) by \(\operatorname{simp}\)
show discount-factor r \(0 \in\) borel-measurable (infinite-coin-toss-space.nat-filtration N0)
using discount-factor-nonrandom[of r 0 infinite-coin-toss-space.nat-filtration
\(N 0]\) by \(\operatorname{simp}\)
qed
moreover have \((\lambda w .(1+r) *\) init \() \in\) 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: infi-nite-coin-toss-space-def)
ultimately show ?thesis
using prev infinite-coin-toss-space.nat-filtration-AE-eq[of q N
\((\lambda w\). discount-factor r \(0 w *(q * u+(1-q) * d) *\) init \()(\lambda w .(1+r) *\)
init) 0] assms
by (simp add: discount-factor-init)
qed
hence \((q * u+(1-q) * d) *\) init \(=(1+r) *\) init by (simp add: dis-count-factor-init)
hence \(q * u+(1-q) * d=1+r\) using \(S 0\)-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)

\section*{qed}
locale \(C R R\)-market-viable \(=C R R\)-market + assumes \(C R R\)-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 \(A E\) 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 \(A E w\) in \(N\). real-cond-exp \(N(G n)\) (discounted-value \(r\) (prices Mkt stk) (Suc \(n\) )) \(w=\)
discounted-value \(r(\lambda m w .(q * u+(1-q) * d) *\) prices Mkt stk \(n\)
w) \((\) Suc \(n) w\)
using assms real-cond-exp-discount-stock[of \(N\) q] qlt qgt by simp
moreover have \(\forall w .(q * u+(1-q) * d) *\) 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 \(A E w\) in \(N\). real-cond-exp \(N(G n)\) (discounted-value \(r\) (prices Mkt stk) (Suc n)) w =
discounted-value \(r(\lambda m w .(1+r) *\) prices Mkt stk \(n w)(S u c n) w\)
by \(\operatorname{simp}\)
moreover have \(\forall w \in\) space \(N\). discounted-value \(r(\lambda m w .(1+r) *\) prices Mkt stk \(n w)(\) Suc \(n) w=\) discounted-value \(r\) ( \(\lambda m\) w. prices Mkt stk \(n w) n w\)
using acceptable-rate by (simp add:discounted-mult-times-rfr)
moreover hence \(\forall w \in\) space \(N\). discounted-value \(r(\lambda m w .(1+r) *\) prices Mkt stk \(n\) ) (Suc n) \(w=\)
discounted-value \(r\) (prices Mkt stk) \(n w\)
using acceptable-rate by (simp add:discounted-value-def)
ultimately show \(A E w\) in \(N\). real-cond-exp \(N(G n)\) (discounted-value \(r\) (prices Mkt stk) (Suc n)) w =
\[
\text { discounted-value } r \text { (prices Mkt stk) } n w \text { by simp }
\]
qed
lemma (in CRR-market-viable) risky-asset-martingale-if:
assumes \(N=\) bernoulli-stream \(q\)
and \(q=(1+r-d) /(u-d)\)
shows martingale \(N G\) (discounted-value \(r\) (prices Mkt stk))
proof (rule disc-martingale-charact)
have qlt: \(0<q\) and \(q g t: q<1\) using assms gt-param lt-param CRR-viable by
```

auto
show }\foralln\mathrm{ . integrable N (discounted-value r (prices Mkt stk) n)
proof
fix n
show integrable N (discounted-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 borel-measurable (infinite-coin-toss-space.nat-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 }\foralln\mathrm{ . sigma-finite-subalgebra N (G n) using qlt qgt by (simp add: assms
bernoulli-sigma-finite)
show }\forallm\mathrm{ . discounted-value r (prices Mkt stk) m f borel-measurable (G m)
proof
fix m
have discounted-value r (\lambdama. prices Mkt stk m) m b borel-measurable (G m)
proof (rule discounted-measurable)
show prices Mkt stk m b borel-measurable (G m) using stock-price-borel-measurable
unfolding adapt-stoch-proc-def by simp
qed
thus discounted-value r (prices Mkt stk) m borel-measurable (G m)
by (metis (mono-tags, lifting) discounted-value-def measurable-cong)
qed
show }\foralln.AEw 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
fix n
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
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 \(\operatorname{simp}\)
\}
\{
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

\subsection*{8.3 Existence of a replicating portfolio}
fun (in CRR-market) rn-rev-price where
rn-rev-price \(N\) der matur \(0 w=\operatorname{der} w \mid\)
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 $\ldots=$ 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 $\operatorname{simp}$
qed
finally show? ?thesis.
qed

```
lemma (in CRR-market) real-exp-eq:
    assumes der \(\in\) 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 \(\in\) 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 \(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration \(N\) matur)
        by (metis \(\langle\) der \(\in\) borel-measurable (nat-filtration matur)〉〈infinite-coin-toss-space
\(q\) 〉
                    \(\operatorname{assms}(2) \operatorname{assms}(3) \operatorname{assms}(4)\) 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 < qq<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 \in borel-measurable (G matur)
and N= bernoulli-stream q
and 0<q
and q<1
shows ( }n\leq\mathrm{ matur ) ఋ(rn-rev-price N cash-flow matur n) f borel-measurable ( }
(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)=
(\lambdaw. 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 ... \in borel-measurable ( }G\mathrm{ (matur - Suc n))
proof (rule borel-measurable-times)
show discount-factor r (Suc 0) \in 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)
\inborel-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) \in 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 \in borel-measurable (nat-filtration n)
and -1<r
shows integrable N (discounted-value r ( }\lambdam\mathrm{ . 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 \in 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 \(\in\) borel-measurable ( \(G\) matur)
and \(N=\) bernoulli-stream \(q\)
and \(0<q\)
and \(q<1\)
shows \(n \leq\) matur \(\Longrightarrow\) rn-rev-price \(N\) der matur \(n w=\)
    expl-cond-expect \(N\) (proj-stoch-proc geom-proc (matur \(-n\) ) (discounted-value \(r\)
\((\lambda m\). der) \(n\) ) \(w\)
proof (induct \(n\) arbitrary: \(w\) )
    case 0
    have der \(\in\) 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 \(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration \(N\) matur)
        by (metis \(\langle\) der \(\in\) borel-measurable (nat-filtration matur)〉〈infinite-coin-toss-space
\(q\) >
            \(\operatorname{assms(2)} \operatorname{assms(3)} \operatorname{assms}(4)\) 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 \(0 w=\operatorname{der} w\) by simp
    also have \(\ldots=\) expl-cond-expect \(N\) (proj-stoch-proc geom-proc matur) (discounted-value
\(r(\lambda m\). der \() 0) w\)
    proof (rule nat-filtration-AE-eq)
    show der \(\in\) borel-measurable (nat-filtration matur) using \(\langle\) der \(\in\) borel-measurable
(nat-filtration matur)>.
    have (discounted-value \(r(\lambda m\). 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 \(\in\) 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 \(\forall w\). 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 \(r(\lambda m\). 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 \(r(\lambda m . d e r) 0)\)
\(\in\) 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 \(\in\) space \(N \rightarrow\) 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 \(\operatorname{simp}\)
show \(\forall r \in\) range (proj-stoch-proc geom-proc matur) \(\cap\) space (stream-space borel).
\(\exists A \in\) sets (stream-space borel). range (proj-stoch-proc geom-proc matur) \(\cap A\) \(=\{r\}\)
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 ( \(\lambda m\). der) 0)
\(\in\) 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 ( \(\lambda m\). 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 \(\in\) borel-measurable (nat-filtration matur) using \(\langle\) der \(\in\) borel-measurable (nat-filtration matur)>.
show \(A E e q N\) der (expl-cond-expect \(N\) (proj-stoch-proc geom-proc matur)
(discounted-value \(r(\lambda m\). der) 0))
using eqn .
show expl-cond-expect \(N\) (proj-stoch-proc geom-proc matur) (discounted-value \(r(\lambda m . d e r) 0)\)
\(\in\) 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 \(\ldots=\) expl-cond-expect \(N(\) proj-stoch-proc geom-proc \((\) matur -0\())\)
(discounted-value \(r(\lambda m\). der) 0) \(w\)
by \(\operatorname{simp}\)
finally show rn-rev-price \(N\) der matur \(0 w=\)
expl-cond-expect \(N\) (proj-stoch-proc geom-proc (matur - 0)) (discounted-value
\(r(\lambda m . d e r) 0) w\).
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 \(\ldots=\) discount-factor \(r(S u c ~ 0) w *\)
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 \(-S u c n)\) ) (rn-rev-price
\(N\) der matur n) \(w\)
proof (rule real-exp-eq[symmetric])
show rn-rev-price \(N\) der matur \(n \in\) 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 \(\ldots=\) 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
\(r(\lambda m . d e r) n)) w\)
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 \(-S u c n))\)
(expl-cond-expect \(N\) (proj-stoch-proc geom-proc (matur \(-n\) )) (discounted-value \(r(\lambda m . \operatorname{der}) n)) w\)
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 \(r(\lambda m . d e r) n)))\)
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 \(n \in\) borel-measurable \(N\)
proof -
have rn-rev-price \(N\) der matur \(n \in\) borel-measurable \((G(\) matur \(-n))\)
by (metis (full-types) Suc.prems Suc-leD assms(1) assms(2) assms(3) \(\operatorname{assms}(4)\) rn-rev-price-rev-borel-adapt)
then show ?thesis
by (metis (no-types) assms(2) bernoulli bernoulli-stream-def filtra-tion-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 \(r(\lambda m . d e r) n) \in\) borel-measurable \(N\)
using Suc.hyps Suc.prems Suc-leD \(\langle r n\)-rev-price \(N\) der matur \(n \in\) 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 \(r(\lambda m\). 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 \(n\) )
\(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration \(N\) (matur - Suc \(n)\) )
by (metis assms(2) assms(3) assms(4) borel-measurable-cond-exp infi-nite-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 \(r(\lambda m . d e r) n))\)
\(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration \(N\) (matur - Suc n))
by (metis assms(2) assms(3) assms(4) borel-measurable-cond-exp infi-nite-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

