Pricing in discrete financial models

Mnacho Echenim

September 23, 2023

Contents

1 Generated subalgebras

This section contains definitions and properties related to generated subalgebras.

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

begin

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

lemma *gen-subalgebra-space*:
 shows *space (gen-subalgebra M G) = space M*
⟨*proof*⟩

lemma *gen-subalgebra-sets*:
 assumes $G \subseteq$ *sets M*
 and $A \in G$
 shows $A \in$ *sets (gen-subalgebra M G)*
⟨*proof*⟩

lemma *gen-subalgebra-sig-sets*:
 assumes $G \subseteq$ *Pow (space M)*
 shows *sets (gen-subalgebra M G) = sigma-sets (space M) G* ⟨*proof*⟩

lemma *gen-subalgebra-sigma-sets*:
 assumes $G \subseteq$ *sets M*
 and *sigma-algebra (space M) G*
 shows *sets (gen-subalgebra M G) = G*
⟨*proof*⟩

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*)
⟨*proof*⟩

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

lemma *fct-gen-subalgebra-sets*:
 shows *sets (fct-gen-subalgebra M N X) = sigma-sets (space M)* {*X* −' *B* ∩ *space M* |*B. B* ∈ *sets N*}
⟨*proof*⟩

lemma *fct-gen-subalgebra-space*:

shows *space* (*fct-gen-subalgebra M N X*) = *space M*
⟨*proof*⟩

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*⟩

lemma *fct-gen-subalgebra-sets-mem*:
assumes *B*∈ *sets N*
shows *X* −' *B* ∩ (*space M*) ∈ *sets* (*fct-gen-subalgebra M N X*) ⟨*proof*⟩

lemma *fct-gen-subalgebra-is-subalgebra*:
assumes *X*∈ *measurable M N*
shows *subalgebra M* (*fct-gen-subalgebra M N X*)
⟨*proof*⟩

lemma *fct-gen-subalgebra-fct-measurable*:
assumes *X* ∈ *space M* → *space N*
shows *X*∈ *measurable* (*fct-gen-subalgebra M N X*) *N*
⟨*proof*⟩

lemma *fct-gen-subalgebra-min*:
assumes *subalgebra M P*
and *f*∈ *measurable P N*
shows *subalgebra P* (*fct-gen-subalgebra M N f*)
⟨*proof*⟩

lemma *fct-preimage-sigma-sets*:
assumes *X*∈ *space M* → *space N*
shows *sigma-sets* (*space M*) {*X* −' *B* ∩ *space M* |*B*. *B* ∈ *sets N*} = {*X* −' *B* ∩ *space M* |*B*. *B* ∈ *sets N*} (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *fct-gen-subalgebra-sigma-sets*:
assumes *X*∈ *space M* → *space N*
shows *sets* (*fct-gen-subalgebra M N X*) = {*X* −' *B* ∩ *space M* |*B*. *B* ∈ *sets N*}
⟨*proof*⟩

lemma *fct-gen-subalgebra-info*:
assumes *f*∈ *space M* → *space N*
and *x*∈ *space M*
and *w*∈ *space M*
and *f x* = *f w*
shows ⋀*A*. *A*∈ *sets* (*fct-gen-subalgebra M N f*) ⟹ (*x*∈ *A*) = (*w*∈ *A*)

⟨*proof*⟩

1.1 Independence between a random variable and a subalgebra.

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

lemma (**in** *prob-space*) *indep-set-mono*:
assumes *indep-set A B*
assumes *A'* ⊆ *A*
assumes *B'* ⊆ *B*
shows *indep-set A' B'*
⟨*proof*⟩

lemma (**in** *prob-space*) *subalgebra-indep-var-indicator*:
fixes *X*::*'a*⇒*real*
assumes *subalgebra-indep-var X N*
and *X* ∈ *borel-measurable M*
and *A* ∈ *sets N*
shows *indep-var borel X borel* (*indicator A*)
⟨*proof*⟩

lemma *fct-gen-subalgebra-cong*:
assumes *space M* = *space P*
and *sets N* = *sets Q*
shows *fct-gen-subalgebra M N X* = *fct-gen-subalgebra P Q X*
⟨*proof*⟩

end

2 Filtrations

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

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

2.1 Basic definitions

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

definition *filtration* :: *'a measure* ⇒ (*'i*::*linorder-bot* ⇒ *'a measure*) ⇒ *bool* **where**
  filtration M F ⟷
    (∀ *t*. *subalgebra M* (*F t*)) ∧
    (∀ *s t*. *s* ≤ *t* ⟶ *subalgebra* (*F t*) (*F s*))

lemma *filtrationI*:
  assumes ∀ *t*. *subalgebra M* (*F t*)
  and ∀ *s t*. *s* ≤ *t* ⟶ *subalgebra* (*F t*) (*F s*)
shows *filtration M F* ⟨*proof*⟩

lemma *filtrationE1*:
  assumes *filtration M F*
  shows *subalgebra M* (*F t*) ⟨*proof*⟩

lemma *filtrationE2*:
  assumes *filtration M F*
  shows *s*≤ *t* ⟹ *subalgebra* (*F t*) (*F s*) ⟨*proof*⟩

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

lemma (**in** *filtrated-prob-space*) *filtration-space*:
  assumes *s* ≤ *t*
  shows *space* (*F s*) = *space* (*F t*) ⟨*proof*⟩

lemma (**in** *filtrated-prob-space*) *filtration-measurable*:
  assumes *f*∈ *measurable* (*F t*) *N*
shows *f*∈ *measurable M N* ⟨*proof*⟩

lemma (**in** *filtrated-prob-space*) *increasing-measurable-info*:
  assumes *f*∈ *measurable* (*F s*) *N*
  and *s* ≤ *t*
  shows *f*∈ *measurable* (*F t*) *N*
⟨*proof*⟩

definition *disc-filtr* :: *'a measure* ⇒ (*nat* ⇒ *'a measure*) ⇒ *bool* **where**
  disc-filtr M F ⟷

$(\forall\, n.\ subalgebra\ M\ (F\ n))\ \wedge$
$(\forall\ n\ m.\ n \leq m \longrightarrow subalgebra\ (F\ m)\ (F\ n))$

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

lemma (**in** *disc-filtr-prob-space*) *subalgebra-filtration*:
assumes *subalgebra N M*
and *filtration M F*
shows *filtration N F*
⟨*proof*⟩

sublocale *disc-filtr-prob-space* ⊆ *filtrated-prob-space*
⟨*proof*⟩

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**
$stoch\text{-}procs\ M\ N = \{X.\ \forall\, t.\ (X\ t) \in measurable\ M\ N\}$

2.2.1 Adapted stochastic processes

definition *adapt-stoch-proc* **where**
$(adapt\text{-}stoch\text{-}proc\ F\ X\ N) \longleftrightarrow (\forall\, t.\ (X\ t) \in measurable\ (F\ t)\ N)$

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

lemma (**in** *filtrated-prob-space*) *adapted-is-dsp*:
assumes *adapt-stoch-proc F X N*
shows $X \in stoch\text{-}procs\ M\ N$
⟨*proof*⟩

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*⟩

lemma (**in** *filtrated-prob-space*) *borel-adapt-stoch-proc-borel-measurable*:
assumes *borel-adapt-stoch-proc F X*
shows $\forall\, n.\ (X\ n) \in borel\text{-}measurable\ M$

⟨*proof*⟩

lemma (**in** *filtrated-prob-space*) *constant-process-borel-adapted*:
shows *borel-adapt-stoch-proc F* (λ *n w. c*)
⟨*proof*⟩

lemma (**in** *filtrated-prob-space*) *borel-adapt-stoch-proc-add*:
fixes *X*::′*b* ⇒ ′*a* ⇒ (′*c*::{*second-countable-topology*, *topological-monoid-add*})
assumes *borel-adapt-stoch-proc F X*
and *borel-adapt-stoch-proc F Y*
shows *borel-adapt-stoch-proc F* (λ*t w. X t w* + *Y t w*) ⟨*proof*⟩

lemma (**in** *filtrated-prob-space*) *borel-adapt-stoch-proc-sum*:
fixes *A*::′*d* ⇒ ′*b* ⇒ ′*a* ⇒ (′*c*::{*second-countable-topology*, *topological-comm-monoid-add*})
assumes ⋀*i. i*∈ *S* ⟹ *borel-adapt-stoch-proc F* (*A i*)
shows *borel-adapt-stoch-proc F* (λ *t w.* (∑ *i*∈ *S. A i t w*)) ⟨*proof*⟩

lemma (**in** *filtrated-prob-space*) *borel-adapt-stoch-proc-times*:
fixes *X*::′*b* ⇒ ′*a* ⇒ (′*c*::{*second-countable-topology*, *real-normed-algebra*})
assumes *borel-adapt-stoch-proc F X*
and *borel-adapt-stoch-proc F Y*
shows *borel-adapt-stoch-proc F* (λ*t w. X t w* ∗ *Y t w*) ⟨*proof*⟩

lemma (**in** *filtrated-prob-space*) *borel-adapt-stoch-proc-prod*:
fixes *A*::′*d* ⇒ ′*b* ⇒ ′*a* ⇒ (′*c*::{*second-countable-topology*, *real-normed-field*})
assumes ⋀*i. i*∈ *S* ⟹ *borel-adapt-stoch-proc F* (*A i*)
shows *borel-adapt-stoch-proc F* (λ *t w.* (∏ *i*∈ *S. A i t w*)) ⟨*proof*⟩

2.2.2 Predictable stochastic processes

definition *predict-stoch-proc* **where**
(*predict-stoch-proc F X N*) ⟷ (*X 0* ∈ *measurable* (*F 0*) *N* ∧ (∀ *n.* (*X* (*Suc n*))
∈ *measurable* (*F n*) *N*))

abbreviation *borel-predict-stoch-proc F X* ≡ *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* ⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *predictable-is-dsp*:
assumes *predict-stoch-proc F X N*
shows *X* ∈ *stoch-procs M N*
⟨*proof*⟩

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* ⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *constant-process-borel-predictable*:
  shows *borel-predict-stoch-proc F* $(\lambda\ n\ w.\ c)$
⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *borel-predict-stoch-proc-add*:
  fixes X::*nat* ⇒ *'a* ⇒ (*'c*::{*second-countable-topology*, *topological-monoid-add*})
  assumes *borel-predict-stoch-proc F X*
  and *borel-predict-stoch-proc F Y*
shows *borel-predict-stoch-proc F* $(\lambda t\ w.\ X\ t\ w + Y\ t\ w)$ ⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *borel-predict-stoch-proc-sum*:
  fixes A::*'d* ⇒ *nat* ⇒ *'a* ⇒ (*'c*::{*second-countable-topology*, *topological-comm-monoid-add*})
  assumes $\bigwedge i.\ i \in S \Longrightarrow$ *borel-predict-stoch-proc F* $(A\ i)$
shows *borel-predict-stoch-proc F* $(\lambda\ t\ w.\ (\sum\ i \in S.\ A\ i\ t\ w))$ ⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *borel-predict-stoch-proc-times*:
  fixes X::*nat* ⇒ *'a* ⇒ (*'c*::{*second-countable-topology*, *real-normed-algebra*})
  assumes *borel-predict-stoch-proc F X*
  and *borel-predict-stoch-proc F Y*
shows *borel-predict-stoch-proc F* $(\lambda t\ w.\ X\ t\ w * Y\ t\ w)$ ⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *borel-predict-stoch-proc-prod*:
  fixes A::*'d* ⇒ *nat* ⇒ *'a* ⇒ (*'c*::{*second-countable-topology*, *real-normed-field*})
  assumes $\bigwedge i.\ i \in S \Longrightarrow$ *borel-predict-stoch-proc F* $(A\ i)$
shows *borel-predict-stoch-proc F* $(\lambda\ t\ w.\ (\prod\ i \in S.\ A\ i\ t\ w))$ ⟨*proof*⟩

definition (**in** *prob-space*) *constant-image* **where**
  constant-image f = (*if* ∃ *c*::*'b*::{*t2-space*}. ∀ *x*∈ *space M*. *f x* = *c then*
    SOME c. ∀ *x* ∈ *space M*. *f x* = *c else undefined*)

lemma (**in** *prob-space*) *constant-imageI*:
  assumes ∃ *c*::*'b*::{*t2-space*}. ∀ *x*∈ *space M*. *f x* = *c*
  shows ∀ *x*∈ *space M*. *f x* = (*constant-image f*)
⟨*proof*⟩

lemma (**in** *prob-space*) *constant-image-pos*:

assumes $\forall x \in$ *space M*. $(0::real) < f\ x$
and $\exists c::real.\ \forall x \in$ *space M*. $f\ x = c$
shows $0 < (constant\text{-}image\ f)$
⟨*proof*⟩

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

lemma *open-exceptI*:
assumes $(x::'b::\{t1\text{-}space\}) \neq y$
shows $open\ (open\text{-}except\ x\ y)$ **and** $x \in open\text{-}except\ x\ y$ **and** $y \notin open\text{-}except\ x\ y$
⟨*proof*⟩

lemma *open-except-set*:
assumes *finite A*
and $(x::'b::\{t1\text{-}space\}) \notin A$
shows $\exists U.\ open\ U \wedge x \in U \wedge U \cap A = \{\}$
⟨*proof*⟩

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

lemma *open-exclude-setI*:
assumes $\exists U.\ open\ U \wedge U \cap A = \{x\}$
shows $open\ (open\text{-}exclude\text{-}set\ x\ A)$ **and** $(open\text{-}exclude\text{-}set\ x\ A) \cap A = \{x\}$
⟨*proof*⟩

lemma *open-exclude-finite*:
assumes *finite A*
and $(x::'b::\{t1\text{-}space\}) \in A$
shows *open-set*: $open\ (open\text{-}exclude\text{-}set\ x\ A)$ **and** *inter-x*: $(open\text{-}exclude\text{-}set\ x\ A) \cap A = \{x\}$
⟨*proof*⟩

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

lemma *triv-measurable-cst*:
fixes $f::'a \Rightarrow 'b::\{t2\text{-}space\}$
assumes $space\ N = space\ M$
and $space\ M \neq \{\}$
and $sets\ N = \{\{\},\ space\ M\}$
and $f \in measurable\ N\ borel$

shows ∃ *c*::′*b*. ∀ *x*∈ *space N*. *f x* = *c*
⟨*proof*⟩

locale *trivial-init-filtrated-prob-space* = *prob-space* +
  fixes *F*
  assumes *info-filtration*: *init-triv-filt M F*

sublocale *trivial-init-filtrated-prob-space* ⊆ *filtrated-prob-space*
  ⟨*proof*⟩

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

sublocale *triv-init-disc-filtr-prob-space* ⊆ *trivial-init-filtrated-prob-space*
⟨*proof*⟩

sublocale *triv-init-disc-filtr-prob-space* ⊆ *disc-filtr-prob-space*
⟨*proof*⟩

lemma (**in** *triv-init-disc-filtr-prob-space*) *adapted-init*:
  assumes *borel-adapt-stoch-proc F x*
  shows ∃ *c*. ∀ *w* ∈ *space M*. ((*x 0 w*)::*real*) = *c*
⟨*proof*⟩

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 ⟷ *sets M* = *sets N* ∧ *filtration M F* ∧ (∀ *t A*. *A* ∈ *sets* (*F t*) ⟶ (*emeasure M A* = *0*) ⟷ (*emeasure N A* = *0*))

lemma *filt-equiv-space*:
  assumes *filt-equiv F M N*
  shows *space M* = *space N* ⟨*proof*⟩

lemma *filt-equiv-sets*:
  assumes *filt-equiv F M N*
  shows *sets M* = *sets N* ⟨*proof*⟩

lemma *filt-equiv-filtration*:
  assumes *filt-equiv F M N*
  shows *filtration N F* ⟨*proof*⟩

lemma (**in** *filtrated-prob-space*) *AE-borel-eq*:
fixes *f*::′*a*⇒*real*
assumes *f*∈ *borel-measurable* (*F t*)
and *g*∈ *borel-measurable* (*F t*)
and *AE w in M. f w* = *g w*
shows {*w*∈ *space M. f w* ≠ *g w*} ∈ *sets* (*F t*) ∧ *emeasure M* {*w*∈ *space M. f w* ≠ *g w*} = *0*
⟨*proof*⟩

lemma (**in** *prob-space*) *filt-equiv-borel-AE-eq*:
  fixes *f*::′*a*⇒ *real*
  assumes *filt-equiv F M N*
and *f*∈ *borel-measurable* (*F t*)
and *g*∈ *borel-measurable* (*F t*)
and *AE w in M. f w* = *g w*
shows *AE w in N. f w* = *g w*
⟨*proof*⟩

lemma *filt-equiv-prob-space-subalgebra*:
  assumes *prob-space N*
  and *filt-equiv F M N*
  and *sigma-finite-subalgebra M G*
shows *sigma-finite-subalgebra N G* ⟨*proof*⟩

lemma *filt-equiv-measurable*:
  assumes *filt-equiv F M N*
  and *f*∈ *measurable M P*
shows *f*∈ *measurable N P* ⟨*proof*⟩

lemma *filt-equiv-imp-subalgebra*:
  assumes *filt-equiv F M N*
shows *subalgebra N M* ⟨*proof*⟩

end

3 Martingales

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

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

lemma *martingaleAE*:
assumes *martingale M F X*
and $t \leq s$
shows *AE w in M. real-cond-exp M* (*F t*) (*X s*) *w* = (*X t*) *w* ⟨*proof*⟩

lemma *martingale-add*:
assumes *martingale M F X*
and *martingale M F Y*
and $\forall$ *m. sigma-finite-subalgebra M* (*F m*)
shows *martingale M F* (λ*n w. X n w* + *Y n w*) ⟨*proof*⟩

lemma *disc-martingale-charact*:
assumes ($\forall$ *n. integrable M* (*X n*))
and *filtration M F*
and $\forall$ *m. sigma-finite-subalgebra M* (*F m*)
and $\forall$ *m. X m* $\in$ *borel-measurable* (*F m*)
and ($\forall$ *n. AE w in M. real-cond-exp M* (*F n*) (*X* (*Suc n*)) *w* = (*X n*) *w*)
shows *martingale M F X* ⟨*proof*⟩

lemma (**in** *finite-measure*) *constant-martingale*:
assumes $\forall$ *t. sigma-finite-subalgebra M* (*F t*)
and *filtration M F*
shows *martingale M F* (λ*n w. c*) ⟨*proof*⟩

end

4 Discrete Conditional Expectation

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

begin

4.1 Preliminary measurability results

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

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

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

lemma *singl-meas-if*:
assumes *f* ∈ *space M* → *space N*
and ∀ *r*∈ *range f*∩ *space N*. ∃ *A*∈ *sets N*. *range f*∩ *A* = {*r*}
shows *point-measurable* (*fct-gen-subalgebra M N f*) (*space N*) *f* ⟨*proof*⟩

lemma *meas-single-meas*:
assumes *f*∈ *measurable M N*
and ∀ *r*∈ *range f*∩ *space N*. ∃ *A*∈ *sets N*. *range f*∩ *A* = {*r*}
shows *point-measurable M* (*space N*) *f*
⟨*proof*⟩

