

A Verified Compiler for Probability Density Functions

Manuel Eberl, Johannes Hölzl and Tobias Nipkow

April 10, 2026

Abstract

Bhat *et al.* [1] developed an inductive compiler that computes density functions for probability spaces described by programs in a probabilistic functional language. In this work, we implement such a compiler for a modified version of this language within the theorem prover Isabelle and give a formal proof of its soundness w.r.t. the semantics of the source and target language. Together with Isabelle’s code generation for inductive predicates, this yields a fully verified, executable density compiler. The proof is done in two steps: First, an abstract compiler working with abstract functions modelled directly in the theorem prover’s logic is defined and proved sound. Then, this compiler is refined to a concrete version that returns a target-language expression.

A detailed presentation of this work can be found in the first author’s master’s thesis [2].

Contents

1	Density Predicates	2
1.1	Probability Densities	3
1.2	Measure spaces with densities	3
1.3	Probability spaces with densities	4
1.4	Parametrized probability densities	6
1.5	Density in the Giry monad	7
2	Measure Space Transformations	13
3	Source Language Values	24
3.1	Values and stock measures	24
3.2	Measures on states	31
3.3	Equalities of measure embeddings	31
3.4	Monadic operations on values	32
3.5	Lifting of functions	34

4	Built-in Probability Distributions	41
4.1	Bernoulli	41
4.2	Uniform	41
4.3	Gaussian	43
4.4	Poisson	44
5	Source Language Syntax and Semantics	45
5.1	Expressions	45
5.2	Typing	48
5.3	Semantics	49
5.3.1	Countable types	51
5.4	Semantics	52
5.5	Measurability	56
5.6	Randomfree expressions	60
6	Density Contexts	66
7	Abstract PDF Compiler	91
7.1	Density compiler predicate	91
7.2	Auxiliary lemmas	93
7.3	Soundness proof	105
8	Target Language Syntax and Semantics	128
8.1	General functions on Expressions	135
8.2	Nonnegative expressions	140
8.3	Randomfree expressions	147
8.4	Builtin density expressions	148
8.5	Integral expressions	150
9	Concrete Density Contexts	155
9.1	Definition	155
9.2	Expressions for density context operations	157
10	Concrete PDF Compiler	164
11	Tests	197

1 Density Predicates

```

theory Density-Predicates
imports HOL-Probability.Probability
begin

```

1.1 Probability Densities

definition *is-subprob-density* :: 'a measure \Rightarrow ('a \Rightarrow ennreal) \Rightarrow bool **where**
is-subprob-density M f \equiv (f \in borel-measurable M) \wedge space M \neq {} \wedge
 $(\forall x \in \text{space } M. f\ x \geq 0) \wedge (\int^+ x. f\ x\ \partial M) \leq 1$

lemma *is-subprob-densityI*[intro]:

$\llbracket f \in \text{borel-measurable } M; \bigwedge x. x \in \text{space } M \implies f\ x \geq 0; \text{space } M \neq \{\}; (\int^+ x. f\ x\ \partial M) \leq 1 \rrbracket$

\implies *is-subprob-density* M f

unfolding *is-subprob-density-def* **by** *simp*

lemma *is-subprob-densityD*[dest]:

is-subprob-density M f \implies f \in borel-measurable M

is-subprob-density M f \implies x \in space M \implies f x \geq 0

is-subprob-density M f \implies space M \neq {}

is-subprob-density M f \implies $(\int^+ x. f\ x\ \partial M) \leq 1$

unfolding *is-subprob-density-def* **by** *simp-all*

1.2 Measure spaces with densities

definition *has-density* :: 'a measure \Rightarrow 'a measure \Rightarrow ('a \Rightarrow ennreal) \Rightarrow bool **where**

has-density M N f \longleftrightarrow (f \in borel-measurable N) \wedge space N \neq {} \wedge M = density N f

lemma *has-densityI*[intro]:

$\llbracket f \in \text{borel-measurable } N; M = \text{density } N\ f; \text{space } N \neq \{\} \rrbracket \implies \text{has-density } M\ N\ f$

unfolding *has-density-def* **by** *blast*

lemma *has-densityD*:

assumes *has-density* M N f

shows f \in borel-measurable N M = density N f space N \neq {}

using *assms* **unfolding** *has-density-def* **by** *simp-all*

lemma *has-density-sets*: *has-density* M N f \implies sets M = sets N

unfolding *has-density-def* **by** *simp*

lemma *has-density-space*: *has-density* M N f \implies space M = space N

unfolding *has-density-def* **by** *simp*

lemma *has-density-emeasure*:

has-density M N f \implies X \in sets M \implies emeasure M X = $\int^+ x. f\ x * \text{indicator } X\ x\ \partial N$

unfolding *has-density-def* **by** (*simp-all* add: *emeasure-density*)

lemma *nn-integral-cong'*: $(\bigwedge x. x \in \text{space } N = \text{simp} \implies f\ x = g\ x) \implies (\int^+ x. f\ x\ \partial N) = (\int^+ x. g\ x\ \partial N)$

by (*simp* add: *simp-implies-def* cong: *nn-integral-cong*)

lemma *has-density-emeasure-space*:

has-density $M N f \implies \text{emeasure } M (\text{space } M) = (\int^{+x}. f x \partial N)$

by (*simp add: has-density-emeasure*) (*simp add: has-density-space cong: nn-integral-cong'*)

lemma *has-density-emeasure-space'*:

has-density $M N f \implies \text{emeasure } (\text{density } N f) (\text{space } (\text{density } N f)) = \int^{+x}. f x \partial N$

by (*frule has-densityD(2)[symmetric]*) (*simp add: has-density-emeasure-space*)

lemma *has-density-imp-is-subprob-density*:

$\llbracket \text{has-density } M N f; (\int^{+x}. f x \partial N) = 1 \rrbracket \implies \text{is-subprob-density } N f$

by (*auto dest: has-densityD*)

lemma *has-density-imp-is-subprob-density'*:

$\llbracket \text{has-density } M N f; \text{prob-space } M \rrbracket \implies \text{is-subprob-density } N f$

by (*auto intro!: has-density-imp-is-subprob-density dest: prob-space.emeasure-space-1 simp: has-density-emeasure-space*)

lemma *has-density-equal-on-space*:

assumes *has-density* $M N f \wedge x. x \in \text{space } N \implies f x = g x$

shows *has-density* $M N g$

proof

from *assms* **show** $g \in \text{borel-measurable } N$

by (*subst measurable-cong[of - - f]*) (*auto dest: has-densityD*)

with *assms* **show** $M = \text{density } N g$

by (*subst density-cong[of - - f]*) (*auto dest: has-densityD*)

from *assms(1)* **show** $\text{space } N \neq \{\}$ **by** (*rule has-densityD*)

qed

lemma *has-density-cong*:

assumes $\wedge x. x \in \text{space } N \implies f x = g x$

shows *has-density* $M N f = \text{has-density } M N g$

using *assms* **by** (*intro iffI*) (*erule has-density-equal-on-space, simp*)⁺

lemma *has-density-dens-AE*:

$\llbracket \text{AE } y \text{ in } N. f y = f' y; f' \in \text{borel-measurable } N;$

$\wedge x. x \in \text{space } M \implies f' x \geq 0; \text{has-density } M N f \rrbracket$

$\implies \text{has-density } M N f'$

unfolding *has-density-def* **by** (*simp cong: density-cong*)

1.3 Probability spaces with densities

lemma *is-subprob-density-imp-has-density*:

$\llbracket \text{is-subprob-density } N f; M = \text{density } N f \rrbracket \implies \text{has-density } M N f$

by (*rule has-densityI*) *auto*

lemma *has-subprob-density-imp-subprob-space'*:

$\llbracket \text{has-density } M N f; \text{is-subprob-density } N f \rrbracket \implies \text{subprob-space } M$

proof (rule subprob-spaceI)
assume *has-density* $M N f$
hence $M = \text{density } N f$ **by** (simp add: *has-density-def*)
also from $\langle \text{has-density } M N f \rangle$ **have** $\text{space } \dots \neq \{\}$ **by** (simp add: *has-density-def*)
finally show $\text{space } M \neq \{\}$.
qed (auto simp add: *has-density-emeasure-space dest: has-densityD*)

lemma *has-subprob-density-imp-subprob-space[dest]*:
is-subprob-density $M f \implies \text{subprob-space } (\text{density } M f)$
by (rule *has-subprob-density-imp-subprob-space'*) auto

definition *has-subprob-density* $M N f \equiv \text{has-density } M N f \wedge \text{subprob-space } M$

lemma *subprob-space-density-not-empty*: $\text{subprob-space } (\text{density } M f) \implies \text{space } M \neq \{\}$
by (subst *space-density[symmetric]*, subst *subprob-space.subprob-not-empty*, assumption) simp

lemma *has-subprob-densityI*:
 $\llbracket f \in \text{borel-measurable } N; M = \text{density } N f; \text{subprob-space } M \rrbracket \implies \text{has-subprob-density } M N f$
unfolding *has-subprob-density-def* **by** (auto simp: *subprob-space-density-not-empty*)

lemma *has-subprob-densityI'*:
assumes $f \in \text{borel-measurable } N$ $\text{space } N \neq \{\}$
 $M = \text{density } N f$ $(\int^+ x. f x \partial N) \leq 1$
shows *has-subprob-density* $M N f$
proof –
from *assms* **have** D : *has-density* $M N f$ **by** blast
moreover from D **and** *assms* **have** *subprob-space* M
by (auto intro!: *subprob-spaceI simp: has-density-emeasure-space emeasure-density cong: nn-integral-cong'*)
ultimately show *thesis* **unfolding** *has-subprob-density-def* **by** simp
qed

lemma *has-subprob-densityD*:
assumes *has-subprob-density* $M N f$
shows $f \in \text{borel-measurable } N \wedge x. x \in \text{space } N \implies f x \geq 0$ $M = \text{density } N f$
subprob-space M
using *assms* **unfolding** *has-subprob-density-def* **by** (auto dest: *has-densityD*)

lemma *has-subprob-density-measurable[measurable-dest]*:
has-subprob-density $M N f \implies f \in N \rightarrow_M \text{borel}$
by (auto dest: *has-subprob-densityD*)

lemma *has-subprob-density-imp-has-density*:
has-subprob-density $M N f \implies \text{has-density } M N f$ **by** (simp add: *has-subprob-density-def*)

lemma *has-subprob-density-equal-on-space*:
assumes *has-subprob-density* $M N f \wedge x. x \in \text{space } N \implies f x = g x$
shows *has-subprob-density* $M N g$
using *assms unfolding has-subprob-density-def* **by** (*auto dest: has-density-equal-on-space*)

lemma *has-subprob-density-cong*:
assumes $\wedge x. x \in \text{space } N \implies f x = g x$
shows *has-subprob-density* $M N f = \text{has-subprob-density } M N g$
using *assms* **by** (*intro iffI*) (*erule has-subprob-density-equal-on-space, simp*)⁺

lemma *has-subprob-density-dens-AE*:
 $\llbracket AE y \text{ in } N. f y = f' y; f' \in \text{borel-measurable } N;$
 $\wedge x. x \in \text{space } M \implies f' x \geq 0; \text{has-subprob-density } M N f \rrbracket$
 $\implies \text{has-subprob-density } M N f'$
unfolding *has-subprob-density-def* **by** (*simp add: has-density-dens-AE*)

1.4 Parametrized probability densities

definition

has-parametrized-subprob-density $M N R f \equiv$
 $(\forall x \in \text{space } M. \text{has-subprob-density } (N x) R (f x)) \wedge \text{case-prod } f \in$
borel-measurable $(M \otimes_M R)$

lemma *has-parametrized-subprob-densityI*:
assumes $\wedge x. x \in \text{space } M \implies N x = \text{density } R (f x)$
assumes $\wedge x. x \in \text{space } M \implies \text{subprob-space } (N x)$
assumes *case-prod* $f \in \text{borel-measurable } (M \otimes_M R)$
shows *has-parametrized-subprob-density* $M N R f$
unfolding *has-parametrized-subprob-density-def* **using** *assms*
by (*intro ballI conjI has-subprob-densityI*) *simp-all*

lemma *has-parametrized-subprob-densityD*:
assumes *has-parametrized-subprob-density* $M N R f$
shows $\wedge x. x \in \text{space } M \implies N x = \text{density } R (f x)$
and $\wedge x. x \in \text{space } M \implies \text{subprob-space } (N x)$
and [*measurable-dest*]: *case-prod* $f \in \text{borel-measurable } (M \otimes_M R)$
using *assms unfolding has-parametrized-subprob-density-def*
by (*auto dest: has-subprob-densityD*)

lemma *has-parametrized-subprob-density-integral*:
assumes *has-parametrized-subprob-density* $M N R f x \in \text{space } M$
shows $(\int^+ y. f x y \partial R) \leq 1$

proof –

have $(\int^+ y. f x y \partial R) = \text{emeasure } (\text{density } R (f x)) (\text{space } (\text{density } R (f x)))$
using *assms*

by (*auto simp: emeasure-density cong: nn-integral-cong' dest: has-parametrized-subprob-densityD*)
also have $\text{density } R (f x) = (N x)$ **using** *assms* **by** (*auto dest: has-parametrized-subprob-densityD*)
also have *emeasure* ... (*space* ...) ≤ 1 **using** *assms*
by (*subst subprob-space.emeasure-space-le-1*) (*auto dest: has-parametrized-subprob-densityD*)

finally show *?thesis* .
qed

lemma *has-parametrized-subprob-density-cong*:
assumes $\bigwedge x. x \in \text{space } M \implies N x = N' x$
shows *has-parametrized-subprob-density* $M N R f = \text{has-parametrized-subprob-density}$
 $M N' R f$
using *assms unfolding has-parametrized-subprob-density-def* **by** *auto*

lemma *has-parametrized-subprob-density-dens-AE*:
assumes $\bigwedge x. x \in \text{space } M \implies AE y \text{ in } R. f x y = f' x y$
case-prod $f' \in \text{borel-measurable } (M \otimes_M R)$
has-parametrized-subprob-density $M N R f$
shows *has-parametrized-subprob-density* $M N R f'$
unfolding *has-parametrized-subprob-density-def*
proof (*intro conjI ballI*)
fix x **assume** $x: x \in \text{space } M$
with *assms(3)* **have** $\text{space } (N x) = \text{space } R$
by (*auto dest!: has-parametrized-subprob-densityD(1)*)
with *assms* **and** x **show** *has-subprob-density* $(N x) R (f' x)$
by (*rule-tac has-subprob-density-dens-AE[of f x]*)
(*auto simp: has-parametrized-subprob-density-def*)
qed *fact*

1.5 Density in the Giry monad

lemma *emeasure-bind-density*:
assumes $\text{space } M \neq \{\}$ $\bigwedge x. x \in \text{space } M \implies \text{has-density } (f x) N (g x)$
 $f \in \text{measurable } M (\text{subprob-algebra } N) X \in \text{sets } N$
shows *emeasure* $(M \gg f) X = \int^+ x. \int^+ y. g x y * \text{indicator } X y \partial N \partial M$
proof –
from *assms* **have** *emeasure* $(M \gg f) X = \int^+ x. \text{emeasure } (f x) X \partial M$
by (*intro emeasure-bind*)
also **have** $\dots = \int^+ x. \int^+ y. g x y * \text{indicator } X y \partial N \partial M$ **using** *assms*
by (*intro nn-integral-cong*) (*simp add: has-density-emeasure sets-kernel*)
finally show *?thesis* .
qed

lemma *bind-density*:
assumes *sigma-finite-measure* M *sigma-finite-measure* N
 $\text{space } M \neq \{\}$ $\bigwedge x. x \in \text{space } M \implies \text{has-density } (f x) N (g x)$
and [*measurable*]: *case-prod* $g \in \text{borel-measurable } (M \otimes_M N) f \in \text{measurable}$
 $M (\text{subprob-algebra } N)$
shows $(M \gg f) = \text{density } N (\lambda y. \int^+ x. g x y \partial M)$
proof (*rule measure-eqI*)
interpret *sfN*: *sigma-finite-measure* N **by** *fact*
interpret *sfNM*: *pair-sigma-finite* $N M$ **unfolding** *pair-sigma-finite-def* **using**
assms **by** *simp*
show *eq: sets* $(M \gg f) = \text{sets } (\text{density } N (\lambda y. \int^+ x. g x y \partial M))$

using *sets-bind*[*OF sets-kernel*[*OF assms*(6)] *assms*(3)] **by** *auto*
fix *X* **assume** $X \in \text{sets } (M \ggg f)$
with *eq* **have** [*measurable*]: $X \in \text{sets } N$ **by** *auto*
with *assms* **have** *emeasure* $(M \ggg f) X = \int^{+x}. \int^{+y}. g \ x \ y * \text{indicator } X \ y$
 $\partial N \ \partial M$
by (*intro emeasure-bind-density*) *simp-all*
also from $\langle X \in \text{sets } N \rangle$ **have** $\dots = \int^{+y}. \int^{+x}. g \ x \ y * \text{indicator } X \ y \ \partial M \ \partial N$
by (*intro sfNM.Fubini'*) *measurable*
also {
fix *y* **assume** $y \in \text{space } N$
have $(\lambda x. g \ x \ y) = \text{case-prod } g \circ (\lambda x. (x, y))$ **by** (*rule ext*) *simp*
also from $\langle y \in \text{space } N \rangle$ **have** $\dots \in \text{borel-measurable } M$
by (*intro measurable-comp*[*OF - assms*(5)] *measurable-Pair2'*)
finally have $(\lambda x. g \ x \ y) \in \text{borel-measurable } M$.
}
hence $\dots = \int^{+y}. (\int^{+x}. g \ x \ y \ \partial M) * \text{indicator } X \ y \ \partial N$
by (*intro nn-integral-cong nn-integral-multc*) *simp-all*
also from $\langle X \in \text{sets } N \rangle$ **and** *assms* **have** $\dots = \text{emeasure } (\text{density } N \ (\lambda y. \int^{+x}. g \ x \ y \ \partial M)) \ X$
by (*subst emeasure-density*) (*simp-all add: sfN.borel-measurable-nn-integral*)
finally show *emeasure* $(M \ggg f) X = \text{emeasure } (\text{density } N \ (\lambda y. \int^{+x}. g \ x \ y \ \partial M)) \ X$.
qed

lemma *bind-has-density*:

assumes *sigma-finite-measure* *M* *sigma-finite-measure* *N*
 $\text{space } M \neq \{\}$ $\bigwedge x. x \in \text{space } M \implies \text{has-density } (f \ x) \ N \ (g \ x)$
 $\text{case-prod } g \in \text{borel-measurable } (M \otimes_M N)$
 $f \in \text{measurable } M \ (\text{subprob-algebra } N)$
shows *has-density* $(M \ggg f) \ N \ (\lambda y. \int^{+x}. g \ x \ y \ \partial M)$
proof
interpret *sigma-finite-measure* *M* **by** *fact*
show $(\lambda y. \int^{+x}. g \ x \ y \ \partial M) \in \text{borel-measurable } N$ **using** *assms*
by (*intro borel-measurable-nn-integral, subst measurable-pair-swap-iff*) *simp*
show $M \ggg f = \text{density } N \ (\lambda y. \int^{+x}. g \ x \ y \ \partial M)$
by (*intro bind-density*) (*simp-all add: assms*)
from $\langle \text{space } M \neq \{\} \rangle$ **obtain** *x* **where** $x \in \text{space } M$ **by** *blast*
with *assms* **have** *has-density* $(f \ x) \ N \ (g \ x)$ **by** *simp*
thus $\text{space } N \neq \{\}$ **by** (*rule has-densityD*)
qed

lemma *bind-has-density'*:

assumes *sfM*: *sigma-finite-measure* *M*
and *sfR*: *sigma-finite-measure* *R*
and *not-empty*: $\text{space } M \neq \{\}$ **and** *dens-M*: *has-density* *M* *N* δM
and *dens-f*: $\bigwedge x. x \in \text{space } M \implies \text{has-density } (f \ x) \ R \ (\delta f \ x)$
and *Mδf*: $\text{case-prod } \delta f \in \text{borel-measurable } (N \otimes_M R)$
and *Mf*: $f \in \text{measurable } N \ (\text{subprob-algebra } R)$

shows *has-density* $(M \ggg f) R (\lambda y. \int^+ x. \delta M x * \delta f x y \partial N)$
proof –
from *dens-M* **have** $M-M: \text{measurable } M = \text{measurable } N$
by (*intro ext measurable-cong-sets*) (*auto dest: has-densityD*)
from *dens-M* **have** $M-MR: \text{measurable } (M \otimes_M R) = \text{measurable } (N \otimes_M R)$
by (*intro ext measurable-cong-sets sets-pair-measure-cong*) (*auto dest: has-densityD*)
have *has-density* $(M \ggg f) R (\lambda y. \int^+ x. \delta f x y \partial M)$
by (*rule bind-has-density*) (*auto simp: assms M-MR M-M*)
moreover {
fix y **assume** $A: y \in \text{space } R$
have $(\lambda x. \delta f x y) = \text{case-prod } \delta f \circ (\lambda x. (x, y))$ **by** (*rule ext*) (*simp add: o-def*)
also have $\dots \in \text{borel-measurable } N$ **by** (*intro measurable-comp[OF - Mδf]*
measurable-Pair2' A)
finally have $M\text{-}\delta f': (\lambda x. \delta f x y) \in \text{borel-measurable } N$.

from *dens-M* **have** $M = \text{density } N \delta M$ **by** (*auto dest: has-densityD*)
also from *dens-M* **have** $(\int^+ x. \delta f x y \partial \dots) = \int^+ x. \delta M x * \delta f x y \partial N$
by (*subst nn-integral-density*) (*auto dest: has-densityD simp: M-δf'*)
finally have $(\int^+ x. \delta f x y \partial M) = \int^+ x. \delta M x * \delta f x y \partial N$.
}
ultimately show *has-density* $(M \ggg f) R (\lambda y. \int^+ x. \delta M x * \delta f x y \partial N)$
by (*rule has-density-equal-on-space*) *simp-all*
qed

lemma *bind-has-subprob-density*:
assumes *subprob-space* M *sigma-finite-measure* N
 $\text{space } M \neq \{\}$ $\bigwedge x. x \in \text{space } M \implies \text{has-density } (f x) N (g x)$
 $\text{case-prod } g \in \text{borel-measurable } (M \otimes_M N)$
 $f \in \text{measurable } M$ (*subprob-algebra* N)
shows *has-subprob-density* $(M \ggg f) N (\lambda y. \int^+ x. g x y \partial M)$
proof (*unfold has-subprob-density-def, intro conjI*)
from *assms* **show** *has-density* $(M \ggg f) N (\lambda y. \int^+ x. g x y \partial M)$
by (*intro bind-has-density*) (*auto simp: subprob-space-imp-sigma-finite*)
from *assms* **show** *subprob-space* $(M \ggg f)$ **by** (*intro subprob-space-bind*)
qed

lemma *bind-has-subprob-density'*:
assumes *has-subprob-density* $M N \delta M$ *space* $R \neq \{\}$ *sigma-finite-measure* R
 $\bigwedge x. x \in \text{space } M \implies \text{has-subprob-density } (f x) R (\delta f x)$
 $\text{case-prod } \delta f \in \text{borel-measurable } (N \otimes_M R)$ $f \in \text{measurable } N$ (*subprob-algebra*
 R)
shows *has-subprob-density* $(M \ggg f) R (\lambda y. \int^+ x. \delta M x * \delta f x y \partial N)$
proof (*unfold has-subprob-density-def, intro conjI*)
from *assms*(1) **have** *space* $M \neq \{\}$ **by** (*intro subprob-space.subprob-not-empty*
has-subprob-densityD)
with *assms* **show** *has-density* $(M \ggg f) R (\lambda y. \int^+ x. \delta M x * \delta f x y \partial N)$
by (*intro bind-has-density' has-densityI*)
(*auto simp: subprob-space-imp-sigma-finite dest: has-subprob-densityD*)
from *assms* **show** *subprob-space* $(M \ggg f)$

by (intro subprob-space-bind) (auto dest: has-subprob-densityD)
qed

lemma null-measure-has-subprob-density:

space $M \neq \{\}$ \implies has-subprob-density (null-measure M) M (λ -. 0)

by (intro has-subprob-densityI)

(auto intro: null-measure-eq-density simp: subprob-space-null-measure-iff)

lemma emeasure-has-parametrized-subprob-density:

assumes has-parametrized-subprob-density M N R f

assumes $x \in$ space M $X \in$ sets R

shows emeasure (N x) $X = \int^+ y. f$ x y * indicator X y ∂R

proof –

from has-parametrized-subprob-densityD(3)[OF assms(1)] and assms(2)

have Mf : f $x \in$ borel-measurable R by simp

have N $x =$ density R (f x)

by (rule has-parametrized-subprob-densityD(1)[OF assms(1,2)])

also from Mf and assms(3) have emeasure ... $X = \int^+ y. f$ x y * indicator X y ∂R

by (rule emeasure-density)

finally show ?thesis .

qed

lemma emeasure-count-space-density-singleton:

assumes $x \in A$ has-density M (count-space A) f

shows emeasure M $\{x\} = f$ x

proof –

from has-densityD[OF assms(2)] have nonneg: $\bigwedge x. x \in A \implies f$ $x \geq 0$ by simp

from assms have M : $M =$ density (count-space A) f by (intro has-densityD)

from assms have emeasure M $\{x\} = \int^+ y. f$ y * indicator $\{x\}$ y ∂ count-space A

by (simp add: M emeasure-density)

also from assms and nonneg have ... = f x

by (subst nn-integral-indicator-singleton) auto

finally show ?thesis .

qed

lemma subprob-count-space-density-le-1:

assumes has-subprob-density M (count-space A) f $x \in A$

shows f $x \leq 1$

proof (cases f $x > 0$)

assume f $x > 0$

from assms interpret subprob-space M by (intro has-subprob-densityD)

from assms have M : $M =$ density (count-space A) f by (intro has-subprob-densityD)

from assms have f $x =$ emeasure M $\{x\}$

by (intro emeasure-count-space-density-singleton[symmetric])

(auto simp: has-subprob-density-def)

also have ... ≤ 1 by (rule subprob-emeasure-le-1)

finally show ?thesis .

qed (*auto simp: not-less intro: order.trans[of - 0 1]*)

lemma *has-density-embed-measure*:

assumes *inj*: $\text{inj } f$ **and** *inv*: $\bigwedge x. x \in \text{space } N \implies f'(f x) = x$

shows *has-density* (*embed-measure* $M f$) (*embed-measure* $N f$) ($\delta \circ f'$) \longleftrightarrow
has-density $M N \delta$

(**is** *has-density* $?M' ?N' ?\delta' \longleftrightarrow$ *has-density* $M N \delta$)

proof

assume *dens*: *has-density* $?M' ?N' ?\delta'$

show *has-density* $M N \delta$

proof

from *dens* **show** $\text{space } N \neq \{\}$ **by** (*auto simp: space-embed-measure dest: has-densityD*)

from *dens* **have** $M\delta f'$: $\delta \circ f' \in \text{borel-measurable } ?N'$ **by** (*rule has-densityD*)

hence $M\delta f'f$: $\delta \circ f' \circ f \in \text{borel-measurable } N$

by (*rule-tac measurable-comp, rule-tac measurable-embed-measure2[OF inj]*)

thus $M\delta$: $\delta \in \text{borel-measurable } N$ **by** (*simp cong: measurable-cong add: inv*)

from *dens* **have** $\text{embed-measure } M f = \text{density } (\text{embed-measure } N f) (\delta \circ f')$

by (*rule has-densityD*)

also **have** $\dots = \text{embed-measure } (\text{density } N (\delta \circ f' \circ f)) f$

by (*simp only: density-embed-measure[OF inj M\delta f']*)

also **have** $\text{density } N (\delta \circ f' \circ f) = \text{density } N \delta$

by (*intro density-cong[OF M\delta f'f M\delta]*) (*simp-all add: inv*)

finally **show** $M = \text{density } N \delta$ **by** (*simp add: embed-measure-eq-iff[OF inj]*)

qed

next

assume *dens*: *has-density* $M N \delta$

show *has-density* $?M' ?N' ?\delta'$

proof

from *dens* **show** $\text{space } ?N' \neq \{\}$ **by** (*auto simp: space-embed-measure dest: has-densityD*)

have $Mf'f$: $(\lambda x. f'(f x)) \in \text{measurable } N N$ **by** (*subst measurable-cong[OF inv] simp-all*)

from *dens* **have** $M\delta$: $\delta \in \text{borel-measurable } N$ **by** (*auto dest: has-densityD*)

from $Mf'f$ **and** *dens* **show** $M\delta f'$: $\delta \circ f' \in \text{borel-measurable } (\text{embed-measure } N f)$

by (*intro measurable-comp*) (*erule measurable-embed-measure1, rule has-densityD*)

have $\text{embed-measure } M f = \text{embed-measure } (\text{density } N \delta) f$

by (*simp only: has-densityD[OF dens]*)

also **from** *inv* **and** *dens* **and** *measurable-comp*[*OF Mf'f M\delta*]

have $\text{density } N \delta = \text{density } N (?'\delta' \circ f)$

by (*intro density-cong[OF M\delta]*) (*simp add: o-def, simp add: inv o-def*)

also **have** $\text{embed-measure } (\text{density } N (?'\delta' \circ f)) f = \text{density } (\text{embed-measure } N f) (\delta \circ f')$

by (*simp only: density-embed-measure[OF inj M\delta f', symmetric]*)

finally **show** $\text{embed-measure } M f = \text{density } (\text{embed-measure } N f) (\delta \circ f')$.

qed

qed

lemma *has-density-embed-measure'*:
assumes *inj*: $inj\ f$ **and** *inv*: $\bigwedge x. x \in space\ N \implies f'\ (f\ x) = x$ **and**
sets-M: $sets\ M = sets\ (embed-measure\ N\ f)$
shows $has-density\ (distr\ M\ N\ f')\ N\ (\delta \circ f) \longleftrightarrow has-density\ M\ (embed-measure\ N\ f)\ \delta$
proof –
have *sets'*: $sets\ (embed-measure\ (distr\ M\ N\ f')\ f) = sets\ (embed-measure\ N\ f)$
by (*simp* *add*: *sets-embed-measure*[*OF inj*])
have *Mff'*: $(\lambda x. f'\ (f\ x)) \in measurable\ N\ N$ **by** (*subst* *measurable-cong*[*OF inv*])
simp-all
have *inv'*: $\bigwedge x. x \in space\ M \implies f\ (f'\ x) = x$
by (*subst* (*asm*) *sets-eq-imp-space-eq*[*OF sets-M*]) (*auto* *simp*: *space-embed-measure*
inv)
have $M = distr\ M\ (embed-measure\ (distr\ M\ N\ f')\ f)\ (\lambda x. f\ (f'\ x))$
by (*subst* *distr-cong*[*OF refl - inv'*, *of - M*]) (*simp-all* *add*: *sets-embed-measure*
inj sets-M)
also have $\dots = embed-measure\ (distr\ M\ N\ f')\ f$
apply (*subst* (*2*) *embed-measure-eq-distr*[*OF inj*], *subst* *distr-distr*)
apply (*subst* *measurable-cong-sets*[*OF refl sets'*], *rule* *measurable-embed-measure2*[*OF*
inj])
apply (*subst* *measurable-cong-sets*[*OF sets-M refl*], *rule* *measurable-embed-measure1*,
rule *Mff'*)
apply (*simp* *cong*: *distr-cong* *add*: *inv*)
done
finally have $M: M = embed-measure\ (distr\ M\ N\ f')\ f$.
show *?thesis* **by** (*subst* (*2*) *M*, *subst* *has-density-embed-measure*[*OF inj inv*,
symmetric])
(*auto* *simp*: *space-embed-measure* *inv* *intro!*: *has-density-cong*)
qed

lemma *has-density-embed-measure''*:
assumes *inj*: $inj\ f$ **and** *inv*: $\bigwedge x. x \in space\ N \implies f'\ (f\ x) = x$ **and**
has-density M ($embed-measure\ N\ f$) δ
shows $has-density\ (distr\ M\ N\ f')\ N\ (\delta \circ f)$
proof (*subst* *has-density-embed-measure'*)
from *assms*(*3*) **show** $sets\ M = sets\ (embed-measure\ N\ f)$ **by** (*auto* *dest*: *has-densityD*)
qed (*insert* *assms*)

lemma *has-subprob-density-embed-measure''*:
assumes *inj*: $inj\ f$ **and** *inv*: $\bigwedge x. x \in space\ N \implies f'\ (f\ x) = x$ **and**
has-subprob-density M ($embed-measure\ N\ f$) δ
shows $has-subprob-density\ (distr\ M\ N\ f')\ N\ (\delta \circ f)$
proof (*unfold* *has-subprob-density-def*, *intro* *conjI*)
from *assms* **show** $has-density\ (distr\ M\ N\ f')\ N\ (\delta \circ f)$
by (*intro* *has-density-embed-measure''* *has-subprob-density-imp-has-density*)
from *assms*(*3*) **have** $sets\ M = sets\ (embed-measure\ N\ f)$ **by** (*auto* *dest*: *has-subprob-densityD*)
hence $M: measurable\ M = measurable\ (embed-measure\ N\ f)$
by (*intro* *ext* *measurable-cong-sets*) *simp-all*
have $(\lambda x. f'\ (f\ x)) \in measurable\ N\ N$ **by** (*simp* *cong*: *measurable-cong* *add*: *inv*)

moreover from *assms* **have** *space (embed-measure N f) ≠ {}*
unfolding *has-subprob-density-def has-density-def* **by** *simp*
ultimately show *subprob-space (distr M N f')* **using** *assms*
by (*intro subprob-space.subprob-space-distr has-subprob-densityD*)
(auto simp: M space-embed-measure intro!: measurable-embed-measure1 dest:
has-subprob-densityD)
qed (*insert assms*)

end

2 Measure Space Transformations

theory *PDF-Transformations*
imports *Density-Predicates*
begin

lemma *not-top-le-1-ennreal[simp]*: $\neg \text{top} \leq (1::\text{ennreal})$
by (*simp add: top-unique*)

lemma *range-int*: $\text{range int} = \{n. n \geq 0\}$

proof (*intro equalityI subsetI*)
fix $n :: \text{int}$ **assume** $n \in \{n. n \geq 0\}$
hence $n = \text{int} (\text{nat } n)$ **by** *simp*
thus $n \in \text{range int}$ **by** *blast*
qed *auto*

lemma *range-exp*: $\text{range} (\text{exp} :: \text{real} \Rightarrow \text{real}) = \{x. x > 0\}$

proof (*intro equalityI subsetI*)
fix $x :: \text{real}$ **assume** $x \in \{x. x > 0\}$
hence $x = \text{exp} (\ln x)$ **by** *simp*
thus $x \in \text{range exp}$ **by** *blast*
qed *auto*

lemma *Int-stable-Icc*: $\text{Int-stable} (\text{range} (\lambda(a, b). \{a .. b::\text{real}\}))$
by (*auto simp: Int-stable-def*)

lemma *distr-mult-real*:

assumes $c \neq 0$ *has-density M lborel (f :: real \Rightarrow ennreal)*
shows *has-density (distr M borel ((* c)) lborel ($\lambda x. f (x / c) * \text{inverse} (\text{abs } c)$))*
(is has-density ?M' - ?f')

proof

from *assms(2)* **have** $M = \text{density lborel } f$ **by** (*rule has-densityD*)
also from *assms* **have** $Mf[\text{measurable}]$: $f \in \text{borel-measurable borel}$
by (*auto dest: has-densityD*)
hence $\text{distr} (\text{density lborel } f) \text{ borel} ((* c) = \text{density lborel } ?f'$ **(is ?M1 = ?M2)**
proof (*intro measure-eqI*)
fix X **assume** $X[\text{measurable}]$: $X \in \text{sets} (\text{distr} (\text{density lborel } f) \text{ borel} ((* c))$
with *assms* **have** $\text{emeasure } ?M1 X = \int^{+x}. f x * \text{indicator } X (c * x) \partial \text{lborel}$
by (*subst emeasure-distr, simp, simp, subst emeasure-density*)

(auto dest: has-densityD intro!: measurable-sets-borel nn-integral-cong
 split: split-indicator)
also from *assms*(1) **and** *X* **have** ... = $\int^{+x}. ?f' x * \text{indicator } X x \partial \text{lborel}$
apply (subst lborel-distr-mult[of inverse c])
apply *simp*
apply (subst nn-integral-density)
apply (simp-all add: nn-integral-distr field-simps)
done
also from *X* **have** ... = *emeasure* ?M2 *X*
by (subst emeasure-density) *auto*
finally show *emeasure* ?M1 *X* = *emeasure* ?M2 *X* .
qed *simp*
finally show *distr* *M* *borel* ((* c) = *density* *lborel* ?f' .
qed (*insert* *assms*, auto dest: has-densityD)

lemma *distr-uminus-real*:

assumes *has-density* *M* *lborel* (*f* :: *real* \Rightarrow *ennreal*)
shows *has-density* (*distr* *M* *borel* *uminus*) *lborel* ($\lambda x. f (- x)$)
proof -
from *assms* **have** *has-density* (*distr* *M* *borel* ((* (- 1))) *lborel*
 ($\lambda x. f (x / -1) * \text{ennreal} (\text{inverse} (\text{abs} (-1)))$)
by (*intro* *distr-mult-real*) *simp-all*
also have (*) (-1) = (*uminus* :: *real* \Rightarrow *real*) **by** (*intro* *ext*) *simp*
also have ($\lambda x. f (x / -1) * \text{ennreal} (\text{inverse} (\text{abs} (-1)))$) = ($\lambda x. f (-x)$)
by (*intro* *ext*) (*simp* add: *one-ennreal-def*[*symmetric*])
finally show ?thesis .
qed

lemma *distr-plus-real*:

assumes *has-density* *M* *lborel* (*f* :: *real* \Rightarrow *ennreal*)
shows *has-density* (*distr* *M* *borel* ((+) *c*) *lborel* ($\lambda x. f (x - c)$)
proof
from *assms* **have** *M* = *density* *lborel* *f* **by** (*rule* *has-densityD*)
also from *assms* **have** *Mf*[*measurable*]: *f* \in *borel-measurable* *borel*
by (*auto* dest: *has-densityD*)
hence *distr* (*density* *lborel* *f*) *borel* ((+) *c*) = *density* *lborel* ($\lambda x. f (x - c)$) (*is*
 ?M1 = ?M2)
proof (*intro* *measure-eqI*)
fix *X* **assume** *X*: *X* \in *sets* (*distr* (*density* *lborel* *f*) *borel* ((+) *c*)
with *assms* **have** *emeasure* ?M1 *X* = $\int^{+x}. f x * \text{indicator } X (c + x) \partial \text{lborel}$
by (*subst* *emeasure-distr*, *simp*, *simp*, *subst* *emeasure-density*)
 (auto dest: *has-densityD* intro!: *measurable-sets-borel* *nn-integral-cong*
 split: *split-indicator*)
also from *X* **have** ... = $\int^{+x}. f (x - c) * \text{indicator } X x \partial \text{lborel}$
by (*subst* *lborel-distr-plus*[*where* *c* = -*c*, *symmetric*], *subst* *nn-integral-distr*)
auto
also from *X* **have** ... = *emeasure* ?M2 *X*
by (*subst* *emeasure-density*)
 (auto *simp*: *emeasure-density* intro!: *measurable-compose*[*OF* *borel-measurable-diff*

$Mf]$
finally show $\text{emeasure } ?M1 X = \text{emeasure } ?M2 X$.
qed simp
finally show $\text{distr } M \text{ borel } ((+) c) = \text{density lborel } (\lambda x. f (x - c))$.
qed (*insert assms, auto dest: has-densityD*)

lemma *count-space-uminus*:
 $\text{count-space UNIV} = \text{distr } (\text{count-space UNIV}) (\text{count-space UNIV}) (\text{uminus} :: ('a :: \text{ring} \Rightarrow -))$
proof (*rule distr-bij-count-space[symmetric]*)
show $\text{bij } (\text{uminus} :: 'a \Rightarrow 'a)$
by (*auto intro!: o-bij[where g=uminus]*)
qed

lemma *count-space-plus*:
 $\text{count-space UNIV} = \text{distr } (\text{count-space UNIV}) (\text{count-space UNIV}) ((+) (c :: ('a :: \text{ring})))$
by (*rule distr-bij-count-space [symmetric]*) *simp*

lemma *distr-uminus-ring-count-space*:
assumes $\text{has-density } M (\text{count-space UNIV}) (f :: - :: \text{ring} \Rightarrow \text{ennreal})$
shows $\text{has-density } (\text{distr } M (\text{count-space UNIV}) \text{uminus}) (\text{count-space UNIV}) (\lambda x. f (- x))$
proof
from *assms* **have** $M = \text{density } (\text{count-space UNIV}) f$ **by** (*rule has-densityD*)
also have $\text{distr } (\text{density } (\text{count-space UNIV}) f) (\text{count-space UNIV}) \text{uminus} = \text{density } (\text{count-space UNIV}) (\lambda x. f (- x))$ (**is** $?M1 = ?M2$)
proof (*intro measure-eqI*)
fix X **assume** $X: X \in \text{sets } (\text{distr } (\text{density } (\text{count-space UNIV}) f) (\text{count-space UNIV}) \text{uminus})$
with *assms* **have** $\text{emeasure } ?M1 X = \int^{+x}. f x * \text{indicator } X (-x) \partial \text{count-space UNIV}$
by (*subst emeasure-distr, simp, simp, subst emeasure-density*)
(auto dest: has-densityD intro!: measurable-sets-borel nn-integral-cong split: split-indicator)
also from X **have** $\dots = \text{emeasure } ?M2 X$
by (*subst count-space-uminus*) (*simp-all add: nn-integral-distr emeasure-density*)
finally show $\text{emeasure } ?M1 X = \text{emeasure } ?M2 X$.
qed simp
finally show $\text{distr } M (\text{count-space UNIV}) \text{uminus} = \text{density } (\text{count-space UNIV}) (\lambda x. f (- x))$.
qed (*insert assms, auto dest: has-densityD*)

lemma *distr-plus-ring-count-space*:
assumes $\text{has-density } M (\text{count-space UNIV}) (f :: - :: \text{ring} \Rightarrow \text{ennreal})$
shows $\text{has-density } (\text{distr } M (\text{count-space UNIV}) ((+) c)) (\text{count-space UNIV}) (\lambda x. f (x - c))$
proof
from *assms* **have** $M = \text{density } (\text{count-space UNIV}) f$ **by** (*rule has-densityD*)

also have $\text{distr } (\text{density } (\text{count-space } UNIV) f) (\text{count-space } UNIV) ((+) c) =$
 $\text{density } (\text{count-space } UNIV) (\lambda x. f (x - c))$ (**is** $?M1 = ?M2$)
proof (*intro measure-eqI*)
fix X **assume** $X: X \in \text{sets } (\text{distr } (\text{density } (\text{count-space } UNIV) f) (\text{count-space } UNIV) ((+) c))$
with *assms* **have** $\text{emeasure } ?M1 X = \int^+ x. f x * \text{indicator } X (c + x)$
 $\partial \text{count-space } UNIV$
by (*subst emeasure-distr, simp, simp, subst emeasure-density*)
(auto dest: has-densityD intro!: measurable-sets-borel nn-integral-cong
split: split-indicator)
also from X **have** $\dots = \text{emeasure } ?M2 X$
by (*subst count-space-plus[of -c] (simp-all add: nn-integral-distr emeasure-density)*)
finally show $\text{emeasure } ?M1 X = \text{emeasure } ?M2 X$.
qed *simp*
finally show $\text{distr } M (\text{count-space } UNIV) ((+) c) = \text{density } (\text{count-space } UNIV) (\lambda x. f (x - c))$.
qed (*insert assms, auto dest: has-densityD*)

lemma *subprob-density-distr-real-eq:*

assumes *dens: has-subprob-density M lborel f*
assumes *Mh: h ∈ borel-measurable borel*
assumes *Mg: g ∈ borel-measurable borel*
assumes *measure-eq:*
 $\bigwedge a b. a \leq b \implies \text{emeasure } (\text{distr } (\text{density } \text{lborel } f) \text{lborel } h) \{a..b\} =$
 $\text{emeasure } (\text{density } \text{lborel } g) \{a..b\}$
shows *has-subprob-density (distr M borel (h :: real ⇒ real)) lborel g*
proof (*rule has-subprob-densityI*)
from *dens* **have** *sets-M: sets M = sets borel* **by** (*auto dest: has-subprob-densityD*)
have *meas-M[simp]: measurable M = measurable borel*
by (*intro ext, subst measurable-cong-sets[OF sets-M refl]*) *auto*
from *Mh* **and** *dens* **show** *subprob-space: subprob-space (distr M borel h)*
by (*intro subprob-space.subprob-space-distr (auto dest: has-subprob-densityD)*)
show $\text{distr } M \text{ borel } h = \text{density } \text{lborel } g$
proof (*rule measure-eqI-generator-eq[OF Int-stable-Icc, of UNIV]*)
{
fix $x :: \text{real}$
obtain $n :: \text{nat}$ **where** $n > \text{abs } x$ **using** *reals-Archimedean2* **by** *auto*
hence $\exists n :: \text{nat}. x \in \{-\text{real } n.. \text{real } n\}$ **by** (*intro exI[of - n]*) *auto*
}
thus $(\bigcup i :: \text{nat}. \{-\text{real } i.. \text{real } i\}) = UNIV$ **by** *blast*
next
fix $i :: \text{nat}$
from *subprob-space* **have** $\text{emeasure } (\text{distr } M \text{ borel } h) \{-\text{real } i.. \text{real } i\} \leq 1$
by (*intro subprob-space.subprob-emeasure-le-1 (auto dest: has-subprob-densityD)*)
thus $\text{emeasure } (\text{distr } M \text{ borel } h) \{-\text{real } i.. \text{real } i\} \neq \infty$ **by** *auto*
next
fix $X :: \text{real set}$ **assume** $X \in \text{range } (\lambda(a,b). \{a..b\})$

then obtain $a \leq b$ **where** $X = \{a..b\}$ **by** *auto*
with *dens* **have** $\text{emeasure } (\text{distr } M \text{ lborel } h) X = \text{emeasure } (\text{density } \text{lborel } g) X$
by (*cases* $a \leq b$) (*auto simp: measure-eq dest: has-subprob-densityD*)
also have $\text{distr } M \text{ lborel } h = \text{distr } M \text{ borel } h$
by (*rule distr-cong*) *auto*
finally show $\text{emeasure } (\text{distr } M \text{ borel } h) X = \text{emeasure } (\text{density } \text{lborel } g) X$.
qed (*auto simp: borel-eq-atLeastAtMost*)
qed (*insert assms, auto*)

lemma *subprob-density-distr-real-exp:*

assumes *dens: has-subprob-density* $M \text{ lborel } f$
shows *has-subprob-density* $(\text{distr } M \text{ borel } \text{exp}) \text{ lborel}$
 $(\lambda x. \text{if } x > 0 \text{ then } f (\ln x) * \text{ennreal } (\text{inverse } x) \text{ else } 0)$
(is *has-subprob-density* $- - ?g$)

proof (*rule subprob-density-distr-real-eq[OF dens]*)

from *dens* **have** [*measurable*]: $f \in \text{borel-measurable borel}$
by (*auto dest: has-subprob-densityD*)

have $Mf: (\lambda x. f (\ln x) * \text{ennreal } (\text{inverse } x)) \in \text{borel-measurable borel}$ **by** *simp*

fix $a \leq b :: \text{real}$ **assume** $a \leq b$

let $?A = \lambda i. \{ \text{inverse } (\text{Suc } i) :: \text{real} \dots \}$

let $?M1 = \text{distr } (\text{density } \text{lborel } f) \text{ lborel } \text{exp}$ **and** $?M2 = \text{density } \text{lborel } ?g$

{
fix $x :: \text{real}$ **assume** $\forall i. x < \text{inverse } (\text{Suc } i)$
hence $x \leq 0$ **by** (*intro tendsto-lowerbound[OF LIMSEQ-inverse-real-of-nat]*)
(auto intro!: always-eventually less-imp-le)
}

hence *decomp*: $\{a..b\} = \{x \in \{a..b\}. x \leq 0\} \cup (\bigcup i. ?A i \cap \{a..b\})$ **(is** $- = ?C \cup ?D$)

by (*auto simp: not-le*)

have *inv-le*: $\bigwedge x i. x \geq \text{inverse } (\text{real } (\text{Suc } i)) \implies \neg(x \leq 0)$

by (*subst not-le, erule dual-order.strict-trans1, simp*)

hence $\text{emeasure } ?M1 \{a..b\} = \text{emeasure } ?M1 ?C + \text{emeasure } ?M1 ?D$

by (*subst decomp, intro plus-emeasure[symmetric]*) *auto*

also have $\text{emeasure } ?M1 ?C = 0$ **by** (*subst emeasure-distr*) *auto*

also have $0 = \text{emeasure } ?M2 ?C$

by (*subst emeasure-density, simp, simp, rule sym, subst nn-integral-0-iff*) *auto*

also have $\text{emeasure } ?M1 (\bigcup i. ?A i \cap \{a..b\}) = (\text{SUP } i. \text{emeasure } ?M1 (?A i \cap \{a..b\}))$

by (*rule SUP-emeasure-incseq[symmetric]*)

(auto simp: incseq-def max-def not-le dest: order.strict-trans1)

also have $\bigwedge i. \text{emeasure } ?M1 (?A i \cap \{a..b\}) = \text{emeasure } ?M2 (?A i \cap \{a..b\})$

proof (*case-tac inverse (Suc i) ≤ b*)

fix i **assume** *True*: $\text{inverse } (\text{Suc } i) \leq b$

let $?a = \text{inverse } (\text{Suc } i)$

from $\langle a \leq b \rangle$ **have** $A: ?A i \cap \{a..b\} = \{\max ?a a..b\}$ **(is** $?E = ?F$) **by** *auto*

hence $\text{emeasure } ?M1 ?E = \text{emeasure } ?M1 ?F$ **by** *simp*

also have *strict-mono-on* $\{\max(\text{inverse}(\text{real}(\text{Suc } i))) a..b\}$ *ln*
by (*rule strict-mono-onI*, *subst ln-less-cancel-iff*)
(auto dest: inv-le simp del: of-nat-Suc)
with $\langle a \leq b \rangle$ *True dens*
have *emeasure* $?M1$ $?F = \text{emeasure}(\text{density lborel } (\lambda x. f(\text{ln } x) * \text{inverse } x))$
 $?F$
by (*intro emeasure-density-distr-interval*)
(auto simp: Mf not-less not-le range-exp dest: has-subprob-densityD dest!:
inv-le
intro!: DERIV-ln continuous-on-inverse continuous-on-id simp del:
of-nat-Suc)
also note $A[\text{symmetric}]$
also have *emeasure* $(\text{density lborel } (\lambda x. f(\text{ln } x) * \text{inverse } x))$ $?E = \text{emeasure}$
 $?M2$ $?E$
by (*subst (1 2) emeasure-density*)
(auto intro!: nn-integral-cong split: split-indicator dest!: inv-le simp del:
of-nat-Suc)
finally show *emeasure* $?M1$ $(?A \text{ } i \cap \{a..b\}) = \text{emeasure } ?M2$ $(?A \text{ } i \cap \{a..b\})$
 $.$
qed simp
hence $(\text{SUP } i. \text{emeasure } ?M1$ $(?A \text{ } i \cap \{a..b\})) = (\text{SUP } i. \text{emeasure } ?M2$ $(?A \text{ } i$
 $\cap \{a..b\}))$ **by** *simp*
also have $\dots = \text{emeasure } ?M2$ $(\bigcup i. ?A \text{ } i \cap \{a..b\})$
by (*rule SUP-emeasure-incseq*)
(auto simp: incseq-def max-def not-le dest: order.strict-trans1)
also have *emeasure* $?M2$ $?C + \text{emeasure } ?M2$ $?D = \text{emeasure } ?M2$ $(?C \cup ?D)$
by (*rule plus-emeasure*) *(auto dest: inv-le simp del: of-nat-Suc)*
also note *decomp[symmetric]*
finally show *emeasure* $?M1$ $\{a..b\} = \text{emeasure } ?M2$ $\{a..b\}$.
qed (*insert dens, auto dest!: has-subprob-densityD(1)*)

lemma *subprob-density-distr-real-inverse-aux:*
assumes *dens: has-subprob-density M lborel f*
shows *has-subprob-density* $(\text{distr } M \text{ borel } (\lambda x. - \text{inverse } x))$ *lborel*
 $(\lambda x. f(-\text{inverse } x) * \text{ennreal}(\text{inverse}(x * x)))$
(is has-subprob-density - - ?g)
proof (*rule subprob-density-distr-real-eq[OF dens]*)
from *dens* **have** $Mf[\text{measurable}]: f \in \text{borel-measurable borel}$ **by** (*auto dest:*
has-subprob-densityD)
show $Mg: ?g \in \text{borel-measurable borel}$ **by** *measurable*

have *surj[simp]: surj* $(\lambda x. - \text{inverse } x :: \text{real})$
by (*intro surjI[of - $\lambda x. - \text{inverse } x$] (simp add: field-simps)*)
fix $a \ b :: \text{real}$ **assume** $a \leq b$
let $?A1 = \lambda i. \{..-\text{inverse}(\text{Suc } i) :: \text{real}\}$ **and** $?A2 = \lambda i. \{\text{inverse}(\text{Suc } i) ::$
 $\text{real} ..\}$
let $?C = \text{if } 0 \in \{a..b\} \text{ then } \{0\} \text{ else } \{\}$
let $?M1 = \text{distr}(\text{density lborel } f) \text{ lborel } (\lambda x. - \text{inverse } x)$ **and** $?M2 = \text{density}$
 $\text{lborel } ?g$

have *inv-le*: $\bigwedge x i. x \geq \text{inverse } (\text{real } (\text{Suc } i)) \implies \neg(x \leq 0)$
by (*subst not-le, erule dual-order.strict-trans1, simp*)
have $\bigwedge x. x > 0 \implies \exists i. x \geq \text{inverse } (\text{Suc } i)$
proof (*rule ccontr*)
fix $x :: \text{real}$ **assume** $x > 0 \neg(\exists i. x \geq \text{inverse } (\text{Suc } i))$
hence $x \leq 0$ **by** (*intro tendsto-lowerbound[OF LIMSEQ-inverse-real-of-nat]*)
(auto intro!: always-eventually less-imp-le simp: not-le)
with $\langle x > 0 \rangle$ **show** *False* **by** *simp*
qed
hence $A: (\bigcup i. ?A2 i) = \{0 < ..\}$ **by** (*auto dest: inv-le simp del: of-nat-Suc*)
moreover **have** $\bigwedge x. x < 0 \implies \exists i. x \leq -\text{inverse } (\text{Suc } i)$
proof (*rule ccontr*)
fix $x :: \text{real}$ **assume** $x < 0 \neg(\exists i. x \leq -\text{inverse } (\text{Suc } i))$
hence $x \geq 0$
by (*intro tendsto-upperbound, simp*)
(auto intro!: always-eventually less-imp-le LIMSEQ-inverse-real-of-nat-add-minus
simp: not-le)
with $\langle x < 0 \rangle$ **show** *False* **by** *simp*
qed
hence $B: (\bigcup i. ?A1 i) = \{.. < 0\}$
by (*auto simp: le-minus-iff[of - inverse x for x] dest!: inv-le simp del: of-nat-Suc*)
ultimately **have** $C: \text{UNIV} = (\bigcup i. ?A1 i) \cup (\bigcup i. ?A2 i) \cup \{0\}$ **by** (*subst A,*
subst B) force
have *UN-Int-distrib*: $\bigwedge f A. (\bigcup i. f i) \cap A = (\bigcup i. f i \cap A)$ **by** *blast*
have *decomp*: $\{a..b\} = (\bigcup i. ?A1 i \cap \{a..b\}) \cup (\bigcup i. ?A2 i \cap \{a..b\}) \cup ?C$ (*is -*
= ?D \cup ?E \cup -)
by (*subst Int-UNIV-left[symmetric], simp only: C Int-Un-distrib2 UN-Int-distrib*)
(simp split: if-split)
have *emeasure ?M1* $\{a..b\} = \text{emeasure } ?M1 ?D + \text{emeasure } ?M1 ?E + \text{emeasure}$
 $?M1 ?C$
apply (*subst decomp*)
apply (*subst plus-emeasure[symmetric], simp, simp, simp*)
apply (*subst plus-emeasure[symmetric]*)
apply (*auto dest!: inv-le simp: not-le le-minus-iff[of - inverse x for x] simp del:*
of-nat-Suc)
done
also **have** $(\lambda x. - \text{inverse } x) -' \{0 :: \text{real}\} = \{0\}$ **by** (*auto simp: field-simps*)
hence *emeasure ?M1* $?C = 0$
by (*subst emeasure-distr*) (*auto split: if-split simp: emeasure-density Mf*)
also **have** *emeasure ?M2* $\{0\} = 0$ **by** (*simp add: emeasure-density*)
hence $0 = \text{emeasure } ?M2 ?C$
by (*rule-tac sym, rule-tac order.antisym, rule-tac order.trans, rule-tac emeasure-*
mono[of - {0}]) simp-all
also **have** *emeasure ?M1* $(\bigcup i. ?A1 i \cap \{a..b\}) = (\text{SUP } i. \text{emeasure } ?M1 (?A1$
 $i \cap \{a..b\}))$
by (*rule SUP-emeasure-incseq[symmetric]*)
(auto simp: incseq-def max-def not-le dest: order.strict-trans1)
also **have** $\bigwedge i. \text{emeasure } ?M1 (?A1 i \cap \{a..b\}) = \text{emeasure } ?M2 (?A1 i \cap \{a..b\})$
proof (*case-tac -inverse (Suc i) \geq a*)

fix i **assume** $\text{True}: -\text{inverse} (\text{Suc } i) \geq a$
let $?a = -\text{inverse} (\text{Suc } i)$
from $\langle a \leq b \rangle$ **have** $A: ?A1 \ i \cap \{a..b\} = \{a.. \min ?a \ b\}$ (**is** $?F = ?G$) **by** *auto*
hence $\text{emeasure } ?M1 \ ?F = \text{emeasure } ?M1 \ ?G$ **by** *simp*
also have *strict-mono-on* $\{a.. \min ?a \ b\}$ $(\lambda x. -\text{inverse } x)$
by (*rule strict-mono-onI*)
*(auto simp: le-minus-iff[*of - inverse x for x*] dest!: *inv-le simp del: of-nat-Suc*)*
with $\langle a \leq b \rangle$ *True dens*
have $\text{emeasure } ?M1 \ ?G = \text{emeasure } ?M2 \ ?G$
by (*intro emeasure-density-distr-interval*)
(auto simp: Mf not-less dest: has-subprob-densityD inv-le
intro!: derivative-eq-intros continuous-on-mult continuous-on-inverse
continuous-on-id)
also note $A[\text{symmetric}]$
finally show $\text{emeasure } ?M1 \ (?A1 \ i \cap \{a..b\}) = \text{emeasure } ?M2 \ (?A1 \ i \cap \{a..b\})$
qed *simp*
hence $(\text{SUP } i. \text{emeasure } ?M1 \ (?A1 \ i \cap \{a..b\})) = (\text{SUP } i. \text{emeasure } ?M2 \ (?A1 \ i \cap \{a..b\}))$ **by** *simp*
also have $\dots = \text{emeasure } ?M2 \ (\bigcup i. ?A1 \ i \cap \{a..b\})$
by (*rule SUP-emeasure-incseq*)
(auto simp: incseq-def max-def not-le dest: order.strict-trans1)
also have $\text{emeasure } ?M1 \ (\bigcup i. ?A2 \ i \cap \{a..b\}) = (\text{SUP } i. \text{emeasure } ?M1 \ (?A2 \ i \cap \{a..b\}))$
by (*rule SUP-emeasure-incseq[symmetric]*)
(auto simp: incseq-def max-def not-le dest: order.strict-trans1)
also have $\bigwedge i. \text{emeasure } ?M1 \ (?A2 \ i \cap \{a..b\}) = \text{emeasure } ?M2 \ (?A2 \ i \cap \{a..b\})$
proof (*case-tac inverse (Suc i) ≤ b*)
fix i **assume** $\text{True}: \text{inverse} (\text{Suc } i) \leq b$
let $?a = \text{inverse} (\text{Suc } i)$
from $\langle a \leq b \rangle$ **have** $A: ?A2 \ i \cap \{a..b\} = \{\max ?a \ a..b\}$ (**is** $?F = ?G$) **by** *auto*
hence $\text{emeasure } ?M1 \ ?F = \text{emeasure } ?M1 \ ?G$ **by** *simp*
also have *strict-mono-on* $\{\max ?a \ a..b\}$ $(\lambda x. -\text{inverse } x)$
by (*rule strict-mono-onI*) *(auto dest!: inv-le simp: not-le simp del: of-nat-Suc)*
with $\langle a \leq b \rangle$ *True dens*
have $\text{emeasure } ?M1 \ ?G = \text{emeasure } ?M2 \ ?G$
by (*intro emeasure-density-distr-interval*)
(auto simp: Mf not-less dest: has-subprob-densityD inv-le
intro!: derivative-eq-intros continuous-on-mult continuous-on-inverse
continuous-on-id)
also note $A[\text{symmetric}]$
finally show $\text{emeasure } ?M1 \ (?A2 \ i \cap \{a..b\}) = \text{emeasure } ?M2 \ (?A2 \ i \cap \{a..b\})$
qed *simp*
hence $(\text{SUP } i. \text{emeasure } ?M1 \ (?A2 \ i \cap \{a..b\})) = (\text{SUP } i. \text{emeasure } ?M2 \ (?A2 \ i \cap \{a..b\}))$ **by** *simp*
also have $\dots = \text{emeasure } ?M2 \ (\bigcup i. ?A2 \ i \cap \{a..b\})$
by (*rule SUP-emeasure-incseq*)
(auto simp: incseq-def max-def not-le dest: order.strict-trans1)

also have $\text{emeasure } ?M2 \text{ ?}D + \text{emeasure } ?M2 \text{ ?}E + \text{emeasure } ?M2 \text{ ?}C = \text{emeasure } ?M2 \text{ \{a..b\}}$
apply (*subst (4) decomp*)
apply (*subst plus-emeasure, simp, simp*)
apply (*auto dest!: inv-le simp: not-le le-minus-iff[of - inverse x for x] simp del: of-nat-Suc*)
apply (*subst plus-emeasure*)
apply (*auto dest!: inv-le simp: not-le le-minus-iff[of - inverse x for x]*)
done
finally show $\text{emeasure } ?M1 \text{ \{a..b\}} = \text{emeasure } ?M2 \text{ \{a..b\}}$.
qed *simp*

lemma *subprob-density-distr-real-inverse:*
assumes *dens: has-subprob-density M lborel f*
shows *has-subprob-density (distr M borel inverse) lborel ($\lambda x. f (\text{inverse } x) * \text{ennreal } (\text{inverse } (x * x))$)*
proof (*unfold has-subprob-density-def, intro conjI*)
let $?g' = (\lambda x. f (-\text{inverse } x) * \text{ennreal } (\text{inverse } (x * x)))$
have *prob: has-subprob-density (distr M borel ($\lambda x. -\text{inverse } x$)) lborel ?g'*
by (*rule subprob-density-distr-real-inverse-aux[OF assms]*)
from *assms have sets-M: sets M = sets borel by (auto dest: has-subprob-densityD)*
have [*simp*]: *measurable M = measurable borel*
by (*intro ext, subst measurable-cong-sets[OF sets-M refl] auto*)
from *prob have dens: has-density (distr M lborel ($\lambda x. -\text{inverse } x$)) lborel ($\lambda x. f (-\text{inverse } x) * \text{ennreal } (\text{inverse } (x * x))$)*
unfolding *has-subprob-density-def by (simp cong: distr-cong)*
from *distr-uminus-real[OF this]*
show *has-density (distr M borel inverse) lborel ($\lambda x. f (\text{inverse } x) * \text{ennreal } (\text{inverse } (x * x))$)*
by (*simp add: distr-distr o-def cong: distr-cong*)
show *subprob-space (distr M borel inverse)*
by (*intro subprob-space.subprob-space-distr has-subprob-densityD[OF assms]*)
simp-all
qed

lemma *distr-convolution-real:*
assumes *has-density M lborel (f :: (real \times real) \Rightarrow ennreal)*
shows *has-density (distr M borel (case-prod (+))) lborel ($\lambda z. \int^+ x. f (x, z - x) \partial \text{lborel}$)*
(is has-density ?M' - ?f')
proof
from *has-densityD[OF assms] have Mf[measurable]: f \in borel-measurable borel*
by *simp*
show *Mf': ($\lambda z. \int^+ x. f (x, z - x) \partial \text{lborel}$) \in borel-measurable lborel by measurable*
able

from *assms have sets-M: sets M = sets borel by (auto dest: has-densityD)*
hence [*simp*]: *space M = UNIV by (subst sets-eq-imp-space-eq[OF sets-M]) simp*
from *sets-M have [simp]: measurable M = measurable borel*

by (intro ext measurable-cong-sets) simp-all
 have M-add: case-prod (+) ∈ borel-measurable (borel :: (real × real) measure)
 by (simp add: borel-prod[symmetric])

show distr M borel (case-prod (+)) = density lborel ?f'
 proof (rule measure-eqI)
 fix X :: real set assume X[measurable]: X ∈ sets (distr M borel (case-prod (+)))
 hence emeasure (distr M borel (case-prod (+))) X = emeasure M ((λ(x, y). x + y) - ' X)
 by (simp-all add: M-add emeasure-distr)
 also from X have ... = ∫⁺z. f z * indicator ((λ(x, y). x + y) - ' X) z ∂(lborel ⊗_M lborel)
 by (simp add: emeasure-density has-densityD[OF assms] measurable-sets-borel[OF M-add] lborel-prod)
 also have ... = ∫⁺x. ∫⁺y. f (x, y) * indicator ((λ(x, y). x + y) - ' X) (x, y) ∂lborel ∂lborel
 apply (rule lborel.nn-integral-fst[symmetric])
 apply measurable
 apply (simp-all add: borel-prod)
 done
 also have ... = ∫⁺x. ∫⁺y. f (x, y) * indicator ((λ(x, y). x + y) - ' X) (x, y) ∂distr lborel borel ((+) (-x)) ∂lborel
 by (rule nn-integral-cong, subst lborel-distr-plus) simp
 also have ... = ∫⁺x. ∫⁺z. f (x, z-x) * indicator ((λ(x, y). x + y) - ' X) (x, z-x) ∂lborel ∂lborel
 apply (rule nn-integral-cong)
 apply (subst nn-integral-distr)
 apply simp-all
 apply measurable
 apply (subst space-count-space)
 apply auto
 done
 also have ... = ∫⁺x. ∫⁺z. f (x, z-x) * indicator X z ∂lborel ∂lborel
 by (intro nn-integral-cong) (simp split: split-indicator)
 also have ... = ∫⁺z. ∫⁺x. f (x, z-x) * indicator X z ∂lborel ∂lborel using X
 by (subst lborel-pair.Fubini')
 (simp-all add: pair-sigma-finite-def)
 also have ... = ∫⁺z. (∫⁺x. f (x, z-x) ∂lborel) * indicator X z ∂lborel
 by (rule nn-integral-cong) (simp split: split-indicator)
 also have ... = emeasure (density lborel ?f') X using X
 by (simp add: emeasure-density)
 finally show emeasure (distr M borel (case-prod (+))) X = emeasure (density lborel ?f') X .
 qed (insert assms, auto dest: has-densityD)
 qed simp-all

lemma distr-convolution-ring-count-space:

assumes C : countable ($UNIV :: 'a$ set)
assumes has-density M (count-space $UNIV$) ($f :: (('a :: ring) \times 'a) \Rightarrow ennreal$)
shows has-density ($distr\ M$ (count-space $UNIV$) (case-prod (+))) (count-space $UNIV$)
 $(\lambda z. \int^+ x. f(x, z - x) \partial count-space\ UNIV)$
(is has-density ?M' - ?f')

proof

let $?CS = count-space\ UNIV :: 'a$ measure **and** $?CSP = count-space\ UNIV :: ('a \times 'a)$ measure
show Mf' : $(\lambda z. \int^+ x. f(x, z - x) \partial count-space\ UNIV) \in borel-measurable\ ?CS$
by *simp*

from *assms* **have** *sets-M*: sets $M = UNIV$ **and** [*simp*]: space $M = UNIV$
by (*auto dest: has-densityD*)
from *assms* **have** [*simp*]: measurable $M = measurable$ (count-space $UNIV$)
by (*intro ext measurable-cong-sets*) (*simp-all add: sets-M*)

interpret sigma-finite-measure $?CS$ **by** (*rule sigma-finite-measure-count-space-countable[OF C]*)

show $distr\ M\ ?CS$ (case-prod (+)) = density $?CS\ ?f'$

proof (*rule measure-egI*)

fix $X :: 'a$ set **assume** X : $X \in sets$ ($distr\ M\ ?CS$ (case-prod (+)))

hence *emeasure* ($distr\ M\ ?CS$ (case-prod (+))) $X = \text{emeasure}\ M\ ((\lambda(x, y). x + y) - 'X)$

by (*simp-all add: emeasure-distr*)

also from X **have** $\dots = \int^+ z. f\ z * indicator\ ((\lambda(x, y). x + y) - 'X)\ z\ \partial(?CS \otimes_M\ ?CS)$

by (*simp add: emeasure-density has-densityD[OF assms(2)] sets-M pair-measure-countable C*)

also have $\dots = \int^+ x. \int^+ y. f(x, y) * indicator\ ((\lambda(x, y). x + y) - 'X)\ (x, y)\ \partial ?CS\ \partial ?CS$

by (*rule nn-integral-fst[symmetric]*) (*simp add: pair-measure-countable C*)

also have $\dots = \int^+ x. \int^+ y. f(x, y) * indicator\ ((\lambda(x, y). x + y) - 'X)\ (x, y)\ \partial distr\ ?CS\ ?CS\ ((+)\ (-x))\ \partial ?CS$

by (*rule nn-integral-cong, subst count-space-plus*) *simp*

also have $\dots = \int^+ x. \int^+ z. f(x, z-x) * indicator\ ((\lambda(x, y). x + y) - 'X)\ (x, z-x)\ \partial ?CS\ \partial ?CS$

by (*rule nn-integral-cong*) (*simp-all add: nn-integral-distr*)

also have $\dots = \int^+ x. \int^+ z. f(x, z-x) * indicator\ X\ z\ \partial ?CS\ \partial ?CS$

by (*intro nn-integral-cong*) (*simp split: split-indicator*)

also have $\dots = \int^+ z. \int^+ x. f(x, z-x) * indicator\ X\ z\ \partial ?CS\ \partial ?CS$ **using** X

by (*subst pair-sigma-finite.Fubini'*)