\section*{qed}
also have \(\ldots=\) 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
\(r(\lambda m . d e r) n)) w\)
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
\(r(\lambda m . \operatorname{der}) n)) w=\)
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 \(r(\lambda m . \operatorname{der}) n)) w\)
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 \(r(\lambda m . d e r) n)))\)
(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 \(r(\lambda m\). 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 \(r(\lambda m\). der \() n) \in\) borel-measurable \(N\)
by \(\operatorname{simp}\)
show expl-cond-expect \(N\) (proj-stoch-proc geom-proc (matur \(-n)\) ) (discounted-value \(r(\lambda m . d e r) n) \in\) 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 ( \(\lambda m\). der) \(n\) ))
(real-cond-exp \(N\) (stoch-proc-filt \(N\) geom-proc borel (matur - \(n\) )) (discounted-value \(r(\lambda m . d e r) n))\)
proof -
have discounted-value \(r\) ( \(\lambda\) m. der) \(n \in\) borel-measurable ( \(G\) matur) using assms discounted-measurable[of der]
by \(\operatorname{simp}\)
hence \(\forall w\). (expl-cond-expect \(N\) (proj-stoch-proc geom-proc (matur \(-n)\) )
(discounted-value \(r(\lambda m\). der \() n)) w=\)
(real-cond-exp \(N\) (stoch-proc-filt \(N\) geom-proc borel (matur \(-n)\) )
(discounted-value \(r(\lambda m\). 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 \(r(\lambda m . \operatorname{der}) n))\)
\(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration N (matur - Suc
n))
by (metis assms(2) assms(3) assms(4) borel-measurable-cond-exp infi-nite-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 \(r(\lambda m . d e r) n))\)
\(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration N (matur - Suc n))
by (metis assms(2) assms(3) assms(4) borel-measurable-cond-exp infi-nite-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 \(\ldots=\) real-cond-exp \(N\) (stoch-proc-filt \(N\) geom-proc borel (matur - Suc \(n)\) )
(discounted-value \(r\) ( \(\lambda\) 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 ( \(\lambda\) m. der) (Suc n))
\(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration \(N(\) matur - Suc n))
by (metis assms(2) assms(3) assms(4) borel-measurable-cond-exp infi-nite-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 ( \(\lambda\) 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 \(r(\lambda m . d e r) n)\) ) a)
\(\in\) 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 ( \(\lambda m\). der) \(n\) ))
\(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration \(N\) (matur - Suc n))
by (metis assms(2) assms(3) assms(4) borel-measurable-cond-exp infi-nite-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
\(r(\lambda m . \operatorname{der}) n))]\)
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 \(N(\lambda w\). discount-factor \(r(S u c ~ 0) w *\)
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 \(r(\lambda m . d e r) n)) w\) )
(real-cond-exp \(N\) (stoch-proc-filt \(N\) geom-proc borel (matur - Suc n)) (discounted-value
\(r(\lambda m\). der \()(\) Suc \(n)))\)
proof -
have AEeq \(N\)
( \(\lambda w\). 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 \(r(\lambda m\). der \() n)) w\) )
( \(\lambda w\). discount-factor \(r\) (Suc 0) \(w *\) real-cond-exp \(N\) (stoch-proc-filt \(N\) geom-proc borel (matur - Suc n))
(discounted-value \(r(\lambda m . d e r) n\) ) \(w\) )
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 \(r(\lambda m\). der \() n))\) )
(real-cond-exp \(N\) (stoch-proc-filt \(N\) geom-proc borel (matur - Suc n))
(discounted-value r ( \(\lambda m\). 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))

\section*{proof -}
have init-triv-filt \(M\) (stoch-proc-filt \(M\) geom-proc borel) using infi-nite-cts-filtration.stoch-proc-filt-triv-init
using info-filtration stock-filtration by auto
moreover have matur \(-(\) Suc \(n) \leq\) matur \(-n\) by simp
ultimately show ?thesis unfolding init-triv-filt-def filtration-def
using \(\operatorname{assms(2)} \operatorname{assms(3)} \operatorname{assms}(4)\) stock-filtration stock-filtration-eq
by auto
qed
show integrable \(N\) (discounted-value \(r(\lambda m\). 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 \(A E e q N\)
( \(\lambda w\). discount-factor \(r\) (Suc 0) \(w\) *
real-cond-exp \(N\) (stoch-proc-filt \(N\) geom-proc borel (matur - Suc n))
(discounted-value \(r(\lambda m\). der \() n\) ) w)
( \(\lambda w\). discount-factor \(r\) (Suc 0) \(w *\) (discounted-value \(r\)
( \(\lambda\) m. 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 \(r(\lambda m\). der) \(n)\) )
(discounted-value \(r\)
( \(\lambda\) m. 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 \(\in\) 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 \(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration \(N\) matur)
by (metis \(\langle d e r \in\) borel-measurable (nat-filtration matur)〉〈infi-nite-coin-toss-space \(q\) )
\(\operatorname{assms(2)} \operatorname{assms}(3) \operatorname{assms}(4)\) 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 \(\operatorname{assms}(2) \operatorname{assms}(3)\) assms(4) bernoulli-sigma-finite sigma-finite-subalgebra.subalg stock-filtration-eq by fastforce
qed
thus ?thesis by auto
qed
moreover have \(\forall w\). ( \(\lambda w\). discount-factor \(r\) (Suc 0) \(w *\) (discounted-value \(r\)
( \(\lambda\) m. real-cond-exp \(N\) (stoch-proc-filt \(N\) geom-proc borel (matur \(-S u c n)\) )
der) \(n\) ) \(w) w=\)
(discounted-value \(r\)
( \(\lambda\) m. 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 \(A E e q N\)
(real-cond-exp \(N\) (stoch-proc-filt \(N\) geom-proc borel (matur - Suc n))
(discounted-value \(r(\lambda m\). der) (Suc n)))
(discounted-value \(r\)
( \(\lambda\) m. 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 \(\in\) 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 \(\in\) borel-measurable (infinite-coin-toss-space.nat-filtration \(N\) matur)
by (metis \(\langle d e r \in\) borel-measurable (nat-filtration matur)〉〈infinite-coin-toss-space
\(q\) >
\(\operatorname{assms(2)} \operatorname{assms}(3) \operatorname{assms}(4)\) 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 \(\ldots=\) expl-cond-expect \(N(\) proj-stoch-proc geom-proc \((\) matur - Suc \(n))\)
(discounted-value \(r\) ( \(\lambda\) m. der) (Suc n)) \(w\)
proof (rule real-exp-eq)
show discounted-value \(r(\lambda\) m. der \()(S u c ~ n) \in\) borel-measurable ( \(G\) matur) using assms discounted-measurable[of der]
by \(\operatorname{simp}\)
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(\lambdam. 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 (\lambdam. 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 \in borel-measurable (G matur)
and n\leqmatur
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 }=>\mp@subsup{}{}{\prime}b::{t0-space
assumes f}\in\mathrm{ borel-measurable ( }Gn\mathrm{ )
shows fw=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\in borel-measurable (nat-filtration n) using assms by (simp add: measur-
able-from-subalg)
thus ?thesis using nat-filtration-info[of f n] by (metis comp-apply)
qed
lemma (in CRR-market) geom-proc-fil-info':
fixes f::bool stream = 'b::{t0-space}
assumes f\in borel-measurable (G n)
shows fw=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\in borel-measurable (nat-filtration n) using assms by (simp add: measur-
able-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 \(\in\) borel-measurable ( \(G\) matur)
and \(N=\) bernoulli-stream \(q\)
and \(0<q\)
and \(q<1\)
and \(n \leq\) matur
shows (rn-price \(N\) cash-flow matur \(n\) ) \(\in\) borel-measurable ( \(G n\) )
proof -
show (rn-price \(N\) cash-flow matur \(n\) ) \(\in\) 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 \(T=\)
( \(\lambda\) n w. if \((S u c n \leq T)\)
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 \(n \leq T\)
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 wnx) = geom-rand-walk udinit (Suc n) (spick \(w n x)\) using geometric-process by simp
also have \(\ldots=(\) case (spick wnx)!! n of True \(\Rightarrow u \mid\) False \(\Rightarrow d) *\) geom-rand-walk
\(u d\) init \(n\) (spick wnx)
by simp
also have \(\ldots=(\) case \(x\) of True \(\Rightarrow u \mid\) False \(\Rightarrow d) *\) geom-rand-walk ud init \(n\) (spick \(w n x\) )
unfolding spick-def by simp
also have \(\ldots=(\) if \(x\) then \(u\) else \(d) *\) geom-rand-walk \(u d\) init \(n(\) spick \(w n x)\) by simp
also have \(\ldots=(\) 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 ( \(\lambda w\). spick \(w n x\) ) \(\in\) 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 ( \(\lambda w\). spick \(w n x\) )
\(\in\) space (fct-gen-subalgebra \(M\) borel (geom-proc \(n)\) ) \(\rightarrow\) space (fct-gen-subalgebra M borel (geom-proc (Suc n)))
by (simp add: bernoulli bernoulli-stream-space fct-gen-subalgebra-space)
show \(\forall y \in\) sets (fct-gen-subalgebra \(M\) borel (geom-proc (Suc n))).
( \(\lambda w\). spick \(w n x))^{\prime} y \cap\) space (fct-gen-subalgebra \(M\) borel (geom-proc \(n\) ))
\(\in\) sets (fct-gen-subalgebra M borel (geom-proc n))
proof
fix \(A\)
assume \(A: A \in\) sets (fct-gen-subalgebra \(M\) borel (geom-proc (Suc n)))
show ( \(\lambda w\). spick \(w n x\) ) -' \(A \cap\) space (fct-gen-subalgebra \(M\) borel (geom-proc
n)) \(\in\)
sets (fct-gen-subalgebra M borel (geom-proc n))
proof -
define \(s p\) where \(s p=(\lambda w\). spick \(w n x)\)
have \(A \in\{(\) geom-proc \((\) Suc \(n))-' B \cap\) space \(M \mid B\). \(B \in\) sets borel \(\}\) using \(A\)
by (simp add:fct-gen-subalgebra-sigma-sets)
from this obtain \(C\) where \(C \in\) sets borel and \(A=(\) geom-proc (Suc n) \()-{ }^{'} C\)
\(\cap\) space \(M\) by auto
hence \(A=(\) geom-proc \((S u c ~ n))-{ }^{\prime} C\) using bernoulli bernoulli-stream-space
by \(\operatorname{simp}\)
hence \(s p-' A=s p-'(\) geom-proc \((S u c n))-{ }^{\prime} C\) by simp
also have \(\ldots=(\) geom-proc \((\) Suc \(n) \circ s p)-{ }^{\prime} C\) by auto
also have \(\ldots=(\lambda w\). (if \(x\) then \(u\) else \(d) *\) geom-proc \(n w)-{ }^{\text {' }} C\) using geom-proc-spick
sp-def by auto
also have \(\ldots \in\) sets (fct-gen-subalgebra \(M\) borel (geom-proc \(n\) ))
proof (cases \(x\) )
case True
hence \((\lambda w\). (if \(x\) then \(u\) else \(d) *\) geom-proc \(n w)-{ }^{'} C=(\lambda w . u *\) geom-proc
```

nw) -' C by simp
moreover have (\lambdaw.u* geom-proc n w) \in borel-measurable (fct-gen-subalgebra
M borel (geom-proc n))
proof -
have geom-proc n \inborel-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 measur-
able-def mem-Collect-eq)
thus ?thesis by simp
qed
ultimately show ?thesis using <C\in sets borel>
by (metis bernoulli bernoulli-stream-preimage fct-gen-subalgebra-space
measurable-sets)
next
case False
hence (\lambdaw. (if x then u else d)* geom-proc n w) - 'C = (\lambdaw.d* geom-proc
nw) -' C by simp
moreover have (\lambdaw.d* geom-proc n w) \in borel-measurable (fct-gen-subalgebra
M borel (geom-proc n))
proof -
have geom-proc n \inborel-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 measur-
able-def mem-Collect-eq)
thus ?thesis by simp
qed
ultimately show ?thesis using <C\in 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:

```

```

    shows (\lambdaw. spick wnx) -'A\in{geom-proc n-'B|B. B\in sets borel}
    proof -
have sets (fct-gen-subalgebra M borel (geom-proc n)) ={geom-proc n - ' B\capspace
M |B.B 新s borel}
by (simp add: fct-gen-subalgebra-sigma-sets)
also have ... = {geom-proc n -' B|B. B 的ts 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 \in sets borel} .