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

lemma *count-pre-disj*:
fixes *i*::*nat*
assumes *countable B*
and *i* ≠ *j*
shows (*countable-preimages B Y*) *i* ∩ (*countable-preimages B Y*) *j* = {}
⟨*proof*⟩

lemma *count-pre-surj*:
assumes *countable B*
and *w* ∈ *Y* −‘*B*
shows ∃ *i*. *w* ∈ (*countable-preimages B Y*) *i*
⟨*proof*⟩

lemma *count-pre-img*:
assumes *x* ∈ (*countable-preimages B Y*) *n*
shows *Y x* = (*from-nat-into B*) *n*
⟨*proof*⟩

lemma *count-pre-union-img*:
  assumes *countable B*
  shows $Y -`B = (\bigcup\ i.\ (countable\text{-}preimages\ B\ Y)\ i)$
⟨*proof*⟩

lemma *count-pre-meas*:
  assumes *point-measurable M (space N) Y*
  and $B \subseteq$ *space N*
  and *countable B*
  shows $\forall\, i.\ (countable\text{-}preimages\ B\ Y)\ i \cap space\ M \in sets\ M$
⟨*proof*⟩

lemma *disct-fct-point-measurable*:
assumes *disc-fct f*
and *point-measurable M (space N) f*
shows $f \in$ *measurable M N* ⟨*proof*⟩

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*⟩

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**
  $img\text{-}dce\ M\ Y\ X = (\lambda\ y.\ if\ measure\ M\ ((Y -`\ \{y\}) \cap space\ M) = 0\ then\ 0\ else$
    $((integral^L\ M\ (\lambda w.\ ((X\ w) * (indicator\ ((Y -`\{y\}) \cap space\ M)\ w))))/(measure$
$M\ ((Y -`\ \{y\}) \cap space\ M))))$

definition *expl-cond-expect* **where**
  $expl\text{-}cond\text{-}expect\ M\ Y\ X = (img\text{-}dce\ M\ Y\ X) \circ Y$

lemma *nn-expl-cond-expect-pos*:
  assumes $\forall\, w \in space\ M.\ 0 \leq X\ w$
shows $\forall\ w \in space\ M.\ 0 \leq (expl\text{-}cond\text{-}expect\ M\ Y\ X)\ w$
⟨*proof*⟩

lemma *expl-cond-expect-const*:

assumes *Y w* = *Y y*
shows *expl-cond-expect M Y X w* = *expl-cond-expect M Y X y*
⟨*proof*⟩

lemma *expl-cond-exp-cong*:
assumes ∀ *w*∈*space M*. *X w* = *Z w*
shows ∀ *w*∈ *space M*. *expl-cond-expect M Y X w* = *expl-cond-expect M Y Z w*
⟨*proof*⟩

lemma *expl-cond-exp-add*:
assumes *integrable M X*
and *integrable M Z*
shows ∀ *w*∈ *space M*. *expl-cond-expect M Y* (λ*x*. *X x* + *Z x*) *w* = *expl-cond-expect M Y X w* + *expl-cond-expect M Y Z w*
⟨*proof*⟩

lemma *expl-cond-exp-diff*:
assumes *integrable M X*
and *integrable M Z*
shows ∀ *w*∈ *space M*. *expl-cond-expect M Y* (λ*x*. *X x* − *Z x*) *w* = *expl-cond-expect M Y X w* − *expl-cond-expect M Y Z w*
⟨*proof*⟩

lemma *expl-cond-expect-prop-sets*:
assumes *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
and *D* = {*w*∈ *space M*. *Y w* ∈ *space N* ∧ (*P* (*expl-cond-expect M Y X w*))}
shows *D*∈ *sets M*
⟨*proof*⟩

lemma *expl-cond-expect-prop-sets2*:
assumes *disc-fct Y*
and *point-measurable* (*fct-gen-subalgebra M N Y*) (*space N*) *Y*
and *D* = {*w*∈ *space M*. *Y w* ∈ *space N* ∧ (*P* (*expl-cond-expect M Y X w*))}
shows *D*∈ *sets* (*fct-gen-subalgebra M N Y*)
⟨*proof*⟩

lemma *expl-cond-expect-disc-fct*:
assumes *disc-fct Y*
shows *disc-fct* (*expl-cond-expect M Y X*)
⟨*proof*⟩

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*⟩

lemma *expl-cond-expect-borel-measurable*:
  assumes *disc-fct Y*
  and *point-measurable M* (*space N*) *Y*
shows (*expl-cond-expect M Y X*) ∈ *borel-measurable M* ⟨*proof*⟩

lemma *expl-cond-exp-borel*:
  assumes *Y* ∈ *space M* → *space N*
  and *disc-fct Y*
  and ∀ *r*∈ *range Y*∩ *space N*. ∃ *A*∈ *sets N*. *range Y*∩ *A* = {*r*}
  shows (*expl-cond-expect M Y X*) ∈ *borel-measurable* (*fct-gen-subalgebra M N Y*)
⟨*proof*⟩

lemma *expl-cond-expect-indic-borel-measurable*:
  assumes *disc-fct Y*
  and *point-measurable M* (*space N*) *Y*
  and *B*⊆ *space N*
  and *countable B*
  shows (λ*w*. *expl-cond-expect M Y X w* ∗ *indicator* (*countable-preimages B Y n* ∩ *space M*) *w*)∈ *borel-measurable M*
⟨*proof*⟩

lemma (**in** *finite-measure*) *dce-prod*:
  assumes *point-measurable M* (*space N*) *Y*
   and *integrable M X*
   and ∀ *w*∈ *space M*. *0* ≤ *X w*
shows ∀ *w*. (*Y w*) ∈ *space N* ⟶ (*expl-cond-expect M Y X*) *w* ∗ *measure M* ((*Y* −‘ {*Y w*})∩ *space M*) = *integral*^{L} *M* (λ*y*. (*X y*) ∗ (*indicator* ((*Y* −‘{*Y w*})∩ *space M*) *y*))
⟨*proof*⟩

lemma *expl-cond-expect-const-exp*:
shows $integral^L$ *M* (λ*y*. *expl-cond-expect M Y X w* ∗ (*indicator* (*Y* −' {*Y w*} ∩ *space M*)) *y*) =
$integral^L$ *M* (λ*y*. *expl-cond-expect M Y X y* ∗ (*indicator* (*Y* −' {*Y w*} ∩ *space M*)) *y*)
⟨*proof*⟩

lemma *nn-expl-cond-expect-const-exp*:
assumes ∀ *w*∈ *space M*. *0* ≤ *X w*
shows $integral^N$ *M* (λ*y*. *expl-cond-expect M Y X w* ∗ (*indicator* (*Y* −' {*Y w*} ∩ *space M*)) *y*) =
$integral^N$ *M* (λ*y*. *expl-cond-expect M Y X y* ∗ (*indicator* (*Y* −' {*Y w*} ∩ *space M*)) *y*)
⟨*proof*⟩

lemma (**in** *finite-measure*) *nn-expl-cond-bounded*:
assumes ∀ *w*∈ *space M*. *0* ≤ *X w*
and *integrable M X*
and *point-measurable M* (*space N*) *Y*
and *w*∈ *space M*
and *Y w*∈ *space N*
shows $integral^N$ *M* (λ*y*. *expl-cond-expect M Y X y* ∗ (*indicator* (*Y* −' {*Y w*} ∩ *space M*)) *y*) < ∞
⟨*proof*⟩

lemma (**in** *finite-measure*) *count-prod*:
fixes *Y*::'*a*⇒'*b*
assumes *B*⊆ *space N*
and *point-measurable M* (*space N*) *Y*
and *integrable M X*
and ∀ *w* ∈ *space M*. *0* ≤ *X w*
shows ∀ *i*. $integral^L$ *M* (λ*y*. (*X y*) ∗ (*indicator* (*countable-preimages B Y i* ∩ *space M*)) *y*) =
$integral^L$ *M* (λ*y*. (*expl-cond-expect M Y X y*) ∗ (*indicator* (*countable-preimages B Y i* ∩ *space M*)) *y*)
⟨*proof*⟩

lemma (**in** *finite-measure*) *count-pre-integrable*:
assumes *point-measurable M* (*space N*) *Y*
and *disc-fct Y*
and *B*⊆ *space N*
and *countable B*
and *integrable M X*

and ∀ *w* ∈ *space M*. *0* ≤ *X w*
shows *integrable M* (*λw*. *expl-cond-expect M Y X w* ∗ *indicator* (*countable-preimages B Y n* ∩ *space M*) *w*)
⟨*proof*⟩

lemma (**in** *finite-measure*) *nn-cond-expl-is-cond-exp-tmp*:
assumes ∀ *w*∈ *space M*. *0* ≤ *X w*
and *integrable M X*
and *disc-fct Y*
and *point-measurable M* (*space M′*) *Y*
shows ∀ *A* ∈ *sets M′*. *integrable M* (*λw*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator* ((*Y* −‘*A*)∩ (*space M*)) *w*)) ∧
integral^L *M* (*λw*. (*X w*) ∗ (*indicator* ((*Y* −‘*A*)∩ (*space M*)) *w*)) =
integral^L *M* (*λw*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator* ((*Y* −‘*A*) ∩ (*space M*))) *w*)
⟨*proof*⟩

lemma (**in** *finite-measure*) *nn-expl-cond-exp-integrable*:
assumes ∀ *w*∈ *space M*. *0* ≤ *X w*
and *integrable M X*
and *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
shows *integrable M* (*expl-cond-expect M Y X*)
⟨*proof*⟩

lemma (**in** *finite-measure*) *nn-cond-expl-is-cond-exp*:
assumes ∀ *w*∈ *space M*. *0* ≤ *X w*
and *integrable M X*
and *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
shows ∀ *A* ∈ *sets N*. *integral*^{L} *M* (*λw*. (*X w*) ∗ (*indicator* ((*Y* −‘*A*)∩ (*space M*)) *w*)) =
integral^L *M* (*λw*. ((*expl-cond-expect M Y X*) *w*) ∗ (*indicator* ((*Y* −‘*A*) ∩ (*space M*))) *w*)
⟨*proof*⟩

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*⟩

lemma (**in** *finite-measure*) *is-cond-exp*:
assumes *integrable M X*
and *disc-fct Y*
and *point-measurable M* (*space N*) *Y*
shows $\forall$ $A \in$ *sets N*. *integral*^{L} *M* (λw. (*X w*) $*$ (*indicator* ((*Y* $-$'*A*)$\cap$ (*space M*)) *w*)) =
integral^L *M* (λw. ((*expl-cond-expect M Y X*) *w*) $*$ (*indicator* ((*Y* $-$'*A*) $\cap$ (*space M*))) *w*)
⟨*proof*⟩

lemma (**in** *finite-measure*) *charact-cond-exp*:
assumes *disc-fct Y*
and *integrable M X*
and *point-measurable M* (*space N*) *Y*
and $Y \in$ *space M* $\to$ *space N*
and $\forall r \in$ *range Y*$\cap$ *space N*. $\exists A \in$ *sets N*. *range Y*$\cap$ $A = \{r\}$
shows *AE w in M*. *real-cond-exp M* (*fct-gen-subalgebra M N Y*) *X w* = *expl-cond-expect M Y X w*
⟨*proof*⟩

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 N Y*) *X w* = *expl-cond-expect M Y X w*
⟨*proof*⟩

end

5 Infinite coin toss space

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

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

begin

5.1 Preliminary results

lemma *decompose-init-prod*:

fixes $n::nat$
shows $(\prod\ i\in \{0..n\}.\ f\ i) = f\ 0 * (\prod\ i\in \{1..n\}.\ f\ i)$
$\langle proof \rangle$

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

lemma *enn2real-sum*: **shows** $finite\ A \Longrightarrow (\bigwedge a.\ a\in A\Longrightarrow f\ a < top) \Longrightarrow enn2real\ (sum\ f\ A) = (\sum\ a\in A.\ enn2real\ (f\ a))$
$\langle proof \rangle$

lemma *ennreal-sum*: **shows** $finite\ A \Longrightarrow (\bigwedge a.\ 0 \leq f\ a) \Longrightarrow (\sum a\in A.\ ennreal\ (f\ a)) = ennreal\ (\sum a\in A.\ f\ a)$
$\langle proof \rangle$

lemma *stake-snth*:
assumes $stake\ n\ w = stake\ n\ x$
shows $Suc\ i \leq n \Longrightarrow snth\ w\ i = snth\ x\ i$
$\langle proof \rangle$

lemma *stake-snth-charact*:
assumes $stake\ n\ w = stake\ n\ x$
shows $\forall\, i < n.\ snth\ w\ i = snth\ x\ i$
$\langle proof \rangle$

lemma *stake-snth'*:
shows $(\bigwedge i.\ Suc\ i \leq n \Longrightarrow snth\ w\ i = snth\ x\ i) \Longrightarrow stake\ n\ w = stake\ n\ x$
$\langle proof \rangle$

lemma *stake-inter-snth*:
fixes x
assumes $Suc\ 0 \leq n$
shows $\{w\in space\ M.\ (stake\ n\ w = stake\ n\ x)\} = (\bigcap\ i \in \{0..\ n-1\}.\ \{w\in space\ M.\ (snth\ w\ i = snth\ x\ i)\})$
$\langle proof \rangle$

lemma *streams-stake-set*:
shows $pw \in streams\ A \Longrightarrow set\ (stake\ n\ pw) \subseteq A$
$\langle proof \rangle$

lemma *stake-finite-universe-induct*:
assumes $finite\ A$

and $A \neq \{\}$
shows $(stake\ (Suc\ n)\ `(streams\ A)) = \{s\#w|\ s\ w.\ s\in A \wedge w\in (stake\ n\ `(streams\ A))\}$ (**is** $?L = ?R$)
$\langle proof \rangle$

lemma *stake-finite-universe-finite*:
assumes *finite* A
and $A \neq \{\}$
shows *finite* $(stake\ n\ `(streams\ A))$
$\langle proof \rangle$

lemma *diff-streams-only-if*:
assumes $w \neq x$
shows $\exists\, n.\ snth\ w\ n \neq snth\ x\ n$
$\langle proof \rangle$

lemma *diff-streams-if*:
assumes $\exists\, n.\ snth\ w\ n \neq snth\ x\ n$
shows $w\neq x$
$\langle proof \rangle$

lemma *sigma-set-union-count*:
assumes $\forall\ y\in A.\ B\ y \in sigma\text{-}sets\ X\ Y$
and *countable* A
shows $(\bigcup\ y\in A.\ B\ y) \in sigma\text{-}sets\ X\ Y$
$\langle proof \rangle$

lemma *sigma-set-inter-init*:
assumes $\bigwedge i.\ i\leq(n::nat) \Longrightarrow A\ i \in sigma\text{-}sets\ sp\ B$
and $B \subseteq Pow\ sp$
shows $(\bigcap\ i\in \{m.\ m\leq n\}.\ A\ i) \in sigma\text{-}sets\ sp\ B$
$\langle proof \rangle$

lemma *adapt-sigma-sets*:
assumes $\bigwedge i.\ i \leq n\Longrightarrow (X\ i) \in measurable\ M\ N$
shows $sigma\text{-}algebra\ (space\ M)\ (sigma\text{-}sets\ (space\ M)\ (\bigcup i\in\{m.\ m \leq n\}.\ \{X\ i\ -`\ A \cap space\ M\ |A.\ A \in sets\ N\}))$
$\langle proof \rangle$

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 $bernoulli\text{-}stream::real \Rightarrow (bool\ stream)\ measure$ **where**

bernoulli-stream p = stream-space (measure-pmf (bernoulli-pmf p))

lemma *bernoulli-stream-space*:
assumes $N = bernoulli\text{-}stream\ p$
shows $space\ N = streams\ UNIV{::}bool$
⟨*proof*⟩

lemma *bernoulli-stream-preimage*:
assumes $N = bernoulli\text{-}stream\ p$
shows $f\ -`\ A \cap (space\ N) = f-`A$
⟨*proof*⟩

lemma *bernoulli-stream-component-probability*:
assumes $N = bernoulli\text{-}stream\ p$ **and** $0 \leq p$ **and** $p \leq 1$
shows $\forall\ n.\ emeasure\ N\ \{w \in space\ N.\ (snth\ w\ n)\} = p$
⟨*proof*⟩

lemma *bernoulli-stream-component-probability-compl*:
assumes $N = bernoulli\text{-}stream\ p$ **and** $0 \leq p$ **and** $p \leq 1$
shows $\forall\ n.\ emeasure\ N\ \{w \in space\ N.\ \neg(snth\ w\ n)\} = 1 - p$
⟨*proof*⟩

lemma *bernoulli-stream-sets*:
assumes $0 < q$
and $q < 1$
and $0 < p$
and $p < 1$
shows $sets\ (bernoulli\text{-}stream\ p) = sets\ (bernoulli\text{-}stream\ q)$ ⟨*proof*⟩

locale *infinite-coin-toss-space* =
fixes $p{::}real$ **and** $M{::}bool\ stream\ measure$
assumes *p-gt-0*: $0 \leq p$
and *p-lt-1*: $p \leq 1$
and *bernoulli*: $M = bernoulli\text{-}stream\ p$

sublocale *infinite-coin-toss-space* ⊆ *prob-space*
⟨*proof*⟩

5.3 Natural filtration on the infinite coin toss space

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

5.3.1 The projection function

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

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

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

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-neq-True*:
shows *pseudo-proj-False n w* ≠ *pseudo-proj-True n w*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-measurable*:
shows *pseudo-proj-False n* ∈ *measurable* (*bernoulli-stream p*) (*bernoulli-stream p*)
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-stake*:
shows *stake n* (*pseudo-proj-True n w*) = *stake n w* ⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-stake*:
shows *stake n* (*pseudo-proj-False n w*) = *stake n w* ⟨*proof*⟩