(*simp-all add: pair-sigma-finite-def sigma-finite-measure-count-space-countable C pair-measure-countable*)

also have $\dots = \int^+ z. (\int^+ x. f(x, z-x)\ \partial ?CS) * indicator\ X\ z\ \partial ?CS$

by (*rule nn-integral-cong*) (*simp split: split-indicator*)

also have $\dots = \text{emeasure}\ (density\ ?CS\ ?f')\ X$ **using** X **by** (*simp add: emeasure-density*)

finally show *emeasure* ($distr\ M\ ?CS$ (case-prod (+))) $X = \text{emeasure}\ (density$

```

?CS ?f' X .
qed (insert assms, auto dest: has-densityD)
qed simp-all

end

```

3 Source Language Values

```

theory PDF-Values
imports Density-Predicates
begin

```

3.1 Values and stock measures

```

datatype pdf-type = UNIT | BOOL | INTEG | REAL | PRODUCT pdf-type
pdf-type

```

```

datatype val = UnitVal
| BoolVal (extract-bool: bool)
| IntVal (extract-int: int)
| RealVal (extract-real: real)
| PairVal (extract-fst: val) (extract-snd: val) (<<|-, -|>> [0, 61] 1000)

```

where

```

extract-bool UnitVal = False
| extract-bool (IntVal i) = False
| extract-bool (RealVal r) = False
| extract-bool (PairVal x y) = False
extract-int UnitVal = 0
| extract-int (BoolVal b) = 0
| extract-int (RealVal r) = 0
| extract-int (PairVal x y) = 0
extract-real UnitVal = 0
| extract-real (BoolVal b) = 0
| extract-real (IntVal i) = 0
| extract-real (PairVal x y) = 0

```

primrec *extract-pair'* where

```

extract-pair' f s <| x, y |> = (f x, s y)

```

definition *map-int-pair* where

```

map-int-pair f g x = (case x of <| IntVal a, IntVal b |> => f a b | - => g x)

```

definition *map-real-pair* where

```

map-real-pair f g x = (case x of <| RealVal a, RealVal b |> => f a b | - => g x)

```

abbreviation *TRUE* \equiv *BoolVal True*

abbreviation *FALSE* \equiv *BoolVal False*

type-synonym *vname* = *nat*

type-synonym $state = vname \Rightarrow val$

lemma $map\text{-}int\text{-}pair[simp]: map\text{-}int\text{-}pair\ f\ g\ <| IntVal\ i,\ IntVal\ j |> = f\ i\ j$
by ($simp\ add: map\text{-}int\text{-}pair\text{-}def$)

lemma $map\text{-}int\text{-}pair\text{-}REAL[simp]: map\text{-}int\text{-}pair\ f\ g\ <| RealVal\ i,\ RealVal\ j |> =$
 $g\ <| RealVal\ i,\ RealVal\ j |>$
by ($simp\ add: map\text{-}int\text{-}pair\text{-}def$)

lemma $map\text{-}real\text{-}pair[simp]: map\text{-}real\text{-}pair\ f\ g\ <| RealVal\ i,\ RealVal\ j |> = f\ i\ j$
by ($simp\ add: map\text{-}real\text{-}pair\text{-}def$)

abbreviation $extract\text{-}pair \equiv extract\text{-}pair'\ id\ id$

abbreviation $extract\text{-}real\text{-}pair \equiv extract\text{-}pair'\ extract\text{-}real\ extract\text{-}real$

abbreviation $extract\text{-}int\text{-}pair \equiv extract\text{-}pair'\ extract\text{-}int\ extract\text{-}int$

definition $RealPairVal \equiv \lambda(x,y). <| RealVal\ x,\ RealVal\ y |>$

definition $IntPairVal \equiv \lambda(x,y). <| IntVal\ x,\ IntVal\ y |>$

lemma $inj\text{-}RealPairVal: inj\ RealPairVal$ **by** ($auto\ simp: RealPairVal\text{-}def\ intro!: injI$)

lemma $inj\text{-}IntPairVal: inj\ IntPairVal$ **by** ($auto\ simp: IntPairVal\text{-}def\ intro!: injI$)

fun $val\text{-}type :: val \Rightarrow pdf\text{-}type$ **where**

$val\text{-}type\ (BoolVal\ b) = BOOL$
 $| val\text{-}type\ (IntVal\ i) = INTEG$
 $| val\text{-}type\ (UnitVal) = UNIT$
 $| val\text{-}type\ (RealVal\ r) = REAL$
 $| val\text{-}type\ (<| v1 , v2 |>) = (PRODUCT\ (val\text{-}type\ v1)\ (val\text{-}type\ v2))$

lemma $val\text{-}type\text{-}eq\text{-}REAL: val\text{-}type\ x = REAL \longleftrightarrow x \in RealVal'UNIV$
by ($cases\ x\ auto$)

lemma $val\text{-}type\text{-}eq\text{-}INTEG: val\text{-}type\ x = INTEG \longleftrightarrow x \in IntVal'UNIV$
by ($cases\ x\ auto$)

definition $type\text{-}universe\ t = \{v. val\text{-}type\ v = t\}$

lemma $type\text{-}universe\text{-}nonempty[simp]: type\text{-}universe\ t \neq \{\}$
by ($induction\ t$) ($auto\ intro: val\text{-}type.simps\ simp: type\text{-}universe\text{-}def$)

lemma $val\text{-}in\text{-}type\text{-}universe[simp]:$
 $v \in type\text{-}universe\ (val\text{-}type\ v)$
by ($simp\ add: type\text{-}universe\text{-}def$)

lemma $BoolVal\text{-}in\text{-}type\text{-}universe[simp]: BoolVal\ v \in type\text{-}universe\ BOOL$
by ($simp\ add: type\text{-}universe\text{-}def$)

lemma *IntVal-in-type-universe*[simp]: *IntVal* $v \in \text{type-universe } \text{INTEG}$
by (*simp add: type-universe-def*)

lemma *type-universe-type*[simp]:
 $w \in \text{type-universe } t \longleftrightarrow \text{val-type } w = t$
by (*simp add: type-universe-def*)

lemma *type-universe-REAL*: *type-universe* *REAL* = *RealVal* ‘ *UNIV*
apply (*auto simp add: set-eq-iff image-iff*)
apply (*case-tac x*)
apply *auto*
done

lemma *type-universe-eq-imp-type-eq*:
assumes *type-universe* $t1 = \text{type-universe } t2$
shows $t1 = t2$

proof –

from *type-universe-nonempty* **obtain** v **where** $A: v \in \text{type-universe } t1$ **by** *blast*
hence $t1 = \text{val-type } v$ **by** *simp*
also from A **and** *assms* **have** $v \in \text{type-universe } t2$ **by** *simp*
hence $\text{val-type } v = t2$ **by** *simp*
finally show *?thesis* .

qed

lemma *type-universe-eq-iff*[simp]: *type-universe* $t1 = \text{type-universe } t2 \longleftrightarrow t1 = t2$
by (*blast intro: type-universe-eq-imp-type-eq*)

primrec *stock-measure* :: *pdf-type* \Rightarrow *val measure* **where**
 $\text{stock-measure } \text{UNIT} = \text{count-space } \{\text{UnitVal}\}$
 $\text{stock-measure } \text{INTEG} = \text{count-space } (\text{range } \text{IntVal})$
 $\text{stock-measure } \text{BOOL} = \text{count-space } (\text{range } \text{BoolVal})$
 $\text{stock-measure } \text{REAL} = \text{embed-measure } \text{lborel } \text{RealVal}$
 $\text{stock-measure } (\text{PRODUCT } t1\ t2) =$
 $\text{embed-measure } (\text{stock-measure } t1 \otimes_M \text{stock-measure } t2) (\lambda(a, b). \langle |a, b| \rangle)$

declare [[*coercion stock-measure*]]

lemma *sigma-finite-stock-measure*[simp]: *sigma-finite-measure* (*stock-measure* t)
by (*induction t*)
(*auto intro!: sigma-finite-measure-count-space-countable sigma-finite-pair-measure sigma-finite-embed-measure injI sigma-finite-lborel*)

lemma *val-case-stock-measurable*:

assumes $t = \text{UNIT} \Longrightarrow c \in \text{space } M$
assumes $\bigwedge b. t = \text{BOOL} \Longrightarrow g\ b \in \text{space } M$
assumes $\bigwedge i. t = \text{INTEG} \Longrightarrow h\ i \in \text{space } M$
assumes $t = \text{REAL} \Longrightarrow j \in \text{measurable borel } M$
assumes *: $\bigwedge t1\ t2. t = \text{PRODUCT } t1\ t2 \Longrightarrow \text{case-prod } k \in \text{measurable } (\text{stock-measure } t1\ t2)$

$t1 \otimes_M \text{stock-measure } t2) M$
shows $(\lambda x. \text{case } x \text{ of } \text{UnitVal} \Rightarrow c \mid \text{BoolVal } b \Rightarrow g \ b \mid \text{IntVal } i \Rightarrow h \ i \mid \text{RealVal } r \Rightarrow j \ r$
 $\mid \text{PairVal } y \ z \Rightarrow k \ y \ z) \in \text{measurable } t \ M$

proof $(\text{cases } t)$
case $(\text{PRODUCT } t1 \ t2)$ **with** $*[\text{of } t1 \ t2]$ **show** $?thesis$
by $(\text{auto intro!} : \text{measurable-embed-measure1 simp} : \text{split-beta}^\wedge)$
qed $(\text{auto intro!} : \text{measurable-embed-measure1 assms})$

lemma $\text{space-stock-measure}[\text{simp}] : \text{space } (\text{stock-measure } t) = \text{type-universe } t$
by $(\text{induction } t)$
 $(\text{auto simp add} : \text{type-universe-def space-pair-measure space-embed-measure}$
 $\text{simp del} : \text{type-universe-type elim} : \text{val-type.elims})$

lemma $\text{type-universe-stock-measure}[\text{measurable}] : \text{type-universe } t \in \text{sets } (\text{stock-measure } t)$
using $\text{sets.top}[\text{of stock-measure } t]$ **by** simp

lemma $\text{inj-RealVal}[\text{simp}] : \text{inj } \text{RealVal}$ **by** $(\text{auto intro!} : \text{inj-onI})$
lemma $\text{inj-IntVal}[\text{simp}] : \text{inj } \text{IntVal}$ **by** $(\text{auto intro!} : \text{inj-onI})$
lemma $\text{inj-BoolVal}[\text{simp}] : \text{inj } \text{BoolVal}$ **by** $(\text{auto intro!} : \text{inj-onI})$
lemma $\text{inj-PairVal}[\text{simp}] : \text{inj } (\lambda(x, y). < \mid x, \ y \mid >)$ **by** $(\text{auto intro} : \text{injI})$

lemma $\text{measurable-PairVal}[\text{measurable}] :$
fixes $t1 \ t2 :: \text{pdf-type}$
shows $\text{case-prod } \text{PairVal} \in \text{measurable } (t1 \otimes_M t2) (\text{PRODUCT } t1 \ t2)$
using $\text{measurable-embed-measure2}[\text{measurable}]$ **by** simp

lemma $\text{measurable-RealVal}[\text{measurable}] : \text{RealVal} \in \text{measurable borel } \text{REAL}$
using $\text{measurable-embed-measure2}[\text{measurable}]$ **by** simp

lemma $\text{nn-integral-BoolVal} :$
assumes $\bigwedge x. f (\text{BoolVal } x) \geq 0$
shows $(\int^{+x}. f \ x \ \partial \text{BOOL}) = f (\text{BoolVal } \text{True}) + f (\text{BoolVal } \text{False})$
proof –
have $A : \text{range } \text{BoolVal} = \{\text{BoolVal } \text{True}, \text{BoolVal } \text{False}\}$ **by** auto
from assms **show** $?thesis$
by $(\text{subst stock-measure.simps}, \text{subst } A, \text{subst nn-integral-count-space-finite})$
 $(\text{simp-all add} : \text{max-def } A)$

qed

lemma $\text{nn-integral-RealVal} :$
 $f \in \text{borel-measurable } \text{REAL} \implies (\int^{+x}. f \ x \ \partial \text{REAL}) = (\int^{+x}. f (\text{RealVal } x) \ \partial \text{borel})$
unfolding $\text{stock-measure.simps}$ **using** $\text{measurable-embed-measure2}[\text{measurable}]$
by $(\text{subst embed-measure-eq-distr}, \text{simp-all add} : \text{nn-integral-distr})$

lemma $\text{nn-integral-IntVal} : (\int^{+x}. f \ x \ \partial \text{INTEG}) = (\int^{+x}. f (\text{IntVal } x) \ \partial \text{count-space UNIV})$

using *measurable-embed-measure1* [*measurable (raw)*]
unfolding *stock-measure.simps embed-measure-count-space* [*OF inj-IntVal, symmetric*]
by (*subst embed-measure-eq-distr* [*OF inj-IntVal*], *simp add: nn-integral-distr space-embed-measure*)

lemma *nn-integral-PairVal*:

f ∈ *borel-measurable (PRODUCT t1 t2)* ⇒
 $(\int^{+x}. f x \partial \text{PRODUCT } t1 \ t2) = (\int^{+x}. f (\text{PairVal } (\text{fst } x) (\text{snd } x)) \partial (t1 \otimes_M t2))$

unfolding *stock-measure.simps*

by (*subst nn-integral-embed-measure*) (*simp-all add: split-beta' inj-on-def*)

lemma *BOOL-E*: $\llbracket \text{val-type } v = \text{BOOL}; \bigwedge b. v = \text{BoolVal } b \implies P \rrbracket \implies P$

by (*cases v*) *auto*

lemma *PROD-E*: $\llbracket \text{val-type } v = \text{PRODUCT } t1 \ t2 ;$

$\bigwedge a \ b. \text{val-type } a = t1 \implies \text{val-type } b = t2 \implies v = \langle | a, b | \rangle \implies P \rrbracket \implies P$

by (*cases v*) *auto*

lemma *REAL-E*: $\llbracket \text{val-type } v = \text{REAL}; \bigwedge b. v = \text{RealVal } b \implies P \rrbracket \implies P$

by (*cases v*) *auto*

lemma *INTEG-E*: $\llbracket \text{val-type } v = \text{INTEG}; \bigwedge i. v = \text{IntVal } i \implies P \rrbracket \implies P$

by (*cases v*) *auto*

lemma *measurable-extract-pair'* [*measurable (raw)*]:

fixes *t1 t2* :: *pdf-type*

assumes [*measurable*]: *f* ∈ *measurable t1 M*

assumes [*measurable*]: *g* ∈ *measurable t2 N*

assumes *h*: *h* ∈ *measurable K (PRODUCT t1 t2)*

shows $(\lambda x. \text{extract-pair}' f g (h x)) \in \text{measurable } K (M \otimes_M N)$

by (*rule measurable-compose* [*OF h[unfolding stock-measure.simps] measurable-embed-measure1*])
(simp add: split-beta')

lemma *measurable-extract-pair* [*measurable*]: *extract-pair* ∈ *measurable (PRODUCT t1 t2) (t1 \otimes_M t2)*

by *measurable*

lemma *measurable-extract-real* [*measurable*]: *extract-real* ∈ *measurable REAL borel*

apply *simp*

apply *measurable*

apply (*rule measurable-embed-measure1*)

apply *simp*

done

lemma *measurable-extract-int* [*measurable*]: *extract-int* ∈ *measurable INTEG (count-space UNIV)*

by *simp measurable*

lemma *measurable-extract-int-pair*[*measurable*]:
extract-int-pair \in *measurable* (*PRODUCT INTEG INTEG*) (*count-space UNIV*
 \otimes_M *count-space UNIV*)
by *measurable*

lemma *measurable-extract-real-pair*[*measurable*]:
extract-real-pair \in *measurable* (*PRODUCT REAL REAL*) (*borel* \otimes_M *borel*)
by *measurable*

lemma *measurable-extract-real-pair'*[*measurable*]:
extract-real-pair \in *measurable* (*PRODUCT REAL REAL*) *borel*
by (*subst borel-prod*[*symmetric*]) *measurable*

lemma *measurable-extract-bool*[*measurable*]: *extract-bool* \in *measurable* *BOOL* (*count-space UNIV*)
by *simp*

lemma *map-int-pair-measurable*[*measurable*]:
assumes *f*: *case-prod* *f* \in *measurable* (*count-space UNIV* \otimes_M *count-space UNIV*) *M*
shows *map-int-pair* *f g* \in *measurable* (*PRODUCT INTEG INTEG*) *M*
proof (*subst measurable-cong*)
fix *w* **assume** *w* \in *space* (*PRODUCT INTEG INTEG*)
then show *map-int-pair* *f g w* = (*case-prod* *f* *o* *extract-int-pair*) *w*
by (*auto simp: space-embed-measure space-pair-measure*)
next
show $(\lambda(x, y). f x y) \circ \text{extract-int-pair} \in \text{measurable}$ (*stock-measure* (*PRODUCT INTEG INTEG*)) *M*
using *measurable-extract-int-pair* *f* **by** (*rule measurable*)
qed

lemma *map-int-pair-measurable-REAL*[*measurable*]:
assumes *g* \in *measurable* (*PRODUCT REAL REAL*) *M*
shows *map-int-pair* *f g* \in *measurable* (*PRODUCT REAL REAL*) *M*
proof (*subst measurable-cong*)
fix *w* **assume** *w* \in *space* (*PRODUCT REAL REAL*)
then show *map-int-pair* *f g w* = *g w*
by (*auto simp: space-embed-measure space-pair-measure map-int-pair-def*)
qed fact

lemma *map-real-pair-measurable*[*measurable*]:
assumes *f*: *case-prod* *f* \in *measurable* (*borel* \otimes_M *borel*) *M*
shows *map-real-pair* *f g* \in *measurable* (*PRODUCT REAL REAL*) *M*
proof (*subst measurable-cong*)
fix *w* **assume** *w* \in *space* (*PRODUCT REAL REAL*)
then show *map-real-pair* *f g w* = (*case-prod* *f* *o* *extract-real-pair*) *w*
by (*auto simp: space-embed-measure space-pair-measure*)
next
show $(\lambda(x, y). f x y) \circ \text{extract-real-pair} \in \text{measurable}$ (*stock-measure* (*PRODUCT*

REAL REAL)) M
using measurable-extract-real-pair f by (rule measurable)
qed

lemma count-space-IntVal-prod[simp]: $INTEG \otimes_M INTEG = \text{count-space } (\text{range IntVal} \times \text{range IntVal})$
by (auto intro!: pair-measure-countable)

lemma count-space-BoolVal-prod[simp]: $BOOL \otimes_M BOOL = \text{count-space } (\text{range BoolVal} \times \text{range BoolVal})$
by (auto intro!: pair-measure-countable)

lemma measurable-stock-measure-val-type:
assumes $f \in \text{measurable } M$ **(stock-measure t) x ∈ space M**
shows $\text{val-type } (f x) = t$
using assms by (auto dest!: measurable-space)

lemma singleton-in-stock-measure[simp]: $\text{val-type } v = t \implies \{v\} \in \text{sets } t$
proof (induction v arbitrary: t)
case (PairVal v1 v2)
have $A: \{<|v1, v2|>\} = (\lambda(v1, v2). <|v1, v2|>) \text{ ' } (\{v1\} \times \{v2\})$ **by simp**
from pair-measureI[OF PairVal.IH, OF refl refl] PairVal.premis[symmetric] show
?case
by (simp only: val-type.simps stock-measure.simps A in-sets-embed-measure)
qed (auto simp: sets-embed-measure)

lemma emeasure-stock-measure-singleton-finite[simp]:
 $\text{emeasure } (\text{stock-measure } (\text{val-type } v)) \{v\} \neq \infty$
proof (induction v)
case (RealVal r)
have $A: \{\text{RealVal } r\} = \text{RealVal } \text{ ' } \{r\}$ **by simp**
have $\{\text{RealVal } \text{ ' } \{r\} \in \text{sets } (\text{embed-measure lborel RealVal})$
by (rule in-sets-embed-measure) simp
thus ?case by (simp only: A val-type.simps stock-measure.simps emeasure-embed-measure
 $\text{inj-RealVal inj-vimage-image-eq) simp}$

next
case (PairVal v1 v2)
let $?M = \lambda x. \text{stock-measure } (\text{val-type } x)$
interpret $\text{sigma-finite-measure } \text{stock-measure } (\text{val-type } v2)$
by (rule sigma-finite-stock-measure)
have $A: \{<|v1, v2|>\} = (\lambda(v1, v2). <|v1, v2|>) \text{ ' } (\{v1\} \times \{v2\})$ **by simp**
have $B: \{v1\} \times \{v2\} \in ?M v1 \otimes_M ?M v2$
by (intro pair-measureI singleton-in-stock-measure) simp-all
hence $\text{emeasure } (?M (<|v1, v2|>)) \{<|v1, v2|>\} = \text{emeasure } (?M v1) \{v1\} * \text{emeasure } (?M v2) \{v2\}$
by (simp only: stock-measure.simps val-type.simps A emeasure-embed-measure-image inj-PairVal
 $\text{inj-vimage-image-eq emeasure-pair-measure-Times singleton-in-stock-measure B})$

with *PairVal.IH* **show** *?case* **by** (*simp* *add: ennreal-mult-eq-top-iff*)
qed *simp-all*

3.2 Measures on states

definition *state-measure* :: *vname set* \Rightarrow (*vname* \Rightarrow *pdf-type*) \Rightarrow *state measure*
where

state-measure *V* $\Gamma \equiv \Pi_M y \in V. \Gamma y$

lemma *state-measure-nonempty*[*simp*]: *space* (*state-measure* *V* Γ) $\neq \{\}$
by (*simp* *add: state-measure-def space-PiM PiE-eq-empty-iff*)

lemma *space-state-measure*: *space* (*state-measure* *V* Γ) = ($\Pi_E y \in V. \text{type-universe}(\Gamma y)$)
by (*simp* *add: state-measure-def space-PiM PiE-eq-empty-iff*)

lemma *state-measure-var-type*:

$\sigma \in \text{space}(\text{state-measure } V \Gamma) \implies x \in V \implies \text{val-type}(\sigma x) = \Gamma x$
by (*auto* *simp: state-measure-def space-PiM dest!: PiE-mem*)

lemma *merge-in-state-measure*:

$x \in \text{space}(\text{state-measure } A \Gamma) \implies y \in \text{space}(\text{state-measure } B \Gamma) \implies$
 $\text{merge } A B (x, y) \in \text{space}(\text{state-measure } (A \cup B) \Gamma)$ **unfolding** *state-measure-def*
by (*rule measurable-space, rule measurable-merge*) (*simp* *add: space-pair-measure*)

lemma *measurable-merge-stock*[*measurable (raw)*]:

$f \in N \rightarrow_M \text{state-measure } V \Gamma \implies g \in N \rightarrow_M \text{state-measure } V' \Gamma \implies$
 $(\lambda x. \text{merge } V V' (f x, g x)) \in N \rightarrow_M \text{state-measure } (V \cup V') \Gamma$
by (*auto* *simp: state-measure-def*)

lemma *comp-in-state-measure*:

assumes $\sigma \in \text{space}(\text{state-measure } V \Gamma)$
shows $\sigma \circ f \in \text{space}(\text{state-measure } (f \text{ - ' } V) (\Gamma \circ f))$
using *assms* **by** (*auto* *simp: state-measure-def space-PiM*)

lemma *sigma-finite-state-measure*[*intro*]:

finite *V* $\implies \text{sigma-finite-measure}(\text{state-measure } V \Gamma)$ **unfolding** *state-measure-def*
by (*auto* *intro!: product-sigma-finite.sigma-finite simp: product-sigma-finite-def*)

3.3 Equalities of measure embeddings

lemma *embed-measure-RealPairVal*:

stock-measure (*PRODUCT REAL REAL*) = *embed-measure lborel RealPairVal*

proof –

have [*simp*]: $(\lambda(x, y). <| x, y |>) \circ (\lambda(x, y). (\text{RealVal } x, \text{RealVal } y)) = \text{RealPairVal}$

unfolding *RealPairVal-def* **by** *auto*

have *stock-measure* (*PRODUCT REAL REAL*) =
 $\text{embed-measure}(\text{embed-measure } \text{lborel}(\lambda(x, y). (\text{RealVal } x, \text{RealVal } y)))$
(*case-prod RealVal*)

by (auto simp: embed-measure-prod sigma-finite-lborel lborel-prod)
 also have ... = embed-measure lborel RealPairVal
 by (subst embed-measure-comp) (auto intro!: injI)
 finally show ?thesis .
 qed

lemma embed-measure-IntPairVal:

stock-measure (PRODUCT INTEG INTEG) = count-space (range IntPairVal)

proof –

have [simp]: $(\lambda(x, y). \langle |x|, |y| \rangle) \text{ ‘ (range IntVal} \times \text{range IntVal) = range IntPairVal}$

by (auto simp: IntPairVal-def)

show ?thesis

using count-space-IntVal-prod by (auto simp: embed-measure-prod embed-measure-count-space)

qed

3.4 Monadic operations on values

definition return-val $x = \text{return (stock-measure (val-type } x)) x$

lemma sets-return-val[measurable-cong]: sets (return-val x) = sets (stock-measure (val-type x))

by (simp add: return-val-def)

lemma measurable-return-val[simp]:

return-val \in measurable (stock-measure t) (subprob-algebra (stock-measure t))

unfolding return-val-def[abs-def]

apply (subst measurable-cong)

apply (subst type-universe-type[THEN iffD1])

apply simp

apply (rule refl)

apply (rule return-measurable)

done

lemma bind-return-val:

assumes space $M \neq \{\}$ $f \in$ measurable M (stock-measure t')

shows $M \gg= (\lambda x. \text{return-val (} f x)) = \text{distr } M \text{ (stock-measure } t') f$

using assms

by (subst bind-return-distr[symmetric])

(auto simp: return-val-def intro!: bind-cong dest: measurable-stock-measure-val-type)

lemma bind-return-val':

assumes val-type $x = t$ $f \in$ measurable (stock-measure t) (stock-measure t')

shows return-val $x \gg= (\lambda x. \text{return-val (} f x)) = \text{return-val (} f x)$

proof –

have return-val $x \gg= (\lambda x. \text{return-val (} f x)) = \text{return (stock-measure } t') (f x)$

apply (subst bind-return-val, unfold return-val-def, simp)

apply (insert assms, simp cong: measurable-cong-sets) []

apply (subst distr-return, simp-all add: assms type-universe-def)

del: type-universe-type)

done
also from *assms(2)* **have** $f x \in \text{space } (\text{stock-measure } t')$
by (*rule measurable-space*)
(simp add: assms(1) type-universe-def del: type-universe-type)
hence $\text{return } (\text{stock-measure } t') (f x) = \text{return-val } (f x)$
by (*simp add: return-val-def*)
finally show *?thesis* .
qed

lemma *bind-return-val''*:
assumes $f \in \text{measurable } (\text{stock-measure } (\text{val-type } x)) (\text{subprob-algebra } M)$
shows $\text{return-val } x \ggg f = f x$
unfolding *return-val-def* **by** (*subst bind-return[OF assms]*) *simp-all*

lemma *bind-assoc-return-val*:
assumes *sets-M*: $\text{sets } M = \text{sets } (\text{stock-measure } t)$
assumes *Mf*: $f \in \text{measurable } (\text{stock-measure } t) (\text{stock-measure } t')$
assumes *Mg*: $g \in \text{measurable } (\text{stock-measure } t') (\text{stock-measure } t'')$
shows $(M \ggg (\lambda x. \text{return-val } (f x))) \ggg (\lambda x. \text{return-val } (g x)) =$
 $M \ggg (\lambda x. \text{return-val } (g (f x)))$

proof –
have $(M \ggg (\lambda x. \text{return-val } (f x))) \ggg (\lambda x. \text{return-val } (g x)) =$
 $M \ggg (\lambda x. \text{return-val } (f x)) \ggg (\lambda x. \text{return-val } (g x))$
apply (*subst bind-assoc*)
apply (*rule measurable-compose[OF - measurable-return-val]*)
apply (*subst measurable-cong-sets[OF sets-M refl]*, *rule Mf*)
apply (*rule measurable-compose[OF Mg measurable-return-val]*, *rule refl*)
done
also have $\dots = M \ggg (\lambda x. \text{return-val } (g (f x)))$
apply (*intro bind-cong refl*)
apply (*subst (asm) sets-eq-imp-space-eq[OF sets-M]*)
apply (*drule measurable-space[OF Mf]*)
apply (*subst bind-return-val'[where t = t' and t' = t'']*)
apply (*simp-all add: Mg*)
done
finally show *?thesis* .
qed

lemma *bind-return-val-distr*:
assumes *sets-M*: $\text{sets } M = \text{sets } (\text{stock-measure } t)$
assumes *Mf*: $f \in \text{measurable } (\text{stock-measure } t) (\text{stock-measure } t')$
shows $M \ggg \text{return-val } \circ f = \text{distr } M (\text{stock-measure } t') f$
proof –
have $M \ggg \text{return-val } \circ f = M \ggg \text{return } (\text{stock-measure } t') \circ f$
apply (*intro bind-cong refl*)
apply (*subst (asm) sets-eq-imp-space-eq[OF sets-M]*)
apply (*drule measurable-space[OF Mf]*)
apply (*simp add: return-val-def o-def*)

done
also have ... = *distr* *M* (*stock-measure* *t*) *f*
apply (*rule* *bind-return-distr*)
apply (*simp* *add: sets-eq-imp-space-eq*[*OF sets-M*])
apply (*subst* *measurable-cong-sets*[*OF sets-M refl*], *rule* *Mf*)
done
finally show ?*thesis* .
qed

3.5 Lifting of functions

definition *lift-RealVal* **where**

lift-RealVal *f* $\equiv \lambda$ *RealVal* *v* \Rightarrow *RealVal* (*f v*) | - \Rightarrow *RealVal* (*f 0*)

definition *lift-IntVal* **where**

lift-IntVal *f* $\equiv \lambda$ *IntVal* *v* \Rightarrow *IntVal* (*f v*) | - \Rightarrow *IntVal* (*f 0*)

definition *lift-RealIntVal* **where**

lift-RealIntVal *f g* $\equiv \lambda$ *IntVal* *v* \Rightarrow *IntVal* (*f v*) | *RealVal* *v* \Rightarrow *RealVal* (*g v*)

definition *lift-RealIntVal2* **where**

lift-RealIntVal2 *f g* \equiv
map-int-pair (λ *a b*. *IntVal* (*f a b*))
(*map-real-pair* (λ *a b*. *RealVal* (*g a b*))
id)

definition *lift-Comp* **where**

lift-Comp *f g* \equiv *map-int-pair* (λ *a b*. *BoolVal* (*f a b*))
(*map-real-pair* (λ *a b*. *BoolVal* (*g a b*))
(λ -. *FALSE*))

lemma *lift-RealVal-eq*: *lift-RealVal* *f* (*RealVal* *x*) = *RealVal* (*f x*)
by (*simp* *add: lift-RealVal-def*)

lemma *lift-RealIntVal-Real*:

x \in *space* (*stock-measure* *REAL*) \implies *lift-RealIntVal* *f g* *x* = *lift-RealVal* *g x*
by (*auto simp: space-embed-measure lift-RealIntVal-def lift-RealVal-def*)

lemma *lift-RealIntVal-Int*:

x \in *space* (*stock-measure* *INTEG*) \implies *lift-RealIntVal* *f g* *x* = *lift-IntVal* *f x*
by (*auto simp: space-embed-measure lift-RealIntVal-def lift-IntVal-def*)

declare *stock-measure.simps*[*simp del*]

lemma *measurable-lift-RealVal*[*measurable*]:

assumes [*measurable*]: *f* \in *borel-measurable borel*
shows *lift-RealVal* *f* \in *measurable REAL REAL*
unfolding *lift-RealVal-def*
by (*auto intro!: val-case-stock-measurable*)

lemma *measurable-lift-IntVal*[*simp*]: *lift-IntVal* *f* \in *range* *IntVal* \rightarrow *range* *IntVal*

by (auto simp: lift-IntVal-def)

lemma measurable-lift-IntVal'[measurable]: lift-IntVal $f \in$ measurable INTEG INTEG
 unfolding lift-IntVal-def
 by (auto intro!: val-case-stock-measurable)

lemma split-apply: (case x of (a, b) \Rightarrow f a b) $y =$ (case x of (a, b) \Rightarrow f a b y)
 by (cases x) simp

lemma measurable-lift-Comp-RealVal[measurable]:
 assumes [measurable]: Measurable.pred (borel \otimes_M borel) (case-prod g)
 shows lift-Comp $f g \in$ measurable (PRODUCT REAL REAL) BOOL
 unfolding lift-Comp-def by measurable

lemma measurable-lift-Comp-IntVal[simp]:
 lift-Comp $f g \in$ measurable (PRODUCT INTEG INTEG) BOOL
 unfolding lift-Comp-def
 by (auto intro!: val-case-stock-measurable)

lemma measurable-lift-RealIntVal-IntVal[simp]: lift-RealIntVal $f g \in$ range IntVal
 \rightarrow range IntVal
 by (auto simp: embed-measure-count-space lift-RealIntVal-def)

lemma measurable-lift-RealIntVal-IntVal'[measurable]:
 lift-RealIntVal $f g \in$ measurable INTEG INTEG
 by (auto simp: lift-RealIntVal-def intro!: val-case-stock-measurable)

lemma measurable-lift-RealIntVal-RealVal[measurable]:
 assumes [measurable]: $g \in$ borel-measurable borel
 shows lift-RealIntVal $f g \in$ measurable REAL REAL
 unfolding lift-RealIntVal-def
 by (auto intro!: val-case-stock-measurable)

lemma measurable-lift-RealIntVal2-IntVal[measurable]:
 lift-RealIntVal2 $f g \in$ measurable (PRODUCT INTEG INTEG) INTEG
 unfolding lift-RealIntVal2-def
 by (auto intro!: val-case-stock-measurable)

lemma measurable-lift-RealIntVal2-RealVal[measurable]:
 assumes [measurable]: case-prod $g \in$ borel-measurable (borel \otimes_M borel)
 shows lift-RealIntVal2 $f g \in$ measurable (PRODUCT REAL REAL) REAL
 unfolding lift-RealIntVal2-def by measurable

lemma distr-lift-RealVal:
 fixes f
 assumes Mf [measurable]: $f \in$ borel-measurable borel
 assumes pdens: has-subprob-density M (stock-measure REAL) δ
 assumes dens': $\bigwedge M \delta$. has-subprob-density M lborel $\delta \implies$ has-density (distr M

borel f *lborel (g δ)*
defines $N \equiv \text{distr } M \text{ (stock-measure } REAL) \text{ (lift-RealVal } f)$
shows *has-density N (stock-measure REAL) (g (λx. δ (RealVal x)) ∘ extract-real)*
proof (*rule has-densityI*)
from *assms(2)* **have** *dens: has-density M (stock-measure REAL) δ*
unfolding *has-subprob-density-def* **by** *simp*
from *dens* **have** *sets-M[measurable-cong]: sets M = sets REAL* **by** (*auto dest: has-densityD*)

note *measurable-embed-measure1[measurable del]*

have $N = \text{distr } M \text{ (stock-measure } REAL) \text{ (lift-RealVal } f)$ **by** (*simp add: N-def*)
also have $\dots = \text{distr } M \text{ (stock-measure } REAL) \text{ (RealVal } \circ f \circ \text{extract-real})$
using *sets-eq-imp-space-eq[OF sets-M]*
by (*intro distr-cong*) (*auto simp: lift-RealVal-def stock-measure.simps space-embed-measure*)
also have $\dots = \text{distr (distr (distr } M \text{ lborel extract-real) borel } f) \text{ (stock-measure } REAL) \text{ RealVal}$
by (*subst distr-distr*)
(*simp-all add: distr-distr[OF - measurable-comp[OF - Mf]] comp-assoc*)
also have *dens'': has-density (distr (distr M lborel extract-real) borel f) lborel (g (δ ∘ RealVal))*
by (*intro dens' has-subprob-density-embed-measure''*) (*insert pdens, simp-all add: extract-real-def stock-measure.simps*)
hence $\text{distr (distr } M \text{ lborel extract-real) borel } f = \text{density lborel (g (δ ∘ RealVal))}$
by (*rule has-densityD*)
also have $\text{distr } \dots \text{ (stock-measure } REAL) \text{ RealVal} = \text{embed-measure } \dots \text{ RealVal}$ (*is - = ?M*)
by (*subst embed-measure-eq-distr[OF inj-RealVal], intro distr-cong*)
(*simp-all add: sets-embed-measure stock-measure.simps*)
also have $\dots = \text{density (embed-measure lborel RealVal) (g (λx. δ (RealVal x))} \circ \text{extract-real}$)
using *dens''[unfolded o-def]*
apply (*subst density-embed-measure', simp, simp add: extract-real-def*)
apply (*erule has-densityD, simp add: o-def*)
done
finally show $N = \text{density (stock-measure } REAL) \text{ (g (λx. δ (RealVal x))} \circ \text{extract-real}$)
by (*simp add: stock-measure.simps*)

from *dens''[unfolded o-def, THEN has-densityD(1)] measurable-extract-real*
show $g \text{ (λx. δ (RealVal x))} \circ \text{extract-real} \in \text{borel-measurable (stock-measure } REAL)$
by (*intro measurable-comp*) *auto*
qed (*subst space-stock-measure, simp*)

lemma *distr-lift-IntVal:*

fixes *f*

assumes *pdens: has-density M (stock-measure INTEG) δ*

assumes *dens': $\bigwedge M \delta. \text{has-density } M \text{ (count-space UNIV) } \delta \implies$*

```

has-density (distr M (count-space UNIV) f) (count-space
UNIV) (g δ)
defines N ≡ distr M (stock-measure INTEG) (lift-IntVal f)
shows has-density N (stock-measure INTEG) (g (λx. δ (IntVal x)) ∘ extract-int)
proof (rule has-densityI)
let ?R = count-space UNIV and ?S = count-space (range IntVal)
have Mf: f ∈ measurable ?R ?R by simp
from assms(1) have dens: has-density M (stock-measure INTEG) δ
unfolding has-subprob-density-def by simp
from dens have sets-M[measurable-cong]: sets M = sets INTEG by (auto dest!:
has-densityD(2))

have N = distr M (stock-measure INTEG) (lift-IntVal f) by (simp add: N-def)
also have ... = distr M (stock-measure INTEG) (IntVal ∘ f ∘ extract-int)
using sets-eq-imp-space-eq[OF sets-M]
by (intro distr-cong) (auto simp: space-embed-measure lift-IntVal-def stock-measure.simps)
also have ... = distr (distr (distr M ?R extract-int) ?R f) (stock-measure IN-
TEG) IntVal
by (subst distr-distr) (simp-all add: distr-distr[OF - measurable-comp[OF - Mf]]
comp-assoc)
also have dens'': has-density (distr (distr M ?R extract-int) ?R f) ?R (g (δ ∘
IntVal))
by (intro dens' has-density-embed-measure'')
(insert dens, simp-all add: extract-int-def embed-measure-count-space stock-measure.simps)
hence distr (distr M ?R extract-int) ?R f = density ?R (g (δ ∘ IntVal))
by (rule has-densityD)
also have distr ... (stock-measure INTEG) IntVal = embed-measure ... IntVal
(is - = ?M)
by (subst embed-measure-eq-distr[OF inj-IntVal], intro distr-cong)
(auto simp: sets-embed-measure subset-image-iff stock-measure.simps)
also have ... = density (embed-measure ?R IntVal) (g (λx. δ (IntVal x)) ∘ ex-
tract-int)
using dens''[unfolded o-def]
apply (subst density-embed-measure', simp, simp add: extract-int-def)
apply (erule has-densityD, simp add: o-def)
done
finally show N = density (stock-measure INTEG) (g (λx. δ (IntVal x)) ∘ ex-
tract-int)
by (simp add: embed-measure-count-space stock-measure.simps)

from dens''[unfolded o-def]
show g (λx. δ (IntVal x)) ∘ extract-int ∈ borel-measurable (stock-measure
INTEG)
by (simp add: embed-measure-count-space stock-measure.simps)
qed (subst space-stock-measure, simp)

lemma distr-lift-RealPairVal:
fixes f f' g
assumes Mf[measurable]: case-prod f ∈ borel-measurable borel

```

assumes $pdens$: *has-subprob-density* M (*stock-measure* ($PRODUCT$ $REAL$ $REAL$))
 δ
assumes $dens'$: $\bigwedge M \delta$. *has-subprob-density* M *lborel* $\delta \implies$ *has-density* (*distr* M *borel* (*case-prod* f)) *lborel* ($g \delta$)
defines $N \equiv$ *distr* M (*stock-measure* $REAL$) (*lift-RealIntVal2* $f' f$)
shows *has-density* N (*stock-measure* $REAL$) ($g (\lambda x. \delta (RealPairVal x)) \circ$ *extract-real*)
proof (*rule has-densityI*)
from $assms(2)$ **have** $dens$: *has-density* M (*stock-measure* ($PRODUCT$ $REAL$ $REAL$)) δ
unfolding *has-subprob-density-def* **by** *simp*
have $sets\text{-}M$ [*measurable-cong*]: $sets\ M = sets$ (*stock-measure* ($PRODUCT$ $REAL$ $REAL$))
by (*auto simp: has-subprob-densityD[OF pdens]*)

have $N =$ *distr* M (*stock-measure* $REAL$) (*lift-RealIntVal2* $f' f$) **by** (*simp add: N-def*)
also have $\dots =$ *distr* M (*stock-measure* $REAL$) (*RealVal* \circ *case-prod* f \circ *extract-real-pair*)
using *sets-eq-imp-space-eq[OF sets-M]*
by (*intro distr-cong*) (*auto simp: lift-RealIntVal2-def space-embed-measure space-pair-measure stock-measure.simps*)
also have $\dots =$ *distr* (*distr* (*distr* M *lborel* *extract-real-pair*) *borel* (*case-prod* f)) $REAL$ $RealVal$
by (*subst distr-distr*) (*simp-all add: distr-distr[OF - measurable-comp[OF - Mf]] comp-assoc*)
also have $dens''$: *has-density* (*distr* (*distr* M *lborel* *extract-real-pair*) *borel* (*case-prod* f)) *lborel*
 $(g (\delta \circ RealPairVal))$ **using** *inj-RealPairVal embed-measure-RealPairVal*
by (*intro dens' has-subprob-density-embed-measure''*)
 $(insert\ pdens,\ simp\text{-}all\ add:\ RealPairVal\text{-}def\ split:\ prod.split)$
hence *distr* (*distr* M *lborel* *extract-real-pair*) *borel* (*case-prod* f) $=$
density *lborel* ($g (\delta \circ RealPairVal)$) **by** (*rule has-densityD*)
also have *distr* \dots (*stock-measure* $REAL$) $RealVal =$ *embed-measure* \dots $RealVal$
 $(is\ - = ?M)$
by (*subst embed-measure-eq-distr[OF inj-RealVal], intro distr-cong*)
 $(simp\text{-}all\ add:\ sets\text{-}embed\text{-}measure\ stock\text{-}measure.simps)$
also have $\dots =$ *density* (*embed-measure* *lborel* $RealVal$) ($g (\lambda x. \delta (RealPairVal x)) \circ$ *extract-real*)
using $dens''$ [*unfolded o-def*]
by (*subst density-embed-measure', simp, simp add: extract-real-def*)
 $(erule\ has\text{-}densityD,\ simp\ add:\ o\text{-}def)$
finally show $N =$ *density* (*stock-measure* $REAL$) ($g (\lambda x. \delta (RealPairVal x)) \circ$ *extract-real*)
by (*simp add: stock-measure.simps*)

from $dens''$ [*unfolded o-def*]
show $g (\lambda x. \delta (RealPairVal x)) \circ$ *extract-real* \in *borel-measurable* (*stock-measure* $REAL$)

hence $\text{distr } (\text{distr } M \text{ (count-space UNIV) extract-int-pair) } ?R \text{ (case-prod } f) =$
 $\text{density } ?R \text{ (} g \text{ (} \delta \circ \text{IntPairVal} \text{)) } \mathbf{by} \text{ (rule has-densityD)}$
also have $\text{distr } \dots \text{ (stock-measure INTEG) IntVal} = \text{embed-measure } \dots \text{ IntVal}$
 (is - = ?M)
by $\text{(subst embed-measure-eq-distr[OF inj-IntVal], intro distr-cong)}$
 $\text{(auto simp: sets-embed-measure subset-image-iff stock-measure.simps)}$
also have $\dots = \text{density (embed-measure } ?R \text{ IntVal) (} g \text{ (} \lambda x. \delta \text{ (IntPairVal } x) \text{))} \circ$
 extract-int)
using $\text{dens''[unfolded o-def]}$
by $\text{(subst density-embed-measure', simp, simp add: extract-int-def)}$
 $\text{(erule has-densityD, simp add: o-def)}$
finally show $N = \text{density (stock-measure INTEG) (} g \text{ (} \lambda x. \delta \text{ (IntPairVal } x) \text{))} \circ$
 extract-int)
by $\text{(simp add: embed-measure-count-space stock-measure.simps)}$

from $\text{dens''[unfolded o-def]}$
show $g \text{ (} \lambda x. \delta \text{ (IntPairVal } x) \text{))} \circ \text{extract-int} \in \text{borel-measurable (stock-measure}$
 INTEG)
by $\text{(simp add: embed-measure-count-space stock-measure.simps)}$
qed $\text{(subst space-stock-measure, simp)}$

end

theory *PDF-Semantics*
imports *PDF-Values*
begin

lemma *measurable-subprob-algebra-density:*
assumes *sigma-finite-measure N*
assumes $\text{space } N \neq \{\}$
assumes $\text{[measurable]: case-prod } f \in \text{borel-measurable } (M \otimes_M N)$
assumes $\bigwedge x. x \in \text{space } M \implies (\int^+ y. f \ x \ y \ \partial N) \leq 1$
shows $(\lambda x. \text{density } N \text{ (} f \ x)) \in \text{measurable } M \text{ (subprob-algebra } N)$
proof $\text{(rule measurable-subprob-algebra)}$
fix x **assume** $x \in \text{space } M$
with *assms* **show** *subprob-space (density N (f x))*
by $\text{(intro subprob-spaceI) (auto simp: emeasure-density cong: nn-integral-cong')}$
next
interpret *sigma-finite-measure N* **by** *fact*
fix X **assume** $X \in \text{sets } N$
hence $(\lambda x. (\int^+ y. f \ x \ y * \text{indicator } X \ y \ \partial N)) \in \text{borel-measurable } M$ **by** *simp*
moreover from X **and** *assms* **have**
 $\bigwedge x. x \in \text{space } M \implies \text{emeasure (density } N \text{ (} f \ x)) \ X = (\int^+ y. f \ x \ y * \text{indicator}$
 $\text{X } y \ \partial N)$
by $\text{(simp add: emeasure-density)}$
ultimately show $(\lambda x. \text{emeasure (density } N \text{ (} f \ x)) \ X) \in \text{borel-measurable } M$
by $\text{(simp only: cong: measurable-cong)}$
qed *simp-all*

4 Built-in Probability Distributions

4.1 Bernoulli

definition *bernoulli-density* :: *real* \Rightarrow *bool* \Rightarrow *ennreal* **where**
bernoulli-density *p b* = (if *p* \in {0..1} then (if *b* then *p* else 1 - *p*) else 0)

definition *bernoulli* :: *val* \Rightarrow *val measure* **where**
bernoulli *p* = *density* *BOOL* (*bernoulli-density* (*extract-real* *p*) *o* *extract-bool*)

lemma *measurable-bernoulli-density*[*measurable*]:
case-prod *bernoulli-density* \in *borel-measurable* (*borel* \otimes_M *count-space UNIV*)
unfolding *bernoulli-density-def*[*abs-def*] **by** *measurable*

lemma *measurable-bernoulli*[*measurable*]: *bernoulli* \in *measurable REAL* (*subprob-algebra* *BOOL*)

unfolding *bernoulli-def*[*abs-def*]
by (*auto intro!*: *measurable-subprob-algebra-density*
simp: *measurable-split-conv nn-integral-BoolVal* *bernoulli-density-def*
ennreal-plus[*symmetric*]
simp del: *ennreal-plus*)

4.2 Uniform

definition *uniform-real-density* :: *real* \times *real* \Rightarrow *real* \Rightarrow *ennreal* **where**
uniform-real-density \equiv $\lambda(a,b) x.$ *ennreal* (if $a < b \wedge x \in \{a..b\}$ then *inverse* ($b - a$) else 0)

definition *uniform-int-density* :: *int* \times *int* \Rightarrow *int* \Rightarrow *ennreal* **where**
uniform-int-density \equiv $\lambda(a,b) x.$ (if $x \in \{a..b\}$ then *inverse* ($\text{nat } (b - a + 1)$) else 0)

lemma *measurable-uniform-density-int*[*measurable*]:
(*case-prod* *uniform-int-density*)
 \in *borel-measurable* ((*count-space UNIV* \otimes_M *count-space UNIV*) \otimes_M *count-space UNIV*)
by (*simp add*: *pair-measure-countable*)

lemma *measurable-uniform-density-real*[*measurable*]:
(*case-prod* *uniform-real-density*) \in *borel-measurable* (*borel* \otimes_M *borel*)

proof –

have (*case-prod* *uniform-real-density*) =
($\lambda x.$ *uniform-real-density* (*fst* (*fst* *x*), *snd* (*fst* *x*)) (*snd* *x*))
by (*rule ext*) (*simp split*: *prod.split*)
also have ... \in *borel-measurable* (*borel* \otimes_M *borel*)
unfolding *uniform-real-density-def*
by (*simp only*: *prod.case*) (*simp add*: *borel-prod*[*symmetric*])
finally show *?thesis* .

qed

definition *uniform-int* :: *val* \Rightarrow *val measure* **where**
uniform-int = *map-int-pair* ($\lambda l u.$ *density* *INTEG* (*uniform-int-density* (*l,u*) *o*
extract-int)) ($\lambda.$ *undefined*)

definition *uniform-real* :: *val* \Rightarrow *val measure* **where**
uniform-real = *map-real-pair* ($\lambda l u.$ *density* *REAL* (*uniform-real-density* (*l,u*) *o*
extract-real)) ($\lambda.$ *undefined*)

lemma *if-bounded*: (*if* $a \leq i \wedge i \leq b$ *then* *v* *else* *0*) = (*v*::*real*) * *indicator* {*a .. b*}
i
by *auto*

lemma *measurable-uniform-int*[*measurable*]:
uniform-int \in *measurable* (*PRODUCT* *INTEG* *INTEG*) (*subprob-algebra* *IN-*
TEG)

unfolding *uniform-int-def*

proof (*rule measurable measurable-subprob-algebra-density*)+
fix *x* :: *int* \times *int*

show integral^N *INTEG* (*uniform-int-density* (*fst x, snd x*) *o* *extract-int*) ≤ 1

proof *cases*

assume *fst x* \leq *snd x* **then show** *?thesis*

by (*cases x*)

(*simp add: uniform-int-density-def comp-def nn-integral-IntVal nn-integral-cmult*
nn-integral-set-ennreal[symmetric] ennreal-of-nat-eq-real-of-nat
if-bounded[where 'a=int] ennreal-mult[symmetric]
del: ennreal-plus)

qed (*simp add: uniform-int-density-def comp-def split-beta' if-bounded[where*
'a=int])

qed (*auto simp: comp-def*)

lemma *density-cong'*:

($\bigwedge x. x \in \text{space } M \implies f x = g x$) \implies *density* *M* *f* = *density* *M* *g*

unfolding *density-def*

by (*auto dest: sets.sets-into-space intro!: nn-integral-cong measure-of-eq*)

lemma *measurable-uniform-real*[*measurable*]:

uniform-real \in *measurable* (*PRODUCT* *REAL* *REAL*) (*subprob-algebra* *REAL*)

unfolding *uniform-real-def*

proof (*rule measurable measurable-subprob-algebra-density*)+

fix *x* :: *real* \times *real*

obtain *l u* **where** [*simp*]: *x* = (*l, u*)

by (*cases x*) *auto*

show ($\int^+ y.$ (*uniform-real-density* (*fst x, snd x*) *o* *extract-real*) *y* ∂ *REAL*) ≤ 1

proof *cases*

assume *l* < *u* **then show** *?thesis*

by (*simp add: nn-integral-RealVal uniform-real-density-def if-bounded nn-integral-cmult*
nn-integral-set-ennreal[symmetric] ennreal-mult[symmetric])

qed (*simp add: uniform-real-density-def comp-def*)

qed (auto simp: comp-def borel-prod)

4.3 Gaussian

definition gaussian-density :: real × real ⇒ real ⇒ ennreal **where**

gaussian-density ≡
λ(m,s) x. (if s > 0 then exp (-(x - m)² / (2 * s²)) / sqrt (2 * pi * s²) else 0)

lemma measurable-gaussian-density[measurable]:

case-prod gaussian-density ∈ borel-measurable (borel ⊗_M borel)

proof–

have case-prod gaussian-density =

(λ(x,y). (if snd x > 0 then exp (-(y - fst x)² / (2 * snd x²)) /
sqrt (2 * pi * snd x²) else 0))

unfolding gaussian-density-def **by** (intro ext) (simp split: prod.split)

also have ... ∈ borel-measurable (borel ⊗_M borel)

by (simp add: borel-prod[symmetric])

finally show ?thesis .

qed

definition gaussian :: val ⇒ val measure **where**

gaussian = map-real-pair (λm s. density REAL (gaussian-density (m,s) o extract-real)) undefined

lemma measurable-gaussian[measurable]: gaussian ∈ measurable (PRODUCT REAL REAL) (subprob-algebra REAL)

unfolding gaussian-def

proof (rule measurable-measurable-subprob-algebra-density)+

fix x :: real × real

show integral^N (stock-measure REAL) (gaussian-density (fst x, snd x) o extract-real) ≤ 1

proof cases

assume snd x > 0

then have integral^N lborel (gaussian-density x) = (∫⁺y. normal-density (fst x) (snd x) y ∂lborel)

by (auto simp add: gaussian-density-def normal-density-def split-beta' intro!: nn-integral-cong)

also have ... = 1

using ⟨snd x > 0⟩

by (subst nn-integral-eq-integral) (auto intro!: normal-density-nonneg)

finally show ?thesis

by (cases x) (simp add: nn-integral-RealVal comp-def)

next

assume ¬ snd x > 0 **then show** ?thesis

by (cases x)

(simp add: nn-integral-RealVal comp-def gaussian-density-def zero-ennreal-def[symmetric])

qed

qed (auto simp: comp-def borel-prod)

4.4 Poisson

definition *poisson-density'* :: *real* \Rightarrow *int* \Rightarrow *ennreal* **where**

poisson-density' rate *k* = *pmf* (*poisson-pmf* rate) (*nat k*) * *indicator* ($\{0 <..\}$ \times $\{0..\}$) (*rate*, *k*)

lemma *measurable-poisson-density'*[*measurable*]:

case-prod poisson-density' \in borel-measurable (borel \otimes_M count-space UNIV)

proof –

have *case-prod poisson-density' =*

*(λ (rate, k). rate \wedge nat k / real-of-nat (fact (nat k)) * exp (–rate) * indicator ($\{0 <..\}$ \times $\{0..\}$) (rate, k))*

by (*auto split: split-indicator simp: fun-eq-iff poisson-density'-def*)

then show *?thesis*

by *simp*

qed

definition *poisson* :: *val* \Rightarrow *val* *measure* **where**

poisson rate = *density* *INTEG* (*poisson-density'* (*extract-real* rate) *o* *extract-int*)

lemma *measurable-poisson*[*measurable*]: *poisson* \in *measurable* *REAL* (*subprob-algebra* *INTEG*)

unfolding *poisson-def*[*abs-def*]

proof (*rule measurable measurable-subprob-algebra-density*)+

fix *r* :: *real*

have [*simp*]: *nat* ' $\{0..\}$ = *UNIV*

by (*auto simp: image-iff intro!: bexI[of - int x for x]*)

{ **assume** $0 < r$

then have ($\int^+ x.$ *ennreal* ($r \wedge$ *nat* *x* * exp (–*r*) * *indicator* ($\{0 <..\}$ \times $\{0..\}$) (*r*, *x*) / (fact (*nat* *x*))) ∂ *count-space* *UNIV*)

= ($\int^+ x.$ *ennreal* (*pmf* (*poisson-pmf* *r*) (*nat* *x*)) ∂ *count-space* $\{0 ..\}$)

by (*auto intro!: nn-integral-cong simp add: nn-integral-count-space-indicator split: split-indicator*)

also have ... = 1

using *measure-pmf.emmeasure-space-1*[*of poisson-pmf r*]

by (*subst nn-integral-pmf'*) (*auto simp: inj-on-def*)

finally have ($\int^+ x.$ *ennreal* ($r \wedge$ *nat* *x* * exp (–*r*) * *indicator* ($\{0 <..\}$ \times $\{0..\}$) (*r*, *x*) / (fact (*nat* *x*))) ∂ *count-space* *UNIV*) = 1

. }

then show *integral*^{*N*} *INTEG* (*poisson-density'* *r* *o* *extract-int*) ≤ 1

by (*cases* $0 < r$)

(*auto simp: nn-integral-IntVal poisson-density'-def zero-ennreal-def[symmetric]*)

qed (*auto simp: comp-def*)

5 Source Language Syntax and Semantics

5.1 Expressions

class *expr* = **fixes** *free-vars* :: 'a ⇒ *vname* set

datatype *pdf-dist* = *Bernoulli* | *UniformInt* | *UniformReal* | *Poisson* | *Gaussian*

datatype *pdf-operator* = *Fst* | *Snd* | *Add* | *Mult* | *Minus* | *Less* | *Equals* | *And* | *Not* | *Or* | *Pow* | *Sqrt* | *Exp* | *Ln* | *Fact* | *Inverse* | *Pi* | *Cast pdf-type*

datatype *expr* =
 Var vname
 | *Val val*
 | *LetVar expr expr* (⟨*LET* - *IN* -> [0, 60] 61)
 | *Operator pdf-operator expr* (**infixl** ⟨*\$\$*⟩ 999)
 | *Pair expr expr* (⟨*<-* , *->*⟩ [0, 60] 1000)
 | *Random pdf-dist expr*
 | *IfThenElse expr expr expr* (⟨*IF* - *THEN* - *ELSE* -> [0, 0, 70] 71)
 | *Fail pdf-type*

type-synonym *tyenv* = *vname* ⇒ *pdf-type*

instantiation *expr* :: *expr*

begin

primrec *free-vars-expr* :: *expr* ⇒ *vname* set **where**
 free-vars-expr (*Var x*) = {*x*}
 | *free-vars-expr* (*Val -*) = {}
 | *free-vars-expr* (*LetVar e1 e2*) = *free-vars-expr e1* ∪ *Suc -* ' *free-vars-expr e2*
 | *free-vars-expr* (*Operator - e*) = *free-vars-expr e*
 | *free-vars-expr* (⟨*e1*, *e2*⟩) = *free-vars-expr e1* ∪ *free-vars-expr e2*
 | *free-vars-expr* (*Random - e*) = *free-vars-expr e*
 | *free-vars-expr* (*IF b THEN e1 ELSE e2*) =
 free-vars-expr b ∪ *free-vars-expr e1* ∪ *free-vars-expr e2*
 | *free-vars-expr* (*Fail -*) = {}

instance ..
end

primrec *free-vars-expr-code* :: *expr* ⇒ *vname* set **where**
 free-vars-expr-code (*Var x*) = {*x*}
 | *free-vars-expr-code* (*Val -*) = {}
 | *free-vars-expr-code* (*LetVar e1 e2*) =
 free-vars-expr-code e1 ∪ (λ*x*. *x* - 1) ' (*free-vars-expr-code e2* - {0})
 | *free-vars-expr-code* (*Operator - e*) = *free-vars-expr-code e*
 | *free-vars-expr-code* (⟨*e1*, *e2*⟩) = *free-vars-expr-code e1* ∪ *free-vars-expr-code e2*
 | *free-vars-expr-code* (*Random - e*) = *free-vars-expr-code e*
 | *free-vars-expr-code* (*IF b THEN e1 ELSE e2*) =

$free\text{-vars-expr-code } b \cup free\text{-vars-expr-code } e1 \cup free\text{-vars-expr-code } e2$
 $| free\text{-vars-expr-code } (Fail \ -) = \{\}$

lemma $free\text{-vars-expr-code}[code]:$

$free\text{-vars } (e::expr) = free\text{-vars-expr-code } e$

proof –

have $\bigwedge A. Suc \ - \ 'A = (\lambda x. x - 1) \ ' (A - \{0\})$ **by force**

thus $?thesis$ **by** (induction e) $simp\text{-all}$

qed

primrec $dist\text{-param-type}$ **where**

$dist\text{-param-type } Bernoulli = REAL$

$| dist\text{-param-type } Poisson = REAL$

$| dist\text{-param-type } Gaussian = PRODUCT REAL REAL$

$| dist\text{-param-type } UniformInt = PRODUCT INTEG INTEG$

$| dist\text{-param-type } UniformReal = PRODUCT REAL REAL$

primrec $dist\text{-result-type}$ **where**

$dist\text{-result-type } Bernoulli = BOOL$

$| dist\text{-result-type } UniformInt = INTEG$

$| dist\text{-result-type } UniformReal = REAL$

$| dist\text{-result-type } Poisson = INTEG$

$| dist\text{-result-type } Gaussian = REAL$

primrec $dist\text{-measure} :: pdf\text{-dist} \Rightarrow val \Rightarrow val\ measure$ **where**

$dist\text{-measure } Bernoulli = bernoulli$

$| dist\text{-measure } UniformInt = uniform\text{-int}$

$| dist\text{-measure } UniformReal = uniform\text{-real}$

$| dist\text{-measure } Poisson = poisson$

$| dist\text{-measure } Gaussian = gaussian$

lemma $measurable\text{-dist-measure}[measurable]:$

$dist\text{-measure } d \in measurable (dist\text{-param-type } d) (subprob\text{-algebra } (dist\text{-result-type } d))$

by (cases d) $simp\text{-all}$

lemma $sets\text{-dist-measure}[simp]:$

$val\text{-type } x = dist\text{-param-type } dst \Longrightarrow$

$sets (dist\text{-measure } dst x) = sets (stock\text{-measure } (dist\text{-result-type } dst))$

by (rule $sets\text{-kernel}[OF measurable\text{-dist-measure}]$) $simp$

lemma $space\text{-dist-measure}[simp]:$

$val\text{-type } x = dist\text{-param-type } dst \Longrightarrow$

$space (dist\text{-measure } dst x) = type\text{-universe } (dist\text{-result-type } dst)$

by (subst $space\text{-stock-measure}[symmetric]$) (intro $sets\text{-eq-imp-space-eq sets\text{-dist-measure}$)

primrec $dist\text{-dens} :: pdf\text{-dist} \Rightarrow val \Rightarrow val \Rightarrow ennreal$ **where**

$dist\text{-dens } Bernoulli x y = bernoulli\text{-density } (extract\text{-real } x) (extract\text{-bool } y)$

| *dist-dens UniformInt* $x\ y = \text{uniform-int-density } (\text{extract-int-pair } x) (\text{extract-int } y)$
| *dist-dens UniformReal* $x\ y = \text{uniform-real-density } (\text{extract-real-pair } x) (\text{extract-real } y)$
| *dist-dens Gaussian* $x\ y = \text{gaussian-density } (\text{extract-real-pair } x) (\text{extract-real } y)$
| *dist-dens Poisson* $x\ y = \text{poisson-density}' (\text{extract-real } x) (\text{extract-int } y)$

lemma *measurable-dist-dens*[*measurable*]:

assumes $f \in \text{measurable } M (\text{stock-measure } (\text{dist-param-type } dst)) (\text{is } - \in \text{measurable } M ?N)$

assumes $g \in \text{measurable } M (\text{stock-measure } (\text{dist-result-type } dst)) (\text{is } - \in \text{measurable } M ?R)$

shows $(\lambda x. \text{dist-dens } dst (f\ x) (g\ x)) \in \text{borel-measurable } M$

apply (*rule measurable-Pair-compose-split*[*of dist-dens dst, OF - assms*])

apply (*subst dist-dens-def, cases dst, simp-all*)

done

lemma *dist-measure-has-density*:

$v \in \text{type-universe } (\text{dist-param-type } dst) \implies$

$\text{has-density } (\text{dist-measure } dst\ v) (\text{stock-measure } (\text{dist-result-type } dst)) (\text{dist-dens } dst\ v)$

proof (*intro has-densityI*)

fix v **assume** $v \in \text{type-universe } (\text{dist-param-type } dst)$

thus $\text{dist-measure } dst\ v = \text{density } (\text{stock-measure } (\text{dist-result-type } dst)) (\text{dist-dens } dst\ v)$

by (*cases dst*)

(*auto simp: bernoulli-def uniform-int-def uniform-real-def poisson-def gaussian-def*)

intro!: density-cong' elim!: PROD-E REAL-E INTEG-E)

qed *simp-all*

lemma *subprob-space-dist-measure*:

$v \in \text{type-universe } (\text{dist-param-type } dst) \implies \text{subprob-space } (\text{dist-measure } dst\ v)$

using *subprob-space-kernel*[*OF measurable-dist-measure, of v dst*] **by** *simp*

lemma *dist-measure-has-subprob-density*:

$v \in \text{type-universe } (\text{dist-param-type } dst) \implies$

$\text{has-subprob-density } (\text{dist-measure } dst\ v) (\text{stock-measure } (\text{dist-result-type } dst)) (\text{dist-dens } dst\ v)$

unfolding *has-subprob-density-def*

by (*auto intro: subprob-space-dist-measure dist-measure-has-density*)

lemma *dist-dens-integral-space*:

assumes $v \in \text{type-universe } (\text{dist-param-type } dst)$

shows $(\int^+ u. \text{dist-dens } dst\ v\ u\ \partial \text{stock-measure } (\text{dist-result-type } dst)) \leq 1$

proof –

let $?M = \text{density } (\text{stock-measure } (\text{dist-result-type } dst)) (\text{dist-dens } dst\ v)$

from *assms* **have** $(\int^+ u. \text{dist-dens } dst\ v\ u\ \partial \text{stock-measure } (\text{dist-result-type } dst))$