```
define \(s p\) where \(s p=(\lambda w\) ．spick \(w n x)\)
from assms（1）obtain \(C\) where \(C \in\) 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 \(s p-' A=s p-{ }^{\prime}(\) geom－proc \((S u c n))-{ }^{\prime} C\) by \(\operatorname{simp}\)
also have \(\ldots=(\) geom－proc \((S u c n) \circ s p)-{ }^{\prime} C\) by auto
also have \(\ldots=(\lambda w\) ．（if \(x\) then \(u\) else \(d) *\) geom－proc \(n w)-{ }^{\prime} C\) using geom－proc－spick sp－def by auto
also have \(\ldots \in\{\) geom－proc \(n-‘ B \mid B . B \in\) sets borel \(\}\)
proof（cases \(x\) ）
case True
hence \((\lambda w\) ．（if \(x\) then \(u\) else \(d) *\) geom－proc \(n w)-{ }^{`} C=(\lambda w . u *\) geom－proc \(n w)-{ }^{\prime} C\) by \(\operatorname{simp}\)
moreover have \((\lambda w . u *\) geom－proc \(n w) \in\) borel－measurable（fct－gen－subalgebra \(M\) borel（geom－proc n））

\section*{proof－}
have geom－proc \(n \in\) 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）

\section*{next}
case False
hence \((\lambda w\) ．（if \(x\) then \(u\) else \(d) *\) geom－proc \(n w)-{ }^{`} C=(\lambda w . d *\) geom－proc \(n w)-{ }^{\prime} C\) by simp
moreover have \((\lambda w . d *\) geom－proc \(n w) \in\) borel－measurable（fct－gen－subalgebra M borel（geom－proc n））

\section*{proof－}
have geom－proc \(n \in\) 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 \(s p-d e f\) ． qed
```

lemma (in CRR-market) geom-spick-lt:
assumes m<n
shows geom-proc m (spick wnx)= geom-proc m w
proof -
have geom-proc m (spick w n x) = geom-proc m (pseudo-proj-True m (spick wn
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 mx) = geom-proc m w
proof (cases x)
case True
have geom-proc m(spick wmx)= geom-proc m (pseudo-proj-True m (spick w
mx))
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 mx) = geom-proc m (pseudo-proj-False m (spick w
mx))
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-fil:
shows (\lambdaw. spick wn x) \in measurable (Gn) (G (Suc n)) unfolding measur-
able-def
proof (intro CollectI conjI)
show (\lambdaw. spick wn x) \in space (Gn) -> space (G (Suc n)) using stock-filtration
by (simp add: bernoulli bernoulli-stream-space stoch-proc-filt-space)
show }\forally\in\operatorname{sets}(G(Sucn)).(\lambdaw. spick wnx) -` y\cap space (Gn)\in sets (Gn
proof
fix }
assume B\in sets (G (Suc n))

```
hence \(B \in(\) sigma-sets \((\) space \(M)(\bigcup i \in\{m . m \leq(S u c n)\} .\{(\) geom-proc \(i-‘ A)\)
\(\cap(\) space \(M) \mid A\). A sets borel \(\}))\)
using stock-filtration stoch-proc-filt-sets geometric-process
proof -
have \(\forall n\). sigma-sets (space \(M)\left(\bigcup n \in\{n a . n a \leq n\}\right.\). \{geom-proc \(n-{ }^{'} R \cap\)
space \(M \mid R . R \in\) sets borel \(\})=\) sets \((G n)\)
by (simp add: geom-rand-walk-borel-measurable stoch-proc-filt-sets stock-filtration)
then show?thesis
using \(\langle B \in \operatorname{sets}(G(S u c n))\rangle\) by blast
qed
hence ( \(\lambda w\). spick wnx)-' \(B \in \operatorname{sets}(G n)\)
proof (induct rule:sigma-sets.induct)
\{
fix \(C\)
assume \(C \in\left(\bigcup i \in\{m . m \leq S u c n\}\right.\). \{geom-proc \(i-{ }^{`} A \cap\) space \(M \mid A\). \(A\) \(\in\) sets borel\})
hence \(\exists m \leq\) Suc \(n . C \in\left\{\right.\) geom-proc \(m-{ }^{\prime} A \cap\) space \(M \mid A . A \in\) sets borel \(\}\) by auto
from this obtain \(m\) where \(m \leq S u c n\) and \(C \in\{\) geom-proc \(m-‘ A \cap\) space \(M \mid A . A \in\) sets borel \(\}\) by auto
note Cprops \(=\) this
from this obtain \(D\) where \(C=\) geom-proc \(m-{ }^{\prime} D \cap\) space \(M\) and \(D \in\) sets borel by auto
hence \(C=\) geom-proc \(m-' D\) using bernoulli bernoulli-stream-space by \(\operatorname{simp}\)
have \(C \in\left\{\right.\) geom-proc \(m-{ }^{\prime} A \mid A . A \in\) sets borel \(\}\) using bernoulli bernoulli-stream-space Cprops by simp
show ( \(\lambda w\). spick wnx)-' \(C \in\) sets \((G n)\)
proof (cases \(m \leq n\) )
case True
have \((\lambda w\). spick \(w n x)-{ }^{\prime} C=(\lambda w\). spick \(w n x)-{ }^{\prime}\) geom-proc \(m-{ }^{\prime} D\)
using \(\left\langle C=\right.\) geom-proc \(\left.m-{ }^{\prime} D\right\rangle\) by simp
also have \(\ldots=(\) geom-proc \(m \circ(\lambda w\). spick \(w n x))-{ }^{\prime} D\) by auto
also have \(\ldots=\) geom-proc \(m-' D\) using geom-spick-lt geom-spick-eq \(\langle m \leq n\rangle\) using le-eq-less-or-eq by auto
also have \(\ldots \in\) sets ( \(G n\) ) using stock-filtration geometric-process〈 \(D \in\) sets borel \(\rangle\)
by (metis (no-types, lifting) True adapt-stoch-proc-def bernoulli bernoulli-stream-preimage geom-rand-walk-borel-measurable increasing-measurable-info measur-
able-sets stoch-proc-filt-adapt stoch-proc-filt-space)
finally show ( \(\lambda w\). spick \(w n x)-{ }^{\prime} C \in \operatorname{sets}(G n)\).
next
case False
hence \(m=\) Suc \(n\) using \(\langle m \leq\) Suc \(n\) 〉 by simp
hence \(\left(\lambda w\right.\). spick wnx) \(-{ }^{\prime} C \in\left\{\right.\) geom-proc \(n-{ }^{\prime} B \mid B . B \in\) sets borel \(\}\) using \(\langle C \in\{\) geom-proc \(m-‘ A \mid A\). A sets borel \(\}\rangle\) geom-spick-Suc by \(\operatorname{simp}\)
also have \(\ldots \subseteq\) sets \((G n)\)

\section*{proof -}
have \(\left\{\right.\) geom-proc \(n-{ }^{\prime} B \mid B . B \in\) sets borel \(\} \subseteq\left\{\right.\) geom-proc \(n-{ }^{\prime} B \cap\) space \(M \mid B . B \in\) sets borel \(\}\)
using bernoulli bernoulli-stream-space by simp
also have \(\ldots \subseteq\left(\bigcup i \in\{m . m \leq n\}\right.\). \{geom-proc \(i-{ }^{\prime} A \cap\) space \(M \mid A . A\) \(\in\) sets borel\})
by auto
also have \(\ldots \subseteq\) 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 \(\ldots=\) 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 \(w n x)-‘\{ \} \in \operatorname{sets}(G n)\) 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 \(w n x)-{ }^{\prime} C \in \operatorname{sets}(G n)\)
hence \((\lambda w\). spick \(w n x)-‘(\) space \(M-C)=(\lambda w\). spick \(w n x)-‘(\) space \(M)-(\lambda w\). spick \(w n x)-{ }^{\prime} C\)
by (simp add: vimage-Diff)
also have \(\ldots=\) space \(M-(\lambda w\). spick \(w n x)-{ }^{\text {' }} C\) using bernoulli bernoulli-stream-space by simp
also have \(\ldots \in\) sets \((G n)\) using \(\left\langle(\lambda w\right.\). spick \(w n x)-{ }^{‘} C \in\) sets \(\left.(G n)\right\rangle\)
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 \(M-C) \in \operatorname{sets}(G n)\).
\}
\{
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 \(i-{ }^{\prime} A \cap\) space \(M \mid A . A \in\) sets borel \(\left.\left.\}\right)\right)\)
and \(\left(\bigwedge i\right.\). \((\lambda w\). spick \(\left.w n x)-{ }^{'} C i \in \operatorname{sets}(G n)\right)\)
hence \((\lambda w\). spick \(w n x)-` \bigcup\left(C^{\prime}\right.\) UNIV \()=(\bigcup i \in U N I V\). \((\lambda w\). spick \(w n\) \(x)-‘(C i))\) by blast
also have \(\ldots \in\) sets \((G n)\) using 〈 \(\backslash i\). ( \(\lambda w\). spick \(w n x)-{ }^{‘} C i \in\) sets \((G\) \(n)\) ) by \(\operatorname{simp}\)
finally show \((\lambda w\). spick \(w n x)-‘ \bigcup\left(C^{\prime} U N I V\right) \in \operatorname{sets}(G n)\).
\}
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

\section*{qed}
qed
lemma (in \(C R R\)-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 \(T(S u c n)(\) spick \(w n\) True \()) \in\) borel-measurable ( \(G n\) )
proof -
have rn-price \(N\) cash-flow \(T(S u c n) \in\) borel-measurable ( \(G(S u c n)\) ) 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 ( \(\lambda w\). (rn-price \(N\) cash-flow \(T\) (Suc n) (spick w n True) -
```

rn-price N cash-flow T (Suc n) (spick w n False))/
((geom-proc n w)* (u-d)))\in borel-measurable (Gn) 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 = ( \lambdaw. delta-price N der matur 0 w)
delta-predict N der matur (Suc n) = ( \lambdaw. delta-price N der matur n w)
lemma (in CRR-market) delta-predict-predict:
assumes der \in 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 pre-
dict-stoch-proc-def
proof (intro conjI)
show delta-predict N der matur 0 \in borel-measurable (G 0) using delta-price-adapted[of
der matur N q]
assms unfolding adapt-stoch-proc-def by force
show }\foralln\mathrm{ . delta-predict }N\mathrm{ der matur (Suc n) G borel-measurable (Gn)
proof
fix n
show delta-predict N der matur (Suc n) \in 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) $\subseteq\{$ 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 vpf 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 vpf asset)) unfolding trading-strategy-def
proof (intro conjI ballI)
show portfolio (qty-single asset (remaining-qty Mkt vpf asset))
by (simp add: self-finance-def single-comp-portfolio)
show \(\bigwedge a\).
\(a \in\) support-set (qty-single asset (remaining-qty Mkt v pf asset)) \(\Longrightarrow\)
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 vpfasset)) = \{asset \}
using single-comp-support by fastforce
fix \(a\)
assume \(a \in\) 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 \(q t y\)-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 \(\operatorname{simp}\)
next
case True
thus \(\bigwedge a . a \in\) support-set (qty-single asset (remaining-qty Mkt vpf asset)) \(\Longrightarrow\) support-set (qty-single asset (remaining-qty Mkt vpf asset)) \(=\{ \} \Longrightarrow\) borel-predict-stoch-proc \(F\) (qty-single asset (remaining-qty Mkt v pf asset)
a) by \(\operatorname{simp}\)
qed
qed
qed
lemma (in CRR-market) self-fin-delta-pf-trad-strat:
assumes der \(\in\) 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 \asset. asset \in support-set (delta-pf N der matur) \Longrightarrow 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 \in 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 \bigwedgeasset. asset \in support-set (delta-pf N der matur) \Longrightarrow
support-set (delta-pf N der matur) ={}\Longrightarrow 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 portfo-
lio-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 \asset. asset \in support-set (delta-pf N der matur) \Longrightarrow borel-adapt-stoch-proc
G (prices Mkt asset)
proof (cases support-set (delta-pf N der matur) = {})
case False
fix asset
assume asset \in 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 \asset. asset \in support-set (delta-pf N der matur) \Longrightarrow borel-adapt-stoch-proc
G (prices Mkt asset)
by simp
qed
qed
qed
definition (in CRR-market) delta-hedging where