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* ⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-prefix*:
assumes *n* ≤ *m*
and *pseudo-proj-True m x* = *pseudo-proj-True m y*
shows *pseudo-proj-True n x* = *pseudo-proj-True n y*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-measurable*:
shows *pseudo-proj-True n* ∈ *measurable* (*bernoulli-stream p*) (*bernoulli-stream p*)
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-finite-image*:
shows *finite* (*range* (*pseudo-proj-True n*))
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-finite-image*:
shows *finite* (*range* (*pseudo-proj-False n*))
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-shift*:
shows *length l* = *n* ⟹ *pseudo-proj-True n* (*shift l* (*sconst True*)) = *shift l* (*sconst True*)
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-suc-img*:
shows *pseudo-proj-True* (*Suc n*) *w* ∈ {*pseudo-proj-True n w*, *pseudo-proj-False n w*}
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *measurable-snth-count-space*:
shows (λ*w*. *snth w n*) ∈ *measurable* (*bernoulli-stream p*) (*count-space* (*UNIV*::*bool set*))
⟨*proof*⟩

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* ⟨*proof*⟩

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*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-snth'*:
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*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-preimage*:
assumes *w* = *pseudo-proj-True n w*
shows (*pseudo-proj-True n*) −‘ {*w*} = ($\bigcap$ *i*∈ {*m*. *Suc m* ≤ *n*}. (λ*w*. *snth w i*) −‘ {*snth w i*})
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-preimage*:
assumes *w* = *pseudo-proj-False n w*
shows (*pseudo-proj-False n*) −‘ {*w*} = ($\bigcap$ *i*∈ {*m*. *Suc m* ≤ *n*}. (λ*w*. *snth w i*) −‘ {*snth w i*})
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-preimage-stake*:
assumes *w* = *pseudo-proj-True n w*
shows (*pseudo-proj-True n*) −‘ {*w*} = {*x*. *stake n x* = *stake n w*}
⟨*proof*⟩

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*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-preimage-stake-space*:
assumes *w* = *pseudo-proj-True n w*
shows (*pseudo-proj-True n*) −‘ {*w*} ∩ *space M* = {*x*∈ *space M*. *stake n x* = *stake n w*}
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-singleton*:
assumes *w* = *pseudo-proj-True n w*
shows (*pseudo-proj-True n*) −‘{*w*} ∩ (*space* (*bernoulli-stream p*)) ∈ *sets* (*bernoulli-stream*

p)
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-False-singleton*:
assumes $w = pseudo\text{-}proj\text{-}False\ n\ w$
shows $(pseudo\text{-}proj\text{-}False\ n) -` \{w\} \cap (space\ (bernoulli\text{-}stream\ p)) \in sets\ (bernoulli\text{-}stream\ p)$
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-inverse-induct*:
assumes $w \in range\ (pseudo\text{-}proj\text{-}True\ n)$
shows $(pseudo\text{-}proj\text{-}True\ n) -` \{w\} =$
$(pseudo\text{-}proj\text{-}True\ (Suc\ n)) -` \{w\} \cup (pseudo\text{-}proj\text{-}True\ (Suc\ n)) -`\{pseudo\text{-}proj\text{-}False\ n\ w\}$
⟨*proof*⟩

5.3.2 Natural filtration locale

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

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

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

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-space*:
shows *space* (*nat-filtration n*) = *UNIV*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-sets*:
shows *sets* (*nat-filtration n*) =
sigma-sets (*space* (*bernoulli-stream p*))
$\{pseudo\text{-}proj\text{-}True\ n\ -`\ B \cap space\ M\ |B.\ B \in sets\ (bernoulli\text{-}stream\ p)\}$
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-singleton*:
assumes *pseudo-proj-True n w* = *w*
shows $pseudo\text{-}proj\text{-}True\ n -`\{w\} \in sets\ (nat\text{-}filtration\ n)$
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-pseudo-proj-True-measurable*:
shows *pseudo-proj-True n* ∈ *measurable* (*nat-filtration n*) *M* ⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-comp-measurable*:
assumes *f* ∈ *measurable M N*
and *f* ∘ *pseudo-proj-True n* = *f*
shows *f* ∈ *measurable* (*nat-filtration n*) *N*
⟨*proof*⟩

definition (**in** *infinite-coin-toss-space*) *set-discriminating* **where**
set-discriminating n f N ≡ (∀ *w*. *f w* ≠ *f* (*pseudo-proj-True n w*) ⟶
(∃ *A*∈*sets N*. (*f w* ∈ *A*) = (*f* (*pseudo-proj-True n w*) ∉ *A*)))

lemma (**in** *infinite-coin-toss-space*) *set-discriminating-if*:
fixes *f*::*bool stream* ⇒ ′*b*::{*t0-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows *set-discriminating n f borel* ⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-not-borel-info*:
assumes *f*∈ *measurable* (*nat-filtration n*) *N*
and *set-discriminating n f N*
shows *f*∘ *pseudo-proj-True n* = *f*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-info*:
fixes *f*::*bool stream* ⇒ ′*b*::{*t0-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows *f*∘ *pseudo-proj-True n* = *f*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-not-borel-info′*:
assumes *f*∈ *measurable* (*nat-filtration n*) *N*
and *set-discriminating n f N*
shows *f*∘ *pseudo-proj-False n* = *f*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-info′*:

fixes *f*::*bool stream* ⇒ ′*b*::{*t0-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows *f*∘ *pseudo-proj-False n* = *f*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-borel-measurable-characterization*:
fixes *f*::*bool stream* ⇒ ′*b*::{*t0-space*}
assumes *f*∈ *borel-measurable M*
shows *f*∈ *borel-measurable* (*nat-filtration n*) ⟷ *f*∘ *pseudo-proj-True n* = *f*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-borel-measurable-init*:
fixes *f*::*bool stream* ⇒ ′*b*::{*t0-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration 0*)
shows *f* = (λ*w*. *f* (*sconst True*))
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-Suc-sets*:
shows *sets* (*nat-filtration n*) ⊆ *sets* (*nat-filtration* (*Suc n*))
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-subalgebra*:
shows *subalgebra M* (*nat-filtration n*) ⟨*proof*⟩

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

lemma (**in** *infinite-coin-toss-space*) *nat-info-filtration*:
shows *init-triv-filt M nat-filtration* ⟨*proof*⟩

sublocale *infinite-cts-filtration* ⊆ *triv-init-disc-filtr-prob-space*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-vimage-finite*:

fixes *f*::*bool stream* ⇒ ′*b*::{*t2-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows *finite* (*f*‘(*space M*)) ⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-borel-measurable-simple*:
fixes *f*::*bool stream* ⇒ ′*b*::{*t2-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows *simple-function M f*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-singleton-range-set*:
fixes *f*::*bool stream* ⇒ ′*b*::{*t2-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows ∃ *A*∈ *sets borel*. *range f* ∩ *A* = {*f x*}
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-borel-measurable-singleton*:
fixes *f*::*bool stream* ⇒ ′*b*::{*t2-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows *f* −‘{*f x*} ∈ *sets* (*nat-filtration n*)
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *borel-adapt-nat-filtration-info*:
fixes *X*::*nat* ⇒ *bool stream* ⇒ ′*b*::{*t0-space*}
assumes *borel-adapt-stoch-proc F X*
and *m* ≤ *n*
shows *X m* (*pseudo-proj-True n w*) = *X m w*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-borel-measurable-integrable*:
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
shows *integrable M f*
⟨*proof*⟩

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

lemma (**in** *infinite-coin-toss-space*) *spickI*:
shows *stake n* (*spick w n v*) = *stake n w* ∧ *snth* (*spick w n v*) *n* = *v*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *spick-eq-pseudo-proj-True*:
shows *spick w n True* = *pseudo-proj-True n w* ⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *spick-eq-pseudo-proj-False*:
shows *spick w n False* = *pseudo-proj-False n w* ⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *spick-pseudo-proj*:
shows *spick* (*pseudo-proj-True* (*Suc n*) *w*) *n v* = *spick w n v*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *spick-pseudo-proj-gen*:
shows *m* < *n* ⟹ *spick* (*pseudo-proj-True n w*) *m v* = *spick w m v*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *spick-nat-filtration-measurable*:
shows (λ*w*. *spick w n v*) ∈ *measurable* (*nat-filtration n*) *M*
⟨*proof*⟩

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*) ⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *set-filt-contain*:
assumes *A*∈ *sets* (*nat-filtration n*)
and *w*∈ *A*
shows *pseudo-proj-True n* −' {*pseudo-proj-True n w*} ⊆ *A*
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *measurable-range-rep*:
fixes *f*::*bool stream* ⇒ '*b*::{*t0-space*}
assumes *f* ∈ *borel-measurable* (*nat-filtration n*)
shows *range f* = (⋃ *r*∈(*proj-rep-set n*). {*f*(*r*)})
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *borel-measurable-stake*:
fixes *f*::*bool stream* ⇒ '*b*::{*t0-space*}
assumes *f*∈ *borel-measurable* (*nat-filtration n*)
and *stake n w* = *stake n y*
shows *f w* = *f y*
⟨*proof*⟩

5.3.3 Probability component

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

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

lemma *prob-component-neq-zero*:
assumes $0 < p$
and $p < 1$
shows *prob-component p w n* ≠ *0* ⟨*proof*⟩

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* ⟨*proof*⟩

lemma *stake-preimage-measurable*:
fixes *x*::*bool stream*
assumes *Suc 0* ≤ *n* **and** *M* = *bernoulli-stream p*
shows {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∈ *sets M*
⟨*proof*⟩

lemma *snth-as-fct*:
fixes *b*
assumes *M* = *bernoulli-stream p*
shows *to-stream* −‘ {*w*∈ *space M*. *snth w i* = *b*} = {*X*::*nat*⇒*bool*. *X i* = *b*}
⟨*proof*⟩

lemma *stake-as-fct*:
assumes *Suc 0* ≤ *n* **and** *M*= *bernoulli-stream p*
shows *to-stream* −‘{*w*∈ *space M*. (*stake n w* = *stake n x*)} = {*X*::*nat*⇒*bool*. ∀ *i*. *0* ≤ *i* ∧ *i* ≤ *n*−*1* ⟶ *X i* = *snth x i*}
⟨*proof*⟩

lemma *bernoulli-stream-npref-prob*:
fixes *x*
assumes *M* = *bernoulli-stream p*
shows *emeasure M* {*w*∈ *space M*. (*stake 0 w* = *stake 0 x*)} = *1*
⟨*proof*⟩

lemma *bernoulli-stream-pref-prob*:
fixes *x*

assumes *M* =*bernoulli-stream p*
and *0* ≤ *p* **and** *p* ≤ *1*
shows *n*≥ *Suc 0*⟹ *emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} = (∏ *i*∈{*0*..*n*−*1*}. *prob-component p x i*)
⟨*proof*⟩

lemma *bernoulli-stream-pref-prob′*:
fixes *x*
assumes *M* = *bernoulli-stream p*
and *p* ≤ *1* **and** *0* ≤ *p*
shows *emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} = (∏ *i*∈{*0*..<*n*}. *prob-component p x i*)
⟨*proof*⟩

lemma *bernoulli-stream-stake-prob*:
fixes *x*
assumes *M* = *bernoulli-stream p*
and *p* ≤ *1* **and** *0* ≤ *p*
shows *measure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} = (∏ *i*∈{*0*..<*n*}. *prob-component p x i*)
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *bernoulli-stream-pseudo-prob*:
fixes *x*
assumes *M* = *bernoulli-stream p*
and *p* ≤ *1* **and** *0* ≤ *p*
and *w*∈ *range* (*pseudo-proj-True n*)
shows *measure M* (*pseudo-proj-True n* −‘{*w*} ∩ *space M*) = (∏ *i*∈{*0*..<*n*}. *prob-component p w i*)
⟨*proof*⟩

lemma *bernoulli-stream-element-prob-rec*:
fixes *x*
assumes *M* = *bernoulli-stream p*
and *0* ≤ *p* **and** *p* ≤ *1*
shows ⋀ *n*. *emeasure M* {*w*∈ *space M*. (*stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*)} =
(*emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∗ *prob-component p x n*)
⟨*proof*⟩

lemma *bernoulli-stream-element-prob-rec′*:
fixes *x*
assumes *M* = *bernoulli-stream p*
and *0* ≤ *p* **and** *p* ≤ *1*
shows ⋀ *n*. *measure M* {*w*∈ *space M*. (*stake* (*Suc n*) *w* = *stake* (*Suc n*) *x*)} =
(*measure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ∗ *prob-component p x n*)
⟨*proof*⟩

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*⟩

lemma (**in** *infinite-coin-toss-space*) *bernoulli-stream-pref-prob-pos*:
fixes *x*
assumes *0* < *p*
and *p* < *1*
shows *emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} > *0*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *bernoulli-stream-pref-prob-neq-zero*:
fixes *x*
assumes *0* < *p*
and *p* < *1*
shows *emeasure M* {*w*∈ *space M*. (*stake n w* = *stake n x*)} ≠ *0*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-element-prob-pref*:
assumes *w*∈ *range* (*pseudo-proj-True n*)
shows *emeasure M* {*y*∈ *space M*. ∃ *x* ∈ (*pseudo-proj-True n* −‘{*w*}). *y* = *c* ## *x*} =
prob-component p (*c*##*w*) *0* ∗ *emeasure M* ((*pseudo-proj-True n*) −‘{*w*} ∩ *space M*)
⟨*proof*⟩

5.3.4 Filtration equivalence for the natural filtration

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-null-set*:
assumes *A*∈ *sets* (*nat-filtration n*)
and *0* < *p*
and *p* < *1*
and *emeasure M A* = *0*
shows *A* = {}
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-AE-zero*:
fixes *f*::*bool stream* ⇒ *real*
assumes *AE w in M*. *f w* = *0*
and *f*∈ *borel-measurable* (*nat-filtration n*)
and *0* < *p*
and *p* < *1*
shows ∀ *w*. *f w* = *0*

⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *nat-filtration-AE-eq*:
fixes *f*::*bool stream* ⇒ *real*
assumes *AE w in M. f w = g w*
and *0 < p*
and *p < 1*
and *f*∈ *borel-measurable* (*nat-filtration n*)
and *g*∈ *borel-measurable* (*nat-filtration n*)
shows *f w = g w*
⟨*proof*⟩

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* ⟨*proof*⟩

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*⟩

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*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-img*:
assumes *pseudo-proj-True n w = w*
shows *w*∈ *range* (*pseudo-proj-True n*)
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *sconst-if*:
assumes ⋀*n. snth w n = True*
shows *w = sconst True*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-True-suc-img-pref*:

shows *range* (*pseudo-proj-True* (*Suc n*)) = {*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *True* ## *w*} ∪
{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *False* ## *w*}
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *reindex-pseudo-proj*:
shows (∑ *w*∈*range* (*pseudo-proj-True n*). *f* (*c* ## *w*)) =
(∑ *y*∈{*y*. ∃ *w* ∈ *range* (*pseudo-proj-True n*). *y* = *c* ## *w*}.*f y*)
⟨*proof*⟩

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*
⟨*proof*⟩

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* ∈ {*pseudo-proj-True n x*, *pseudo-proj-False n x*}
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-proj-Suc-preimage*:
shows *range* (*pseudo-proj-True* (*Suc n*)) ∩ (*pseudo-proj-True n*) −‘ {*pseudo-proj-True n x*} =
{*pseudo-proj-True n x*, *pseudo-proj-False n x*}
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *f-borel-Suc-preimage*:
assumes *f*∈ *measurable* (*F n*) *N*
and *set-discriminating n f N*
shows *range* (*pseudo-proj-True* (*Suc n*)) ∩ *f* −‘ {*f x*} =
(*pseudo-proj-True n*) ‘ (*f* −‘ {*f x*}) ∪ (*pseudo-proj-False n*) ‘ (*f* −‘ {*f x*})
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *pseudo-proj-preimage*:
assumes *g*∈ *measurable* (*F n*) *N*
and *set-discriminating n g N*
shows *pseudo-proj-True n* −‘ (*g* −‘ {*g z*}) = *pseudo-proj-True n* −‘ (*pseudo-proj-True n* ‘(*g* −‘ {*g z*}))
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *borel-pseudo-proj-preimage*:

fixes *g*::*bool stream* ⇒ ′*b*::{*t0-space*}
assumes *g*∈ *borel-measurable* (*F n*)
shows *pseudo-proj-True n* −' (*g* −' {*g z*}) = *pseudo-proj-True n* −' (*pseudo-proj-True n* '(*g* −' {*g z*}))
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *pseudo-proj-False-preimage*:
assumes *g*∈ *measurable* (*F n*) *N*
and *set-discriminating n g N*
shows *pseudo-proj-False n* −' (*g* −' {*g z*}) = *pseudo-proj-False n* −' (*pseudo-proj-False n* '(*g* −' {*g z*}))
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *borel-pseudo-proj-False-preimage*:
fixes *g*::*bool stream* ⇒ ′*b*::{*t0-space*}
assumes *g*∈ *borel-measurable* (*F n*)
shows *pseudo-proj-False n* −' (*g* −' {*g z*}) = *pseudo-proj-False n* −' (*pseudo-proj-False n* '(*g* −' {*g z*}))
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *pseudo-proj-preimage′*:
assumes *g*∈ *measurable* (*F n*) *N*
and *set-discriminating n g N*
shows *pseudo-proj-True n* −' (*g* −' {*g z*}) = *g* −' {*g z*}
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *borel-pseudo-proj-preimage′*:
fixes *g*::*bool stream* ⇒ ′*b*::{*t0-space*}
assumes *g*∈ *borel-measurable* (*F n*)
shows *pseudo-proj-True n* −' (*g* −' {*g z*}) = *g* −' {*g z*}
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *pseudo-proj-False-preimage′*:
assumes *g*∈ *measurable* (*F n*) *N*
and *set-discriminating n g N*
shows *pseudo-proj-False n* −' (*g* −' {*g z*}) = *g* −' {*g z*}
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *borel-pseudo-proj-False-preimage′*:
fixes *g*::*bool stream* ⇒ ′*b*::{*t0-space*}
assumes *g*∈ *borel-measurable* (*F n*)
shows *pseudo-proj-False n* −' (*g* −' {*g z*}) = *g* −' {*g z*}
⟨*proof*⟩