$=$

$emeasure\ ?M\ (space\ ?M)$
by (*subst space-density, subst emeasure-density*)
 (*auto intro!: measurable-dist-dens cong: nn-integral-cong*)
also have $?M = dist\ measure\ dst\ v$ **using** *dist-measure-has-density[OF assms]*
by (*auto dest: has-densityD*)
also from *assms* **have** $emeasure\ \dots\ (space\ \dots) \leq 1$
by (*intro subprob-space.emeasure-space-le-1 subprob-space-dist-measure*)
finally show *?thesis* .
qed

5.2 Typing

primrec *op-type* :: *pdf-operator* \Rightarrow *pdf-type* \Rightarrow *pdf-type option* **where**
op-type Add $x =$
 (*case x of*
 $PRODUCT\ INTEG\ INTEG \Rightarrow Some\ INTEG$
 $| PRODUCT\ REAL\ REAL \Rightarrow Some\ REAL$
 $| - \Rightarrow None$)
| op-type Mult $x =$
 (*case x of*
 $PRODUCT\ INTEG\ INTEG \Rightarrow Some\ INTEG$
 $| PRODUCT\ REAL\ REAL \Rightarrow Some\ REAL$
 $| - \Rightarrow None$)
| op-type Minus $x =$
 (*case x of*
 $INTEG \Rightarrow Some\ INTEG$
 $| REAL \Rightarrow Some\ REAL$
 $| - \Rightarrow None$)
| op-type Equals $x =$
 (*case x of*
 $PRODUCT\ t1\ t2 \Rightarrow if\ t1 = t2\ then\ Some\ BOOL\ else\ None$
 $| - \Rightarrow None$)
| op-type Less $x =$
 (*case x of*
 $PRODUCT\ INTEG\ INTEG \Rightarrow Some\ BOOL$
 $| PRODUCT\ REAL\ REAL \Rightarrow Some\ BOOL$
 $| - \Rightarrow None$)
| op-type (Cast t) $x =$
 (*case (x, t) of*
 $(BOOL, INTEG) \Rightarrow Some\ INTEG$
 $| (BOOL, REAL) \Rightarrow Some\ REAL$
 $| (INTEG, REAL) \Rightarrow Some\ REAL$
 $| (REAL, INTEG) \Rightarrow Some\ INTEG$
 $| - \Rightarrow None$)
| op-type Or $x = (case\ x\ of\ PRODUCT\ BOOL\ BOOL \Rightarrow Some\ BOOL\ | - \Rightarrow None)$
| op-type And $x = (case\ x\ of\ PRODUCT\ BOOL\ BOOL \Rightarrow Some\ BOOL\ | - \Rightarrow None)$
| op-type Not $x = (case\ x\ of\ BOOL \Rightarrow Some\ BOOL\ | - \Rightarrow None)$
| op-type Inverse $x = (case\ x\ of\ REAL \Rightarrow Some\ REAL\ | - \Rightarrow None)$

$| \text{op-type Fact } x = (\text{case } x \text{ of } \text{INTEG} \Rightarrow \text{Some } \text{INTEG} \mid - \Rightarrow \text{None})$
 $| \text{op-type Sqrt } x = (\text{case } x \text{ of } \text{REAL} \Rightarrow \text{Some } \text{REAL} \mid - \Rightarrow \text{None})$
 $| \text{op-type Exp } x = (\text{case } x \text{ of } \text{REAL} \Rightarrow \text{Some } \text{REAL} \mid - \Rightarrow \text{None})$
 $| \text{op-type Ln } x = (\text{case } x \text{ of } \text{REAL} \Rightarrow \text{Some } \text{REAL} \mid - \Rightarrow \text{None})$
 $| \text{op-type Pi } x = (\text{case } x \text{ of } \text{UNIT} \Rightarrow \text{Some } \text{REAL} \mid - \Rightarrow \text{None})$
 $| \text{op-type Pow } x = (\text{case } x \text{ of}$
 $\quad \text{PRODUCT REAL INTEG} \Rightarrow \text{Some REAL}$
 $\quad \mid \text{PRODUCT INTEG INTEG} \Rightarrow \text{Some INTEG}$
 $\quad \mid - \Rightarrow \text{None})$
 $| \text{op-type Fst } x = (\text{case } x \text{ of } \text{PRODUCT } t \mid - \Rightarrow \text{Some } t \mid - \Rightarrow \text{None})$
 $| \text{op-type Snd } x = (\text{case } x \text{ of } \text{PRODUCT } - t \mid - \Rightarrow \text{Some } t \mid - \Rightarrow \text{None})$

5.3 Semantics

abbreviation (*input*) *de-bruijn-insert* (**infixr** $\langle \cdot \rangle$ 65) **where**

de-bruijn-insert $x f \equiv \text{case-nat } x f$

inductive *expr-typing* :: *tyenv* \Rightarrow *expr* \Rightarrow *pdf-type* \Rightarrow *bool* ($\langle (1- / \vdash / (- : / -)) \rangle$)
 $[50, 0, 50]$ 50) **where**

$| \text{et-var}: \Gamma \vdash \text{Var } x : \Gamma x$
 $| \text{et-val}: \Gamma \vdash \text{Val } v : \text{val-type } v$
 $| \text{et-let}: \Gamma \vdash e1 : t1 \Longrightarrow t1 \cdot \Gamma \vdash e2 : t2 \Longrightarrow \Gamma \vdash \text{LetVar } e1 e2 : t2$
 $| \text{et-op}: \Gamma \vdash e : t \Longrightarrow \text{op-type oper } t = \text{Some } t' \Longrightarrow \Gamma \vdash \text{Operator oper } e : t'$
 $| \text{et-pair}: \Gamma \vdash e1 : t1 \Longrightarrow \Gamma \vdash e2 : t2 \Longrightarrow \Gamma \vdash \langle e1, e2 \rangle : \text{PRODUCT } t1 t2$
 $| \text{et-rand}: \Gamma \vdash e : \text{dist-param-type } \text{dst} \Longrightarrow \Gamma \vdash \text{Random } \text{dst } e : \text{dist-result-type } \text{dst}$
 $| \text{et-if}: \Gamma \vdash b : \text{BOOL} \Longrightarrow \Gamma \vdash e1 : t \Longrightarrow \Gamma \vdash e2 : t \Longrightarrow \Gamma \vdash \text{IF } b \text{ THEN } e1$
 $\text{ELSE } e2 : t$
 $| \text{et-fail}: \Gamma \vdash \text{Fail } t : t$

lemma *expr-typing-cong'*:

$\Gamma \vdash e : t \Longrightarrow (\bigwedge x. x \in \text{free-vars } e \Longrightarrow \Gamma x = \Gamma' x) \Longrightarrow \Gamma' \vdash e : t$

proof (*induction arbitrary*; Γ' *rule*: *expr-typing.induct*)

case (*et-let* $\Gamma e1 t1 e2 t2 \Gamma'$)

have $\Gamma' \vdash e1 : t1$ **using** *et-let.prem*s **by** (*intro et-let.IH(1)*) *auto*

moreover have *case-nat* $t1 \Gamma' \vdash e2 : t2$

using *et-let.prem*s **by** (*intro et-let.IH(2)*) (*auto split: nat.split*)

ultimately show *?case* **by** (*auto intro!*: *expr-typing.intros*)

qed (*auto intro!*: *expr-typing.intros*)

lemma *expr-typing-cong*:

$(\bigwedge x. x \in \text{free-vars } e \Longrightarrow \Gamma x = \Gamma' x) \Longrightarrow \Gamma \vdash e : t \longleftrightarrow \Gamma' \vdash e : t$

by (*intro iffI*) (*simp-all add: expr-typing-cong'*)

inductive-cases *expr-typing-valE*[*elim*]: $\Gamma \vdash \text{Val } v : t$

inductive-cases *expr-typing-varE*[*elim*]: $\Gamma \vdash \text{Var } x : t$

inductive-cases *expr-typing-letE*[*elim*]: $\Gamma \vdash \text{LetVar } e1 e2 : t$

inductive-cases *expr-typing-ifE*[*elim*]: $\Gamma \vdash \text{IfThenElse } b e1 e2 : t$

inductive-cases *expr-typing-opE*[*elim*]: $\Gamma \vdash \text{Operator oper } e : t$

inductive-cases *expr-typing-pairE*[*elim*]: $\Gamma \vdash \langle e1, e2 \rangle : t$

inductive-cases *expr-typing-randE*[*elim*]: $\Gamma \vdash \text{Random } \text{dst } e : t$
inductive-cases *expr-typing-failE*[*elim*]: $\Gamma \vdash \text{Fail } t : t'$

lemma *expr-typing-unique*:
 $\Gamma \vdash e : t \implies \Gamma \vdash e : t' \implies t = t'$
apply (*induction arbitrary: t' rule: expr-typing.induct*)
apply *blast*
apply *blast*
apply (*erule expr-typing-letE, blast*)
apply (*erule expr-typing-opE, simp*)
apply (*erule expr-typing-pairE, blast*)
apply (*erule expr-typing-randE, blast*)
apply (*erule expr-typing-ifE, blast*)
apply *blast*
done

fun *expr-type* :: *tyenv* \Rightarrow *expr* \Rightarrow *pdf-type option* **where**
expr-type Γ (*Var* *x*) = *Some* (Γ *x*)
| *expr-type* Γ (*Val* *v*) = *Some* (*val-type* *v*)
| *expr-type* Γ (*LetVar* *e1* *e2*) =
 (*case* *expr-type* Γ *e1* of
 Some *t* \Rightarrow *expr-type* (*case-nat* *t* Γ) *e2*
 | *None* \Rightarrow *None*)
| *expr-type* Γ (*Operator* *oper* *e*) =
 (*case* *expr-type* Γ *e* of *Some* *t* \Rightarrow *op-type* *oper* *t* | *None* \Rightarrow *None*)
| *expr-type* Γ (*<**e1*, *e2**>*) =
 (*case* (*expr-type* Γ *e1*, *expr-type* Γ *e2*) of
 (*Some* *t1*, *Some* *t2*) \Rightarrow *Some* (*PRODUCT* *t1* *t2*)
 | - \Rightarrow *None*)
| *expr-type* Γ (*Random* *dst* *e*) =
 (*if* *expr-type* Γ *e* = *Some* (*dist-param-type* *dst*) *then*
 Some (*dist-result-type* *dst*)
 else *None*)
| *expr-type* Γ (*IF* *b* *THEN* *e1* *ELSE* *e2*) =
 (*if* *expr-type* Γ *b* = *Some* *BOOL* *then*
 (*case* (*expr-type* Γ *e1*, *expr-type* Γ *e2*) of
 (*Some* *t*, *Some* *t'*) \Rightarrow *if* *t* = *t'* *then* *Some* *t* *else* *None*
 | - \Rightarrow *None*) *else* *None*)
| *expr-type* Γ (*Fail* *t*) = *Some* *t*

lemma *expr-type-Some-iff*: *expr-type* Γ *e* = *Some* *t* \longleftrightarrow $\Gamma \vdash e : t$
apply *rule*
apply (*induction e arbitrary: Γ t,*
 auto intro!: expr-typing.intros split: option.split-asm if-split-asm) \square
apply (*induction rule: expr-typing.induct, auto simp del: fun-upd-apply*)
done

lemmas *expr-typing-code*[*code-unfold*] = *expr-type-Some-iff*[*symmetric*]

5.3.1 Countable types

primrec *countable-type* :: *pdf-type* \Rightarrow *bool* **where**

```

  countable-type UNIT = True
| countable-type BOOL = True
| countable-type INTEG = True
| countable-type REAL = False
| countable-type (PRODUCT t1 t2) = (countable-type t1  $\wedge$  countable-type t2)

```

lemma *countable-type-countable*[*dest*]:

```

  countable-type t  $\implies$  countable (space (stock-measure t))
by (induction t)
  (auto simp: pair-measure-countable space-embed-measure space-pair-measure
stock-measure.simps)

```

lemma *countable-type-imp-count-space*:

```

  countable-type t  $\implies$  stock-measure t = count-space (type-universe t)

```

proof (*subst space-stock-measure*[*symmetric*], *induction t*)

```

case (PRODUCT t1 t2)

```

```

  hence countable: countable-type t1 countable-type t2 by simp-all

```

```

  note A = PRODUCT.IH(1)[OF countable(1)] and B = PRODUCT.IH(2)[OF
countable(2)]

```

```

  show stock-measure (PRODUCT t1 t2) = count-space (space (stock-measure
(PRODUCT t1 t2)))

```

```

    apply (subst (1 2) stock-measure.simps)

```

```

    apply (subst (1 2) A, subst (1 2) B)

```

```

    apply (subst (1 2) pair-measure-countable)

```

```

    apply (auto intro: countable-type-countable simp: countable simp del: space-stock-measure)

```

```

  [2]

```

```

    apply (subst (1 2) embed-measure-count-space, force intro: injI)

```

```

    apply simp

```

```

    done

```

```

qed (simp-all add: stock-measure.simps)

```

lemma *return-val-countable*:

```

  assumes countable-type (val-type v)

```

```

  shows return-val v = density (stock-measure (val-type v)) (indicator {v}) (is
?M1 = ?M2)

```

proof (*rule measure-eqI*)

```

  let ?M3 = count-space (type-universe (val-type v))

```

```

  fix X assume asm: X  $\in$  ?M1

```

```

  with assms have emeasure ?M2 X =  $\int^+$  x. indicator {v} x * indicator X x
     $\partial$ count-space (type-universe (val-type v))

```

```

  by (simp add: return-val-def emeasure-density countable-type-imp-count-space)

```

```

  also have ( $\lambda$ x. indicator {v} x * indicator X x :: ennreal) = ( $\lambda$ x. indicator (X
 $\cap$  {v}) x)

```

```

  by (rule ext, subst Int-commute) (simp split: split-indicator)

```

```

  also have nn-integral ?M3 ... = emeasure ?M3 (X  $\cap$  {v})

```

```

  by (subst nn-integral-indicator[symmetric]) auto

```

```

  also from asm have ... = emeasure ?M1 X by (auto simp: return-val-def split:

```

split-indicator)

finally show $\text{emeasure } ?M1 X = \text{emeasure } ?M2 X \dots$

qed (*simp add: return-val-def*)

5.4 Semantics

definition *bool-to-int* :: $\text{bool} \Rightarrow \text{int}$ **where**

bool-to-int $b = (\text{if } b \text{ then } 1 \text{ else } 0)$

lemma *measurable-bool-to-int*[*measurable*]:

bool-to-int $\in \text{measurable}$ (*count-space UNIV*) (*count-space UNIV*)

by (*rule measurable-count-space*)

definition *bool-to-real* :: $\text{bool} \Rightarrow \text{real}$ **where**

bool-to-real $b = (\text{if } b \text{ then } 1 \text{ else } 0)$

lemma *measurable-bool-to-real*[*measurable*]:

bool-to-real $\in \text{borel-measurable}$ (*count-space UNIV*)

by (*rule borel-measurable-count-space*)

definition *safe-ln* :: $\text{real} \Rightarrow \text{real}$ **where**

safe-ln $x = (\text{if } x > 0 \text{ then } \ln x \text{ else } 0)$

lemma *safe-ln-gt-0*[*simp*]: $x > 0 \implies \text{safe-ln } x = \ln x$

by (*simp add: safe-ln-def*)

lemma *borel-measurable-safe-ln*[*measurable*]: *safe-ln* $\in \text{borel-measurable borel}$

unfolding *safe-ln-def*[*abs-def*] **by** *simp*

definition *safe-sqrt* :: $\text{real} \Rightarrow \text{real}$ **where**

safe-sqrt $x = (\text{if } x \geq 0 \text{ then } \text{sqrt } x \text{ else } 0)$

lemma *safe-sqrt-ge-0*[*simp*]: $x \geq 0 \implies \text{safe-sqrt } x = \text{sqrt } x$

by (*simp add: safe-sqrt-def*)

lemma *borel-measurable-safe-sqrt*[*measurable*]: *safe-sqrt* $\in \text{borel-measurable borel}$

unfolding *safe-sqrt-def*[*abs-def*] **by** *simp*

fun *op-sem* :: $\text{pdf-operator} \Rightarrow \text{val} \Rightarrow \text{val}$ **where**

op-sem *Add* = *lift-RealIntVal2* (+) (+)

| *op-sem* *Mult* = *lift-RealIntVal2* (*) (*)

| *op-sem* *Minus* = *lift-RealIntVal* *uminus* *uminus*

| *op-sem* *Equals* = $(\lambda <|v1, v2|> \Rightarrow \text{BoolVal } (v1 = v2))$

| *op-sem* *Less* = *lift-Comp* (<) (<)

| *op-sem* *Or* = $(\lambda <| \text{BoolVal } a, \text{BoolVal } b |> \Rightarrow \text{BoolVal } (a \vee b))$

| *op-sem* *And* = $(\lambda <| \text{BoolVal } a, \text{BoolVal } b |> \Rightarrow \text{BoolVal } (a \wedge b))$

| *op-sem* *Not* = $(\lambda \text{BoolVal } a \Rightarrow \text{BoolVal } (\neg a))$

```

| op-sem (Cast t) = (case t of
  INTEG ⇒ (λ BoolVal b ⇒ IntVal (bool-to-int b)
    | RealVal r ⇒ IntVal (floor r))
  | REAL ⇒ (λ BoolVal b ⇒ RealVal (bool-to-real b)
    | IntVal i ⇒ RealVal (real-of-int i)))
| op-sem Inverse = lift-RealVal inverse
| op-sem Fact = lift-IntVal (λi::int. fact (nat i))
| op-sem Sqrt = lift-RealVal safe-sqrt
| op-sem Exp = lift-RealVal exp
| op-sem Ln = lift-RealVal safe-ln
| op-sem Pi = (λ-. RealVal pi)
| op-sem Pow = (λ <|RealVal x, IntVal n|> ⇒ if n < 0 then RealVal 0 else RealVal
(x ^ nat n)
  | <|IntVal x, IntVal n|> ⇒ if n < 0 then IntVal 0 else IntVal (x ^
nat n))
| op-sem Fst = fst ∘ extract-pair
| op-sem Snd = snd ∘ extract-pair

```

The semantics of expressions. Assumes that the expression given is well-typed.

primrec *expr-sem* :: *state* ⇒ *expr* ⇒ *val measure* **where**

```

  expr-sem σ (Var x) = return-val (σ x)
| expr-sem σ (Val v) = return-val v
| expr-sem σ (LET e1 IN e2) =
  do {
    v ← expr-sem σ e1;
    expr-sem (v · σ) e2
  }
| expr-sem σ (oper $$ e) =
  do {
    x ← expr-sem σ e;
    return-val (op-sem oper x)
  }
| expr-sem σ <v, w> =
  do {
    x ← expr-sem σ v;
    y ← expr-sem σ w;
    return-val <|x, y|>
  }
| expr-sem σ (IF b THEN e1 ELSE e2) =
  do {
    b' ← expr-sem σ b;
    if b' = TRUE then expr-sem σ e1 else expr-sem σ e2
  }
| expr-sem σ (Random dst e) =
  do {
    x ← expr-sem σ e;
    dist-measure dst x
  }

```

| $\text{expr-sem } \sigma \text{ (Fail } t) = \text{null-measure (stock-measure } t)$

lemma *expr-sem-pair-vars*: $\text{expr-sem } \sigma \langle \text{Var } x, \text{Var } y \rangle = \text{return-val } \langle |\sigma x, \sigma y| \rangle$
by (*simp add: return-val-def bind-return*[**where** $N = \text{PRODUCT (val-type } (\sigma x))$
 $(\text{val-type } (\sigma y))$]
cong: bind-cong-simp)

Well-typed expressions produce a result in the measure space that corresponds to their type

lemma *op-sem-val-type*:

op-type oper (val-type v) = Some t' \implies val-type (op-sem oper v) = t'
by (*cases oper*) (*auto split: val.split if-split-asm pdf-type.split-asm*
simp: lift-RealIntVal-def lift-Comp-def
lift-IntVal-def lift-RealVal-def lift-RealIntVal2-def
elim!: PROD-E INTEG-E REAL-E)

lemma *sets-expr-sem*:

$\Gamma \vdash w : t \implies (\forall x \in \text{free-vars } w. \text{val-type } (\sigma x) = \Gamma x) \implies$
 $\text{sets (expr-sem } \sigma w) = \text{sets (stock-measure } t)$

proof (*induction arbitrary: σ rule: expr-typing.induct*)

case (*et-var* $\Gamma x \sigma$)

thus *?case by (simp add: return-val-def)*

next

case (*et-val* $\Gamma v \sigma$)

thus *?case by (simp add: return-val-def)*

next

case (*et-let* $\Gamma e1 t1 e2 t2 \sigma$)

hence $\text{sets (expr-sem } \sigma e1) = \text{sets (stock-measure } t1)$ **by** *simp*

from *sets-eq-imp-space-eq[OF this]*

have A : $\text{space (expr-sem } \sigma e1) = \text{type-universe } t1$ **by** (*simp add:*)

hence B : $(\text{SOME } x. x \in \text{space (expr-sem } \sigma e1)) \in \text{space (expr-sem } \sigma e1)$ (**is** *?v \in -*)

unfolding *some-in-eq by simp*

with A *et-let have* $\text{sets (expr-sem (case-nat ?v } \sigma) e2) = \text{sets (stock-measure } t2)$

by (*intro et-let.IH(2)*) (*auto split: nat.split*)

with B **show** $\text{sets (expr-sem } \sigma (\text{LetVar } e1 e2)) = \text{sets (stock-measure } t2)$

by (*subst expr-sem.simps, subst bind-nonempty*) *auto*

next

case (*et-op* $\Gamma e t \text{ oper } t' \sigma$)

from *et-op.IH[of σ]* **and** *et-op.prem*s

have [*simp*]: $\text{sets (expr-sem } \sigma e) = \text{sets (stock-measure } t)$ **by** *simp*

from *sets-eq-imp-space-eq[OF this]*

have [*simp*]: $\text{space (expr-sem } \sigma e) = \text{type-universe } t$ **by** (*simp add:*)

have $(\text{SOME } x. x \in \text{space (expr-sem } \sigma e)) \in \text{space (expr-sem } \sigma e)$

unfolding *some-in-eq by simp*

with *et-op show ?case by (simp add: bind-nonempty return-val-def op-sem-val-type)*

next

case (*et-pair* $\Gamma e1 t1 e2 t2 \sigma$)

hence $[simp]$: $space (expr\text{-}sem \ \sigma \ e1) = type\text{-}universe \ t1$
 $space (expr\text{-}sem \ \sigma \ e2) = type\text{-}universe \ t2$
by $(simp\text{-}all \ add: \ sets\text{-}eq\text{-}imp\text{-}space\text{-}eq)$
have $(SOME \ x. \ x \in space (expr\text{-}sem \ \sigma \ e1)) \in space (expr\text{-}sem \ \sigma \ e1)$
 $(SOME \ x. \ x \in space (expr\text{-}sem \ \sigma \ e2)) \in space (expr\text{-}sem \ \sigma \ e2)$
unfolding $some\text{-}in\text{-}eq$ **by** $simp\text{-}all$
with $et\text{-}pair.hyps$ **show** $?case$ **by** $(simp \ add: \ bind\text{-}nonempty \ return\text{-}val\text{-}def)$
next
case $(et\text{-}rand \ \Gamma \ e \ dst \ \sigma)$
from $et\text{-}rand.IH[of \ \sigma] \ et\text{-}rand.prem$
have $sets (expr\text{-}sem \ \sigma \ e) = sets (stock\text{-}measure (dist\text{-}param\text{-}type \ dst))$ **by** $simp$
from $this \ sets\text{-}eq\text{-}imp\text{-}space\text{-}eq[OF \ this]$
show $?case$
apply $simp\text{-}all$
apply $(subst \ sets\text{-}bind)$
apply $auto$
done
next
case $(et\text{-}if \ \Gamma \ b \ e1 \ t \ e2 \ \sigma)$
have $sets (expr\text{-}sem \ \sigma \ b) = sets (stock\text{-}measure \ \text{BOOL})$
using $et\text{-}if.prem$ **by** $(intro \ et\text{-}if.IH) \ simp$
from $sets\text{-}eq\text{-}imp\text{-}space\text{-}eq[OF \ this]$
have $space (expr\text{-}sem \ \sigma \ b) \neq \{\}$ **by** $simp$
moreover **have** $sets (expr\text{-}sem \ \sigma \ e1) = sets (stock\text{-}measure \ t)$
 $sets (expr\text{-}sem \ \sigma \ e2) = sets (stock\text{-}measure \ t)$
using $et\text{-}if.prem$ **by** $(intro \ et\text{-}if.IH, \ simp)+$
ultimately **show** $?case$ **by** $(simp \ add: \ bind\text{-}nonempty)$
qed $simp\text{-}all$

lemma $space\text{-}expr\text{-}sem$:

$\Gamma \vdash w : t \implies (\forall x \in free\text{-}vars \ w. \ val\text{-}type (\sigma \ x) = \Gamma \ x)$
 $\implies space (expr\text{-}sem \ \sigma \ w) = type\text{-}universe \ t$

by $(subst \ space\text{-}stock\text{-}measure[symmetric]) (intro \ sets\text{-}expr\text{-}sem \ sets\text{-}eq\text{-}imp\text{-}space\text{-}eq)$

lemma $measurable\text{-}expr\text{-}sem\text{-}eq$:

$\Gamma \vdash e : t \implies \sigma \in space (state\text{-}measure \ V \ \Gamma) \implies free\text{-}vars \ e \subseteq V \implies$
 $measurable (expr\text{-}sem \ \sigma \ e) = measurable (stock\text{-}measure \ t)$

by $(intro \ ext \ measurable\text{-}cong\text{-}sets \ sets\text{-}expr\text{-}sem)$

$(auto \ simp: \ state\text{-}measure\text{-}def \ space\text{-}PiM \ dest: \ PiE\text{-}mem)$

lemma $measurable\text{-}expr\text{-}semI$:

$\Gamma \vdash e : t \implies \sigma \in space (state\text{-}measure \ V \ \Gamma) \implies free\text{-}vars \ e \subseteq V \implies$
 $f \in measurable (stock\text{-}measure \ t) \ M \implies f \in measurable (expr\text{-}sem \ \sigma \ e) \ M$

by $(subst \ measurable\text{-}expr\text{-}sem\text{-}eq)$

lemma $expr\text{-}sem\text{-}eq\text{-}on\text{-}vars$:

$(\bigwedge x. \ x \in free\text{-}vars \ e \implies \sigma_1 \ x = \sigma_2 \ x) \implies expr\text{-}sem \ \sigma_1 \ e = expr\text{-}sem \ \sigma_2 \ e$

proof $(induction \ e \ arbitrary: \ \sigma_1 \ \sigma_2)$

case $(LetVar \ e1 \ e2 \ \sigma_1 \ \sigma_2)$

```

    from LetVar.premis have A: expr-sem  $\sigma 1$   $e 1$  = expr-sem  $\sigma 2$   $e 1$  by (rule
LetVar.IH(1)) simp-all
    from LetVar.premis show ?case
    by (subst (1 2) expr-sem.simps, subst A)
      (auto intro!: bind-cong LetVar.IH(2) split: nat.split)
next
  case (Operator oper e  $\sigma 1$   $\sigma 2$ )
  from Operator.IH[OF Operator.premis] show ?case by simp
next
  case (Pair  $e 1$   $e 2$   $\sigma 1$   $\sigma 2$ )
  from Pair.premis have expr-sem  $\sigma 1$   $e 1$  = expr-sem  $\sigma 2$   $e 1$  by (intro Pair.IH)
  auto
  moreover from Pair.premis have expr-sem  $\sigma 1$   $e 2$  = expr-sem  $\sigma 2$   $e 2$  by (intro
Pair.IH) auto
  ultimately show ?case by simp
next
  case (Random dst e  $\sigma 1$   $\sigma 2$ )
  from Random.premis have A: expr-sem  $\sigma 1$   $e$  = expr-sem  $\sigma 2$   $e$  by (rule Ran-
dom.IH) simp-all
  show ?case
  by (subst (1 2) expr-sem.simps, subst A) (auto intro!: bind-cong)
next
  case (IfThenElse b  $e 1$   $e 2$   $\sigma 1$   $\sigma 2$ )
  have A: expr-sem  $\sigma 1$   $b$  = expr-sem  $\sigma 2$   $b$ 
    expr-sem  $\sigma 1$   $e 1$  = expr-sem  $\sigma 2$   $e 1$ 
    expr-sem  $\sigma 1$   $e 2$  = expr-sem  $\sigma 2$   $e 2$ 
  using IfThenElse.premis by (intro IfThenElse.IH, simp)+
  thus ?case by (simp only: expr-sem.simps A)
qed simp-all

```

5.5 Measurability

lemma *borel-measurable-eq*[*measurable (raw)*]:

assumes [*measurable*]: $f \in \text{borel-measurable } M$ $g \in \text{borel-measurable } M$
shows *Measurable.pred* M ($\lambda x. f x = (g x :: \text{real})$)

proof –

have *: ($\lambda x. f x = g x$) = ($\lambda x. f x - g x = 0$)

by *simp*

show ?thesis

unfolding * **by** *measurable*

qed

lemma *measurable-equals*:

($\lambda(x,y). x = y$) $\in \text{measurable (stock-measure } t \otimes_M \text{ stock-measure } t)$ (*count-space UNIV*)

proof (*induction t*)

case *REAL*

let ?f = $\lambda x. \text{extract-real (fst } x) = \text{extract-real (snd } x)$

show ?case

```

proof (subst measurable-cong)
  fix x assume x ∈ space (stock-measure REAL ⊗M stock-measure REAL)
  thus (λ(x,y). x = y) x = ?f x
  by (auto simp: space-pair-measure elim!: REAL-E)
next
  show ?f ∈ measurable (stock-measure REAL ⊗M stock-measure REAL)
(count-space UNIV)
  by measurable
qed
next
case (PRODUCT t1 t2)
let ?g = λ(x,y). x = y
let ?f = λx. ?g (fst (extract-pair (fst x)), fst (extract-pair (snd x))) ∧
      ?g (snd (extract-pair (fst x)), snd (extract-pair (snd x)))
show ?case
proof (subst measurable-cong)
  fix x assume x ∈ space (stock-measure (PRODUCT t1 t2) ⊗M stock-measure
(PRODUCT t1 t2))
  thus (λ(x,y). x = y) x = ?f x
  apply (auto simp: space-pair-measure)
  apply (elim PROD-E)
  apply simp
  done
next
note PRODUCT[measurable]
  show Measurable.pred (stock-measure (PRODUCT t1 t2) ⊗M stock-measure
(PRODUCT t1 t2)) ?f
  by measurable
qed
qed (simp-all add: pair-measure-countable stock-measure.simps)

lemma measurable-equals-stock-measure[measurable (raw)]:
  assumes f ∈ measurable M (stock-measure t) g ∈ measurable M (stock-measure
t)
  shows Measurable.pred M (λx. f x = g x)
  using measurable-compose[OF measurable-Pair[OF assms] measurable-equals] by
simp

lemma measurable-op-sem:
  assumes op-type oper t = Some t'
  shows op-sem oper ∈ measurable (stock-measure t) (stock-measure t')
proof (cases oper)
  case Fst with assms show ?thesis by (simp split: pdf-type.split-asm)
next
  case Snd with assms show ?thesis by (simp split: pdf-type.split-asm)
next
  case Equals with assms show ?thesis
  by (auto intro!: val-case-stock-measurable split: if-split-asm)
next

```

```

case Pow with assms show ?thesis
  apply (auto intro!: val-case-stock-measurable split: pdf-type.splits)
  apply (subst measurable-cong[where
    g= $\lambda(x, n).$  if extract-int n < 0 then RealVal 0 else RealVal (extract-real x ^
nat (extract-int n))]
  apply (auto simp: space-pair-measure elim!: REAL-E INTEG-E)
  done
next
  case Less with assms show ?thesis
  by (auto split: pdf-type.splits)
qed (insert assms, auto split: pdf-type.split-asm intro!: val-case-stock-measurable)

```

definition *shift-var-set* :: *vname set* \Rightarrow *vname set* **where**
shift-var-set *V* = *insert* 0 (*Suc* ' *V*)

lemma *shift-var-set-0*[*simp*]: 0 \in *shift-var-set* *V*
by (*simp add: shift-var-set-def*)

lemma *shift-var-set-Suc*[*simp*]: *Suc* *x* \in *shift-var-set* *V* \longleftrightarrow *x* \in *V*
by (*auto simp add: shift-var-set-def*)

lemma *case-nat-update-0*[*simp*]: (*case-nat* *x* σ)(0 := *y*) = *case-nat* *y* σ
by (*intro ext*) (*simp split: nat.split*)

lemma *case-nat-delete-var-1*[*simp*]:
case-nat *x* (*case-nat* *y* σ) \circ *case-nat* 0 ($\lambda x.$ *Suc* (*Suc* *x*)) = *case-nat* *x* σ
by (*intro ext*) (*simp split: nat.split*)

lemma *delete-var-1-vimage*[*simp*]:
case-nat 0 ($\lambda x.$ *Suc* (*Suc* *x*)) - ' (*shift-var-set* (*shift-var-set* *V*)) = *shift-var-set*
V
by (*auto simp: shift-var-set-def split: nat.split-asm*)

lemma *measurable-case-nat*[*measurable*]:
assumes *g* \in *measurable* *R* *N* *h* \in *measurable* *R* (*Pi*_{*M*} *V* *M*)
shows ($\lambda x.$ *case-nat* (*g* *x*) (*h* *x*)) \in *measurable* *R* (*Pi*_{*M*} (*shift-var-set* *V*) (*case-nat*
N *M*))

proof (*rule measurable-Pair-compose-split*[*OF - assms*])
have ($\lambda(t, f).$ $\lambda x \in$ *shift-var-set* *V.* *case-nat* *t* *f* *x*)
 \in *measurable* (*N* \otimes _{*M*} *Pi*_{*M*} *V* *M*) (*Pi*_{*M*} (*shift-var-set* *V*) (*case-nat* *N* *M*))

(**is** *?P*)
unfolding *shift-var-set-def*
by (*subst measurable-split-conv, rule measurable-restrict*) (*auto split: nat.split-asm*)
also have $\bigwedge x f.$ *f* \in *space* (*Pi*_{*M*} *V* *M*) \Longrightarrow *x* \notin *V* \Longrightarrow *undefined* = *f* *x*
by (*rule sym, subst (asm) space-PiM, erule PiE-arb*)
hence *?P* \longleftrightarrow ($\lambda(t, f).$ *case-nat* *t* *f*)
 \in *measurable* (*N* \otimes _{*M*} *Pi*_{*M*} *V* *M*) (*Pi*_{*M*} (*shift-var-set* *V*) (*case-nat* *N* *M*))
(**is** - = *?P*)

by (*intro measurable-cong ext*)
 (*auto split: nat.split simp: inj-image-mem-iff space-pair-measure shift-var-set-def*)
 finally show ?P .
 qed

lemma *measurable-case-nat*'[*measurable*]:
 assumes $g \in \text{measurable } R$ (*stock-measure t*) $h \in \text{measurable } R$ (*state-measure V* Γ)
 shows $(\lambda x. \text{case-nat } (g x) (h x)) \in \text{measurable } R$ (*state-measure (shift-var-set V) (case-nat t* Γ))
proof –
 have $A: (\lambda x. \text{stock-measure } (\text{case-nat } t \Gamma x)) = \text{case-nat } (\text{stock-measure } t) (\lambda x. \text{stock-measure } (\Gamma x))$
 by (*intro ext (simp split: nat.split)*)
 show ?thesis using **assms unfolding state-measure-def** by (*simp add: A*)
 qed

lemma *case-nat-in-state-measure*[*intro*]:
 assumes $x \in \text{type-universe } t1$ $\sigma \in \text{space } (\text{state-measure } V \Gamma)$
 shows $\text{case-nat } x \sigma \in \text{space } (\text{state-measure } (\text{shift-var-set } V) (\text{case-nat } t1 \Gamma))$
apply (*rule measurable-space[OF measurable-case-nat']*)
apply (*rule measurable-ident-sets[OF refl], rule measurable-const[OF assms(2)]*)
using *assms*
apply *simp*
done

lemma *subset-shift-var-set*:
 $\text{Suc } - ' A \subseteq V \implies A \subseteq \text{shift-var-set } V$
by (*rule subsetI, rename-tac x, case-tac x (auto simp: shift-var-set-def)*)

lemma *measurable-expr-sem*[*measurable*]:
 assumes $\Gamma \vdash e : t$ and $\text{free-vars } e \subseteq V$
 shows $(\lambda \sigma. \text{expr-sem } \sigma e) \in \text{measurable } (\text{state-measure } V \Gamma)$
 (*subprob-algebra (stock-measure t)*)
using *assms*
proof (*induction arbitrary: V rule: expr-typing.induct*)
case (*et-var* Γx)
 have $A: (\lambda \sigma. \text{expr-sem } \sigma (\text{Var } x)) = \text{return-val} \circ (\lambda \sigma. \sigma x)$ by (*simp add: o-def*)
with *et-var* **show** ?case **unfolding** *state-measure-def*
by (*subst A (rule measurable-comp[OF measurable-component-singleton], simp-all)*)
next
case (*et-val* Γv)
thus ?case **by** (*auto intro!: measurable-const subprob-space-return simp: space-subprob-algebra return-val-def*)
next
case (*et-let* $\Gamma e1 t1 e2 t2 V$)
 have $A: (\lambda v. \text{stock-measure } (\text{case-nat } t1 \Gamma v)) = \text{case-nat } (\text{stock-measure } t1) (\lambda v. \text{stock-measure } (\Gamma v))$
by (*rule ext (simp split: nat.split)*)

```

from et-let.prems and et-let.hyps show ?case
  apply (subst expr-sem.simps, intro measurable-bind)
  apply (rule et-let.IH(1), simp)
  apply (rule measurable-compose[OF - et-let.IH(2)][of shift-var-set V])
  apply (simp-all add: subset-shift-var-set)
  done
next
  case (et-op  $\Gamma$  e t oper t')
  thus ?case by (auto intro!: measurable-bind2 measurable-compose[OF - measurable-return-val]
    measurable-op-sem cong: measurable-cong)
next
  case (et-pair t t1 t2  $\Gamma$  e1 e2)
  have inj ( $\lambda(a,b). \langle |a, b| \rangle$ ) by (auto intro: injI)
  with et-pair show ?case
    apply (subst expr-sem.simps)
    apply (rule measurable-bind, (auto) [])
    apply (rule measurable-bind[OF measurable-compose[OF measurable-fst]],
      (auto) [])
    apply (rule measurable-compose[OF - measurable-return-val], simp)
    done
next
  case (et-rand  $\Gamma$  e dst V)
  from et-rand.prems and et-rand.hyps show ?case
    by (auto intro!: et-rand.IH measurable-compose[OF measurable-snd]
      measurable-bind measurable-dist-measure)
next
  case (et-if  $\Gamma$  b e1 t e2 V)
  let ?M =  $\lambda e t. (\lambda \sigma. \text{expr-sem } \sigma e) \in$ 
    measurable (state-measure V  $\Gamma)$  (subprob-algebra (stock-measure
t))
  from et-if.prems have A[measurable]: ?M b BOOL ?M e1 t ?M e2 t by (intro
et-if.IH, simp)+
  show ?case by (subst expr-sem.simps, rule measurable-bind[OF A(1)]) simp-all
next
  case (et-fail  $\Gamma$  t V)
  show ?case
    by (auto intro!: measurable-subprob-algebra subprob-spaceI simp;)
qed

```

5.6 Randomfree expressions

```

fun randomfree :: expr  $\Rightarrow$  bool where
  randomfree (Val -) = True
| randomfree (Var -) = True
| randomfree (Pair e1 e2) = (randomfree e1  $\wedge$  randomfree e2)
| randomfree (Operator - e) = randomfree e
| randomfree (LetVar e1 e2) = (randomfree e1  $\wedge$  randomfree e2)
| randomfree (IfThenElse b e1 e2) = (randomfree b  $\wedge$  randomfree e1  $\wedge$  randomfree

```

$e2$)
| *randomfree* (*Random* -) = *False*
| *randomfree* (*Fail* -) = *False*

primrec *expr-sem-rf* :: *state* \Rightarrow *expr* \Rightarrow *val* **where**

expr-sem-rf - (*Val* v) = v
| *expr-sem-rf* σ (*Var* x) = σ x
| *expr-sem-rf* σ ($\langle e1, e2 \rangle$) = $\langle |expr-sem-rf$ σ $e1$, *expr-sem-rf* σ $e2| \rangle$
| *expr-sem-rf* σ (*Operator* *oper* e) = *op-sem* *oper* (*expr-sem-rf* σ e)
| *expr-sem-rf* σ (*LetVar* $e1$ $e2$) = *expr-sem-rf* (*expr-sem-rf* σ $e1$ \cdot σ) $e2$
| *expr-sem-rf* σ (*IfThenElse* b $e1$ $e2$) =
 (*if* *expr-sem-rf* σ b = *BoolVal* *True* then *expr-sem-rf* σ $e1$ else *expr-sem-rf* σ
 $e2$)
| *expr-sem-rf* - (*Random* -) = *undefined*
| *expr-sem-rf* - (*Fail* -) = *undefined*

lemma *measurable-expr-sem-rf*[*measurable*]:

$\Gamma \vdash e : t \Longrightarrow \text{randomfree } e \Longrightarrow \text{free-vars } e \subseteq V \Longrightarrow$
 $(\lambda \sigma. \text{expr-sem-rf } \sigma e) \in \text{measurable } (\text{state-measure } V \Gamma) (\text{stock-measure } t)$

proof (*induction arbitrary: V rule: expr-typing.induct*)

case (*et-val* Γ v V)

thus ?*case by* (*auto intro!*: *measurable-const simp*!)

next

case (*et-var* Γ x V)

thus ?*case by* (*auto simp*: *state-measure-def intro!*: *measurable-component-singleton*)

next

case (*et-pair* Γ $e1$ $t1$ $e2$ $t2$ V)

have *inj* $(\lambda(x,y). \langle |x, y| \rangle)$ **by** (*auto intro: injI*)

with *et-pair show* ?*case by simp*

next

case (*et-op* Γ e t *oper* t' V)

thus ?*case by* (*auto intro!*: *measurable-compose*[*OF - measurable-op-sem*])

next

case (*et-let* Γ $e1$ $t1$ $e2$ $t2$ V)

hence $M1$: $(\lambda \sigma. \text{expr-sem-rf } \sigma e1) \in \text{measurable } (\text{state-measure } V \Gamma) (\text{stock-measure } t1)$

and $M2$: $(\lambda \sigma. \text{expr-sem-rf } \sigma e2) \in \text{measurable } (\text{state-measure } (\text{shift-var-set } V) (\text{case-nat } t1 \Gamma))$
 $(\text{stock-measure } t2)$

using *subset-shift-var-set*

by (*auto intro!*: *et-let.IH(1)*[*of V*] *et-let.IH(2)*[*of shift-var-set V*])

have $(\lambda \sigma. \text{expr-sem-rf } \sigma (\text{LetVar } e1 e2)) =$

$(\lambda \sigma. \text{expr-sem-rf } \sigma e2) \circ (\lambda(\sigma,y). \text{case-nat } y \sigma) \circ (\lambda \sigma. (\sigma, \text{expr-sem-rf}$

$\sigma e1))$ (**is** - = ?*f*)

by (*intro ext*) *simp*

also have ?*f* $\in \text{measurable } (\text{state-measure } V \Gamma) (\text{stock-measure } t2)$

apply (*intro measurable-comp*, *rule measurable-Pair*, *rule measurable-ident-sets*[*OF refl*])

```

  apply (rule M1, subst measurable-split-conv, rule measurable-case-nat')
  apply (rule measurable-snd, rule measurable-fst, rule M2)
done
finally show ?case .
qed (simp-all add: expr-sem-rf-def)

lemma expr-sem-rf-sound:
   $\Gamma \vdash e : t \implies \text{randomfree } e \implies \text{free-vars } e \subseteq V \implies \sigma \in \text{space } (\text{state-measure } V \Gamma) \implies$ 
  return-val (expr-sem-rf  $\sigma$   $e$ ) = expr-sem  $\sigma$   $e$ 
proof (induction arbitrary:  $V$   $\sigma$  rule: expr-typing.induct)
  case (et-val  $\Gamma$   $v$ )
  thus ?case by simp
next
  case (et-var  $\Gamma$   $x$ )
  thus ?case by simp
next
  case (et-pair  $\Gamma$   $e1$   $t1$   $e2$   $t2$   $V$   $\sigma$ )
  let ?M = state-measure  $V$   $\Gamma$ 
  from et-pair.hyps and et-pair.prem
  have e1: return-val (expr-sem-rf  $\sigma$   $e1$ ) = expr-sem  $\sigma$   $e1$  and
  e2: return-val (expr-sem-rf  $\sigma$   $e2$ ) = expr-sem  $\sigma$   $e2$ 
  by (auto intro!: et-pair.IH[of  $V$ ])

  from e1 and et-pair.prem have space (return-val (expr-sem-rf  $\sigma$   $e1$ )) = type-universe
  t1
  by (subst e1, subst space-expr-sem[OF et-pair.hyps(1)])
  (auto dest: state-measure-var-type)
  hence A: val-type (expr-sem-rf  $\sigma$   $e1$ ) = t1 expr-sem-rf  $\sigma$   $e1$   $\in$  type-universe t1
  by (auto simp add: return-val-def)
  from e2 and et-pair.prem have space (return-val (expr-sem-rf  $\sigma$   $e2$ )) = type-universe
  t2
  by (subst e2, subst space-expr-sem[OF et-pair.hyps(2)])
  (auto dest: state-measure-var-type)
  hence B: val-type (expr-sem-rf  $\sigma$   $e2$ ) = t2 expr-sem-rf  $\sigma$   $e2$   $\in$  type-universe t2
  by (auto simp add: return-val-def)

  have expr-sem  $\sigma$  (< $e1$ ,  $e2$ >) = expr-sem  $\sigma$   $e1$   $\ggg$ 
  ( $\lambda v. \text{expr-sem } \sigma$   $e2$   $\ggg$  ( $\lambda w. \text{return-val } (<|v, w|>)$ )) by simp
  also have expr-sem  $\sigma$   $e1$  = return (stock-measure t1) (expr-sem-rf  $\sigma$   $e1$ )
  using e1 by (simp add: et-pair.prem return-val-def A)
  also have ...  $\ggg$  ( $\lambda v. \text{expr-sem } \sigma$   $e2$   $\ggg$  ( $\lambda w. \text{return-val } (<|v, w|>)$ )) =
  ...  $\ggg$  ( $\lambda v. \text{return-val } (<|v, \text{expr-sem-rf } \sigma$   $e2|>)$ )
proof (intro bind-cong refl)
  fix  $v$  assume  $v \in \text{space } (\text{return } (\text{stock-measure } t1) (\text{expr-sem-rf } \sigma$   $e1))$ 
  hence  $v$ : val-type  $v$  = t1  $v \in \text{type-universe } t1$  by (simp-all add:)
  have expr-sem  $\sigma$   $e2$   $\ggg$  ( $\lambda w. \text{return-val } (<|v, w|>)$ ) =
  return (stock-measure t2) (expr-sem-rf  $\sigma$   $e2$ )  $\ggg$  ( $\lambda w. \text{return-val}$ 
  (<| $v, w$ |>))

```

```

    using e2 by (simp add: et-pair.premis return-val-def B)
  also have ... = return (stock-measure t2) (expr-sem-rf σ e2) ≫=
    (λw. return (stock-measure (PRODUCT t1 t2)) (<|v,w|>))
  proof (intro bind-cong refl)
    fix w assume w ∈ space (return (stock-measure t2) (expr-sem-rf σ e2))
    hence w: val-type w = t2 by (simp add:)
    thus return-val (<|v,w|>) = return (stock-measure (PRODUCT t1 t2))
  (<|v,w|>)
    by (auto simp: return-val-def v w)
  qed
  also have ... = return-val (<|v, expr-sem-rf σ e2|>)
    using v B
    by (subst bind-return[where N=PRODUCT t1 t2]) (auto simp: return-val-def)
  finally show expr-sem σ e2 ≫= (λw. return-val (<|v,w|>)) = return-val (<|v,
  expr-sem-rf σ e2|>) .
  qed
  also have (λv. <|v, expr-sem-rf σ e2|>) ∈ measurable (stock-measure t1) (stock-measure
  (PRODUCT t1 t2))
    using B by (auto intro!: injI)
  hence return (stock-measure t1) (expr-sem-rf σ e1) ≫= (λv. return-val (<|v,
  expr-sem-rf σ e2|>)) =
    return-val (<|expr-sem-rf σ e1, expr-sem-rf σ e2|>)
    by (subst bind-return, rule measurable-compose[OF - measurable-return-val])
    (auto simp: A)
  finally show return-val (expr-sem-rf σ (<e1,e2>)) = expr-sem σ (<e1, e2>)
  by simp
next
  case (et-if Γ b e1 t e2 V σ)
  let ?P = λe. expr-sem σ e = return-val (expr-sem-rf σ e)
  from et-if.premis have A: ?P b ?P e1 ?P e2 by ((intro et-if.IH[symmetric],
  simp-all) [])+
  from et-if.premis and et-if.hyps have space (expr-sem σ b) = type-universe
  BOOL
  by (intro space-expr-sem) (auto simp: state-measure-var-type)
  hence [simp]: val-type (expr-sem-rf σ b) = BOOL by (simp add: A return-val-def)
  have B: return-val (expr-sem-rf σ e1) ∈ space (subprob-algebra (stock-measure
  t))
    return-val (expr-sem-rf σ e2) ∈ space (subprob-algebra (stock-measure t))
  using et-if.hyps and et-if.premis
  by ((subst A[symmetric], intro measurable-space[OF measurable-expr-sem], auto)
  [])+
  thus ?case
    by (auto simp: A bind-return-val'[where M=t])
next
  case (et-op Γ e t oper t' V)
  let ?M = PiM V (λx. stock-measure (Γ x))
  from et-op.premis have e: return-val (expr-sem-rf σ e) = expr-sem σ e
    by (intro et-op.IH[of V]) auto

```

with *et-op.prem*s **have** *space* (return-val (expr-sem-rf σ *e*)) = type-universe *t*
by (subst *e*, subst space-expr-sem[OF *et-op.hyps*(1)])
(auto dest: state-measure-var-type)
hence *A*: val-type (expr-sem-rf σ *e*) = *t* expr-sem-rf σ *e* \in type-universe *t*
by (auto simp add: return-val-def)
from *et-op.prem*s *e*
have expr-sem σ (Operator oper *e*) =
return-val (expr-sem-rf σ *e*) \ggg (λv . return-val (op-sem oper *v*)) **by**
simp
also have ... = return-val (op-sem oper (expr-sem-rf σ *e*))
by (subst return-val-def, rule bind-return,
rule measurable-compose[OF measurable-op-sem measurable-return-val])
(auto simp: *A et-op.hyps*)
finally show return-val (expr-sem-rf σ (Operator oper *e*)) = expr-sem σ (Operator
oper *e*) **by** *simp*
next
case (et-let Γ *e1 t1 e2 t2 V*)
let ?*M* = state-measure *V* Γ **and** ?*N* = state-measure (shift-var-set *V*) (case-nat
t1 Γ)
let ? σ' = case-nat (expr-sem-rf σ *e1*) σ
from et-let.prems **have** *e1*: return-val (expr-sem-rf σ *e1*) = expr-sem σ *e1*
by (auto intro!: et-let.IH(1)[of *V*])
from et-let.prems **have** *S*: space (return-val (expr-sem-rf σ *e1*)) = type-universe
t1
by (subst *e1*, subst space-expr-sem[OF et-let.hyps(1)])
(auto dest: state-measure-var-type)
hence *A*: val-type (expr-sem-rf σ *e1*) = *t1* expr-sem-rf σ *e1* \in type-universe *t1*
by (auto simp add: return-val-def)
with et-let.prems **have** *e2*: $\bigwedge \sigma$. $\sigma \in$ space ?*N* \implies return-val (expr-sem-rf σ *e2*)
= expr-sem σ *e2*
using subset-shift-var-set
by (intro et-let.IH(2)[of shift-var-set *V*]) (auto simp del: fun-upd-apply)

from et-let.prems **have** expr-sem σ (LetVar *e1 e2*) =
return-val (expr-sem-rf σ *e1*) \ggg (λv . expr-sem (case-nat
v σ) *e2*)
by (simp add: *e1*)
also from et-let.prems
have ... = return-val (expr-sem-rf σ *e1*) \ggg (λv . return-val (expr-sem-rf
(case-nat *v* σ) *e2*))
by (intro bind-cong refl, subst *e2*) (auto simp: *S*)
also from et-let **have** *Me2*[measurable]: ($\lambda \sigma$. expr-sem-rf σ *e2*) \in measurable
?*N* (stock-measure *t2*)
using subset-shift-var-set **by** (intro measurable-expr-sem-rf) auto
have ($\lambda (\sigma, y)$. case-nat *y* σ) \circ (λy . (σ , *y*)) \in measurable (stock-measure *t1*) ?*N*
using $\langle \sigma \in$ space ?*M* \rangle **by** *simp*
have return-val (expr-sem-rf σ *e1*) \ggg (λv . return-val (expr-sem-rf (case-nat
v σ) *e2*)) =
return-val (expr-sem-rf ? σ' *e2*) **using** $\langle \sigma \in$ space ?*M* \rangle

by (*subst return-val-def, intro bind-return, subst A*)
 (*rule measurable-compose[OF - measurable-return-val[of t2]], simp-all*)
finally show ?*case by simp*
qed *simp-all*

lemma *val-type-expr-sem-rf*:

assumes $\Gamma \vdash e : t$ *randomfree e free-vars e* $\subseteq V$ $\sigma \in \text{space}$ (*state-measure V* Γ)
shows *val-type* (*expr-sem-rf* σ e) = t

proof –

have *type-universe* (*val-type* (*expr-sem-rf* σ e)) = *space* (*return-val* (*expr-sem-rf* σ e))

by (*simp add: return-val-def*)

also from *assms* **have** *return-val* (*expr-sem-rf* σ e) = *expr-sem* σ e

by (*intro expr-sem-rf-sound*) *auto*

also from *assms* **have** *space* ... = *type-universe t*

by (*intro space-expr-sem[of Γ]*)

(*auto simp: state-measure-def space-PiM dest: PiE-mem*)

finally show ?*thesis by simp*

qed

lemma *expr-sem-rf-eq-on-vars*:

($\bigwedge x. x \in \text{free-vars } e \implies \sigma 1 x = \sigma 2 x$) $\implies \text{expr-sem-rf } \sigma 1 e = \text{expr-sem-rf } \sigma 2 e$

proof (*induction e arbitrary: $\sigma 1 \sigma 2$*)

case (*Operator oper e $\sigma 1 \sigma 2$*)

hence *expr-sem-rf* $\sigma 1 e = \text{expr-sem-rf } \sigma 2 e$ **by** (*intro Operator.IH*) *auto*

thus ?*case by simp*

next

case (*LetVar e1 e2 $\sigma 1 \sigma 2$*)

hence $A: \text{expr-sem-rf } \sigma 1 e1 = \text{expr-sem-rf } \sigma 2 e1$ **by** (*intro LetVar.IH*) *auto*

{

fix y **assume** $y \in \text{free-vars } e2$

hence *case-nat* (*expr-sem-rf* $\sigma 1 e1$) $\sigma 1 y = \text{case-nat}$ (*expr-sem-rf* $\sigma 2 e1$) $\sigma 2 y$

using *LetVar(3)* **by** (*auto simp add: A split: nat.split*)

}

hence *expr-sem-rf* (*case-nat* (*expr-sem-rf* $\sigma 1 e1$) $\sigma 1$) $e2 =$

expr-sem-rf (*case-nat* (*expr-sem-rf* $\sigma 2 e1$) $\sigma 2$) $e2$ **by** (*intro LetVar.IH*)

simp

thus ?*case by simp*

next

case (*Pair e1 e2 $\sigma 1 \sigma 2$*)

have *expr-sem-rf* $\sigma 1 e1 = \text{expr-sem-rf } \sigma 2 e1$ *expr-sem-rf* $\sigma 1 e2 = \text{expr-sem-rf } \sigma 2 e2$

by (*intro Pair.IH, simp add: Pair*)**+**

thus ?*case by simp*

next

case (*IfThenElse b e1 e2 $\sigma 1 \sigma 2$*)

have *expr-sem-rf* $\sigma 1 b = \text{expr-sem-rf } \sigma 2 b$ *expr-sem-rf* $\sigma 1 e1 = \text{expr-sem-rf } \sigma 2 e1$

expr-sem-rf $\sigma 1 e2 = \text{expr-sem-rf } \sigma 2 e2$ **by** (*intro IfThenElse.IH, simp add:*

```

IfThenElse)+
  thus ?case by simp
next
  case (Random dst e σ1 σ2)
  have expr-sem-rf σ1 e = expr-sem-rf σ2 e by (intro Random.IH) (simp add:
Random)
  thus ?case by simp
qed auto

```

end

6 Density Contexts

```

theory PDF-Density-Contexts
imports PDF-Semantics
begin

```

```

lemma measurable-proj-state-measure[measurable (raw)]:
   $i \in V \implies (\lambda x. x i) \in \text{measurable } (\text{state-measure } V \Gamma) (\Gamma i)$ 
  unfolding state-measure-def by measurable

```

```

lemma measurable-dens-ctxt-fun-upd[measurable (raw)]:
   $f \in N \rightarrow_M \text{state-measure } V' \Gamma \implies V = V' \cup \{x\} \implies$ 
   $g \in N \rightarrow_M \text{stock-measure } (\Gamma x) \implies$ 
   $(\lambda \omega. (f \omega)(x := g \omega)) \in N \rightarrow_M \text{state-measure } V \Gamma$ 
  unfolding state-measure-def
  by (rule measurable-fun-upd[where J=V']) auto

```

```

lemma measurable-case-nat-Suc-PiM:
   $(\lambda \sigma. \sigma \circ \text{Suc}) \in \text{measurable } (\text{PiM } (\text{Suc } 'A) (\text{case-nat } M N)) (\text{PiM } A N)$ 
proof -
  have  $(\lambda \sigma. \lambda x \in A. \sigma (\text{Suc } x)) \in \text{measurable}$ 
     $(\text{PiM } (\text{Suc } 'A) (\text{case-nat } M N)) (\text{PiM } A (\lambda x. \text{case-nat } M N (\text{Suc } x)))$  (is ?A)
  by measurable
  also have ?A  $\longleftrightarrow$  ?thesis
  by (force intro!: measurable-cong ext simp: state-measure-def space-PiM dest:
PiE-mem)
  finally show ?thesis .
qed

```

```

lemma measurable-case-nat-Suc:
   $(\lambda \sigma. \sigma \circ \text{Suc}) \in \text{measurable } (\text{state-measure } (\text{Suc } 'A) (\text{case-nat } t \Gamma)) (\text{state-measure }
A \Gamma)$ 
proof -
  have  $(\lambda \sigma. \lambda x \in A. \sigma (\text{Suc } x)) \in \text{measurable}$ 
     $(\text{state-measure } (\text{Suc } 'A) (\text{case-nat } t \Gamma)) (\text{state-measure } A (\lambda i. \text{case-nat } t \Gamma$ 

```

(*Suc i*)) (is ?A)
unfolding *state-measure-def* **by** *measurable*
also have ?A \longleftrightarrow ?thesis
by (*force intro!*: *measurable-cong ext simp: state-measure-def space-PiM dest: PiE-mem*)
finally show ?thesis .
qed

A density context holds a set of variables V , their types (using Γ), and a common density function δ of the finite product space of all the variables in V . δ takes a state $\sigma \in (\prod_E x \in V. \text{type-universe } (\Gamma x))$ and returns the common density of these variables.

type-synonym *dens-ctxt* = *vname set* \times *vname set* \times (*vname* \Rightarrow *pdf-type*) \times (*state* \Rightarrow *ennreal*)

type-synonym *expr-density* = *state* \Rightarrow *val* \Rightarrow *ennreal*

definition *empty-dens-ctxt* :: *dens-ctxt* **where**
empty-dens-ctxt = ($\{\}$, $\{\}$, $\lambda\cdot. \text{undefined}$, $\lambda\cdot. 1$)

definition *state-measure'*
:: *vname set* \Rightarrow *vname set* \Rightarrow (*vname* \Rightarrow *pdf-type*) \Rightarrow *state* \Rightarrow *state measure*
where
state-measure' $V V' \Gamma \varrho =$
distr (*state-measure* $V \Gamma$) (*state-measure* ($V \cup V'$) Γ) ($\lambda\sigma. \text{merge } V V' (\sigma, \varrho)$)

The marginal density of a variable x is obtained by integrating the common density δ over all the remaining variables.

definition *marg-dens* :: *dens-ctxt* \Rightarrow *vname* \Rightarrow *expr-density* **where**
marg-dens = ($\lambda(V, V', \Gamma, \delta) x \varrho v. \int^+ \sigma. \delta (\text{merge } V V' (\sigma(x := v), \varrho)) \partial \text{state-measure } (V - \{x\}) \Gamma$)

definition *marg-dens2* :: *dens-ctxt* \Rightarrow *vname* \Rightarrow *vname* \Rightarrow *expr-density* **where**
marg-dens2 $\equiv (\lambda(V, V', \Gamma, \delta) x y \varrho v.$
 $\int^+ \sigma. \delta (\text{merge } V V' (\sigma(x := \text{fst } (\text{extract-pair } v), y := \text{snd } (\text{extract-pair } v)), \varrho))$
 $\partial \text{state-measure } (V - \{x, y\}) \Gamma$)

definition *dens-ctxt-measure* :: *dens-ctxt* \Rightarrow *state* \Rightarrow *state measure* **where**
dens-ctxt-measure $\equiv \lambda(V, V', \Gamma, \delta) \varrho. \text{density } (\text{state-measure}' V V' \Gamma \varrho) \delta$

definition *branch-prob* :: *dens-ctxt* \Rightarrow *state* \Rightarrow *ennreal* **where**
branch-prob $\mathcal{Y} \varrho = \text{emeasure } (\text{dens-ctxt-measure } \mathcal{Y} \varrho) (\text{space } (\text{dens-ctxt-measure } \mathcal{Y} \varrho))$

lemma *dens-ctxt-measure-nonempty[simp]*:
 $\text{space } (\text{dens-ctxt-measure } \mathcal{Y} \varrho) \neq \{\}$
unfolding *dens-ctxt-measure-def state-measure'-def* **by** (*cases* \mathcal{Y}) *simp*

lemma *sets-dens-ctxt-measure-eq*[*measurable-cong*]:

sets (dens-ctxt-measure (V, V', Γ, δ) ρ) = sets (state-measure (V ∪ V') Γ)

by (*simp-all add: dens-ctxt-measure-def state-measure'-def*)

lemma *measurable-dens-ctxt-measure-eq*:

measurable (dens-ctxt-measure (V, V', Γ, δ) ρ) = measurable (state-measure (V ∪ V') Γ)

by (*intro ext measurable-cong-sets*)

(*simp-all add: dens-ctxt-measure-def state-measure'-def*)

lemma *space-dens-ctxt-measure*:

space (dens-ctxt-measure (V, V', Γ, δ) ρ) = space (state-measure (V ∪ V') Γ)

unfolding *dens-ctxt-measure-def state-measure'-def* **by** *simp*

definition *apply-dist-to-dens* :: *pdf-dist* \Rightarrow (*state* \Rightarrow *val* \Rightarrow *ennreal*) \Rightarrow (*state* \Rightarrow *val* \Rightarrow *ennreal*) **where**

*apply-dist-to-dens dst f = (λρ y. ∫⁺x. f ρ x * dist-dens dst x y ∂stock-measure (dist-param-type dst))*

definition *remove-var* :: *state* \Rightarrow *state* **where**

remove-var σ = (λx. σ (Suc x))

lemma *measurable-remove-var*[*measurable*]:

remove-var ∈ measurable (state-measure (shift-var-set V) (case-nat t Γ)) (state-measure V Γ)

proof –

have ($\lambda\sigma. \lambda x \in V. \sigma (Suc x) \in measurable$

$(state-measure (shift-var-set V) (case-nat t \Gamma)) (state-measure V (\lambda x. case-nat t \Gamma (Suc x)))$)

(**is** $?f \in ?M$)

unfolding *state-measure-def shift-var-set-def* **by** *measurable*

also have $\bigwedge x f. x \notin V \implies f \in space (state-measure (shift-var-set V) (case-nat t \Gamma)) \implies$

$f (Suc x) = undefined$ **unfolding** *state-measure-def*

by (*subst (asm) space-PiM, drule PiE-arb[of - - Suc x for x]*)

(*simp-all add: space-PiM shift-var-set-def inj-image-mem-iff*)

hence $?f \in ?M \iff remove-var \in ?M$ **unfolding** *remove-var-def[abs-def] state-measure-def*

by (*intro measurable-cong ext*) (*auto simp: space-PiM intro!: sym[of - undefined]*)

finally show *?thesis* **by** *simp*

qed

lemma *measurable-case-nat-undefined*[*measurable*]:

case-nat undefined ∈ measurable (state-measure A Γ) (state-measure (Suc' A) (case-nat t Γ)) (is - ∈ ?M)

proof –

have ($\lambda\sigma. \lambda x \in Suc' A. case-nat undefined \sigma x \in ?M$ (**is** $?f \in -$))

unfolding *state-measure-def* **by** (*rule measurable-restrict*) *auto*

also have $?f \in ?M \iff ?thesis$

by (intro measurable-cong ext)
 (auto simp: state-measure-def space-PiM dest: PiE-mem split: nat.split)
 finally show ?thesis .
 qed

definition insert-dens

:: vname set \Rightarrow vname set \Rightarrow expr-density \Rightarrow (state \Rightarrow ennreal) \Rightarrow state \Rightarrow
 ennreal **where**
 insert-dens V V' f $\delta \equiv \lambda \sigma. \delta$ (remove-var σ) * f (remove-var σ) (σ 0)

definition if-dens :: (state \Rightarrow ennreal) \Rightarrow (state \Rightarrow val \Rightarrow ennreal) \Rightarrow bool \Rightarrow
 (state \Rightarrow ennreal) **where**
 if-dens δ f b $\equiv \lambda \sigma. \delta$ σ * f σ (BoolVal b)

definition if-dens-det :: (state \Rightarrow ennreal) \Rightarrow expr \Rightarrow bool \Rightarrow (state \Rightarrow ennreal)
where
 if-dens-det δ e b $\equiv \lambda \sigma. \delta$ σ * (if expr-sem-rf σ e = BoolVal b then 1 else 0)

lemma measurable-if-dens:

assumes [measurable]: $\delta \in$ borel-measurable M
assumes [measurable]: case-prod f \in borel-measurable (M \otimes_M count-space (range BoolVal))
shows if-dens δ f b \in borel-measurable M
unfolding if-dens-def **by** measurable

lemma measurable-if-dens-det:

assumes e: $\Gamma \vdash e$: BOOL randomfree e free-vars e \subseteq V
assumes [measurable]: $\delta \in$ borel-measurable (state-measure V Γ)
shows if-dens-det δ e b \in borel-measurable (state-measure V Γ)
unfolding if-dens-det-def
proof (intro borel-measurable-times-ennreal assms measurable-If)
have {x \in space (state-measure V Γ). expr-sem-rf x e = BoolVal b} =
 ($\lambda \sigma. \text{expr-sem-rf } \sigma$ e) - ' {BoolVal b} \cap space (state-measure V Γ) **by**
 auto
also have ... \in sets (state-measure V Γ)
by (rule measurable-sets, rule measurable-expr-sem-rf[OF e]) simp-all
finally show {x \in space (state-measure V Γ). expr-sem-rf x e = BoolVal b}
 \in sets (state-measure V Γ) .