```
```

lemma (in CRR-market) geom-proc-eq-snth:
shows $(\bigwedge m$. $m \leq$ Suc $n \Longrightarrow$ geom-proc $m x=$ geom-proc $m y) \Longrightarrow$
( $\wedge m . m \leq n \Longrightarrow$ snth $x m=$ snth $y m$ )
proof (induct $n$ )
case 0
assume asm: $(\bigwedge m . m \leq$ Suc $0 \Longrightarrow$ geom-proc $m x=$ geom-proc $m y)$ and $m \leq$
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 $x 0 \neq$ snth y 0
show False
proof (cases snth $x 0$ )
case True
hence $\neg$ snth y 0 using $\langle$ snth $x 0 \neq$ 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 $\prec \neg$
snth y 0 〉 by simp
ultimately have geom-proc (Suc 0) $x \neq$ geom-proc (Suc 0) y using SO-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 $\langle$ snth $x 0 \neq$ 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) $x \neq$ geom-proc (Suc 0) y using S0-positive
down-lt-up by simp
thus ? thesis using $\backslash$ geom-proc (Suc 0) $x=$ geom-proc (Suc 0) $y>$ by simp
qed
qed
thus $\wedge m$. $(\bigwedge m$. $m \leq$ Suc $0 \Longrightarrow$ geom-proc $m x=$ geom-proc $m y) \Longrightarrow m \leq 0$
$\Longrightarrow x!!m=y!!m$ by $\operatorname{simp}$
next
case (Suc n)
assume fst: $(\backslash m$. $(\bigwedge m$. $m \leq$ Suc $n \Longrightarrow$ geom-proc $m x=$ geom-proc $m y) \Longrightarrow$
$m \leq n \Longrightarrow x!!m=y!!m)$
and scd: $(\bigwedge m . m \leq \operatorname{Suc}($ Suc $n) \Longrightarrow$ geom-proc $m x=$ geom-proc $m y)$ and $m$
$\leq$ 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\leqSuc 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\not=y!!
thus False
proof (cases x !! m)
case True
hence \neg y !! m using <x !! m\not=y !! m> by simp
have geom-proc (Suc (Suc n)) x=u*geom-proc (Suc n) x using geomet-
ric-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\rangle\langle\neg y !! m> by simp
ultimately have geom-proc (Suc (Suc n)) x\not= 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 = y !! m> by simp
have geom-proc (Suc (Suc n)) x=d* geom-proc (Suc n) x using geomet-
ric-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\rangle\langley!! m> by simp
ultimately have geom-proc (Suc (Suc n)) x\not= 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 (\bigwedgem. m\leqn\Longrightarrowgeom-proc m x = geom-proc m y) \Longrightarrow
(pseudo-proj-True (n) x = pseudo-proj-True (n) y)

```
```

proof -
assume a1: \bigwedgem. m\leqn\Longrightarrowgeom-proc m x = geom-proc my
obtain nn :: bool stream }=>\mathrm{ bool stream }=>\mathrm{ nat }=>\mathrm{ nat where
\forall1 x2 x3. (\existsv4<Suc (Suc x3). geom-proc v4 x2 \# geom-proc v4 x1) =(nn x1
x2 x3 < Suc (Suc x3) ^ geom-proc (nn x1 x2 x3) x2 f= geom-proc (nn x1 x2 x3)
x1)
by moura
then have f2: \foralln s sa na. (nn sa s n<Suc (Suc n) ^ geom-proc (nn sa s n) s
\not= geom-proc (nn sa s n) sa\vee\negna<Suc n)\vee s!! na=sa!! na
by (meson geom-proc-eq-snth less-Suc-eq-le)
obtain nna :: bool stream }=>\mathrm{ bool stream }=>\mathrm{ nat }=>\mathrm{ nat where
f3: \forallx0 x1 x2. ( \existsv3. Suc v3 < Suc x2 ^ x1 !! v3 f= x0 !! v3) = (Suc (nna x0
x1 x2) < Suc x2 ^ x1 !! nna x0 x1 x2 \# = x0 !! nna x0 x1 x2)
by moura
obtain nnb :: nat }=>\mathrm{ nat where
f4: }\forallx0.(\existsv2.x0=Suc v2) =(x0=Suc (nnb x0))
by moura
moreover
{ assume }\negnnyx(nnb n)<Suc (Suc (nnb n))\vee geom-proc (nn y x (nnb n))
x= geom-proc (nn y x (nnb n)) y
moreover
{ assume \neg nna y x n<Suc (nnb n)
then have ᄀSuc (nna y x n) < Suc n \vee x !! nna y x n= y !! nna y x n
using f4 by (metis (no-types) Suc-le-D Suc-le-lessD less-Suc-eq-le) }
ultimately have pseudo-proj-True n x = pseudo-proj-True n y \vee ᄀSuc (nna
y x n) < Suc n \vee x !! nna y x n=y !! nna y x n
using f2 by meson }
ultimately have pseudo-proj-True n x = pseudo-proj-True n y \vee ᄀSuc (nna y
x n)<Suc n \vee x !! nna y x n=y !! nna y x n
using a1 Suc-le-D less-Suc-eq-le by presburger
then show ?thesis
using f3 by (meson less-Suc-eq-le pseudo-proj-True-snth')
qed

```
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 \(\operatorname{simp}\)
qed
thus ?thesis using geom-proc-eq-pseudo-proj-True[of may] by auto

\section*{qed}
lemma (in \(C R R\)-market-viable) rn-rev-price-cond-expect:
assumes \(N=\) bernoulli-stream \(q\)
and \(0<q\)
and \(q<1\)
and der \(\in\) borel-measurable ( \(G\) matur)
and Suc \(n \leq\) 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 \(-S u c n)(p s e u d o-p r o j-F a l s e ~ n w))\)
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) \(\in\) 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 \(n \in\) nat-filtration \(n \rightarrow_{M}\) 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 \(\operatorname{simp}\)
show proj-stoch-proc geom-proc \(n-‘\{\) proj-stoch-proc geom-proc \(n w\} \in\) 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 \(\operatorname{simp}\)
show \(\forall y z\). proj-stoch-proc geom-proc \(n y=\) proj-stoch-proc geom-proc \(n z \wedge y\) !! \(n=z!!n \longrightarrow\)
rn-rev-price \(N\) der matur (matur - Suc \(n\) ) \(y=r n\)-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 \(n z \wedge y\)
!! \(n=z\) !! \(n\)
hence pseudo-proj-True \(n y=\) pseudo-proj-True \(n z\) using proj-stoch-eq-pseudo-proj-True[of \(n y z]\) by \(\operatorname{simp}\)
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) \(z\)
proof -
have f1: \(\forall n\) s sa. \((\exists n a\). Suc \(n a \leq n \wedge s!!n a \neq s a!!n a) \vee\) pseudo-proj-True \(n s=\) pseudo-proj-True \(n\) sa
```

    by (meson pseudo-proj-True-snth')
    obtain nn :: bool stream }=>\mathrm{ bool stream }=>\mathrm{ nat }=>\mathrm{ nat where
    \forall0 x1 x2. ( \existsv3. Suc v3 \leqx2 ^ x1 !! v3 = x0 !! v3) = (Suc (nn x0 x1 x2)
    \leqx2 ^ x1 !! nn x0 x1 x2 = x0 !! nn x0 x1 x2)
by moura
then have f2: \foralln s sa.Suc (nn sa s n) \leqn^s !! nn sa s n\not= sa!! nn sa
s n \vee pseudo-proj-True n s=pseudo-proj-True n sa
using f1 by presburger
have f3: stake n y = stake n (pseudo-proj-True nz)
by (metis <pseudo-proj-True n y = pseudo-proj-True n z> pseudo-proj-True-stake)
{ assume stake (Suc n) z\not= stake (Suc n) (pseudo-proj-True (Suc n) y)
then have stake n y@ [y !! n] = stake nz @ [z !! n]
by (metis (no-types) pseudo-proj-True-stake stake-Suc)
then have stake (Suc n) z= stake (Suc n) (pseudo-proj-True (Suc n) y)
using f3 by (simp add: < y !! n = z !! n> pseudo-proj-True-stake) }
then have \negSuc (nnzy(Suc n)) \leqSuc n\vee y!!nn zy(Suc n) = z!! nn
zy(Suc n)
by (metis (no-types) pseudo-proj-True-stake stake-snth)
then show ?thesis
using f2 by blast
qed
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) \in 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) \in 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 \(C R R\)-market-viable) rn-price-eq-ind:
    assumes \(N=\) bernoulli-stream \(q\)
and \(n<\) matur
and \(0<q\)
and \(q<1\)
```