5.3.6 Integrals and conditional expectations on the natural filtration

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

lemma (**in** *infinite-cts-filtration*) *cst-nn-integral*:
fixes f::*bool stream*⇒*real*
assumes $f \in$ *borel-measurable* (F *0*)
and $\bigwedge w.\ 0 \leq f\ w$
and f (*sconst True*) = c
shows *integral*^{N} *M f* = *ennreal c* ⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *suc-measurable*:
fixes f::*bool stream* ⇒ *'b*::{*t0-space*}
assumes $f \in$ *borel-measurable* (F (*Suc n*))
shows ($\lambda w.\ f$ (c ## w)) $\in$ *borel-measurable* (F n)
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *F-n-nn-integral-pos*:
fixes f::*bool stream*⇒*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}
∩ *space M*)) ∗ *ennreal* (f w))
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *F-n-integral-pos*:
fixes f::*bool stream*⇒*real*
assumes $f \in$ *borel-measurable* (F n)
and $\forall w.\ 0 \leq f\ w$
shows *has-bochner-integral M f*
($\sum$ $w \in$ *range* (*pseudo-proj-True n*). (*measure M* ((*pseudo-proj-True n*) −‘{w}
∩ *space M*)) ∗ (f w))
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *F-n-integral*:
fixes f::*bool stream*⇒*real*
assumes $f \in$ *borel-measurable* (F n)
shows *has-bochner-integral M f*
($\sum$ $w \in$ *range* (*pseudo-proj-True n*). (*measure M* ((*pseudo-proj-True n*) −‘{w}
∩ *space M*)) ∗ (f w))

⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *F-n-integral-prob-comp*:
fixes *f*::*bool stream*⇒*real*
assumes *f*∈ *borel-measurable* (*F n*)
shows *has-bochner-integral M f*
(∑ *w*∈ *range* (*pseudo-proj-True n*). (*prod* (*prob-component p w*) {*0*..<*n*}) ∗ (*f* *w*))
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *expect-prob-comp*:
fixes *f*::*bool stream*⇒*real*
assumes *f*∈ *borel-measurable* (*F n*)
shows *expectation f* =
(∑ *w*∈ *range* (*pseudo-proj-True n*). (*prod* (*prob-component p w*) {*0*..<*n*}) ∗ (*f* *w*))
⟨*proof*⟩

lemma *sum-union-disjoint′*:
assumes *finite A*
and *finite B*
and *A* ∩ *B* = {}
and *A* ∪ *B* = *C*
shows *sum g C* = *sum g A* + *sum g B*
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *borel-Suc-expectation*:
fixes *f*::*bool stream*⇒ *real*
assumes *f*∈ *borel-measurable* (*F* (*Suc n*))
and *g*∈ *measurable* (*F n*) *N*
and *set-discriminating n g N*
and *g* −‘ {*g z*} ∈ *sets* (*F n*)
and ∀ *y z*. (*g y* = *g z* ∧ *snth y n* = *snth z n*) ⟶ *f y* = *f z*
shows *expectation* (λ*x*. *f x* ∗ *indicator* (*g* −‘ {*g z*}) *x*) =
prob (*g* −‘ {*g z*}) ∗ (*p* ∗ *f* (*pseudo-proj-True n z*) +
(*1* −*p*) ∗ *f* (*pseudo-proj-False n z*))
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *borel-Suc-expectation-pseudo-proj*:
fixes *f*::*bool stream*⇒ *real*
assumes *f*∈ *borel-measurable* (*F* (*Suc n*))
shows *expectation* (λ*x*. *f x* ∗ *indicator* (*pseudo-proj-True n* −‘ {*pseudo-proj-True* *n z*}) *x*) =
prob (*pseudo-proj-True n* −‘ {*pseudo-proj-True n z*}) ∗
(*p* ∗ (*f* (*pseudo-proj-True n z*)) + (*1*−*p*) ∗ (*f* (*pseudo-proj-False n z*)))
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *f-borel-Suc-expl-cond-expect*:
assumes $f \in$ *borel-measurable* $(F\ (Suc\ n))$
and $g \in$ *measurable* $(F\ n)\ N$
and *set-discriminating* $n\ g\ N$
and $g\ -`\ \{g\ w\} \in sets\ (F\ n)$
and $\forall\, y\ z.\ (g\ y = g\ z \wedge snth\ y\ n = snth\ z\ n) \longrightarrow f\ y = f\ z$
and $0 < p$
and $p < 1$
shows $expl\text{-}cond\text{-}expect\ M\ g\ f\ w = p * f\ (pseudo\text{-}proj\text{-}True\ n\ w) + (1 - p) * f\ (pseudo\text{-}proj\text{-}False\ n\ w)$
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *f-borel-Suc-real-cond-exp*:
assumes $f \in$ *borel-measurable* $(F\ (Suc\ n))$
and $g \in$ *measurable* $(F\ n)\ N$
and *set-discriminating* $n\ g\ N$
and $\forall\, w.\ g\ -`\ \{g\ w\} \in sets\ (F\ n)$
and $\forall\, r \in range\ g \cap space\ N.\ \exists\, A \in sets\ N.\ range\ g \cap A = \{r\}$
and $\forall\, y\ z.\ (g\ y = g\ z \wedge snth\ y\ n = snth\ z\ n) \longrightarrow f\ y = f\ z$
and $0 < p$
and $p < 1$
shows $AE\ w\ in\ M.\ real\text{-}cond\text{-}exp\ M\ (fct\text{-}gen\text{-}subalgebra\ M\ N\ g)\ f\ w = p * f\ (pseudo\text{-}proj\text{-}True\ n\ w) + (1 - p) * f\ (pseudo\text{-}proj\text{-}False\ n\ w)$
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *f-borel-Suc-real-cond-exp-proj*:
assumes $f \in$ *borel-measurable* $(F\ (Suc\ n))$
and $0 < p$
and $p < 1$
shows $AE\ w\ in\ M.\ real\text{-}cond\text{-}exp\ M\ (fct\text{-}gen\text{-}subalgebra\ M\ M\ (pseudo\text{-}proj\text{-}True\ n))\ f\ w =$
$p * f\ (pseudo\text{-}proj\text{-}True\ n\ w) + (1 - p) * f\ (pseudo\text{-}proj\text{-}False\ n\ w)$
⟨*proof*⟩

5.4 Images of stochastic processes by prefixes of streams

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

5.4.1 Definitions

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

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

lemma *smap-stoch-proc-length*:
  shows *length* (*smap-stoch-proc n f k w*) = *n*
  ⟨*proof*⟩

lemma *smap-stoch-proc-nth*:
  shows *Suc p* ≤ *Suc n* ⟹ *nth* (*smap-stoch-proc* (*Suc n*) *f k w*) *p* = *f* (*k*+*p*) *w*
⟨*proof*⟩

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

lemma *proj-stoch-proc-component*:
  shows *k* < *n* ⟹ (*snth* (*proj-stoch-proc f n w*) *k*) = *f k w*
    and *n* ≤ *k* ⟹ (*snth* (*proj-stoch-proc f n w*) *k*) = *f n w*
⟨*proof*⟩

lemma *proj-stoch-proc-component′*:
  assumes *k* ≤ *n*
  shows *f k x* = *snth* (*proj-stoch-proc f n x*) *k*
  ⟨*proof*⟩

lemma *proj-stoch-proc-eq-snth*:
  assumes *proj-stoch-proc f n x* = *proj-stoch-proc f n y*
and *k* ≤ *n*
shows *f k x* = *f k y*
⟨*proof*⟩

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*⟩

lemma *proj-stoch-adapted-if-adapted*:
  assumes *filtration M F*
  and *adapt-stoch-proc F f N*
  shows *proj-stoch-proc f n* ∈ *measurable* (*F n*) (*stream-space N*)
⟨*proof*⟩

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*) ⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *proj-stoch-proj-invariant*:
fixes $X::nat \Rightarrow bool\ stream \Rightarrow 'b::\{t0\text{-}space\}$
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*⟩

lemma (**in** *infinite-cts-filtration*) *proj-stoch-set-finite-range*:
fixes $X::nat \Rightarrow bool\ stream \Rightarrow 'b::\{t0\text{-}space\}$
assumes *borel-adapt-stoch-proc F X*
shows *finite (range (proj-stoch-proc X n))*
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *proj-stoch-set-discriminating*:
fixes $X::nat \Rightarrow bool\ stream \Rightarrow 'b::\{t0\text{-}space\}$
assumes *borel-adapt-stoch-proc F X*
shows *set-discriminating n (proj-stoch-proc X n) N*
⟨*proof*⟩

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}* = $(\bigcap i \in \{m.\ m \leq n\}.$ $(X\ i)\ -`\ \{X\ i\ w\})$
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *proj-stoch-singleton-set*:
fixes $X::nat \Rightarrow bool\ stream \Rightarrow ('b::t2\text{-}space)$
assumes *borel-adapt-stoch-proc F X*
shows *(proj-stoch-proc X n)* $-`$ *{proj-stoch-proc X n w}* $\in$ *sets (F n)*
⟨*proof*⟩

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

lemma (**in** *infinite-cts-filtration*) *proj-stoch-range-singleton*:
fixes $X::nat \Rightarrow bool\ stream \Rightarrow ('b::t2\text{-}space)$
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*⟩

definition (**in** *infinite-cts-filtration*) *stream-space-single* **where**

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

lemma (**in** *infinite-cts-filtration*) *stream-space-singleI*:
assumes ∃ *U*. *U*∈ *sets* (*stream-space borel*) ∧ *U*∩ (*range X*) = {*r*}
shows *stream-space-single X r* ∈ *sets* (*stream-space borel*) ∧ *stream-space-single X r* ∩ (*range X*) = {*r*}
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*)
fixes *X*::*nat* ⇒ *bool stream* ⇒ (*'b*::*t2-space*)
assumes *borel-adapt-stoch-proc F X*
and *r*∈ *range* (*proj-stoch-proc X n*)
shows *stream-space-single-set*: *stream-space-single* (*proj-stoch-proc X n*) *r* ∈ *sets* (*stream-space borel*)
and *stream-space-single-preimage*: *stream-space-single* (*proj-stoch-proc X n*) *r* ∩ *range* (*proj-stoch-proc X n*) = {*r*}
⟨*proof*⟩

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

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

lemma (**in** *infinite-cts-filtration*) *comp-proj-i-finite*:
fixes *X*::*nat* ⇒ *bool stream* ⇒ *'b*::{*t0-space*}
assumes *borel-adapt-stoch-proc F X*
shows *finite* (*comp-proj-i X n i y*)
⟨*proof*⟩

lemma *stoch-proc-comp-proj-i-preimage*:
assumes $i \leq n$
shows (*X i*) −' {*X i x*} = (⋃*z*∈ *comp-proj-i X n i* (*X i x*). (*proj-stoch-proc X n*) −' {*z*})
⟨*proof*⟩

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

lemma (**in** *infinite-cts-filtration*) *comp-proj-finite*:
fixes *X*::*nat* ⇒ *bool stream* ⇒ *'b*::{*t0-space*}
assumes *borel-adapt-stoch-proc F X*
shows *finite* (*comp-proj X n y*)
⟨*proof*⟩

lemma *stoch-proc-comp-proj-preimage*:
shows $(X\ n) -` \{X\ n\ x\} = (\bigcup z\in \mathit{comp\text{-}proj}\ X\ n\ (X\ n\ x).\ (\mathit{proj\text{-}stoch\text{-}proc}\ X\ n) -` \{z\})$
⟨*proof*⟩

lemma *comp-proj-stoch-proc-preimage*:
shows $(\mathit{proj\text{-}stoch\text{-}proc}\ X\ n) -` \{\mathit{proj\text{-}stoch\text{-}proc}\ X\ n\ x\} = (\bigcap\ i\in \{m.\ m\le n\}.\ (X\ i) -`\{X\ i\ x\})$
⟨*proof*⟩

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

lemma *stoch-proc-filt-space*:
shows $\mathit{space}\ (\mathit{stoch\text{-}proc\text{-}filt}\ M\ X\ N\ n) = \mathit{space}\ M$ ⟨*proof*⟩

lemma *stoch-proc-filt-sets*:
assumes $\bigwedge i.\ i \le n \Longrightarrow (X\ i) \in \mathit{measurable}\ M\ N$
shows $\mathit{sets}\ (\mathit{stoch\text{-}proc\text{-}filt}\ M\ X\ N\ n) = (\mathit{sigma\text{-}sets}\ (\mathit{space}\ M)\ (\bigcup\ i\in \{m.\ m\le n\}.\ \{(X\ i -`A) \cap (\mathit{space}\ M) \mid A.\ A\in \mathit{sets}\ N\}))$
⟨*proof*⟩

lemma *stoch-proc-filt-adapt*:
assumes $\bigwedge n.\ X\ n \in \mathit{measurable}\ M\ N$
shows $\mathit{adapt\text{-}stoch\text{-}proc}\ (\mathit{stoch\text{-}proc\text{-}filt}\ M\ X\ N)\ X\ N$ ⟨*proof*⟩

lemma *stoch-proc-filt-disc-filtr*:
assumes $\bigwedge i.\ (X\ i) \in \mathit{measurable}\ M\ N$
shows $\mathit{disc\text{-}filtr}\ M\ (\mathit{stoch\text{-}proc\text{-}filt}\ M\ X\ N)$ ⟨*proof*⟩

lemma *gen-subalgebra-eq-space-sets*:
assumes $\mathit{space}\ M = \mathit{space}\ N$
and $P = Q$
and $P\subseteq \mathit{Pow}\ (\mathit{space}\ M)$

shows *sets* (*gen-subalgebra M P*) = *sets* (*gen-subalgebra N Q*) ⟨*proof*⟩

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*) ⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *stoch-proc-filt-triv-init*:
fixes *X*::*nat* ⇒ *bool stream* ⇒ *real*
assumes *borel-adapt-stoch-proc nat-filtration X*
shows *init-triv-filt M* (*stoch-proc-filt M X borel*) ⟨*proof*⟩

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

lemma (**in** *infinite-cts-filtration*) *proj-stoch-pre-borel*:
fixes *X*::*nat* ⇒ *bool stream* ⇒ (*'b*::*t2-space*)
assumes *borel-adapt-stoch-proc F X*
shows *proj-stoch-proc X n* −' {*proj-stoch-proc X n x*} ∈ *sets* (*stoch-proc-filt M X borel n*)
⟨*proof*⟩

lemma (**in** *infinite-cts-filtration*) *stoch-proc-filt-gen*:
fixes *X*::*nat* ⇒ *bool stream* ⇒ (*'b*::*t2-space*)
assumes *borel-adapt-stoch-proc F X*
shows *stoch-proc-filt M X borel n* = *fct-gen-subalgebra M* (*stream-space borel*) (*proj-stoch-proc X n*)
⟨*proof*⟩

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*) ⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*)

assumes *N* = *bernoulli-stream q*
and *0* ≤ *q*
and *q* ≤ *1*
and *0* < *p*
and *p* < *1*
and *filt-equiv nat-filtration M N*
shows *filt-equiv-sgt*: *0* < *q* **and** *filt-equiv-slt*: *q* < *1*
⟨*proof*⟩

lemma *stoch-proc-filt-filt-equiv*:
assumes *filt-equiv F M N*
shows *stoch-proc-filt M f P n* = *stoch-proc-filt N f P n* ⟨*proof*⟩

lemma *filt-equiv-filt*:
assumes *filt-equiv F M N*
and *filtration M G*
shows *filtration N G* ⟨*proof*⟩

lemma *filt-equiv-borel-AE-eq-iff*:
fixes *f*::*'a*⇒ *real*
assumes *filt-equiv F M N*
and *f*∈ *borel-measurable* (*F t*)
and *g*∈ *borel-measurable* (*F t*)
and *prob-space N*
and *prob-space M*
shows (*AE w in M*. *f w* = *g w*) ⟷ (*AE w in N*. *f w* = *g w*)
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *filt-equiv-triv-init*:
assumes *filt-equiv F M N*
and *init-triv-filt M G*
shows *init-triv-filt N G* ⟨*proof*⟩

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

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

begin

6 Geometric random walk

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

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

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

lemma (**in** *prob-grw*) *geom-rand-walk-borel-measurable*:
shows (*geom-proc n*) ∈ *borel-measurable M*
⟨*proof*⟩

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

lemma (**in** *prob-grw*) *geom-rand-walk-pseudo-proj-False*:
shows *geom-proc n* = *geom-proc n* ∘ *pseudo-proj-False n*
⟨*proof*⟩

lemma (**in** *prob-grw*) *geom-rand-walk-borel-adapted*:
shows *borel-adapt-stoch-proc nat-filtration geom-proc*
⟨*proof*⟩

lemma (**in** *prob-grw*) *grw-succ-img*:
assumes (*geom-proc n*) −‘ {*x*} ≠ {}
shows (*geom-proc* (*Suc n*)) ‘ ((*geom-proc n*) −‘ {*x*}) = {*u*∗*x*, *d*∗*x*}
⟨*proof*⟩

lemma (**in** *prob-grw*) *geom-rand-walk-strictly-positive*:
assumes *0* < *init*
and *0* < *d*
and *d* < *u*
shows ∀ *n w*. *0* < *geom-proc n w*
⟨*proof*⟩

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*⟩

end

7 Fair Prices

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

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

7.1 Preliminary results

lemma (**in** *prob-space*) *finite-borel-measurable-integrable*:
assumes *f*∈ *borel-measurable M*
and *finite* (*f*‘(*space M*))
shows *integrable M f*
⟨*proof*⟩