qed simp-all

locale density-context =

fixes V V' Γ δ
assumes subprob-space-dens:
 $\bigwedge \rho. \rho \in$ space (state-measure V' Γ) \implies subprob-space (dens-ctxt-measure
 (V, V', Γ , δ) ρ)
and finite-vars[simp]: finite V finite V'
and measurable-dens[measurable]:
 $\delta \in$ borel-measurable (state-measure (V \cup V') Γ)
and disjoint: V \cap V' = {}

begin

abbreviation $\mathcal{Y} \equiv (V, V', \Gamma, \delta)$

lemma *branch-prob-altdef*:

assumes $\varrho: \varrho \in \text{space } (\text{state-measure } V' \Gamma)$

shows *branch-prob* $\mathcal{Y} \varrho = \int^+ x. \delta (\text{merge } V V' (x, \varrho)) \partial \text{state-measure } V \Gamma$

proof –

have *branch-prob* $\mathcal{Y} \varrho =$

$\int^+ x. \delta (\text{merge } V V' (x, \varrho)) * \text{indicator } (\text{space } (\text{state-measure } (V \cup V')$

$\Gamma))$

$(\text{merge } V V' (x, \varrho)) \partial \text{state-measure } V \Gamma$

using ϱ **unfolding** *branch-prob-def*[*abs-def*] *dens-ctxt-measure-def* *state-measure'-def*

by (*simp add: emeasure-density ennreal-mult'' ennreal-indicator nn-integral-distr*)

also from ϱ **have** $\dots = \int^+ x. \delta (\text{merge } V V' (x, \varrho)) \partial \text{state-measure } V \Gamma$

by (*intro nn-integral-cong*) (*simp split: split-indicator add: merge-in-state-measure*)

finally show *?thesis* .

qed

lemma *measurable-branch-prob*[*measurable*]:

branch-prob $\mathcal{Y} \in \text{borel-measurable } (\text{state-measure } V' \Gamma)$

proof –

interpret *sigma-finite-measure* *state-measure* $V \Gamma$ **by** *auto*

show *?thesis*

by (*simp add: branch-prob-altdef cong: measurable-cong*)

qed

lemma *measurable-marg-dens'*:

assumes [*simp*]: $x \in V$

shows *case-prod* (*marg-dens* $\mathcal{Y} x$) $\in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } (\Gamma x))$

proof –

interpret *sigma-finite-measure* *state-measure* $(V - \{x\}) \Gamma$

unfolding *state-measure-def*

by (*rule product-sigma-finite.sigma-finite, simp-all add: product-sigma-finite-def*)

from *assms* **have** $V = \text{insert } x (V - \{x\})$ **by** *blast*

hence $A: \text{PiM } V = \text{PiM } \dots$ **by** *simp*

show *?thesis* **unfolding** *marg-dens-def*

by (*simp add: insert-absorb*)

qed

lemma *insert-Diff*: $\text{insert } x (A - B) = \text{insert } x A - (B - \{x\})$

by *auto*

lemma *measurable-marg-dens2'*:

assumes $x \in V y \in V$

shows *case-prod* (*marg-dens2* $\mathcal{Y} x y$) \in

$\text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } (\text{PRODUCT } (\Gamma x) (\Gamma y)))$

proof –
interpret *sigma-finite-measure state-measure* $(V - \{x, y\}) \Gamma$
unfolding *state-measure-def*
by (*rule product-sigma-finite.sigma-finite, simp-all add: product-sigma-finite-def*)
have [*measurable*]: $V = \text{insert } x (V - \{x, y\}) \cup \{y\}$
using *assms by blast*
show *?thesis unfolding marg-dens2-def*
by *simp*
qed

lemma *measurable-marg-dens*:
assumes $x \in V \ \varrho \in \text{space } (\text{state-measure } V' \Gamma)$
shows $\text{marg-dens } \mathcal{Y} \ x \ \varrho \in \text{borel-measurable } (\text{stock-measure } (\Gamma \ x))$
using *assms by (intro measurable-Pair-compose-split[OF measurable-marg-dens'])*
simp-all

lemma *measurable-marg-dens2*:
assumes $x \in V \ y \in V \ x \neq y \ \varrho \in \text{space } (\text{state-measure } V' \Gamma)$
shows $\text{marg-dens2 } \mathcal{Y} \ x \ y \ \varrho \in \text{borel-measurable } (\text{stock-measure } (\text{PRODUCT } (\Gamma \ x) (\Gamma \ y)))$
using *assms by (intro measurable-Pair-compose-split[OF measurable-marg-dens2'])*
simp-all

lemma *measurable-state-measure-component*:
 $x \in V \implies (\lambda \sigma. \sigma \ x) \in \text{measurable } (\text{state-measure } V \Gamma) (\text{stock-measure } (\Gamma \ x))$
unfolding *state-measure-def*
by (*auto intro!: measurable-component-singleton*)

lemma *measurable-dens-ctxt-measure-component*:
 $x \in V \implies (\lambda \sigma. \sigma \ x) \in \text{measurable } (\text{dens-ctxt-measure } (V, V', \Gamma, \delta) \ \varrho) (\text{stock-measure } (\Gamma \ x))$
unfolding *dens-ctxt-measure-def state-measure'-def state-measure-def*
by (*auto intro!: measurable-component-singleton*)

lemma *space-dens-ctxt-measure-dens-ctxt-measure'*:
assumes $x \in V$
shows $\text{space } (\text{state-measure } V \Gamma) = \{\sigma(x := y) \mid \sigma \ y. \ \sigma \in \text{space } (\text{state-measure } (V - \{x\}) \Gamma) \wedge y \in \text{type-universe } (\Gamma \ x)\}$

proof –
from *assms have insert x (V - {x}) = V by auto*
hence $\text{state-measure } V \Gamma = \text{Pi}_M (\text{insert } x (V - \{x\})) (\lambda y. \text{stock-measure } (\Gamma \ y))$
unfolding *state-measure-def by simp*
also have $\text{space } \dots = \{\sigma(x := y) \mid \sigma \ y. \ \sigma \in \text{space } (\text{state-measure } (V - \{x\}) \Gamma) \wedge y \in \text{type-universe } (\Gamma \ x)\}$
unfolding *state-measure-def space-PiM PiE-insert-eq*
by (*simp add: image-def Bex-def blast*)
finally show *?thesis .*
qed

lemma *state-measure-integral-split*:

assumes $x \in A$ *finite* A

assumes $f \in$ *borel-measurable* (*state-measure* $A \Gamma$)

shows $(\int^+ \sigma. f \sigma \partial$ *state-measure* $A \Gamma) =$

$$(\int^+ y. \int^+ \sigma. f (\sigma(x := y)) \partial$$
state-measure $(A - \{x\}) \Gamma \partial$ *stock-measure*

$(\Gamma x))$

proof –

interpret *product-sigma-finite* $\lambda y. \text{stock-measure } (\Gamma y)$

unfolding *product-sigma-finite-def* **by** *auto*

from *assms* **have** [*simp*]: *insert* $x A = A$ **by** *auto*

have $(\int^+ \sigma. f \sigma \partial$ *state-measure* $A \Gamma) = (\int^+ \sigma. f \sigma \partial \Pi_M v \in \text{insert } x (A - \{x\}).$
stock-measure $(\Gamma v))$

unfolding *state-measure-def* **by** *simp*

also have $\dots = \int^+ y. \int^+ \sigma. f (\sigma(x := y)) \partial$ *state-measure* $(A - \{x\}) \Gamma \partial$ *stock-measure*
 (Γx)

using *assms* **unfolding** *state-measure-def*

by (*subst product-nn-integral-insert-rev*) *simp-all*

finally show *?thesis* .

qed

lemma *fun-upd-in-state-measure*:

$[\sigma \in$ *space* (*state-measure* $A \Gamma$); $y \in$ *space* (*stock-measure* $(\Gamma x))]$

$\implies \sigma(x := y) \in$ *space* (*state-measure* (*insert* $x A$) Γ)

unfolding *state-measure-def* **by** (*auto simp: space-PiM split: if-split-asm*)

lemma *marg-dens-integral*:

fixes $X :: \text{val set}$ **assumes** $x \in V$ **and** [*measurable*]: $X \in$ *sets* (*stock-measure* (Γx))

assumes $\varrho \in$ *space* (*state-measure* $V' \Gamma$)

defines $X' \equiv (\lambda \sigma. \sigma x) -' X \cap$ *space* (*state-measure* $V \Gamma$)

shows $(\int^+ y. \text{marg-dens } \mathcal{Y} x \varrho y * \text{indicator } X y \partial$ *stock-measure* $(\Gamma x)) =$

$$(\int^+ \sigma. \delta (\text{merge } V V' (\sigma, \varrho)) * \text{indicator } X' \sigma \partial$$
state-measure $V \Gamma)$

proof –

from *assms* **have** [*simp*]: *insert* $x V = V$ **by** *auto*

interpret *product-sigma-finite* $\lambda y. \text{stock-measure } (\Gamma y)$

unfolding *product-sigma-finite-def* **by** *auto*

have $(\int^+ \sigma. \delta (\text{merge } V V' (\sigma, \varrho)) * \text{indicator } X' \sigma \partial$ *state-measure* $V \Gamma) =$

$$\int^+ y. \int^+ \sigma. \delta (\text{merge } V V' (\sigma(x := y), \varrho)) * \text{indicator } X' (\sigma(x := y))$$

 ∂ *state-measure* $(V - \{x\}) \Gamma \partial$ *stock-measure* (Γx) **using** *assms(1-3)*

by (*subst state-measure-integral-split[of x]*) (*auto simp: X'-def*)

also have $\dots = \int^+ y. \int^+ \sigma. \delta (\text{merge } V V' (\sigma(x := y), \varrho)) * \text{indicator } X y$
 ∂ *state-measure* $(V - \{x\}) \Gamma \partial$ *stock-measure* (Γx)

by (*intro nn-integral-cong*)

(*auto simp: X'-def split: split-indicator dest: fun-upd-in-state-measure*)

also have $\dots = (\int^+ y. \text{marg-dens } \mathcal{Y} x \varrho y * \text{indicator } X y \partial$ *stock-measure* $(\Gamma x))$

using *measurable-dens-ctxt-fun-upd* **unfolding** *marg-dens-def* **using** *assms(1-3)*

by (*intro nn-integral-cong*) (*simp split: split-indicator*)

finally show *?thesis ..*
qed

lemma *marg-dens2-integral:*

fixes $X :: \text{val set}$
assumes $x \in V \ y \in V \ x \neq y$ **and** *[measurable]:* $X \in \text{sets (stock-measure (PRODUCT (\Gamma x) (\Gamma y)))}$
assumes $\rho \in \text{space (state-measure V' \Gamma)}$
defines $X' \equiv (\lambda \sigma. \langle |\sigma x, \sigma y| \rangle) - ' X \cap \text{space (state-measure V \Gamma)}$
shows $(\int^+ z. \text{marg-dens2 } \mathcal{Y} \ x \ y \ \rho \ z * \text{indicator } X \ z \ \partial \text{stock-measure (PRODUCT (\Gamma x) (\Gamma y)))} =$
 $(\int^+ \sigma. \delta (\text{merge } V \ V' (\sigma, \rho)) * \text{indicator } X' \ \sigma \ \partial \text{state-measure } V \ \Gamma)$

proof –

let $?M = \text{stock-measure (PRODUCT (\Gamma x) (\Gamma y))}$
let $?M' = \text{stock-measure (\Gamma x) } \otimes_M \text{ stock-measure (\Gamma y)}$
interpret *product-sigma-finite* $\lambda x. \text{stock-measure (\Gamma x)}$
unfolding *product-sigma-finite-def* **by** *simp*
from *assms* **have** $(\int^+ z. \text{marg-dens2 } \mathcal{Y} \ x \ y \ \rho \ z * \text{indicator } X \ z \ \partial ?M) =$
 $\int^+ z. \text{marg-dens2 } \mathcal{Y} \ x \ y \ \rho \ (\text{case-prod } \text{PairVal } z) * \text{indicator } X \ (\text{case-prod } \text{PairVal } z) \ \partial ?M'$
by *(subst nn-integral-PairVal)*
(auto simp add: split-beta' intro!: borel-measurable-times-ennreal measurable-marg-dens2)

have $V'' : V - \{x, y\} = V - \{y\} - \{x\}$
by *auto*

from *assms* **have** $A : V = \text{insert } y \ (V - \{y\})$ **by** *blast*

from *assms* **have** $B : \text{insert } x \ (V - \{x, y\}) = V - \{y\}$ **by** *blast*

from *assms* **have** $X'[\text{measurable}] : X' \in \text{sets (state-measure V \Gamma)}$ **unfolding** $X'\text{-def}$

by *(intro measurable-sets[OF - assms(4)], unfold state-measure-def, subst stock-measure.simps)*
(rule measurable-Pair-compose-split[OF measurable-embed-measure2], rule inj-PairVal,
erule measurable-component-singleton, erule measurable-component-singleton)

have $V[\text{simp}] : \text{insert } y \ (V - \{y\}) = V \ \text{insert } x \ (V - \{x, y\}) = V - \{y\} \ \text{insert } y \ V = V$

and *[measurable]:* $x \in V - \{y\}$

using *assms* **by** *auto*

have $(\int^+ \sigma. \delta (\text{merge } V \ V' (\sigma, \rho)) * \text{indicator } X' \ \sigma \ \partial \text{state-measure } V \ \Gamma) =$
 $(\int^+ \sigma. \delta (\text{merge } V \ V' (\sigma, \rho)) * \text{indicator } X' \ \sigma \ \partial \text{state-measure (insert } y \ (\text{insert } x \ (V - \{x, y\}))) \ \Gamma)$

using *assms* **by** *(intro arg-cong2[where f=nn-integral] arg-cong2[where f=state-measure])*
auto

also **have** $\dots = \int^+ w. \int^+ v. \int^+ \sigma. \delta (\text{merge } V \ V' (\sigma(x := v, y := w), \rho)) * \text{indicator } X' (\sigma(x := v, y := w)) \ \partial \text{state-measure } (V - \{x, y\}) \ \Gamma \ \partial \text{stock-measure } (\Gamma x) \ \partial \text{stock-measure } (\Gamma y))$

```

(is - = ?I)
  unfolding state-measure-def
  using assms
  apply (subst product-nn-integral-insert-rev)
  apply (auto simp: state-measure-def[symmetric])
  apply (rule nn-integral-cong)
  apply (subst state-measure-def)
  apply (subst V(2)[symmetric])
  apply (subst product-nn-integral-insert-rev)
  apply (auto simp: state-measure-def[symmetric])
  apply measurable
  apply simp-all
  done
also from assms(1-5)
  have  $\bigwedge v w \sigma. v \in \text{space } (\text{stock-measure } (\Gamma x)) \implies w \in \text{space } (\text{stock-measure } (\Gamma y))$ 
     $\implies \sigma \in \text{space } (\text{state-measure } (V - \{x, y\}) \Gamma)$ 
     $\implies \sigma(x := v, y := w) \in X' \longleftrightarrow \langle |v, w| \rangle \in X$ 
  by (simp add: X'-def space-state-measure PiE-iff extensional-def)
  hence ?I =  $\int^+ w. \int^+ v. \int^+ \sigma. \delta (\text{merge } V V' (\sigma(x := v, y := w), \varrho)) * \text{indicator } X \langle |v, w| \rangle$ 
     $\partial \text{state-measure } (V - \{x, y\}) \Gamma \partial \text{stock-measure } (\Gamma x) \partial \text{stock-measure } (\Gamma y)$ 
  by (intro nn-integral-cong) (simp split: split-indicator)
  also from assms(5)
  have ... =  $\int^+ w. \int^+ v. (\int^+ \sigma. \delta (\text{merge } V V' (\sigma(x := v, y := w), \varrho)) \partial \text{state-measure } (V - \{x, y\}) \Gamma)$ 
     $* \text{indicator } X \langle |v, w| \rangle \partial \text{stock-measure } (\Gamma x) \partial \text{stock-measure } (\Gamma y)$ 
  using assms
  apply (simp add: ennreal-mult'' ennreal-indicator)
  by (intro nn-integral-cong nn-integral-multc) (simp-all add: )
  also have ... =  $\int^+ w. \int^+ v. \text{marg-dens2 } \mathcal{Y} x y \varrho \langle |v, w| \rangle * \text{indicator } X \langle |v, w| \rangle$ 
     $\partial \text{stock-measure } (\Gamma x) \partial \text{stock-measure } (\Gamma y)$ 
  by (intro nn-integral-cong) (simp add: marg-dens2-def)
  also from assms(4)
  have ... =  $\int^+ z. \text{marg-dens2 } \mathcal{Y} x y \varrho (\text{case-prod PairVal } z) * \text{indicator } X (\text{case-prod PairVal } z)$ 
     $\partial (\text{stock-measure } (\Gamma x) \otimes_M \text{stock-measure } (\Gamma y))$ 
  using assms
  by (subst pair-sigma-finite.nn-integral-snd[symmetric])
    (auto simp add: pair-sigma-finite-def intro!: borel-measurable-times-ennreal measurable-compose[OF - measurable-marg-dens2])
  also have ... =  $\int^+ z. \text{marg-dens2 } \mathcal{Y} x y \varrho z * \text{indicator } X z \partial \text{stock-measure } (\text{PRODUCT } (\Gamma x) (\Gamma y))$ 
  apply (subst stock-measure.simps, subst embed-measure-eq-distr, rule inj-PairVal)
  apply (rule nn-integral-distr[symmetric], intro measurable-embed-measure2 inj-PairVal)
  apply (subst stock-measure.simps[symmetric])
  apply (intro borel-measurable-times-ennreal)

```

```

apply simp
apply (intro measurable-marg-dens2)
apply (insert assms)
apply simp-all
done
finally show ?thesis ..
qed

```

The space described by the marginal density is the same as the space obtained by projecting x (resp. x and y) out of the common distribution of all variables.

lemma *density-marg-dens-eq*:

```

assumes  $x \in V \ \varrho \in \text{space } (\text{state-measure } V' \Gamma)$ 
shows  $\text{density } (\text{stock-measure } (\Gamma \ x)) \ (\text{marg-dens } \mathcal{Y} \ x \ \varrho) =$ 
 $\text{distr } (\text{dens-ctxt-measure } (V, V', \Gamma, \delta) \ \varrho) \ (\text{stock-measure } (\Gamma \ x)) \ (\lambda\sigma. \ \sigma \ x)$ 
(is ?M1 = ?M2)

```

proof (*rule measure-eqI*)

```

fix  $X$  assume  $X: X \in \text{sets } ?M1$ 
let  $?X' = (\lambda\sigma. \ \sigma \ x) - ' X \cap \text{space } (\text{state-measure } V \Gamma)$ 
let  $?X'' = (\lambda\sigma. \ \sigma \ x) - ' X \cap \text{space } (\text{state-measure } (V \cup V') \Gamma)$ 
from  $X$  have  $\text{emeasure } ?M1 \ X = \int^+ \sigma. \ \delta \ (\text{merge } V \ V' \ (\sigma, \ \varrho)) * \text{indicator } ?X'$ 
 $\sigma \ \partial \text{state-measure } V \ \Gamma$ 

```

using *assms measurable-marg-dens measurable-dens*

by (*subst emeasure-density*)

(*auto simp: emeasure-distr nn-integral-distr*

dens-ctxt-measure-def state-measure'-def emeasure-density marg-dens-integral)

```

also from assms have  $\dots = \int^+ \sigma. \ \delta \ (\text{merge } V \ V' \ (\sigma, \ \varrho)) * \text{indicator } ?X''$ 
 $(\text{merge } V \ V' \ (\sigma, \ \varrho)) \ \partial \text{state-measure}$ 

```

$V \ \Gamma$

by (*intro nn-integral-cong*)

(*auto split: split-indicator simp: space-state-measure merge-def PiE-iff extensional-def*)

also from X **and** *assms* **have** $\dots = \text{emeasure } ?M2 \ X$ **using** *measurable-dens*

by (*auto simp: emeasure-distr emeasure-density nn-integral-distr ennreal-indicator ennreal-mult''*

dens-ctxt-measure-def state-measure'-def state-measure-def)

finally show $\text{emeasure } ?M1 \ X = \text{emeasure } ?M2 \ X$.

qed *simp*

lemma *density-marg-dens2-eq*:

```

assumes  $x \in V \ y \in V \ x \neq y \ \varrho \in \text{space } (\text{state-measure } V' \Gamma)$ 
defines  $M \equiv \text{stock-measure } (\text{PRODUCT } (\Gamma \ x) \ (\Gamma \ y))$ 
shows  $\text{density } M \ (\text{marg-dens2 } \mathcal{Y} \ x \ y \ \varrho) =$ 
 $\text{distr } (\text{dens-ctxt-measure } (V, V', \Gamma, \delta) \ \varrho) \ M \ (\lambda\sigma. \ \langle |\sigma \ x, \sigma \ y| \rangle)$  (is ?M1
 $= ?M2)$ 

```

proof (*rule measure-eqI*)

fix X **assume** $X: X \in \text{sets } ?M1$

let $?X' = (\lambda\sigma. \ \langle |\sigma \ x, \sigma \ y| \rangle) - ' X \cap \text{space } (\text{state-measure } V \ \Gamma)$

let $?X'' = (\lambda\sigma. \ \langle |\sigma \ x, \sigma \ y| \rangle) - ' X \cap \text{space } (\text{state-measure } (V \cup V') \ \Gamma)$

from *assms* **have** *meas*[*measurable*]: $(\lambda\sigma. \langle |\sigma x, \sigma y| \rangle) \in \text{measurable } (\text{state-measure } (V \cup V') \Gamma)$
 $(\text{stock-measure } (\text{PRODUCT } (\Gamma x) (\Gamma y)))$
unfolding *state-measure-def*
apply (*subst stock-measure.simps*)
apply (*rule measurable-Pair-compose-split*[*OF measurable-embed-measure2*[*OF inj-PairVal*]])
apply (*rule measurable-component-singleton, simp*)
done
from *assms*($1-4$) *X meas* **have** *emeasure ?M2 X = emeasure (dens-ctxt-measure \mathcal{Y} ϱ) ?X''*
apply (*subst emeasure-distr*)
apply (*subst measurable-dens-ctxt-measure-eq, unfold state-measure-def M-def*)
apply (*simp-all add: space-dens-ctxt-measure state-measure-def*)
done
also from *assms*($1-4$) *X meas*
have $\dots = \int^+ \sigma. \delta (\text{merge } V V' (\sigma, \varrho)) * \text{indicator } ?X'' (\text{merge } V V' (\sigma, \varrho))$
 $\partial \text{state-measure } V \Gamma$
(is - = ?I) unfolding *dens-ctxt-measure-def state-measure'-def M-def*
by (*simp add: emeasure-density nn-integral-distr ennreal-indicator ennreal-mult''*)
also from *assms*($1-4$) *X*
have $\bigwedge \sigma. \sigma \in \text{space } (\text{state-measure } V \Gamma) \implies \text{merge } V V' (\sigma, \varrho) \in ?X'' \longleftrightarrow \sigma \in ?X'$
by (*auto simp: space-state-measure merge-def PiE-iff extensional-def*)
hence $?I = \int^+ \sigma. \delta (\text{merge } V V' (\sigma, \varrho)) * \text{indicator } ?X' \sigma \partial \text{state-measure } V \Gamma$
by (*intro nn-integral-cong*) (*simp split: split-indicator*)
also from *assms X* **have** $\dots = \int^+ z. \text{marg-dens2 } \mathcal{Y} x y \varrho z * \text{indicator } X z \partial M$
unfolding *M-def*
by (*subst marg-dens2-integral*) *simp-all*
also from *X* **have** $\dots = \text{emeasure } ?M1 X$
using *assms measurable-dens unfolding M-def*
by (*subst emeasure-density, intro measurable-marg-dens2*) *simp-all*
finally show *emeasure ?M1 X = emeasure ?M2 X ..*
qed *simp*

lemma *measurable-insert-dens*[*measurable*]:
assumes *Mf*[*measurable*]: *case-prod f* $\in \text{borel-measurable } (\text{state-measure } (V \cup V') \Gamma \otimes_M \text{stock-measure } t)$
shows *insert-dens V V' f* δ
 $\in \text{borel-measurable } (\text{state-measure } (\text{shift-var-set } (V \cup V')) (\text{case-nat } t \Gamma))$
proof –
have $(\lambda\sigma. \sigma 0) \in \text{measurable } (\text{state-measure } (\text{shift-var-set } (V \cup V')) (\text{case-nat } t \Gamma))$
 $(\text{stock-measure } (\text{case-nat } t \Gamma 0))$ **unfolding** *state-measure-def*
unfolding *shift-var-set-def* **by** *measurable*
thus *?thesis* **unfolding** *insert-dens-def*[*abs-def*] **by** *simp*

qed

lemma *nn-integral-dens-ctxt-measure*:

assumes $\varrho \in \text{space } (\text{state-measure } V' \Gamma)$

$f \in \text{borel-measurable } (\text{state-measure } (V \cup V') \Gamma)$

shows $(\int^+ x. f x \partial \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho) =$

$\int^+ x. \delta (\text{merge } V V' (x, \varrho)) * f (\text{merge } V V' (x, \varrho)) \partial \text{state-measure } V \Gamma$

unfolding *dens-ctxt-measure-def state-measure'-def* **using** *assms measurable-dens*
by (*simp only: prod.case, subst nn-integral-density*)

(*auto simp: nn-integral-distr state-measure-def*)

lemma *shift-var-set-Un[simp]*: *shift-var-set* $V \cup \text{Suc } 'V' = \text{shift-var-set } (V \cup V')$

unfolding *shift-var-set-def* **by** (*simp add: image-Un*)

lemma *emeasure-dens-ctxt-measure-insert*:

fixes $t f \varrho$

defines $M \equiv \text{dens-ctxt-measure } (\text{shift-var-set } V, \text{Suc } 'V', \text{case-nat } t \Gamma, \text{insert-dens } V V' f \delta) \varrho$

assumes *dens: has-parametrized-subprob-density* (*state-measure* $(V \cup V') \Gamma$) F
(*stock-measure* t) f

assumes $\varrho: \varrho \in \text{space } (\text{state-measure } (\text{Suc } 'V') (\text{case-nat } t \Gamma))$

assumes $X: X \in \text{sets } M$

shows *emeasure* $M X =$

$\int^+ x. \text{insert-dens } V V' f \delta (\text{merge } (\text{shift-var-set } V) (\text{Suc } 'V') (x, \varrho)) *$
 $\text{indicator } X (\text{merge } (\text{shift-var-set } V) (\text{Suc } 'V') (x, \varrho))$
 $\partial \text{state-measure } (\text{shift-var-set } V) (\text{case-nat } t \Gamma) (\text{is } - = ?I)$

proof–

note [*measurable*] = *has-parametrized-subprob-density* $D(\beta)[OF \text{dens}]$

have [*measurable*]:

$(\lambda \sigma. \text{merge } (\text{shift-var-set } V) (\text{Suc } 'V') (\sigma, \varrho))$

$\in \text{measurable } (\text{state-measure } (\text{shift-var-set } V) (\text{case-nat } t \Gamma))$

$(\text{state-measure } (\text{shift-var-set } (V \cup V')) (\text{case-nat } t \Gamma))$

using ϱ **unfolding** *state-measure-def*

by (*simp del: shift-var-set-Un add: shift-var-set-Un[symmetric]*)

from *assms* **have** *emeasure* $M X = (\int^+ x. \text{indicator } X x \partial M)$

by (*subst nn-integral-indicator*)

(*simp-all add: dens-ctxt-measure-def state-measure'-def*)

also have $MI: \text{indicator } X \in \text{borel-measurable}$

$(\text{state-measure } (\text{shift-var-set } (V \cup V')) (\text{case-nat } t \Gamma))$

using X **unfolding** *M-def dens-ctxt-measure-def state-measure'-def* **by** *simp*

have $(\int^+ x. \text{indicator } X x \partial M) = ?I$

using X **unfolding** *M-def dens-ctxt-measure-def state-measure'-def*

apply (*simp only: prod.case*)

apply (*subst nn-integral-density*)

apply (*simp-all add: nn-integral-density nn-integral-distr MI*)

done

finally show *?thesis* .

qed

lemma *merge-Suc-aux'*:

$\varrho \in \text{space } (\text{state-measure } (\text{Suc } ' V') (\text{case-nat } t \Gamma)) \implies$
 $(\lambda\sigma. \text{merge } V V' (\sigma, \varrho \circ \text{Suc})) \in \text{measurable } (\text{state-measure } V \Gamma) (\text{state-measure } (V \cup V') \Gamma)$
by (*unfold state-measure-def*,
rule measurable-compose[OF measurable-Pair measurable-merge], *simp*,
rule measurable-const, auto simp: space-PiM dest: PiE-mem)

lemma *merge-Suc-aux*:

$\varrho \in \text{space } (\text{state-measure } (\text{Suc } ' V') (\text{case-nat } t \Gamma)) \implies$
 $(\lambda\sigma. \delta (\text{merge } V V' (\sigma, \varrho \circ \text{Suc}))) \in \text{borel-measurable } (\text{state-measure } V \Gamma)$
by (*rule measurable-compose[OF - measurable-dens]*, *unfold state-measure-def*,
rule measurable-compose[OF measurable-Pair measurable-merge], *simp*,
rule measurable-const, auto simp: space-PiM dest: PiE-mem)

lemma *nn-integral-PiM-Suc*:

assumes *fin*: $\bigwedge i. \text{sigma-finite-measure } (N i)$
assumes *Mf*: $f \in \text{borel-measurable } (Pi_M V N)$
shows $(\int^{+x}. f x \partial \text{distr } (Pi_M (\text{Suc}' V)) (\text{case-nat } M N)) (Pi_M V N) (\lambda\sigma. \sigma \circ \text{Suc}) =$
 $(\int^{+x}. f x \partial Pi_M V N)$
(is nn-integral (?M1 V) - = -)

using *Mf*

proof (*induction arbitrary: f*

rule: finite-induct[OF finite-vars(1), case-names empty insert])

case *empty*

show *?case* **by** (*auto simp add: PiM-empty nn-integral-distr intro!: nn-integral-cong*)

next

case (*insert v V*)

let *?V* = *insert v V* **and** *?M3* = $Pi_M (\text{insert } (\text{Suc } v) (\text{Suc } ' V)) (\text{case-nat } M N)$

let *?M4* = $Pi_M (\text{insert } (\text{Suc } v) (\text{Suc } ' V)) (\text{case-nat } (\text{count-space } \{\}) N)$

let *?M4'* = $Pi_M (\text{Suc } ' V) (\text{case-nat } (\text{count-space } \{\}) N)$

have *A*: *?M3* = *?M4* **by** (*intro PiM-cong auto*)

interpret *product-sigma-finite case-nat (count-space {})* *N*

unfolding *product-sigma-finite-def*

by (*auto intro: fin sigma-finite-measure-count-space-countable split: nat.split*)

interpret *sigma-finite-measure N v* **by** (*rule assms*)

note *Mf[measurable] = insert(4)*

from *insert* **have** $(\int^{+x}. f x \partial ?M1 ?V) = \int^{+x}. f (x \circ \text{Suc}) \partial ?M4$

by (*subst A[symmetric]*, *subst nn-integral-distr*)

(simp-all add: measurable-case-nat-Suc-PiM image-insert[symmetric] del:

image-insert)

also from *insert* **have** $\dots = \int^{+x}. \int^{+y}. f (x(\text{Suc } v := y) \circ \text{Suc}) \partial N v \partial ?M4'$

apply (*subst product-nn-integral-insert, simp, blast, subst image-insert[symmetric]*)

apply (*erule measurable-compose[OF measurable-case-nat-Suc-PiM]*, *simp*)

done

also have $(\lambda x y. x(\text{Suc } v := y) \circ \text{Suc}) = (\lambda x y. (x \circ \text{Suc})(v := y))$

by (intro ext) (simp add: o-def)
 also have $?M_4' = Pi_M (Suc \text{ ' } V) (case\text{-}nat\ M\ N)$ by (intro PiM-cong) auto
 also from insert have $(\int^{+x}. \int^{+y}. f ((x \circ Suc)(v := y)) \partial N\ v\ \partial \dots) =$
 $(\int^{+x}. \int^{+y}. f (x(v := y)) \partial N\ v\ \partial ?M_1\ V)$
 by (subst nn-integral-distr)
 (simp-all add: borel-measurable-nn-integral measurable-case-nat-Suc-PiM)
 also from insert have $\dots = (\int^{+x}. \int^{+y}. f (x(v := y)) \partial N\ v\ \partial Pi_M\ V\ N)$
 by (intro insert(β)) measurable
 also from insert have $\dots = (\int^{+x}. f\ x\ \partial Pi_M\ ?V\ N)$
 by (subst product-sigma-finite.product-nn-integral-insert)
 (simp-all add: assms product-sigma-finite-def)
 finally show ?case .
 qed

lemma PiM-Suc:

assumes $\bigwedge i. \text{sigma-finite-measure } (N\ i)$
 shows $distr (Pi_M (Suc \text{ ' } V) (case\text{-}nat\ M\ N)) (Pi_M\ V\ N) (\lambda \sigma. \sigma \circ Suc) = Pi_M$
 $V\ N$ (is ?M1 = ?M2)
 by (intro measure-eqI)
 (simp-all add: nn-integral-indicator[symmetric] nn-integral-PiM-Suc assms
 del: nn-integral-indicator)

lemma distr-state-measure-Suc:

$distr (state\text{-}measure (Suc \text{ ' } V) (case\text{-}nat\ t\ \Gamma)) (state\text{-}measure\ V\ \Gamma) (\lambda \sigma. \sigma \circ Suc)$
 $=$
 $state\text{-}measure\ V\ \Gamma$ (is ?M1 = ?M2)
 unfolding state-measure-def
 apply (subst (2) PiM-Suc[*of* $\lambda x. \text{stock-measure } (\Gamma\ x)\ \text{stock-measure } t, \text{symmet-}$
ric], simp)
 apply (intro distr-cong PiM-cong)
 apply (simp-all split: nat.split)
 done

lemma emeasure-dens-ctxt-measure-insert':

fixes $t\ f\ \varrho$
 defines $M \equiv \text{dens-ctxt-measure } (\text{shift-var-set } V, Suc \text{ ' } V', case\text{-}nat\ t\ \Gamma, \text{insert-dens}$
 $V\ V'\ f\ \delta)\ \varrho$
 assumes $\text{dens: has-parametrized-subprob-density } (state\text{-}measure\ (V \cup V')\ \Gamma)\ F$
(stock-measure t) f
 assumes $\varrho: \varrho \in \text{space } (state\text{-}measure\ (Suc \text{ ' } V') (case\text{-}nat\ t\ \Gamma))$
 assumes $X: X \in \text{sets } M$
 shows $\text{emeasure } M\ X = \int^{+\sigma}. \delta (\text{merge } V\ V' (\sigma, \varrho \circ Suc)) * (\int^{+y}. f (\text{merge } V$
 $V' (\sigma, \varrho \circ Suc))\ y *$
 $\text{indicator } X (\text{merge } (\text{shift-var-set } V) (Suc \text{ ' } V') (case\text{-}nat\ y\ \sigma, \varrho))$
 $\partial \text{stock-measure } t) \partial \text{state-measure } V\ \Gamma$ (is - = ?I)

proof –

let $?m = \lambda x\ y. \text{merge } (\text{insert } 0\ (Suc \text{ ' } V)) (Suc \text{ ' } V') (x(0 := y), \varrho)$
 from dens have Mf :
 $\text{case-prod } f \in \text{borel-measurable } (state\text{-}measure\ (V \cup V')\ \Gamma) \otimes_M \text{stock-measure}$

t)

by (*rule has-parametrized-subprob-densityD*)

note [*measurable*] = *Mf*[*unfolded state-measure-def*]

have *meas-merge*: ($\lambda x. \text{merge } (\text{shift-var-set } V) (\text{Suc } V') (x, \varrho)$)
 \in *measurable* (*state-measure* (*shift-var-set* *V*) (*case-nat* *t* Γ))
(*state-measure* (*shift-var-set* ($V \cup V'$)) (*case-nat* *t* Γ))

using ϱ **unfolding** *state-measure-def* *shift-var-set-def*

by (*simp* *add*: *image-Un* *image-insert*[*symmetric*] *Un-insert-left*[*symmetric*]
del: *image-insert* *Un-insert-left*)

note *measurable-insert-dens'* =
measurable-insert-dens[*unfolded shift-var-set-def* *state-measure-def*]

have *meas-merge'*: ($\lambda x. \text{merge } (\text{shift-var-set } V) (\text{Suc } V') (\text{case-nat } (\text{snd } x) (\text{fst } x), \varrho)$)
 \in *measurable* (*state-measure* *V* $\Gamma \otimes_M$ *stock-measure* *t*)
(*state-measure* (*shift-var-set* ($V \cup V'$)) (*case-nat* *t* Γ))

by (*rule measurable-compose*[*OF - meas-merge*]) *simp*

have *meas-integral*: ($\lambda \sigma. \int^+ y. \delta (\text{merge } V V' (\sigma, \varrho \circ \text{Suc})) * f (\text{merge } V V' (\sigma, \varrho \circ \text{Suc})) y *$
indicator *X* (*merge* (*shift-var-set* *V*) (*Suc* V') (*case-nat* *y* σ, ϱ))

∂ *stock-measure* *t*) \in *borel-measurable* (*state-measure* *V* Γ)

apply (*rule sigma-finite-measure.borel-measurable-nn-integral*, *simp*)

apply (*subst measurable-split-conv*, *intro borel-measurable-times-ennreal*)

apply (*rule measurable-compose*[*OF measurable-fst merge-Suc-aux*[*OF* ϱ]])

apply (*rule measurable-Pair-compose-split*[*OF* *Mf*])

apply (*rule measurable-compose*[*OF measurable-fst merge-Suc-aux'*[*OF* ϱ]],
simp)

apply (*rule measurable-compose*[*OF meas-merge'* *borel-measurable-indicator*])

apply (*insert* *X*, *simp* *add*: *M-def dens-ctxt-measure-def state-measure'-def*)

done

have *meas'*: $\bigwedge x. x \in \text{space } (\text{state-measure } V \Gamma)$
 $\implies (\lambda y. f (\text{merge } V V' (x, \varrho \circ \text{Suc})) y *$
indicator *X* (*merge* (*shift-var-set* *V*) (*Suc* V') (*case-nat* *y* x, ϱ)))

\in *borel-measurable* (*stock-measure* *t*) **using** *X*

apply (*intro borel-measurable-times-ennreal*)

apply (*rule measurable-Pair-compose-split*[*OF* *Mf*])

apply (*rule measurable-const*, *erule measurable-space*[*OF merge-Suc-aux'*[*OF* ϱ]])

apply (*simp*, *rule measurable-compose*[*OF - borel-measurable-indicator*])

apply (*rule measurable-compose*[*OF measurable-case-nat'*])

apply (*rule measurable-ident-sets*[*OF refl*], *erule measurable-const*)

apply (*rule meas-merge*, *simp* *add*: *M-def dens-ctxt-measure-def state-measure'-def*)

done

have *emeasure* *M* *X* =
 $\int^+ x. \text{insert-dens } V V' f \delta (\text{merge } (\text{shift-var-set } V) (\text{Suc } V') (x, \varrho)) *$
indicator *X* (*merge* (*shift-var-set* *V*) (*Suc* V') (x, ϱ))

∂ *state-measure* (*shift-var-set* *V*) (*case-nat* *t* Γ)

using *assms* **unfolding** *M-def* **by** (*intro emeasure-dens-ctxt-measure-insert*)

also have ... = $\int^+ x. \int^+ y. \text{insert-dens } V \ V' \ f \ \delta \ (\text{?m } x \ y) *$
 $\text{indicator } X \ (\text{?m } x \ y) \ \partial\text{stock-measure } t \ \partial\text{state-measure } (\text{Suc}'V)$
(case-nat t Γ)
(is - = ?I) using $\varrho \ X \ \text{meas-merge}$
unfolding *shift-var-set-def M-def dens-ctxt-measure-def state-measure'-def state-measure-def*
apply *(subst product-sigma-finite.product-nn-integral-insert)*
apply *(auto simp: product-sigma-finite-def) [3]*
apply *(intro borel-measurable-times-ennreal)*
apply *(rule measurable-compose[OF - measurable-insert-dens¹], simp)*
apply *(simp-all add: measurable-compose[OF - borel-measurable-indicator] im-*
age-Un)
done
also have $\bigwedge \sigma \ y. \sigma \in \text{space } (\text{state-measure } (\text{Suc}'V) \ (\text{case-nat } t \ \Gamma)) \implies$
 $y \in \text{space } (\text{stock-measure } t) \implies$
 $(\text{remove-var } (\text{merge } (\text{insert } 0 \ (\text{Suc}'V)) \ (\text{Suc}'V')) \ (\sigma(\theta:=y), \varrho))$
=
 $\text{merge } V \ V' \ (\sigma \circ \text{Suc}, \varrho \circ \text{Suc})$
by *(auto simp: merge-def remove-var-def)*
hence $\text{?I} = \int^+ \sigma. \int^+ y. \delta \ (\text{merge } V \ V' \ (\sigma \circ \text{Suc}, \varrho \circ \text{Suc})) * f \ (\text{merge } V \ V' \ (\sigma$
 $\circ \text{Suc}, \varrho \circ \text{Suc})) \ y *$
 $\text{indicator } X \ (\text{?m } \sigma \ y)$
 $\partial\text{stock-measure } t \ \partial\text{state-measure } (\text{Suc}'V) \ (\text{case-nat } t \ \Gamma) \ (\text{is - = ?I})$
by *(intro nn-integral-cong)*
(auto simp: insert-dens-def inj-image-mem-iff merge-def split: split-indicator
nat.split)
also have $m\text{-eq}: \bigwedge x \ y. \text{?m } x \ y = \text{merge } (\text{shift-var-set } V) \ (\text{Suc}'V') \ (\text{case-nat } y \ (x$
 $\circ \text{Suc}), \varrho)$
by *(intro ext) (auto simp add: merge-def shift-var-set-def split: nat.split)*
have $\text{?I} = \int^+ \sigma. \int^+ y. \delta \ (\text{merge } V \ V' \ (\sigma, \varrho \circ \text{Suc})) * f \ (\text{merge } V \ V' \ (\sigma, \varrho \circ$
 $\text{Suc})) \ y *$
 $\text{indicator } X \ (\text{merge } (\text{shift-var-set } V) \ (\text{Suc}'V') \ (\text{case-nat } y \ \sigma, \varrho))$
 $\partial\text{stock-measure } t \ \partial\text{state-measure } V \ \Gamma \ \text{using } \varrho \ X$
apply *(subst distr-state-measure-Suc[symmetric, of t])*
apply *(subst nn-integral-distr)*
apply *(rule measurable-case-nat-Suc)*
apply *simp*
apply *(rule meas-integral)*
apply *(intro nn-integral-cong)*
apply *(simp add: m-eq)*
done
also have ... = $\int^+ \sigma. \delta \ (\text{merge } V \ V' \ (\sigma, \varrho \circ \text{Suc})) * (\int^+ y. f \ (\text{merge } V \ V' \ (\sigma,$
 $\varrho \circ \text{Suc})) \ y *$
 $\text{indicator } X \ (\text{merge } (\text{shift-var-set } V) \ (\text{Suc}'V') \ (\text{case-nat } y \ \sigma, \varrho))$
 $\partial\text{stock-measure } t \ \partial\text{state-measure } V \ \Gamma \ \text{using } \varrho \ X$
apply *(intro nn-integral-cong)*
apply *(subst nn-integral-cmult[symmetric])*
apply *(erule meas')*
apply *(simp add: mult.assoc)*
done

finally show *?thesis* .
qed

lemma *density-context-insert*:

assumes *dens*: *has-parametrized-subprob-density* (*state-measure* ($V \cup V'$) Γ) F
(*stock-measure* t) f

shows *density-context* (*shift-var-set* V) (*Suc* ' V') (*case-nat* t Γ) (*insert-dens* V
 $V' f \delta$)

(**is** *density-context* $?V ?V' ?\Gamma' ?\delta'$)

unfolding *density-context-def*

proof (*intro allI conjI impI*)

note *measurable-insert-dens*[*OF has-parametrized-subprob-densityD*(β)[*OF dens*]]

thus *insert-dens* $V V' f \delta$

\in *borel-measurable* (*state-measure* (*shift-var-set* $V \cup \text{Suc ' } V'$) (*case-nat* t
 Γ))

unfolding *shift-var-set-def* **by** (*simp only: image-Un Un-insert-left*)

next

fix ϱ **assume** ϱ : $\varrho \in \text{space}$ (*state-measure* $?V' ?\Gamma'$)

hence ϱ' : $\varrho \circ \text{Suc} \in \text{space}$ (*state-measure* $V' \Gamma$)

by (*auto simp: state-measure-def space-PiM dest: PiE-mem*)

note $\text{dens}' = \text{has-parametrized-subprob-densityD}$ [*OF dens*]

note $Mf[\text{measurable}] = \text{dens}'(\beta)$

have $M\text{-merge}$: (λx . *merge* (*shift-var-set* V) (*Suc* ' V') (x , ϱ))

\in *measurable* (Pi_M (*insert* 0 (*Suc* ' V)) (λy . *stock-measure* (*case-nat*
 $t \Gamma y$)))

(*state-measure* (*shift-var-set* ($V \cup V'$) (*case-nat* $t \Gamma$))

using ϱ **by** (*subst shift-var-set-Un[symmetric]*, *unfold state-measure-def*)

(*simp add: shift-var-set-def del: shift-var-set-Un Un-insert-left*)

show *subprob-space* (*dens-ctxt-measure* ($?V, ?V', ?\Gamma', ?\delta'$) ϱ) (**is** *subprob-space*
 $?M$)

proof (*rule subprob-spaceI*)

interpret *product-sigma-finite* (λy . *stock-measure* (*case y of* $0 \Rightarrow t \mid \text{Suc } x \Rightarrow$
 Γx))

by (*simp add: product-sigma-finite-def*)

have *Suc-state-measure*:

$\bigwedge x$. $x \in \text{space}$ (*state-measure* (*Suc* ' V) (*case-nat* $t \Gamma$)) \implies

merge $V V' (x \circ \text{Suc}, \varrho \circ \text{Suc}) \in \text{space}$ (*state-measure* ($V \cup V'$) Γ)

using ϱ

by (*intro merge-in-state-measure*) (*auto simp: state-measure-def space-PiM*
dest: PiE-mem)

have $S[\text{simp}]$: $\bigwedge x X$. $\text{Suc } x \in \text{Suc ' } X \iff x \in X$ **by** (*rule inj-image-mem-iff*)

simp

let $?M = \text{dens-ctxt-measure}$ ($?V, ?V', ?\Gamma', ?\delta'$) ϱ

from ϱ **have** $\bigwedge \sigma$. $\sigma \in \text{space}$ (*state-measure* $?V ?\Gamma'$) \implies *merge* $?V ?V' (\sigma, \varrho)$
 $\in \text{space } ?M$

by (*auto simp: dens-ctxt-measure-def state-measure'-def simp del: shift-var-set-Un*
intro!: merge-in-state-measure)

hence $\text{emeasure } ?M \text{ (space } ?M) =$
 $\int^{+\sigma}. \text{insert-dens } V \ V' \ f \ \delta \ (\text{merge } ?V \ ?V' \ (\sigma, \varrho)) \ \partial \text{state-measure } ?V \ ?\Gamma'$
by (*subst emeasure-dens-ctxt-measure-insert*[*OF dens* ϱ], *simp*, *intro nn-integral-cong*)
(*simp split: split-indicator*)
also have $\dots = \int^{+\sigma}. \text{insert-dens } V \ V' \ f \ \delta \ (\text{merge } ?V \ ?V' \ (\sigma, \varrho))$
 $\partial \text{state-measure } (\text{insert } 0 \ (\text{Suc } ' \ V)) \ ?\Gamma'$
by (*simp add: shift-var-set-def*)
also have $\dots = \int^{+\sigma}. \int^{+x}. \text{insert-dens } V \ V' \ f \ \delta \ (\text{merge } ?V \ ?V' \ (\sigma(0 := x),$
 $\varrho))$
 $\partial \text{stock-measure } t \ \partial \text{state-measure } (\text{Suc } ' \ V) \ ?\Gamma'$
unfolding *state-measure-def* **using** *M-merge*
by (*subst product-nn-integral-insert*) *auto*
also have $\dots = \int^{+\sigma}. \int^{+x}. \delta \ (\text{remove-var } (\text{merge } ?V \ ?V' \ (\sigma(0 := x), \varrho))) \ *$
 $f \ (\text{remove-var } (\text{merge } ?V \ ?V' \ (\sigma(0 := x), \varrho))) \ x$
 $\partial \text{stock-measure } t \ \partial \text{state-measure } (\text{Suc } ' \ V) \ ?\Gamma' \ (\mathbf{is} \ - = ?I)$
by (*intro nn-integral-cong*) (*auto simp: insert-dens-def merge-def shift-var-set-def*)
also have $\bigwedge \sigma \ x. \text{remove-var } (\text{merge } ?V \ ?V' \ (\sigma(0 := x), \varrho)) = \text{merge } V \ V' \ (\sigma$
 $\circ \text{Suc}, \varrho \circ \text{Suc})$
by (*intro ext*) (*auto simp: remove-var-def merge-def shift-var-set-def o-def*)
hence $?I = \int^{+\sigma}. \int^{+x}. \delta \ (\text{merge } V \ V' \ (\sigma \circ \text{Suc}, \varrho \circ \text{Suc})) \ * \ f \ (\text{merge } V \ V'$
 $(\sigma \circ \text{Suc}, \varrho \circ \text{Suc})) \ x$
 $\partial \text{stock-measure } t \ \partial \text{state-measure } (\text{Suc } ' \ V) \ ?\Gamma' \ \mathbf{by} \ \text{simp}$
also have $\dots = \int^{+\sigma}. \delta \ (\text{merge } V \ V' \ (\sigma \circ \text{Suc}, \varrho \circ \text{Suc})) \ *$
 $(\int^{+x}. f \ (\text{merge } V \ V' \ (\sigma \circ \text{Suc}, \varrho \circ \text{Suc})) \ x \ \partial \text{stock-measure } t)$
 $\partial \text{state-measure } (\text{Suc } ' \ V) \ ?\Gamma' \ (\mathbf{is} \ - = ?I)$
using ϱ *disjoint*
apply (*intro nn-integral-cong nn-integral-cmult*)
apply (*rule measurable-Pair-compose-split*[*OF Mf*], *rule measurable-const*)
apply (*auto intro!: Suc-state-measure*)
done
also {
fix σ **assume** $\sigma: \sigma \in \text{space } (\text{state-measure } (\text{Suc } ' \ V) \ ?\Gamma')$
let $? \sigma' = \text{merge } V \ V' \ (\sigma \circ \text{Suc}, \varrho \circ \text{Suc})$
let $?N = \text{density } (\text{stock-measure } t) \ (f \ ? \sigma')$
have $(\int^{+x}. f \ (\text{merge } V \ V' \ (\sigma \circ \text{Suc}, \varrho \circ \text{Suc})) \ x \ \partial \text{stock-measure } t) = \text{emeasure}$
 $?N \ (\text{space } ?N)$
using $\text{dens}'(\varrho) \ \text{Suc-state-measure}$ [*OF* σ]
by (*simp-all cong: nn-integral-cong' add: emeasure-density*)
also have $?N = F \ ? \sigma' \ \mathbf{by} \ (\text{subst } \text{dens}') \ (\text{simp-all add: Suc-state-measure } \sigma)$
also have *subprob-space* $(F \ ? \sigma') \ \mathbf{by} \ (\text{rule } \text{dens}') \ (\text{simp-all add: Suc-state-measure}$
 $\sigma)$
hence $\text{emeasure } (F \ ? \sigma') \ (\text{space } (F \ ? \sigma')) \leq 1 \ \mathbf{by} \ (\text{rule } \text{subprob-space.emeasure-space-le-1})$
finally have $(\int^{+x}. f \ (\text{merge } V \ V' \ (\sigma \circ \text{Suc}, \varrho \circ \text{Suc})) \ x \ \partial \text{stock-measure } t)$
 $\leq 1 .$
}
hence $?I \leq \int^{+\sigma}. \delta \ (\text{merge } V \ V' \ (\sigma \circ \text{Suc}, \varrho \circ \text{Suc})) \ * \ 1 \ \partial \text{state-measure } (\text{Suc}$
 $' \ V) \ ?\Gamma'$
by (*intro nn-integral-mono mult-left-mono*) (*simp-all add: Suc-state-measure*)
also have $\dots = \int^{+\sigma}. \delta \ (\text{merge } V \ V' \ (\sigma, \varrho \circ \text{Suc}))$

$\partial \text{distr} (\text{state-measure} (\text{Suc} \text{ ` } V) \text{ ?}\Gamma') (\text{state-measure} V \Gamma) (\lambda \sigma.$
 $\sigma \circ \text{Suc})$
(is - = nn-integral ?N -)
using ϱ **by** (*subst nn-integral-distr*) (*simp-all add: measurable-case-nat-Suc merge-Suc-aux*)
also have $\text{?N} = \text{state-measure} V \Gamma$ **by** (*rule distr-state-measure-Suc*)
also have $(\int^{+\sigma}. \delta (\text{merge } V V' (\sigma, \varrho \circ \text{Suc})) \partial \text{state-measure } V \Gamma) =$
 $(\int^{+\sigma}. 1 \partial \text{dens-ctxt-measure } \mathcal{Y} (\varrho \circ \text{Suc}))$ **(is - = nn-integral ?N -)**
by (*subst nn-integral-dens-ctxt-measure*) (*simp-all add: \varrho'*)
also have $\dots = (\int^{+\sigma}. \text{indicator} (\text{space } \text{?N}) \sigma \partial \text{?N})$
by (*intro nn-integral-cong*) (*simp split: split-indicator*)
also have $\dots = \text{emeasure } \text{?N} (\text{space } \text{?N})$ **by** *simp*
also have $\dots \leq 1$ **by** (*simp-all add: subprob-space.emeasure-space-le-1 subprob-space-dens \varrho'*)
finally show $\text{emeasure } \text{?M} (\text{space } \text{?M}) \leq 1$.
qed (*simp-all add: space-dens-ctxt-measure state-measure-def space-PiM PiE-eq-empty-iff*)
qed (*insert disjoint, auto simp: shift-var-set-def*)

lemma *dens-ctxt-measure-insert:*

assumes $\varrho: \varrho \in \text{space} (\text{state-measure } V' \Gamma)$
assumes $\text{meas-M}: M \in \text{measurable} (\text{state-measure} (V \cup V') \Gamma)$ (*subprob-algebra (stock-measure t)*)
assumes $\text{meas-f}[\text{measurable}]: \text{case-prod } f \in \text{borel-measurable} (\text{state-measure} (V \cup V') \Gamma \otimes_M \text{stock-measure } t)$
assumes $\text{has-dens}: \bigwedge \varrho. \varrho \in \text{space} (\text{state-measure} (V \cup V') \Gamma) \implies$
 $\text{has-subprob-density } (M \varrho) (\text{stock-measure } t) (f \varrho)$
shows $\text{do } \{ \sigma \leftarrow \text{dens-ctxt-measure} (V, V', \Gamma, \delta) \varrho;$
 $y \leftarrow M \sigma;$
 $\text{return } (\text{state-measure} (\text{shift-var-set} (V \cup V')) (\text{case-nat } t \Gamma)) (\text{case-nat } y \sigma) \}$ =
 $\text{dens-ctxt-measure} (\text{shift-var-set } V, \text{Suc} \text{ ` } V', \text{case-nat } t \Gamma, \text{insert-dens } V V' f \delta)$
 $(\text{case-nat } \text{undefined } \varrho)$
(is $\text{bind } \text{?N} (\lambda -. \text{bind } - (\lambda -. \text{return } \text{?R } -)) = \text{dens-ctxt-measure} (\text{?V}, \text{?V}', \text{?}\Gamma', \text{?}\delta')$
 $-)$

proof (*intro measure-eqI*)

let $\text{?lhs} = \text{?N} \gg (\lambda \sigma. M \sigma \gg (\lambda y. \text{return } \text{?R} (\text{case-nat } y \sigma)))$
let $\text{?rhs} = \text{dens-ctxt-measure} (\text{?V}, \text{?V}', \text{?}\Gamma', \text{?}\delta') (\text{case-nat } \text{undefined } \varrho)$

have $\text{meas-f}': \bigwedge M g h. g \in \text{measurable } M (\text{state-measure} (V \cup V') \Gamma) \implies$
 $h \in \text{measurable } M (\text{stock-measure } t) \implies$
 $(\lambda x. f (g x) (h x)) \in \text{borel-measurable } M$ **by** *measurable*

have $t: t = \text{?}\Gamma' 0$ **by** *simp*

have *nonempty*: $\text{space } \text{?N} \neq \{\}$

by (*auto simp: dens-ctxt-measure-def state-measure'-def state-measure-def space-PiM PiE-eq-empty-iff*)

have $\text{meas-N-eq}: \text{measurable } \text{?N} = \text{measurable} (\text{state-measure} (V \cup V') \Gamma)$

```

    by (intro ext measurable-cong-sets) (auto simp: dens-ctxt-measure-def state-measure'-def)
  have meas-M':  $M \in \text{measurable } ?N$  (subprob-algebra (stock-measure t))
    by (subst meas-N-eq) (rule meas-M)
  have meas-N':  $\bigwedge R. \text{measurable } (?N \otimes_M R) = \text{measurable } (\text{state-measure } (V \cup V'))$ 
 $\Gamma \otimes_M R$ 
    by (intro ext measurable-cong-sets[OF - refl] sets-pair-measure-cong)
      (simp-all add: dens-ctxt-measure-def state-measure'-def)
  have meas-M-eq:  $\bigwedge \varrho. \varrho \in \text{space } ?N \implies \text{measurable } (M \varrho) = \text{measurable } (\text{stock-measure } t)$ 
    by (intro ext measurable-cong-sets sets-kernel[OF meas-M']) simp-all
  have meas-rhs:  $\bigwedge M. \text{measurable } M \text{ ?rhs} = \text{measurable } M \text{ ?R}$ 
    by (intro ext measurable-cong-sets) (simp-all add: dens-ctxt-measure-def state-measure'-def)
  have subprob-algebra-rhs:  $\text{subprob-algebra } \text{?rhs} = \text{subprob-algebra } (\text{state-measure } (\text{shift-var-set } (V \cup V'))) \text{ ?}\Gamma'$ 
    (shift-var-set (V ∪ V')) ?Γ')
  unfolding dens-ctxt-measure-def state-measure'-def by (intro subprob-algebra-cong)
  auto
  have nonempty':  $\bigwedge \varrho. \varrho \in \text{space } ?N \implies \text{space } (M \varrho) \neq \{\}$ 
    by (rule subprob-space.subprob-not-empty)
      (auto dest: has-subprob-densityD has-dens simp: space-dens-ctxt-measure)
  have merge-in-space:  $\bigwedge x. x \in \text{space } (\text{state-measure } V \Gamma) \implies$ 
 $\text{merge } V \text{ } V' (x, \varrho) \in \text{space } (\text{dens-ctxt-measure } \mathcal{Y} \varrho)$ 
    by (simp add: space-dens-ctxt-measure merge-in-state-measure  $\varrho$ )

  have sets ?lhs = sets (state-measure (shift-var-set (V ∪ V'))) ?Γ')
    using nonempty' by (subst sets-bind, subst sets-bind) auto
  thus sets-eq: sets ?lhs = sets ?rhs
    unfolding dens-ctxt-measure-def state-measure'-def by simp

  have meas-merge[measurable]:
    ( $\lambda \sigma. \text{merge } V \text{ } V' (\sigma, \varrho) \in \text{measurable } (\text{state-measure } V \Gamma) (\text{state-measure } (V \cup V') \Gamma)$ )
    using  $\varrho$  unfolding state-measure-def by - measurable

  fix X assume X ∈ sets ?lhs
  hence X:  $X \in \text{sets } \text{?rhs}$  by (simp add: sets-eq)
  hence emeasure ?lhs  $X = \int^+ \sigma. \text{emeasure } (M \sigma \gg (\lambda y. \text{return } ?R (\text{case-nat } y \sigma))) X \partial ?N$ 
    by (intro emeasure-bind measurable-bind[OF meas-M'])
      (simp, rule measurable-compose[OF - return-measurable],
      simp-all add: dens-ctxt-measure-def state-measure'-def)
  also from X have ... =
 $\int^+ x. \delta (\text{merge } V \text{ } V' (x, \varrho)) * \text{emeasure } (M (\text{merge } V \text{ } V' (x, \varrho)) \gg (\lambda y. \text{return } ?R (\text{case-nat } y (\text{merge } V \text{ } V' (x, \varrho)))))) X \partial \text{state-measure } V$ 
 $\Gamma$ 
  apply (subst nn-integral-dens-ctxt-measure[OF  $\varrho$ ])
  apply (rule measurable-emeasure-kernel[OF measurable-bind[OF meas-M]])
  apply (rule measurable-compose[OF - return-measurable], simp)
  apply (simp-all add: dens-ctxt-measure-def state-measure'-def)
  done

```

also from X have ... = $\int^+ x. \delta (\text{merge } V \ V' (x, \varrho)) * (\int^+ y. \text{indicator } X (\text{case-nat } y (\text{merge } V \ V' (x, \varrho))) \partial M (\text{merge } V \ V' (x, \varrho))) \partial \text{state-measure } V \ \Gamma$ (**is - =** ?I)

apply (*intro nn-integral-cong*)
apply (*subst emeasure-bind, rule nonempty', simp add: merge-in-space*)
apply (*rule measurable-compose[OF - return-measurable], simp add: merge-in-space meas-M-eq*)
apply (*simp-all add: dens-ctxt-measure-def state-measure'-def*)
done

also have $\bigwedge x. x \in \text{space } (\text{state-measure } V \ \Gamma) \implies M (\text{merge } V \ V' (x, \varrho)) = \text{density } (\text{stock-measure } t) (f (\text{merge } V \ V' (x, \varrho)))$
by (*intro has-subprob-densityD[OF has-dens]*) (*simp add: merge-in-state-measure* ϱ)

hence ?I = $\int^+ x. \delta (\text{merge } V \ V' (x, \varrho)) * (\int^+ y. \text{indicator } X (\text{case-nat } y (\text{merge } V \ V' (x, \varrho))) \partial \text{density } (\text{stock-measure } t) (f (\text{merge } V \ V' (x, \varrho)))) \partial \text{state-measure } V \ \Gamma$
by (*intro nn-integral-cong simp*)

also have ... = $\int^+ x. \delta (\text{merge } V \ V' (x, \varrho)) * (\int^+ y. f (\text{merge } V \ V' (x, \varrho)) y * \text{indicator } X (\text{case-nat } y (\text{merge } V \ V' (x, \varrho)))) \partial \text{stock-measure } t \ \partial \text{state-measure } V \ \Gamma$ (**is - =** ?I) **using** X
by (*intro nn-integral-cong, subst nn-integral-density, simp*)
(auto simp: mult.assoc dens-ctxt-measure-def state-measure'-def intro!: merge-in-state-measure ϱ AE-I'[of {}] has-subprob-densityD[OF has-dens])

also have A: *case-nat undefined $\varrho \circ \text{Suc} = \varrho$* **by** (*intro ext*) *simp*
have B: $\bigwedge x y. x \in \text{space } (\text{state-measure } V \ \Gamma) \implies y \in \text{space } (\text{stock-measure } t)$
 \implies

$(\text{case-nat } y (\text{merge } V \ V' (x, \varrho))) = (\text{merge } (\text{shift-var-set } V) (\text{Suc } 'V') (\text{case-nat } y \ x, \text{case-nat } \text{undefined } \varrho))$
by (*intro ext*) (*auto simp add: merge-def shift-var-set-def split: nat.split*)

have C: $\bigwedge x. x \in \text{space } (\text{state-measure } V \ \Gamma) \implies (\int^+ y. f (\text{merge } V \ V' (x, \varrho)) y * \text{indicator } X (\text{case-nat } y (\text{merge } V \ V' (x, \varrho)))) \partial \text{stock-measure } t = \int^+ y. f (\text{merge } V \ V' (x, \varrho)) y * \text{indicator } X (\text{merge } (\text{shift-var-set } V) (\text{Suc } 'V') (\text{case-nat } y \ x, \text{case-nat } \text{undefined } \varrho)) \partial \text{stock-measure } t$
by (*intro nn-integral-cong*) (*simp add: B*)

have ?I = *emeasure ?rhs X* **using** X
apply (*subst emeasure-dens-ctxt-measure-insert'[where F = M]*)
apply (*insert has-dens, simp add: has-parametrized-subprob-density-def*)
apply (*rule measurable-space[OF measurable-case-nat-undefined ϱ], simp*)
apply (*intro nn-integral-cong, simp add: A C*)
done

finally show *emeasure ?lhs X = emeasure ?rhs X* .
qed

lemma *density-context-if-dens*:

assumes *has-parametrized-subprob-density* (*state-measure* ($V \cup V'$) Γ) M
(*count-space* (*range BoolVal*)) f

shows *density-context* $V V' \Gamma$ (*if-dens* $\delta f b$)

unfolding *density-context-def*

proof (*intro allI conjI impI subprob-spaceI*)

note $D = \text{has-parametrized-subprob-densityD}[OF \text{ assms}]$

from $D(\beta)$ **show** $M: \text{if-dens } \delta f b \in \text{borel-measurable} (\text{state-measure} (V \cup V') \Gamma)$

Γ)

by (*intro measurable-if-dens*) *simp-all*

fix ϱ **assume** $\varrho: \varrho \in \text{space} (\text{state-measure } V' \Gamma)$

hence [*measurable*]: $(\lambda\sigma. \text{merge } V V' (\sigma, \varrho)) \in$
measurable (*state-measure* $V \Gamma$) (*state-measure* ($V \cup V'$) Γ)

unfolding *state-measure-def* **by** *simp*

{

fix σ **assume** $\sigma \in \text{space} (\text{state-measure } V \Gamma)$

with ϱ **have** $\sigma\varrho: \text{merge } V V' (\sigma, \varrho) \in \text{space} (\text{state-measure} (V \cup V') \Gamma)$

by (*intro merge-in-state-measure*)

with *assms* **have** *has-subprob-density* ($M (\text{merge } V V' (\sigma, \varrho))$) (*count-space* (*range BoolVal*))
($f (\text{merge } V V' (\sigma, \varrho))$)

unfolding *has-parametrized-subprob-density-def* **by** *auto*

with $\sigma\varrho$ **have** $f (\text{merge } V V' (\sigma, \varrho)) (\text{BoolVal } b) \leq 1 \ \delta (\text{merge } V V' (\sigma, \varrho)) \geq$
 0

by (*auto intro: subprob-count-space-density-le-1*)

} **note** *dens-props = this*

from ϱ **interpret** *subprob-space dens-ctxt-measure* $\mathcal{Y} \varrho$ **by** (*rule subprob-space-dens*)

let $?M = \text{dens-ctxt-measure} (V, V', \Gamma, \text{if-dens } \delta f b) \varrho$

have *emeasure* $?M (\text{space } ?M) =$
 $\int^{+x}. \text{if-dens } \delta f b (\text{merge } V V' (x, \varrho)) \partial \text{state-measure } V \Gamma$

using $M \varrho$ **unfolding** *dens-ctxt-measure-def state-measure'-def*

by (*simp only: prod.case space-density*)
(*auto simp: nn-integral-distr emeasure-density cong: nn-integral-cong'*)

also from ϱ **have** $\dots \leq \int^{+x}. \delta (\text{merge } V V' (x, \varrho)) * 1 \partial \text{state-measure } V \Gamma$

unfolding *if-dens-def* **using** *dens-props*

by (*intro nn-integral-mono mult-left-mono*) *simp-all*

also from ϱ **have** $\dots = \text{branch-prob } \mathcal{Y} \varrho$ **by** (*simp add: branch-prob-altdef*)

also have $\dots = \text{emeasure} (\text{dens-ctxt-measure } \mathcal{Y} \varrho) (\text{space} (\text{dens-ctxt-measure } \mathcal{Y} \varrho))$

by (*simp add: branch-prob-def*)

also have $\dots \leq 1$ **by** (*rule emeasure-space-le-1*)

finally show *emeasure* $?M (\text{space } ?M) \leq 1$.