and der $\in$ 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-price $N$ der matur (Suc n) (pseudo-proj-False $n w)$
proof -
define $V$ where $V=r n$-price $N$ der matur
let ? $m=$ matur $-S u c n$
have matur $-n=$ Suc ?m by (simp add: assms Suc-diff-Suc Suc-le-lessD)
have $(1+r) * V n w=(1+r) * r n$-price-ind $N$ der matur $n w$ using rn-price-eq
assms unfolding $V$-def by simp
also have $\ldots=(1+r) * r n$-rev-price $N$ der matur (Suc ?m) $w$ using $<m a t u r ~-n$
= Suc ?m>
unfolding rn-price-ind-def by simp
also have $\ldots=(1+r) *$ discount-factor $r($ Suc 0$) w *$
expl-cond-expect $N$ (proj-stoch-proc geom-proc (matur - Suc ?m) $)$
(rn-rev-price $N$ der matur ?m) w
by $\operatorname{simp}$
also have $\ldots=$ expl-cond-expect $N$ (proj-stoch-proc geom-proc (matur - Suc
? $m$ )) (rn-rev-price $N$ der matur ?m) $w$
unfolding discount-factor-def using acceptable-rate by auto
also have $\ldots$. $=$ expl-cond-expect $N$ (proj-stoch-proc geom-proc $n$ ) (rn-rev-price $N$
der matur ?m) w
using 〈matur $-n=$ Suc ? $m$ 〉 by simp
also have $\ldots=(q * r n$-rev-price $N$ der matur ?m (pseudo-proj-True $n w)+$
$(1-q) *$ rn-rev-price $N$ der matur ?m (pseudo-proj-False $n w)$ )
using rn-rev-price-cond-expect $[$ of $N q$ der matur $n w]$ assms by simp
also have $\ldots=q *$ rn-price-ind $N$ der matur (Suc n) (pseudo-proj-True n w) +
$(1-q) * r n-p r i c e-i n d ~ N$ der matur (Suc n) (pseudo-proj-False $n$ w) unfolding
rn-price-ind-def by simp
also have $\ldots=q *$ rn-price $N$ der matur (Suc n) (pseudo-proj-True $n w)+$
( $1-q$ ) * rn-price $N$ der matur (Suc $n$ ) (pseudo-proj-False $n$ w) using
rn-price-eq assms by simp
also have $\ldots=q * V($ Suc $n)(p s e u d o-p r o j-T r u e ~ n w)+(1-q) * V($ Suc $n)$
(pseudo-proj-False $n$ w)
unfolding $V$-def by simp
finally have $(1+r) * V n w=q * V(S u c n)$ (pseudo-proj-True $n w)+(1-$
$q) * V$ (Suc $n$ ) (pseudo-proj-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-val-process Mkt (self-finance Mkt v pf asset) (Suc (Suc n)) w=
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)
    proof -
have cls-val-process Mkt (self-finance Mkt v pf asset) (Suc (Suc n)) w =cls-val-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 * remain-
ing-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 }\forallnw. prices Mkt asset n w\not=
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 O w)
proof -

```
have cls-val-process Mkt (self-finance Mkt vpf 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) wusing assms
by (simp add: self-finance-updated)
also have \(\ldots=\) 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 \(\operatorname{simp}\)
also have \(\ldots=\) cls-val-process Mkt pf (Suc 0) \(w+\)
prices Mkt asset (Suc 0) w* ((remaining-qty Mkt vpf asset 0 w) +
( \(v\) - val-process Mkt pf \(0 w) /(\) prices Mkt asset \(0 w))\)
by \(\operatorname{simp}\)
also have \(\ldots=\) 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 \(\ldots=\) cls-val-process Mkt pf (Suc 0) \(w+\)
prices Mkt asset (Suc 0) w * ((prices Mkt asset 0 w) * (remaining-qty Mkt vpf 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 \(\ldots=\) 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 vpf asset \(0 w)+\)
\(v\) - val-process Mkt pf \(0 w\) ) by \(\operatorname{simp}\)
also have \(\ldots=\) 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 vpf 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 \(\ldots=\) 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 \(\operatorname{simp}\)
finally show ?thesis .
qed
lemma self-finance-updated-ind:
assumes portfolio pf
and \(\forall n w\). prices Mkt asset \(n w \neq 0\)
shows cls-val-process Mkt (self-finance Mkt vpfasset) (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 \existsm. n=Suc m by (simp add: not0-implies-Suc)
from this obtain m}\mathrm{ 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) nw-
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 $n w \neq 0$ using positive by (metis less-irrefl)
qed
also have $\ldots=$ 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 \(r\) ( \(\lambda m\). der) matur)
have cls-val-process Mkt \(X\) (Suc \(n\) ) \(w=\) cls-val-process \(M k t\) (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 \(\ldots=\) geom-proc (Suc \(n\) ) \(w *\) delta-price \(N\) der matur \(n w+\)

w)
using delta-pf-updated by simp
also have \(\ldots=\) geom-proc \((\operatorname{Suc} n) w *\) 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 \(\Rightarrow\) real and matur::nat
assumes \(N=\) bernoulli-stream \(((1+r-d) /(u-d))\)
and der \(\in\) borel-measurable ( \(G\) matur)
shows \(\bigwedge n w . n \leq\) matur \(\Longrightarrow\)
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 \(q\) props \(=\) this
define init where init \(=(\) prob-space.expectation \(N\) (discounted-value \(r(\lambda m\). der) matur))
define \(X\) where \(X=\) val-process Mkt (delta-hedging \(N\) der matur)
define \(V\) where \(V=r n\)-price \(N\) der matur
define \(\Delta\) where \(\Delta=\) delta-price \(N\) der matur
\{
fix \(n\)
fix \(w\)
have \(n \leq\) matur \(\Longrightarrow X n w=V n w\)
proof (induct \(n\) )
case 0
have v0: V \(0 \in\) borel-measurable ( \(\left.\begin{array}{ll}G & 0\end{array}\right)\) 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 \(r\) ( \(\lambda m\). der) matur)] delta-pf-support
unfolding \(\quad X\)-def init-def delta-hedging-def self-fin-delta-pf-def init-def
by (metis finite-insert infinite-imp-nonempty infinite-super less-irrefl portfo-
lio-def positive)
also have \(\ldots=\) V 0 w
proof -

matur) \(x=\) integral \({ }^{L} N\) (discounted-value \(r(\lambda m\). 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 \(\operatorname{simp}\)
show infinite-cts-filtration p M nat-filtration using bernoulli-nat-filtration[of M p] bernoulli psgt pslt
by \(\operatorname{simp}\)
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 \(N\left(\begin{array}{ll}G & 0\end{array}\right)\) and sets \(\left(\begin{array}{ll}G & 0\end{array}\right)=\{\{ \}\), space \(N\}\) using stock-filtration unfolding init-triv-filt-def
filtration-def bot-nat-def by auto
show integrable \(N\) (discounted-value \(r\) ( \(\lambda m\). der) matur)
proof (rule bernoulli-discounted-integrable)
show der \(\in\) 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 \(r(\lambda m\). der \()\) matur \()=\)
 bernoulli-stream-space[of \(N q]\)
by (simp add: assms(1) \(q\)-def)
also have \(\ldots=\) real-cond-exp \(N\) (stoch-proc-filt M geom-proc borel 0) (discounted-value \(r(\lambda m\). der \()\) matur \() w\)
using stock-filtration by simp
also have \(\ldots=\) real-cond-exp \(N\) (stoch-proc-filt \(N\) geom-proc borel 0) (discounted-value \(r(\lambda m . d e r)\) matur \() w\)
using stoch-proc-filt-filt-equiv[of nat-filtration \(M\) N geom-proc]
bernoulli-stream-equiv \([o f N] q\)-def qprops assms pslt psgt by auto
also have \(\ldots=\) expl-cond-expect \(N\) (proj-stoch-proc geom-proc 0) (discounted-value \(r\) ( \(\lambda\) m. 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 \(r\) ( \(\lambda m\). der) matur)
proof (rule bernoulli-discounted-integrable)
show der \(\in\) 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

\section*{qed}
finally show \(X 0 w=V 0 w\).
next
case (Suc n)
hence \(n<\) matur by simp
```

        show ?case
    proof -
    have Xnw = V nw 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)*Vnw=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
wn 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*\Delta nw+(1 +r)* (Xn w -
geom-proc n w*\Delta nw)
using delta-hedging-cls-val-process unfolding X-def }\Delta\mathrm{ -def by simp
also have ... = u* geom-proc n w*\Delta nw+(1 +r)* (X nw-
geom-proc n w*\Delta nw)
using True geometric-process by simp
also have ...=u* geom-proc n w*\Delta nw+(1+r)*Xnw-(1+r)

* geom-proc n w *\Delta n w
by (simp add: right-diff-distrib)
also have ... = (1+r)*Xnw+geom-proc n w*\Delta n w*u-geom-proc
nw*\Deltan w* (1 +r)
by (simp add: mult.commute mult.left-commute)
also have ... =(1+r)*Xnw+geom-proc n w*\Delta nw* (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: \Delta-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 \(\ldots=(1+r) * X n w+((V\) (Suc \(n)\) (pseudo-proj-True \(n w)-\) \(V(\) Suc \(n)(\) pseudo-proj-False \(n w)) *(u-(1+r)) /(u-d)\)

\section*{proof -}
have geom-proc n \(w \neq 0\)
by (metis SO-positive down-lt-up down-positive geom-rand-walk-strictly-positive less-irrefl)

\section*{then show ?thesis}
by \(\operatorname{simp}\)
qed
also have \(\ldots=(1+r) * X n w+((V\) (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 \(\ldots=(u-d) /(u-d)-(1+r-d) /(u-d)\) using down-lt-up by simp
also have \(\ldots=(u-d-(1+r-d)) /(u-d)\) using diff-divide-distrib[of \(u-d 1+r-d]\) by simp
also have \(\ldots=(u-(1+r)) /(u-d)\) by simp
finally have \(1-q=(u-(1+r)) /(u-d)\).
thus ?thesis by simp
qed
also have \(\ldots=(1+r) * X n w+(1-q) * V(S u c n)\) (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 \(\ldots=(1+r) * V n w+(1-q) * V(S u c n)\) (pseudo-proj-True \(n w)\) -
\((1-q) * V(\) Suc \(n)(\) pseudo-proj-False \(n w)\) using \(\langle X n w=V n w\rangle\) by simp
also have \(\ldots=q * V(\) Suc \(n)(\) pseudo-proj-True \(n w)+(1-q) * V(S u c\) n) \((\) pseudo-proj-False \(n w)+\)
\[
(1-q) * V(\text { Suc } n)(\text { pseudo-proj-True } n w)-(1-q) * V(\text { Suc } n)
\] (pseudo-proj-False n w)
using assms Suc rn-price-eq-ind \([\) of \(N q n\) matur der \(w]\langle n<\) matur \(\rangle\) qprops unfolding \(V\)-def \(q\)-def
by \(\operatorname{simp}\)
also have \(\ldots=q * V(\) Suc \(n)(\) pseudo-proj-True \(n w)+(1-q) * V(\) Suc n) (pseudo-proj-True \(n w\) ) by simp
also have \(\ldots=V(\) Suc \(n)(\) pseudo-proj-True \(n w)\)
using distrib-right[of q1-qV(Suc n) (pseudo-proj-True n w)] by simp
also have \(\ldots=V(\) Suc \(n) w\)
proof -
have \(V(\) Suc \(n) \in\) borel-measurable ( \(G\) (Suc n)) unfolding \(V\)-def \(q\)-def proof (rule rn-price-borel-adapt)
show der \(\in\) 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 \(n \leq\) 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 \(\ldots=V(S u c n)(\) pseudo-proj-True (Suc n) w) using True
by (simp add: pseq spick-eq-pseudo-proj-True)
also have \(\ldots=V(\) Suc \(n) w\) using \(\langle V(\) Suc \(n) \in \operatorname{borel-measurable~}(G\)
(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 \(\ldots=\) geom-proc \((\) Suc \(n) w * \Delta n w+(1+r) *(X n w-\) geom-proc \(n w * \Delta n w\) )
using delta-hedging-cls-val-process unfolding \(X\)-def \(\Delta\)-def by simp
also have \(\ldots=d *\) geom-proc \(n w * \Delta n w+(1+r) *(X n w-\) geom-proc \(n w * \Delta n w\) )
using False geometric-process by simp
also have \(\ldots=d *\) geom-proc \(n w * \Delta n w+(1+r) * X n w-(1+r)\) * geom-proc \(n w * \Delta n w\)
by (simp add: right-diff-distrib)
also have \(\ldots=(1+r) * X n w+\) geom-proc \(n w * \Delta n w * d\) - geom-proc \(n w * \Delta n w *(1+r)\)
by (simp add: mult.commute mult.left-commute)
also have \(\ldots=(1+r) * X n w+\) geom-proc \(n w * \Delta n w *(d-(1+r))\) by (simp add: right-diff-distrib)
also have \(\ldots=(1+r) * X n w+\) geom-proc \(n w *\left(\begin{array}{l}\text { (Suc } n)\end{array}\right.\)
(pseudo-proj-True \(n w)-V(\) Suc \(n)(p s e u d o-p r o j-F a l s e ~ n w)) /\)
(geom-proc (Suc n) (spick w True) - geom-proc (Suc n) (spick wn False \()\) ) \((d-(1+r))\)
using Suc \(V\)-def by (simp add: \(\Delta\)-def delta-price-def geom-rand-walk-diff-induct)
also have \(\ldots=(1+r) * X n w+\) 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 \(\ldots=(1+r) * X n w+((V\) (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 \(w \neq 0\)
by (metis S0-positive down-lt-up down-positive geom-rand-walk-strictly-positive less-irrefl)
then show ?thesis
by simp
qed
also have \(\ldots=(1+r) * X n w+((V\) (Suc \(n)\) (pseudo-proj-True \(n w)-\) \(V(\) Suc \(n)(\) pseudo-proj-False \(n\) w) \() *(-q))\)
proof -
have \(0-q=0-(1+r-d) /(u-d)\) unfolding \(q\)-def by simp
also have \(\ldots=(d-(1+r)) /(u-d)\) by (simp add: minus-divide-left)
finally have \(0-q=(d-(1+r)) /(u-d)\).
thus?thesis by simp
qed
also have \(\ldots=(1+r) * X n w+(-V(S u c n)(p s e u d o-p r o j-T r u e ~ n w) *\) \(q+V(\) Suc \(n)(p s e u d o-p r o j-F a l s e ~ n ~ w) * ~ q) ~\)
by (metis (no-types, opaque-lifting) add.inverse-inverse distrib-right minus-mult-commute minus-real-def mult-minus-left)
also have \(\ldots=(1+r) * X n w-q * V(S u c n)\) (pseudo-proj-True \(n w)\) \(+q * V(\) Suc \(n)\) (pseudo-proj-False \(n w)\) by simp
also have \(\ldots=(1+r) * V n w-q * V\) (Suc n) (pseudo-proj-True \(n w)+\) \(q * V(\) Suc \(n)\) (pseudo-proj-False \(n w\) ) using \(\langle X n w=V n w\rangle\) by simp
also have \(\ldots=q * V(S u c n)(\) pseudo-proj-True \(n w)+(1-q) * V(S u c\) n) (pseudo-proj-False \(n\) w) -
\(q * V(\) Suc \(n)(\) pseudo-proj-True \(n w)+q * V(\) Suc \(n)(p s e u d o-p r o j-F a l s e\) \(n w)\)
using assms Suc rn-price-eq-ind[of \(N q\) matur der \(w]\langle n<\) matur \(\rangle\) qprops unfolding \(V\)-def \(q\)-def
by simp
also have \(\ldots=(1-q) * V(\) Suc \(n)\) (pseudo-proj-False \(n w)+q * V(\) Suc \(n\) ) (pseudo-proj-False \(n w\) ) by simp
also have \(\ldots=V(\) Suc \(n)(\) pseudo-proj-False \(n w)\)
using distrib-right[of \(q 1-q V(\) Suc \(n)\) (pseudo-proj-False \(n\) \(w)]\) by simp
also have \(\ldots=V(\) Suc \(n) w\)
proof -
have \(V(\) Suc \(n) \in\) borel-measurable \((G(S u c n))\) unfolding \(V\)-def \(q\)-def proof (rule rn-price-borel-adapt)
show der \(\in\) 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 \(n \leq\) matur using Suc by simp
qed
hence \(V(\) Suc \(n)\) (pseudo-proj-False \(n w)=V\) (Suc \(n\) ) (pseudo-proj-False (Suc n) (pseudo-proj-False \(n\) w))
using geom-proc-filt-info' \([\) of \(V\) (Suc n) Suc n] by simp
also have \(\ldots=V\) (Suc n) (pseudo-proj-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-measurable ( }
    (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 \nw. n\leqmatur \Longrightarrow
val-process Mkt (self-fin-delta-pf N der matur (integral }\mp@subsup{}{}{L}N\mathrm{ (discounted-value
r(\lambdam.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 }\in\mathrm{ borel-measurable ( }G\mathrm{ 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}\mathrm{ where }q=(1+r-d)/(u-d
have }0<q\mathrm{ 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 \in 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 }\forallnw.prices Mkt risk-free-asset (Suc n) w\not=0 using positiv
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 \in borel-measurable (G matur)
and N}=\mathrm{ 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}\mathrm{ 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 unfold-
ing 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 \longleftrightarrow(\forall matur. }\forall\mathrm{ der }\in\mathrm{ borel-measurable (F matur). ( }\exists\mathrm{ p. replicat-
ing-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 }\forall\mathrm{ der }\in\mathrm{ borel-measurable ( }G\mathrm{ matur ). ( }\exists\mathrm{ p.replicating-portfolio p der matur)
proof
fix der::bool stream }=>\mathrm{ real
assume der }\in\mathrm{ borel-measurable ( }G\mathrm{ 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 \in borel-measurable (G matur)> by simp
thus \exists pf. replicating-portfolio pf der matur by auto

```

\section*{qed}
qed
```