7.1.1 On the almost everywhere filter

lemma *AE-eq-trans*[*trans*]:
assumes *AE x in M. A x* = *B x*
and *AE x in M. B x* = *C x*
shows *AE x in M. A x* = *C x*
⟨*proof*⟩

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

lemma *AE-add*:
assumes *AE w in M. f w* = *g w*
and *AE w in M. f′ w* = *g′ w*
shows *AE w in M. f w* + *f′ w* = *g w* + *g′ w* ⟨*proof*⟩

lemma *AE-sum*:
assumes *finite I*

and ∀ *i*∈*I*. *AE w in M*. *f i w* = *g i w*
shows *AE w in M*. (∑*i*∈ *I*. *f i w*) = (∑*i*∈ *I*. *g i w*) ⟨*proof*⟩

lemma *AE-eq-cst*:
assumes *AE w in M*. (λ*w*. *c*) *w* = (λ*w*. *d*) *w*
and *emeasure M* (*space M*) ≠ *0*
shows *c* = *d*
⟨*proof*⟩

7.1.2 On conditional expectations

lemma (**in** *prob-space*) *subalgebra-sigma-finite*:
assumes *subalgebra M N*
shows *sigma-finite-subalgebra M N* ⟨*proof*⟩

lemma (**in** *prob-space*) *trivial-subalg-cond-expect-AE*:
assumes *subalgebra M N*
and *sets N* = {{}, *space M*}
and *integrable M f*
shows *AE x in M*. *real-cond-exp M N f x* = (λ*x*. *expectation f*) *x*
⟨*proof*⟩

lemma (**in** *prob-space*) *triv-subalg-borel-eq*:
assumes *subalgebra M F*
and *sets F* = {{}, *space M*}
and *AE x in M*. *f x* = (*c*::′*b*::{*t2-space*})
and *f*∈ *borel-measurable F*
shows ∀*x*∈ *space M*. *f x* = *c*
⟨*proof*⟩

lemma (**in** *prob-space*) *trivial-subalg-cond-expect-eq*:
assumes *subalgebra M N*
and *sets N* = {{}, *space M*}
and *integrable M f*
shows ∀*x*∈ *space M*. *real-cond-exp M N f x* = *expectation f*
⟨*proof*⟩

lemma (**in** *sigma-finite-subalgebra*) *real-cond-exp-cong′*:
assumes ∀*w* ∈ *space M*. *f w* = *g w*
and *f*∈ *borel-measurable M*
shows *AE w in M*. *real-cond-exp M F f w* = *real-cond-exp M F g w*
⟨*proof*⟩

lemma (**in** *sigma-finite-subalgebra*) *real-cond-exp-bsum* :
fixes $f::'b \Rightarrow 'a \Rightarrow real$
assumes [*measurable*]: $\bigwedge i.\ i{\in}I \Longrightarrow integrable\ M\ (f\ i)$
shows $AE\ x\ in\ M.\ real\text{-}cond\text{-}exp\ M\ F\ (\lambda x.\ \sum i{\in}I.\ f\ i\ x)\ x = (\sum i{\in}I.\ real\text{-}cond\text{-}exp\ M\ F\ (f\ i)\ x)$
⟨*proof*⟩

7.2 Financial formalizations

7.2.1 Markets

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

typedef (*'a*,*'c*) *discrete-market* = {(*s*::(*'c set*), *a*::*'c* ⇒ (*nat* ⇒ *'a* ⇒ *real*)). *stk-strict-subs s*} ⟨*proof*⟩

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

definition *assets* **where**

assets Mkt = *UNIV*

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

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

lemma *prices-of*:
shows *prices* (*discrete-market-of S A*) = *A*
⟨*proof*⟩

lemma *stocks-of*:
assumes *UNIV* ≠ *S*
shows *stocks* (*discrete-market-of S A*) = *S*
⟨*proof*⟩

lemma *mkt-stocks-assets*:
shows *stk-strict-subs* (*stocks Mkt*) ⟨*proof*⟩

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**
$qty\text{-}empty = (\lambda\ (x::'a)\ (n::nat)\ w.\ 0::real)$

definition *qty-single* **where**
$qty\text{-}single\ asset\ qt\text{-}proc = (qty\text{-}empty(asset := qt\text{-}proc))$

definition $qty\text{-}sum::('b \Rightarrow nat \Rightarrow 'a \Rightarrow real) \Rightarrow ('b \Rightarrow nat \Rightarrow 'a \Rightarrow real) \Rightarrow ('b \Rightarrow nat \Rightarrow 'a \Rightarrow real)$ **where**
$qty\text{-}sum\ pf1\ pf2 = (\lambda x\ n\ w.\ pf1\ x\ n\ w + pf2\ x\ n\ w)$

definition $qty\text{-}mult\text{-}comp::('b \Rightarrow nat \Rightarrow 'a \Rightarrow real) \Rightarrow (nat \Rightarrow 'a \Rightarrow real) \Rightarrow ('b \Rightarrow nat \Rightarrow 'a \Rightarrow real)$ **where**
$qty\text{-}mult\text{-}comp\ pf1\ qty = (\lambda x\ n\ w.\ (pf1\ x\ n\ w) * (qty\ n\ w))$

definition $qty\text{-}rem\text{-}comp::('b \Rightarrow nat \Rightarrow 'a \Rightarrow real) \Rightarrow 'b \Rightarrow ('b \Rightarrow nat \Rightarrow 'a \Rightarrow real)$ **where**
$qty\text{-}rem\text{-}comp\ pf1\ x = pf1(x:=(\lambda n\ w.\ 0))$

definition *qty-replace-comp* **where**
$qty\text{-}replace\text{-}comp\ pf1\ x\ pf2 = qty\text{-}sum\ (qty\text{-}rem\text{-}comp\ pf1\ x)\ (qty\text{-}mult\text{-}comp\ pf2\ (pf1\ x))$

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

definition $support\text{-}set::('b \Rightarrow nat \Rightarrow 'a \Rightarrow real) \Rightarrow 'b\ set$ **where**
$support\text{-}set\ p = \{x.\ \exists\ n\ w.\ p\ x\ n\ w \neq 0\}$

lemma *qty-empty-support-set*:
shows $support\text{-}set\ qty\text{-}empty = \{\}$ ⟨*proof*⟩
lemma *sum-support-set*:
shows $support\text{-}set\ (qty\text{-}sum\ pf1\ pf2) \subseteq (support\text{-}set\ pf1) \cup (support\text{-}set\ pf2)$
⟨*proof*⟩

lemma *mult-comp-support-set*:
shows $support\text{-}set\ (qty\text{-}mult\text{-}comp\ pf1\ qty) \subseteq (support\text{-}set\ pf1)$
⟨*proof*⟩

lemma *remove-comp-support-set*:
shows $support\text{-}set\ (qty\text{-}rem\text{-}comp\ pf1\ x) \subseteq ((support\text{-}set\ pf1) - \{x\})$
⟨*proof*⟩

lemma *replace-comp-support-set*:
shows $support\text{-}set\ (qty\text{-}replace\text{-}comp\ pf1\ x\ pf2) \subseteq (support\text{-}set\ pf1 - \{x\}) \cup support\text{-}set\ pf2$
⟨*proof*⟩

lemma *single-comp-support*:
shows $support\text{-}set\ (qty\text{-}single\ asset\ qty) \subseteq \{asset\}$

⟨*proof*⟩

lemma *single-comp-nz-support*:
  assumes $\exists$ *n w. qty n w* $\neq$ *0*
  shows *support-set* (*qty-single asset qty*) = {*asset*}
⟨*proof*⟩

Portfolios **definition** *portfolio* **where**
  portfolio p $\longleftrightarrow$ *finite* (*support-set p*)

definition *stock-portfolio* :: (*'a*, *'b*) *discrete-market* $\Rightarrow$ (*'b* $\Rightarrow$ *nat* $\Rightarrow$ *'a* $\Rightarrow$ *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*) ⟨*proof*⟩

lemma *sum-basic-support-set*:
  assumes *stock-portfolio Mkt pf1*
  and *stock-portfolio Mkt pf2*
shows *stock-portfolio Mkt* (*qty-sum pf1 pf2*) ⟨*proof*⟩

lemma *mult-comp-portfolio*:
  assumes *portfolio pf1*
shows *portfolio* (*qty-mult-comp pf1 qty*) ⟨*proof*⟩

lemma *mult-comp-basic-support-set*:
  assumes *stock-portfolio Mkt pf1*
shows *stock-portfolio Mkt* (*qty-mult-comp pf1 qty*) ⟨*proof*⟩

lemma *remove-comp-portfolio*:
  assumes *portfolio pf1*
shows *portfolio* (*qty-rem-comp pf1 x*) ⟨*proof*⟩

lemma *remove-comp-basic-support-set*:
  assumes *stock-portfolio Mkt pf1*
shows *stock-portfolio Mkt* (*qty-mult-comp pf1 qty*) ⟨*proof*⟩

lemma *replace-comp-portfolio*:
  assumes *portfolio pf1*
  and *portfolio pf2*
shows *portfolio* (*qty-replace-comp pf1 x pf2*) ⟨*proof*⟩

lemma *replace-comp-stocks*:
  assumes *support-set pf1* ⊆ *stocks Mkt* ∪ {*x*}
  and *support-set pf2* ⊆ *stocks Mkt*
shows *support-set* (*qty-replace-comp pf1 x pf2*) ⊆ *stocks Mkt*
⟨*proof*⟩

lemma *single-comp-portfolio*:
  shows *portfolio* (*qty-single asset qty*)
  ⟨*proof*⟩

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

lemma *subset-val-process′*:
  assumes *finite A*
  and *support-set p* ⊆ *A*
shows *val-process Mkt p n w* = (*sum* (λ*x.* ((*prices Mkt*) *x n w*) ∗ (*p x* (*Suc n*) *w*)) *A*)
⟨*proof*⟩

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

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

lemma *remove-comp-values*:
  assumes *x* ≠ *y*
  shows ∀ *n w. pf1 x n w* = (*qty-rem-comp pf1 y*) *x n w*

⟨*proof*⟩

lemma *remove-comp-val-process*:
assumes *portfolio pf1*
shows $\forall$ *n w. val-process Mkt* (*qty-rem-comp pf1 y*) *n w* = ((*val-process Mkt pf1*) *n w*) − (*prices Mkt y n w*)∗ (*pf1 y* (*Suc n*) *w*)
⟨*proof*⟩

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*⟩

lemma *qty-single-val-process*:
shows *val-process Mkt* (*qty-single asset qty*) *n w* =
prices Mkt asset n w ∗ *qty* (*Suc n*) *w*
⟨*proof*⟩

7.2.3 Trading strategies

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

Discrete predictable processes

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

definition (**in** *disc-filtr-prob-space*) *support-adapt*:: (′*a*, ′*b*) *discrete-market* ⇒ (′*b* ⇒ *nat* ⇒ ′*a* ⇒ *real*) ⇒ *bool* **where**
support-adapt Mkt pf ⟷ (∀ *asset* ∈ *support-set pf. borel-adapt-stoch-proc F* (*prices Mkt asset*))

lemma (**in** *disc-filtr-prob-space*) *quantity-adapted*:
assumes ∀ *asset* ∈ *support-set p. p asset* (*Suc n*) ∈ *borel-measurable* (*F n*)
∀ *asset* ∈ *support-set p. prices Mkt asset n* ∈ *borel-measurable* (*F n*)
shows *val-process Mkt p n* ∈ *borel-measurable* (*F n*)
⟨*proof*⟩

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*) ⟨*proof*⟩

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*) ⟨*proof*⟩

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* ⟨*proof*⟩

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

lemma (**in** *disc-equity-market*) *initial-valueI*:
assumes *trading-strategy pf*
and *support-adapt Mkt pf*
shows $\forall$ *w*$\in$ *space M*. *val-process Mkt pf 0 w* = *initial-value pf* ⟨*proof*⟩

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*⟩

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*⟩

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*) ⟨*proof*⟩

lemma (**in** *disc-equity-market*) *qty-empty-trading-strat*:
  shows *trading-strategy qty-empty* ⟨*proof*⟩

lemma (**in** *disc-equity-market*) *sum-trading-strat*:
  assumes *trading-strategy pf1*
  and *trading-strategy pf2*
shows *trading-strategy* (*qty-sum pf1 pf2*)
⟨*proof*⟩

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*⟩

lemma (**in** *disc-equity-market*) *remove-comp-trading-strat*:
  assumes *trading-strategy pf1*
shows *trading-strategy* (*qty-rem-comp pf1 x*)
⟨*proof*⟩

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*) ⟨*proof*⟩

7.2.4 Self-financing portfolios

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

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

lemma (**in** *disc-filtr-prob-space*) *quantity-updated-borel*:
  assumes ∀ *n*. ∀ *asset* ∈ *support-set p*. *p asset* (*Suc n*) ∈ *borel-measurable* (*F n*)
and ∀ *n*. ∀ *asset* ∈ *support-set p*. *prices Mkt asset n* ∈ *borel-measurable* (*F n*)
shows ∀ *n*. *cls-val-process Mkt p n* ∈ *borel-measurable* (*F n*)
⟨*proof*⟩

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*⟩

lemma *subset-cls-val-process*:
assumes *finite A*
and *support-set p* ⊆ *A*
shows ∀ *n w*. *cls-val-process Mkt p* (*Suc n*) *w* = (*sum* (λ*x*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*)) *A*)
⟨*proof*⟩

lemma *subset-cls-val-process*′:
assumes *finite A*
and *support-set p* ⊆ *A*
shows *cls-val-process Mkt p* (*Suc n*) *w* = (*sum* (λ*x*. ((*prices Mkt*) *x* (*Suc n*) *w*) ∗ (*p x* (*Suc n*) *w*)) *A*)
⟨*proof*⟩

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

lemma *sum-cls-val-process0*:
assumes *portfolio pf1*
and *portfolio pf2*
shows ∀ *w*. *cls-val-process Mkt* (*qty-sum pf1 pf2*) *0 w* =
(*cls-val-process Mkt pf1*) *0 w* + (*cls-val-process Mkt pf2*) *0 w* ⟨*proof*⟩

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

lemma *mult-comp-cls-val-process0*:
assumes *portfolio pf1*
shows ∀ *w*. *cls-val-process Mkt* (*qty-mult-comp pf1 qty*) *0 w* =
((*cls-val-process Mkt pf1*) *0 w*) ∗ (*qty* (*Suc 0*) *w*) ⟨*proof*⟩

lemma *mult-comp-cls-val-process-Suc*:
assumes *portfolio pf1*
shows ∀ *n w*. *cls-val-process Mkt* (*qty-mult-comp pf1 qty*) (*Suc n*) *w* =

((*cls-val-process Mkt pf1*) (*Suc n*) *w*) ∗ (*qty* (*Suc n*) *w*)
⟨*proof*⟩

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

lemma *remove-comp-cls-val-process-Suc*:
assumes *portfolio pf1*
shows ∀ *n w*. *cls-val-process Mkt* (*qty-rem-comp pf1 y*) (*Suc n*) *w* =
((*cls-val-process Mkt pf1*) (*Suc n*) *w*) − (*prices Mkt y* (*Suc n*) *w*)∗ (*pf1 y* (*Suc n*) *w*)
⟨*proof*⟩

lemma *replace-comp-cls-val-process0*:
assumes ∀ *w*. *prices Mkt x 0 w* = *cls-val-process Mkt pf2 0 w*
and *portfolio pf1*
and *portfolio pf2*
shows ∀ *w*. *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) *0 w* = *cls-val-process Mkt pf1 0 w*
⟨*proof*⟩

lemma *replace-comp-cls-val-process-Suc*:
assumes ∀ *n w*. *prices Mkt x* (*Suc n*) *w* = *cls-val-process Mkt pf2* (*Suc n*) *w*
and *portfolio pf1*
and *portfolio pf2*
shows ∀ *n w*. *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) (*Suc n*) *w* = *cls-val-process Mkt pf1* (*Suc n*) *w*
⟨*proof*⟩

lemma *replace-comp-cls-val-process*:
assumes ∀ *n w*. *prices Mkt x n w* = *cls-val-process Mkt pf2 n w*
and *portfolio pf1*
and *portfolio pf2*
shows ∀ *n w*. *cls-val-process Mkt* (*qty-replace-comp pf1 x pf2*) *n w* = *cls-val-process Mkt pf1 n w*
⟨*proof*⟩

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*⟩

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

lemma *self-financingE*:
assumes *self-financing Mkt p*
shows ∀ *n*. *val-process Mkt p n* = *cls-val-process Mkt p n*
⟨*proof*⟩

lemma *static-portfolio-self-financing*:
assumes ∀ *x* ∈ *support-set p*. (∀ *w i*. *p x i w* = *p x* (*Suc i*) *w*)
shows *self-financing Mkt p*
⟨*proof*⟩

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*⟩

lemma *mult-time-constant-self-financing*:
assumes *portfolio pf1*
and *self-financing Mkt pf1*
and ∀ *n w*. *qty n w* = *qty* (*Suc n*) *w*
shows *self-financing Mkt* (*qty-mult-comp pf1 qty*)
⟨*proof*⟩

lemma *replace-comp-self-financing*:
assumes ∀ *n w*. *prices Mkt x n w* = *cls-val-process Mkt pf2 n w*
and *portfolio pf1*
and *portfolio pf2*
and *self-financing Mkt pf1*
and *self-financing Mkt pf2*
shows *self-financing Mkt* (*qty-replace-comp pf1 x pf2*)
⟨*proof*⟩

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

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

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*) ⟨*proof*⟩

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

lemma (**in** *disc-equity-market*) *remaining-qty-adapted*:
assumes *borel-adapt-stoch-proc F* (*prices Mkt asset*)
and *trading-strategy pf*
and *support-adapt Mkt pf*
shows *borel-adapt-stoch-proc F* (*remaining-qty Mkt v pf asset*) ⟨*proof*⟩