qed (*insert disjoint, auto*)

lemma *density-context-if-dens-det*:

assumes $e: \Gamma \vdash e : \text{BOOL randomfree } e \text{ free-vars } e \subseteq V \cup V'$

shows *density-context* $V V' \Gamma$ (*if-dens-det* $\delta e b$)
unfolding *density-context-def*
proof (*intro allI conjI impI subprob-spaceI*)
from *assms* **show** M : *if-dens-det* $\delta e b \in \text{borel-measurable}$ (*state-measure* $(V \cup V') \Gamma$)
by (*intro measurable-if-dens-det*) *simp-all*

fix ϱ **assume** $\varrho \in \text{space}$ (*state-measure* $V' \Gamma$)
hence [*measurable*]: $(\lambda\sigma. \text{merge } V V' (\sigma, \varrho)) \in \text{measurable}$ (*state-measure* $V \Gamma$) (*state-measure* $(V \cup V') \Gamma$)
unfolding *state-measure-def* **by** *simp*

from ϱ **interpret** *subprob-space dens-ctxt-measure* $\mathcal{Y} \varrho$ **by** (*rule subprob-space-dens*)
let $?M = \text{dens-ctxt-measure}$ $(V, V', \Gamma, \text{if-dens-det } \delta e b) \varrho$
have *emeasure* $?M$ (*space* $?M$) =
 $\int^+ x. \text{if-dens-det } \delta e b (\text{merge } V V' (x, \varrho)) \partial \text{state-measure } V \Gamma$
using $M \varrho$ **unfolding** *dens-ctxt-measure-def state-measure'-def*
by (*simp only: prod.case space-density*)
(*auto simp: nn-integral-distr emeasure-density cong: nn-integral-cong'*)
also from ϱ **have** $\dots \leq \int^+ x. \delta (\text{merge } V V' (x, \varrho)) * 1 \partial \text{state-measure } V \Gamma$
unfolding *if-dens-det-def*
by (*intro nn-integral-mono mult-left-mono*) (*simp-all add: merge-in-state-measure*)
also from ϱ **have** $\dots = \text{branch-prob } \mathcal{Y} \varrho$ **by** (*simp add: branch-prob-altdef*)
also have $\dots = \text{emeasure}$ (*dens-ctxt-measure* $\mathcal{Y} \varrho$) (*space* (*dens-ctxt-measure* $\mathcal{Y} \varrho$))
by (*simp add: branch-prob-def*)
also have $\dots \leq 1$ **by** (*rule emeasure-space-le-1*)
finally show *emeasure* $?M$ (*space* $?M$) ≤ 1 .
qed (*insert disjoint assms, auto intro: measurable-if-dens-det*)

lemma *density-context-empty[simp]*: *density-context* $\{\} (V \cup V') \Gamma (\lambda-. 1)$
unfolding *density-context-def*
proof (*intro allI conjI impI subprob-spaceI*)
fix ϱ **assume** $\varrho \in \text{space}$ (*state-measure* $(V \cup V') \Gamma$)
let $?M = \text{dens-ctxt-measure}$ $(\{\}, V \cup V', \Gamma, \lambda-. 1) \varrho$
from ϱ **have** $\bigwedge \sigma. \text{merge } \{\} (V \cup V') (\sigma, \varrho) = \varrho$
by (*intro ext*) (*auto simp: merge-def state-measure-def space-PiM*)
with ϱ **show** *emeasure* $?M$ (*space* $?M$) ≤ 1
unfolding *dens-ctxt-measure-def state-measure'-def*
by (*auto simp: emeasure-density emeasure-distr state-measure-def PiM-empty*)
qed *auto*

lemma *dens-ctxt-measure-bind-const*:
assumes $\varrho \in \text{space}$ (*state-measure* $V' \Gamma$) *subprob-space* N
shows *dens-ctxt-measure* $\mathcal{Y} \varrho \gg (\lambda-. N) = \text{density } N (\lambda-. \text{branch-prob } \mathcal{Y} \varrho)$ (*is* $?M1 = ?M2$)
proof (*rule measure-eqI*)
have [*simp*]: *sets* $?M1 = \text{sets } N$ **by** (*auto simp: space-subprob-algebra assms*)

thus sets $?M1 = \text{sets } ?M2$ **by** simp
fix X **assume** $X: X \in \text{sets } ?M1$
with *assms* **have** $\text{emeasure } ?M1 X = \text{emeasure } N X * \text{branch-prob } \mathcal{Y} \varrho$
unfolding *branch-prob-def* **by** (*subst emeasure-bind-const'*) (*auto simp: sub-prob-space-dens*)
also from X **have** $\text{emeasure } N X = \int^+ x. \text{indicator } X x \partial N$ **by** simp
also from X **have** $\dots * \text{branch-prob } \mathcal{Y} \varrho = \int^+ x. \text{branch-prob } \mathcal{Y} \varrho * \text{indicator } X x \partial N$
by (*subst nn-integral-cmult*) (*auto simp: branch-prob-def field-simps*)
also from X **have** $\dots = \text{emeasure } ?M2 X$ **by** (*simp add: emeasure-density*)
finally show $\text{emeasure } ?M1 X = \text{emeasure } ?M2 X$.
qed

lemma *nn-integral-dens-ctxt-measure-restrict*:

assumes $\varrho \in \text{space } (\text{state-measure } V' \Gamma)$ $f \varrho \geq 0$
assumes $f \in \text{borel-measurable } (\text{state-measure } V' \Gamma)$
shows $(\int^+ x. f (\text{restrict } x V') \partial \text{dens-ctxt-measure } \mathcal{Y} \varrho) = \text{branch-prob } \mathcal{Y} \varrho * f \varrho$
proof –
have $(\int^+ x. f (\text{restrict } x V') \partial \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho) =$
 $\int^+ x. \delta (\text{merge } V V' (x, \varrho)) * f (\text{restrict } (\text{merge } V V' (x, \varrho)) V')$
 $\partial \text{state-measure } V \Gamma$
(is - = ?I)
by (*subst nn-integral-dens-ctxt-measure, simp add: assms,*
rule measurable-compose[OF measurable-restrict], unfold state-measure-def,
rule measurable-component-singleton, insert assms, simp-all add: state-measure-def)
also from *assms(1)* **and** *disjoint*
have $\bigwedge x. x \in \text{space } (\text{state-measure } V \Gamma) \implies \text{restrict } (\text{merge } V V' (x, \varrho)) V' = \varrho$
by (*intro ext*) (*auto simp: restrict-def merge-def state-measure-def space-PiM dest: PiE-mem*)
hence $?I = \int^+ x. \delta (\text{merge } V V' (x, \varrho)) * f \varrho \partial \text{state-measure } V \Gamma$
by (*intro nn-integral-cong*) *simp*
also have $\dots = (\int^+ x. f \varrho \partial \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho)$
by (*subst nn-integral-dens-ctxt-measure*) (*simp-all add: assms*)
also have $\dots = f \varrho * \text{branch-prob } \mathcal{Y} \varrho$
by (*subst nn-integral-const*)
(simp-all add: assms branch-prob-def)
finally show *?thesis* **by** (*simp add: field-simps*)
qed

lemma *expr-sem-op-eq-distr*:

assumes $\Gamma \vdash \text{oper } \$\$ e : t'$ *free-vars* $e \subseteq V \cup V'$ $\varrho \in \text{space } (\text{state-measure } V' \Gamma)$
defines $M \equiv \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho$
shows $M \ggg (\lambda \sigma. \text{expr-sem } \sigma (\text{oper } \$\$ e)) =$
 $\text{distr } (M \ggg (\lambda \sigma. \text{expr-sem } \sigma e)) (\text{stock-measure } t') (\text{op-sem oper})$
proof –
from *assms(1)* **obtain** t **where** $t1: \Gamma \vdash e : t$ **and** $t2: \text{op-type oper } t = \text{Some } t'$

by *auto*
let $?N = \text{stock-measure } t$ **and** $?R = \text{subprob-algebra } (\text{stock-measure } t')$

{
fix x **assume** $x \in \text{space } (\text{stock-measure } t)$
with $t1$ *assms*(2,3) **have** $\text{val-type } x = t$
by (*auto simp: state-measure-def space-PiM dest: PiE-mem*)
hence $\text{return-val } (\text{op-sem oper } x) = \text{return } (\text{stock-measure } t') (\text{op-sem oper } x)$
unfolding *return-val-def* **by** (*subst op-sem-val-type*) (*simp-all add: t2*)
} note $\text{return-op-sem} = \text{this}$

from *assms* **and** $t1$ **have** $M\text{-e}: (\lambda\sigma. \text{expr-sem } \sigma e) \in \text{measurable } M (\text{subprob-algebra } (\text{stock-measure } t))$
by (*simp add: M-def measurable-dens-ctxt-measure-eq measurable-expr-sem*)
from *return-op-sem*
have $M\text{-cong}: (\lambda x. \text{return-val } (\text{op-sem oper } x)) \in \text{measurable } ?N ?R \longleftrightarrow$
 $(\lambda x. \text{return } (\text{stock-measure } t') (\text{op-sem oper } x)) \in \text{measurable } ?N$
 $?R$
by (*intro measurable-cong simp*)
have $M\text{-ret}: (\lambda x. \text{return-val } (\text{op-sem oper } x)) \in \text{measurable } (\text{stock-measure } t) ?R$
by (*subst M-cong, intro measurable-compose[OF measurable-op-sem[OF t2]]*)
return-measurable

from $M\text{-e}$ **have** [*simp*]: $\text{sets } (M \gg (\lambda\sigma. \text{expr-sem } \sigma e)) = \text{sets } (\text{stock-measure } t)$
by (*intro sets-bind*) (*auto simp: M-def space-subprob-algebra dest!: measurable-space*)
from *measurable-cong-sets[OF this refl]*
have $M\text{-op}: \text{op-sem oper} \in \text{measurable } (M \gg (\lambda\sigma. \text{expr-sem } \sigma e)) (\text{stock-measure } t')$
by (*auto intro!: measurable-op-sem t2*)
have [*simp*]: $\text{space } (M \gg (\lambda\sigma. \text{expr-sem } \sigma e)) = \text{space } (\text{stock-measure } t)$
by (*rule sets-eq-imp-space-eq simp*)

from $M\text{-e}$ **and** $M\text{-ret}$ **have** $M \gg (\lambda\sigma. \text{expr-sem } \sigma (\text{oper } \$\$ e)) =$
 $(M \gg (\lambda\sigma. \text{expr-sem } \sigma e)) \gg (\lambda x. \text{return-val } (\text{op-sem } \text{oper } x))$
unfolding $M\text{-def}$ **by** (*subst expr-sem.simps, intro bind-assoc[symmetric]*) *simp-all*
also have $\dots = (M \gg (\lambda\sigma. \text{expr-sem } \sigma e)) \gg (\lambda x. \text{return } (\text{stock-measure } t') (\text{op-sem oper } x))$
by (*intro bind-cong refl*) (*simp add: return-op-sem*)
also have $\dots = \text{distr } (M \gg (\lambda\sigma. \text{expr-sem } \sigma e)) (\text{stock-measure } t') (\text{op-sem oper})$
by (*subst bind-return-distr[symmetric]*) (*simp-all add: o-def M-op*)
finally show *?thesis* .

qed

end

lemma *density-context-equiv*:
assumes $\bigwedge \sigma. \sigma \in \text{space } (\text{state-measure } (V \cup V') \Gamma) \implies \delta \sigma = \delta' \sigma$
assumes [*simp, measurable*]: $\delta' \in \text{borel-measurable } (\text{state-measure } (V \cup V') \Gamma)$
assumes *density-context* $V V' \Gamma \delta$
shows *density-context* $V V' \Gamma \delta'$
proof (*unfold density-context-def, intro conjI allI impI subprob-spaceI*)
interpret *density-context* $V V' \Gamma \delta$ **by** *fact*
fix ϱ **assume** $\varrho: \varrho \in \text{space } (\text{state-measure } V' \Gamma)$
let $?M = \text{dens-ctxt-measure } (V, V', \Gamma, \delta') \varrho$
let $?N = \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho$
from ϱ **have** $\text{emeasure } ?M (\text{space } ?M) = \int^{+x}. \delta' (\text{merge } V V' (x, \varrho)) \partial \text{state-measure } V \Gamma$
unfolding *dens-ctxt-measure-def state-measure'-def*
apply (*simp only: prod.case, subst space-density*)
apply (*simp add: emeasure-density cong: nn-integral-cong'*)
apply (*subst nn-integral-distr, simp add: state-measure-def, simp-all*)
done
also from ϱ **have** $\dots = \int^{+x}. \delta (\text{merge } V V' (x, \varrho)) \partial \text{state-measure } V \Gamma$
by (*intro nn-integral-cong, subst assms(1)*) (*simp-all add: merge-in-state-measure*)
also from ϱ **have** $\dots = \text{branch-prob } (V, V', \Gamma, \delta) \varrho$ **by** (*simp add: branch-prob-altdef*)
also have $\dots = \text{emeasure } ?N (\text{space } ?N)$ **by** (*simp add: branch-prob-def*)
also from ϱ **have** $\dots \leq 1$ **by** (*intro subprob-space.emeasure-space-le-1 subprob-space-dens*)
finally show $\text{emeasure } ?M (\text{space } ?M) \leq 1$.
qed (*insert assms, auto simp: density-context-def*)

end

7 Abstract PDF Compiler

theory *PDF-Compiler-Pred*
imports *PDF-Semantics PDF-Density-Contexts PDF-Transformations Density-Predicates*
begin

7.1 Density compiler predicate

Predicate version of the probability density compiler that compiles an expression to a probability density function of its distribution. The density is a HOL function of type $\text{val} \Rightarrow \text{ennreal}$.

inductive *expr-has-density* :: $\text{dens-ctxt} \Rightarrow \text{expr} \Rightarrow (\text{state} \Rightarrow \text{val} \Rightarrow \text{ennreal}) \Rightarrow \text{bool}$

$\langle \langle (1 \vdash_d / (- \Rightarrow / -)) \rangle [50, 0, 50] 50 \rangle$ **where**
hd-AE: $\llbracket (V, V', \Gamma, \delta) \vdash_d e \Rightarrow f; \Gamma \vdash e : t; \bigwedge \varrho. \varrho \in \text{space } (\text{state-measure } V' \Gamma) \implies \text{AE } x \text{ in stock-measure } t. f \varrho x = f' \varrho x; \text{case-prod } f' \in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t) \rrbracket$
 $\implies (V, V', \Gamma, \delta) \vdash_d e \Rightarrow f'$

$|$ *hd-dens-ctxt-cong*:
 $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow (\bigwedge \sigma. \sigma \in \text{space } (\text{state-measure } (V \cup V') \Gamma) \Rightarrow \delta \sigma = \delta' \sigma)$
 $\Longrightarrow (V, V', \Gamma, \delta') \vdash_d e \Rightarrow f$

$|$ *hd-val*: *countable-type* (*val-type* v) \Longrightarrow
 $(V, V', \Gamma, \delta) \vdash_d \text{Val } v \Rightarrow (\lambda \varrho x. \text{branch-prob } (V, V', \Gamma, \delta) \varrho * \text{indicator } \{v\} x)$

$|$ *hd-var*: $x \in V \Longrightarrow (V, V', \Gamma, \delta) \vdash_d \text{Var } x \Rightarrow \text{marg-dens } (V, V', \Gamma, \delta) x$

$|$ *hd-let*: $\llbracket (\{\}, V \cup V', \Gamma, \lambda-. 1) \vdash_d e1 \Rightarrow f; (\text{shift-var-set } V, \text{Suc } V', \text{the}(\text{expr-type } \Gamma \ e1) \cdot \Gamma, \text{insert-dens } V \ V' \ f \ \delta) \vdash_d e2 \Rightarrow g \rrbracket$
 $\Longrightarrow (V, V', \Gamma, \delta) \vdash_d \text{LetVar } e1 \ e2 \Rightarrow (\lambda \varrho. g \ (\text{case-nat } \text{undefined } \varrho))$

$|$ *hd-rand*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow (V, V', \Gamma, \delta) \vdash_d \text{Random } \text{dst } e \Rightarrow \text{apply-dist-to-dens } \text{dst } f$

$|$ *hd-rand-det*: *randomfree* $e \Longrightarrow \text{free-vars } e \subseteq V' \Longrightarrow$
 $(V, V', \Gamma, \delta) \vdash_d \text{Random } \text{dst } e \Rightarrow$
 $(\lambda \varrho x. \text{branch-prob } (V, V', \Gamma, \delta) \varrho * \text{dist-dens } \text{dst } (\text{expr-sem-rf } \varrho \ e))$

$x)$

$|$ *hd-fail*: $(V, V', \Gamma, \delta) \vdash_d \text{Fail } t \Rightarrow (\lambda-. 0)$

$|$ *hd-pair*: $x \in V \Longrightarrow y \in V \Longrightarrow x \neq y \Longrightarrow (V, V', \Gamma, \delta) \vdash_d \langle \text{Var } x, \text{Var } y \rangle \Rightarrow \text{marg-dens2 } (V, V', \Gamma, \delta) x \ y$

$|$ *hd-if*: $\llbracket (\{\}, V \cup V', \Gamma, \lambda-. 1) \vdash_d b \Rightarrow f; (V, V', \Gamma, \text{if-dens } \delta \ f \ \text{True}) \vdash_d e1 \Rightarrow g1; (V, V', \Gamma, \text{if-dens } \delta \ f \ \text{False}) \vdash_d e2 \Rightarrow g2 \rrbracket$
 $\Longrightarrow (V, V', \Gamma, \delta) \vdash_d \text{IF } b \ \text{THEN } e1 \ \text{ELSE } e2 \Rightarrow (\lambda \varrho x. g1 \ \varrho \ x + g2 \ \varrho \ x)$

$|$ *hd-if-det*: $\llbracket \text{randomfree } b; (V, V', \Gamma, \text{if-dens-det } \delta \ b \ \text{True}) \vdash_d e1 \Rightarrow g1; (V, V', \Gamma, \text{if-dens-det } \delta \ b \ \text{False}) \vdash_d e2 \Rightarrow g2 \rrbracket$
 $\Longrightarrow (V, V', \Gamma, \delta) \vdash_d \text{IF } b \ \text{THEN } e1 \ \text{ELSE } e2 \Rightarrow (\lambda \varrho x. g1 \ \varrho \ x + g2 \ \varrho \ x)$

$|$ *hd-fst*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow$
 $(V, V', \Gamma, \delta) \vdash_d \text{Fst } \$\$ \ e \Rightarrow$
 $(\lambda \varrho x. \int^+ y. f \ \varrho \ \langle |x, y| \rangle \ \partial \text{stock-measure } (\text{the } (\text{expr-type } \Gamma \ (\text{Snd } \$\$ \ e))))$

$|$ *hd-snd*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow$
 $(V, V', \Gamma, \delta) \vdash_d \text{Snd } \$\$ \ e \Rightarrow$
 $(\lambda \varrho y. \int^+ x. f \ \varrho \ \langle |x, y| \rangle \ \partial \text{stock-measure } (\text{the } (\text{expr-type } \Gamma \ (\text{Fst } \$\$ \ e))))$

$|$ *hd-op-discr*: *countable-type* (*the* (*expr-type* Γ (*oper* $\$ \$ \ e$))) $\Longrightarrow (V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow$
 $(V, V', \Gamma, \delta) \vdash_d \text{oper } \$ \$ \ e \Rightarrow (\lambda \varrho y. \int^+ x. (\text{if } \text{op-sem } \text{oper } x = y \ \text{then } 1 \ \text{else } 0) * f \ \varrho \ x \ \partial \text{stock-measure } (\text{the } (\text{expr-type } \Gamma \ e)))$

$|$ *hd-neg*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow$
 $(V, V', \Gamma, \delta) \vdash_d \text{Minus } \$ \$ \ e \Rightarrow (\lambda \sigma x. f \ \sigma \ (\text{op-sem } \text{Minus } x))$

$|$ *hd-addc*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow \text{randomfree } e' \Longrightarrow \text{free-vars } e' \subseteq V' \Longrightarrow$
 $(V, V', \Gamma, \delta) \vdash_d \text{Add } \$ \$ \ \langle e, e' \rangle \Rightarrow$
 $(\lambda \varrho x. f \ \varrho \ (\text{op-sem } \text{Add } \langle |x, \text{expr-sem-rf } \varrho \ (\text{Minus } \$ \$ \ e') | \rangle))$

$|$ *hd-multc*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow \text{val-type } c = \text{REAL} \Longrightarrow c \neq \text{RealVal } 0 \Longrightarrow$
 $(V, V', \Gamma, \delta) \vdash_d \text{Mult } \$ \$ \ \langle e, \text{Val } c \rangle \Rightarrow$
 $(\lambda \varrho x. f \ \varrho \ (\text{op-sem } \text{Mult } \langle |x, \text{op-sem } \text{Inverse } c | \rangle) *$

$$\text{inverse (abs (extract-real c))}$$
| *hd-exp*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow$
 $(V, V', \Gamma, \delta) \vdash_d \text{Exp } \$\$ e \Rightarrow$
 $(\lambda \sigma x. \text{if extract-real } x > 0 \text{ then}$
 $f \sigma (\text{lift-RealVal safe-ln } x) * \text{inverse (extract-real } x) \text{ else } 0)$
| *hd-inv*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow$
 $(V, V', \Gamma, \delta) \vdash_d \text{Inverse } \$\$ e \Rightarrow (\lambda \sigma x. f \sigma (\text{op-sem Inverse } x) * \text{inverse (extract-real } x) \wedge 2)$
| *hd-add*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow$
 $(V, V', \Gamma, \delta) \vdash_d \text{Add } \$\$ e \Rightarrow (\lambda \sigma z. \int^+ x. f \sigma <|x, \text{op-sem Add } <|z,$
op-sem Minus $x|>|>$
 $\partial \text{stock-measure (val-type } z))$

lemmas *expr-has-density-intros* =
hd-val hd-var hd-let hd-rand hd-rand-det hd-fail hd-pair hd-if hd-if-det
hd-fst hd-snd hd-op-discr hd-neg hd-addc hd-multc hd-exp hd-inv hd-add

7.2 Auxiliary lemmas

lemma *has-subprob-density-distr-Fst*:

fixes *t1 t2 f*
defines $N \equiv \text{stock-measure (PRODUCT } t1 \ t2)$
defines $N' \equiv \text{stock-measure } t1$
defines $\text{fst}' \equiv \text{op-sem } Fst$
defines $f' \equiv \lambda x. \int^+ y. f <|x, y|> \partial \text{stock-measure } t2$
assumes *dens*: *has-subprob-density* $M \ N \ f$
shows *has-subprob-density* $(\text{distr } M \ N' \ \text{fst}')$ $N' \ f'$
proof (*intro has-subprob-densityI measure-eqI*)
from *dens* **interpret** *subprob-space* M **by** (*rule has-subprob-densityD*)
from *dens* **have** *M-M*: *measurable* $M = \text{measurable } N$
by (*intro ext measurable-cong-sets*) (*auto dest: has-subprob-densityD*)
hence *meas-fst*: $\text{fst}' \in \text{measurable } M \ N'$ **unfolding** *fst'-def*
by (*subst op-sem.simps*) (*simp add: N'-def N-def M-M*)
thus *subprob-space* $(\text{distr } M \ N' \ \text{fst}')$
by (*rule subprob-space-distr*) (*simp-all add: N'-def*)

interpret *sigma-finite-measure* *stock-measure* $t2$ **by** *simp*
have *f[measurable]*: $f \in \text{borel-measurable (stock-measure (PRODUCT } t1 \ t2))$
using *dens* **by** (*auto simp: has-subprob-density-def has-density-def N-def*)
then show *meas-f'*: $f' \in \text{borel-measurable } N'$ **unfolding** *f'-def N'-def*
by *measurable*

let $?M1 = \text{distr } M \ N' \ \text{fst}'$ **and** $?M2 = \text{density } N' \ f'$
show *sets* $?M1 = \text{sets } ?M2$ **by** *simp*
fix X **assume** $X \in \text{sets } ?M1$
hence X : $X \in \text{sets } N' \ X \in \text{sets } N'$ **by** (*simp-all add: N'-def*)
then have *[measurable]*: $X \in \text{sets (stock-measure } t1)$
by (*simp add: N'-def*)

from *meas-fst* **and** $X(1)$ **have** $\text{emeasure } ?M1 \ X = \text{emeasure } M \ (fst' - ' X \cap \text{space } M)$
by (*rule emeasure-distr*)
also from *dens* **have** $M: M = \text{density } N \ f$ **by** (*rule has-subprob-densityD*)
from *this* **and** *meas-fst* **have** *meas-fst'*: $fst' \in \text{measurable } N \ N'$ **by** *simp*
with *dens* **and** X **have** $\text{emeasure } M \ (fst' - ' X \cap \text{space } M) =$
 $\int^{+x}. f \ x * \text{indicator } (fst' - ' X \cap \text{space } N) \ x \ \partial N$
by (*subst (1 2) M, subst space-density, subst emeasure-density*)
(*erule has-subprob-densityD, erule measurable-sets, simp, simp*)
also have $N = \text{distr } (N' \otimes_M \text{stock-measure } t2) \ N$ (*case-prod PairVal*) (**is - =**
 $?N$)
unfolding $N\text{-def } N'\text{-def } \text{stock-measure.simps}$ **by** (*rule embed-measure-eq-distr*)
(*simp add: inj-PairVal*)
hence $\bigwedge f. \text{nn-integral } N \ f = \text{nn-integral } \dots \ f$ **by** *simp*
also from *dens* **and** X
have $(\int^{+x}. f \ x * \text{indicator } (fst' - ' X \cap \text{space } N) \ x \ \partial ?N) =$
 $\int^{+x}. f \ (\text{case-prod } \text{PairVal } x) * \text{indicator } (fst' - ' X \cap \text{space } N) \ (\text{case-prod}$
 $\text{PairVal } x)$
 $\partial(N' \otimes_M \text{stock-measure } t2)$
by (*intro nn-integral-distr*)
(*simp-all add: measurable-embed-measure2 N-def N'-def fst'-def*)
also from *has-subprob-densityD(1)[OF dens]* **and** X
have $\dots = \int^{+x}. \int^{+y}. f \ \langle |x,y| \rangle * \text{indicator } (fst' - ' X \cap \text{space } N) \ \langle |x, y| \rangle$
 $\partial \text{stock-measure } t2 \ \partial N'$
(**is - =** $?I$)
by (*subst sigma-finite-measure.nn-integral-fst[symmetric]*)
(*auto simp: N-def N'-def fst'-def comp-def simp del: space-stock-measure*)
also from X **have** $A: \bigwedge x \ y. x \in \text{space } N' \implies y \in \text{space } (\text{stock-measure } t2) \implies$
 $\text{indicator } (fst' - ' X \cap \text{space } N) \ \langle |x, y| \rangle = \text{indicator } X \ x$
by (*auto split: split-indicator simp: fst'-def N-def*)
 $\text{space-embed-measure space-pair-measure } N'\text{-def}$)
have $?I = \int^{+x}. \int^{+y}. f \ \langle |x,y| \rangle * \text{indicator } X \ x \ \partial \text{stock-measure } t2 \ \partial N'$ (**is -**
 $= ?I$)
by (*intro nn-integral-cong*) (*simp add: A*)
also have $A: \bigwedge x. x \in \text{space } N' \implies (\lambda y. f \ \langle |x,y| \rangle) = f \circ \text{case-prod } \text{PairVal} \circ$
 $(\lambda y. (x,y))$
by (*intro ext*) *simp*
from *dens* **have** $?I = \int^{+x}. (\int^{+y}. f \ \langle |x,y| \rangle \ \partial \text{stock-measure } t2) * \text{indicator } X$
 $x \ \partial N'$
by (*intro nn-integral-cong nn-integral-multc, subst A*)
(*auto intro!: measurable-comp f measurable-PairVal simp: N'-def*)
also from *meas-f'* **and** $X(2)$ **have** $\dots = \text{emeasure } ?M2 \ X$ **unfolding** $f'\text{-def}$
by (*rule emeasure-density[symmetric]*)
finally show $\text{emeasure } ?M1 \ X = \text{emeasure } ?M2 \ X .$
qed

lemma *has-subprob-density-distr-Snd*:

fixes $t1 \ t2 \ f$

defines $N \equiv \text{stock-measure } (\text{PRODUCT } t1 \ t2)$

```

defines  $N' \equiv \text{stock-measure } t2$ 
defines  $\text{snd}' \equiv \text{op-sem } \text{Snd}$ 
defines  $f' \equiv \lambda y. \int^+ x. f \langle |x,y| \rangle \partial \text{stock-measure } t1$ 
assumes  $\text{dens}: \text{has-subprob-density } M \ N \ f$ 
shows  $\text{has-subprob-density } (\text{distr } M \ N' \ \text{snd}') \ N' \ f'$ 
proof (intro has-subprob-densityI measure-eqI)
from  $\text{dens}$  interpret  $\text{subprob-space } M$  by (rule has-subprob-densityD)
from  $\text{dens}$  have  $M\text{-}M: \text{measurable } M = \text{measurable } N$ 
  by (intro ext measurable-cong-sets) (auto dest: has-subprob-densityD)
hence  $\text{meas-snd}: \text{snd}' \in \text{measurable } M \ N'$  unfolding  $\text{snd}'\text{-def}$ 
  by (subst op-sem.simps) (simp add: N'-def N-def M-M)
thus  $\text{subprob-space } (\text{distr } M \ N' \ \text{snd}')$ 
  by (rule subprob-space-distr) (simp-all add: N'-def)

interpret  $t1: \text{sigma-finite-measure } \text{stock-measure } t1$  by simp
have  $A: (\lambda(x, y). f \langle |x, y| \rangle) = f \circ \text{case-prod } \text{PairVal}$ 
  by (intro ext) (simp add: o-def split: prod.split)
have  $f[\text{measurable}]: f \in \text{borel-measurable } (\text{stock-measure } (\text{PRODUCT } t1 \ t2))$ 
  using  $\text{dens}$  by (auto simp: has-subprob-density-def has-density-def N-def)
then show  $\text{meas-f}' : f' \in \text{borel-measurable } N'$  unfolding  $f'\text{-def } N'\text{-def}$ 
  by measurable

interpret  $N': \text{sigma-finite-measure } N'$ 
  unfolding  $N'\text{-def}$  by (rule sigma-finite-stock-measure)

interpret  $N'\text{-}t1: \text{pair-sigma-finite } t1 \ N'$  proof qed

let  $?M1 = \text{distr } M \ N' \ \text{snd}'$  and  $?M2 = \text{density } N' \ f'$ 
show  $\text{sets } ?M1 = \text{sets } ?M2$  by simp
fix  $X$  assume  $X \in \text{sets } ?M1$ 
hence  $X: X \in \text{sets } N' \ X \in \text{sets } N'$  by (simp-all add: N'-def)
then have  $[\text{measurable}]: X \in \text{sets } (\text{stock-measure } t2)$ 
  by (simp add: N'-def)

from  $\text{meas-snd}$  and  $X(1)$  have  $\text{emeasure } ?M1 \ X = \text{emeasure } M \ (\text{snd}' -' X \cap \text{space } M)$ 
  by (rule emeasure-distr)
also from  $\text{dens}$  have  $M: M = \text{density } N \ f$  by (rule has-subprob-densityD)
from  $\text{this}$  and  $\text{meas-snd}$  have  $\text{meas-snd}' : \text{snd}' \in \text{measurable } N \ N'$  by simp
with  $\text{dens}$  and  $X$  have  $\text{emeasure } M \ (\text{snd}' -' X \cap \text{space } M) =$ 
  
$$\int^+ x. f \ x * \text{indicator } (\text{snd}' -' X \cap \text{space } N) \ x \ \partial N$$

  by (subst (1 2) M, subst space-density, subst emeasure-density)
  (erule has-subprob-densityD, erule measurable-sets, simp, simp)
also have  $N = \text{distr } (\text{stock-measure } t1 \ \otimes_M \ N') \ N$  (case-prod PairVal) (is - = ?N)
  unfolding  $N\text{-def } N'\text{-def } \text{stock-measure.simps}$  by (rule embed-measure-eq-distr)
  (simp add: inj-PairVal)
hence  $\bigwedge f. \text{nn-integral } N \ f = \text{nn-integral } \dots \ f$  by simp
also from  $\text{dens}$  and  $X$ 

```

have $(\int^+ x. f x * \text{indicator } (\text{snd}' - ' X \cap \text{space } N) x \partial ?N) =$
 $\int^+ x. f (\text{case-prod } \text{PairVal } x) * \text{indicator } (\text{snd}' - ' X \cap \text{space } N)$
(case-prod PairVal x)
 $\partial(\text{stock-measure } t1 \otimes_M N')$
by *(intro nn-integral-distr)*
(simp-all add: measurable-embed-measure2 N-def N'-def snd'-def)
also from *has-subprob-densityD(1)[OF dens]* **and** X
have $\dots = \int^+ y. \int^+ x. f \langle |x,y| \rangle * \text{indicator } (\text{snd}' - ' X \cap \text{space } N) \langle |x, y| \rangle$
 $\partial \text{stock-measure } t1 \partial N'$
(is $\dots = ?I$
by *(subst N'-t1.nn-integral-snd[symmetric])*
(auto simp: N-def N'-def snd'-def comp-def simp del: space-stock-measure)
also from X **have** $A: \bigwedge x y. x \in \text{space } N' \implies y \in \text{space } (\text{stock-measure } t1) \implies$
 $\text{indicator } (\text{snd}' - ' X \cap \text{space } N) \langle |y, x| \rangle = \text{indicator } X x$
by *(auto split: split-indicator simp: snd'-def N-def*
space-embed-measure space-pair-measure N'-def)
have $?I = \int^+ y. \int^+ x. f \langle |x,y| \rangle * \text{indicator } X y \partial \text{stock-measure } t1 \partial N'$ **(is** $-$
 $= ?I$
by *(intro nn-integral-cong) (simp add: A)*
also have $A: \bigwedge y. y \in \text{space } N' \implies (\lambda x. f \langle |x,y| \rangle) = f \circ \text{case-prod } \text{PairVal} \circ$
 $(\lambda x. (x,y))$
by *(intro ext) simp*
from *dens* **have** $?I = \int^+ y. (\int^+ x. f \langle |x,y| \rangle \partial \text{stock-measure } t1) * \text{indicator } X$
 $y \partial N'$
by *(intro nn-integral-cong nn-integral-multc) (auto simp: N'-def)*
also from *meas-f'* **and** $X(2)$ **have** $\dots = \text{emeasure } ?M2 X$ **unfolding** f' -*def*
by *(rule emeasure-density[symmetric])*
finally show $\text{emeasure } ?M1 X = \text{emeasure } ?M2 X$.
qed

lemma *dens-ctxt-measure-empty-bind:*

assumes $\varrho \in \text{space } (\text{state-measure } V' \Gamma)$

assumes $f[\text{measurable}]: f \in \text{measurable } (\text{state-measure } V' \Gamma) (\text{subprob-algebra } N)$

shows $\text{dens-ctxt-measure } (\{\}, V', \Gamma, \lambda-. 1) \varrho \ggg f = f \varrho$ **(is** $\text{bind } ?M - = ?R$
proof *(intro measure-eqI)*

from *assms* **have** $\text{nonempty: space } ?M \neq \{\}$

by *(auto simp: dens-ctxt-measure-def state-measure'-def state-measure-def space-PiM)*

moreover have $\text{meas: measurable } ?M = \text{measurable } (\text{state-measure } V' \Gamma)$

by *(intro ext measurable-cong-sets) (auto simp: dens-ctxt-measure-def state-measure'-def)*

moreover from *assms* **have** $[\text{simp}]: \text{sets } (f \varrho) = \text{sets } N$

by *(intro sets-kernel[OF assms(2)])*

ultimately show $\text{sets-eq: sets } (?M \ggg f) = \text{sets } ?R$ **using** *assms*

by *(subst sets-bind[OF sets-kernel[OF f]])*

(simp-all add: dens-ctxt-measure-def state-measure'-def state-measure-def)

from *assms* **have** $[\text{simp}]: \bigwedge \sigma. \text{merge } \{\} V' (\sigma, \varrho) = \varrho$

by *(intro ext) (auto simp: merge-def state-measure-def space-PiM)*

```

fix X assume X: X ∈ sets (?M ≫ f)
hence emeasure (?M ≫ f) X = ∫+x. emeasure (f x) X ∂?M using assms
  by (subst emeasure-bind[OF nonempty]) (simp-all add: nonempty meas sets-eq
  cong: measurable-cong-sets)
also have ... = ∫+(x::state). emeasure (f ρ) X ∂count-space {λ-. undefined}
  unfolding dens-ctxt-measure-def state-measure'-def state-measure-def using X
  assms
    apply (simp only: prod.case)
    apply (subst nn-integral-density)
    apply (auto intro!: measurable-compose[OF - measurable-emeasure-subprob-algebra]
      simp: state-measure-def sets-eq PiM-empty) [3]
    apply (subst nn-integral-distr)
    apply (auto intro!: measurable-compose[OF - measurable-emeasure-subprob-algebra]
      simp: state-measure-def sets-eq PiM-empty)
  done
also have ... = emeasure (f ρ) X
  by (subst nn-integral-count-space-finite) (simp-all add: max-def)
finally show emeasure (?M ≫ f) X = emeasure (f ρ) X .
qed

```

```

lemma (in density-context) bind-dens-ctxt-measure-cong:
  assumes fg:  $\bigwedge \sigma. (\bigwedge x. x \in V' \implies \sigma x = \rho x) \implies f \sigma = g \sigma$ 
  assumes ρ[measurable]: ρ ∈ space (state-measure V' Γ)
  assumes Mf[measurable]: f ∈ measurable (state-measure (V ∪ V') Γ) (subprob-algebra
  N)
  assumes Mg[measurable]: g ∈ measurable (state-measure (V ∪ V') Γ) (subprob-algebra
  N)
  defines M ≡ dens-ctxt-measure (V, V', Γ, δ) ρ
  shows M ≫ f = M ≫ g
proof -
  have [measurable]: (λσ. merge V V' (σ, ρ)) ∈ measurable (state-measure V Γ)
  (state-measure (V ∪ V') Γ)
  using ρ unfolding state-measure-def by simp
  show ?thesis
  using disjoint
  apply (simp add: M-def dens-ctxt-measure-def state-measure'-def density-distr)
  apply (subst (1 2) bind-distr)
  apply measurable
  apply (intro bind-cong-AE[where B=N] AE-I2 refl fg)
  apply measurable
  done
qed

```

```

lemma (in density-context) bin-op-randomfree-restructure:
  assumes t1:  $\Gamma \vdash e : t$  and t2:  $\Gamma \vdash e' : t'$  and t3: op-type oper (PRODUCT t
  t') = Some tr
  assumes rf: randomfree e' and vars1: free-vars e ⊆ V ∪ V' and vars2: free-vars
  e' ⊆ V'
  assumes ρ: ρ ∈ space (state-measure V' Γ)

```

```

defines  $M \equiv \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho$ 
defines  $v \equiv \text{expr-sem-rf } \varrho \ e'$ 
shows  $M \gg \ (\lambda\sigma. \text{expr-sem } \sigma \ (\text{oper } \$\$ \langle e, e' \rangle)) =$ 
 $\text{distr } (M \gg \ (\lambda\sigma. \text{expr-sem } \sigma \ e)) \ (\text{stock-measure } \text{tr}) \ (\lambda w. \text{op-sem } \text{oper}$ 
 $\langle |w, v| \rangle)$ 
proof -
from assms have  $\text{vars1}' : \bigwedge \sigma. \sigma \in \text{space } M \implies \forall x \in \text{free-vars } e. \text{val-type } (\sigma \ x)$ 
 $= \Gamma \ x$ 
and  $\text{vars2}' : \bigwedge \sigma. \sigma \in \text{space } M \implies \forall x \in \text{free-vars } e'. \text{val-type } (\sigma \ x) = \Gamma$ 
 $x$ 
by (auto simp: M-def space-dens-ctxt-measure state-measure-def space-PiM
dest: PiE-mem)
have  $\text{Me} : (\lambda\sigma. \text{expr-sem } \sigma \ e) \in$ 
 $\text{measurable } (\text{state-measure } (V \cup V') \ \Gamma) \ (\text{subprob-algebra } (\text{stock-measure}$ 
 $t))$ 
by (rule measurable-expr-sem[OF t1 vars1])

from assms have  $e' : \bigwedge \sigma. \sigma \in \text{space } M \implies \text{expr-sem } \sigma \ e' = \text{return-val } (\text{expr-sem-rf}$ 
 $\sigma \ e')$ 
by (intro expr-sem-rf-sound[symmetric]) (auto simp: M-def space-dens-ctxt-measure)
from assms have  $\text{vt-e}' : \bigwedge \sigma. \sigma \in \text{space } M \implies \text{val-type } (\text{expr-sem-rf } \sigma \ e') = t'$ 
by (intro val-type-expr-sem-rf) (auto simp: M-def space-dens-ctxt-measure)

let  $?tt' = \text{PRODUCT } t \ t'$ 
{
fix  $\sigma$  assume  $\sigma : \sigma \in \text{space } M$ 
with  $\text{vars2}$  have [simp]:  $\text{measurable } (\text{expr-sem } \sigma \ e') = \text{measurable } (\text{stock-measure}$ 
 $t')$ 
by (intro measurable-expr-sem-eq[OF t2, of - V ∪ V']) (auto simp: M-def
 $\text{space-dens-ctxt-measure}$ )
from  $\sigma$  have [simp]:  $\text{space } (\text{expr-sem } \sigma \ e) = \text{space } (\text{stock-measure } t)$ 
 $\text{space } (\text{expr-sem } \sigma \ e') = \text{space } (\text{stock-measure } t')$ 
using  $\text{space-expr-sem}[OF \ t1 \ \text{vars1}' [OF \ \sigma]] \ \text{space-expr-sem}[OF \ t2 \ \text{vars2}' [OF$ 
 $\sigma]]$  by simp-all
have  $\text{expr-sem } \sigma \ e \gg \ (\lambda x. \text{expr-sem } \sigma \ e' \gg \ (\lambda y. \text{return-val } \langle |x, y| \rangle)) =$ 
 $\text{expr-sem } \sigma \ e \gg \ (\lambda x. \text{return-val } (\text{expr-sem-rf } \sigma \ e')) \gg \ (\lambda y. \text{return-val}$ 
 $\langle |x, y| \rangle)$ 
by (intro bind-cong refl, subst e'[OF σ]) simp
also have  $\dots = \text{expr-sem } \sigma \ e \gg \ (\lambda x. \text{return-val } \langle |x, \text{expr-sem-rf } \sigma \ e'| \rangle)$ 
using  $\sigma \ \text{vars2}$ 
by (intro bind-cong refl, subst bind-return-val'[of - t' - ?tt'])
(auto simp: vt-e' M-def space-dens-ctxt-measure
 $\text{intro!: measurable-PairVal})$ 
finally have  $\text{expr-sem } \sigma \ e \gg \ (\lambda x. \text{expr-sem } \sigma \ e' \gg \ (\lambda y. \text{return-val } \langle |x, y| \rangle))$ 
 $=$ 
 $\text{expr-sem } \sigma \ e \gg \ (\lambda x. \text{return-val } \langle |x, \text{expr-sem-rf } \sigma \ e'| \rangle) .$ 
}
hence  $M \gg \ (\lambda\sigma. \text{expr-sem } \sigma \ (\text{oper } \$\$ \langle e, e' \rangle)) =$ 
 $M \gg \ (\lambda\sigma. (\text{expr-sem } \sigma \ e \gg \ (\lambda x. \text{return-val } \langle |x, \text{expr-sem-rf } \sigma \ e'| \rangle))$ 

```

$\gg= (\lambda x. \text{return-val } (\text{op-sem oper } x))$ (**is - = ?T**)
by (*intro bind-cong refl*) (*simp only: expr-sem.simps*)
also have [*measurable*]: $\bigwedge \sigma. \sigma \in \text{space } M \implies \text{expr-sem-rf } \sigma \ e' \in \text{space } t'$
by (*simp add: type-universe-def vt-e' del: type-universe-type*)
note [*measurable*] = *measurable-op-sem*[*OF t3*]
hence $?T = M \gg= (\lambda \sigma. \text{expr-sem } \sigma \ e \gg= (\lambda x. \text{return-val } (\text{op-sem oper } <|x, \text{expr-sem-rf } \sigma \ e'|>)))$
(**is - = ?T**)
by (*intro bind-cong*[*OF refl*], *subst bind-assoc-return-val*[*of - t - ?tt' - tr*])
(*auto simp: sets-expr-sem*[*OF t1 vars1*'])
also have *eq*: $\bigwedge \sigma. (\bigwedge x. x \in V' \implies \sigma \ x = \varrho \ x) \implies \text{expr-sem-rf } \sigma \ e' = \text{expr-sem-rf } \varrho \ e'$
using *vars2* **by** (*intro expr-sem-rf-eq-on-vars*) *auto*
have [*measurable*]: $(\lambda \sigma. \text{expr-sem-rf } \sigma \ e') \in \text{measurable } (\text{state-measure } (V \cup V') \Gamma)$ (*stock-measure t'*)
using *vars2* **by** (*intro measurable-expr-sem-rf*[*OF t2 rf*]) *blast*
note [*measurable*] = *Me measurable-bind measurable-return-val*
have *expr-sem-rf-space*: $\text{expr-sem-rf } \varrho \ e' \in \text{space } (\text{stock-measure } t')$
using *val-type-expr-sem-rf*[*OF t2 rf vars2*']
by (*simp add: type-universe-def del: type-universe-type*)
hence $?T = M \gg= (\lambda \sigma. \text{expr-sem } \sigma \ e \gg= (\lambda x. \text{return-val } (\text{op-sem oper } <|x, \text{expr-sem-rf } \varrho \ e'|>)))$
using ϱ **unfolding** *M-def*
by (*intro bind-dens-ctxt-measure-cong, subst eq*) (*simp, simp, simp, measurable*)
also have $\dots = (M \gg= (\lambda \sigma. \text{expr-sem } \sigma \ e)) \gg=$
 $\text{return-val } \circ (\lambda x. \text{op-sem oper } <|x, \text{expr-sem-rf } \varrho \ e'|>)$
using *expr-sem-rf-space*
by (*subst bind-assoc*[*of - - stock-measure t - stock-measure tr, symmetric*])
(*simp-all add: M-def measurable-dens-ctxt-measure-eq o-def*)
also have $\dots = \text{distr } (M \gg= (\lambda \sigma. \text{expr-sem } \sigma \ e))$ (*stock-measure tr*)
 $(\lambda x. \text{op-sem oper } <|x, \text{expr-sem-rf } \varrho \ e'|>)$ **using** *Me*
expr-sem-rf-space
by (*subst bind-return-val-distr*[*of - t - tr*])
(*simp-all add: M-def sets-expr-sem*[*OF t1 vars1*'])
finally show *?thesis unfolding v-def* .
qed

lemma *addc-density-measurable*:

assumes *Mf*: *case-prod f* \in *borel-measurable* (*state-measure* $V' \Gamma \otimes_M$ *stock-measure* *t*)

assumes *t-disj*: $t = \text{REAL} \vee t = \text{INTEG}$ **and** $t: \Gamma \vdash e' : t$

assumes *rf*: *randomfree e'* **and** *vars*: *free-vars e'* $\subseteq V'$

defines $f' \equiv (\lambda \varrho \ x. f \ \varrho \ (\text{op-sem Add } <|x, \text{expr-sem-rf } \varrho \ (\text{Minus } \ \$\$ \ e')|>))$

shows *case-prod f'* \in *borel-measurable* (*state-measure* $V' \Gamma \otimes_M$ *stock-measure* *t*)

proof (*insert t-disj, elim disjE*)

assume *A*: $t = \text{REAL}$

from *A* **and** *t* **have** $t': \Gamma \vdash e' : \text{REAL}$ **by** *simp*

with *rf vars* **have** *vt-e'*:

$\wedge \varrho. \varrho \in \text{space } (\text{state-measure } V' \Gamma) \implies \text{val-type } (\text{expr-sem-rf } \varrho \ e') = \text{REAL}$
by (intro val-type-expr-sem-rf) simp-all
let $?f' = \lambda \sigma \ x. \text{ let } c = \text{expr-sem-rf } \sigma \ e'$
 $\text{ in } f \ \sigma \ (\text{RealVal } (\text{extract-real } x - \text{extract-real } c))$
note $Mf[\text{unfolded } A, \text{measurable}]$ **and** $rf[\text{measurable}]$ **and** $\text{vars}[\text{measurable}]$ **and**
 $t[\text{unfolded } A, \text{measurable}]$
have $\text{case-prod } ?f' \in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t)$
unfolding *Let-def A by measurable*
also have $\text{case-prod } ?f' \in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t) \longleftrightarrow$
 $\text{case-prod } f' \in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t)$
by (intro measurable-cong)
(auto simp: Let-def space-pair-measure A space-embed-measure f'-def lift-RealIntVal2-def lift-RealIntVal-def extract-real-def dest!: vt-e' split: val.split)
finally show *?thesis .*
next
assume $A: t = \text{INTEG}$
with t **have** $t': \Gamma \vdash e' : \text{INTEG}$ **by** simp
with rf **vars** **have** $vt-e'$:
 $\wedge \varrho. \varrho \in \text{space } (\text{state-measure } V' \Gamma) \implies \text{val-type } (\text{expr-sem-rf } \varrho \ e') = \text{INTEG}$
by (intro val-type-expr-sem-rf) simp-all
let $?f' = \lambda \sigma \ x. \text{ let } c = \text{expr-sem-rf } \sigma \ e'$
 $\text{ in } f \ \sigma \ (\text{IntVal } (\text{extract-int } x - \text{extract-int } c))$
note $Mf[\text{unfolded } A, \text{measurable}]$ **and** $rf[\text{measurable}]$ **and** $\text{vars}[\text{measurable}]$ **and**
 $t[\text{unfolded } A, \text{measurable}]$
have $Mdiff: \text{case-prod } ((-) :: \text{int} \Rightarrow -) \in$
 $\text{measurable } (\text{count-space UNIV} \otimes_M \text{count-space UNIV}) (\text{count-space UNIV})$ **by** simp
have $\text{case-prod } ?f' \in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t)$
unfolding *Let-def A by measurable*
also have $\text{case-prod } ?f' \in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t) \longleftrightarrow$
 $\text{case-prod } f' \in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t)$
by (intro measurable-cong)
(auto simp: Let-def space-pair-measure A space-embed-measure f'-def lift-RealIntVal2-def lift-RealIntVal-def extract-int-def dest!: vt-e' split: val.split)
finally show *?thesis .*
qed

lemma (in *density-context*) *emeasure-bind-if-dens-ctxt-measure:*

assumes $\varrho: \varrho \in \text{space } (\text{state-measure } V' \Gamma)$

defines $M \equiv \text{dens-ctxt-measure } \mathcal{Y} \ \varrho$

assumes $Mf[\text{measurable}]: f \in \text{measurable } M$ (*subprob-algebra (stock-measure*

```

BOOL))
assumes Mg[measurable]:  $g \in \text{measurable } M \text{ (subprob-algebra } R)$ 
assumes Mh[measurable]:  $h \in \text{measurable } M \text{ (subprob-algebra } R)$ 
assumes densf: has-parametrized-subprob-density (state-measure ( $V \cup V'$ )  $\Gamma$ )
  f (stock-measure BOOL)  $\delta f$ 
assumes densg: has-parametrized-subprob-density (state-measure  $V' \Gamma$ )
  ( $\lambda \rho. \text{dens-ctxt-measure } (V, V', \Gamma, \lambda \sigma. \delta \sigma * \delta f \sigma \text{ (BoolVal True)})$ )
 $\rho \ggg g$ )  $R \delta g$ 
assumes densh: has-parametrized-subprob-density (state-measure  $V' \Gamma$ )
  ( $\lambda \rho. \text{dens-ctxt-measure } (V, V', \Gamma, \lambda \sigma. \delta \sigma * \delta f \sigma \text{ (BoolVal False)})$ )
 $\rho \ggg h$ )  $R \delta h$ 
defines  $P \equiv \lambda b. b = \text{BoolVal True}$ 
shows  $M \ggg (\lambda x. f x \ggg (\lambda b. \text{if } P \text{ then } g x \text{ else } h x)) = \text{density } R (\lambda x. \delta g \rho$ 
 $x + \delta h \rho x)$ 
  (is ?lhs = ?rhs)
proof (intro measure-eqI)
  have sets-lhs: sets ?lhs = sets  $R$ 
    apply (subst sets-bind-measurable[of - -  $R$ ])
    apply measurable
    apply (simp-all add: P-def M-def)
    done
  thus sets ?lhs = sets ?rhs by simp

fix  $X$  assume  $X \in \text{sets } ?lhs$ 
hence  $X$ :  $X \in \text{sets } R$  by (simp only: sets-lhs)
from  $Mf$  have [simp]:  $\bigwedge x. x \in \text{space } M \implies \text{sets } (f x) = \text{sets } (\text{stock-measure}$ 
 $\text{BOOL})$ 
  by (rule sets-kernel)
note [simp] = sets-eq-imp-space-eq[OF this]
from has-parametrized-subprob-densityD( $\mathcal{B}$ )[OF densf]
  have  $M \delta f$ [measurable]:  $(\lambda(x, y). \delta f x y)$ 
     $\in \text{borel-measurable } (\text{state-measure } (V \cup V') \Gamma \otimes_M \text{stock-measure } \text{BOOL})$ 
    by (simp add: M-def dens-ctxt-measure-def state-measure'-def)
  have [measurable]: Measurable.pred (stock-measure BOOL)  $P$ 
    unfolding  $P$ -def by simp
  have BoolVal-in-space:  $\text{BoolVal True} \in \text{space } (\text{stock-measure } \text{BOOL})$ 
     $\text{BoolVal False} \in \text{space } (\text{stock-measure } \text{BOOL})$  by auto
  from  $Mg$  have  $Mg'$ [measurable]:  $g \in \text{measurable } (\text{state-measure } (V \cup V') \Gamma)$ 
  (subprob-algebra  $R$ )
    by (simp add: M-def measurable-dens-ctxt-measure-eq)
  from  $Mh$  have  $Mh'$ [measurable]:  $h \in \text{measurable } (\text{state-measure } (V \cup V') \Gamma)$ 
  (subprob-algebra  $R$ )
    by (simp add: M-def measurable-dens-ctxt-measure-eq)
  from densf have densf': has-parametrized-subprob-density  $M f$  (stock-measure
  BOOL)  $\delta f$ 
    unfolding has-parametrized-subprob-density-def
    apply (subst measurable-cong-sets, subst sets-pair-measure-cong)
    apply (unfold M-def dens-ctxt-measure-def state-measure'-def, (subst prod.case)+)
   $\square$ 

```

apply (*subst sets-density, subst sets-distr, rule refl, rule refl, rule refl, rule refl*)
apply (*auto simp: M-def space-dens-ctxt-measure*)
done

interpret *dc-True: density-context V V' Γ λσ. δ σ * δf σ (BoolVal True)*
using *density-context-if-dens[of - δf True] densf unfolding if-dens-def by*
(*simp add: stock-measure.simps*)
interpret *dc-False: density-context V V' Γ λσ. δ σ * δf σ (BoolVal False)*
using *density-context-if-dens[of - δf False] densf unfolding if-dens-def by*
(*simp add: stock-measure.simps*)

have *emeasure (M ≫ (λx. f x ≫ (λb. if P b then g x else h x))) X =*
 $\int^{+x}. \text{emeasure } (f x \gg (\lambda b. \text{if } P b \text{ then } g x \text{ else } h x)) X \partial M$ **using** *X*
by (*subst emeasure-bind[of - - R], simp add: M-def, intro measurable-bind[OF*
Mf], measurable)
also have $\dots = \int^{+x}. \int^{+b}. \text{emeasure } (\text{if } P b \text{ then } g x \text{ else } h x) X \partial f x \partial M$
by (*intro nn-integral-cong*) (*simp-all add: X emeasure-bind[where N=R]*)
also have $\dots = \int^{+x}. \int^{+b}. \text{emeasure } (\text{if } P b \text{ then } g x \text{ else } h x) X * \delta f x b$
 $\partial \text{stock-measure } \text{BOOL } \partial M$
using *has-parametrized-subprob-densityD[OF densf]*
by (*intro nn-integral-cong*)
(*simp-all add: AE-count-space field-simps nn-integral-density*
M-def space-dens-ctxt-measure stock-measure.simps)
also have $\dots = \int^{+x}. \text{emeasure } (g x) X * \delta f x (\text{BoolVal True}) +$
 $\text{emeasure } (h x) X * \delta f x (\text{BoolVal False}) \partial M$
using *has-parametrized-subprob-densityD[OF densf']*
by (*intro nn-integral-cong, subst nn-integral-BoolVal*)
(*auto simp: P-def nn-integral-BoolVal*)
also have $\dots = (\int^{+x}. \text{emeasure } (g x) X * \delta f x (\text{BoolVal True}) \partial M) +$
 $(\int^{+x}. \text{emeasure } (h x) X * \delta f x (\text{BoolVal False}) \partial M)$ **using** *X*
using *has-parametrized-subprob-densityD[OF densf'] BoolVal-in-space*
by (*intro nn-integral-add*) (*auto simp:*)
also have $(\int^{+x}. \text{emeasure } (g x) X * \delta f x (\text{BoolVal True}) \partial M) =$
 $\int^{+x}. \delta (\text{merge } V V' (x, \varrho)) * \delta f (\text{merge } V V' (x, \varrho)) (\text{BoolVal True})$
*
 $(\text{emeasure } (g (\text{merge } V V' (x, \varrho)))) X \partial \text{state-measure } V \Gamma$
using *X has-parametrized-subprob-densityD[OF densf] BoolVal-in-space un-*
folding *M-def*
by (*subst nn-integral-dens-ctxt-measure*) (*simp-all add: ρ mult-ac*)
also have $\dots = \text{emeasure } (\text{density } R (\delta g \varrho)) X$ **using** *ρ X*
apply (*subst dc-True.nn-integral-dens-ctxt-measure[symmetric], simp-all*) \square
apply (*subst emeasure-bind[of - - R, symmetric], simp-all add: measurable-dens-ctxt-measure-eq*)
 \square
apply (*subst has-parametrized-subprob-densityD(1)[OF densg], simp-all*)
done
also have $(\int^{+x}. \text{emeasure } (h x) X * \delta f x (\text{BoolVal False}) \partial M) =$
 $\int^{+x}. \delta (\text{merge } V V' (x, \varrho)) * \delta f (\text{merge } V V' (x, \varrho)) (\text{BoolVal False})$
*
 $(\text{emeasure } (h (\text{merge } V V' (x, \varrho)))) X \partial \text{state-measure } V \Gamma$

```

    using X has-parametrized-subprob-densityD[OF densf] BoolVal-in-space un-
folding M-def
    by (subst nn-integral-dens-ctxt-measure) (simp-all add:  $\varrho$  mult-ac)
    also have ... = emeasure (density R ( $\delta h \varrho$ )) X using  $\varrho$  X
    apply (subst dc-False.nn-integral-dens-ctxt-measure[symmetric], simp-all) []
    apply (subst emeasure-bind[of - - R, symmetric], simp-all add: measurable-dens-ctxt-measure-eq)
  []
  apply (subst has-parametrized-subprob-densityD(1)[OF densh], simp-all)
  done
  also have emeasure (density R ( $\delta g \varrho$ )) X + emeasure (density R ( $\delta h \varrho$ )) X =
    emeasure (density R ( $\lambda x. \delta g \varrho x + \delta h \varrho x$ )) X using X  $\varrho$ 
  using has-parametrized-subprob-densityD(2,3)[OF densg]
    has-parametrized-subprob-densityD(2,3)[OF densh]
  by (intro emeasure-density-add) simp-all
  finally show emeasure ?lhs X = emeasure ?rhs X .
qed

```

lemma (in density-context) emeasure-bind-if-det-dens-ctxt-measure:

```

fixes f
assumes  $\varrho$ :  $\varrho \in \text{space (state-measure } V' \Gamma)$ 
defines M  $\equiv$  dens-ctxt-measure  $\mathcal{Y} \varrho$ 
defines P  $\equiv$   $\lambda b. f b = \text{BoolVal True}$  and P'  $\equiv$   $\lambda b. f b = \text{BoolVal False}$ 
assumes dc1: density-context V V'  $\Gamma$  ( $\lambda \sigma. \delta \sigma * (\text{if } P \sigma \text{ then } 1 \text{ else } 0)$ )
assumes dc2: density-context V V'  $\Gamma$  ( $\lambda \sigma. \delta \sigma * (\text{if } P' \sigma \text{ then } 1 \text{ else } 0)$ )
assumes Mf[measurable]:  $f \in \text{measurable } M$  (stock-measure BOOL)
assumes Mg[measurable]:  $g \in \text{measurable } M$  (subprob-algebra R)
assumes Mh[measurable]:  $h \in \text{measurable } M$  (subprob-algebra R)
assumes densg: has-parametrized-subprob-density (state-measure V'  $\Gamma$ )
    ( $\lambda \varrho. \text{dens-ctxt-measure (V, V', \Gamma, \lambda \sigma. \delta \sigma * (\text{if } P \sigma \text{ then } 1 \text{ else } 0))$ )
 $\varrho \ggg g$ ) R  $\delta g$ 
assumes densh: has-parametrized-subprob-density (state-measure V'  $\Gamma$ )
    ( $\lambda \varrho. \text{dens-ctxt-measure (V, V', \Gamma, \lambda \sigma. \delta \sigma * (\text{if } P' \sigma \text{ then } 1 \text{ else } 0))$ )
 $\varrho \ggg h$ ) R  $\delta h$ 
shows M  $\ggg$  ( $\lambda x. \text{if } P x \text{ then } g x \text{ else } h x$ ) = density R ( $\lambda x. \delta g \varrho x + \delta h \varrho x$ )
    (is ?lhs = ?rhs)
proof (intro measure-eqI)
  have [measurable]: Measurable.pred M P
    unfolding P-def by (rule pred-eq-const1[OF Mf]) simp
  have [measurable]: Measurable.pred M P'
    unfolding P'-def by (rule pred-eq-const1[OF Mf]) simp
  have sets-lhs: sets ?lhs = sets R
    by (subst sets-bind-measurable[of - - R]) (simp-all, simp add: M-def)
  thus sets ?lhs = sets ?rhs by simp
  from Mg have Mg'[measurable]:  $g \in \text{measurable (state-measure (V } \cup V') \Gamma)$ 
    (subprob-algebra R)
    by (simp add: M-def measurable-dens-ctxt-measure-eq)
  from Mh have Mh'[measurable]:  $h \in \text{measurable (state-measure (V } \cup V') \Gamma)$ 
    (subprob-algebra R)
    by (simp add: M-def measurable-dens-ctxt-measure-eq)

```

```

have [simp]:  $\bigwedge x. x \in \text{space } M \implies \text{sets } (g \ x) = \text{sets } R$ 
  by (rule sets-kernel[OF Mg])
have [simp]:  $\bigwedge x. x \in \text{space } M \implies \text{sets } (h \ x) = \text{sets } R$ 
  by (rule sets-kernel[OF Mh])
have [simp]:  $\text{sets } M = \text{sets } (\text{state-measure } (V \cup V') \ \Gamma)$ 
  by (simp add: M-def dens-ctxt-measure-def state-measure'-def)
then have [measurable-cong]:  $\text{sets } (\text{state-measure } (V \cup V') \ \Gamma) = \text{sets } M \ ..$ 
have [simp]:  $\text{range } \text{BoolVal} = \{\text{BoolVal True}, \text{BoolVal False}\}$  by auto

fix X assume X  $\in \text{sets } ?lhs$ 
hence X[measurable]:  $X \in \text{sets } R$  by (simp only: sets-lhs)

interpret dc-True: density-context V V'  $\Gamma \ \lambda\sigma. \delta \ \sigma * (\text{if } P \ \sigma \text{ then } 1 \text{ else } 0)$  by
fact
interpret dc-False: density-context V V'  $\Gamma \ \lambda\sigma. \delta \ \sigma * (\text{if } P' \ \sigma \text{ then } 1 \text{ else } 0)$  by
fact

have emeasure (M  $\ggg (\lambda x. \text{if } P \ x \text{ then } g \ x \text{ else } h \ x)$ ) X =
   $\int^+ x. (\text{if } P \ x \text{ then } \text{emeasure } (g \ x) \ X \text{ else } \text{emeasure } (h \ x) \ X) \ \partial M$  using X
  by (subst emeasure-bind[of - R], simp add: M-def, measurable)
  (intro nn-integral-cong, simp)
also have ... =  $\int^+ x. (\text{if } P \ x \text{ then } 1 \text{ else } 0) * \text{emeasure } (g \ x) \ X +$ 
   $(\text{if } P' \ x \text{ then } 1 \text{ else } 0) * \text{emeasure } (h \ x) \ X \ \partial M$  using X
  using measurable-space[OF Mf]
  by (intro nn-integral-cong) (auto simp add: P-def P'-def stock-measure.simps)
also have ... =  $(\int^+ x. (\text{if } P \ x \text{ then } 1 \text{ else } 0) * \text{emeasure } (g \ x) \ X \ \partial M) +$ 
   $(\int^+ x. (\text{if } P' \ x \text{ then } 1 \text{ else } 0) * \text{emeasure } (h \ x) \ X \ \partial M)$  using X
  by (intro nn-integral-add) (simp-all add:)
also have ... =  $(\int^+ y. \delta g \ \varrho \ y * \text{indicator } X \ y \ \partial R) + (\int^+ y. \delta h \ \varrho \ y * \text{indicator}$ 
X y  $\partial R)$ 
  unfolding M-def using  $\varrho \ X$ 
  apply (simp add: nn-integral-dens-ctxt-measure)
  apply (subst (1 2) mult.assoc[symmetric])
  apply (subst dc-True.nn-integral-dens-ctxt-measure[symmetric], simp, simp)
  apply (subst dc-False.nn-integral-dens-ctxt-measure[symmetric], simp, simp)
  apply (subst (1 2) emeasure-bind[symmetric], simp-all add: measurable-dens-ctxt-measure-eq)
  apply measurable
  apply (subst emeasure-has-parametrized-subprob-density[OF densg], simp, simp)
  apply (subst emeasure-has-parametrized-subprob-density[OF densh], simp-all)
  done
also have ... =  $\text{emeasure } (\text{density } R \ (\lambda x. \delta g \ \varrho \ x + \delta h \ \varrho \ x)) \ X$  using X  $\varrho$ 
  using has-parametrized-subprob-densityD(2,3)[OF densg]
  using has-parametrized-subprob-densityD(2,3)[OF densh]
  apply (subst (1 2) emeasure-density[symmetric], simp-all) []
  apply (intro emeasure-density-add, simp-all)
  done
finally show emeasure ?lhs X = emeasure ?rhs X .
qed

```

7.3 Soundness proof

lemma *restrict-state-measure[measurable]*:

$(\lambda x. \text{restrict } x \ V') \in \text{measurable } (\text{state-measure } (V \cup V') \ \Gamma) \ (\text{state-measure } V' \ \Gamma)$

by (*simp add: state-measure-def*)

lemma *expr-has-density-sound-op*:

assumes *dens-ctxt: density-context* $V \ V' \ \Gamma \ \delta$

assumes *dens: has-parametrized-subprob-density* $(\text{state-measure } V' \ \Gamma)$

$(\lambda \varrho. \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \ \varrho \ggg (\lambda \sigma. \text{expr-sem } \sigma \ e))$

$(\text{stock-measure } t) \ f$

assumes *Mg: case-prod* $g \in \text{borel-measurable } (\text{state-measure } V' \ \Gamma \otimes_M \text{stock-measure } t')$

assumes *dens'*: $\bigwedge M \ \varrho. \text{has-subprob-density } M \ (\text{stock-measure } t) \ (f \ \varrho) \implies$
 $\text{has-density } (\text{distr } M \ (\text{stock-measure } t') \ (\text{op-sem } \text{oper}))$

$(\text{stock-measure } t') \ (g \ \varrho)$

assumes *t1*: $\Gamma \vdash e : t$ **and** *t2* : *op-type* $\text{oper } t = \text{Some } t'$

assumes *free-vars*: $\text{free-vars } (\text{oper } \ \$\$ \ e) \subseteq V \cup V'$

shows *has-parametrized-subprob-density* $(\text{state-measure } V' \ \Gamma)$

$(\lambda \varrho. \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \ \varrho \ggg (\lambda \sigma. \text{expr-sem } \sigma \ (\text{oper } \ \$\$ \ e)))$