lemma subalgebras-filtration:
assumes filtration M F
and }\forallt\mathrm{ . subalgebra (Ft) (Gt)
and}\forallst.s\leqt\longrightarrow\mathrm{ subalgebra (Gt) (Gs)
shows filtration M G unfolding filtration-def
proof (intro conjI allI impI)
{
fix }
have subalgebra (F t) (Gt) using assms by simp
moreover have subalgebra M (Ft) using assms unfolding filtration-def by
simp
ultimately show subalgebra M (Gt) by (metis subalgebra-def subsetCE sub-
setI)
}
{
fix st::'b
assume s\leqt
thus subalgebra (G t) (Gs) 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 \([o f ~ M F G]\) 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 \(A \in\) sets \((G t)\)
    hence \(A \in\) 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 \(\in\) borel-measurable ( \(G\) matur)

\section*{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 \(r\) ( \(\lambda\) m. pyf) matur) \(w)\) )
pyf matur
proof -
define dpf where \(d p f=(\) discounted-value \(r\) ( \(\lambda\) m. pyf) matur \()\)
define \(q\) where \(q=(1+r-d) /(u-d)\)
have \(\exists p f\). 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 \(M k t\) (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 \(C R R\)-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 \(\in\) borel-measurable ( \(G\) matur) using assms by simp
show viable-market Mkt using CRR-viable by simp
have infinite-cts-filtration p Mat-filtration using bernoulli-nat-filtration[of M
p] bernoulli psgt pslt
by simp
thus sets \(\left(\begin{array}{ll}G & 0\end{array}\right)=\{\{ \}\), 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 \(\forall n . \forall\) asset \(\in\) support-set pf. finite (prices Mkt asset \(n\) 'space M)
proof (intro allI ballI)
fix \(n\)
fix asset
assume asset \(\in\) support-set pf
hence prices Mkt asset \(n \in\) borel-measurable ( \(G n\) ) using readable pfprop unfolding replicating-portfolio-def stock-portfolio-def adapt-stoch-proc-def

\section*{by auto}
hence prices Mkt asset \(n \in\) borel-measurable ( \(n a t\)-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 \(\forall n\). \(\forall\) asset \(\in\) support-set \(p f\). finite ( \(p f\) asset \(n\) ' space \(M\) )
proof (intro allI ballI)
fix \(n\)
```

    fix asset
    assume asset \in support-set pf
    hence pf asset n \in 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 \in 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}Ndpf
(\sumw\in 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 \in borel-measurable (G matur) using assms discounted-measurable[of
pyf G matur]
unfolding dpf-def by simp
thus dpf \in 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

```

\section*{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.

```

\section*{9 Effective computation definitions and results}

\subsection*{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 $0=\{[]\}$
$\mid$ gener-bool-list $($ Suc $n)=\{$ True $\# w \mid w . w \in$ gener-bool-list $n\} \cup\{$ False $\# w \mid w$.
$w \in$ gener-bool-list $n\}$
lemma gener-bool-list-elem-length:
shows $\backslash x . x \in$ gener-bool-list $n \Longrightarrow$ length $x=n$
proof (induction $n$ )
case 0
fix $x$
assume $x \in$ gener-bool-list 0
hence $x=[]$ by $\operatorname{simp}$
thus length $x=0$ by simp
next
case (Suc n)
fix $x$
assume $x \in$ gener-bool-list (Suc n)
hence mem: $x \in\{$ True $\# w \mid w . w \in$ gener-bool-list $n\} \cup\{$ False $\# w \mid w . w \in$
gener-bool-list $n\}$ by simp
show length $x=$ Suc $n$
proof (cases $x \in\{$ True $\# w \mid w . w \in$ gener-bool-list $n\}$ )
case True
hence $\exists w \in$ gener-bool-list $n . x=$ True $\# w$ by auto
from this obtain $w$ where $w \in$ gener-bool-list $n$ and $x=$ True $\# w$ by auto
hence length $w=n$ using Suc by simp
thus length $x=$ Suc $n$ using $\langle x=$ True $\# w\rangle$ by simp
next
case False
hence $x \in\{$ False $\# w \mid w$. $w \in$ gener-bool-list $n\}$ using mem by auto
hence $\exists w \in$ gener-bool-list $n . x=$ False $\# w$ by auto
from this obtain $w$ where $w \in$ gener-bool-list $n$ and $x=$ False $\# w$ by auto
hence length $w=n$ using Suc by simp
thus length $x=$ Suc $n$ using $\langle x=$ False $\# w\rangle$ 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| sw. s\inUNIV ^ w\in
(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| sw.s\in{True, False } ^w\in(stake n'(streams UNIV))}
by simp
also have ... = {s\#w| s w.s\in{True, False} ^ w\in gener-bool-list n} using
Suc by simp
also have ... = {s\#w| sw. s\in{True} ^ w\in gener-bool-list n} \cup{s\#w| sw.
s\in{ False} ^ w\in gener-bool-list n} by auto
also have ... = {True \#w|w.w\in gener-bool-list n} \cup{False\#w | w.w\in
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.fw=g(stake n w)
shows (\sumw\in range (pseudo-proj-True n).f w)=(\sumy\in(gener-bool-list n).g
y)
proof (rule sum.reindex-cong)
show inj-on (\lambda l. shift l (sconst True)) (gener-bool-list n)
proof
fix }x
assume x\in gener-bool-list n
and y\in gener-bool-list n
and x@- sconst True = y@- sconst True
have length x = n using gener-bool-list-elem-length <x\in gener-bool-list n〉 by
simp
have length y = n using gener-bool-list-elem-length 〈y\in gener-bool-list n> by
simp
show }x=
proof -
have }\foralli<n. nth xi=nth y
proof (intro allI impI)
fix }
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 \(\langle l e n g t h ~ x=n\rangle\langle l e n g t h ~ y=n\rangle\) by (simp add: list-eq-iff-nth-eq) qed
qed
have range (pseudo-proj-True \(n)=\{\) shift \(l\) (sconst True) \(\mid l\). \(l \in(\) stake \(n\) 'streams (UNIV::bool set))
unfolding pseudo-proj-True-def by auto
also have \(\ldots=\{\) shift \(l\) (sconst True) \(\mid\) l. \(l \in(\) gener-bool-list \(n)\}\) using stake-gener-bool-list by \(\operatorname{simp}\)
also have \(\ldots=(\lambda l . l\) @- sconst True)'gener-bool-list \(n\) by auto
finally show range (pseudo-proj-True \(n)=(\lambda l . l @-\) sconst True)'gener-bool-list \(n\).
fix \(x\)
assume \(x \in\) gener-bool-list \(n\)
have length \(x=n\) using gener-bool-list-elem-length \(\langle x \in\) gener-bool-list \(n\rangle\) by simp
have \(f(x @-\) sconst True \()=g(\) stake \(n(x @-\) sconst True \())\) using assms by simp
also have \(\ldots=g x\) using «length \(x=n\) 〉 by (simp add: stake-shift)
finally show \(f(x @-\) sconst True \()=g x\).
qed

\subsection*{9.2 Probability components for lists}
fun lprob-comp where
lprob-comp ( \(p:\) :real) []\(=1\)
|lprob-comp \(p(x \# x s)=(\) if \(x\) then \(p\) else \((1-p)) *\) lprob-comp \(p x s\)
lemma lprob-comp-last:
shows lprob-comp \(p(x s @[x])=(\) lprob-comp \(p x s) *(\) 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 \(\ldots=(\) if \(x\) then \(p\) else \((1-p))\) by simp
also have \(\ldots=(\) 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 \(p(x s @[x])\) by \(\operatorname{simp}\)
also have \(\ldots=(\) if a then \(p\) else \((1-p)) *(\) lprob-comp \(p x s) *(\) if \(x\) then \(p\) else \((1-p))\) using Cons by simp
also have \(\ldots=\) 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.atLeastO-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