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*) ⟨*proof*⟩

lemma *self-finance-init*:

assumes ∀ *w. prices Mkt asset 0 w* ≠ *0*
and *portfolio pf*
shows *val-process Mkt* (*self-finance Mkt v pf asset*) *0 w* = *v*
⟨*proof*⟩

lemma *self-finance-succ*:
assumes *prices Mkt asset* (*Suc n*) *w* ≠ *0*
and *portfolio pf*
shows *val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* = *prices Mkt asset* (*Suc n*) *w* ∗ *remaining-qty Mkt v pf asset* (*Suc n*) *w* +
cls-val-process Mkt pf (*Suc n*) *w*
⟨*proof*⟩

lemma *self-finance-updated*:
assumes *prices Mkt asset* (*Suc n*) *w* ≠ *0*
and *portfolio pf*
shows *cls-val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* =
cls-val-process Mkt pf (*Suc n*) *w* + *prices Mkt asset* (*Suc n*) *w* ∗ (*remaining-qty Mkt v pf asset*) (*Suc n*) *w*
⟨*proof*⟩

lemma *self-finance-charact*:
assumes ∀ *n w. prices Mkt asset* (*Suc n*) *w* ≠ *0*
and *portfolio pf*
shows *self-financing Mkt* (*self-finance Mkt v pf asset*)
⟨*proof*⟩

7.2.5 Replicating portfolios

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

lemma (**in** *disc-filtr-prob-space*) *price-structure-init*:
assumes *price-structure pyf T* π *pr*
shows ∀ *w*∈ *space M*. *pr 0 w* = π ⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *price-structure-borel-measurable*:
assumes *price-structure pyf T* π *pr*
shows *pr T* ∈ *borel-measurable* (*F T*) ⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *price-structure-maturity*:
assumes *price-structure pyf T* π *pr*
shows *AE w in M*. *pr T w* = *pyf w* ⟨*proof*⟩

definition (**in** *disc-equity-market*) *replicating-portfolio* **where**

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

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

lemma (**in** *disc-equity-market*) *replicating-price-process*:
assumes *replicating-portfolio pf der matur*
and *support-adapt Mkt pf*
shows *price-structure der matur* (*initial-value pf*) (*cls-val-process Mkt pf*)
⟨*proof*⟩

7.2.6 Arbitrages

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

lemma (**in** *disc-filtr-prob-space*) *arbitrage-processE*:
assumes *arbitrage-process Mkt p*
shows (∃ *m.* (*self-financing Mkt p*) ∧ (*trading-strategy p*) ∧
(∀ *w* ∈ *space M. cls-val-process Mkt p 0 w = 0*) ∧
(*AE w in M. 0* ≤ *cls-val-process Mkt p m w*) ∧
0 < 𝒫(*w in M. cls-val-process Mkt p m w* > *0*))
⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *arbitrage-processI*:
assumes (∃ *m.* (*self-financing Mkt p*) ∧ (*trading-strategy p*) ∧
(∀ *w* ∈ *space M. cls-val-process Mkt p 0 w = 0*) ∧
(*AE w in M. 0* ≤ *cls-val-process Mkt p m w*) ∧
0 < 𝒫(*w in M. cls-val-process Mkt p m w* > *0*))
shows *arbitrage-process Mkt p* ⟨*proof*⟩

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

lemma (**in** *disc-filtr-prob-space*) *arbitrage-val-process*:
assumes *arbitrage-process Mkt pf1*
and *self-financing Mkt pf2*

and *trading-strategy pf2*
and ∀ *n w. cls-val-process Mkt pf1 n w* = *cls-val-process Mkt pf2 n w*
shows *arbitrage-process Mkt pf2*
⟨*proof*⟩

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

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

lemma *coincides-cls-val-process'*:
assumes *coincides-on Mkt Mkt2 A*
and *support-set pf* ⊆ *A*
shows ∀ *n w. cls-val-process Mkt pf* (*Suc n*) *w* = *cls-val-process Mkt2 pf* (*Suc n*) *w*
⟨*proof*⟩

lemma *coincides-cls-val-process*:
assumes *coincides-on Mkt Mkt2 A*
and *support-set pf* ⊆ *A*
shows ∀ *n w. cls-val-process Mkt pf n w* = *cls-val-process Mkt2 pf n w*
⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *coincides-on-self-financing*:
assumes *coincides-on Mkt Mkt2 A*
and *support-set p* ⊆ *A*
and *self-financing Mkt p*
shows *self-financing Mkt2 p*
⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *coincides-on-arbitrage*:
assumes *coincides-on Mkt Mkt2 A*
and *support-set p* ⊆ *A*
and *arbitrage-process Mkt p*
shows *arbitrage-process Mkt2 p*
⟨*proof*⟩

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* ⟨*proof*⟩

lemma *coincides-stocks-val-process*:
assumes *stock-portfolio Mkt pf*
and *coincides-on Mkt Mkt2* (*stocks Mkt*)
shows ∀ *n w. val-process Mkt pf n w* = *val-process Mkt2 pf n w* ⟨*proof*⟩

lemma *coincides-stocks-cls-val-process*:
assumes *stock-portfolio Mkt pf*
and *coincides-on Mkt Mkt2* (*stocks Mkt*)
shows ∀ *n w. cls-val-process Mkt pf n w* = *cls-val-process Mkt2 pf n w* ⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *coincides-on-adapted-val-process*:
assumes *coincides-on Mkt Mkt2 A*
and *support-set p* ⊆ *A*
and *borel-adapt-stoch-proc F* (*val-process Mkt p*)
shows *borel-adapt-stoch-proc F* (*val-process Mkt2 p*) ⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *coincides-on-adapted-cls-val-process*:
assumes *coincides-on Mkt Mkt2 A*
and *support-set p* ⊆ *A*
and *borel-adapt-stoch-proc F* (*cls-val-process Mkt p*)
shows *borel-adapt-stoch-proc F* (*cls-val-process Mkt2 p*) ⟨*proof*⟩

7.2.7 Fair prices

definition (**in** *disc-filtr-prob-space*) *fair-price* **where**
fair-price Mkt π *pyf matur* ⟷
(∃ *pr. price-structure pyf matur* π *pr* ∧
(∀ *x Mkt2 p.* (*x*∉ *stocks Mkt* ⟶
((*coincides-on Mkt Mkt2* (*stocks Mkt*)) ∧ (*prices Mkt2 x* = *pr*) ∧ *portfolio p* ∧ *support-set p* ⊆ *stocks Mkt* ∪ {*x*} ⟶
¬ *arbitrage-process Mkt2 p*))))

lemma (**in** *disc-filtr-prob-space*) *fair-priceI*:
assumes *fair-price Mkt* π *pyf matur*
shows (∃ *pr. price-structure pyf matur* π *pr* ∧
(∀ *x.* (*x*∉ *stocks Mkt* ⟶
(∀ *Mkt2 p.* (*coincides-on Mkt Mkt2* (*stocks Mkt*)) ∧ (*prices Mkt2 x* = *pr*) ∧ *portfolio p* ∧ *support-set p* ⊆ *stocks Mkt* ∪ {*x*} ⟶
¬ *arbitrage-process Mkt2 p*)))) ⟨*proof*⟩

Existence when replicating portfolio **lemma** (**in** *disc-equity-market*) *replicating-fair-price*:
assumes *viable-market Mkt*
and *replicating-portfolio pf der matur*

and *support-adapt Mkt pf*
shows *fair-price Mkt* (*initial-value pf*) *der matur*
⟨*proof*⟩

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

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

lemma (**in** *disc-market-pos-stock*) *pos-stock-borel-adapted*:
  shows *borel-adapt-stoch-proc F* (*prices Mkt pos-stock*)
  ⟨*proof*⟩

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

lemma (**in** *disc-filtr-prob-space*) *static-quantities-trading-strat*:
  assumes *static-quantities p*
  and *finite* (*support-set p*)
  shows *trading-strategy p* ⟨*proof*⟩

lemma *two-component-support-set*:
  assumes ∃ *n w*. *a n w* ≠ *0*
  and ∃ *n w*. *b n w*≠ *0*
  and *x* ≠ *y*
shows *support-set* ((λ (*x*::′*b*) (*n*::*nat*) (*w*::′*a*). *0*::*real*)(*x*:= *a*, *y*:= *b*)) = {*x*,*y*}
⟨*proof*⟩

lemma *two-component-val-process*:
  assumes *arb-pf* = ((λ (*x*::′*b*) (*n*::*nat*) (*w*::′*a*). *0*::*real*)(*x*:= *a*, *y*:= *b*))
  and *portfolio arb-pf*
  and *x* ≠ *y*
  and ∃ *n w*. *a n w* ≠ *0*
  and ∃ *n w*. *b n w*≠ *0*
  shows *val-process Mkt arb-pf n w* =
    prices Mkt y n w ∗ *b* (*Suc n*) *w* + *prices Mkt x n w* ∗ *a* (*Suc n*) *w*
⟨*proof*⟩

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*:=*pr*)) = *support-set p* $\cup$ {*x*}
⟨*proof*⟩

lemma *fix-asset-price*:
shows $\exists$ *x Mkt2. x* $\notin$ *stocks Mkt* $\wedge$
coincides-on Mkt Mkt2 (*stocks Mkt*) $\wedge$
prices Mkt2 x = *pr*
⟨*proof*⟩

lemma (**in** *disc-market-pos-stock*) *arbitrage-portfolio-properties*:
assumes *price-structure der matur* π *pr*
and *replicating-portfolio pf der matur*
and (*coincides-on Mkt Mkt2* (*stocks Mkt*))
and (*prices Mkt2 x* = *pr*)
and *x*$\notin$ *stocks Mkt*
and *diff-inv* = (π − *initial-value pf*) / *constant-image* (*prices Mkt pos-stock 0*)
and *diff-inv* $\neq$ *0*
and *arb-pf* = (λ (*x*::'*b*) (*n*::*nat*) (*w*::'*a*). *0*::*real*)(*x*:= (λ *n w.* −*1*), *pos-stock* := (λ *n w. diff-inv*))
and *contr-pf* = *qty-sum arb-pf pf*
shows *self-financing Mkt2 contr-pf*
and *trading-strategy contr-pf*
and $\forall$ *w*$\in$ *space M. cls-val-process Mkt2 contr-pf 0 w* = *0*
and *0* < *diff-inv* ⟶ (*AE w in M. 0* < *cls-val-process Mkt2 contr-pf matur w*)
and *diff-inv* < *0* ⟶ (*AE w in M. 0* > *cls-val-process Mkt2 contr-pf matur w*)
and *support-set arb-pf* = {*x*, *pos-stock*}
and *portfolio contr-pf*
⟨*proof*⟩

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*⟩

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* (*Suc 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*⟩

lemma (**in** *disc-equity-market*) *mult-comp-val-process-measurable*:
  assumes *val-process Mkt2 pf n* ∈*borel-measurable* (*F n*)
  and *portfolio pf*
  and *qty* (*Suc n*) ∈ *borel-measurable* (*F n*)
shows *val-process Mkt2* (*qty-mult-comp pf qty*) *n* ∈ *borel-measurable* (*F n*)
  ⟨*proof*⟩

lemma (**in** *disc-market-pos-stock*) *repl-fair-price-unique*:
  assumes *replicating-portfolio pf der matur*
  and *fair-price Mkt* π *der matur*
shows π = *initial-value pf*
⟨*proof*⟩

7.3 Risk-neutral probability space

7.3.1 risk-free rate and discount factor processes

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

lemma *disc-rfr-proc-borel-measurable*:
  shows (*disc-rfr-proc r*) *n* ∈ *borel-measurable M*
⟨*proof*⟩

lemma *disc-rfr-proc-nonrandom*:
  fixes *r*::*real*
  shows ⋀*n*. *disc-rfr-proc r n* ∈ *borel-measurable* (*F 0*) ⟨*proof*⟩

lemma (**in** *disc-equity-market*) *disc-rfr-constant-time*:
shows ∃ *c*. ∀ *w* ∈ *space* (*F 0*). (*disc-rfr-proc r n*) *w* = *c*
⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *disc-rfr-proc-borel-adapted*:
  shows *borel-adapt-stoch-proc F* (*disc-rfr-proc r*)
⟨*proof*⟩

lemma *disc-rfr-proc-positive*:

assumes $-1 < r$
shows $\bigwedge n\ w\ .\ 0 < disc\text{-}rfr\text{-}proc\ r\ n\ w$
⟨*proof*⟩

lemma (**in** *prob-space*) *disc-rfr-constant-time-pos*:
assumes $-1 < r$
shows $\exists\, c > 0.\ \forall\, w \in space\ M.\ (disc\text{-}rfr\text{-}proc\ r\ n)\ w = c$
⟨*proof*⟩

lemma *disc-rfr-proc-Suc-div*:
assumes $-1 < r$
shows $\bigwedge w.\ disc\text{-}rfr\text{-}proc\ r\ (Suc\ n)\ w / disc\text{-}rfr\text{-}proc\ r\ n\ w = 1{+}r$
⟨*proof*⟩

definition *discount-factor* **where**
$discount\text{-}factor\ r\ n = (\lambda w.\ inverse\ (disc\text{-}rfr\text{-}proc\ r\ n\ w))$

lemma *discount-factor-times-rfr*:
assumes $-1 < r$
shows $(1{+}r) * discount\text{-}factor\ r\ (Suc\ n)\ w = discount\text{-}factor\ r\ n\ w$ ⟨*proof*⟩

lemma *discount-factor-borel-measurable*:
shows $discount\text{-}factor\ r\ n \in borel\text{-}measurable\ M$ ⟨*proof*⟩

lemma *discount-factor-init*:
shows $discount\text{-}factor\ r\ 0 = (\lambda w.\ 1)$ ⟨*proof*⟩

lemma *discount-factor-nonrandom*:
shows $discount\text{-}factor\ r\ n \in borel\text{-}measurable\ M$ ⟨*proof*⟩

lemma *discount-factor-positive*:
assumes $-1 < r$
shows $\bigwedge n\ w\ .\ 0 < discount\text{-}factor\ r\ n\ w$ ⟨*proof*⟩

lemma (**in** *prob-space*) *discount-factor-constant-time-pos*:
assumes $-1 < r$
shows $\exists\, c > 0.\ \forall\, w \in space\ M.\ (discount\text{-}factor\ r\ n)\ w = c$ ⟨*proof*⟩

locale *rsk-free-asset* =
fixes *Mkt r risk-free-asset*
assumes *acceptable-rate*: $-1 < r$

and *rf-price*: *prices Mkt risk-free-asset* = *disc-rfr-proc r*
and *rf-stock*: *risk-free-asset* ∈ *stocks Mkt*

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

sublocale *rfr-disc-equity-market* ⊆ *disc-market-pos-stock* - - - *risk-free-asset*
⟨*proof*⟩

7.3.2 Discounted value of a stochastic process

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

lemma (**in** *rfr-disc-equity-market*) *discounted-rfr*:
shows *discounted-value r* (*prices Mkt risk-free-asset*) *n w* = *1* ⟨*proof*⟩

lemma *discounted-init*:
shows ∀ *w*. *discounted-value r X 0 w* = *X 0 w* ⟨*proof*⟩

lemma *discounted-mult*:
shows ∀ *n w*. *discounted-value r* (λ*m x*. *X m x* ∗ *Y m x*) *n w* = *X n w* ∗ (*discounted-value r Y*) *n w*
⟨*proof*⟩

lemma *discounted-mult'*:
shows *discounted-value r* (λ*m x*. *X m x* ∗ *Y m x*) *n w* = *X n w* ∗ (*discounted-value r Y*) *n w*
⟨*proof*⟩

lemma *discounted-mult-times-rfr*:
assumes −*1* < *r*
shows *discounted-value r* (λ*m w*. (*1*+*r*) ∗ *X w*) (*Suc n*) *w* = *discounted-value r* (λ*m w*. *X w*) *n w*
⟨*proof*⟩

lemma *discounted-cong*:
assumes ∀ *n w*. *X n w* = *Y n w*
shows ∀ *n w*. *discounted-value r X n w* = *discounted-value r Y n w*
⟨*proof*⟩

lemma *discounted-cong'*:
assumes *X n w* = *Y n w*
shows *discounted-value r X n w* = *discounted-value r Y n w*
⟨*proof*⟩

lemma *discounted-AE-cong*:
assumes *AE w in N. X n w = Y n w*
shows *AE w in N. discounted-value r X n w = discounted-value r Y n w*
⟨*proof*⟩

lemma *discounted-sum*:
assumes *finite I*
shows ∀ *n w*. (∑ *i*∈ *I*. (*discounted-value r* (λ*m x*. *f i m x*)) *n w*) = (*discounted-value r* (λ*m x*. (∑ *i*∈ *I*. *f i m x*)) *n w*)
⟨*proof*⟩

lemma *discounted-adapted*:
assumes *borel-adapt-stoch-proc F X*
shows *borel-adapt-stoch-proc F* (*discounted-value r X*) ⟨*proof*⟩

lemma *discounted-measurable*:
assumes *X*∈ *borel-measurable N*
shows *discounted-value r* (λ*m*. *X*) *m* ∈ *borel-measurable N* ⟨*proof*⟩

lemma (**in** *prob-space*) *discounted-integrable*:
assumes *integrable N* (*X n*)
and −*1* < *r*
and *space N* = *space M*
shows *integrable N* (*discounted-value r X n*) ⟨*proof*⟩

7.3.3 Results on risk-neutral probability spaces

definition (**in** *rfr-disc-equity-market*) *risk-neutral-prob* **where**
risk-neutral-prob N ⟷ (*prob-space N*) ∧ (∀ *asset* ∈ *stocks Mkt*. *martingale N F* (*discounted-value r* (*prices Mkt asset*)))

lemma *integrable-val-process*:
assumes ∀ *asset* ∈ *support-set pf*. *integrable M* (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*)
shows *integrable M* (*val-process Mkt pf n*)
⟨*proof*⟩

lemma *integrable-self-fin-uvp*:
assumes ∀ *asset* ∈ *support-set pf*. *integrable M* (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*)
and *self-financing Mkt pf*
shows *integrable M* (*cls-val-process Mkt pf n*)
⟨*proof*⟩