$(\text{stock-measure } t') \ g$

proof –

interpret *density-context* $V \ V' \ \Gamma \ \delta$ **by fact**

show *?thesis unfolding* *has-parametrized-subprob-density-def*

proof (*intro conjI ballI impI*)

show *case-prod* $g \in \text{borel-measurable } (\text{state-measure } V' \ \Gamma \otimes_M \text{stock-measure } t')$ **by fact**

fix ϱ **assume** $\varrho : \varrho \in \text{space } (\text{state-measure } V' \ \Gamma)$

let $?M = \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \ \varrho$

have $Me : (\lambda \sigma. \text{expr-sem } \sigma \ e) \in \text{measurable } ?M \ (\text{subprob-algebra } (\text{stock-measure } t))$

by (*subst measurable-dens-ctxt-measure-eq*)

(*insert assms t1, auto intro!: measurable-expr-sem*)

from *dens* **and** ϱ **have** *dens: has-subprob-density* $(?M \ggg (\lambda \sigma. \text{expr-sem } \sigma \ e))$
 $(\text{stock-measure } t) \ (f \ \varrho)$

unfolding *has-parametrized-subprob-density-def* **by auto**

have *has-subprob-density* $(\text{distr } (?M \ggg (\lambda \sigma. \text{expr-sem } \sigma \ e)) \ (\text{stock-measure } t') \ (\text{op-sem } \text{oper}))$

$(\text{stock-measure } t') \ (g \ \varrho)$ (**is** *has-subprob-density* $?N \ -$)

proof (*unfold has-subprob-density-def, intro conjI*)

show *subprob-space* $?N$

apply (*intro subprob-space.subprob-space-distr has-subprob-densityD[OF dens]*)

apply (*subst measurable-cong-sets[OF sets-bind-measurable refl]*)

apply (*rule Me*)

apply (*simp-all add: measurable-op-sem t2*)

done

from *dens* **show** *has-density* $?N \ (\text{stock-measure } t') \ (g \ \varrho)$

by (intro dens¹) (simp add: has-subprob-density-def)
 qed
 also from *assms* and ϱ
 have $?N = ?M \gg (\lambda\sigma. \text{expr-sem } \sigma \text{ (oper } \$\$ e))$
 by (intro *expr-sem-op-eq-distr*[*symmetric*] *expr-typing.intros*) *simp-all*
 finally show *has-subprob-density* ... (stock-measure t') (g ϱ) .
 qed
 qed

lemma *expr-has-density-sound-aux*:
 assumes $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Gamma \vdash e : t$
 density-context $V V' \Gamma \delta$ *free-vars* $e \subseteq V \cup V'$
 shows *has-parametrized-subprob-density* (state-measure $V' \Gamma$)
 $(\lambda\varrho. \text{do } \{\sigma \leftarrow \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho; \text{expr-sem } \sigma e\})$
 (stock-measure t)
 $(\lambda\varrho x. f \varrho x)$
 using *assms*
proof (*induction arbitrary: t rule: expr-has-density.induct*[*split-format* (*complete*)])
 case (*hd-AE* $V V' \Gamma \delta e f t f' t'$)
 from $\langle \Gamma \vdash e : t \rangle$ and $\langle \Gamma \vdash e : t \rangle$ have $t[\text{simp}] : t' = t$
 by (*rule expr-typing-unique*)
 have *has-parametrized-subprob-density* (state-measure $V' \Gamma$)
 $(\lambda\varrho. \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho \gg (\lambda\sigma. \text{expr-sem } \sigma e))$ (stock-measure
 t) *f* (is $?P$)
 by (*intro hd-AE.IH*) *fact+*
 from *has-parametrized-subprob-density-dens-AE*[*OF hd-AE.hyps*(3,4) *this*] **show**
 $?case$ by *simp*
next

 case (*hd-dens-ctxt-cong* $V V' \Gamma \delta e f \delta' t$)
interpret dc' : *density-context* $V V' \Gamma \delta'$ by *fact*
from *hd-dens-ctxt-cong.hyps* and $dc'.\text{measurable-dens}$
 have $[\text{simp}] : \delta \in \text{borel-measurable } (\text{state-measure } (V \cup V') \Gamma)$
 by (*erule-tac subst*[*OF measurable-cong, rotated*]) *simp*
hence *density-context* $V V' \Gamma \delta$
 by (*intro density-context-equiv*[*OF hd-dens-ctxt-cong.hyps*(2)[*symmetric*]])
 (*insert hd-dens-ctxt-cong.prem*s *hd-dens-ctxt-cong.hyps*, *simp-all*)
hence *has-parametrized-subprob-density* (state-measure $V' \Gamma$)
 $(\lambda\varrho. \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho \gg (\lambda\sigma. \text{expr-sem } \sigma e))$ (stock-measure
 t) *f* (is $?P$)
 using *hd-dens-ctxt-cong.prem*s *hd-dens-ctxt-cong.hyps*
 by (*intro hd-dens-ctxt-cong.IH*) *simp-all*
also have $\bigwedge \sigma. \sigma \in \text{space } (\text{state-measure } V' \Gamma) \Longrightarrow$
 $\text{dens-ctxt-measure } (V, V', \Gamma, \delta') \sigma = \text{dens-ctxt-measure } (V, V', \Gamma, \delta)$
 σ
 by (*auto simp: dens-ctxt-measure-def state-measure'-def AE-distr-iff hd-dens-ctxt-cong.hyps*
 intro! : density-cong)
hence $?P \longleftrightarrow ?case$ by (*intro has-parametrized-subprob-density-cong*) *simp*
finally show $?case$.

next
case (*hd-val* v V V' Γ δ t)
hence [*simp*]: $t = \text{val-type } v$ **by** *auto*
interpret *density-context* V V' Γ δ **by** *fact*
show ?*case*
proof (*rule has-parametrized-subprob-densityI*)
show $(\lambda(\varrho, y). \text{branch-prob } (V, V', \Gamma, \delta) \varrho * \text{indicator } \{v\} y) \in$
 $\text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t)$
by (*subst measurable-split-conv*)
 $(\text{auto intro!}: \text{measurable-compose}[\text{OF measurable-snd borel-measurable-indicator}]$
 $\text{borel-measurable-times-ennreal})$
fix ϱ **assume** $\varrho: \varrho \in \text{space } (\text{state-measure } V' \Gamma)$
have *return-prob-space*: *prob-space* (*return-val* v) **unfolding** *return-val-def*
by (*simp add: prob-space-return*)
thus *subprob-space* (*dens-ctxt-measure* $(V, V', \Gamma, \delta) \varrho \ggg (\lambda\sigma. \text{expr-sem } \sigma (\text{Val}$
 $v)))$ **using** ϱ
by (*auto simp: return-val-def*
 $\text{intro!}: \text{measurable-compose}[\text{OF measurable-const return-measurable}]$
subprob-space-bind
 $\text{subprob-space-dens hd-val.prem}$)
from *hd-val.hyps* **have** *stock-measure* (*val-type* v) = *count-space* (*type-universe*
 t)
by (*simp add: countable-type-imp-count-space*)
thus *dens-ctxt-measure* $\mathcal{Y} \varrho \ggg (\lambda\sigma. \text{expr-sem } \sigma (\text{Val } v)) =$
 $\text{density } (\text{stock-measure } t) (\lambda x. \text{branch-prob } \mathcal{Y} \varrho * \text{indicator } \{v\} x)$
by (*subst expr-sem.simps, subst dens-ctxt-measure-bind-const, insert re-*
turn-prob-space)
 $(\text{auto simp: return-val-def return-count-space-eq-density } \varrho$
 $\text{density-density-eq field-simps intro!}: \text{prob-space-imp-subprob-space})$
qed

next
case (*hd-var* x V V' Γ δ t)
hence $t = \Gamma x$ **by** *auto*
interpret *density-context* V V' Γ δ **by** *fact*
from *hd-var* **have** $x \in V$ **by** *simp*
show ?*case*
proof (*rule has-parametrized-subprob-densityI*)
fix ϱ **assume** $\varrho: \varrho \in \text{space } (\text{state-measure } V' \Gamma)$
have *subprob-space* (*dens-ctxt-measure* $\mathcal{Y} \varrho \ggg (\lambda\sigma. \text{return } (\text{stock-measure } t)$
 $(\sigma x)))$
 $(\text{is subprob-space } (?M \ggg ?f))$ **using** *hd-var* ϱ
by (*intro subprob-space-bind*)
 $(\text{auto simp: return-val-def } t \text{ intro!}: \text{subprob-space-bind subprob-space-dens}$
 $\text{measurable-compose}[\text{OF measurable-dens-ctxt-measure-component}$
return-measurable])
also from *hd-var.hyps* **have** $?M \ggg ?f = ?M \ggg (\lambda\sigma. \text{return-val } (\sigma x))$
by (*intro bind-cong*) (*auto simp: return-val-def t space-dens-ctxt-measure*

state-measure-def space-PiM dest!: PiE-mem)

finally show *subprob-space* (?M \gg ($\lambda\sigma. \text{expr-sem } \sigma$ (Var x))) **by** *simp*

from *hd-var interpret dcm: subprob-space dens-ctxt-measure* $\mathcal{Y} \varrho$
by (*intro subprob-space-dens* ϱ)
let ?M1 = *dens-ctxt-measure* $\mathcal{Y} \varrho \gg$ ($\lambda\sigma. \text{expr-sem } \sigma$ (Var x))
let ?M2 = *density* (*stock-measure* t) ($\lambda v. \text{marg-dens } \mathcal{Y} x \varrho v$)
have $\forall \sigma \in \text{space}$ (*dens-ctxt-measure* $\mathcal{Y} \varrho$). *val-type* (σx) = t **using** *hd-var*
by (*auto simp: space-dens-ctxt-measure space-PiM PiE-iff*
state-measure-def intro: type-universe-type)
hence ?M1 = *dens-ctxt-measure* $\mathcal{Y} \varrho \gg$ (*return* (*stock-measure* t) \circ ($\lambda\sigma. \sigma$
x))
by (*intro bind-cong-All*) (*simp add: return-val-def*)
also have ... = *distr* (*dens-ctxt-measure* $\mathcal{Y} \varrho$) (*stock-measure* t) ($\lambda\sigma. \sigma x$)
using *dcm.subprob-not-empty hd-var*
by (*subst bind-return-distr*) (*auto intro!: measurable-dens-ctxt-measure-component*)
also have ... = ?M2 **using** *density-marg-dens-eq*[OF $\langle x \in V \rangle$]
by (*simp add: t hd-var.prem*s ϱ)
finally show ?M1 = ?M2 .
qed (*auto intro!: measurable-marg-dens' simp: hd-var t*)

next

case (*hd-let* V V' $\Gamma e1 f \delta e2 g t$)
let ?t = *the* (*expr-type* $\Gamma e1$)
let ? Γ' = *case-nat* ?t Γ **and** ? δ' = *insert-dens* V V' f δ
let ? \mathcal{Y}' = (*shift-var-set* V, *Suc*'V', ? Γ' , ? δ')
from *hd-let.prem*s **have** t1: $\Gamma \vdash e1 : ?t$ **and** t2: ? $\Gamma' \vdash e2 : t$
by (*auto simp: expr-type-Some-iff*[*symmetric*] *split: option.split-asm*)
interpret dc: *density-context* V V' $\Gamma \delta$ **by** *fact*

show ?*case unfolding* *has-parametrized-subprob-density-def*
proof (*intro ballI conjI*)
have *density-context* { } (V \cup V') Γ ($\lambda a. 1$) **by** (*rule dc.density-context-empty*)
moreover note *hd-let.prem*s
ultimately have *has-parametrized-subprob-density* (*state-measure* (V \cup V')
 Γ)
($\lambda\varrho. \text{dens-ctxt-measure}$ ({ }, V \cup V', $\Gamma, \lambda a. 1$) $\varrho \gg$ ($\lambda\sigma. \text{expr-sem}$
 $\sigma e1$))
(*stock-measure* ?t) f (**is** ?P)
by (*intro hd-let.IH*(1)) (*auto intro!: t1*)
also have ?P \longleftrightarrow *has-parametrized-subprob-density* (*state-measure* (V \cup V')
 Γ)
($\lambda\sigma. \text{expr-sem } \sigma e1$) (*stock-measure* ?t) f **using** *hd-let.prem*s
by (*intro has-parametrized-subprob-density-cong dens-ctxt-measure-empty-bind*)
(*auto simp: dens-ctxt-measure-def state-measure'-def*
intro!: measurable-expr-sem[OF t1])
finally have f: *has-parametrized-subprob-density* (*state-measure* (V \cup V') Γ)
($\lambda\varrho. \text{expr-sem } \varrho e1$) (*stock-measure* ?t) f .
have g: *has-parametrized-subprob-density* (*state-measure* (*Suc*'V') ? Γ')

```

      (λρ. dens-ctxt-measure ?Y' ρ ≫= (λσ. expr-sem σ e2)) (stock-measure
t) g
  using hd-let.premis hd-let.hyps f subset-shift-var-set
  by (intro hd-let.IH(2) t2 dc.density-context-insert)
     (auto dest: has-parametrized-subprob-densityD)

  note g[measurable]
  thus (λ(ρ, x). g (case-nat undefined ρ) x) ∈ borel-measurable (state-measure
V' Γ ⊗M stock-measure t)
  by simp

  fix ρ assume ρ: ρ ∈ space (state-measure V' Γ)
  let ?M = dens-ctxt-measure (V, V', Γ, δ) ρ and
      ?N = state-measure (shift-var-set (V ∪ V')) ?Γ'
  have M-dcm: measurable ?M = measurable (state-measure (V ∪ V') Γ)
  by (intro ext measurable-cong-sets)
     (auto simp: dens-ctxt-measure-def state-measure-def state-measure'-def)
  have M-dcm': ∧N. measurable (?M ⊗M N) = measurable (state-measure
(V ∪ V') Γ ⊗M N)
  by (intro ext measurable-cong-sets)
     (auto simp: dens-ctxt-measure-def state-measure-def state-measure'-def)
  have ?M ≫= (λσ. expr-sem σ (LetVar e1 e2)) =
    do {σ ← ?M; y ← expr-sem σ e1; return ?N (case-nat y σ)} ≫= (λσ.
expr-sem σ e2)
  (is - = bind ?R -)
  using hd-let.premis subset-shift-var-set
  apply (simp only: expr-sem.simps, intro double-bind-assoc)
  apply (rule measurable-expr-sem[OF t2], simp)
  apply (subst M-dcm, rule measurable-expr-sem[OF t1], simp)
  apply (subst M-dcm', simp)
  done
  also from t1 and hd-let.premis
  have (λσ. expr-sem σ e1) ∈
    measurable (state-measure (V ∪ V') Γ) (subprob-algebra (stock-measure
?t))
  by (intro measurable-expr-sem) auto
  hence ?R = dens-ctxt-measure ?Y' (case-nat undefined ρ) using hd-let.premis
hd-let.hyps f ρ
  by (intro dc.dens-ctxt-measure-insert) (auto simp: has-parametrized-subprob-density-def)
  also have case-nat undefined ρ ∈ space (state-measure (Suc'V') ?Γ')
  by (rule measurable-space[OF measurable-case-nat-undefined ρ])
  with g have has-subprob-density (dens-ctxt-measure ?Y' (case-nat undefined
ρ) ≫=
    (λσ. expr-sem σ e2)) (stock-measure t) (g (case-nat undefined
ρ))
  using ρ unfolding has-parametrized-subprob-density-def by auto
  finally show has-subprob-density (?M ≫= (λσ. expr-sem σ (LetVar e1 e2)))
(stock-measure t)
    (g (case-nat undefined ρ)) .

```

qed

next

case (hd-rand-det e V' V Γ δ dst t)

then have [measurable]: $\Gamma \vdash e : \text{dist-param-type } \text{dst } \text{randomfree } e \text{ free-vars } e \subseteq V'$

by auto

interpret density-context V V' Γ δ by fact

from hd-rand-det have t: $t = \text{dist-result-type } \text{dst}$ by auto

{

fix ρ assume ρ: $\rho \in \text{space } (\text{state-measure } V' \Gamma)$

let ?M = dens-ctxt-measure (V, V', Γ, δ) ρ and ?t = dist-param-type dst

have ?M $\ggg (\lambda \sigma. \text{expr-sem } \sigma (\text{Random } \text{dst } e)) =$

$?M \ggg (\lambda \sigma. \text{return-val } (\text{expr-sem-rf } \sigma e) \ggg \text{dist-measure } \text{dst})$ (is - =

?N)

using hd-rand-det by (subst expr-sem.simps, intro bind-cong refl, subst expr-sem-rf-sound)

(auto simp: dens-ctxt-measure-def state-measure'-def)

also from hd-rand-det have A: $\bigwedge \sigma. \sigma \in \text{space } ?M \implies \text{val-type } (\text{expr-sem-rf } \sigma e) = ?t$

by (intro val-type-expr-sem-rf) (auto simp: dens-ctxt-measure-def state-measure'-def)

hence ?N = ?M $\ggg (\lambda \sigma. \text{return } (\text{stock-measure } ?t) (\text{expr-sem-rf } \sigma e) \ggg \text{dist-measure } \text{dst})$

using hd-rand-det unfolding return-val-def

by (intro bind-cong) (auto simp: dens-ctxt-measure-def state-measure'-def)

also have ... = ?M $\ggg (\lambda \sigma. \text{dist-measure } \text{dst } (\text{expr-sem-rf } \sigma e))$

unfolding return-val-def

by (intro bind-cong refl bind-return, rule measurable-dist-measure)

(auto simp: type-universe-def A simp del: type-universe-type)

finally have ?M $\ggg (\lambda \sigma. \text{expr-sem } \sigma (\text{Random } \text{dst } e)) =$

$?M \ggg (\lambda \sigma. \text{dist-measure } \text{dst } (\text{expr-sem-rf } \sigma e))$.

} note A = this

have has-parametrized-subprob-density (state-measure V' Γ)

($\lambda \rho. \text{dens-ctxt-measure } \mathcal{Y} \rho \ggg (\lambda \sigma. \text{dist-measure } \text{dst } (\text{expr-sem-rf } \sigma e))$)

($\text{stock-measure } t$) ($\lambda \rho x. \text{branch-prob } \mathcal{Y} \rho * \text{dist-dens } \text{dst } (\text{expr-sem-rf } \rho e) x$)

proof (unfold has-parametrized-subprob-density-def, intro conjI ballI)

show M: ($\lambda (\rho, v). \text{branch-prob } \mathcal{Y} \rho * \text{dist-dens } \text{dst } (\text{expr-sem-rf } \rho e) v$)

$\in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t)$

by (subst t) measurable

fix ρ assume ρ: $\rho \in \text{space } (\text{state-measure } V' \Gamma)$

let ?M = dens-ctxt-measure (V, V', Γ, δ) ρ and ?t = dist-param-type dst

have ?M $\ggg (\lambda \sigma. \text{expr-sem } \sigma (\text{Random } \text{dst } e)) =$

$?M \ggg (\lambda \sigma. \text{return-val } (\text{expr-sem-rf } \sigma e) \ggg \text{dist-measure } \text{dst})$ (is - =

?N)

using *hd-rand-det* **by** (*subst expr-sem.simps, intro bind-cong refl, subst expr-sem-rf-sound*)
(auto simp: dens-ctxt-measure-def state-measure'-def)
also from *hd-rand-det* **have** $A: \bigwedge \sigma. \sigma \in \text{space } ?M \implies \text{val-type } (\text{expr-sem-rf } \sigma \ e) = ?t$
by (*intro val-type-expr-sem-rf*) (*auto simp: dens-ctxt-measure-def state-measure'-def*)
hence $?N = ?M \ggg (\lambda \sigma. \text{return } (\text{stock-measure } ?t) (\text{expr-sem-rf } \sigma \ e) \ggg \text{dist-measure } \text{dst})$
using *hd-rand-det* **unfolding** *return-val-def*
by (*intro bind-cong*) (*auto simp: dens-ctxt-measure-def state-measure'-def*)
also have $\dots = ?M \ggg (\lambda \sigma. \text{dist-measure } \text{dst } (\text{expr-sem-rf } \sigma \ e))$
unfolding *return-val-def*
by (*intro bind-cong refl bind-return, rule measurable-dist-measure*)
(auto simp: type-universe-def A simp del: type-universe-type)
also have *has-subprob-density* ($?M \ggg (\lambda \sigma. \text{dist-measure } \text{dst } (\text{expr-sem-rf } \sigma \ e))$) (*stock-measure t*)
 $(\lambda v. \int^+ \sigma. \text{dist-dens } \text{dst } (\text{expr-sem-rf } (\text{restrict } \sigma \ V') \ e) \ v \ \partial ?M)$
(is has-subprob-density ?N ?R ?f)
proof (*rule bind-has-subprob-density*)
show $\text{space } ?M \neq \{\}$ **unfolding** *dens-ctxt-measure-def state-measure'-def state-measure-def*
by (*auto simp: space-PiM PiE-eq-empty-iff*)
show $(\lambda \sigma. \text{dist-measure } \text{dst } (\text{expr-sem-rf } \sigma \ e)) \in \text{measurable } ?M$ (*subprob-algebra (stock-measure t)*)
unfolding *dens-ctxt-measure-def state-measure'-def*
by (*subst t, rule measurable-compose[OF - measurable-dist-measure], simp*)
(insert hd-rand-det, auto intro!: measurable-expr-sem-rf)
show $(\lambda(x, y). \text{dist-dens } \text{dst } (\text{expr-sem-rf } (\text{restrict } x \ V') \ e) \ y)$
 $\in \text{borel-measurable } (?M \otimes_M \text{stock-measure } t)$
unfolding *t by measurable*
fix σ **assume** $\sigma: \sigma \in \text{space } ?M$
hence $\sigma': \text{restrict } \sigma \ V' \in \text{space } (\text{state-measure } V' \ \Gamma)$
unfolding *dens-ctxt-measure-def state-measure'-def state-measure-def restrict-def*
by (*auto simp: space-PiM*)
from *hd-rand-det* **have** *restr: expr-sem-rf (restrict sigma V') e = expr-sem-rf sigma e*
by (*intro expr-sem-rf-eq-on-vars*) *auto*
from *hd-rand-det* **have** $\text{val-type } (\text{expr-sem-rf } (\text{restrict } \sigma \ V') \ e) = \text{dist-param-type } \text{dst}$
by (*auto intro!: val-type-expr-sem-rf[OF - - - sigma']*)
also note *restr*
finally have *has-density (dist-measure dst (expr-sem-rf sigma e)) (stock-measure t)*
 $(\text{dist-dens } \text{dst } (\text{expr-sem-rf } \sigma \ e))$ **using** *hd-rand-det*
by (*subst t, intro dist-measure-has-density*)
(auto intro!: val-type-expr-sem-rf simp: type-universe-def dens-ctxt-measure-def state-measure'-def simp del: type-universe-type)
thus *has-density (dist-measure dst (expr-sem-rf sigma e)) (stock-measure t)*

```

      (dist-dens dst (expr-sem-rf (restrict σ V') e)) by (simp add: restr)
    qed (insert ρ, auto intro!: subprob-space-dens)
  moreover have val-type (expr-sem-rf ρ e) = dist-param-type dst using hd-rand-det
  ρ
    by (intro val-type-expr-sem-rf) auto
  hence expr-sem-rf ρ e ∈ type-universe (dist-param-type dst)
    by (simp add: type-universe-def del: type-universe-type)
  ultimately show has-subprob-density (?M ≧≧ (λσ. dist-measure dst (expr-sem-rf
  σ e)))
    (stock-measure t) (λv. branch-prob Y ρ * dist-dens dst (expr-sem-rf
  ρ e) v)
    using hd-rand-det
    apply (rule-tac has-subprob-density-equal-on-space, simp)
    apply (intro nn-integral-dens-ctxt-measure-restrict)
    apply (simp-all add: t ρ)
    done
  qed
  with A show ?case by (subst has-parametrized-subprob-density-cong) (simp-all
  add: A)

next
  case (hd-rand V V' Γ δ e f dst t)
  let ?t = dist-param-type dst
  from hd-rand.premis have t1: Γ ⊢ e : ?t and t2: t = dist-result-type dst by auto
  interpret density-ctxt V V' Γ δ by fact
  have dens[measurable]: has-parametrized-subprob-density (state-measure V' Γ)
    (λρ. dens-ctxt-measure (V, V', Γ, δ) ρ ≧≧ (λσ. expr-sem σ e)) (stock-measure
  ?t) f
  using hd-rand.premis by (intro hd-rand.IH) auto
  show ?case
  proof (unfold has-parametrized-subprob-density-def, intro ballI conjI impI)
  interpret sigma-finite-measure stock-measure (dist-param-type dst) by simp
  show case-prod (apply-dist-to-dens dst f) ∈
    borel-measurable (state-measure V' Γ ⊗M stock-measure t)
  unfolding apply-dist-to-dens-def t2 by measurable

  fix ρ assume ρ: ρ ∈ space (state-measure V' Γ)
  let ?M = dens-ctxt-measure (V, V', Γ, δ) ρ
  have meas-M: measurable ?M = measurable (state-measure (V ∪ V') Γ)
  by (intro ext measurable-cong-sets) (auto simp: dens-ctxt-measure-def state-measure'-def)
  from hd-rand have Me: (λσ. expr-sem σ e) ∈ measurable ?M (subprob-algebra
  (stock-measure ?t))
  by (subst meas-M, intro measurable-expr-sem[OF t1]) auto
  hence ?M ≧≧ (λσ. expr-sem σ (Random dst e)) = (?M ≧≧ (λσ. expr-sem σ
  e)) ≧≧ dist-measure dst
  (is - = ?N)
  by (subst expr-sem.simps, intro bind-assoc[OF Me, symmetric])
  (insert hd-rand, auto intro!: measurable-dist-measure)
  also have space ?M ≠ {}

```

by (auto simp: dens-ctxt-measure-def state-measure'-def state-measure-def
 space-PiM PiE-eq-empty-iff)

with dens ϱ Me **have** has-subprob-density ?N (stock-measure t) (apply-dist-to-dens
 dst f ϱ)

unfolding apply-dist-to-dens-def has-parametrized-subprob-density-def
by (subst t2, intro bind-has-subprob-density')
 (auto simp: hd-rand.IH space-bind-measurable
 intro!: measurable-dist-dens dist-measure-has-subprob-density)

finally show has-subprob-density (?M \gg (lambda sigma. expr-sem sigma (Random dst e)))
 (stock-measure t)
 (apply-dist-to-dens dst f ϱ) .

qed

next

case (hd-fail V V' Γ δ t t')

interpret density-context V V' Γ δ **by** fact

have has-parametrized-subprob-density (state-measure V' Γ)
 (lambda -. null-measure (stock-measure t)) (stock-measure t') (lambda -. 0) (**is** ?P)

using hd-fail **by** (auto simp: has-parametrized-subprob-density-def
 intro!: null-measure-has-subprob-density)

also have ?P \longleftrightarrow ?case

by (intro has-parametrized-subprob-density-cong)
 (auto simp: dens-ctxt-measure-bind-const subprob-space-null-measure-iff)

finally show ?case .

next

case (hd-pair x V y V' Γ δ t)

interpret density-context V V' Γ δ **by** fact

let ?R = stock-measure t

from hd-pair.premis **have** t: t = PRODUCT (Γ x) (Γ y) **by** auto

have meas: (lambda sigma. <|sigma x, sigma y|>) in measurable (state-measure (V union V') Γ) ?R
using hd-pair **unfolding** t state-measure-def **by** simp

have has-parametrized-subprob-density (state-measure V' Γ)
 (lambda rho. distr (dens-ctxt-measure (V, V', Γ , δ) rho) ?R (lambda sigma. <|sigma x, sigma y|>))
 (stock-measure t) (marg-dens2 \mathcal{Y} x y)

proof (rule has-parametrized-subprob-densityI)

fix rho **assume** rho: rho in space (state-measure V' Γ)

let ?M = dens-ctxt-measure (V, V', Γ , δ) rho

from hd-pair.hyps rho **show** distr ?M ?R (lambda sigma. <|sigma x, sigma y|>) = density ?R
 (marg-dens2 \mathcal{Y} x y rho)

by (subst (1 2) t, rule density-marg-dens2-eq[symmetric])

from rho **interpret** subprob-space ?M **by** (rule subprob-space-dens)

show subprob-space (distr (dens-ctxt-measure (V, V', Γ , δ) rho) ?R (lambda sigma. <|sigma x, sigma
 y|>))

by (rule subprob-space-distr)

(simp-all add: meas measurable-dens-ctxt-measure-eq)

qed (auto simp: t intro!: measurable-marg-dens2' hd-pair.hyps simp del: stock-measure.simps)

also from *hd-pair.hyps*
have $(\lambda \rho. \text{distr } (\text{dens-ctxt-measure } (V, V', \Gamma, \delta) \rho) \text{ ?R } (\lambda \sigma. \langle |\sigma x, \sigma y| \rangle)) =$
 $(\lambda \rho. \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \rho \gg (\lambda \sigma. \text{return-val } \langle |\sigma x, \sigma$
 $y| \rangle))$
by (*intro ext bind-return-val[symmetric]*) (*simp-all add: meas measurable-dens-ctxt-measure-eq*)
finally show *?case by (simp only: expr-sem-pair-vars)*

next
case (*hd-if V V' Γ b f δ e1 g1 e2 g2 t*)
interpret *dc: density-context V V' Γ δ by fact*
from *hd-if.prem*s **have** *tb: Γ ⊢ b : BOOL and t1: Γ ⊢ e1 : t and t2: Γ ⊢ e2 : t*
by *auto*

have *has-parametrized-subprob-density (state-measure (V ∪ V') Γ)*
 $(\lambda \rho. \text{dens-ctxt-measure } (\{\}, V \cup V', \Gamma, \lambda a. 1) \rho \gg (\lambda \sigma. \text{expr-sem } \sigma b))$
(stock-measure BOOL) f
(is ?P) using *hd-if.prem*s *tb by (intro hd-if.IH(1)) auto*
also have *?P* \longleftrightarrow *has-parametrized-subprob-density (state-measure (V ∪ V') Γ)*
 $(\lambda \sigma. \text{expr-sem } \sigma b)$ *(stock-measure BOOL) f (is - = ?P) using*
*hd-if.prem*s
by (*intro has-parametrized-subprob-density-cong dens-ctxt-measure-empty-bind*)
(auto simp: dens-ctxt-measure-def state-measure'-def
intro!: measurable-expr-sem[OF tb])
finally have *f: ?P .*

let *?M = λ ρ. dens-ctxt-measure (V, V', Γ, δ) ρ*
let *?M' = λ b ρ. dens-ctxt-measure (V, V', Γ, if-dens δ f b) ρ*

from *f* **have** *dc': ∧ b. density-context V V' Γ (if-dens δ f b)*
by (*intro dc.density-context-if-dens*) (*simp add: stock-measure.simps*)
have *g1[measurable]: has-parametrized-subprob-density (state-measure V' Γ)*
 $(\lambda \rho. \text{?M' True } \rho \gg (\lambda \sigma. \text{expr-sem } \sigma e1))$ *(stock-measure t) g1 using*
*hd-if.prem*s
by (*intro hd-if.IH(2)[OF t1 dc'] simp*)
have *g2[measurable]: has-parametrized-subprob-density (state-measure V' Γ)*
 $(\lambda \rho. \text{?M' False } \rho \gg (\lambda \sigma. \text{expr-sem } \sigma e2))$ *(stock-measure t) g2 using*
*hd-if.prem*s
by (*intro hd-if.IH(3)[OF t2 dc'] simp*)

show *?case*
proof (*rule has-parametrized-subprob-densityI*)
show $(\lambda (\rho, x). g1 \rho x + g2 \rho x) \in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M$
stock-measure t)
by *measurable*
fix ρ **assume** $\rho \in \text{space } (\text{state-measure } V' \Gamma)$
show *subprob-space (?M ρ ≫ (λ σ. expr-sem σ (IF b THEN e1 ELSE e2)))*
using ρ *hd-if.prem*s
by (*intro subprob-space-bind[of - - stock-measure t], simp add: dc.subprob-space-dens*)
(auto intro!: measurable-expr-sem simp: measurable-dens-ctxt-measure-eq)

$\text{simp del: expr-sem.simps}$
show $?M \varrho \gg (\lambda\sigma. \text{expr-sem } \sigma \text{ (IF } b \text{ THEN } e1 \text{ ELSE } e2)) =$
 $\text{density (stock-measure } t) (\lambda x. g1 \varrho x + g2 \varrho x) \text{ using } \varrho \text{ hd-if.prem } f$
 $g1 \ g2$
by ($\text{subst expr-sem.simps, intro dc.emmeasure-bind-if-dens-ctxt-measure}$)
 $(\text{auto simp: measurable-dens-ctxt-measure-eq if-dens-def}$
 $\text{simp del: stock-measure.simps intro!: measurable-expr-sem})$
qed

next

case ($\text{hd-if-det } b \ V \ V' \ \Gamma \ \delta \ e1 \ g1 \ e2 \ g2 \ t$)
interpret $\text{dc: density-context } V \ V' \ \Gamma \ \delta$ **by fact**
from $\text{hd-if-det.prem } \langle \text{randomfree } b \rangle$
have $\text{tb}[\text{measurable (raw)}]: \Gamma \vdash b : \text{BOOL}$ **and** $[\text{measurable (raw)}]: \text{randomfree } b$
and $\text{t1}[\text{measurable (raw)}]: \Gamma \vdash e1 : t$
and $\text{t2}[\text{measurable (raw)}]: \Gamma \vdash e2 : t$
and $\text{fv-b}[\text{measurable (raw)}]: \text{free-vars } b \subseteq V \cup V'$
and $\text{fv-e1}[\text{measurable (raw)}]: \text{free-vars } e1 \subseteq V \cup V'$
and $\text{fv-e2}[\text{measurable (raw)}]: \text{free-vars } e2 \subseteq V \cup V'$ **by auto**
let $?M = \lambda\varrho. \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \ \varrho$
let $?M' = \lambda x \ \varrho. \text{dens-ctxt-measure } (V, V', \Gamma, \text{if-dens-det } \delta \ b \ x) \ \varrho$
let $?N = \lambda\varrho. ?M \varrho \gg (\lambda\sigma. \text{if expr-sem-rf } \sigma \ b = \text{BoolVal True then expr-sem } \sigma$
 $e1 \text{ else expr-sem } \sigma \ e2)$

from $\text{hd-if-det.hyps hd-if-det.prem } \text{tb}$
have $\text{dc}' : \bigwedge x. \text{density-context } V \ V' \ \Gamma \ (\text{if-dens-det } \delta \ b \ x)$
by ($\text{intro dc.density-context-if-dens-det}$) simp-all
have $g1[\text{measurable}]: \text{has-parametrized-subprob-density (state-measure } V' \ \Gamma)$
 $(\lambda\varrho. ?M' \ \text{True } \varrho \gg (\lambda\sigma. \text{expr-sem } \sigma \ e1)) \text{ (stock-measure } t) \ g1$ **using**
 hd-if-det.prem
by ($\text{intro hd-if-det.IH(1)[OF]}$) ($\text{simp-all add: dc' t1}$)
have $g2[\text{measurable}]: \text{has-parametrized-subprob-density (state-measure } V' \ \Gamma)$
 $(\lambda\varrho. ?M' \ \text{False } \varrho \gg (\lambda\sigma. \text{expr-sem } \sigma \ e2)) \text{ (stock-measure } t) \ g2$ **using**
 hd-if-det.prem
by ($\text{intro hd-if-det.IH(2)[OF]}$) ($\text{simp-all add: dc' t2}$)

note $\text{val-type-expr-sem-rf[OF tb, of } V \cup V', \text{ simp]}$

have $\text{has-parametrized-subprob-density (state-measure } V' \ \Gamma) \ ?N$
 $(\text{stock-measure } t) (\lambda a \ b. g1 \ a \ b + g2 \ a \ b) \text{ (is } ?P)$
proof ($\text{rule has-parametrized-subprob-densityI}$)
show $(\lambda(\varrho, x). g1 \ \varrho \ x + g2 \ \varrho \ x) \in \text{borel-measurable (state-measure } V' \ \Gamma \ \otimes_M$
 $\text{stock-measure } t)$
by measurable
fix ϱ **assume** $\varrho : \text{space (state-measure } V' \ \Gamma)$
show $\text{subprob-space } (?N \ \varrho)$
using $\varrho \ \text{hd-if-det.prem } \text{hd-if-det.hyps } \text{t1 } \text{t2}$
by ($\text{intro subprob-space-bind[of - - stock-measure t]}$) ($\text{auto simp add: dc.subprob-space-dens}$)
show $?N \ \varrho = \text{density (stock-measure } t) (\lambda x. g1 \ \varrho \ x + g2 \ \varrho \ x)$

```

    using  $\rho$  hd-if-det.premis g1 g2 dc' hd-if-det.premis unfolding if-dens-det-def
    by (intro dc.emmeasure-bind-if-det-dens-ctxt-measure)
      (simp-all add: space-dens-ctxt-measure)
qed
also from hd-if-det.premis hd-if-det.hyps have ?P  $\longleftrightarrow$  ?case
  apply (intro has-parametrized-subprob-density-cong bind-cong refl)
  apply (subst expr-sem.simps)
  apply (subst expr-sem-rf-sound[OF tb, of  $V \cup V'$ , symmetric]) []
  apply (simp-all add: space-dens-ctxt-measure bind-return-val''[where  $M = \text{stock-measure}$ 
t])
  done
  finally show ?case .

next
case (hd-fst  $V V' \Gamma \delta e f t$ )
interpret density-context  $V V' \Gamma \delta$  by fact
from hd-fst.premis obtain  $t'$  where  $t: \Gamma \vdash e : \text{PRODUCT } t t'$ 
  by (elim expr-typing-opE) (auto split: pdf-type.split-asm)
hence  $\Gamma \vdash \text{Snd } \$\$ e : t'$  by (intro expr-typing.intros) auto
hence  $t2$ : the (expr-type  $\Gamma (\text{Snd } \$\$ e) = t'$  by (simp add: expr-type-Some-iff[symmetric])
let ?N = stock-measure (PRODUCT  $t t'$ )
have dens[measurable]: has-parametrized-subprob-density (state-measure  $V' \Gamma$ )
  ( $\lambda \rho$ . dens-ctxt-measure ( $V, V', \Gamma, \delta$ )  $\rho \ggg (\lambda \sigma$ . expr-sem  $\sigma e$ ) ?N f
by (intro hd-fst.IH) (insert hd-fst.premis hd-fst.hyps  $t$ , auto)

let ?f =  $\lambda \rho x. \int^+ y. f \rho \langle x, y \rangle \partial \text{stock-measure } t'$ 
have has-parametrized-subprob-density (state-measure  $V' \Gamma$ )
  ( $\lambda \rho$ . dens-ctxt-measure ( $V, V', \Gamma, \delta$ )  $\rho \ggg (\lambda \sigma$ . expr-sem  $\sigma (Fst \$\$ e)$ )
(stock-measure  $t$ ) ?f
  unfolding has-parametrized-subprob-density-def
proof (intro conjI ballI impI)
  interpret sigma-finite-measure stock-measure  $t'$  by simp
  show case-prod ?f  $\in$  borel-measurable (state-measure  $V' \Gamma \otimes_M$  stock-measure
t)
    by measurable

fix  $\rho$  assume  $\rho: \rho \in \text{space (state-measure } V' \Gamma)$ 
let ?M = dens-ctxt-measure ( $V, V', \Gamma, \delta$ )  $\rho$ 
from dens and  $\rho$  have has-subprob-density (?M  $\ggg (\lambda \sigma$ . expr-sem  $\sigma e)$ ) ?N
(f  $\rho$ )
  unfolding has-parametrized-subprob-density-def by auto
hence has-subprob-density (distr (?M  $\ggg (\lambda \sigma$ . expr-sem  $\sigma e)$ ) (stock-measure
t) (op-sem Fst))
  (stock-measure  $t$ ) (?f  $\rho$ ) (is has-subprob-density ?R -)
  by (intro has-subprob-density-distr-Fst) simp
also from hd-fst.premis and  $\rho$  have ?R = ?M  $\ggg (\lambda \sigma$ . expr-sem  $\sigma (Fst \$\$ e)$ )
by (intro expr-sem-op-eq-distr[symmetric]) simp-all
  finally show has-subprob-density ... (stock-measure  $t$ ) (?f  $\rho$ ) .
qed

```

thus $?case$ **by** (*subst t2*)

next

case (*hd-snd V V' Γ δ e f t'*)

interpret *density-context V V' Γ δ* **by** *fact*

from *hd-snd.prem*s **obtain** *t* **where** $t: \Gamma \vdash e : PRODUCT\ t\ t'$

by (*elim expr-typing-opE*) (*auto split: pdf-type.split-asm*)

hence $\Gamma \vdash Fst\ \$\$ e : t$ **by** (*intro expr-typing.intros*) *auto*

hence *t2*: *the* (*expr-type* Γ (*Fst* $\$ \$ e$)) = *t* **by** (*simp add: expr-type-Some-iff[symmetric]*)

let $?N = stock\ measure\ (PRODUCT\ t\ t')$

have *dens*[*measurable*]: *has-parametrized-subprob-density* (*state-measure V' Γ*)

$(\lambda \varrho. dens\ ctxt\ measure\ (V, V', \Gamma, \delta)\ \varrho \gg (\lambda \sigma. expr\ sem\ \sigma\ e))\ ?N\ f$

by (*intro hd-snd.IH*) (*insert hd-snd.prem*s *hd-snd.hyps t, auto*)

let $?f = \lambda \varrho\ y. \int^+ x. f\ \varrho\ <|x,y|>\ \partial stock\ measure\ t$

have *has-parametrized-subprob-density* (*state-measure V' Γ*)

$(\lambda \varrho. dens\ ctxt\ measure\ (V, V', \Gamma, \delta)\ \varrho \gg (\lambda \sigma. expr\ sem\ \sigma\ (Snd\ \$ \$ e)))$

(stock-measure t') $?f$

unfolding *has-parametrized-subprob-density-def*

proof (*intro conjI ballI impI*)

interpret *sigma-finite-measure stock-measure t* **by** *simp*

show *case-prod* $?f \in borel\ measurable\ (state\ measure\ V'\ \Gamma \otimes_M stock\ measure\ t')$

by *measurable*

fix ϱ **assume** $\varrho: \varrho \in space\ (state\ measure\ V'\ \Gamma)$

let $?M = dens\ ctxt\ measure\ (V, V', \Gamma, \delta)\ \varrho$

from *dens* **and** ϱ **have** *has-subprob-density* ($?M \gg (\lambda \sigma. expr\ sem\ \sigma\ e)$) $?N$

(f ϱ)

unfolding *has-parametrized-subprob-density-def* **by** *auto*

hence *has-subprob-density* (*distr* ($?M \gg (\lambda \sigma. expr\ sem\ \sigma\ e)$) (*stock-measure t'*) (*op-sem Snd*))

$(stock\ measure\ t')\ (?f\ \varrho)$ (**is** *has-subprob-density* $?R$ - -)

by (*intro has-subprob-density-distr-Snd*) *simp*

also **from** *hd-snd.prem*s **and** ϱ **have** $?R = ?M \gg (\lambda \sigma. expr\ sem\ \sigma\ (Snd\ \$ \$ e))$

e)

by (*intro expr-sem-op-eq-distr[symmetric]*) *simp-all*

finally **show** *has-subprob-density ... (stock-measure t')* ($?f\ \varrho$) .

qed

thus $?case$ **by** (*subst t2*)

next

case (*hd-op-discr Γ oper e V V' δ f t'*)

interpret *density-context V V' Γ δ* **by** *fact*

from *hd-op-discr.prem*s **obtain** *t* **where** $t1: \Gamma \vdash e : t$ **and** $t2: op\ type\ oper\ t = Some\ t'$ **by** *auto*

have *dens*[*measurable*]: *has-parametrized-subprob-density* (*state-measure V' Γ*)

$(\lambda \varrho. dens\ ctxt\ measure\ (V, V', \Gamma, \delta)\ \varrho \gg (\lambda \sigma. expr\ sem\ \sigma\ e))$

(stock-measure t) *f*

by (*intro hd-op-discr.IH*) (*insert hd-op-discr.premis hd-op-discr.hyps t1, auto*)
from *hd-op-discr t1* **have** *expr-type* Γ $e = \text{Some } t$ **and** *expr-type* Γ (*oper* $\$ \$ e$)
 $= \text{Some } t'$
by (*simp-all add: expr-type-Some-iff[symmetric]*)
hence $t1'$: *the* (*expr-type* Γe) $= t$ **and** $t2'$: *the* (*expr-type* Γ (*oper* $\$ \$ e$)) $= t'$
by *auto*
with *hd-op-discr* **have** *countable: countable-type* t' **by** *simp*

have A : *has-parametrized-subprob-density* (*state-measure* $V' \Gamma$)
 $(\lambda \varrho. \text{distr } (\text{dens-ctxt-measure } (V, V', \Gamma, \delta)) \varrho \gg (\lambda \sigma. \text{expr-sem } \sigma e))$
 $(\text{stock-measure } t') (\text{op-sem } \text{oper}))$
 $(\lambda a b. \int^+ x. (\text{if } \text{op-sem } \text{oper } x = b \text{ then } 1 \text{ else } 0) * f a x \partial \text{stock-measure } t)$

proof (*intro has-parametrized-subprob-densityI*)
let $?f = \lambda \varrho y. \int^+ x. (\text{if } \text{op-sem } \text{oper } x = y \text{ then } 1 \text{ else } 0) * f \varrho x \partial \text{stock-measure } t$

note *sigma-finite-measure.borel-measurable-nn-integral[OF sigma-finite-stock-measure, measurable]*
show *case-prod* $?f \in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t')$

using *measurable-op-sem[OF t2]* **by** *measurable*

fix ϱ **assume** $\varrho: \varrho \in \text{space } (\text{state-measure } V' \Gamma)$
let $?M = \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho$
let $?N = ?M \gg (\lambda \sigma. \text{expr-sem } \sigma e)$

from *dens* **and** ϱ **have** dens' : *has-subprob-density* $?N$ (*stock-measure* t) ($f \varrho$)
unfolding *has-parametrized-subprob-density-def* **by** *auto*
from *hd-op-discr.premis t1*
have $M\text{-e}$: $(\lambda \sigma. \text{expr-sem } \sigma e) \in \text{measurable } ?M$ (*subprob-algebra* (*stock-measure* t))

by (*auto simp: measurable-dens-ctxt-measure-eq intro!: measurable-expr-sem*)
from $M\text{-e}$ **have** meas-N : *measurable* $?N = \text{measurable } (\text{stock-measure } t)$
by (*intro ext measurable-cong-sets*) (*simp-all add: sets-bind-measurable*)
from dens' **and** $t2$ **show** *subprob-space* (*distr* $?N$ (*stock-measure* t') (*op-sem* *oper*))

by (*intro subprob-space.subprob-space-distr*)
 $(\text{auto dest: has-subprob-densityD intro!: measurable-op-sem simp: meas-N})$

from *countable* **have** *count-space: stock-measure* $t' = \text{count-space } (\text{type-universe } t')$

by (*rule countable-type-imp-count-space*)
from dens' **have** $?N = \text{density } (\text{stock-measure } t) (f \varrho)$ **by** (*rule has-subprob-densityD*)
also {
fix $x \in \text{type-universe } t$
with $M\text{-e}$ **have** *val-type* $x = t$ **by** (*auto simp:*)
hence *val-type* (*op-sem* *oper* x) $= t'$ **by** (*intro op-sem-val-type*) (*simp add: t2*)

```

} note op-sem-type-universe = this
from countable countable-type-countable measurable-op-sem[OF t2] dens'
have distr ... (stock-measure t') (op-sem oper) = density (stock-measure t') (?f
ρ)
  by (subst count-space, subst distr-density-discrete)
  (auto simp: meas-N count-space intro!: op-sem-type-universe dest: has-subprob-densityD)
finally show distr ?N (stock-measure t') (op-sem oper) = density (stock-measure
t') (?f ρ) .
qed
from hd-op-discr.prems
  have B:  $\bigwedge \rho. \rho \in \text{space (state-measure } V' \Gamma) \implies$ 
distr (dens-ctxt-measure (V, V',  $\Gamma, \delta$ )  $\rho$ )  $\ggg$  ( $\lambda \sigma. \text{expr-sem } \sigma e$ )
(stock-measure t') (op-sem oper) =
dens-ctxt-measure (V, V',  $\Gamma, \delta$ )  $\rho$   $\ggg$  ( $\lambda \sigma. \text{expr-sem } \sigma (oper \ \$\$ e)$ )
  by (intro expr-sem-op-eq-distr[symmetric] simp-all)
show ?case by (simp only: has-parametrized-subprob-density-cong[OF B[symmetric]]
t1' A)

next
case (hd-neg V V'  $\Gamma \delta e f t'$ )
from hd-neg.prems obtain t where t1:  $\Gamma \vdash e : t$  and t2: op-type Minus t =
Some t' by auto
  have dens: has-parametrized-subprob-density (state-measure V'  $\Gamma$ )
( $\lambda \rho. \text{dens-ctxt-measure (V, V',  $\Gamma, \delta$ )  $\rho$   $\ggg$  ( $\lambda \sigma. \text{expr-sem } \sigma e$ )$ 
(stock-measure t) f
  by (intro hd-neg.IH (insert hd-neg.prems hd-neg.hyps t1, auto))
  with hd-neg and t1 and t2 show ?case
  proof (intro expr-has-density-sound-op[where t = t])
    from t2 have [measurable]: lift-RealIntVal uminus uminus  $\in$  measurable (stock-measure
t') (stock-measure t)
    by (simp split: pdf-type.split-asm)
    from dens have [measurable]: case-prod f  $\in$  borel-measurable (state-measure
V'  $\Gamma$   $\otimes_M$  stock-measure t)
    by (blast dest: has-parametrized-subprob-densityD)
    show ( $\lambda (\rho, x). f \rho (op\text{-sem } \text{Minus } x)$ )
       $\in$  borel-measurable (state-measure V'  $\Gamma$   $\otimes_M$  stock-measure t') by simp

  fix M  $\rho$  assume dens': has-subprob-density M (stock-measure t) (f  $\rho$ )
hence space-M: space M = space (stock-measure t) by (auto dest: has-subprob-densityD)
from t2 have t-disj: (t = REAL  $\wedge$  t' = REAL)  $\vee$  (t = INTEG  $\wedge$  t' = INTEG)
  by (auto split: pdf-type.split-asm)
  thus has-density (distr M (stock-measure t') (op-sem Minus))
(stock-measure t') ( $\lambda x. f \rho (op\text{-sem } \text{Minus } x)$ ) (is ?thesis)
proof (elim disjE conjE)
  assume A: t = REAL t' = REAL
  have has-density (distr M (stock-measure t') (lift-RealVal uminus)) (stock-measure
t')
    ( $\lambda x. f \rho (RealVal (-x))$ )  $\circ$  extract-real using dens'
  by (simp only: A, intro distr-lift-RealVal)

```

```

      (auto intro!: distr-uminus-real dest: has-subprob-density-imp-has-density)
    also have distr M (stock-measure t') (lift-RealVal uminus) =
      distr M (stock-measure t') (lift-RealIntVal uminus uminus) using
dens'
      by (intro distr-cong) (auto intro!: lift-RealIntVal-Real[symmetric] simp:
space-M A)
      also have has-density ... (stock-measure t') ((λx. f ρ (RealVal (-x))) ∘
extract-real) ↔
      has-density ... (stock-measure t') (λx. f ρ (lift-RealIntVal uminus
uminus x))
      by (intro has-density-cong)
      (auto simp: lift-RealIntVal-def extract-real-def A space-embed-measure split:
val.split)
      finally show ?thesis by simp
    next
      assume A: t = INTEG t' = INTEG
      have has-density (distr M (stock-measure t') (lift-IntVal uminus)) (stock-measure
t')
      ((λx. f ρ (IntVal (-x))) ∘ extract-int) using dens'
      by (simp only: A, intro distr-lift-IntVal)
      (auto intro!: distr-uminus-ring-count-space simp: has-subprob-density-def)
      also have distr M (stock-measure t') (lift-IntVal uminus) =
      distr M (stock-measure t') (lift-RealIntVal uminus uminus) using
dens'
      by (intro distr-cong) (auto intro!: lift-RealIntVal-Int[symmetric] simp: space-M
A)
      also have has-density ... (stock-measure t') ((λx. f ρ (IntVal (-x))) ∘ ex-
tract-int) ↔
      has-density ... (stock-measure t') (λx. f ρ (lift-RealIntVal uminus
uminus x))
      by (intro has-density-cong)
      (auto simp: lift-RealIntVal-def extract-int-def A space-embed-measure split:
val.split)
      finally show ?thesis by simp
    qed
  qed auto

next
  case (hd-exp V V' Γ δ e f t')
  from hd-exp.prem have t1: Γ ⊢ e : REAL and t2: t' = REAL
  by (auto split: pdf-type.split-asm)
  have dens[measurable]: has-parametrized-subprob-density (state-measure V' Γ)
(λρ. dens-ctxt-measure (V, V', Γ, δ) ρ ≫ (λσ. expr-sem σ e))
(stock-measure REAL) f
  by (intro hd-exp.IH) (insert hd-exp.prem hd-exp.hyps t1, auto)
  with hd-exp and t1 and t2 show ?case
  proof (intro expr-has-density-sound-op[where t = REAL])
    from t2 have [measurable]: lift-RealVal safe-ln ∈ measurable (stock-measure
REAL) (stock-measure REAL)

```

```

    by (simp split: pdf-type.split-asm)
  from dens have Mf[measurable]: case-prod f ∈ borel-measurable (state-measure
V' Γ ⊗M stock-measure REAL)
    by (blast dest: has-parametrized-subprob-densityD)
  let ?f = λϱ x. if extract-real x > 0 then
    f ϱ (lift-RealVal safe-ln x) * inverse (extract-real x) else 0
  show case-prod ?f ∈ borel-measurable (state-measure V' Γ ⊗M stock-measure
t')
    unfolding t2 by measurable
  fix M ϱ assume dens': has-subprob-density M (stock-measure REAL) (f ϱ)
  hence space-M: space M = space (stock-measure REAL) by (auto dest: has-subprob-densityD)
  have has-density (distr M (stock-measure t') (lift-RealVal exp)) (stock-measure
t')
    ((λx. if 0 < x then f ϱ (RealVal (ln x)) * ennreal (inverse x) else 0)
    ◦ extract-real) (is has-density - - ?f') using dens'
  apply (simp only: t2)
  apply (rule distr-lift-RealVal[where g = λf x. if x > 0 then f (ln x) * ennreal
(inverse x) else 0])
  apply (auto intro!: subprob-density-distr-real-exp intro: has-subprob-density-imp-has-density)
  done
  also have ?f' = ?f ϱ
    by (intro ext) (simp add: o-def lift-RealVal-def extract-real-def split: val.split)
  finally show has-density (distr M (stock-measure t') (op-sem Exp)) (stock-measure
t') ...
    by simp
qed auto

next
case (hd-inv V V' Γ δ e f t')
from hd-inv.prem have t1: Γ ⊢ e : REAL and t2: t' = REAL
  by (auto split: pdf-type.split-asm)
have dens: has-parametrized-subprob-density (state-measure V' Γ)
  (λϱ. dens-ctxt-measure (V, V', Γ, δ) ϱ ≫ (λσ. expr-sem σ e))
(stock-measure REAL) f
  by (intro hd-inv.IH) (insert hd-inv.prem hd-inv.hyps t1, auto)
with hd-inv and t1 and t2 show ?case
  proof (intro expr-has-density-sound-op[where t = REAL])
    from t2 have [measurable]: lift-RealVal inverse ∈ measurable (stock-measure
REAL) (stock-measure REAL)
      by (simp split: pdf-type.split-asm)
    from dens have Mf[measurable]: case-prod f ∈ borel-measurable (state-measure
V' Γ ⊗M stock-measure REAL)
      by (blast dest: has-parametrized-subprob-densityD)
    let ?f = λϱ x. f ϱ (op-sem Inverse x) * inverse (extract-real x) ^ 2
    have [measurable]: extract-real ∈ borel-measurable (stock-measure REAL) by
simp
    show case-prod ?f ∈ borel-measurable (state-measure V' Γ ⊗M stock-measure
t') by (simp add: t2)
    fix M ϱ assume dens': has-subprob-density M (stock-measure REAL) (f ϱ)

```

hence *space-M*: $\text{space } M = \text{space } (\text{stock-measure } REAL)$ **by** (*auto dest: has-subprob-densityD*)
have *has-density* (*distr M (stock-measure t') (lift-RealVal inverse)*) (*stock-measure t'*)
 $((\lambda x. f \varrho (\text{RealVal } (\text{inverse } x)) * \text{ennreal } (\text{inverse } (x * x)))$
 $\circ \text{extract-real})$ (**is** *has-density - - ?f'*) **using** *dens'*
apply (*simp only: t2*)
apply (*rule distr-lift-RealVal*)
apply (*auto intro!: subprob-density-distr-real-inverse intro: has-subprob-density-imp-has-density simp del: inverse-mult-distrib*)
done
also have $?f' = ?f \varrho$
by (*intro ext*) (*simp add: o-def lift-RealVal-def extract-real-def power2-eq-square split: val.split*)
finally show *has-density* (*distr M (stock-measure t') (op-sem Inverse)*) (*stock-measure t'*) ...
by *simp*
qed *auto*

next

case (*hd-addc V V' Γ δ e f e' t*)
interpret *density-context V V' Γ δ* **by** *fact*
from *hd-addc.prem*s **have** $t1: \Gamma \vdash e : t$ **and** $t2: \Gamma \vdash e' : t$ **and** *t-disj: t = REAL*
 $\vee t = INTEG$
by (*elim expr-typing-opE, (auto split: pdf-type.split-asm)[]*) +
hence $t4: \text{op-type Add } (PRODUCT\ t\ t) = \text{Some } t$ **by** *auto*
have *dens: has-parametrized-subprob-density (state-measure V' Γ)*
 $(\lambda \varrho. \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho \ggg (\lambda \sigma. \text{expr-sem } \sigma\ e))$
(*stock-measure t*) *f*
by (*rule hd-addc.IH*) (*insert hd-addc.prem*s *t1, auto*)
show $?case$ (**is** *has-parametrized-subprob-density - ?N - ?f*)
proof (*unfold has-parametrized-subprob-density-def has-subprob-density-def, intro conjI ballI*)
from $t2$ *t-disj* *hd-addc.prem*s *hd-addc.hyps*
show *case-prod ?f \in borel-measurable (state-measure V' Γ \otimes_M stock-measure t)*
by (*intro addc-density-measurable has-parametrized-subprob-densityD[OF dens]*) *auto*

fix ϱ **assume** $\varrho \in \text{space } (\text{state-measure } V' \Gamma)$
let $?M = \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \varrho$
let $?v1 = \text{extract-int } (\text{expr-sem-rf } \varrho\ e')$ **and** $?v2 = \text{extract-real } (\text{expr-sem-rf } \varrho\ e')$
from *dens* **and** ϱ **have** *dens': has-subprob-density (?M \ggg ($\lambda \sigma. \text{expr-sem } \sigma\ e$)) (stock-measure t) (f ϱ)*
unfolding *has-parametrized-subprob-density-def has-subprob-density-def* **by** *auto*

have $Me: (\lambda \sigma. \text{expr-sem } \sigma\ e) \in$

```

    measurable (state-measure (V ∪ V') Γ) (subprob-algebra (stock-measure
t))
  using t1 hd-addc.premis by (intro measurable-expr-sem) simp-all
  from hd-addc.premis hd-addc.hyps ρ have vt-e': val-type (expr-sem-rf ρ e') = t
  by (intro val-type-expr-sem-rf[OF t2]) auto
  have space-e: space (?M ≫ (λσ. expr-sem σ e)) = type-universe t
  by (subst space-bind-measurable, subst measurable-dens-ctxt-measure-eq)
    (rule Me, simp, simp add:)
  from hd-addc.premis show subprob-space (?N ρ)
  by (intro subprob-space-bind subprob-space-dens[OF ρ],
    subst measurable-dens-ctxt-measure-eq)
    (rule measurable-expr-sem, auto)

  let ?N' = distr (?M ≫ (λσ. expr-sem σ e)) (stock-measure t)
    (lift-RealIntVal ((+) ?v1) ((+) ?v2))
  have has-density ?N' (stock-measure t) (?f ρ) using t-disj
  proof (elim disjE)
    assume t: t = REAL
    let ?N'' = distr (?M ≫ (λσ. expr-sem σ e)) (stock-measure t) (lift-RealVal
((+) ?v2))
    let ?f' = (λx. f ρ (RealVal (x - ?v2))) ∘ extract-real
    from dens' have has-density ?N'' (stock-measure t) ?f'
    by (subst (1 2) t, intro distr-lift-RealVal)
      (auto simp: t intro!: distr-plus-real dest: has-subprob-density-imp-has-density)
    also have ?N'' = ?N'
    by (intro distr-cong)
      (auto simp: lift-RealVal-def lift-RealIntVal-def extract-real-def vt-e' space-e
t
dest: split: val.split)
    also have has-density ?N' (stock-measure t) ?f' = has-density ?N'' (stock-measure
t) (?f ρ)
    using vt-e' by (intro has-density-cong)
      (auto simp: lift-RealIntVal-def t extract-real-def space-embed-measure
lift-RealIntVal2-def split: val.split)
    finally show has-density ?N' (stock-measure t) (?f ρ) .
  next
  assume t: t = INTEG
  let ?N'' = distr (?M ≫ (λσ. expr-sem σ e)) (stock-measure t) (lift-IntVal
((+) ?v1))
  let ?f' = (λx. f ρ (IntVal (x - ?v1))) ∘ extract-int
  from dens' have has-density ?N'' (stock-measure t) ?f'
  by (subst (1 2) t, intro distr-lift-IntVal)
    (auto simp: t intro!: distr-plus-ring-count-space dest: has-subprob-density-imp-has-density)
  also have ?N'' = ?N'
  by (intro distr-cong)
    (auto simp: lift-IntVal-def lift-RealIntVal-def extract-real-def vt-e' space-e t
split: val.split)
  also have has-density ?N' (stock-measure t) ?f' = has-density ?N'' (stock-measure
t) (?f ρ)

```

```

using  $vt-e'$  by (intro has-density-cong)
  (auto simp: lift-RealIntVal-def t extract-int-def space-embed-measure
   lift-RealIntVal2-def split: val.split)
finally show has-density ?N' (stock-measure t) (?f  $\rho$ ) .
qed
also have  $?N' = \text{distr } (?M \gg (\lambda\sigma. \text{expr-sem } \sigma e)) (\text{stock-measure } t)$ 
  ( $\lambda w. \text{op-sem Add } \langle |w, \text{expr-sem-rf } \rho e' | \rangle$ ) using t-disj vt-e'
by (intro distr-cong, simp, simp)
  (auto split: val.split simp: lift-RealIntVal-def
   lift-RealIntVal2-def space-e extract-real-def extract-int-def)
also have  $\dots = ?N \rho$ 
  using hd-addc.premis hd-addc.hyps t-disj  $\rho$ 
  by (intro bin-op-randomfree-restructure[OF t1 t2, symmetric]) auto
finally show has-density (?N  $\rho$ ) (stock-measure t) (?f  $\rho$ ) .
qed

next

case (hd-multc V V'  $\Gamma$   $\delta$  e f c t)
interpret density-context V V'  $\Gamma$   $\delta$  by fact
from hd-multc.premis hd-multc.hyps
  have  $t1: \Gamma \vdash e : \text{REAL}$  and  $t2: \text{val-type } c = \text{REAL}$  and  $t: t = \text{REAL}$ 
  by (elim expr-typing-opE expr-typing-valE expr-typing-pairE,
   (auto split: pdf-type.split-asm) [])+
  have  $t4: \text{op-type Mult } (\text{PRODUCT REAL REAL}) = \text{Some REAL}$  by simp
  have  $\text{dens[measurable]: has-parametrized-subprob-density (state-measure } V' \Gamma)$ 
  ( $\lambda\rho. \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \rho \gg (\lambda\sigma. \text{expr-sem } \sigma e)$ )
(stock-measure t) f
  by (rule hd-multc.IH) (insert hd-multc.premis t1 t, auto)
  show  $?case$  (is has-parametrized-subprob-density - ?N - ?f)
  proof (unfold has-parametrized-subprob-density-def has-subprob-density-def, intro
conjI ballI)
  let  $?MR = \text{stock-measure } t$  and  $?MP = \text{stock-measure } (\text{PRODUCT } t t)$ 
  have  $M\text{-mult[measurable]: (op-sem Mult)} \in \text{measurable } ?MP ?MR$  by (simp
add: measurable-op-sem t)
  show  $\text{case-prod } ?f \in \text{borel-measurable } (\text{state-measure } V' \Gamma \otimes_M \text{stock-measure } t)$ 
  by measurable (insert t2, auto simp: t val-type-eq-REAL lift-RealVal-def)

  fix  $\rho$  assume  $\rho: \rho \in \text{space } (\text{state-measure } V' \Gamma)$ 
  let  $?M = \text{dens-ctxt-measure } (V, V', \Gamma, \delta) \rho$ 
  from  $\text{dens}$  and  $\rho$  have  $\text{dens': has-subprob-density } (?M \gg (\lambda\sigma. \text{expr-sem } \sigma e))$ 
  (stock-measure t) (f  $\rho$ )
  unfolding has-parametrized-subprob-density-def has-subprob-density-def by
auto

  have  $Me: (\lambda\sigma. \text{expr-sem } \sigma e) \in$ 
  measurable (state-measure (V  $\cup$  V')  $\Gamma$ ) (subprob-algebra (stock-measure
REAL))

```

```

using t1 hd-multc.premis by (intro measurable-expr-sem) simp-all
have space-e: space (?M ≫ (λσ. expr-sem σ e)) = range RealVal
by (subst space-bind-measurable, subst measurable-dens-ctxt-measure-eq)
    (rule Me, simp, simp add: t space-embed-measure type-universe-REAL)
from hd-multc.premis show subprob-space (?N ρ)
by (intro subprob-space-bind subprob-space-dens[OF ρ],
    subst measurable-dens-ctxt-measure-eq)
    (rule measurable-expr-sem, auto)

let ?N' = distr (?M ≫ (λσ. expr-sem σ e)) (stock-measure t)
    (lift-RealVal ((* (extract-real c)))
let ?g = λf x. f (x / extract-real c) * ennreal (inverse (abs (extract-real c)))
let ?f' = (λx. f ρ (RealVal (x / extract-real c)) *
    inverse (abs (extract-real c))) ∘ extract-real
from hd-multc.hyps have extract-real c ≠ 0
by (auto simp: extract-real-def split: val.split)
with dens' have has-density ?N' (stock-measure REAL) ?f'
by (subst t, intro distr-lift-RealVal[where g = ?g])
    (auto simp: t intro!: distr-mult-real dest: has-subprob-density-imp-has-density)
also have has-density ?N' (stock-measure REAL) ?f' =
    has-density ?N' (stock-measure REAL) (?f ρ)
using hd-multc.hyps
by (intro has-density-cong)
    (auto simp: lift-RealVal-def t extract-real-def space-embed-measure
    lift-RealIntVal2-def field-simps split: val.split)
finally have has-density ?N' (stock-measure REAL) (?f ρ) .
also have ?N' = distr (?M ≫ (λσ. expr-sem σ e)) (stock-measure t)
    (λw. op-sem Mult <|w, expr-sem-rf ρ (Val c)|>) using
hd-multc.hyps
by (intro distr-cong, simp, simp)
    (auto simp: lift-RealVal-def lift-RealIntVal2-def space-e extract-real-def
    split: val.split)
also have ... = ?N ρ
using hd-multc.premis hd-multc.hyps ρ
by (intro bin-op-randomfree-restructure[OF t1, symmetric])
    (auto simp: t intro!: expr-typing.intros)
finally show has-density (?N ρ) (stock-measure t) (?f ρ) by (simp only: t)
qed

next

case (hd-add V V' Γ δ e f t)
interpret density-context V V' Γ δ by fact
from hd-add.premis hd-add.hyps
have t1: Γ ⊢ e : PRODUCT t t and t2: op-type Add (PRODUCT t t) = Some
t and
    t-disj: t = REAL ∨ t = INTEG
by (elim expr-typing-opE expr-typing-valE expr-typing-pairE,
    (auto split: pdf-type.split-asm) [])+