```

\subsection*{9.3 Geometric process applied to lists}
fun lrev-geom where
lrev-geom \(u d v[]=v\)
\(\mid\) lrev-geom ud \(v(x \# x s)=(\) if \(x\) then \(u\) else \(d) *\) lrev-geom udves
fun lgeom-proc where lgeom-proc \(u d v l=l\) lev-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 \(\ldots=\) lrev-geom \(u d v\) [] by simp
also have \(\ldots=\) lrev-geom udv (rev (stake 0 w) ) by simp
also have \(\ldots=\) lgeom-proc \(u d v\) (stake 0 w ) by simp
finally show geom-rand-walk udvow=lgeom-proc udv (stake \(0 w)\).
next
case (Suc n)
have snth \(w n=n\)th (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 udv (Suc n) w \(=((\lambda\) True \(\Rightarrow u \mid\) False \(\Rightarrow d)(\) snth \(w n))\)
* geom-rand-walk udv \(n\) w by simp
```

    also have ... = (if (snth wn) then u else d)* geom-rand-walk u d v n w by simp
    also have }\ldots=(\mathrm{ if (snth wn) then u else d)*lgeom-proc udv (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 udv(rev((stake n w)@ [snth wn])) by simp
also have ... = lrev-geom udv(rev((stake nw)@ [nth (stake (Suc n)w) n]))
using <snth wn = 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 udv (Suc n) w = lgeom-proc u d v (stake (Suc n)
w).
qed
lemma lgeom-proc-take:
assumes }i\leq
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

```

\subsection*{9.4 Effective computation of discounted values}

\section*{fun det-discount where}
det-discount ( \(r::\) real) \(0=1\)
\(\mid\) det-discount \(r(\) Suc \(n)=(\) inverse \((1+r)) *(\) det-discount \(r n)\)
```

lemma det-discounted:
shows discounted-value $r X n w=($ det-discount $r n) *\left(\begin{array}{lll}X & n & w\end{array}\right)$ 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 $\ldots=($ det-discount r 0$) *\left(\begin{array}{ll}X & 0 \\ \text { w }\end{array}\right)$ by simp
finally show inverse (disc-rfr-proc r $0 w) * X 0 w=($ det-discount r 0$) *\left(\begin{array}{ll}X & 0\end{array}\right.$
w) .
next
case (Suc n)
have inverse (disc-rfr-proc r (Suc n) w) * (Suc n) $w=$
inverse $((1+r) *($ disc-rfr-proc $r) n w) * X(S u c n) w$ by simp
also have $\ldots=($ inverse $(1+r)) *$ inverse $(($ disc-rfr-proc r) $n w) * X($ Suc $n) w$
by $\operatorname{simp}$
also have $\ldots=($ inverse $(1+r)) *($ det-discount $r n) * X($ Suc $n) w$ using Suc[of
$\lambda n . X(S u c n)]$ by auto

```
also have \(\ldots=(\) 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

\section*{10 Pricing results on options}

\subsection*{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 \((\mathrm{S}\) - K, 0) at that time.
definition (in CRR-market) call-option where
call-option \((T:: n a t)(K::\) real \()=(\lambda w . \max (\) prices Mkt stk \(T w-K) 0)\)
lemma (in CRR-market) call-borel:
shows call-option \(T K \in\) borel-measurable ( \(G T\) ) unfolding call-option-def
proof (rule borel-measurable-max)
show \((\lambda x .0) \in\) borel-measurable \((G T)\) by simp
show \((\lambda x\). prices Mkt stk \(T x-K) \in\) borel-measurable \((G T)\)
proof (rule borel-measurable-diff)
show prices Mkt stk \(T \in\) borel-measurable ( \(G T\) )
by (metis adapt-stoch-proc-def stock-price-borel-measurable)
qed \(\operatorname{simp}\)
qed
lemma (in CRR-market-viable) call-option-lgeom:
shows call-option \(T K w=\max ((l g e o m-p r o c u d i n i t(s t a k e ~ T w))-K) 0\)
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 \(r(\lambda m\). (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 \(\lambda m\). call-option \(T K T w]\) call-option-lgeom[of \(T K\)
\(w]\) by \(\operatorname{simp}\)
lemma (in CRR-market-viable) call-effect-compute:
shows ( \(\sum w \in\) range (pseudo-proj-True matur). (prod (prob-component pr \(w\) ) \(\{0 . .<\) matur \(\}\) ) *
(discounted-value \(r(\lambda m\). (call-option matur \(K))\) matur \(w))=\)
( \(\sum y \in(\) 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 \(r(\lambda m\). call-option matur K) matur \(w=\)
lprob-comp pr (stake matur \(w) *\) discounted-value \(r\) ( \(\lambda m\). call-option matur K) matur \(w\)
using lprob-comp-stake by simp
also have \(\ldots=\) 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 \(r(\lambda m\). 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 \(K=\left(\sum y \in(\right.\) gener-bool-list matur \()\). lprob-comp \(((1+\) \(r-d) /(u-d)) y *(\) det-discount \(r\) matur \() *\)
\((\max ((\) lgeom-proc ud 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
( \(\sum w \in\) range (pseudo-proj-True matur). (prod (prob-component \(((1+r-d)\) \(/(u-d)) w)\{0 . .<\) matur \(\}) *\)
(discounted-value \(r(\lambda m\). (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

\subsection*{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 \(\max (\mathrm{K}\) - \(\mathrm{S}, 0\) ) at that time.
definition (in \(C R R\)-market) put-option where
put-option \((T:: n a t)(K::\) real \()=(\lambda w . \max (K-\) prices Mkt stk \(T w) 0)\)
lemma (in CRR-market) put-borel:
shows put-option \(T K \in\) borel-measurable \((G T)\) unfolding put-option-def proof (rule borel-measurable-max)
show \((\lambda x .0) \in\) borel-measurable \((G T)\) by simp
show \((\lambda x . K-p r i c e s ~ M k t ~ s t k ~ T x) \in\) borel-measurable \((G T)\)
proof (rule borel-measurable-diff)
show prices Mkt stk \(T \in\) borel-measurable ( \(G T\) )
by (metis adapt-stoch-proc-def stock-price-borel-measurable)

> qed \(\operatorname{simp}\)
> qed
lemma (in CRR-market-viable) put-option-lgeom:
shows put-option \(T K w=\max (K-(\) lgeom-proc ud 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 \(r(\lambda m\). (put-option \(T K)) T w)=\)
(det-discount r \(T) *(\max (K-(l g e o m-p r o c ~ u d i n i t ~(s t a k e ~ T w))) ~ 0) ~\)
using det-discounted[of r \(\lambda\) m. put-option \(T K T w]\) put-option-lgeom[of T K w]
by \(\operatorname{simp}\)
lemma (in CRR-market-viable) put-effect-compute:
shows ( \(\sum w \in\) range (pseudo-proj-True matur). (prod (prob-component pr w) \(\{0 . .<\) matur \(\}\) ) *
(discounted-value \(r(\lambda m\). (put-option matur \(K))\) matur \(w))=\)
( \(\sum y \in\) (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 \(r\) ( \(\lambda\) m. put-option
matur \(K\) ) matur \(w=\)
lprob-comp pr (stake matur \(w) *\) discounted-value \(r\) ( \(\lambda\) m. put-option matur \(K\) )
matur \(w\)
using lprob-comp-stake by simp
also have \(\ldots=\) 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 \(r(\lambda m\). put-option matur \(K\) ) matur \(w=\)
lprob-comp pr (stake matur w) *

matur \(w)\) ))) 0).
qed
fun put-price where
put-price \(u\) d init \(r\) matur \(K=\left(\sum y \in\right.\) (gener-bool-list matur \()\). lprob-comp \(((1+\) \(r-d) /(u-d)) y *(\) det-discount \(r\) matur \() *\)
\((\max (K-(\) lgeom-proc ud 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
( \(\sum w \in\) range (pseudo-proj-True matur). (prod (prob-component \(((1+r-d)\) \(/(u-d)) w)\{0 . .<\) matur \(\})\) *
(discounted-value \(r\) ( \(\lambda m\). (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

\subsection*{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 \(\max (\mathrm{Sn} .0<=\mathrm{n}\) \(<=\mathrm{T})\) - ST at that time.
definition (in CRR-market) lbk-option where
lbk-option \((T:: n a t)=(\lambda w . \operatorname{Max}((\lambda i .(\) prices Mkt stk \() i w)\{0\).. \(T\})-(\) prices Mkt stk \(T w)\) )
lemma borel-measurable-Max-finite:
fixes \(f::^{\prime} a \Rightarrow{ }^{\prime} b \Rightarrow^{\prime} c::\{\) second-countable-topology, linorder-topology\}
assumes \(0<(n:: n a t)\)
shows \(\bigwedge A\). card \(A=n \Longrightarrow \forall a \in A . f a \in\) borel-measurable \(M \Longrightarrow(\lambda w\). Max \(\left.\left((\lambda a . f a w)^{\prime} A\right)\right) \in\) borel-measurable \(M\) using assms
proof (induct \(n\) )
case 0
show \(\bigwedge A\). card \(A=0 \Longrightarrow \forall a \in A . f a \in\) borel-measurable \(M \Longrightarrow(0:: n a t)<0\) \(\Longrightarrow\left(\lambda w . \operatorname{Max}\left((\lambda a . f a w)^{\prime} A\right)\right) \in\) borel-measurable \(M\)