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* ⊆ *stocks Mkt*
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *integrable N* (λ*w*. *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*)
shows ∀ *x* ∈ *support-set pf*. *AE w in N*.
(*real-cond-exp N* (*F n*) (*discounted-value r* (λ*m y*. *prices Mkt x m y* ∗ *pf x m y*) (*Suc n*))) *w* =
discounted-value r (λ*m y*. *prices Mkt x m y* ∗ *pf x* (*Suc m*) *y*) *n w*
⟨*proof*⟩

lemma (**in** *rfr-disc-equity-market*) *self-fin-trad-strat-mart*:
assumes *risk-neutral-prob N*
and *filt-equiv F M N*
and *trading-strategy pf*
and *self-financing Mkt pf*
and *stock-portfolio Mkt pf*
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *integrable N* (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*)
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *integrable N* (λ*w*. *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*)
shows *martingale N F* (*discounted-value r* (*cls-val-process Mkt pf*))
⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *finite-integrable-vp*:
assumes ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*prices Mkt asset n* ‘(*space M*))
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*pf asset n* ‘(*space M*))
and *prob-space N*
and *filt-equiv F M N*
and *trading-strategy pf*
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *prices Mkt asset n* ∈ *borel-measurable M*
shows ∀ *n*. ∀ *asset*∈*support-set pf*. *integrable N* (λ*w*. *prices Mkt asset n w* ∗ *pf asset* (*Suc n*) *w*)
⟨*proof*⟩

lemma (**in** *disc-filtr-prob-space*) *finite-integrable-uvp*:
assumes ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*prices Mkt asset n* ‘(*space M*))
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*pf asset n* ‘(*space M*))
and *prob-space N*
and *filt-equiv F M N*
and *trading-strategy pf*
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *prices Mkt asset n* ∈ *borel-measurable M*

shows ∀ *n*. ∀ *asset*∈*support-set pf*. *integrable N* (*λw*. *prices Mkt asset* (*Suc n*) *w* ∗ *pf asset* (*Suc n*) *w*)
⟨*proof*⟩

lemma (**in** *rfr-disc-equity-market*) *self-fin-trad-strat-mart-finite*:
assumes *risk-neutral-prob N*
and *filt-equiv F M N*
and *trading-strategy pf*
and *self-financing Mkt pf*
and *support-set pf* ⊆ *stocks Mkt*
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*prices Mkt asset n* ‘(*space M*))
and ∀ *n*. ∀ *asset* ∈ *support-set pf*. *finite* (*pf asset n* ‘(*space M*))
and ∀ *asset*∈ *stocks Mkt*. *borel-adapt-stoch-proc F* (*prices Mkt asset*)
shows *martingale N F* (*discounted-value r* (*cls-val-process Mkt pf*))
⟨*proof*⟩

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

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

end

8 The Cox Ross Rubinstein model

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

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

begin

locale *CRR-hyps* = *prob-grw* + *rsk-free-asset* +
  fixes *stk*
assumes *stocks*: *stocks Mkt* = {*stk*, *risk-free-asset*}
  and *stk-price*: *prices Mkt stk* = *geom-proc*
  and *S0-positive*: *0* < *init*
  and *down-positive*: *0* < *d* **and** *down-lt-up*: *d* < *u*
  and *psgt*: *0* < *p*
  and *pslt*: *p* < *1*

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

8.1 Preliminary results on the market

lemma (**in** *CRR-market*) *case-asset*:
  assumes *asset* ∈ *stocks Mkt*
  shows *asset* = *stk* ∨ *asset* = *risk-free-asset*
⟨*proof*⟩

lemma (**in** *CRR-market*)
  assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *bernoulli-gen-filtration*: *filtration N G*
and *bernoulli-sigma-finite*: ∀ *n*. *sigma-finite-subalgebra N* (*G n*)
⟨*proof*⟩

sublocale *CRR-market* ⊆ *rfr-disc-equity-market* - *G*
⟨*proof*⟩

lemma (**in** *CRR-market*) *two-stocks*:
shows *stk* ≠ *risk-free-asset*
⟨*proof*⟩

lemma (**in** *CRR-market*) *stock-pf-vp-expand*:
  assumes *stock-portfolio Mkt pf*
  shows *val-process Mkt pf n w* = *geom-proc n w* ∗ *pf stk* (*Suc n*) *w* +
    disc-rfr-proc r n w ∗ *pf risk-free-asset* (*Suc n*) *w*
⟨*proof*⟩

lemma (**in** *CRR-market*) *stock-pf-uvp-expand*:
  assumes *stock-portfolio Mkt pf*
  shows *cls-val-process Mkt pf* (*Suc n*) *w* = *geom-proc* (*Suc n*) *w* ∗ *pf stk* (*Suc n*) *w* +
    disc-rfr-proc r (*Suc n*) *w* ∗ *pf risk-free-asset* (*Suc n*) *w*
⟨*proof*⟩

lemma (**in** *CRR-market*) *pos-pf-neg-uvp*:
  assumes *stock-portfolio Mkt pf*
  and *d* < *1+r*
  and *0* < *pf stk* (*Suc n*) (*spick w n False*)
  and *val-process Mkt pf n* (*spick w n False*) ≤ *0*
shows *cls-val-process Mkt pf* (*Suc n*) (*spick w n False*) < *0*
⟨*proof*⟩

lemma (**in** *CRR-market*) *neg-pf-neg-uvp*:
  assumes *stock-portfolio Mkt pf*
  and *1+r* < *u*
  and *pf stk* (*Suc n*) (*spick w n True*) < *0*
  and *val-process Mkt pf n* (*spick w n True*) ≤ *0*
shows *cls-val-process Mkt pf* (*Suc n*) (*spick w n True*) < *0*
⟨*proof*⟩

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* ≠ *0*
  and *val-process Mkt pf n w* ≤ *0*
shows *cls-val-process Mkt pf* (*Suc n*) *w* < *0*
⟨*proof*⟩

lemma (**in** *CRR-market*) *neg-pf-exists*:
  assumes *stock-portfolio Mkt pf*

and *trading-strategy pf*
and *1+r < u*
and *d < 1+r*
and *val-process Mkt pf n w ≤ 0*
and *pf stk (Suc n) w ≠ 0 ∨ pf risk-free-asset (Suc n) w ≠ 0*
shows ∃ *y. cls-val-process Mkt pf (Suc n) y < 0*
⟨*proof*⟩

lemma (**in** *CRR-market*) *non-zero-components*:
assumes *val-process Mkt pf n y ≠ 0*
and *stock-portfolio Mkt pf*
shows *pf stk (Suc n) y ≠ 0 ∨ pf risk-free-asset (Suc n) y ≠ 0*
⟨*proof*⟩

lemma (**in** *CRR-market*) *neg-pf-Suc*:
assumes *stock-portfolio Mkt pf*
and *trading-strategy pf*
and *self-financing Mkt pf*
and *1+r < u*
and *d < 1+r*
and *cls-val-process Mkt pf n w < 0*
shows *n ≤ m* ⟹ ∃ *y. cls-val-process Mkt pf m y < 0*
⟨*proof*⟩

lemma (**in** *CRR-market*) *viable-if*:
assumes *1+r < u*
and *d < 1+r*
shows *viable-market Mkt* ⟨*proof*⟩

lemma (**in** *CRR-market*) *viable-only-if-d*:
assumes *viable-market Mkt*
shows *d < 1+r*
⟨*proof*⟩

lemma (**in** *CRR-market*) *viable-only-if-u*:
assumes *viable-market Mkt*
shows *1+r < u*
⟨*proof*⟩

lemma (**in** *CRR-market*) *viable-iff*:
shows *viable-market Mkt* ⟷ (*d < 1+r* ∧ *1+r < u*) ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma (**in** *CRR-market*) *CRR-infinite-cts-filtration*:
shows *infinite-cts-filtration p M nat-filtration*
⟨*proof*⟩

lemma (**in** *CRR-market*) *proj-stoch-proc-geom-disc-fct*:
shows *disc-fct* (*proj-stoch-proc geom-proc n*) ⟨*proof*⟩

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*⟩

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*⟩

lemma (**in** *CRR-market*) *bernoulli-AE-cond-exp*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *integrable N X*
shows *AE w in N*. *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)) *X w* =
expl-cond-expect N (*proj-stoch-proc geom-proc n*) *X w*
⟨*proof*⟩

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*⟩

lemma (**in** *CRR-market*) *expl-cond-eq-sets*:
assumes *N* = *bernoulli-stream q*
shows *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) *X* ∈
borel-measurable (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
⟨*proof*⟩

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*⟩

lemma (**in** *CRR-market*) *geom-proc-real-cond-exp-AE*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *real-cond-exp N* (*fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*))
(*geom-proc* (*Suc n*)) *w* = *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) (*geom-proc* (*Suc n*)) *w*
⟨*proof*⟩

lemma (**in** *CRR-market*) *geom-proc-stoch-proc-filt*:
assumes *N*= *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *stoch-proc-filt N geom-proc borel n* = *fct-gen-subalgebra N* (*stream-space borel*) (*proj-stoch-proc geom-proc n*)
⟨*proof*⟩

lemma (**in** *CRR-market*) *bernoulli-cond-exp*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *integrable N X*
shows *real-cond-exp N* (*stoch-proc-filt N geom-proc borel n*) *X w* = *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) *X w*
⟨*proof*⟩

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*⟩

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* (λ*x*. *discount-factor r n x* ∗ *X x*) *w* = *discount-factor r n w* ∗ (*real-cond-exp M G X*) *w*
⟨*proof*⟩

lemma (**in** *prob-space*) *discounted-value-real-cond-exp*:
assumes *integrable M X*
and −*1* < *r*
and *subalgebra M G*
shows *AE w in M*. *real-cond-exp M G* ((*discounted-value r* (λ *m*. *X*)) *n*) *w* = *discounted-value r* (λ*m*. (*real-cond-exp M G X*)) *n w* ⟨*proof*⟩

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*⟩

lemma (**in** *CRR-market*) *bernoulli-expl-cond-expect-adapt*:
assumes $N =$ *bernoulli-stream q*
and $0 < q$
and $q < 1$
shows *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) *f*∈ *borel-measurable* (*G n*)
⟨*proof*⟩

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 ($\lambda m\ w.$ $(q * u + (1 - q) * d)$ * *prices Mkt stk n w*) (*Suc n*) *w*
⟨*proof*⟩

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*⟩

locale *CRR-market-viable* = *CRR-market* +
assumes *CRR-viable*: *viable-market Mkt*

lemma (**in** *CRR-market-viable*) *real-cond-exp-discount-stock-q-const*:
assumes $N =$ *bernoulli-stream q*
and $q = (1+r-d) / (u-d)$
shows *AE w in N. real-cond-exp N* (*G n*)
(*discounted-value r* (*prices Mkt stk*) (*Suc n*)) *w* =
discounted-value r (*prices Mkt stk*) *n w*
⟨*proof*⟩

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*⟩

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* ⟷ $q= (1 + r - d) / (u - d)$
⟨*proof*⟩

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* ⟷ $q= (1 + r - d) / (u - d)$
⟨*proof*⟩

8.3 Existence of a replicating portfolio

fun (**in** *CRR-market*) *rn-rev-price* **where**
rn-rev-price N der matur 0 w = der w |
rn-rev-price N der matur (*Suc n*) *w = discount-factor r* (*Suc 0*) *w* ∗
expl-cond-expect N (*proj-stoch-proc geom-proc* (*matur* − *Suc n*)) (*rn-rev-price N der matur n*) *w*

lemma (**in** *CRR-market*) *stock-filtration-eq*:
assumes $N = $ *bernoulli-stream q*
and $0 < q$
and $q < 1$
shows *G n = stoch-proc-filt N geom-proc borel n*
⟨*proof*⟩

lemma (**in** *CRR-market*) *real-exp-eq*:
assumes *der*∈ *borel-measurable* (*G matur*)

and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *real-cond-exp N* (*stoch-proc-filt N geom-proc borel n*) *der w* =
expl-cond-expect N (*proj-stoch-proc geom-proc n*) *der w*
⟨*proof*⟩

lemma (**in** *CRR-market*) *rn-rev-price-rev-borel-adapt*:
assumes *cash-flow* ∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows (*n* ≤ *matur*) ⟹ (*rn-rev-price N cash-flow matur n*) ∈ *borel-measurable* (*G* (*matur* − *n*))
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *bernoulli-discounted-integrable*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *der* ∈ *borel-measurable* (*nat-filtration n*)
and −*1* < *r*
shows *integrable N* (*discounted-value r* (λ*m. der*) *m*)
⟨*proof*⟩

lemma (**in** *CRR-market*) *rn-rev-expl-cond-expect*:
assumes *der*∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *n* ≤ *matur* ⟹ *rn-rev-price N der matur n w* =
expl-cond-expect N (*proj-stoch-proc geom-proc* (*matur* − *n*)) (*discounted-value r* (λ*m. der*) *n*) *w*
⟨*proof*⟩

definition (**in** *CRR-market*) *rn-price* **where**
rn-price N der matur n w = *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) (*discounted-value r* (λ*m. der*) (*matur* − *n*)) *w*

definition (**in** *CRR-market*) *rn-price-ind* **where**
rn-price-ind N der matur n w = *rn-rev-price N der matur* (*matur* − *n*) *w*

lemma (**in** *CRR-market*) *rn-price-eq*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*

and *der* ∈ *borel-measurable* (*G matur*)
and *n* ≤ *matur*
shows *rn-price N der matur n w* = *rn-price-ind N der matur n w* ⟨*proof*⟩

lemma (**in** *CRR-market*) *geom-proc-filt-info*:
fixes *f*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *f* ∈ *borel-measurable* (*G n*)
shows *f w* = *f* (*pseudo-proj-True n w*)
⟨*proof*⟩

lemma (**in** *CRR-market*) *geom-proc-filt-info'*:
fixes *f*::*bool stream* ⇒ *'b*::{*t0-space*}
assumes *f* ∈ *borel-measurable* (*G n*)
shows *f w* = *f* (*pseudo-proj-False n w*)
⟨*proof*⟩

lemma (**in** *CRR-market*) *rn-price-borel-adapt*:
assumes *cash-flow* ∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *n* ≤ *matur*
shows (*rn-price N cash-flow matur n*) ∈ *borel-measurable* (*G n*)
⟨*proof*⟩

definition (**in** *CRR-market*) *delta-price* **where**
delta-price N cash-flow T =
(λ *n w. if* (*Suc n* ≤ *T*)
then (*rn-price N cash-flow T* (*Suc n*) (*pseudo-proj-True n w*) − *rn-price N cash-flow T* (*Suc n*) (*pseudo-proj-False n w*))/
(*geom-proc* (*Suc n*) (*spick w n True*) − *geom-proc* (*Suc n*) (*spick w n False*))
else 0)

lemma (**in** *CRR-market*) *delta-price-eq*:
assumes *Suc n* ≤ *T*
shows *delta-price N cash-flow T n w* = (*rn-price N cash-flow T* (*Suc n*) (*spick w n True*) − *rn-price N cash-flow T* (*Suc n*) (*spick w n False*))/
((*geom-proc n w*) ∗ (*u* − *d*))
⟨*proof*⟩

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*⟩

lemma (**in** *CRR-market*) *spick-red-geom*:
shows (λ*w. spick w n x*) ∈ *measurable* (*fct-gen-subalgebra M borel* (*geom-proc n*)) (*fct-gen-subalgebra M borel* (*geom-proc* (*Suc n*)))
⟨*proof*⟩

lemma (**in** *CRR-market*) *geom-spick-Suc*:
assumes *A* ∈ {(*geom-proc* (*Suc n*)) −' *B* |*B. B* ∈ *sets borel*}
shows (λ*w. spick w n x*) −'*A* ∈ {*geom-proc n* −'*B* | *B. B*∈ *sets borel*}
⟨*proof*⟩

lemma (**in** *CRR-market*) *geom-spick-lt*:
assumes *m*< *n*
shows *geom-proc m* (*spick w n x*) = *geom-proc m w*
⟨*proof*⟩

lemma (**in** *CRR-market*) *geom-spick-eq*:
shows *geom-proc m* (*spick w m x*) = *geom-proc m w*
⟨*proof*⟩

lemma (**in** *CRR-market*) *spick-red-geom-filt*:
shows (λ*w. spick w n x*) ∈ *measurable* (*G n*) (*G* (*Suc n*)) ⟨*proof*⟩

lemma (**in** *CRR-market*) *delta-price-adapted*:
fixes *cash-flow*::*bool stream* ⇒ *real*
assumes *cash-flow* ∈ *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*)
⟨*proof*⟩

fun (**in** *CRR-market*) *delta-predict* **where**
delta-predict N der matur 0 = (λ*w. delta-price N der matur 0 w*) |
delta-predict N der matur (*Suc n*) = (λ*w. delta-price N der matur n w*)

lemma (**in** *CRR-market*) *delta-predict-predict*:
assumes *der* ∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *borel-predict-stoch-proc G* (*delta-predict N der matur*) ⟨*proof*⟩

definition (**in** *CRR-market*) *delta-pf* **where**
delta-pf N der matur = *qty-single stk* (*delta-predict N der matur*)

lemma (**in** *CRR-market*) *delta-pf-support*:
shows *support-set* (*delta-pf N der matur*) ⊆ {*stk*} ⟨*proof*⟩

definition (**in** *CRR-market*) *self-fin-delta-pf* **where**
self-fin-delta-pf N der matur v0 = *self-finance Mkt v0* (*delta-pf N der matur*) *risk-free-asset*

lemma (**in** *disc-equity-market*) *self-finance-trading-strat*:
assumes *trading-strategy pf*
and *portfolio pf*
and *borel-adapt-stoch-proc F* (*prices Mkt asset*)
and *support-adapt Mkt pf*
shows *trading-strategy* (*self-finance Mkt v pf asset*) ⟨*proof*⟩

lemma (**in** *CRR-market*) *self-fin-delta-pf-trad-strat*:
assumes *der*∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
shows *trading-strategy* (*self-fin-delta-pf N der matur v0*) ⟨*proof*⟩

definition (**in** *CRR-market*) *delta-hedging* **where**
delta-hedging N der matur = *self-fin-delta-pf N der matur*
(*prob-space.expectation N* (*discounted-value r* (λ*m. der*) *matur*))

lemma (**in** *CRR-market*) *geom-proc-eq-snth*:
shows (⋀*m. m* ≤ *Suc n* ⟹ *geom-proc m x* = *geom-proc m y*) ⟹
(⋀*m. m* ≤ *n* ⟹ *snth x m* = *snth y m*)
⟨*proof*⟩

lemma (**in** *CRR-market*) *geom-proc-eq-pseudo-proj-True*:
shows (⋀*m. m* ≤ *n* ⟹ *geom-proc m x* = *geom-proc m y*) ⟹
(*pseudo-proj-True* (*n*) *x* = *pseudo-proj-True* (*n*) *y*)
⟨*proof*⟩