```

```

let ?tp = PRODUCT t t
have dens[measurable]: has-parametrized-subprob-density (state-measure V' Γ)
  (λρ. dens-ctxt-measure (V, V', Γ, δ) ρ ≫= (λσ. expr-sem σ e))
(stock-measure ?tp) f
  by (rule hd-add.IH) (insert hd-add.prem1 t1 t2 t-disj, auto)
from hd-add.prem1 hd-add.hyps t1 t2 t-disj show ?case (is has-parametrized-subprob-density
- ?N - ?f)
proof (intro expr-has-density-sound-op[OF - dens])
  note sigma-finite-measure.borel-measurable-nn-integral[OF sigma-finite-stock-measure,
measurable]
  have [measurable]: op-type Minus t = Some t
    using t-disj by auto
  note measurable-op-sem[measurable] t2[measurable]
  let ?f' = λρ z. ∫+ x. f ρ <|x, op-sem Add <|z, op-sem Minus x|>>
∂stock-measure t
  have case-prod ?f' ∈ borel-measurable (state-measure V' Γ ⊗M stock-measure
t)
    by measurable
  also have case-prod ?f' ∈ borel-measurable (state-measure V' Γ ⊗M stock-measure
t) ↔
case-prod ?f ∈ borel-measurable (state-measure V' Γ ⊗M
stock-measure t)
    by (intro measurable-cong) (auto simp: space-pair-measure)
  finally show case-prod ?f ∈ borel-measurable (state-measure V' Γ ⊗M stock-measure
t) .

fix M ρ assume dens': has-subprob-density M (stock-measure (PRODUCT t
t)) (f ρ)
  hence Mf[measurable]: f ρ ∈ borel-measurable (stock-measure (PRODUCT t
t)) by (rule has-subprob-densityD)
  let ?M = dens-ctxt-measure (V, V', Γ, δ) ρ
  show has-density (distr M (stock-measure t) (op-sem Add)) (stock-measure t)
(?f ρ)
proof (insert t-disj, elim disjE)
  assume t: t = REAL
  let ?f'' = (λz. ∫+ x. f ρ (RealPairVal (x, z - x)) ∂lborel) ∘ extract-real
  have has-density (distr M (stock-measure t) (op-sem Add)) (stock-measure t)
?f''
    using dens'
    by (simp only: t op-sem.simps, intro distr-lift-RealPairVal)
      (simp-all add: borel-prod[symmetric] has-subprob-density-imp-has-density
distr-convolution-real)
  also have ?f'' = (λz. ∫+ x. f ρ (RealPairVal (x, extract-real z - x)) ∂lborel)
(is - = ?f'')
    by (auto simp add: t space-embed-measure extract-real-def)
  also have ∧z. val-type z = REAL ⇒
(λx. f ρ <|x, op-sem Add <|z, op-sem Minus x|>>) ∈ borel-measurable
(stock-measure REAL)
    by (rule Mf[THEN measurable-compose-rev]) (simp add: t)

```

hence $\text{has-density (distr } M \text{ (stock-measure } t) \text{ (op-sem Add)) (stock-measure } t) \text{ ?}f'' \longleftrightarrow$
 $\text{has-density (distr } M \text{ (stock-measure } t) \text{ (op-sem Add)) (stock-measure } t)$
 $(\text{?}f \varrho)$
by (*intro has-density-cong, simp add: t space-embed-measure del: op-sem.simps*)
(auto simp add: nn-integral-RealVal RealPairVal-def lift-RealIntVal2-def lift-RealIntVal-def val-type-eq-REAL)
finally show
next
assume $t = \text{INTEG}$
let $\text{?}f'' = (\lambda z. \int^+ x. f \varrho (\text{IntPairVal } (x, z - x)) \partial \text{count-space UNIV}) \circ$
 extract-int
have $\text{has-density (distr } M \text{ (stock-measure } t) \text{ (op-sem Add)) (stock-measure } t)$
 $\text{?}f''$
using *dens'*
by (*simp only: t op-sem.simps, intro distr-lift-IntPairVal*)
(simp-all add: has-subprob-density-imp-has-density
distr-convolution-ring-count-space)
also have $\text{?}f'' = (\lambda z. \int^+ x. f \varrho (\text{IntPairVal } (x, \text{extract-int } z - x)) \partial \text{count-space}$
 $\text{UNIV})$
(is - = ?}f'')
by (*auto simp add: t space-embed-measure extract-int-def*)
also have $\text{has-density (distr } M \text{ (stock-measure } t) \text{ (op-sem Add)) (stock-measure } t)$
 $\text{?}f'' \longleftrightarrow$
 $\text{has-density (distr } M \text{ (stock-measure } t) \text{ (op-sem Add)) (stock-measure } t)$
 $(\text{?}f \varrho)$
by (*intro has-density-cong, simp add: t space-embed-measure del: op-sem.simps*)
(auto simp: nn-integral-IntVal IntPairVal-def val-type-eq-INTEG
lift-RealIntVal2-def lift-RealIntVal-def)
finally show
qed
qed
qed

lemma *hd-cong*:

assumes $(V, V', \Gamma, \delta) \vdash_a e \Rightarrow f \text{ density-context } V V' \Gamma \delta \Gamma \vdash e : t \text{ free-vars } e \subseteq$
 $V \cup V'$
assumes $\bigwedge \varrho x. \varrho \in \text{space (state-measure } V' \Gamma) \Longrightarrow x \in \text{space (stock-measure } t)$
 $\Longrightarrow f \varrho x = f' \varrho x$
shows $(V, V', \Gamma, \delta) \vdash_a e \Rightarrow f'$
proof (*rule hd-AE[OF assms(1,3) AE-I2[OF assms(5)]]*)
note $\text{dens} = \text{expr-has-density-sound-aux}[OF \text{assms}(1,3,2,4)]$
note $\text{dens}' = \text{has-parametrized-subprob-densityD}[OF \text{this}]$
show $(\lambda(\varrho, x). f' \varrho x) \in \text{borel-measurable (state-measure } V' \Gamma \otimes_M \text{stock-measure } t)$
using *assms(5) dens'(3)*
by (*subst measurable-cong[of - - case-prod f]*) (*auto simp: space-pair-measure*)
qed *auto*

```

lemma prob-space-empty-dens-ctxt[simp]:
  prob-space (dens-ctxt-measure ({} , {} ,  $\Gamma$  , ( $\lambda$ -. 1)) ( $\lambda$ -. undefined))
    unfolding density-context-def
    by (auto simp: dens-ctxt-measure-def state-measure'-def state-measure-def
      emeasure-distr emeasure-density PiM-empty intro!: prob-spaceI)

lemma branch-prob-empty-ctxt[simp]: branch-prob ({} , {} ,  $\Gamma$  , ( $\lambda$ -. 1)) ( $\lambda$ -. undefined)
  = 1
    unfolding branch-prob-def by (subst prob-space.emeasure-space-1) simp-all

lemma expr-has-density-sound:
  assumes ({} , {} ,  $\Gamma$  , ( $\lambda$ -. 1))  $\vdash_a$   $e \Rightarrow f \Gamma \vdash e : t$  free-vars  $e = \{\}$ 
  shows has-subprob-density (expr-sem  $\sigma$   $e$ ) (stock-measure  $t$ ) (f ( $\lambda$ -. undefined))
proof -
  let ?M = dens-ctxt-measure ({} , {} ,  $\Gamma$  ,  $\lambda$ -. 1) ( $\lambda$ -. undefined)
  have density-context {} {}  $\Gamma$  ( $\lambda$ -. 1)
    unfolding density-context-def
    by (auto simp: dens-ctxt-measure-def state-measure'-def state-measure-def
      emeasure-distr emeasure-density PiM-empty intro!: subprob-spaceI)
  from expr-has-density-sound-aux[OF assms(1,2) this] assms(3)
    have has-parametrized-subprob-density (state-measure {}  $\Gamma$ )
      ( $\lambda$ q. dens-ctxt-measure ({} , {} ,  $\Gamma$  ,  $\lambda$ -. 1)  $q \ggg (\lambda\sigma. \text{expr-sem } \sigma \ e)$ )
    (stock-measure  $t$ ) f
    by blast
  also have state-measure {}  $\Gamma = \text{count-space } \{\lambda$ -. undefined $\}$ 
    by (rule measure-eqI) (simp-all add: state-measure-def PiM-empty emeasure-density)
  finally have has-subprob-density (?M  $\ggg (\lambda\sigma. \text{expr-sem } \sigma \ e)$ )
    (stock-measure  $t$ ) (f ( $\lambda$ -. undefined))
    unfolding has-parametrized-subprob-density-def by simp
  also from assms have ( $\lambda\sigma. \text{expr-sem } \sigma \ e$ )  $\in$  measurable (state-measure {}  $\Gamma$ )
    (subprob-algebra (stock-measure  $t$ ))
    by (intro measurable-expr-sem) auto
  hence ?M  $\ggg (\lambda\sigma. \text{expr-sem } \sigma \ e) = \text{expr-sem } (\lambda$ -. undefined)  $e$ 
    by (intro dens-ctxt-measure-empty-bind) (auto simp: state-measure-def PiM-empty)
  also from assms have ... = expr-sem  $\sigma \ e$  by (intro expr-sem-eq-on-vars) auto
  finally show ?thesis .
qed

end

```

8 Target Language Syntax and Semantics

```

theory PDF-Target-Semantics
imports PDF-Semantics
begin

```

```

datatype cexpr =
  CVar vname

```

| *CVal* *val*
| *CPair* *cepr cepr* ($\langle \langle -, - \rangle_c \rangle [0, 0] 1000$)
| *COperator* *pdf-operator cepr* (**infixl** $\langle \text{\$}_c \rangle 999$)
| *CIf* *cepr cepr cepr* ($\langle \text{IF}_c - \text{THEN} - \text{ELSE} - \rangle [0, 0, 10] 10$)
| *CIntegral* *cepr pdf-type* ($\langle \int_c - \partial - \rangle [61] 110$)

abbreviation (*input*) *cepr-fun* :: (*cepr* \Rightarrow *cepr*) \Rightarrow *cepr* (**binder** $\langle \lambda_c \rangle 10$)
where

cepr-fun *f* \equiv *f* (*CVar* 0)

abbreviation *cepr-Add* (**infixl** $\langle +_c \rangle 65$) **where**

cepr-Add *a b* \equiv *Add* $\text{\$}_c \langle a, b \rangle_c$

abbreviation *cepr-Minus* ($\langle -_c - \rangle [81] 80$) **where**

cepr-Minus *a* \equiv *Minus* $\text{\$}_c a$

abbreviation *cepr-Sub* (**infixl** $\langle -_c \rangle 65$) **where**

cepr-Sub *a b* \equiv *a* $+_c -_c b$

abbreviation *cepr-Mult* (**infixl** $\langle *_c \rangle 70$) **where**

cepr-Mult *a b* \equiv *Mult* $\text{\$}_c \langle a, b \rangle_c$

abbreviation *inverse_c* *e* \equiv *Inverse* $\text{\$}_c e$

abbreviation *cepr-Div* (**infixl** $\langle /_c \rangle 70$) **where**

cepr-Div *a b* \equiv *a* $*_c \text{inverse}_c b$

abbreviation *fact_c* *e* \equiv *Fact* $\text{\$}_c e$

abbreviation *sqrt_c* *e* \equiv *Sqrt* $\text{\$}_c e$

abbreviation *exp_c* *e* \equiv *Exp* $\text{\$}_c e$

abbreviation *ln_c* *e* \equiv *Ln* $\text{\$}_c e$

abbreviation *fst_c* *e* \equiv *Fst* $\text{\$}_c e$

abbreviation *snd_c* *e* \equiv *Snd* $\text{\$}_c e$

abbreviation *cepr-Pow* (**infixl** $\langle \hat{\ }_c \rangle 75$) **where**

cepr-Pow *a b* \equiv *Pow* $\text{\$}_c \langle a, b \rangle_c$

abbreviation *cepr-And* (**infixl** $\langle \wedge_c \rangle 35$) **where**

cepr-And *a b* \equiv *And* $\text{\$}_c \langle a, b \rangle_c$

abbreviation *cepr-Or* (**infixl** $\langle \vee_c \rangle 30$) **where**

cepr-Or *a b* \equiv *Or* $\text{\$}_c \langle a, b \rangle_c$

abbreviation *cepr-Not* ($\langle \neg_c - \rangle [40] 40$) **where**

cepr-Not *a* \equiv *Not* $\text{\$}_c a$

abbreviation *cepr-Equals* (**infixl** $\langle =_c \rangle 70$) **where**

cepr-Equals *a b* \equiv *Equals* $\text{\$}_c \langle a, b \rangle_c$

abbreviation *cepr-Less* (**infixl** $\langle <_c \rangle 70$) **where**

cepr-Less *a b* \equiv *Less* $\text{\$}_c \langle a, b \rangle_c$

abbreviation *cepr-LessEq* (**infixl** $\langle \leq_c \rangle 70$) **where**

cepr-LessEq *a b* \equiv *a* $=_c b \vee_c a <_c b$

abbreviation *cepr-RealCast* ($\langle \langle - \rangle_c \rangle [0] 90$) **where**

cepr-RealCast *a* \equiv *Cast REAL* $\text{\$}_c a$

abbreviation *CReal* **where**

CReal *x* \equiv *CVal* (*RealVal* *x*)

abbreviation *CInt* **where**

CInt *x* \equiv *CVal* (*IntVal* *x*)

abbreviation π_c **where**

$\pi_c \equiv$ *Pi* $\text{\$}_c$ (*CVal* *UnitVal*)

instantiation $cexpr :: expr$
begin

primrec $free\text{-}vars\text{-}cexpr :: cexpr \Rightarrow vname\ set$ **where**
 $free\text{-}vars\text{-}cexpr\ (CVar\ x) = \{x\}$
 $| free\text{-}vars\text{-}cexpr\ (CVal\ -) = \{\}$
 $| free\text{-}vars\text{-}cexpr\ (oper\ \$\$_c\ e) = free\text{-}vars\text{-}cexpr\ e$
 $| free\text{-}vars\text{-}cexpr\ (<e1,\ e2>_c) = free\text{-}vars\text{-}cexpr\ e1 \cup free\text{-}vars\text{-}cexpr\ e2$
 $| free\text{-}vars\text{-}cexpr\ (IF_c\ b\ THEN\ e1\ ELSE\ e2) =$
 $free\text{-}vars\text{-}cexpr\ b \cup free\text{-}vars\text{-}cexpr\ e1 \cup free\text{-}vars\text{-}cexpr\ e2$
 $| free\text{-}vars\text{-}cexpr\ (\int_c\ e\ \partial t) = Suc\ -' free\text{-}vars\text{-}cexpr\ e$

instance ..
end

inductive $cexpr\text{-}typing :: tyenv \Rightarrow cexpr \Rightarrow pdf\text{-}type \Rightarrow bool$ ($\langle (1-/ \vdash_c / (- :/ -)) \rangle$
 $[50,0,50]$ 50) **where**
 $cet\text{-}val: \Gamma \vdash_c\ CVal\ v: val\text{-}type\ v$
 $| cet\text{-}var: \Gamma \vdash_c\ CVar\ x: \Gamma\ x$
 $| cet\text{-}pair: \Gamma \vdash_c\ e1: t1 \Longrightarrow \Gamma \vdash_c\ e2: t2 \Longrightarrow \Gamma \vdash_c\ <e1,\ e2>_c: PRODUCT\ t1\ t2$
 $| cet\text{-}op: \Gamma \vdash_c\ e: t \Longrightarrow op\text{-}type\ oper\ t = Some\ t' \Longrightarrow \Gamma \vdash_c\ oper\ \$\$_c\ e: t'$
 $| cet\text{-}if: \Gamma \vdash_c\ b: BOOL \Longrightarrow \Gamma \vdash_c\ e1: t \Longrightarrow \Gamma \vdash_c\ e2: t$
 $\Longrightarrow \Gamma \vdash_c\ IF_c\ b\ THEN\ e1\ ELSE\ e2: t$
 $| cet\text{-}int: t \cdot \Gamma \vdash_c\ e: REAL \Longrightarrow \Gamma \vdash_c\ \int_c\ e\ \partial t: REAL$

lemma $cet\text{-}val'$: $t = val\text{-}type\ v \Longrightarrow \Gamma \vdash_c\ CVal\ v: t$
by ($simp\ add: cet\text{-}val$)

lemma $cet\text{-}var'$: $t = \Gamma\ x \Longrightarrow \Gamma \vdash_c\ CVar\ x: t$
by ($simp\ add: cet\text{-}var$)

lemma $cet\text{-}not$: $\Gamma \vdash_c\ e: BOOL \Longrightarrow \Gamma \vdash_c\ \neg_c\ e: BOOL$
by ($intro\ cet\text{-}op[where\ t = BOOL]$ $cet\text{-}pair, simp, simp$)

lemma $cet\text{-}and$: $\Gamma \vdash_c\ e1: BOOL \Longrightarrow \Gamma \vdash_c\ e2: BOOL \Longrightarrow \Gamma \vdash_c\ e1 \wedge_c\ e2: BOOL$
and

$cet\text{-}or: \Gamma \vdash_c\ e1: BOOL \Longrightarrow \Gamma \vdash_c\ e2: BOOL \Longrightarrow \Gamma \vdash_c\ e1 \vee_c\ e2: BOOL$
by ($intro\ cet\text{-}op[where\ t = PRODUCT\ BOOL\ BOOL]$ $cet\text{-}pair, simp, simp,$
 $simp$) $+$

lemma $cet\text{-}minus\text{-}real$: $\Gamma \vdash_c\ e: REAL \Longrightarrow \Gamma \vdash_c\ -_c\ e: REAL$ **and**

$cet\text{-}inverse$: $\Gamma \vdash_c\ e: REAL \Longrightarrow \Gamma \vdash_c\ inverse_c\ e: REAL$ **and**

$cet\text{-}sqrt$: $\Gamma \vdash_c\ e: REAL \Longrightarrow \Gamma \vdash_c\ sqrt_c\ e: REAL$ **and**

$cet\text{-}exp$: $\Gamma \vdash_c\ e: REAL \Longrightarrow \Gamma \vdash_c\ exp_c\ e: REAL$ **and**

$cet\text{-}ln$: $\Gamma \vdash_c\ e: REAL \Longrightarrow \Gamma \vdash_c\ ln_c\ e: REAL$

by ($rule\ cet\text{-}op[where\ t = REAL], simp, simp$) $+$

lemma $cet\text{-}pow\text{-}real$: $\Gamma \vdash_c\ e1: REAL \Longrightarrow \Gamma \vdash_c\ e2: INTEG \Longrightarrow \Gamma \vdash_c\ e1 \hat{\wedge}_c\ e2: REAL$

by (intro cet-op[**where** $t = \text{PRODUCT REAL INTEG}$] cet-pair) simp-all

lemma cet-add-real: $\Gamma \vdash_c e1 : \text{REAL} \implies \Gamma \vdash_c e2 : \text{REAL} \implies \Gamma \vdash_c e1 +_c e2 : \text{REAL}$ **and**

cet-mult-real: $\Gamma \vdash_c e1 : \text{REAL} \implies \Gamma \vdash_c e2 : \text{REAL} \implies \Gamma \vdash_c e1 *_c e2 : \text{REAL}$ **and**

cet-less-real: $\Gamma \vdash_c e1 : \text{REAL} \implies \Gamma \vdash_c e2 : \text{REAL} \implies \Gamma \vdash_c e1 <_c e2 : \text{BOOL}$

by (intro cet-op[**where** $t = \text{PRODUCT REAL REAL}$] cet-pair, simp, simp, simp)+

lemma cet-eq: $\Gamma \vdash_c e1 : t \implies \Gamma \vdash_c e2 : t \implies \Gamma \vdash_c e1 =_c e2 : \text{BOOL}$

by (intro cet-op[**where** $t = \text{PRODUCT } t \ t$] cet-pair, simp, simp, simp)+

lemma cet-less-eq-real: $\Gamma \vdash_c e1 : \text{REAL} \implies \Gamma \vdash_c e2 : \text{REAL} \implies \Gamma \vdash_c e1 \leq_c e2 : \text{BOOL}$

by (intro cet-less-real cet-or cet-eq)

lemma cet-minus-int: $\Gamma \vdash_c e : \text{INTEG} \implies \Gamma \vdash_c -_c e : \text{INTEG}$

by (rule cet-op[**where** $t = \text{INTEG}$], simp, simp)+

lemma cet-add-int: $\Gamma \vdash_c e1 : \text{INTEG} \implies \Gamma \vdash_c e2 : \text{INTEG} \implies \Gamma \vdash_c e1 +_c e2 : \text{INTEG}$ **and**

cet-mult-int: $\Gamma \vdash_c e1 : \text{INTEG} \implies \Gamma \vdash_c e2 : \text{INTEG} \implies \Gamma \vdash_c e1 *_c e2 : \text{INTEG}$ **and**

cet-less-int: $\Gamma \vdash_c e1 : \text{INTEG} \implies \Gamma \vdash_c e2 : \text{INTEG} \implies \Gamma \vdash_c e1 <_c e2 : \text{BOOL}$

by (intro cet-op[**where** $t = \text{PRODUCT INTEG INTEG}$] cet-pair, simp, simp, simp)+

lemma cet-less-eq-int: $\Gamma \vdash_c e1 : \text{INTEG} \implies \Gamma \vdash_c e2 : \text{INTEG} \implies \Gamma \vdash_c e1 \leq_c e2 : \text{BOOL}$

by (intro cet-less-int cet-or cet-eq)

lemma cet-sub-int: $\Gamma \vdash_c e1 : \text{INTEG} \implies \Gamma \vdash_c e2 : \text{INTEG} \implies \Gamma \vdash_c e1 -_c e2 : \text{INTEG}$

by (intro cet-minus-int cet-add-int)

lemma cet-fst: $\Gamma \vdash_c e : \text{PRODUCT } t \ t' \implies \Gamma \vdash_c \text{fst}_c e : t$ **and**

cet-snd: $\Gamma \vdash_c e : \text{PRODUCT } t \ t' \implies \Gamma \vdash_c \text{snd}_c e : t'$

by (erule cet-op, simp)+

lemma cet-cast-real: $\Gamma \vdash_c e : \text{BOOL} \implies \Gamma \vdash_c \langle e \rangle_c : \text{REAL}$

by (intro cet-op[**where** $t = \text{BOOL}$]) simp-all

lemma cet-cast-real-int: $\Gamma \vdash_c e : \text{INTEG} \implies \Gamma \vdash_c \langle e \rangle_c : \text{REAL}$

by (intro cet-op[**where** $t = \text{INTEG}$]) simp-all

lemma cet-sub-real: $\Gamma \vdash_c e1 : \text{REAL} \implies \Gamma \vdash_c e2 : \text{REAL} \implies \Gamma \vdash_c e1 -_c e2 :$

REAL

by (*intro cet-minus-real cet-add-real*)

lemma *cet-pi*: $\Gamma \vdash_c \pi_c : REAL$

by (*rule cet-op, rule cet-val, simp*)

lemmas *cet-op-intros* =

cet-minus-real cet-exp cet-sqrt cet-ln cet-inverse cet-pow-real cet-pi
cet-cast-real cet-add-real cet-mult-real cet-less-real
cet-not cet-and cet-or

inductive-cases *cexpr-typing-valE*[*elim*]: $\Gamma \vdash_c CVal\ v : t$

inductive-cases *cexpr-typing-varE*[*elim*]: $\Gamma \vdash_c CVar\ x : t$

inductive-cases *cexpr-typing-pairE*[*elim*]: $\Gamma \vdash_c \langle e1, e2 \rangle_c : t$

inductive-cases *cexpr-typing-opE*[*elim*]: $\Gamma \vdash_c oper\ \$\$_c\ e : t$

inductive-cases *cexpr-typing-ifE*[*elim*]: $\Gamma \vdash_c IF_c\ b\ THEN\ e1\ ELSE\ e2 : t$

inductive-cases *cexpr-typing-intE*[*elim*]: $\Gamma \vdash_c \int_c\ e\ \partial t : t'$

primrec *cexpr-type* :: *tyenv* \Rightarrow *cexpr* \Rightarrow *pdf-type option* **where**

cexpr-type - (*CVal* *v*) = *Some* (*val-type* *v*)

| *cexpr-type* Γ (*CVar* *x*) = *Some* (Γ *x*)

| *cexpr-type* Γ ($\langle e1, e2 \rangle_c$) = (*case* (*cexpr-type* Γ *e1*, *cexpr-type* Γ *e2*) *of*
(*Some* *t1*, *Some* *t2*) \Rightarrow *Some* (*PRODUCT* *t1* *t2*)
| - \Rightarrow *None*)

| *cexpr-type* Γ (*oper* $\$_{\$}_c$ *e*) = (*case* *cexpr-type* Γ *e* *of*
Some *t* \Rightarrow *op-type* *oper* *t*
| - \Rightarrow *None*)

| *cexpr-type* Γ (*IF*_{*c*} *b* *THEN* *e1* *ELSE* *e2*) =
(*if* *cexpr-type* Γ *b* = *Some* *BOOL* *then*
case (*cexpr-type* Γ *e1*, *cexpr-type* Γ *e2*) *of*
(*Some* *t*, *Some* *t'*) \Rightarrow *if* *t* = *t'* *then* *Some* *t* *else* *None*
| - \Rightarrow *None*
else *None*)

| *cexpr-type* Γ ($\int_c\ e\ \partial t$) =
(*if* *cexpr-type* (*case-nat* *t* Γ) *e* = *Some* *REAL* *then* *Some* *REAL* *else* *None*)

lemma *cexpr-type-Some-iff*: *cexpr-type* Γ *e* = *Some* *t* \longleftrightarrow $\Gamma \vdash_c e : t$

apply *rule*

apply (*induction* *e* *arbitrary*: Γ *t*,

auto *intro!*: *cexpr-typing.intros* *split*: *option.split-asm* *if-split-asm*) []

apply (*induction* *rule*: *cexpr-typing.induct*, *auto*)

done

lemmas *cexpr-typing-code*[*code-unfold*] = *cexpr-type-Some-iff*[*symmetric*]

lemma *cexpr-typing-cong'*:

assumes $\Gamma \vdash_c e : t \wedge x. x \in \text{free-vars } e \implies \Gamma\ x = \Gamma'\ x$

shows $\Gamma' \vdash_c e : t$

using *assms*

proof (*induction arbitrary: Γ' rule: cexpr-typing.induct*)
case (*cet-int t Γ e Γ'*)
hence $\bigwedge x. x \in \text{free-vars } e \implies \text{case-nat } t \ \Gamma \ x = \text{case-nat } t \ \Gamma' \ x$
by (*auto split: nat.split*)
from *cet-int.IH[OF this]* **show** *?case* **by** (*auto intro!: cexpr-typing.intros*)
qed (*auto intro!: cexpr-typing.intros*)

lemma *cexpr-typing-cong*:
assumes $\bigwedge x. x \in \text{free-vars } e \implies \Gamma \ x = \Gamma' \ x$
shows $\Gamma \vdash_c e : t \longleftrightarrow \Gamma' \vdash_c e : t$
by (*rule iffI (erule cexpr-typing-cong', simp add: assms)+*)

primrec *cexpr-sem* :: *state* \Rightarrow *cexpr* \Rightarrow *val* **where**
cexpr-sem σ (CVal *v*) = *v*
cexpr-sem σ (CVar *x*) = $\sigma \ x$
cexpr-sem σ $\langle e1, e2 \rangle_c = \langle \text{cexpr-sem } \sigma \ e1, \text{cexpr-sem } \sigma \ e2 \rangle$
cexpr-sem σ (*oper* $\$_{\$c}$ *e*) = *op-sem oper (cexpr-sem* σ *e*)
cexpr-sem σ (*IF*_{*c*} *b THEN e1 ELSE e2*) = (*if cexpr-sem* σ *b* = *TRUE* then
cexpr-sem σ *e1* else *cexpr-sem* σ *e2*)
cexpr-sem σ ($\int_c e \ \partial t$) = *RealVal* ($\int x. \text{extract-real (cexpr-sem (x \cdot \sigma) e) } \partial(\text{stock-measure } t)$)

definition *cexpr-equiv* :: *cexpr* \Rightarrow *cexpr* \Rightarrow *bool* **where**
cexpr-equiv *e1 e2* $\equiv \forall \sigma. \text{cexpr-sem } \sigma \ e1 = \text{cexpr-sem } \sigma \ e2$

lemma *cexpr-equiv-commute*: *cexpr-equiv* *e1 e2* \longleftrightarrow *cexpr-equiv* *e2 e1*
by (*auto simp: cexpr-equiv-def*)

lemma *val-type-cexpr-sem[simp]*:
assumes $\Gamma \vdash_c e : t$ *free-vars e* $\subseteq V$ $\sigma \in \text{space (state-measure } V \ \Gamma)$
shows *val-type (cexpr-sem* σ *e*) = *t*
using *assms* **by** (*induction arbitrary: $\sigma \ V$ rule: cexpr-typing.induct*)
(*auto intro: state-measure-var-type op-sem-val-type*)

lemma *cexpr-sem-eq-on-vars*:
assumes $\bigwedge x. x \in \text{free-vars } e \implies \sigma \ x = \sigma' \ x$
shows *cexpr-sem* $\sigma \ e = \text{cexpr-sem } \sigma' \ e$
using *assms*
proof (*induction e arbitrary: $\sigma \ \sigma'$*)
case (*CPair e1 e2 $\sigma \ \sigma'$*)
from *CPair.prem*s **show** *?case* **by** (*auto intro!: CPair.IH*)
next
case (*COperator oper e $\sigma \ \sigma'$*)
from *COperator.prem*s **show** *?case* **by** (*auto simp: COperator.IH[of $\sigma \ \sigma'$]*)
next
case (*CIf b e1 e2 $\sigma \ \sigma'$*)
from *CIf.prem*s **show** *?case* **by** (*auto simp: CIf.IH[of $\sigma \ \sigma'$]*)

next
case (*CIntegral* *e t σ σ'*)
have *cexpr-sem* σ ($\int_c e \partial t$) = *RealVal* ($\int x.$ *extract-real* (*cexpr-sem* (*case-nat* *x* σ) *e*) ∂ *stock-measure* *t*)
by *simp*
also from *CIntegral.prem*s **have** *A*: ($\lambda v.$ *cexpr-sem* (*case-nat* *v* σ) *e*) = ($\lambda v.$ *cexpr-sem* (*case-nat* *v* σ') *e*)
by (*intro ext CIntegral.IH*) (*auto split: nat.split*)
also have *RealVal* ($\int x.$ *extract-real* (*cexpr-sem* (*case-nat* *x* σ') *e*) ∂ *stock-measure* *t*) = *cexpr-sem* σ' ($\int_c e \partial t$)
by *simp*
finally show *?case* .
qed *simp-all*

definition *eval-cexpr* :: *cexpr* \Rightarrow *state* \Rightarrow *val* \Rightarrow *real* **where**
eval-cexpr *e* σ *v* = *extract-real* (*cexpr-sem* (*case-nat* *v* σ) *e*)

lemma *measurable-cexpr-sem*[*measurable*]:

$\Gamma \vdash_c e : t \implies \text{free-vars } e \subseteq V \implies$

$(\lambda \sigma. \text{cexpr-sem } \sigma \ e) \in \text{measurable } (\text{state-measure } V \ \Gamma) \ (\text{stock-measure } t)$

proof (*induction arbitrary: V rule: cexpr-typing.induct*)

case (*cet-op oper t t' Γ e*)

thus *?case using measurable-op-sem by simp*

next

case (*cet-int t Γ e*)

interpret *sigma-finite-measure stock-measure t by simp*

let *?M* = ($\prod_M x \in V.$ *stock-measure* ($\Gamma \ x$)) \otimes_M *stock-measure t*

let *?N* = *embed-measure lborel RealVal*

have **[measurable]*: ($\lambda a.$ *cexpr-sem a e*) \in *measurable* (*state-measure* (*shift-var-set* *V*) (*case-nat t* Γ)) *REAL*

using *cet-int.prem*s *subset-shift-var-set*

by (*intro cet-int.IH*) *simp*

show *?case*

by *simp*

qed (*simp-all add: state-measure-def inj-PairVal*)

lemma *measurable-eval-cexpr*[*measurable*]:

assumes *case-nat t* $\Gamma \vdash_c e : \text{REAL}$

assumes *free-vars e* \subseteq *shift-var-set V*

shows *case-prod* (*eval-cexpr e*) \in *borel-measurable* (*state-measure* *V* Γ \otimes_M *stock-measure t*)

unfolding *eval-cexpr-def*[*abs-def*] **using** *measurable-cexpr-sem*[*OF assms*] **by** *simp*

lemma *cexpr-sem-Add*:

assumes $\Gamma \vdash_c e1 : \text{REAL}$ $\Gamma \vdash_c e2 : \text{REAL}$

assumes $\sigma \in \text{space } (\text{state-measure } V \ \Gamma)$ *free-vars e1* $\subseteq V$ *free-vars e2* $\subseteq V$

shows *extract-real* (*cexpr-sem* σ (*e1* +_{*c*} *e2*)) = *extract-real* (*cexpr-sem* σ *e1*) +

extract-real (*cexpr-sem* σ *e2*)
using *val-type-cexpr-sem*[*OF assms*(1,4,3)] *val-type-cexpr-sem*[*OF assms*(2,5,3)]
by (*auto simp: lift-RealIntVal2-def extract-real-def split: val.split*)

lemma *cexpr-sem-Mult*:

assumes $\Gamma \vdash_c e1 : REAL \ \Gamma \vdash_c e2 : REAL$
assumes $\sigma \in \text{space } (\text{state-measure } V \ \Gamma) \ \text{free-vars } e1 \subseteq V \ \text{free-vars } e2 \subseteq V$
shows *extract-real* (*cexpr-sem* σ (*e1* *_c *e2*)) = *extract-real* (*cexpr-sem* σ *e1*) *
extract-real (*cexpr-sem* σ *e2*)
using *val-type-cexpr-sem*[*OF assms*(1,4,3)] *val-type-cexpr-sem*[*OF assms*(2,5,3)]
by (*auto simp: lift-RealIntVal2-def extract-real-def split: val.split*)

8.1 General functions on Expressions

Transform variable names in an expression.

primrec *map-vars* :: (*vname* \Rightarrow *vname*) \Rightarrow *cexpr* \Rightarrow *cexpr* **where**
map-vars *f* (*CVal* *v*) = *CVal* *v*
| *map-vars* *f* (*CVar* *x*) = *CVar* (*f* *x*)
| *map-vars* *f* (*<e1, e2>*_c) = *<map-vars f e1, map-vars f e2>*_c
| *map-vars* *f* (*oper* \$\$_c *e*) = *oper* \$\$_c (*map-vars f e*)
| *map-vars* *f* (*IF*_c *b* *THEN* *e1* *ELSE* *e2*) = (*IF*_c *map-vars f b* *THEN* *map-vars f*
e1 *ELSE* *map-vars f e2*)
| *map-vars* *f* ($\int_c e \ \partial t$) = \int_c *map-vars* (*case-nat* 0 ($\lambda x.$ *Suc* (*f* *x*))) *e* ∂t

lemma *free-vars-map-vars*[*simp*]:

free-vars (*map-vars f e*) = *f* ‘ *free-vars e*

proof (*induction e arbitrary: f*)

case (*CIntegral e t f*)

{

fix *x* *A* **assume** *Suc x* \in *A*

hence *Suc* (*f* *x*) \in *case-nat* 0 ($\lambda x.$ *Suc* (*f* *x*)) ‘ *A*

by (*subst image-iff, intro bexI*[*of - Suc x*]) (*simp split: nat.split*)

}

with *CIntegral* **show** ?*case* **by** (*auto split: nat.split-asm*)

qed *auto*

lemma *cexpr-typing-map-vars*:

$\Gamma \circ f \vdash_c e : t \implies \Gamma \vdash_c \text{map-vars } f \ e : t$

proof (*induction* $\Gamma \circ f \ e \ t$ *arbitrary: Γ f rule: cexpr-typing.induct*)

case (*cet-int t e Γ*)

have *case-nat* *t* ($\Gamma \circ f$) = *case-nat* *t* $\Gamma \circ$ (*case-nat* 0 ($\lambda x.$ *Suc* (*f* *x*)))

by (*intro ext*) (*auto split: nat.split*)

from *cet-int*(2)[*OF this*] **show** ?*case* **by** (*auto intro!: cexpr-typing.intros*)

qed (*auto intro!: cexpr-typing.intros*)

lemma *cexpr-sem-map-vars*:

cexpr-sem σ (*map-vars f e*) = *cexpr-sem* ($\sigma \circ f$) *e*

proof (*induction e arbitrary: σ f*)

case (*CIntegral e t σ f*)

```

{
  fix x
  have cexpr-sem (case-nat x σ) (map-vars (case-nat 0 (λx. Suc (f x))) e) =
    cexpr-sem (case-nat x σ ∘ case-nat 0 (λx. Suc (f x))) e
  by (rule CIntegral.IH)
  also have case-nat x σ ∘ case-nat 0 (λx. Suc (f x)) = case-nat x (λa. σ (f a))
  by (intro ext) (auto simp add: o-def split: nat.split)
  finally have cexpr-sem (case-nat x σ) (map-vars (case-nat 0 (λx. Suc (f x)))
e) =
    cexpr-sem (case-nat x (λa. σ (f a))) e .
}
thus ?case by simp
qed simp-all

```

definition $insert\text{-}var :: vname \Rightarrow (vname \Rightarrow 'a) \Rightarrow 'a \Rightarrow vname \Rightarrow 'a$ **where**
 $insert\text{-}var\ v\ f\ x\ w \equiv$ if $w = v$ then x else if $w > v$ then $f\ (w - 1)$ else $f\ w$

lemma $insert\text{-}var\ 0[simp]: insert\text{-}var\ 0\ f\ x = case\text{-}nat\ x\ f$
by (intro ext) (simp add: insert-var-def split: nat.split)

Substitutes expression e for variable x in e'.

primrec $cepr\text{-}subst :: vname \Rightarrow cepr \Rightarrow cepr \Rightarrow cepr$ **where**
 $cepr\text{-}subst\ -\ -\ (CVal\ v) = CVal\ v$
 $| cepr\text{-}subst\ x\ e\ (CVar\ y) = insert\text{-}var\ x\ CVar\ e\ y$
 $| cepr\text{-}subst\ x\ e\ \langle e1, e2 \rangle_c = \langle cepr\text{-}subst\ x\ e\ e1, cepr\text{-}subst\ x\ e\ e2 \rangle_c$
 $| cepr\text{-}subst\ x\ e\ (oper\ \$\$_c\ e') = oper\ \$\$_c\ (cepr\text{-}subst\ x\ e\ e')$
 $| cepr\text{-}subst\ x\ e\ (IF_c\ b\ THEN\ e1\ ELSE\ e2) =$
 $(IF_c\ cepr\text{-}subst\ x\ e\ b\ THEN\ cepr\text{-}subst\ x\ e\ e1\ ELSE\ cepr\text{-}subst\ x\ e\ e2)$
 $| cepr\text{-}subst\ x\ e\ (\int_c\ e'\ \partial t) = (\int_c\ cepr\text{-}subst\ (Suc\ x)\ (map\text{-}vars\ Suc\ e)\ e'\ \partial t)$

lemma $cepr\text{-}sem\text{-}cepr\text{-}subst\text{-}aux:$
 $cepr\text{-}sem\ \sigma\ (cepr\text{-}subst\ x\ e\ e') = cepr\text{-}sem\ (insert\text{-}var\ x\ \sigma\ (cepr\text{-}sem\ \sigma\ e))$
 e'

proof (induction e' arbitrary: x e σ)
case (CIntegral e' t x e σ)
have $A: \bigwedge y. insert\text{-}var\ (Suc\ x)\ (case\text{-}nat\ y\ \sigma)\ (cepr\text{-}sem\ \sigma\ e) =$
 $case\text{-}nat\ y\ (insert\text{-}var\ x\ \sigma\ (cepr\text{-}sem\ \sigma\ e))$
by (intro ext) (simp add: insert-var-def split: nat.split)
show ?case **by** (simp add: o-def A cepr-sem-map-vars CIntegral.IH)
qed (simp-all add: insert-var-def)

This corresponds to a Let-binding; the variable with index 0 is substituted with the given expression.

lemma $cepr\text{-}sem\text{-}cepr\text{-}subst:$
 $cepr\text{-}sem\ \sigma\ (cepr\text{-}subst\ 0\ e\ e') = cepr\text{-}sem\ (case\text{-}nat\ (cepr\text{-}sem\ \sigma\ e)\ \sigma)\ e'$
using $cepr\text{-}sem\text{-}cepr\text{-}subst\text{-}aux$ **by** simp

lemma $cepr\text{-}typing\text{-}subst\text{-}aux:$
assumes $insert\text{-}var\ x\ \Gamma\ t \vdash_c e' : t' \ \Gamma \vdash_c e : t$

```

  shows  $\Gamma \vdash_c \text{cexpr-subst } x \ e \ e' : t'$ 
using assms
proof (induction e' arbitrary:  $x \ \Gamma \ e \ t'$ )
  case CVar
  thus ?case by (auto intro!: cexpr-typing.intros simp: insert-var-def)
next
  case COperator
  thus ?case by (auto simp: cexpr-type-Some-iff[symmetric] split: option.split-asm)
next
  case (CIntegral e' t')
  have  $t' : t' = \text{REAL}$  using CIntegral.prem1 by auto
  have  $\text{case-nat } t'' \ (\text{insert-var } x \ \Gamma \ t) \vdash_c \ e' : t'$  using CIntegral.prem1 by auto
  also have  $\text{case-nat } t'' \ (\text{insert-var } x \ \Gamma \ t) = \text{insert-var } (\text{Suc } x) \ (\text{case-nat } t'' \ \Gamma) \ t$ 
    by (intro ext) (simp add: insert-var-def split: nat.split)
  finally have  $\text{insert-var } (\text{Suc } x) \ (\text{case-nat } t'' \ \Gamma) \ t \vdash_c \ e' : t'$ .
  moreover from CIntegral.prem2 have  $\text{case-nat } t'' \ \Gamma \vdash_c \ \text{map-vars } \text{Suc } e : t$ 
    by (intro cexpr-typing-map-vars) (simp add: o-def)
  ultimately have  $\text{case-nat } t'' \ \Gamma \vdash_c \ \text{cexpr-subst } (\text{Suc } x) \ (\text{map-vars } \text{Suc } e) \ e' : t'$ 
    by (rule CIntegral.IH)
  thus ?case by (auto intro: cet-int simp: t')
qed (auto intro!: cexpr-typing.intros)

```

```

lemma cexpr-typing-subst[intro]:
  assumes  $\Gamma \vdash_c \ e : t \ \text{case-nat } t \ \Gamma \vdash_c \ e' : t'$ 
  shows  $\Gamma \vdash_c \ \text{cexpr-subst } 0 \ e \ e' : t'$ 
  using cexpr-typing-subst-aux assms by simp

```

```

lemma free-vars-cexpr-subst-aux:
   $\text{free-vars } (\text{cexpr-subst } x \ e \ e') \subseteq (\lambda y. \text{if } y \geq x \text{ then } y + 1 \text{ else } y) -' \text{free-vars } e' \cup$ 
 $\text{free-vars } e$ 
  (is  $\text{free-vars } - \subseteq ?f \ x \ -' \cup -$ )
proof (induction e' arbitrary:  $x \ e$ )
  case (CVar y x e)
  show ?case by (auto simp: insert-var-def)
next
  case (CPair e'1 e'2 x e)
  from CPair.IH[of x e] show ?case by auto
next
  case (COperator - - x e)
  from COperator.IH[of x e] show ?case by auto
next
  case (CIf b e'1 e'2 x e)
  from CIf.IH[of x e] show ?case by auto
next
  case (CIntegral e' t x e)
  have  $\text{free-vars } (\text{cexpr-subst } x \ e \ (\int_c \ e' \ \partial t)) \subseteq$ 
 $\text{Suc } -' \ (\ ?f \ (\text{Suc } x) \ -' \ \text{free-vars } e') \cup$ 
 $\text{Suc } -' \ (\text{free-vars } (\text{map-vars } \text{Suc } e))$  (is  $- \subseteq ?A \cup ?B$ )

```

by (simp only: cexpr-subst.simps free-vars-cexpr.simps
 vimage-mono CIntegral.IH vimage-Un[symmetric])
 also have ?B = free-vars e by (simp add: inj-vimage-image-eq)
 also have ?A \subseteq ?f x - ' free-vars ($\int_c e' \partial t$) by auto
 finally show ?case by blast
 qed simp-all

lemma free-vars-cexpr-subst:

free-vars (cexpr-subst 0 e e') \subseteq Suc - ' free-vars e' \cup free-vars e
 by (rule order.trans[OF free-vars-cexpr-subst-aux]) (auto simp: shift-var-set-def)

primrec cexpr-comp-aux :: vname \Rightarrow cexpr \Rightarrow cexpr \Rightarrow cexpr **where**

cexpr-comp-aux - - (CVal v) = CVal v
 | cexpr-comp-aux x e (CVar y) = (if x = y then e else CVar y)
 | cexpr-comp-aux x e <e1, e2>_c = <cexpr-comp-aux x e e1, cexpr-comp-aux x e e2>_c
 | cexpr-comp-aux x e (oper \$\$_c e') = oper \$\$_c (cexpr-comp-aux x e e')
 | cexpr-comp-aux x e (IF_c b THEN e1 ELSE e2) =
 (IF_c cexpr-comp-aux x e b THEN cexpr-comp-aux x e e1 ELSE cexpr-comp-aux
 x e e2)
 | cexpr-comp-aux x e ($\int_c e' \partial t$) = (\int_c cexpr-comp-aux (Suc x) (map-vars Suc e)
 e' ∂t)

lemma cexpr-sem-cexpr-comp-aux:

cexpr-sem σ (cexpr-comp-aux x e e') = cexpr-sem ($\sigma(x :=$ cexpr-sem σ e)) e'

proof (induction e' arbitrary: x e σ)

case (CIntegral e' t x e σ)

have $\bigwedge y. (case\text{-nat } y \sigma)(\text{Suc } x := \text{cexpr-sem } (case\text{-nat } y \sigma) (\text{map-vars } \text{Suc } e)) =$
 $case\text{-nat } y (\sigma(x := \text{cexpr-sem } \sigma e))$

by (intro ext) (auto simp: cexpr-sem-map-vars o-def split: nat.split)

thus ?case by (auto intro!: integral-cong simp: CIntegral.IH simp del: fun-upd-apply)

qed (simp-all add: insert-var-def)

definition cexpr-comp (infixl <◦_c> 55) **where**

cexpr-comp b a \equiv cexpr-comp-aux 0 a b

lemma cexpr-typing-cexpr-comp-aux:

assumes $\Gamma(x := t1) \vdash_c e' : t2$ $\Gamma \vdash_c e : t1$

shows $\Gamma \vdash_c \text{cexpr-comp-aux } x e e' : t2$

using assms

proof (induction e' arbitrary: Γ e x t2)

case COperator

thus ?case by (elim cexpr-typing-opE) (auto intro!: cexpr-typing.intros) \square

next

case CPair

thus ?case by (elim cexpr-typing-pairE) (auto intro!: cexpr-typing.intros) \square

next

case ($CIntegral\ e'\ t\ \Gamma\ e\ x\ t2$)
from $CIntegral.premis$ **have** $[simp]: t2 = REAL$ **by** $auto$
from $CIntegral.premis$ **have** $case\text{-}nat\ t\ (\Gamma(x := t1)) \vdash_c e' : REAL$ **by** ($elim\ cexpr\text{-}typing\text{-}intE$)
also **have** $case\text{-}nat\ t\ (\Gamma(x := t1)) = (case\text{-}nat\ t\ \Gamma)(Suc\ x := t1)$
by ($intro\ ext$) ($simp\ split: nat.split$)
finally **have** $\dots \vdash_c e' : REAL$.
thus $\Gamma \vdash_c cexpr\text{-}comp\text{-}aux\ x\ e\ (\int_c e' \partial t) : t2$
by ($auto\ intro!: cexpr\text{-}typing.intros\ CIntegral.IH\ cexpr\text{-}typing.map\text{-}vars$
 $simp: o\text{-}def\ CIntegral.premis$)
qed ($auto\ intro!: cexpr\text{-}typing.intros$)

lemma $cexpr\text{-}typing\text{-}cexpr\text{-}comp[intro]$:
assumes $case\text{-}nat\ t1\ \Gamma \vdash_c g : t2$
assumes $case\text{-}nat\ t2\ \Gamma \vdash_c f : t3$
shows $case\text{-}nat\ t1\ \Gamma \vdash_c f \circ_c g : t3$
proof ($unfold\ cexpr\text{-}comp\text{-}def, intro\ cexpr\text{-}typing\text{-}cexpr\text{-}comp\text{-}aux$)
have $(case\text{-}nat\ t1\ \Gamma)(0 := t2) = case\text{-}nat\ t2\ \Gamma$
by ($intro\ ext$) ($simp\ split: nat.split$)
with $assms$ **show** $(case\text{-}nat\ t1\ \Gamma)(0 := t2) \vdash_c f : t3$ **by** $simp$
qed ($insert\ assms$)

lemma $free\text{-}vars\text{-}cexpr\text{-}comp\text{-}aux$:
 $free\text{-}vars\ (cexpr\text{-}comp\text{-}aux\ x\ e\ e') \subseteq (free\text{-}vars\ e' - \{x\}) \cup free\text{-}vars\ e$
proof ($induction\ e'\ arbitrary: x\ e$)
case ($CIntegral\ e'\ t\ x\ e$)
note $IH = CIntegral.IH[of\ Suc\ x\ map\text{-}vars\ Suc\ e]$
have $free\text{-}vars\ (cexpr\text{-}comp\text{-}aux\ x\ e\ (\int_c e' \partial t)) =$
 $Suc - ' free\text{-}vars\ (cexpr\text{-}comp\text{-}aux\ (Suc\ x)\ (map\text{-}vars\ Suc\ e)\ e')$ **by** $simp$
also **have** $\dots \subseteq Suc - ' (free\text{-}vars\ e' - \{Suc\ x\} \cup free\text{-}vars\ (map\text{-}vars\ Suc\ e))$
by ($rule\ vimage\text{-}mono, rule\ CIntegral.IH$)
also **have** $\dots \subseteq free\text{-}vars\ (\int_c e' \partial t) - \{x\} \cup free\text{-}vars\ e$
by ($auto\ simp\ add: vimage\text{-}Diff\ vimage\text{-}image\text{-}eq$)
finally **show** $?case$.
qed ($simp, blast?$) $+$

lemma $free\text{-}vars\text{-}cexpr\text{-}comp$:
 $free\text{-}vars\ (cexpr\text{-}comp\ e\ e') \subseteq (free\text{-}vars\ e - \{0\}) \cup free\text{-}vars\ e'$
by ($simp\ add: free\text{-}vars\text{-}cexpr\text{-}comp\text{-}aux\ cexpr\text{-}comp\text{-}def$)

lemma $free\text{-}vars\text{-}cexpr\text{-}comp'$:
 $free\text{-}vars\ (cexpr\text{-}comp\ e\ e') \subseteq free\text{-}vars\ e \cup free\text{-}vars\ e'$
using $free\text{-}vars\text{-}cexpr\text{-}comp$ **by** $blast$

lemma $cexpr\text{-}sem\text{-}cexpr\text{-}comp$:
 $cexpr\text{-}sem\ \sigma\ (f \circ_c g) = cexpr\text{-}sem\ (\sigma(0 := cexpr\text{-}sem\ \sigma\ g))\ f$
unfolding $cexpr\text{-}comp\text{-}def$ **by** ($simp\ add: cexpr\text{-}sem\text{-}cexpr\text{-}comp\text{-}aux$)

lemma *eval-cexpr-comp*:

$$\text{eval-cexpr } (f \circ_c g) \sigma x = \text{eval-cexpr } f \sigma (\text{cexpr-sem } (\text{case-nat } x \sigma) g)$$

proof –

$$\text{have } (\text{case-nat } x \sigma)(0 := \text{cexpr-sem } (\text{case-nat } x \sigma) g) = \text{case-nat } (\text{cexpr-sem } (\text{case-nat } x \sigma) g) \sigma$$

$$\text{by } (\text{intro ext}) (\text{auto split: nat.split})$$

$$\text{thus ?thesis by } (\text{simp add: eval-cexpr-def cexpr-sem-cexpr-comp})$$

qed

primrec *cexpr-subst-val-aux* :: $\text{nat} \Rightarrow \text{cexpr} \Rightarrow \text{val} \Rightarrow \text{cexpr}$ **where**

$$\text{cexpr-subst-val-aux } - (\text{CVal } v) - = \text{CVal } v$$

$$| \text{cexpr-subst-val-aux } x (\text{CVar } y) v = \text{insert-var } x \text{ CVar } (\text{CVal } v) y$$

$$| \text{cexpr-subst-val-aux } x (\text{IF}_c b \text{ THEN } e1 \text{ ELSE } e2) v =$$

$$(\text{IF}_c \text{ cexpr-subst-val-aux } x b v \text{ THEN } \text{cexpr-subst-val-aux } x e1 v \text{ ELSE } \text{cexpr-subst-val-aux } x e2 v)$$

$$| \text{cexpr-subst-val-aux } x (\text{oper } \$_\$_c e) v = \text{oper } \$_\$_c (\text{cexpr-subst-val-aux } x e v)$$

$$| \text{cexpr-subst-val-aux } x \langle e1, e2 \rangle_c v = \langle \text{cexpr-subst-val-aux } x e1 v, \text{cexpr-subst-val-aux } x e2 v \rangle_c$$

$$| \text{cexpr-subst-val-aux } x (\int_c e \partial t) v = \int_c \text{cexpr-subst-val-aux } (\text{Suc } x) e v \partial t$$

lemma *cexpr-subst-val-aux-eq-cexpr-subst*:

$$\text{cexpr-subst-val-aux } x e v = \text{cexpr-subst } x (\text{CVal } v) e$$

$$\text{by } (\text{induction } e \text{ arbitrary: } x) \text{ simp-all}$$

definition *cexpr-subst-val* :: $\text{cexpr} \Rightarrow \text{val} \Rightarrow \text{cexpr}$ **where**

$$\text{cexpr-subst-val } e v \equiv \text{cexpr-subst-val-aux } 0 e v$$

lemma *cexpr-sem-cexpr-subst-val[simp]*:

$$\text{cexpr-sem } \sigma (\text{cexpr-subst-val } e v) = \text{cexpr-sem } (\text{case-nat } v \sigma) e$$

$$\text{by } (\text{simp add: cexpr-subst-val-def cexpr-subst-val-aux-eq-cexpr-subst cexpr-sem-cexpr-subst})$$

lemma *cexpr-typing-subst-val[intro]*:

$$\text{case-nat } t \Gamma \vdash_c e : t' \Longrightarrow \text{val-type } v = t \Longrightarrow \Gamma \vdash_c \text{cexpr-subst-val } e v : t'$$

$$\text{by } (\text{auto simp: cexpr-subst-val-def cexpr-subst-val-aux-eq-cexpr-subst intro!: cet-val'})$$

lemma *free-vars-cexpr-subst-val-aux*:

$$\text{free-vars } (\text{cexpr-subst-val-aux } x e v) = (\lambda y. \text{if } y \geq x \text{ then } \text{Suc } y \text{ else } y) -'$$

$$\text{free-vars } e$$

$$\text{by } (\text{induction } e \text{ arbitrary: } x) (\text{auto simp: insert-var-def split: if-split-asm})$$

lemma *free-vars-cexpr-subst-val[simp]*:

$$\text{free-vars } (\text{cexpr-subst-val } e v) = \text{Suc } -' \text{ free-vars } e$$

$$\text{by } (\text{simp add: cexpr-subst-val-def free-vars-cexpr-subst-val-aux})$$

8.2 Nonnegative expressions

definition *nonneg-cexpr* $V \Gamma e \equiv$

$$\forall \sigma \in \text{space } (\text{state-measure } V \Gamma). \text{extract-real } (\text{cexpr-sem } \sigma e) \geq 0$$

lemma *nonneg-cexprI*:

$(\bigwedge \sigma. \sigma \in \text{space } (\text{state-measure } V \Gamma) \implies \text{extract-real } (\text{cexpr-sem } \sigma e) \geq 0) \implies$
 $\text{nonneg-cexpr } V \Gamma e$

unfolding *nonneg-cexpr-def by simp*

lemma *nonneg-cexprD*:

$\text{nonneg-cexpr } V \Gamma e \implies \sigma \in \text{space } (\text{state-measure } V \Gamma) \implies \text{extract-real}$
 $(\text{cexpr-sem } \sigma e) \geq 0$

unfolding *nonneg-cexpr-def by simp*

lemma *nonneg-cexpr-map-vars*:

assumes *nonneg-cexpr* $(f - ' V) (\Gamma \circ f) e$

shows *nonneg-cexpr* $V \Gamma (\text{map-vars } f e)$

by (*intro nonneg-cexprI, subst cexpr-sem-map-vars, intro nonneg-cexprD[OF assms]*)
(auto simp: state-measure-def space-PiM)

lemma *nonneg-cexpr-subset*:

assumes *nonneg-cexpr* $V \Gamma e$ $V \subseteq V'$ *free-vars* $e \subseteq V$

shows *nonneg-cexpr* $V' \Gamma e$

proof (*intro nonneg-cexprI*)

fix σ **assume** $\sigma \in \text{space } (\text{state-measure } V' \Gamma)$

with *assms(2)* **have** $\text{restrict } \sigma V \in \text{space } (\text{state-measure } V \Gamma)$

by (*auto simp: state-measure-def space-PiM restrict-def*)

from *nonneg-cexprD[OF assms(1) this]* **have** $\text{extract-real } (\text{cexpr-sem } (\text{restrict } \sigma$
 $V) e) \geq 0$.

also have $\text{cexpr-sem } (\text{restrict } \sigma V) e = \text{cexpr-sem } \sigma e$ **using** *assms(3)*

by (*intro cexpr-sem-eq-on-vars*) *auto*

finally show $\text{extract-real } (\text{cexpr-sem } \sigma e) \geq 0$.

qed

lemma *nonneg-cexpr-Mult*:

assumes $\Gamma \vdash_c e1 : \text{REAL}$ $\Gamma \vdash_c e2 : \text{REAL}$

assumes *free-vars* $e1 \subseteq V$ *free-vars* $e2 \subseteq V$

assumes *N1*: *nonneg-cexpr* $V \Gamma e1$ **and** *N2*: *nonneg-cexpr* $V \Gamma e2$

shows *nonneg-cexpr* $V \Gamma (e1 *_c e2)$

proof (*rule nonneg-cexprI*)

fix σ **assume** $\sigma \in \text{space } (\text{state-measure } V \Gamma)$

hence $\text{extract-real } (\text{cexpr-sem } \sigma (e1 *_c e2)) = \text{extract-real } (\text{cexpr-sem } \sigma e1) *$
 $\text{extract-real } (\text{cexpr-sem } \sigma e2)$

using *assms* **by** (*subst cexpr-sem-Mult[of \Gamma - - V]*) *simp-all*

also have $\dots \geq 0$ **using** σ *N1 N2* **by** (*intro mult-nonneg-nonneg nonneg-cexprD*)

finally show $\text{extract-real } (\text{cexpr-sem } \sigma (e1 *_c e2)) \geq 0$.

qed

lemma *nonneg-indicator*:

assumes $\Gamma \vdash_c e : \text{BOOL}$ *free-vars* $e \subseteq V$

shows *nonneg-cexpr* $V \Gamma (\langle e \rangle_c)$

proof (*intro nonneg-cexprI*)

fix ρ **assume** $\rho \in \text{space } (\text{state-measure } V \Gamma)$

with *assms* **have** *val-type* (*cexpr-sem* ϱ *e*) = *BOOL* **by** (*rule val-type-cexpr-sem*)
thus *extract-real* (*cexpr-sem* ϱ ($\langle e \rangle_c$)) ≥ 0
by (*auto simp: extract-real-def bool-to-real-def split: val.split*)
qed

lemma *nonneg-cexpr-comp-aux*:

assumes *nonneg*: *nonneg-cexpr* *V* ($\Gamma(x := t1)$) *e* **and** *x*:*x* \in *V*
assumes *t2*: $\Gamma(x:=t1) \vdash_c e : t2$ **and** *t1*: $\Gamma \vdash_c f : t1$ **and** *vars*: *free-vars* *f* \subseteq *V*
shows *nonneg-cexpr* *V* Γ (*cexpr-comp-aux* *x f e*)
proof (*intro nonneg-cexprI*)
fix σ **assume** σ : $\sigma \in \text{space}(\text{state-measure } V \Gamma)$
have *extract-real* (*cexpr-sem* σ (*cexpr-comp-aux* *x f e*)) =
extract-real (*cexpr-sem* ($\sigma(x := \text{cexpr-sem } \sigma f)$) *e*)
by (*simp add: cexpr-sem-cexpr-comp-aux*)
also from *val-type-cexpr-sem*[*OF t1 vars* σ] **have** *cexpr-sem* $\sigma f \in \text{type-universe}$
t1 **by** *auto*
with σ *x* **have** $\sigma(x := \text{cexpr-sem } \sigma f) \in \text{space}(\text{state-measure } V (\Gamma(x := t1)))$
by (*auto simp: state-measure-def space-PiM shift-var-set-def split: if-split-asm*)
hence *extract-real* (*cexpr-sem* ($\sigma(x := \text{cexpr-sem } \sigma f)$) *e*) ≥ 0
by(*intro nonneg-cexprD[OF assms(1)]*)
finally show *extract-real* (*cexpr-sem* σ (*cexpr-comp-aux* *x f e*)) ≥ 0 .
qed

lemma *nonneg-cexpr-comp*:

assumes *nonneg-cexpr* (*shift-var-set* *V*) (*case-nat* *t2* Γ) *e*
assumes *case-nat* *t1* $\Gamma \vdash_c f : t2$ *free-vars* *f* \subseteq *shift-var-set* *V*
shows *nonneg-cexpr* (*shift-var-set* *V*) (*case-nat* *t1* Γ) (*e* \circ_c *f*)
proof (*intro nonneg-cexprI*)
fix σ **assume** σ : $\sigma \in \text{space}(\text{state-measure}(\text{shift-var-set } V) (\text{case-nat } t1 \Gamma))$
have *extract-real* (*cexpr-sem* σ (*e* \circ_c *f*)) = *extract-real* (*cexpr-sem* ($\sigma(0 :=$
cexpr-sem $\sigma f)$) *e*)
by (*simp add: cexpr-sem-cexpr-comp*)
also from *val-type-cexpr-sem*[*OF assms(2,3)* σ] **have** *cexpr-sem* $\sigma f \in \text{type-universe}$
t2 **by** *auto*
with σ **have** $\sigma(0 := \text{cexpr-sem } \sigma f) \in \text{space}(\text{state-measure}(\text{shift-var-set } V)$
(*case-nat* *t2* Γ))
by (*auto simp: state-measure-def space-PiM shift-var-set-def split: if-split-asm*)
hence *extract-real* (*cexpr-sem* ($\sigma(0 := \text{cexpr-sem } \sigma f)$) *e*) ≥ 0
by(*intro nonneg-cexprD[OF assms(1)]*)
finally show *extract-real* (*cexpr-sem* σ (*e* \circ_c *f*)) ≥ 0 .
qed

lemma *nonneg-cexpr-subst-val*:

assumes *nonneg-cexpr* (*shift-var-set* *V*) (*case-nat* *t* Γ) *e* *val-type* *v* = *t*
shows *nonneg-cexpr* *V* Γ (*cexpr-subst-val* *e v*)
proof (*intro nonneg-cexprI*)
fix σ **assume** σ : $\sigma \in \text{space}(\text{state-measure } V \Gamma)$
moreover from *assms(2)* **have** *v* $\in \text{type-universe } t$ **by** *auto*
ultimately show *extract-real* (*cexpr-sem* σ (*cexpr-subst-val* *e v*)) ≥ 0

by (auto intro!: nonneg-cexprD[OF assms(1)])
qed

lemma nonneg-cexpr-int:

assumes nonneg-cexpr (shift-var-set V) (case-nat t Γ) e

shows nonneg-cexpr V Γ ($\int_c e \partial t$)

proof (intro nonneg-cexprI)

fix σ assume σ : $\sigma \in \text{space } (\text{state-measure } V \Gamma)$

have extract-real (cexpr-sem σ ($\int_c e \partial t$)) = $\int x.$ extract-real (cexpr-sem (case-nat x σ) e) $\partial \text{stock-measure } t$

by (simp add: extract-real-def)

also from σ have ... ≥ 0

by (intro integral-nonneg-AE AE-I2 nonneg-cexprD[OF assms]) auto

finally show extract-real (cexpr-sem σ ($\int_c e \partial t$)) ≥ 0 .

qed

Subprobability density expressions

definition subprob-cexpr V V' Γ e \equiv

$\forall \rho \in \text{space } (\text{state-measure } V' \Gamma).$

$(\int^+ \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } V V' (\sigma, \rho)) e) \partial \text{state-measure } V \Gamma) \leq$

1

lemma subprob-cexprI:

assumes $\bigwedge \rho. \rho \in \text{space } (\text{state-measure } V' \Gamma) \implies$

$(\int^+ \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } V V' (\sigma, \rho)) e) \partial \text{state-measure}$

V $\Gamma) \leq 1$

shows subprob-cexpr V V' Γ e

using assms unfolding subprob-cexpr-def by simp

lemma subprob-cexprD:

assumes subprob-cexpr V V' Γ e

shows $\bigwedge \rho. \rho \in \text{space } (\text{state-measure } V' \Gamma) \implies$

$(\int^+ \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } V V' (\sigma, \rho)) e) \partial \text{state-measure}$

V $\Gamma) \leq 1$

using assms unfolding subprob-cexpr-def by simp

lemma subprob-indicator:

assumes subprob: subprob-cexpr V V' Γ e1 and nonneg: nonneg-cexpr (V \cup V') Γ e1

assumes t1: $\Gamma \vdash_c e1 : \text{REAL}$ and t2: $\Gamma \vdash_c e2 : \text{BOOL}$

assumes vars1: free-vars e1 $\subseteq V \cup V'$ and vars2: free-vars e2 $\subseteq V \cup V'$

shows subprob-cexpr V V' Γ (e1 *_c $\langle e2 \rangle_c$)

proof (intro subprob-cexprI)

fix ρ assume ρ : $\rho \in \text{space } (\text{state-measure } V' \Gamma)$

from t2 have t2': $\Gamma \vdash_c \langle e2 \rangle_c : \text{REAL}$ by (rule cet-op) simp-all

from vars2 have vars2': free-vars ($\langle e2 \rangle_c$) $\subseteq V \cup V'$ by simp

let ?eval = $\lambda \sigma e. \text{extract-real } (\text{cexpr-sem } (\text{merge } V V' (\sigma, \rho)) e)$

have $(\int^+ \sigma. ?eval \sigma (e1 *_{c} \langle e2 \rangle_c) \partial \text{state-measure } V \Gamma) =$