> proof -
fix \(A::^{\prime}\) a set
assume card \(A=0\) and \(\forall a \in A . f a \in\) borel-measurable \(M\) and ( \(0:: n a t\) ) \(<0\) thus \((\lambda w . \operatorname{Max}((\lambda a . f a w) ‘ A)) \in\) borel-measurable \(M\) by simp qed
next
case Suc
show \(\bigwedge n A .(\bigwedge A\). card \(A=n \Longrightarrow\)
\(\forall a \in A . f a \in\) borel-measurable \(M \Longrightarrow 0<n \Longrightarrow(\lambda w . \operatorname{Max}((\lambda a . f a\)
w) ' \(A\) ) \() \in\) borel-measurable \(M) \Longrightarrow\)
card \(A=\) Suc \(n \Longrightarrow\)
\(\forall a \in A . f a \in\) borel-measurable \(M \Longrightarrow 0<\) Suc \(n \Longrightarrow(\lambda w . \operatorname{Max}((\lambda a . f a\)
w) ' \(A\) )) \(\in\) borel-measurable \(M\)
proof -
fix \(n\)
fix \(A::^{\prime} a\) set
assume ameas: \((\bigwedge A\). card \(A=n \Longrightarrow\)
\(\forall a \in A . f a \in\) borel-measurable \(M \Longrightarrow 0<n \Longrightarrow(\lambda w . \operatorname{Max}((\lambda a . f a\)
w) ' \(A\) ) \() \in\) borel-measurable \(M\) )
and card \(A=\) Suc \(n\)
and \(\forall a \in A . f a \in\) borel-measurable \(M\)
and \(0<\) Suc \(n\)
from 〈card \(A=\) Suc \(n\rangle\) have aprop：\(A \neq\{ \} \wedge\) finite \(A\) using card－eq－0－iff［of A］by \(\operatorname{simp}\)
hence \(\exists x . x \in A\) by auto
from this obtain \(a\) where \(a \in A\) by auto
hence Suc（card \((A-\{a\}))=\) Suc \(n\) using aprop card－Suc－Diff1 \([\) of \(A]\)＜card \(A=\) Suc \(n>\) by auto
hence \(\operatorname{card}(A-\{a\})=n\) by simp
show \((\lambda w . \operatorname{Max}((\lambda a . f a w) ‘ A)) \in\) borel－measurable \(M\)
proof（cases \(n=0\) ）
case False
hence \(0<n\) by \(\operatorname{simp}\)
moreover have \(\forall a \in A-\{a\}\) ．\(f a \in\) borel－measurable \(M\) using \(\langle\forall a \in A\) ．\(f a\) \(\in\) borel－measurable \(M>\) by simp
ultimately have \(\left(\lambda w . \operatorname{Max}\left((\lambda a . f a w)^{\prime}(A-\{a\})\right)\right) \in\) borel－measurable \(M\) using \(\langle\operatorname{card}(A-\{a\})=n\rangle\)
ameas［of \(A-\{a\}]\) by \(\operatorname{simp}\)
moreover have fáborel－measurable \(M\) using \(\langle\forall a \in A\) ．f \(a \in\) borel－measurable \(M\rangle\langle a \in A\rangle\) by simp
ultimately have \((\lambda w . \max (f a w)(\operatorname{Max}((\lambda a . f a w) '(A-\{a\})))) \in\) borel－measurable \(M\)
using borel－measurable－max by simp
moreover have \(\Lambda w . \max (f a w)\left(\operatorname{Max}\left((\lambda a . f a w)^{\prime}(A-\{a\})\right)\right)=\operatorname{Max}\) （（ \(\lambda a . f a w)\)＇\(A\) ）
proof－
fix \(w\)
define \(F A\) where \(F A=\left((\lambda a . f a w)^{\prime}(A-\{a\})\right)\)
have finite FA unfolding FA－def using aprop by simp
have \(A-\{a\} \neq\{ \}\) using aprop False \(\langle\) card \((A-\{a\})=n\rangle\) card－eq－0－iff［of
\(A-\{a\}]\) by simp
hence \(F A \neq\{ \}\) unfolding \(F A\)－def by simp
have \(\max (f a w)(\operatorname{Max} F A)=\operatorname{Max}(\operatorname{insert}(f a w) F A)\) using〈finite \(F A\) 〉 \(\langle F A \neq\{ \}\rangle\) by \(\operatorname{simp}\)
hence \(\max (f a w)(\operatorname{Max}((\lambda a . f a w) ‘(A-\{a\})))=\operatorname{Max}(\operatorname{insert}(f a w)\)
\(((\lambda a . f a w) \cdot(A-\{a\})))\)
unfolding FA－def by simp
also have \(\ldots=\operatorname{Max}((\lambda a . f a w)\)＇\(A\) ）
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 \(\ldots=\left((\lambda a . f a w)^{\prime} A\right)\) using \(\langle a \in A\rangle\) 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 \(\max (f a w)(\operatorname{Max}((\lambda a . f a w) \cdot(A-\{a\})))=\operatorname{Max}((\lambda a . f a\)
w）＇\(A\) ）．
qed
ultimately show \((\lambda w . \operatorname{Max}((\lambda a . f a w) ' A)) \in\) borel－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 ... = A using <a\inA\rangle by blast
finally have {a}=A .
hence }\Lambdaw.(\lambdaa.faw)'A={faw} by aut
hence }\w.\operatorname{Max}((\lambdaa.faw)'A)=Max{faw} by sim
hence }\w.\operatorname{Max}((\lambdaa.faw)'A)=faw by sim
hence ( }\lambdaw.\operatorname{Max}((\lambdaa.faw)'A))=fa by sim
thus (\lambdaw. Max ((\lambdaa.f a w)'A)) \in borel-measurable M using }\forall \A\inA.fa
borel-measurable M>
<a\in A> by simp
qed
qed
qed
lemma (in CRR-market) lbk-borel:
shows lbk-option T \in borel-measurable (GT) unfolding lbk-option-def
proof (rule borel-measurable-diff)
show (\lambdax. Max ((\lambdai.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 }\foralli\in{0..T}.prices Mkt stk i\in borel-measurable (GT
proof
fix }
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 increas-
ing-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 ((\lambdai.(lgeom-proc u d init (stake i w))){0 .. T})

- (lgeom-proc u d init (stake Tw))
using geom-lgeom stk-price geometric-process unfolding lbk-option-def by simp
lemma (in CRR-market-viable) disc-lbk-option-lgeom:
shows (discounted-value r (\lambdam. (lbk-option T)) Tw)=
(det-discount r T) * (Max ((\lambdai. (lgeom-proc u d init (take i (stake T w))))`{0
.. T}) - (lgeom-proc u d init (stake T w)))

```
using det-discounted[of \(r \lambda m\). lbk-option \(T \quad T \quad w]\) 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-proj-True matur). (prod (prob-component pr w) \(\{0 . .<\) matur \(\}) *\)
(discounted-value \(r(\lambda\) m. (lbk-option matur \())\) matur \(w))=\)
( \(\sum y \in\) (gener-bool-list matur). lprob-comp pr \(y *(\) det-discount \(r\) matur \() *\)
(Max \(((\lambda i\). (lgeom-proc u d init \((\) take \(i \operatorname{y}))) ‘\{0\).. matur \(\})\) - (lgeom-proc ud init \(y\) )))
proof (rule pseudo-range-stake)
fix \(w\)
have \(\operatorname{prod}\) (prob-component pr \(w\) ) \(\{0 . .<\) matur \(\} *\) discounted-value \(r(\lambda m\). lbk-option matur) matur \(w=\)
lprob-comp \(\operatorname{pr}\) (stake matur \(w) *\) discounted-value \(r\) ( \(\lambda\) m. lbk-option matur) matur \(w\)
using lprob-comp-stake by simp
also have \(\ldots=\) lprob-comp pr (stake matur \(w) *\)
(det-discount \(r\) matur \() *(\operatorname{Max}((\lambda i\). (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 \(r(\lambda m\). lbk-option matur) matur \(w=\)
lprob-comp pr (stake matur w) *
(det-discount \(r\) matur ) * (Max (( \(\lambda\) i. (lgeom-proc ud init (take \(i\) (stake matur \(w)))\) ) \(\{0\).. matur \(\})-\)
(lgeom-proc u d init (stake matur \(w))\) ).
qed

\section*{fun lbk-price where}
lbk-price \(u d\) init \(r\) matur \(=\left(\sum y \in(\right.\) gener-bool-list matur \()\). lprob-comp \(((1+r-\) d) \(/(u-d)) y *(\) det-discount \(r\) matur \() *\)
(Max \(((\lambda i\). (lgeom-proc u d init \((\) take \(i\) y \())) ‘\{0\).. matur \(\})\) - (lgeom-proc ud 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
( $\sum w \in$ range (pseudo-proj-True matur). (prod (prob-component $((1+r-d)$
$/(u-d)) w)\{0 . .<$ matur $\}) *$
(discounted-value $r(\lambda m$. (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

```

\section*{qed}
value lbk-price 1.20 .8100 .032

\subsection*{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 $(T:: n a t)=\left(\lambda w .\left(\sum i \in\{1 . . T\}\right.\right.$. prices Mkt stk $\left.\left.i w\right) / T\right)$
lemma (in CRR-market) asian-borel:
shows asian-option $T \in$ borel-measurable $(G T)$ unfolding asian-option-def
proof -
have $\left(\lambda w .\left(\sum i \in\{1 . . T\}\right.\right.$. prices Mkt stk $\left.\left.i w\right)\right) \in$ borel-measurable $(G T)$
proof (rule borel-measurable-sum)
fix $i$
assume $i \in\{1 . . T\}$
show prices Mkt stk $i \in$ borel-measurable ( $G T$ )
by (metis $\langle i \in\{1 . . T\}\rangle$ adapt-stoch-proc-def atLeastAtMost-iff increasing-measurable-info
stock-price-borel-measurable)
qed
from this show $\left(\lambda w .\left(\sum i=1 . . T\right.\right.$. prices Mkt stk $\left.i w\right) /$ real $\left.T\right) \in$ borel-measurable
( $G T$ ) by $\operatorname{simp}$
qed
lemma (in CRR-market-viable) asian-option-lgeom:
shows asian-option $T w=\left(\sum i \in\{1 . . T\}\right.$. lgeom-proc ud init (stake $\left.\left.i w\right)\right) / T$
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 $r(\lambda m$. (asian-option $T)) T w)=$
(det-discount $r T) *\left(\sum i \in\{1 . . T\}\right.$. lgeom-proc $u$ d init (take $i($ stake $\left.\left.T w)\right)\right) /$
$T$
proof -
have $\forall i \in\{1 . . T\}$. 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 $\left(\sum i \in\{1 . . T\}\right.$. lgeom-proc ud init $($ stake $\left.i w)\right)=$
( $\sum i \in\{1 . . T\}$. lgeom-proc $u d$ init (take $i($ stake $\left.T w)\right)$ by auto
thus ?thesis
using det-discounted $[$ of $r \lambda$. asian-option $T T w]$ asian-option-lgeom[of $T w]$
by auto
qed
lemma (in CRR-market-viable) asian-effect-compute:

```
shows ( \(\sum w \in\) range (pseudo-proj-True matur). (prod (prob-component pr \(w\) ) \(\{0 . .<\) matur \(\}) *\)
(discounted-value \(r(\lambda\). (asian-option matur \())\) matur \(w))=\)
( \(\sum y \in\) (gener-bool-list matur). lprob-comp pr \(y *\) (det-discount r matur) \(*\)
( \(\sum i \in\{1\).. 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 \(r\) ( \(\lambda\) m. asian-option matur) matur \(w=\)
lprob-comp pr (stake matur \(w) *\) discounted-value \(r\) ( \(\lambda\) m. asian-option matur)
matur \(w\)
using lprob-comp-stake by simp
also have \(\ldots=\) lprob-comp pr (stake matur \(w) *\)
(det-discount \(r\) matur \() *\left(\sum i \in\{1 .\right.\). 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 \(r(\lambda m\). asian-option matur) matur \(w=\)
lprob-comp pr (stake matur \(w\) ) *
(det-discount \(r\) matur \() *\left(\sum i \in\{1 .\right.\). matur \(\}\). lgeom-proc \(u d\) init (take \(i\) (stake matur \(w)\) ))/ matur .
qed
fun asian-price where
asian-price \(u d\) init \(r\) matur \(=\left(\sum y \in(\right.\) gener-bool-list matur \()\). lprob-comp \(((1+r\) \(-d) /(u-d)) y *(\) det-discount \(r\) matur \() *\)
( \(\sum i \in\{1\).. matur \(\}\). lgeom-proc ud 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 ud init r matur)
(asian-option matur) matur
proof -
have fair-price Mkt
( \(\sum w \in\) range (pseudo-proj-True matur). (prod (prob-component \(((1+r-d)\)
\(/(u-d)) w)\{0 . .<\) matur \(\}) *\)
(discounted-value \(r\) ( \(\lambda\) m. (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```