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*⟩

lemma (**in** *CRR-market-viable*) *rn-rev-price-cond-expect*:
assumes *N* = *bernoulli-stream q*

and *0 <q*
and *q < 1*
and *der* ∈ *borel-measurable* (*G matur*)
and *Suc n* ≤ *matur*
shows *expl-cond-expect N* (*proj-stoch-proc geom-proc n*) (*rn-rev-price N der matur* (*matur* − *Suc n*)) *w*=
(*q* ∗ *rn-rev-price N der matur* (*matur* − *Suc n*) (*pseudo-proj-True n w*) +
(*1* − *q*) ∗ *rn-rev-price N der matur* (*matur* − *Suc n*) (*pseudo-proj-False n w*))
⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *rn-price-eq-ind*:
assumes *N* = *bernoulli-stream q*
and *n* < *matur*
and *0* < *q*
and *q* < *1*
and *der* ∈ *borel-measurable* (*G matur*)
shows (*1*+*r*) ∗ *rn-price N der matur n w* = *q* ∗ *rn-price N der matur* (*Suc n*) (*pseudo-proj-True n w*) +
(*1* − *q*) ∗ *rn-price N der matur* (*Suc n*) (*pseudo-proj-False n w*)
⟨*proof*⟩

lemma *self-finance-updated-suc-suc*:
assumes *portfolio pf*
and ∀ *n*. *prices Mkt asset n w* ≠ *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*⟩

lemma *self-finance-updated-suc-0*:
assumes *portfolio pf*
and ∀ *n w*. *prices Mkt asset n w* ≠ *0*
shows *cls-val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc 0*) *w* = *cls-val-process Mkt pf* (*Suc 0*) *w* +
(*prices Mkt asset* (*Suc 0*) *w* / (*prices Mkt asset 0 w*)) ∗
(*val-process Mkt* (*self-finance Mkt v pf asset*) *0 w* −
val-process Mkt pf 0 w)
⟨*proof*⟩

lemma *self-finance-updated-ind*:
assumes *portfolio pf*
and ∀ *n w*. *prices Mkt asset n w* ≠ *0*

shows *cls-val-process Mkt* (*self-finance Mkt v pf asset*) (*Suc n*) *w* = *cls-val-process Mkt pf* (*Suc n*) *w* +
(*prices Mkt asset* (*Suc n*) *w* / (*prices Mkt asset n w*)) ∗
(*val-process Mkt* (*self-finance Mkt v pf asset*) *n w* −
val-process Mkt pf n w)
⟨*proof*⟩

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*⟩

lemma (**in** *CRR-market*) *delta-pf-portfolio*:
shows *portfolio* (*delta-pf N der matur*) ⟨*proof*⟩

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* ⟨*proof*⟩

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* ⟨*proof*⟩

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*⟩

lemma (**in** *CRR-market-viable*) *delta-hedging-eq-derivative-price*:
fixes *der*::*bool stream* ⇒ *real* **and** *matur*::*nat*
assumes *N* = *bernoulli-stream* ((*1* + *r* − *d*) / (*u* − *d*))
and *der*∈ *borel-measurable* (*G matur*)
shows ⋀*n w*. *n*≤ *matur* ⟹
val-process Mkt (*delta-hedging N der matur*) *n w* =
(*rn-price N der matur*) *n w*

⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *delta-hedging-same-cash-flow*:
assumes *der* ∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream* ((*1* + *r* − *d*) / (*u* − *d*))
shows *cls-val-process Mkt* (*delta-hedging N der matur*) *matur w* =
der w
⟨*proof*⟩

lemma (**in** *CRR-market*) *delta-hedging-trading-strat*:
assumes *N* = *bernoulli-stream q*
and *0* < *q*
and *q* < *1*
and *der* ∈ *borel-measurable* (*G matur*)
shows *trading-strategy* (*delta-hedging N der matur*) ⟨*proof*⟩

lemma (**in** *CRR-market*) *delta-hedging-self-financing*:
shows *self-financing Mkt* (*delta-hedging N der matur*) ⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *delta-hedging-replicating*:
assumes *der* ∈ *borel-measurable* (*G matur*)
and *N* = *bernoulli-stream* ((*1* + *r* − *d*) / (*u* − *d*))
shows *replicating-portfolio* (*delta-hedging N der matur*) *der matur*
⟨*proof*⟩

definition (**in** *disc-equity-market*) *complete-market* **where**
complete-market ⟷ (∀ *matur*. ∀ *der*∈ *borel-measurable* (*F matur*). (∃ *p*. *replicating-portfolio p der matur*))

lemma (**in** *CRR-market-viable*) *CRR-market-complete*:
shows *complete-market* ⟨*proof*⟩

lemma *subalgebras-filtration*:
assumes *filtration M F*
and ∀ *t*. *subalgebra* (*F t*) (*G t*)
and ∀ *s t*. *s* ≤ *t* ⟶ *subalgebra* (*G t*) (*G s*)
shows *filtration M G* ⟨*proof*⟩

lemma *subfilt-filt-equiv*:
assumes *filt-equiv F M N*
and ∀ *t*. *subalgebra* (*F t*) (*G t*)
and ∀ *s t*. *s* ≤ *t* ⟶ *subalgebra* (*G t*) (*G s*)
shows *filt-equiv G M N* ⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *CRR-market-fair-price*:

assumes *pyf* ∈ *borel-measurable* (*G matur*)
shows *fair-price Mkt*
(∑ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component* ((*1* + *r* − *d*) / (*u* − *d*)) *w*) {*0*..<*matur*}) ∗
((*discounted-value r* (λ*m*. *pyf*) *matur*) *w*))
pyf matur
⟨*proof*⟩

end
theory *Option-Price-Examples* **imports** *CRR-Model*

begin

This file contains pricing results for four options in the Cox-Ross-Rubinstein model. The first section contains results relating some functions to the more abstract counterparts that were used to prove fairness and completeness results. The second section contains the pricing results for a few options; some path-dependent and others not.

9 Effective computation definitions and results

9.1 Generation of lists of boolean elements

The function gener-bool-list permits to generate lists of boolean elements. It is used to generate a list representative of the range of boolean streams by the function pseudo-proj-True.

fun *gener-bool-list* **where**
gener-bool-list 0 = {[]}
| *gener-bool-list* (*Suc n*) = {*True* # *w*| *w*. *w*∈ *gener-bool-list n*} ∪ {*False* # *w*| *w*. *w*∈ *gener-bool-list n*}

lemma *gener-bool-list-elem-length*:
shows ⋀*x*. *x*∈ *gener-bool-list n* ⟹ *length x* = *n*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *stake-gener-bool-list*:
shows *stake n*‘*streams* (*UNIV*::*bool set*) = *gener-bool-list n*
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *pseudo-range-stake*:
assumes ⋀*w*. *f w* = *g* (*stake n w*)
shows (∑ *w*∈ *range* (*pseudo-proj-True n*). *f w*) = (∑ *y*∈ (*gener-bool-list n*). *g y*)
⟨*proof*⟩

9.2 Probability components for lists

fun *lprob-comp* **where**

lprob-comp (*p*::*real*) [] = *1*
| *lprob-comp p* (*x* # *xs*) = (*if x then p else* (*1*−*p*)) ∗ *lprob-comp p xs*

lemma *lprob-comp-last*:
shows *lprob-comp p* (*xs* @ [*x*]) = (*lprob-comp p xs*) ∗ (*if x then p else* (*1* − *p*))
⟨*proof*⟩

lemma (**in** *infinite-coin-toss-space*) *lprob-comp-stake*:
shows (*prod* (*prob-component pr w*) {*0*..<*matur*}) = *lprob-comp pr* (*stake matur w*)
⟨*proof*⟩

9.3 Geometric process applied to lists

fun *lrev-geom* **where**
lrev-geom u d v [] = *v*
| *lrev-geom u d v* (*x*#*xs*) = (*if x then u else d*) ∗ *lrev-geom u d v xs*

fun *lgeom-proc* **where** *lgeom-proc u d v l* = *lrev-geom u d v* (*rev l*)

lemma (**in** *infinite-coin-toss-space*) *geom-lgeom*:
shows *geom-rand-walk u d v n w* = *lgeom-proc u d v* (*stake n w*)
⟨*proof*⟩

lemma *lgeom-proc-take*:
assumes *i* ≤ *n*
shows *lgeom-proc u d init* (*stake i w*) = *lgeom-proc u d init* (*take i* (*stake n w*))
⟨*proof*⟩

9.4 Effective computation of discounted values

fun *det-discount* **where**
det-discount (*r*::*real*) *0* = *1*
| *det-discount r* (*Suc n*) = (*inverse* (*1*+*r*)) ∗ (*det-discount r n*)

lemma *det-discounted*:
shows *discounted-value r X n w* = (*det-discount r n*) ∗ (*X n w*) ⟨*proof*⟩

10 Pricing results on options

10.1 Call option

A call option is parameterized by a strike K and maturity T. If S denotes the price of the (unique) risky asset at time T, then the option pays max(S - K, 0) at that time.

definition (**in** *CRR-market*) *call-option* **where**

call-option (*T*::*nat*) (*K*::*real*) = (λ *w*. *max* (*prices Mkt stk T w* − *K*) *0*)

lemma (**in** *CRR-market*) *call-borel*:
shows *call-option T K* ∈ *borel-measurable* (*G T*) ⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *call-option-lgeom*:
shows *call-option T K w* = *max* ((*lgeom-proc u d init* (*stake T w*)) − *K*) *0*
⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *disc-call-option-lgeom*:
shows (*discounted-value r* (λ*m*. (*call-option T K*)) *T w*) =
(*det-discount r T*) ∗ (*max* ((*lgeom-proc u d init* (*stake T w*)) − *K*) *0*)
⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *call-effect-compute*:
shows (∑ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component pr w*) {*0*..<*matur*}) ∗
(*discounted-value r* (λ*m*. (*call-option matur K*)) *matur w*)) =
(∑ *y*∈ (*gener-bool-list matur*). *lprob-comp pr y* ∗ (*det-discount r matur*) ∗
(*max* ((*lgeom-proc u d init* (*take matur y*)) − *K*) *0*))
⟨*proof*⟩

fun *call-price* **where**
call-price u d init r matur K = (∑ *y*∈ (*gener-bool-list matur*). *lprob-comp* ((*1* + *r* − *d*) / (*u* − *d*)) *y* ∗ (*det-discount r matur*) ∗
(*max* ((*lgeom-proc u d init* (*take matur* (*take matur y*))) − *K*) *0*))

Evaluating the function above returns the fair price of a call option.

lemma (**in** *CRR-market-viable*) *call-price*:
shows *fair-price Mkt*
(*call-price u d init r matur K*)
(*call-option matur K*) *matur*
⟨*proof*⟩

10.2 Put option

A put option is also parameterized by a strike K and maturity T. If S denotes the price of the (unique) risky asset at time T, then the option pays max(K - S, 0) at that time.

definition (**in** *CRR-market*) *put-option* **where**
put-option (*T*::*nat*) (*K*::*real*) = (λ *w*. *max* (*K* − *prices Mkt stk T w*) *0*)

lemma (**in** *CRR-market*) *put-borel*:
shows *put-option T K* ∈ *borel-measurable* (*G T*) ⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *put-option-lgeom*:
shows *put-option T K w* = *max* (*K* − (*lgeom-proc u d init* (*stake T w*))) *0*
⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *disc-put-option-lgeom*:
shows (*discounted-value r* (λ*m.* (*put-option T K*)) *T w*) =
(*det-discount r T*) ∗ (*max* (*K* − (*lgeom-proc u d init* (*stake T w*))) *0*)
⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *put-effect-compute*:
shows ($\sum$ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component pr w*) {*0*..<*matur*}) ∗
(*discounted-value r* (λ*m.* (*put-option matur K*)) *matur w*)) =
($\sum$ *y*∈ (*gener-bool-list matur*). *lprob-comp pr y* ∗ (*det-discount r matur*) ∗
(*max* (*K* − (*lgeom-proc u d init* (*take matur y*))) *0*))
⟨*proof*⟩

fun *put-price* **where**
put-price u d init r matur K = ($\sum$ *y*∈ (*gener-bool-list matur*). *lprob-comp* ((*1* + *r* − *d*) / (*u* − *d*)) *y* ∗ (*det-discount r matur*) ∗
(*max* (*K* − (*lgeom-proc u d init* (*take matur* (*take matur y*)))) *0*))

Evaluating the function above returns the fair price of a put option.

lemma (**in** *CRR-market-viable*) *put-price*:
shows *fair-price Mkt*
(*put-price u d init r matur K*)
(*put-option matur K*) *matur*
⟨*proof*⟩

10.3 Lookback option

A lookback option is parameterized by a maturity T. If Sn denotes the price of the (unique) risky asset at time n, then the option pays max(Sn. 0 <= n <= T) - ST at that time.

definition (**in** *CRR-market*) *lbk-option* **where**
lbk-option (*T*::*nat*) = (λ *w. Max* ((λ*i.* (*prices Mkt stk*) *i w*)‘{*0* .. *T*}) − (*prices Mkt stk T w*))

lemma *borel-measurable-Max-finite*:
fixes *f*::′*a* ⇒ ′*b* ⇒ ′*c*::{*second-countable-topology*, *linorder-topology*}
assumes *0* < (*n*::*nat*)
shows ⋀*A. card A* = *n* ⟹ ∀ *a* ∈ *A. f a* ∈ *borel-measurable M* ⟹ (λ*w. Max* ((λ*a. f a w*)‘*A*)) ∈ *borel-measurable M* ⟨*proof*⟩

lemma (**in** *CRR-market*) *lbk-borel*:
shows *lbk-option T* ∈ *borel-measurable* (*G T*) ⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *lbk-option-lgeom*:
shows *lbk-option T w* = *Max* ((λ*i.* (*lgeom-proc u d init* (*stake i w*)))‘{*0* .. *T*}) − (*lgeom-proc u d init* (*stake T w*))

⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *disc-lbk-option-lgeom*:
shows (*discounted-value r* (*λm.* (*lbk-option T*)) *T w*) =
(*det-discount r T*) ∗ (*Max* ((*λi.* (*lgeom-proc u d init* (*take i* (*stake T w*))))‘{*0* .. *T*}) − (*lgeom-proc u d init* (*stake T w*)))
⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *lbk-effect-compute*:
shows ($\sum$ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component pr w*) {*0*..<*matur*}) ∗
(*discounted-value r* (*λm.* (*lbk-option matur*)) *matur w*)) =
($\sum$ *y*∈ (*gener-bool-list matur*). *lprob-comp pr y* ∗ (*det-discount r matur*) ∗
(*Max* ((*λi.* (*lgeom-proc u d init* (*take i y*)))‘{*0* .. *matur*}) − (*lgeom-proc u d init y*)))
⟨*proof*⟩

fun *lbk-price* **where**
lbk-price u d init r matur = ($\sum$ *y*∈ (*gener-bool-list matur*). *lprob-comp* ((*1* + *r* − *d*) / (*u* − *d*)) *y* ∗ (*det-discount r matur*) ∗
(*Max* ((*λi.* (*lgeom-proc u d init* (*take i y*)))‘{*0* .. *matur*}) − (*lgeom-proc u d init y*)))

Evaluating the function above returns the fair price of a lookback option.

lemma (**in** *CRR-market-viable*) *lbk-price*:
shows *fair-price Mkt*
(*lbk-price u d init r matur*)
(*lbk-option matur*) *matur*
⟨*proof*⟩

value *lbk-price 1.2 0.8 10 0.03 2*

10.4 Asian option

An asian option is parameterized by a maturity T. This option pays the average price of the risky asset at time T.

definition (**in** *CRR-market*) *asian-option* **where**
asian-option (*T*::*nat*) = (*λ w.* ($\sum$ *i*∈ {*1*.. *T*}. *prices Mkt stk i w*)/*T*)

lemma (**in** *CRR-market*) *asian-borel*:
shows *asian-option T* ∈ *borel-measurable* (*G T*) ⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *asian-option-lgeom*:
shows *asian-option T w* = ($\sum$ *i*∈ {*1*.. *T*}. *lgeom-proc u d init* (*stake i w*))/ *T*
⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *disc-asian-option-lgeom*:
shows (*discounted-value r* (*λm.* (*asian-option T*)) *T w*) =
(*det-discount r T*) ∗ (∑ *i*∈ {*1*.. *T*}. *lgeom-proc u d init* (*take i* (*stake T w*)))/ *T*
⟨*proof*⟩

lemma (**in** *CRR-market-viable*) *asian-effect-compute*:
shows (∑ *w*∈ *range* (*pseudo-proj-True matur*). (*prod* (*prob-component pr w*) {*0*..<*matur*}) ∗
(*discounted-value r* (*λm.* (*asian-option matur*)) *matur w*)) =
(∑ *y*∈ (*gener-bool-list matur*). *lprob-comp pr y* ∗ (*det-discount r matur*) ∗
(∑ *i*∈ {*1*.. *matur*}. *lgeom-proc u d init* (*take i y*))/ *matur*)
⟨*proof*⟩

fun *asian-price* **where**
asian-price u d init r matur = (∑ *y*∈ (*gener-bool-list matur*). *lprob-comp* ((*1* + *r* − *d*) / (*u* − *d*)) *y* ∗ (*det-discount r matur*) ∗
(∑ *i*∈ {*1*.. *matur*}. *lgeom-proc u d init* (*take i y*))/ *matur*)

Evaluating the function above returns the fair price of an asian option.

lemma (**in** *CRR-market-viable*) *asian-price*:
shows *fair-price Mkt*
(*asian-price u d init r matur*)
(*asian-option matur*) *matur*
⟨*proof*⟩

end