$(\int^+ \sigma. ?eval \sigma e1 * ?eval \sigma (\langle e2 \rangle_c) \partial \text{state-measure } V \Gamma)$

by (*intro nn-integral-cong*)
 (*simp only: cexpr-sem-Mult[OF t1 t2' merge-in-state-measure[OF - ϱ] vars1 vars2']*)
 also {
 fix σ assume $\sigma: \sigma \in \text{space } (\text{state-measure } V \Gamma)$
 with ϱ have *val-type* (*cexpr-sem* (*merge* $V V' (\sigma, \varrho)$) $e2$) = *BOOL*
 by (*intro val-type-cexpr-sem[OF t2 vars2] merge-in-state-measure*)
 hence *?eval* $\sigma (\langle e2 \rangle_c) \in \{0, 1\}$
 by (*cases cexpr-sem* (*merge* $V V' (\sigma, \varrho)$) $e2$) (*auto simp: extract-real-def bool-to-real-def*)
 moreover have *?eval* $\sigma e1 \geq 0$ using *nonneg $\varrho \sigma$*
 by (*auto intro!: nonneg-cexprD merge-in-state-measure*)
 ultimately have *?eval* $\sigma e1 * ?eval \sigma (\langle e2 \rangle_c) \leq ?eval \sigma e1$
 by (*intro mult-right-le-one-le*) *auto*
 }
 hence $(\int^{+\sigma}. ?eval \sigma e1 * ?eval \sigma (\langle e2 \rangle_c) \partial \text{state-measure } V \Gamma) \leq$
 $(\int^{+\sigma}. ?eval \sigma e1 \partial \text{state-measure } V \Gamma)$
 by (*intro nn-integral-mono*) (*simp add: ennreal-leI*)
 also from *subprob* and ϱ have $\dots \leq 1$ by (*rule subprob-cexprD*)
 finally show $(\int^{+\sigma}. ?eval \sigma (e1 *_c \langle e2 \rangle_c) \partial \text{state-measure } V \Gamma) \leq 1$.
 qed

lemma *measurable-cexpr-sem'*:

assumes $\varrho: \varrho \in \text{space } (\text{state-measure } V' \Gamma)$
 assumes $e: \Gamma \vdash_c e : \text{REAL free-vars } e \subseteq V \cup V'$
 shows $(\lambda \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } V V' (\sigma, \varrho)) e))$
 $\in \text{borel-measurable } (\text{state-measure } V \Gamma)$
 apply (*rule measurable-compose[OF - measurable-extract-real]*)
 apply (*rule measurable-compose[OF - measurable-cexpr-sem[OF e]]*)
 apply (*insert ϱ , unfold state-measure-def, rule measurable-compose[OF - measurable-merge], simp*)
 done

lemma *measurable-fun-upd-state-measure[measurable]*:

assumes $v \notin V$
 shows $(\lambda(x,y). y(v := x)) \in \text{measurable } (\text{stock-measure } (\Gamma v) \otimes_M \text{state-measure } V \Gamma)$
 $(\text{state-measure } (\text{insert } v V) \Gamma)$
 unfolding *state-measure-def* by *simp*

lemma *integrable-cexpr-projection*:

assumes *fin: finite* V
 assumes *disjoint: $V \cap V' = \{\}$* $v \notin V v \notin V'$
 assumes $\varrho: \varrho \in \text{space } (\text{state-measure } V' \Gamma)$
 assumes $e: \Gamma \vdash_c e : \text{REAL free-vars } e \subseteq \text{insert } v V \cup V'$
 assumes *int: integrable* (*state-measure* (*insert* $v V$) Γ)
 $(\lambda \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{insert } v V) V' (\sigma, \varrho)) e))$
 (is *integrable - ?f'*)

shows $AE\ x$ in *stock-measure* $(\Gamma\ v)$.
integrable (*state-measure* $V\ \Gamma$)
 $(\lambda\sigma.$ *extract-real* (*cepr-sem* (*merge* V (*insert* $v\ V'$) $(\sigma, \varrho(v := x))$) e))
(is $AE\ x$ in $?N$. *integrable* $?M$ ($?f\ x$))
proof (*unfold real-integrable-def*, *intro AE-conjI*)
show $AE\ x$ in $?N$. $?f\ x \in$ *borel-measurable* $?M$ **using** $\varrho\ e$ *disjoint*
by (*intro AE-I2 measurable-cepr-sem'*)
(auto simp: state-measure-def space-PiM dest: PiE-mem split: if-split-asm)

let $?f'' = \lambda x\ \sigma.$ *extract-real* (*cepr-sem* (*merge* (*insert* $v\ V$) V' $(\sigma(v := x), \varrho)$)
 e)
 $\{$
fix $x\ \sigma$ **assume** $x \in$ *space* $?N\ \sigma \in$ *space* $?M$
hence *merge* (*insert* $v\ V$) V' $(\sigma(v := x), \varrho) =$ *merge* V (*insert* $v\ V'$) $(\sigma, \varrho(v$
 $:= x))$
using *disjoint* **by** (*intro ext*) (*simp add: merge-def split: if-split-asm*)
hence $?f''\ x\ \sigma = ?f\ x\ \sigma$ **by** *simp*
 $\}$ **note** f'' -*eq-f* = *this*

interpret *product-sigma-finite* $(\lambda v.$ *stock-measure* $(\Gamma\ v))$
by (*simp add: product-sigma-finite-def*)
interpret *sigma-finite-measure state-measure* $V\ \Gamma$
by (*rule sigma-finite-state-measure[OF fin]*)

from *int* **have** $(\int^{+\sigma}.$ *ennreal* ($?f'\ \sigma$) ∂ *state-measure* (*insert* $v\ V$) $\Gamma) \neq \infty$
by (*simp add: real-integrable-def*)
also **have** $(\int^{+\sigma}.$ *ennreal* ($?f'\ \sigma$) ∂ *state-measure* (*insert* $v\ V$) $\Gamma) =$
 $\int^{+x}.$ $\int^{+\sigma}.$ *ennreal* ($?f''\ x\ \sigma$) $\partial ?M\ \partial ?N$ (**is** $- = ?I$)
using *fin disjoint e \varrho*
by (*unfold state-measure-def*, *subst product-nn-integral-insert-rev*)
(auto intro!: measurable-compose[OF - measurable-ennreal] measurable-cepr-sem'[unfolded
state-measure-def])
finally **have** $AE\ x$ in $?N$. $(\int^{+\sigma}.$ *ennreal* ($?f''\ x\ \sigma$) $\partial ?M) \neq \infty$ (**is** $?P$) **using** e
disjoint
by (*intro nn-integral-PInf-AE*)
(auto simp: measurable-split-conv intro!: borel-measurable-nn-integral measur-
able-compose[OF - measurable-ennreal]
measurable-compose[OF - measurable-cepr-sem'[OF \varrho]])
moreover **have** $\bigwedge x. x \in$ *space* $?N \implies (\int^{+\sigma}.$ *ennreal* ($?f''\ x\ \sigma$) $\partial ?M) = (\int^{+\sigma}.$
ennreal ($?f\ x\ \sigma$) $\partial ?M)$
by (*intro nn-integral-cong*) (*simp add: f''-eq-f*)
hence $?P \iff (AE\ x$ in $?N$. $(\int^{+\sigma}.$ *ennreal* ($?f\ x\ \sigma$) $\partial ?M) \neq \infty)$ **by** (*intro*
AE-cong) *simp*
ultimately **show** $AE\ x$ in $?N$. $(\int^{+\sigma}.$ *ennreal* ($?f\ x\ \sigma$) $\partial ?M) \neq \infty$ **by** *simp*

from *int* **have** $(\int^{+\sigma}.$ *ennreal* $(-?f'\ \sigma)$ ∂ *state-measure* (*insert* $v\ V$) $\Gamma) \neq \infty$
by (*simp add: real-integrable-def*)
also **have** $(\int^{+\sigma}.$ *ennreal* $(-?f'\ \sigma)$ ∂ *state-measure* (*insert* $v\ V$) $\Gamma) =$
 $\int^{+x}.$ $\int^{+\sigma}.$ *ennreal* $(-?f''\ x\ \sigma)$ $\partial ?M\ \partial ?N$ (**is** $- = ?I$)

using *fin disjoint e ρ*
by (*unfold state-measure-def, subst product-nn-integral-insert-rev*)
(auto intro!: measurable-compose[OF - measurable-ennreal] borel-measurable-uminus
measurable-cexpr-sem'[unfolded state-measure-def])
finally have $AE\ x\ in\ ?N. (\int^+\sigma. ennreal\ (-?f''\ x\ \sigma)\ \partial?M) \neq \infty$ (**is** $?P$) **using**
e disjoint
by (*intro nn-integral-PInf-AE*)
(auto simp: measurable-split-conv intro!: borel-measurable-nn-integral measur-
able-compose[OF - measurable-ennreal]
measurable-compose[OF - measurable-cexpr-sem'[OF ρ]] borel-measurable-uminus)
moreover have $\bigwedge x. x \in space\ ?N \implies (\int^+\sigma. ennreal\ (-?f''\ x\ \sigma)\ \partial?M) =$
 $(\int^+\sigma. ennreal\ (-?f\ x\ \sigma)\ \partial?M)$
by (*intro nn-integral-cong*) (*simp add: f''-eq-f*)
hence $?P \longleftrightarrow (AE\ x\ in\ ?N. (\int^+\sigma. ennreal\ (-?f\ x\ \sigma)\ \partial?M) \neq \infty)$ **by** (*intro*
AE-cong) *simp*
ultimately show $AE\ x\ in\ ?N. (\int^+\sigma. ennreal\ (-?f\ x\ \sigma)\ \partial?M) \neq \infty$ **by** *simp*
qed

definition *cdens-ctxt-invar :: vname list \Rightarrow vname list \Rightarrow tyenv \Rightarrow cexpr \Rightarrow bool*
where

cdens-ctxt-invar vs vs' $\Gamma\ \delta \equiv$
distinct (vs @ vs') \wedge
free-vars $\delta \subseteq set\ (vs\ @\ vs')$ \wedge
 $\Gamma \vdash_c \delta : REAL \wedge$
nonneg-cexpr (set vs \cup set vs') $\Gamma\ \delta \wedge$
subprob-cexpr (set vs) (set vs') $\Gamma\ \delta$

lemma *cdens-ctxt-invarI:*

$\llbracket distinct\ (vs\ @\ vs');\ free-vars\ \delta \subseteq set\ (vs\ @\ vs');\ \Gamma \vdash_c \delta : REAL;$
 $nonneg-cexpr\ (set\ vs\ \cup\ set\ vs')\ \Gamma\ \delta;$
 $subprob-cexpr\ (set\ vs)\ (set\ vs')\ \Gamma\ \delta \rrbracket \implies$
 $cdens-ctxt-invar\ vs\ vs'\ \Gamma\ \delta$
by (*simp add: cdens-ctxt-invar-def*)

lemma *cdens-ctxt-invarD:*

assumes *cdens-ctxt-invar vs vs' $\Gamma\ \delta$*
shows *distinct (vs @ vs')* *free-vars $\delta \subseteq set\ (vs\ @\ vs')$ $\Gamma \vdash_c \delta : REAL$*
nonneg-cexpr (set vs \cup set vs') $\Gamma\ \delta$ subprob-cexpr (set vs) (set vs') $\Gamma\ \delta$
using *assms* **by** (*simp-all add: cdens-ctxt-invar-def*)

lemma *cdens-ctxt-invar-empty:*

assumes *cdens-ctxt-invar vs vs' $\Gamma\ \delta$*
shows *cdens-ctxt-invar $\llbracket (vs\ @\ vs')\ \Gamma\ (CReal\ 1)$*
using *cdens-ctxt-invarD[OF assms]*
by (*intro cdens-ctxt-invarI*)
(auto simp: cexpr-type-Some-iff[symmetric] extract-real-def state-measure-def
PiM-empty
intro!: nonneg-cexprI subprob-cexprI)

lemma *cdens-ctxt-invar-imp-integrable*:
assumes *cdens-ctxt-invar vs vs' Γ δ* **and** ϱ : $\varrho \in \text{space } (\text{state-measure } (\text{set } vs') \Gamma)$
shows *integrable (state-measure (set vs) Γ)*
 $(\lambda\sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } vs) (\text{set } vs') (\sigma, \varrho)) \delta))$ **(is**
integrable ?M ?f)
unfolding *integrable-iff-bounded*
proof (*intro conjI*)
note *invar = cdens-ctxt-invarD[OF assms(1)]*
show $?f \in \text{borel-measurable } ?M$
apply (*rule measurable-compose[OF - measurable-extract-real]*)
apply (*rule measurable-compose[OF - measurable-cexpr-sem[OF invar(3,2)]]*)
apply (*simp only: state-measure-def set-append, rule measurable-compose[OF -*
measurable-merge])
apply (*rule measurable-Pair, simp, insert assms(2), simp add: state-measure-def*)
done

have *nonneg: $\bigwedge\sigma. \sigma \in \text{space } ?M \implies ?f \sigma \geq 0$*
using $\langle \text{nonneg-cexpr } (\text{set } vs \cup \text{set } vs') \Gamma \delta \rangle$
by (*rule nonneg-cexprD, intro merge-in-state-measure[OF - ϱ]*)
with $\langle \text{subprob-cexpr } (\text{set } vs) (\text{set } vs') \Gamma \delta \rangle$ **and** ϱ
show $(\int^+ \sigma. \text{ennreal } (\text{norm } (?f \sigma)) \partial ?M) < \infty$ **unfolding** *subprob-cexpr-def*
by (*auto simp: less-top[symmetric] top-unique cong: nn-integral-cong*)
qed

8.3 Randomfree expressions

Translates an expression with no occurrences of Random or Fail into an equivalent target language expression.

primrec *expr-rf-to-cexpr* :: *expr \Rightarrow cexpr* **where**
 $\text{expr-rf-to-cexpr } (\text{Val } v) = \text{CVal } v$
 $\text{expr-rf-to-cexpr } (\text{Var } x) = \text{CVar } x$
 $\text{expr-rf-to-cexpr } \langle e1, e2 \rangle = \langle \text{expr-rf-to-cexpr } e1, \text{expr-rf-to-cexpr } e2 \rangle_c$
 $\text{expr-rf-to-cexpr } (\text{oper } \$\$ e) = \text{oper } \$\$_c (\text{expr-rf-to-cexpr } e)$
 $\text{expr-rf-to-cexpr } (\text{IF } b \text{ THEN } e1 \text{ ELSE } e2) =$
 $(\text{IF}_c \text{expr-rf-to-cexpr } b \text{ THEN } \text{expr-rf-to-cexpr } e1 \text{ ELSE } \text{expr-rf-to-cexpr } e2)$
 $\text{expr-rf-to-cexpr } (\text{LET } e1 \text{ IN } e2) =$
 $\text{cexpr-subst } 0 (\text{expr-rf-to-cexpr } e1) (\text{expr-rf-to-cexpr } e2)$
 $\text{expr-rf-to-cexpr } (\text{Random } -) = \text{undefined}$
 $\text{expr-rf-to-cexpr } (\text{Fail } -) = \text{undefined}$

lemma *cexpr-sem-expr-rf-to-cexpr*:
 $\text{randomfree } e \implies \text{cexpr-sem } \sigma (\text{expr-rf-to-cexpr } e) = \text{expr-sem-rf } \sigma e$
by (*induction e arbitrary: σ*) (*auto simp: cexpr-sem-cexpr-subst*)

lemma *cexpr-typing-expr-rf-to-cexpr*[*intro*]:
assumes $\Gamma \vdash e : t$ *randomfree e*
shows $\Gamma \vdash_c \text{expr-rf-to-cexpr } e : t$
using *assms* **by** (*induction rule: expr-typing.induct*) (*auto intro!: cexpr-typing.intros*)

lemma *free-vars-expr-rf-to-cexpr*:
randomfree $e \implies \text{free-vars } (\text{expr-rf-to-cexpr } e) \subseteq \text{free-vars } e$
proof (*induction* e)
case (*LetVar* $e1\ e2$)
thus *?case*
by (*simp only: free-vars-cexpr.simps expr-rf-to-cexpr.simps,*
intro order.trans[OF free-vars-cexpr-subst]) *auto*
qed *auto*

8.4 Builtin density expressions

primrec *dist-dens-cexpr* :: *pdf-dist* \Rightarrow *cexpr* \Rightarrow *cexpr* \Rightarrow *cexpr* **where**
dist-dens-cexpr Bernoulli $p\ x = (\text{IF}_c\ \text{CReal } 0 \leq_c p \wedge_c p \leq_c\ \text{CReal } 1\ \text{THEN}$
 $\quad \text{IF}_c\ x\ \text{THEN } p\ \text{ELSE } \text{CReal } 1 -_c p$
 $\quad \text{ELSE } \text{CReal } 0)$
| *dist-dens-cexpr UniformInt* $p\ x = (\text{IF}_c\ \text{fst}_c\ p \leq_c\ \text{snd}_c\ p \wedge_c\ \text{fst}_c\ p \leq_c\ x \wedge_c\ x \leq_c$
 $\text{snd}_c\ p\ \text{THEN}$
 $\quad \text{inverse}_c\ (\langle \text{snd}_c\ p -_c\ \text{fst}_c\ p +_c\ \text{CInt } 1 \rangle_c)\ \text{ELSE}$
 $\text{CReal } 0)$
| *dist-dens-cexpr UniformReal* $p\ x = (\text{IF}_c\ \text{fst}_c\ p <_c\ \text{snd}_c\ p \wedge_c\ \text{fst}_c\ p \leq_c\ x \wedge_c\ x \leq_c$
 $\text{snd}_c\ p\ \text{THEN}$
 $\quad \text{inverse}_c\ (\text{snd}_c\ p -_c\ \text{fst}_c\ p)\ \text{ELSE } \text{CReal } 0)$
| *dist-dens-cexpr Gaussian* $p\ x = (\text{IF}_c\ \text{CReal } 0 <_c\ \text{snd}_c\ p\ \text{THEN}$
 $\quad \text{exp}_c\ (-_c((x -_c\ \text{fst}_c\ p) \wedge_c\ \text{CInt } 2 /_c (\text{CReal } 2 *_c\ \text{snd}_c$
 $p \wedge_c\ \text{CInt } 2))) /_c$
 $\quad \text{sqrt}_c (\text{CReal } 2 *_c\ \pi_c *_c\ \text{snd}_c\ p \wedge_c\ \text{CInt } 2)\ \text{ELSE}$
 $\text{CReal } 0)$
| *dist-dens-cexpr Poisson* $p\ x = (\text{IF}_c\ \text{CReal } 0 <_c\ p \wedge_c\ \text{CInt } 0 \leq_c\ x\ \text{THEN}$
 $\quad p \wedge_c\ x /_c \langle \text{fact}_c\ x \rangle_c *_c\ \text{exp}_c\ (-_c\ p)\ \text{ELSE } \text{CReal } 0)$

lemma *free-vars-dist-dens-cexpr*:
 $\text{free-vars } (\text{dist-dens-cexpr } \text{dst } e1\ e2) \subseteq \text{free-vars } e1 \cup \text{free-vars } e2$
by (*subst dist-dens-cexpr-def, cases dst*) *simp-all*

lemma *cexpr-typing-dist-dens-cexpr*:
assumes $\Gamma \vdash_c e1 : \text{dist-param-type } \text{dst } \Gamma \vdash_c e2 : \text{dist-result-type } \text{dst}$
shows $\Gamma \vdash_c \text{dist-dens-cexpr } \text{dst } e1\ e2 : \text{REAL}$
using *assms*
apply (*subst dist-dens-cexpr-def, cases dst*)

apply (*simp, intro cet-op-intros cet-if cet-val' cet-var' cet-eq, simp-all*) []

apply (*simp, intro cet-if cet-and cet-or cet-less-int cet-eq*)

apply (*erule cet-fst cet-snd | simp*) $+$

apply (*rule cet-inverse, rule cet-op[where $t = \text{INTEG}$], intro cet-add-int cet-minus-int*)

apply (*simp-all add: cet-val' cet-fst cet-snd*) [5]

apply (*simp, intro cet-if cet-op-intros cet-eq cet-fst cet-snd, simp-all add: cet-val'*)

□

apply (*simp*, *intro cet-if cet-and*, *rule cet-less-real*, *simp add: cet-val'*, *simp*)
apply (*rule cet-less-eq-int*, *simp add: cet-val'*, *simp*)
apply (*intro cet-mult-real cet-pow-real cet-inverse cet-cast-real-int cet-exp cet-minus-real*
cet-op[**where** *oper = Fact and t = INTEG*] *cet-var'*, *simp-all add:*
cet-val') [2]

apply (*simp*, *intro cet-if cet-op-intros cet-val'*, *simp-all add: cet-fst cet-snd*)
done

lemma *val-type-eq-BOOL*: *val-type x = BOOL* \longleftrightarrow *x* \in *BoolVal'UNIV*
by (*cases x*) *auto*

lemma *val-type-eq-INTEG*: *val-type x = INTEG* \longleftrightarrow *x* \in *IntVal'UNIV*
by (*cases x*) *auto*

lemma *val-type-eq-PRODUCT*: *val-type x = PRODUCT t1 t2* \longleftrightarrow
($\exists a b.$ *val-type a = t1* \wedge *val-type b = t2* \wedge *x = <| a, b |>*)
by (*cases x*) *auto*

lemma *cepr-sem-dist-dens-cepr-nonneg*:
assumes $\Gamma \vdash_c e1 : \text{dist-param-type } dst$ $\Gamma \vdash_c e2 : \text{dist-result-type } dst$
assumes *free-vars e1* \subseteq *V* *free-vars e2* \subseteq *V*
assumes $\sigma \in \text{space } (\text{state-measure } V \Gamma)$
shows *ennreal* (*extract-real* (*cepr-sem* σ (*dist-dens-cepr* *dst* *e1* *e2*))) =
dist-dens *dst* (*cepr-sem* σ *e1*) (*cepr-sem* σ *e2*) \wedge
 $0 \leq \text{extract-real } (\text{cepr-sem } \sigma (\text{dist-dens-cepr } \text{dst } e1 e2))$

proof –

from *val-type-cepr-sem*[*OF assms(1,3,5)*] **and** *val-type-cepr-sem*[*OF assms(2,4,5)*]
have *cepr-sem* σ *e1* \in *space* (*stock-measure* (*dist-param-type* *dst*)) **and**
cepr-sem σ *e2* \in *space* (*stock-measure* (*dist-result-type* *dst*))
by (*auto simp: type-universe-def simp del: type-universe-type*)
thus *?thesis*
by (*subst dist-dens-cepr-def*, *cases dst*)
(*auto simp:*
lift-Comp-def lift-RealVal-def lift-RealIntVal-def lift-RealIntVal2-def
bernoulli-density-def val-type-eq-REAL val-type-eq-BOOL val-type-eq-PRODUCT
val-type-eq-INTEG
uniform-int-density-def uniform-real-density-def
lift-IntVal-def poisson-density'-def one-ennreal-def
field-simps gaussian-density-def)

qed

lemma *cepr-sem-dist-dens-cepr*:
assumes $\Gamma \vdash_c e1 : \text{dist-param-type } dst$ $\Gamma \vdash_c e2 : \text{dist-result-type } dst$
assumes *free-vars e1* \subseteq *V* *free-vars e2* \subseteq *V*
assumes $\sigma \in \text{space } (\text{state-measure } V \Gamma)$

shows $\text{ennreal } (\text{extract-real } (\text{cexpr-sem } \sigma (\text{dist-dens-cexpr } \text{dst } e1 \ e2))) =$
 $\text{dist-dens } \text{dst } (\text{cexpr-sem } \sigma \ e1) (\text{cexpr-sem } \sigma \ e2)$
using $\text{cexpr-sem-dist-dens-cexpr-nonneg}[OF \ \text{assms}]$ **by** simp

lemma $\text{nonneg-dist-dens-cexpr}$:

assumes $\Gamma \vdash_c e1 : \text{dist-param-type } \text{dst}$ $\Gamma \vdash_c e2 : \text{dist-result-type } \text{dst}$

assumes $\text{free-vars } e1 \subseteq V$ $\text{free-vars } e2 \subseteq V$

shows $\text{nonneg-cexpr } V \ \Gamma (\text{dist-dens-cexpr } \text{dst } e1 \ e2)$

proof $(\text{intro } \text{nonneg-cexprI})$

fix σ **assume** $g: \sigma \in \text{space } (\text{state-measure } V \ \Gamma)$

from $\text{cexpr-sem-dist-dens-cexpr-nonneg}[OF \ \text{assms } \text{this}]$

show $0 \leq \text{extract-real } (\text{cexpr-sem } \sigma (\text{dist-dens-cexpr } \text{dst } e1 \ e2))$

by simp

qed

8.5 Integral expressions

definition $\text{integrate-var} :: \text{tyenv} \Rightarrow \text{vname} \Rightarrow \text{cexpr} \Rightarrow \text{cexpr}$ **where**

$\text{integrate-var } \Gamma \ v \ e = \int_c \text{map-vars } (\lambda w. \text{if } v = w \text{ then } 0 \text{ else } \text{Suc } w) \ e \ \partial(\Gamma \ v)$

definition $\text{integrate-vars} :: \text{tyenv} \Rightarrow \text{vname list} \Rightarrow \text{cexpr} \Rightarrow \text{cexpr}$ **where**

$\text{integrate-vars } \Gamma = \text{foldr } (\text{integrate-var } \Gamma)$

lemma $\text{cexpr-sem-integrate-var}$:

$\text{cexpr-sem } \sigma (\text{integrate-var } \Gamma \ v \ e) =$

$\text{RealVal } (\int x. \text{extract-real } (\text{cexpr-sem } (\sigma(v := x)) \ e) \ \partial\text{stock-measure } (\Gamma \ v))$

proof –

let $?f = (\lambda w. \text{if } v = w \text{ then } 0 \text{ else } \text{Suc } w)$

have $\text{cexpr-sem } \sigma (\text{integrate-var } \Gamma \ v \ e) =$

$\text{RealVal } (\int x. \text{extract-real } (\text{cexpr-sem } (\text{case-nat } x \ \sigma \circ ?f) \ e) \ \partial\text{stock-measure}$

$(\Gamma \ v))$

by $(\text{simp } \text{add: } \text{extract-real-def } \text{integrate-var-def } \text{cexpr-sem-map-vars})$

also have $(\lambda x. \text{case-nat } x \ \sigma \circ ?f) = (\lambda x. \sigma(v := x))$

by $(\text{intro } \text{ext}) (\text{simp } \text{add: } \text{o-def } \text{split: } \text{if-split})$

finally show $?thesis$.

qed

lemma $\text{cexpr-sem-integrate-var}'$:

$\text{extract-real } (\text{cexpr-sem } \sigma (\text{integrate-var } \Gamma \ v \ e)) =$

$(\int x. \text{extract-real } (\text{cexpr-sem } (\sigma(v := x)) \ e) \ \partial\text{stock-measure } (\Gamma \ v))$

by $(\text{subst } \text{cexpr-sem-integrate-var}, \text{simp } \text{add: } \text{extract-real-def})$

lemma $\text{cexpr-typing-integrate-var}[\text{simp}]$:

$\Gamma \vdash_c e : \text{REAL} \Longrightarrow \Gamma \vdash_c \text{integrate-var } \Gamma \ v \ e : \text{REAL}$

unfolding integrate-var-def

by $(\text{rule } \text{cexpr-typing.intros}, \text{rule } \text{cexpr-typing-map-vars})$

$(\text{erule } \text{cexpr-typing-cong}', \text{simp } \text{split: } \text{nat.split})$

lemma $\text{cexpr-typing-integrate-vars}[\text{simp}]$:

$\Gamma \vdash_c e : REAL \implies \Gamma \vdash_c \text{integrate-vars } \Gamma \text{ vs } e : REAL$
by (*induction vs arbitrary: e*)
(*simp-all add: integrate-vars-def*)

lemma *free-vars-integrate-var*[*simp*]:
 $\text{free-vars } (\text{integrate-var } \Gamma \ v \ e) = \text{free-vars } e - \{v\}$
by (*auto simp: integrate-var-def*)

lemma *free-vars-integrate-vars*[*simp*]:
 $\text{free-vars } (\text{integrate-vars } \Gamma \ \text{vs } \ e) = \text{free-vars } e - \text{set } \text{vs}$
by (*induction vs arbitrary: e*) (*auto simp: integrate-vars-def*)

lemma (*in product-sigma-finite*) *product-integral-insert'*:
fixes $f :: - \Rightarrow \text{real}$
assumes *finite* $I \ i \notin I$ *integrable* $(\text{Pi}_M (\text{insert } i \ I) \ M) \ f$
shows $\text{integral}^L (\text{Pi}_M (\text{insert } i \ I) \ M) \ f = \text{LINT } y | M \ i. \ \text{LINT } x | \text{Pi}_M \ I \ M. \ f \ (x(i := y))$
:= y)
proof –
interpret *pair-sigma-finite* $M \ i \ \text{Pi}_M \ I \ M$
by (*simp-all add: sigma-finite-assms pair-sigma-finite-def sigma-finite-measures*)
interpret M_i : *sigma-finite-measure* $M \ i$
by (*simp add: assms sigma-finite-measures*)
from *assms*(3) **have** *int*: *integrable* $(M \ i \ \otimes_M \ \text{Pi}_M \ I \ M) \ (\lambda(x, y). \ f \ (y(i := x)))$
unfolding *real-integrable-def*
apply (*elim conjE*)
apply (*subst* (1 2) *nn-integral-snd*[*symmetric*])
apply ((*subst* (*asm*) (1 2) *product-nn-integral-insert*[*OF assms*(1,2)]),
auto intro!: *measurable-compose*[*OF - measurable-ennreal*] *borel-measurable-uminus*)
[])+
done
from *assms* **have** $\text{integral}^L (\text{Pi}_M (\text{insert } i \ I) \ M) \ f = \text{LINT } x | \text{Pi}_M \ I \ M. \ \text{LINT } y | M \ i. \ f \ (x(i := y))$
by (*rule product-integral-insert*)
also from *int* **have** $\dots = \text{LINT } y | M \ i. \ \text{LINT } x | \text{Pi}_M \ I \ M. \ f \ (x(i := y))$
by (*rule Fubini-integral*)
finally show *?thesis* .
qed

lemma *cepr-sem-integrate-vars*:
assumes $\varrho : \varrho \in \text{space } (\text{state-measure } V' \ \Gamma)$
assumes *disjoint*: *distinct vs set vs* $\cap V' = \{\}$
assumes *integrable* $(\text{state-measure } (\text{set } \text{vs}) \ \Gamma)$
 $(\lambda \sigma. \ \text{extract-real } (\text{cepr-sem } (\text{merge } (\text{set } \text{vs}) \ V' \ (\sigma, \varrho)) \ e))$
assumes $e : \Gamma \vdash_c e : REAL$ *free-vars* $e \subseteq \text{set } \text{vs} \cup V'$
shows $\text{extract-real } (\text{cepr-sem } \varrho \ (\text{integrate-vars } \Gamma \ \text{vs } \ e)) =$
 $\int \sigma. \ \text{extract-real } (\text{cepr-sem } (\text{merge } (\text{set } \text{vs}) \ V' \ (\sigma, \varrho)) \ e) \ \partial \text{state-measure}$
 $(\text{set } \text{vs}) \ \Gamma$
using *assms*

```

proof (induction vs arbitrary:  $\varrho$   $V'$ )
  case Nil
  hence  $\bigwedge v. (if\ v \in V' \text{ then } \varrho\ v \text{ else undefined}) = \varrho\ v$ 
    by (auto simp: state-measure-def space-PiM)
  thus ?case by (auto simp: integrate-vars-def state-measure-def merge-def PiM-empty)
next
  case (Cons v vs  $\varrho$   $V'$ )
  interpret product-sigma-finite  $\lambda v. \text{stock-measure } (\Gamma\ v)$ 
    by (simp add: product-sigma-finite-def)
  interpret sigma-finite-measure state-measure (set vs)  $\Gamma$ 
    by (simp add: sigma-finite-state-measure)
  have  $\varrho'$ :  $\bigwedge x. x \in \text{type-universe } (\Gamma\ v) \implies \varrho(v := x) \in \text{space } (\text{state-measure } (\text{insert } v\ V')\ \Gamma)$ 
  using Cons.prem1 by (auto simp: state-measure-def space-PiM split: if-split-asm)
  have extract-real (cepr-sem  $\varrho$  (integrate-vars  $\Gamma$  (v # vs) e)) =
     $\int x. \text{extract-real } (\text{cepr-sem } (\varrho(v := x)) (\text{integrate-vars } \Gamma\ vs\ e))\ \partial \text{stock-measure } (\Gamma\ v)$ 
    (is - = ?I) by (simp add: integrate-vars-def cepr-sem-integrate-var extract-real-def)
  also from Cons.prem4 have int: integrable (state-measure (insert v (set vs))  $\Gamma$ )
    ( $\lambda \sigma. \text{extract-real } (\text{cepr-sem } (\text{merge } (\text{insert } v\ (\text{set } vs))\ V'\ (\sigma, \varrho))\ e)$ ) by simp
  have AE x in stock-measure ( $\Gamma\ v$ ).
    extract-real (cepr-sem ( $\varrho(v := x)$ ) (integrate-vars  $\Gamma$  vs e)) =
     $\int \sigma. \text{extract-real } (\text{cepr-sem } (\text{merge } (\text{set } vs)\ (\text{insert } v\ V')\ (\sigma, \varrho(v := x))))\ e$ 
     $\partial \text{state-measure } (\text{set } vs)\ \Gamma$ 
    apply (rule AE-mp[OF - AE-I2[OF impI]])
    apply (rule integrable-cepr-projection[OF - - - - - int])
    apply (insert Cons.prem5, auto) [7]
    apply (subst Cons.IH, rule  $\varrho'$ , insert Cons.prem5, auto)
    done
  hence ?I =  $\int x. \int \sigma. \text{extract-real } (\text{cepr-sem } (\text{merge } (\text{set } vs)\ (\text{insert } v\ V')\ (\sigma, \varrho(v := x))))\ e$ 
     $\partial \text{state-measure } (\text{set } vs)\ \Gamma\ \partial \text{stock-measure } (\Gamma\ v)$  using Cons.prem5
    apply (intro integral-cong-AE)
    apply (rule measurable-compose[OF measurable-Pair-compose-split[OF measurable-fun-upd-state-measure[of v V'  $\Gamma$ ]])
    apply (simp, simp, simp, rule measurable-compose[OF - measurable-extract-real])
    apply (rule measurable-cepr-sem, simp, (auto) [])
    apply (rule borel-measurable-lebesgue-integral)
    apply (subst measurable-split-conv)
    apply (rule measurable-compose[OF - measurable-extract-real])
    apply (rule measurable-compose[OF - measurable-cepr-sem[of  $\Gamma$  - - set vs  $\cup$  insert v V']])
    apply (unfold state-measure-def, rule measurable-compose[OF - measurable-merge])
    apply simp-all
    done
  also have ( $\lambda x\ \sigma. \text{merge } (\text{set } vs)\ (\text{insert } v\ V')\ (\sigma, \varrho(v := x))$ ) =
    ( $\lambda x\ \sigma. \text{merge } (\text{set } (v \# vs))\ V'\ (\sigma(v := x), \varrho)$ )

```

using *Cons.prem*s **by** (*intro ext*) (*auto simp: merge-def split: if-split*)
also have $(\int x. \int \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } (v\#vs))) V' (\sigma(v := x), \varrho)) e)$
 $\quad \partial\text{state-measure } (\text{set } vs) \Gamma \partial\text{stock-measure } (\Gamma v) =$
 $\quad \int \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } (v\#vs))) V' (\sigma, \varrho)) e$
 $\quad \partial\text{state-measure } (\text{set } (v\#vs)) \Gamma$
using *Cons.prem*s **unfolding** *state-measure-def*
by (*subst* (ϱ) *set-simps*, *subst product-integral-insert'*) *simp-all*
finally show *?case* .
qed

lemma *cexpr-sem-integrate-vars'*:

assumes $\varrho: \varrho \in \text{space } (\text{state-measure } V' \Gamma)$
assumes *disjoint*: $\text{distinct } vs \text{ set } vs \cap V' = \{\}$
assumes *nonneg*: $\text{nonneg-cexpr } (\text{set } vs \cup V') \Gamma e$
assumes *integrable* (*state-measure* (*set* *vs*) Γ)
 $(\lambda\sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } vs) V' (\sigma, \varrho)) e))$
assumes $e: \Gamma \vdash_c e : \text{REAL free-vars } e \subseteq \text{set } vs \cup V'$
shows $\text{ennreal } (\text{extract-real } (\text{cexpr-sem } \varrho (\text{integrate-vars } \Gamma vs e))) =$
 $\int^+ \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } vs) V' (\sigma, \varrho)) e) \partial\text{state-measure}$
 $(\text{set } vs) \Gamma$
proof –
from *assms* **have** $\text{extract-real } (\text{cexpr-sem } \varrho (\text{integrate-vars } \Gamma vs e)) =$
 $\int \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } vs) V' (\sigma, \varrho)) e) \partial\text{state-measure } (\text{set}$
 $vs) \Gamma$
by (*intro cexpr-sem-integrate-vars*)
also have $\text{ennreal } \dots =$
 $\int^+ \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } vs) V' (\sigma, \varrho)) e) \partial\text{state-measure } (\text{set}$
 $vs) \Gamma$
using *assms*
by (*intro nn-integral-eq-integral[symmetric] AE-I2*)
 $(\text{auto intro!}: \text{nonneg-cexprD } \text{merge-in-state-measure})$
finally show *?thesis* .
qed

lemma *nonneg-cexpr-sem-integrate-vars*:

assumes $\varrho: \varrho \in \text{space } (\text{state-measure } V' \Gamma)$
assumes *disjoint*: $\text{distinct } vs \text{ set } vs \cap V' = \{\}$
assumes *nonneg*: $\text{nonneg-cexpr } (\text{set } vs \cup V') \Gamma e$
assumes $e: \Gamma \vdash_c e : \text{REAL free-vars } e \subseteq \text{set } vs \cup V'$
shows $\text{extract-real } (\text{cexpr-sem } \varrho (\text{integrate-vars } \Gamma vs e)) \geq 0$
using *assms*
proof (*induction vs arbitrary: \varrho V'*)
case *Nil*
hence $\bigwedge v. (\text{if } v \in V' \text{ then } \varrho v \text{ else undefined}) = \varrho v$
by (*auto simp: state-measure-def space-PiM*)
with *Nil* **show** *?case*
by (*auto simp: integrate-vars-def state-measure-def merge-def PiM-empty non-*
 neg-cexprD)

next
case (*Cons v vs ρ V'*)
have $\rho': \bigwedge x. x \in \text{type-universe } (\Gamma v) \implies \rho(v := x) \in \text{space } (\text{state-measure } (\text{insert } v V') \Gamma)$
using *Cons.premis(1)* **by** (*auto simp: state-measure-def space-PiM split: if-split-asm*)
have $\text{extract-real } (\text{cexpr-sem } \rho (\text{integrate-vars } \Gamma (v \# vs) e)) =$
 $\int x. \text{extract-real } (\text{cexpr-sem } (\rho(v := x)) (\text{integrate-vars } \Gamma vs e)) \partial \text{stock-measure}$
 (Γv)
by (*simp add: integrate-vars-def cexpr-sem-integrate-var extract-real-def*)
also have $\dots \geq 0$
by (*rule integral-nonneg-AE, rule AE-I2, subst Cons.IH[OF ρ']*) (*insert Cons.premis, auto*)
finally show $\text{extract-real } (\text{cexpr-sem } \rho (\text{integrate-vars } \Gamma (v \# vs) e)) \geq 0 .$
qed

lemma *nonneg-cexpr-sem-integrate-vars'*:
 $\text{distinct } vs \implies \text{set } vs \cap V' = \{\} \implies \text{nonneg-cexpr } (\text{set } vs \cup V') \Gamma e \implies \Gamma \vdash_c e$
 $: \text{REAL} \implies$
 $\text{free-vars } e \subseteq \text{set } vs \cup V' \implies \text{nonneg-cexpr } V' \Gamma (\text{integrate-vars } \Gamma vs e)$
apply (*intro nonneg-cexprI allI*)
apply (*rule nonneg-cexpr-sem-integrate-vars[where V'=V']*)
apply *auto*
done

lemma *cexpr-sem-integral-nonneg*:
assumes *finite*: $(\int^+ x. \text{extract-real } (\text{cexpr-sem } (\text{case-nat } x \sigma) e) \partial \text{stock-measure } t) < \infty$
assumes *nonneg*: $\text{nonneg-cexpr } (\text{shift-var-set } V) (\text{case-nat } t \Gamma) e$
assumes *t*: $\text{case-nat } t \Gamma \vdash_c e : \text{REAL}$ **and** *vars*: $\text{free-vars } e \subseteq \text{shift-var-set } V$
assumes $\rho: \sigma \in \text{space } (\text{state-measure } V \Gamma)$
shows $\text{ennreal } (\text{extract-real } (\text{cexpr-sem } \sigma (\int_c e \partial t))) =$
 $\int^+ x. \text{extract-real } (\text{cexpr-sem } (\text{case-nat } x \sigma) e) \partial \text{stock-measure } t$

proof –
let $?f = \lambda x. \text{extract-real } (\text{cexpr-sem } (\text{case-nat } x \sigma) e)$
have *meas*: $?f \in \text{borel-measurable } (\text{stock-measure } t)$
apply (*rule measurable-compose[OF - measurable-extract-real]*)
apply (*rule measurable-compose[OF measurable-case-nat' measurable-cexpr-sem]*)
apply (*rule measurable-ident-sets[OF refl], rule measurable-const[OF ρ]*)
apply (*simp-all add: t vars*)
done
from this and finite and nonneg have *int*: $\text{integrable } (\text{stock-measure } t) ?f$
by (*auto intro!: integrableI-nonneg nonneg-cexprD case-nat-in-state-measure[OF - ρ]*)

have $\text{extract-real } (\text{cexpr-sem } \sigma (\int_c e \partial t)) =$
 $\int x. \text{extract-real } (\text{cexpr-sem } (\text{case-nat } x \sigma) e) \partial \text{stock-measure } t$
by (*simp add: extract-real-def*)
also have $\text{ennreal } \dots = \int^+ x. \text{extract-real } (\text{cexpr-sem } (\text{case-nat } x \sigma) e) \partial \text{stock-measure } t$

by (*subst nn-integral-eq-integral*[*OF int AE-I2*])
 (*auto intro!*: *nonneg-cexprD*[*OF nonneg*] *case-nat-in-state-measure*[*OF - ρ*])
 finally show *?thesis* .
 qed

lemma *has-parametrized-subprob-density-cexpr-sem-integral*:
 assumes *dens*: *has-parametrized-subprob-density* (*state-measure V' Γ*) *M* (*stock-measure t*)
 ($\lambda \rho x. \int^+ y. \text{eval-cexpr } f \text{ (case-nat } x \ \rho) \ y \ \partial \text{stock-measure } t'$)
 assumes *nonneg*: *nonneg-cexpr* (*shift-var-set* (*shift-var-set V'*)) (*case-nat t'* (*case-nat t Γ*)) *f*
 assumes *tf*: *case-nat t'* (*case-nat t Γ*) $\vdash_c f : \text{REAL}$
 assumes *varsf*: *free-vars f* \subseteq *shift-var-set* (*shift-var-set V'*)
 assumes *ρ*: $\rho \in \text{space (state-measure } V' \ \Gamma)$
 shows *AE x in stock-measure t*.
 ($\int^+ y. \text{eval-cexpr } f \text{ (case-nat } x \ \rho) \ y \ \partial \text{stock-measure } t'$) = *ennreal* (*eval-cexpr* ($\int_c f \ \partial t'$) $\rho \ x$)
proof (*rule AE-mp*[*OF - AE-I2*[*OF impI*]])
 interpret *sigma-finite-measure stock-measure t'* by *simp*
 let *?f* = $\lambda x. \int^+ y. \text{eval-cexpr } f \text{ (case-nat } x \ \rho) \ y \ \partial \text{stock-measure } t'$
 from *has-parametrized-subprob-density-integral*[*OF dens ρ*]
 have ($\int^+ x. ?f \ x \ \partial \text{stock-measure } t$) $\neq \infty$ by (*auto simp: eval-cexpr-def top-unique*)
 thus *AE x in stock-measure t. ?f x $\neq \infty$* using ρ *tf varsf* by (*intro nn-integral-PInf-AE*)
simp-all
 fix *x* assume *x*: $x \in \text{space (stock-measure } t)$ and *finite*: $?f \ x \neq \infty$
 have *nonneg'*: *AE y in stock-measure t'. eval-cexpr f (case-nat x ρ) y ≥ 0*
 unfolding *eval-cexpr-def* using $\rho \ x$
 by (*intro AE-I2 nonneg-cexprD*[*OF nonneg*]) (*auto intro!*: *case-nat-in-state-measure*)
 hence *integrable (stock-measure t')* ($\lambda y. \text{eval-cexpr } f \text{ (case-nat } x \ \rho) \ y$)
 using $x \ \rho$ *tf varsf finite* by (*intro integrableI-nonneg*) (*simp-all add: top-unique less-top*)
 thus $?f \ x = \text{ennreal (eval-cexpr (}\int_c f \ \partial t') \ \rho \ x)$ using *nonneg'*
 by (*simp add: extract-real-def nn-integral-eq-integral eval-cexpr-def*)
 qed

end

9 Concrete Density Contexts

theory *PDF-Target-Density-Contexts*
imports *PDF-Density-Contexts PDF-Target-Semantics*
begin

9.1 Definition

type-synonym *cdens-ctxt* = *vname list* \times *vname list* \times *tyenv* \times *cexpr*

definition *dens-ctxt-α* :: *cdens-ctxt* \Rightarrow *dens-ctxt* **where**
dens-ctxt-α $\equiv \lambda(vs, vs', \Gamma, \delta). (\text{set } vs, \text{set } vs', \Gamma, \lambda\sigma. \text{extract-real (cexpr-sem } \sigma \ \delta))$

definition *shift-vars* :: *nat list* \Rightarrow *nat list* **where**

shift-vars *vs* = 0 # map *Suc* *vs*

lemma *set-shift-vars*[*simp*]: *set* (*shift-vars* *vs*) = *shift-var-set* (*set* *vs*)

unfolding *shift-vars-def* *shift-var-set-def* **by** *simp*

definition *is-density-expr* :: *cdens-ctxt* \Rightarrow *pdf-type* \Rightarrow *cexpr* \Rightarrow *bool* **where**

is-density-expr $\equiv \lambda(vs, vs', \Gamma, \delta) t e.$

case-nat *t* $\Gamma \vdash_c e : REAL \wedge$

free-vars *e* \subseteq *shift-var-set* (*set* *vs'*) \wedge

nonneg-cexpr (*shift-var-set* (*set* *vs'*)) (*case-nat* *t* Γ) *e*

lemma *is-density-exprI*:

case-nat *t* $\Gamma \vdash_c e : REAL \implies$

free-vars *e* \subseteq *shift-var-set* (*set* *vs'*) \implies

nonneg-cexpr (*shift-var-set* (*set* *vs'*)) (*case-nat* *t* Γ) *e* \implies

is-density-expr (*vs*, *vs'*, Γ , δ) *t e*

unfolding *is-density-expr-def* **by** *simp*

lemma *is-density-exprD*:

assumes *is-density-expr* (*vs*, *vs'*, Γ , δ) *t e*

shows *case-nat* *t* $\Gamma \vdash_c e : REAL$ *free-vars* *e* \subseteq *shift-var-set* (*set* *vs'*)

and *is-density-exprD-nonneg*: *nonneg-cexpr* (*shift-var-set* (*set* *vs'*)) (*case-nat* *t* Γ) *e*

using *assms* **unfolding** *is-density-expr-def* **by** *simp-all*

lemma *density-context- α* :

assumes *cdens-ctxt-invar* *vs vs' Γ δ*

shows *density-context* (*set* *vs*) (*set* *vs'*) Γ ($\lambda\sigma. \text{extract-real} (\text{cexpr-sem } \sigma \delta)$)

proof (*unfold* *density-context-def*, *intro* *allI* *ballI* *conjI* *impI* *subprob-spaceI*)

show ($\lambda x. \text{ennreal} (\text{extract-real} (\text{cexpr-sem } x \delta))$)

\in *borel-measurable* (*state-measure* (*set* *vs* \cup *set* *vs'*) Γ)

apply (*intro* *measurable-compose*[*OF* - *measurable-ennreal*] *measurable-compose*[*OF* - *measurable-extract-real*])

apply (*insert* *cdens-ctxt-invarD*[*OF* *assms*], *auto*)

done

note [*measurable*] = *this*

fix ϱ **assume** ϱ : $\varrho \in \text{space} (\text{state-measure} (\text{set } vs') \Gamma)$

let $?M = \text{dens-ctxt-measure} (\text{set } vs, \text{set } vs', \Gamma, \lambda x. \text{ennreal} (\text{extract-real} (\text{cexpr-sem } x \delta))) \varrho$

from ϱ **have** ($\lambda\sigma. \text{merge} (\text{set } vs) (\text{set } vs') (\sigma, \varrho)$)

\in *measurable* (*state-measure* (*set* *vs*) Γ) (*state-measure* (*set* *vs* \cup

set *vs'*) Γ)

unfolding *state-measure-def* **by** *simp*

hence *emeasure* $?M$ (*space* $?M$) =

$\int^+ x. \text{ennreal} (\text{extract-real} (\text{cexpr-sem} (\text{merge} (\text{set } vs) (\text{set } vs') (x, \varrho)))$

δ))
 ∂ state-measure (set vs) Γ (is - = ?I)
using ρ **unfolding** dens-ctxt-measure-def state-measure'-def
by (simp add: emeasure-density nn-integral-distr, intro nn-integral-cong)
(simp-all split: split-indicator add: merge-in-state-measure)
also from cdens-ctxt-invarD[OF assms] **have** subprob-cexpr (set vs) (set vs') Γ
 δ **by** simp
with ρ **have** ?I \leq 1 **unfolding** subprob-cexpr-def **by** blast
finally show emeasure ?M (space ?M) \leq 1 .
qed (insert cdens-ctxt-invarD[OF assms], simp-all add: nonneg-cexpr-def)

9.2 Expressions for density context operations

definition marg-dens-cexpr :: tyenv \Rightarrow vname list \Rightarrow vname \Rightarrow cexpr \Rightarrow cexpr
where

marg-dens-cexpr Γ vs x e =
map-vars (λy . if $y = x$ then 0 else Suc y) (integrate-vars Γ (filter (λy . $y \neq x$)
vs) e)

lemma free-vars-marg-dens-cexpr:

assumes cdens-ctxt-invar vs vs' Γ δ

shows free-vars (marg-dens-cexpr Γ vs x δ) \subseteq shift-var-set (set vs')

proof –

have free-vars (marg-dens-cexpr Γ vs x δ) \subseteq shift-var-set (free-vars δ – set vs)

unfolding marg-dens-cexpr-def shift-var-set-def **by** auto

also from cdens-ctxt-invarD[OF assms] **have** ... \subseteq shift-var-set (set vs')

unfolding shift-var-set-def **by** auto

finally show ?thesis .

qed

lemma cexpr-typing-marg-dens-cexpr[*intro*]:

$\Gamma \vdash_c \delta : REAL \implies$ case-nat (Γ x) $\Gamma \vdash_c$ marg-dens-cexpr Γ vs x $\delta : REAL$

unfolding marg-dens-cexpr-def

by (rule cexpr-typing-map-vars, rule cexpr-typing-cong', erule cexpr-typing-integrate-vars)
simp

lemma cexpr-sem-marg-dens:

assumes cdens-ctxt-invar vs vs' Γ δ

assumes x: $x \in$ set vs **and** ρ : $\rho \in$ space (state-measure (set vs') Γ)

shows AE v in stock-measure (Γ x).

ennreal (extract-real (cexpr-sem (case-nat v ρ) (marg-dens-cexpr Γ vs x
 δ))) =

marg-dens (dens-ctxt- α (vs,vs', Γ , δ)) x ρ v

proof –

note invar = cdens-ctxt-invarD[OF assms(1)]

let ?vs = filter (λy . $y \neq x$) vs

note cdens-ctxt-invar-imp-integrable[OF assms(1) ρ]

moreover from x **have** insert-eq: insert x {xa \in set vs. xa \neq x} = set vs **by**
auto

ultimately have *integrable*:
AE v in stock-measure (Γx).
integrable (*state-measure* (*set ?vs*) Γ)
 $(\lambda\sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } ?vs) (\text{insert } x (\text{set } vs')) (\sigma, \varrho(x := v)))) \delta)$
using *invar x* ϱ **by** (*intro integrable-cexpr-projection*) *auto*

show *?thesis*
proof (*rule AE-mp[OF integrable]*, *rule AE-I2*, *intro impI*)
fix *v* **assume** *v*: $v \in \text{space } (\text{stock-measure } (\Gamma x))$
assume *integrable*:
integrable (*state-measure* (*set ?vs*) Γ)
 $(\lambda\sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } ?vs) (\text{insert } x (\text{set } vs')) (\sigma, \varrho(x := v)))) \delta)$

from *v* **and** ϱ **have** ϱ' : $(\varrho(x := v)) \in \text{space } (\text{state-measure } (\text{set } (x\#vs')) \Gamma)$
by (*auto simp: state-measure-def space-PiM split: if-split-asm*)
have *cexpr-sem* (*case-nat v* ϱ) (*marg-dens-cexpr* Γ *vs* *x* δ) =
cexpr-sem (*case-nat v* $\varrho \circ (\lambda y. \text{if } y = x \text{ then } 0 \text{ else } \text{Suc } y)$)
 $(\text{integrate-vars } \Gamma [y \leftarrow vs . y \neq x] \delta)$ (**is** - = *?A*)
unfolding *marg-dens-cexpr-def* **by** (*simp add: cexpr-sem-map-vars*)
also have $\bigwedge y. y \in \text{free-vars } (\text{integrate-vars } \Gamma [y \leftarrow vs . y \neq x] \delta)$
 $\implies (\text{case-nat } v \varrho \circ (\lambda y. \text{if } y = x \text{ then } 0 \text{ else } \text{Suc } y)) y = (\varrho(x :=$
 $v)) y$
unfolding *o-def* **by** *simp*
hence *?A* = *cexpr-sem* ($\varrho(x := v)$) (*integrate-vars* $\Gamma [y \leftarrow vs . y \neq x] \delta$) **by** (*rule*
cexpr-sem-eq-on-vars)
also from *x* **have** *insert x* $\{xa \in \text{set } vs. xa \neq x\} \cup \text{set } vs' = \text{set } vs \cup \text{set } vs'$
by *auto*
hence *extract-real* (*cexpr-sem* ($\varrho(x := v)$)) (*integrate-vars* $\Gamma [y \leftarrow vs . y \neq x] \delta$)
= $\int^+ \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } ?vs) (\text{insert } x (\text{set } vs')) (\sigma, \varrho(x := v)))) \delta)$
 $\partial \text{state-measure } (\text{set } ?vs) \Gamma$
using ϱ' *invar integrable* **by** (*subst cexpr-sem-integrate-vars'*) (*auto*)
also from *x* **have** $(\lambda\sigma. \text{merge } (\text{set } ?vs) (\text{insert } x (\text{set } vs')) (\sigma, \varrho(x := v))) =$
 $(\lambda\sigma. \text{merge } (\text{set } vs) (\text{set } vs') (\sigma(x := v), \varrho))$
by (*intro ext*) (*auto simp: merge-def*)
also from *x* **have** *set ?vs* = *set vs* - $\{x\}$ **by** *auto*
also have $(\int^+ \sigma. \text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } vs) (\text{set } vs') (\sigma(x := v), \varrho)) \delta)$
 $\partial \text{state-measure } (\text{set } vs - \{x\}) \Gamma) =$
 $\text{marg-dens } (\text{dens-ctxt-}\alpha (vs, vs', \Gamma, \delta)) x \varrho v$
unfolding *marg-dens-def dens-ctxt-}\alpha-def **by** *simp*
finally show *ennreal* (*extract-real* (*cexpr-sem* ($\lambda a. \text{case } a \text{ of } 0 \implies v \mid \text{Suc } a \implies$
 ϱa))
 $(\text{marg-dens-cexpr } \Gamma \text{ vs } x \delta))) =$
 $\text{marg-dens } (\text{dens-ctxt-}\alpha (vs, vs', \Gamma, \delta)) x \varrho v .$*

qed

qed

lemma *nonneg-cexpr-sem-marg-dens*:

assumes *cdens-ctxt-invar* *vs vs' Γ δ*

assumes *x*: $x \in \text{set } vs$ **and** *ϱ* : $\varrho \in \text{space } (\text{state-measure } (\text{set } vs') \Gamma)$

assumes *v*: $v \in \text{type-universe } (\Gamma x)$

shows *extract-real* (*cexpr-sem* (*case-nat* *v ϱ*) (*marg-dens-cexpr* $\Gamma vs x \delta$)) ≥ 0

proof –

note *invar* = *cdens-ctxt-invarD*[*OF* *assms*(1)]

from *assms* **have** *ϱ* : *case-nat* *v ϱ* $\circ (\lambda y. \text{if } y = x \text{ then } 0 \text{ else } \text{Suc } y)$
 $\in \text{space } (\text{state-measure } (\text{set } (x\#vs')) \Gamma)$

by (*force simp: state-measure-def space-PiM o-def split: if-split-asm*)

moreover from *x* **have** *insert* *x* $\{xa \in \text{set } vs. xa \neq x\} \cup \text{set } vs' = \text{set } vs \cup \text{set } vs'$ **by** *auto*

ultimately show *?thesis* **using** *assms invar unfolding marg-dens-cexpr-def*

by (*subst cexpr-sem-map-vars, intro nonneg-cexpr-sem-integrate-vars[of - set (x#vs')]*) *auto*

qed

definition *marg-dens2-cexpr* :: *tyenv* \Rightarrow *vname list* \Rightarrow *vname* \Rightarrow *vname* \Rightarrow *cexpr*
 \Rightarrow *cexpr* **where**

marg-dens2-cexpr $\Gamma vs x y e =$

(*cexpr-comp-aux* (*Suc* *x*) (*fst_c* (*CVar* 0))

(*cexpr-comp-aux* (*Suc* *y*) (*snd_c* (*CVar* 0))

(*map-vars* (*Suc* (*integrate-vars* Γ (*filter* ($\lambda z. z \neq x \wedge z \neq y$) *vs*) *e*))))

lemma *free-vars-marg-dens2-cexpr*:

assumes *cdens-ctxt-invar* *vs vs' Γ δ*

shows *free-vars* (*marg-dens2-cexpr* $\Gamma vs x y \delta$) \subseteq *shift-var-set* (*set* *vs'*)

proof –

have *free-vars* (*marg-dens2-cexpr* $\Gamma vs x y \delta$) \subseteq

shift-var-set (*free-vars* $\delta - \text{set } vs$)

unfolding *marg-dens2-cexpr-def* **using** *cdens-ctxt-invarD*[*OF* *assms*(1)]

apply (*intro order.trans*[*OF* *free-vars-cexpr-comp-aux*] *Un-least*)

apply (*subst Diff-subset-conv, intro order.trans*[*OF* *free-vars-cexpr-comp-aux*])

apply (*auto simp: shift-var-set-def*)

done

also from *cdens-ctxt-invarD*[*OF* *assms*(1)] **have** ... \subseteq *shift-var-set* (*set* *vs'*)

unfolding *shift-var-set-def* **by** *auto*

finally show *?thesis* .

qed

lemma *cexpr-typing-marg-dens2-cexpr*[*intro*]:

assumes $\Gamma \vdash_c \delta : \text{REAL}$

shows *case-nat* (*PRODUCT* (Γx) (Γy)) $\Gamma \vdash_c$ *marg-dens2-cexpr* $\Gamma vs x y \delta :$
REAL

proof –

have A : (case-nat (PRODUCT (Γ x) (Γ y)) Γ) (Suc x := Γ x , Suc y := Γ y) \circ
 Suc = Γ
by (intro ext) (auto split: nat.split)
show ?thesis **unfolding** marg-dens2-cexpr-def
apply (rule cexpr-typing-cexpr-comp-aux[of - - Γ x])
apply (rule cexpr-typing-cexpr-comp-aux[of - - Γ y])
apply (rule cexpr-typing-map-vars, subst A , rule cexpr-typing-integrate-vars[OF
 assms])
apply (rule cet-op, rule cet-var, simp, rule cet-op, rule cet-var, simp)
done
qed

lemma cexpr-sem-marg-dens2:

assumes cdens-ctxt-invar vs vs' Γ δ
assumes x : $x \in \text{set } vs$ **and** y : $y \in \text{set } vs$ **and** $x \neq y$
assumes ϱ : $\varrho \in \text{space } (\text{state-measure } (\text{set } vs') \Gamma)$
shows AE z in stock-measure (PRODUCT (Γ x) (Γ y)).
 $\text{ennreal } (\text{extract-real } (\text{cexpr-sem } (\text{case-nat } z \varrho) (\text{marg-dens2-cexpr } \Gamma \text{ } vs \text{ } x$
 $y \delta))) =$
 $\text{marg-dens2 } (\text{dens-ctxt-}\alpha (vs, vs', \Gamma, \delta)) \text{ } x \text{ } y \text{ } \varrho \text{ } z$

proof –

note invar = cdens-ctxt-invarD[OF assms(1)]
let ?f = λx . $\text{ennreal } (\text{extract-real } (\text{cexpr-sem } x \delta))$
let ?vs = filter (λz . $z \neq x \wedge z \neq y$) vs
interpret product-sigma-finite λx . stock-measure (Γ x)
unfolding product-sigma-finite-def **by** simp
interpret sf-PiM: sigma-finite-measure PiM (set ?vs) (λx . stock-measure (Γ x))
by (intro sigma-finite) simp

have meas: ($\lambda \sigma$. $\text{extract-real } (\text{cexpr-sem } (\text{merge } (\text{set } vs) (\text{set } vs') (\sigma, \varrho)) \delta)$
 $\in \text{borel-measurable } (\text{state-measure } (\text{set } vs) \Gamma)$) **using** assms invar
by (intro measurable-cexpr-sem') simp-all
from x y **have** insert-eq: insert x (insert y (set ?vs)) = set vs **by** auto
from x y **have** insert-eq': insert y (insert x (set ?vs)) = set vs **by** auto
have meas-upd1: ($\lambda (\sigma, v)$. $\sigma(y := v)$) \in
 $\text{measurable } (\text{PiM } (\text{insert } x (\text{set } vs)) (\lambda x$. stock-measure (Γ x)) \otimes_M stock-measure
(Γ y))
 $(\text{PiM } (\text{insert } y (\text{insert } x (\text{set } vs))) (\lambda x$. stock-measure (Γ x)))
using measurable-add-dim[of y insert x (set ?vs) λx . stock-measure (Γ x)]
by (simp only: insert-eq', simp)
hence meas-upd2: (λxa . (snd xa) ($x := \text{fst } (\text{fst } xa)$, $y := \text{snd } (\text{fst } xa)$))
 $\in \text{measurable } ((\text{stock-measure } (\Gamma$ x) \otimes_M stock-measure (Γ y)) \otimes_M
 $\text{PiM } (\text{set } ?vs) (\lambda y$. stock-measure (Γ y)))
 $(\text{PiM } (\text{set } vs) (\lambda y$. stock-measure (Γ y)))
by (subst insert-eq'[symmetric], intro measurable-Pair-compose-split[OF mea-
surable-add-dim])
(simp-all del: fun-upd-apply)

from x y **have** A : set $vs = \{x, y\} \cup \text{set } ?vs$ **by** auto

have $(\int^{+\sigma}. ?f \text{ (merge (set vs) (set vs') (\sigma, \varrho)) } \partial\text{state-measure (set vs) } \Gamma) =$
 $(\int^{+\sigma'}. \int^{+\sigma}. ?f \text{ (merge (set vs) (set vs') (merge \{x, y\} (set ?vs) (\sigma', \sigma), \varrho))$
 $(\sigma), \varrho))$
 $\partial\text{state-measure (set ?vs) } \Gamma \partial\text{state-measure } \{x, y\} \Gamma$ (**is** $- = ?I$)
using *meas insert-eq unfolding state-measure-def*
by (*subst A, subst product-nn-integral-fold*) (*simp-all add: measurable-compose[OF*
 $- \text{measurable-ennreal}]$)
also have $\bigwedge \sigma' \sigma. \text{merge (set vs) (set vs') (merge \{x, y\} (set ?vs) (\sigma', \sigma), \varrho) =}$
 $\text{merge (set vs) (set vs') (\sigma(x := \sigma' x, y := \sigma' y), \varrho)}$
by (*intro ext*) (*auto simp: merge-def*)
hence $?I = (\int^{+\sigma'}. \int^{+\sigma}. ?f \text{ (merge (set vs) (set vs') (\sigma(x := \sigma' x, y := \sigma' y),$
 $\varrho))$
 $\partial\text{state-measure (set ?vs) } \Gamma \partial\text{state-measure } \{x, y\} \Gamma)$ **by** *simp*
also have $\dots = \int^{+z}. \int^{+\sigma}. ?f \text{ (merge (set vs) (set vs') (\sigma(x := fst z, y := snd$
 $z), \varrho))$
 $\partial\text{state-measure (set ?vs) } \Gamma \partial(\text{stock-measure } (\Gamma x) \otimes_M \text{stock-measure}$
 $(\Gamma y))$
(is $- = ?I)$ **using** $\langle x \neq y \rangle$ *meas-upd2 \varrho invar unfolding state-measure-def*
by (*subst product-nn-integral-pair, subst measurable-split-conv,*
intro sf-PiM.borel-measurable-nn-integral)
(auto simp: measurable-split-conv state-measure-def intro!: measurable-compose[OF
 $- \text{measurable-ennreal}]$
 $\text{measurable-compose[OF - measurable-cexpr-sem]} \text{measurable-Pair })$
finally have $(\int^{+\sigma}. ?f \text{ (merge (set vs) (set vs') (\sigma, \varrho)) } \partial\text{state-measure (set vs)$
 $\Gamma) = ?I .$
moreover have $(\int^{+\sigma}. ?f \text{ (merge (set vs) (set vs') (\sigma, \varrho)) } \partial\text{state-measure (set$
 $vs) \Gamma) \neq \infty$
using *cdens-ctxt-invar-imp-integrable[OF assms(1) \varrho] unfolding real-integrable-def*
by *simp*
ultimately have $?I \neq \infty$ **by** *simp*
hence $AE z \text{ in stock-measure } (\Gamma x) \otimes_M \text{stock-measure } (\Gamma y).$
 $(\int^{+\sigma}. ?f \text{ (merge (set vs) (set vs') (\sigma(x := fst z, y := snd z), \varrho))$
 $\partial\text{state-measure (set ?vs) } \Gamma) \neq \infty$ (**is** $AE z \text{ in } -. ?P z$)
using *meas-upd2 \varrho invar unfolding state-measure-def*
by (*intro nn-integral-PInf-AE sf-PiM.borel-measurable-nn-integral*)
(auto intro!: measurable-compose[OF - measurable-ennreal] measurable-compose[OF
 $- \text{measurable-cexpr-sem}]$
 $\text{measurable-Pair simp: measurable-split-conv state-measure-def})$
hence $AE z \text{ in stock-measure } (\Gamma x) \otimes_M \text{stock-measure } (\Gamma y).$
 $\text{ennreal (extract-real (cexpr-sem (case-nat (case-prod PairVal z) \varrho)$
 $(\text{marg-dens2-cexpr } \Gamma \text{ vs } x \ y \ \delta))) =}$
 $\text{marg-dens2 (dens-ctxt-}\alpha \text{ (vs, vs', } \Gamma, \delta)) \ x \ y \ \varrho \text{ (case-prod PairVal z)}$
proof (*rule AE-mp[OF - AE-I2[OF impI]]*)
fix z **assume** $z \in \text{space (stock-measure } (\Gamma x) \otimes_M \text{stock-measure } (\Gamma y))$
assume *fin: ?P z*
have $\bigwedge \sigma. \text{merge (set vs) (set vs') (\sigma(x := fst z, y := snd z), \varrho) =}$
 $\text{merge (set ?vs) (\{x, y\} \cup \text{set vs') (\sigma, \varrho(x := fst z, y := snd z))}$ **using**
 $x \ y$
by (*intro ext*) (*simp add: merge-def*)

hence A: $(\int^+ \sigma. ?f (\text{merge } (\text{set } vs) (\text{set } vs')) (\sigma(x := \text{fst } z, y := \text{snd } z), \varrho)$
 $\partial \text{state-measure } (\text{set } ?vs) \Gamma =$
 $(\int^+ \sigma. ?f (\text{merge } (\text{set } ?vs) (\{x,y\} \cup \text{set } vs')) (\sigma, \varrho(x := \text{fst } z, y := \text{snd } z)))$
 $\partial \text{state-measure } (\text{set } ?vs) \Gamma$ (**is** $- = \int^+ \sigma. \text{ennreal } (?g \sigma) \partial ?M$)
by (*intro nn-integral-cong*) *simp*
have ϱ' : $\varrho(x := \text{fst } z, y := \text{snd } z) \in \text{space } (\text{state-measure } (\{x, y\} \cup \text{set } vs')) \Gamma$
using $z \varrho$ **unfolding** *state-measure-def*
by (*auto simp: space-PiM space-pair-measure split: if-split-asm*)
have *integrable: integrable ?M ?g*
proof (*intro integrableI-nonneg[OF - AE-I2]*)
show $?g \in \text{borel-measurable } ?M$ **using** *invar* ϱ' **by** (*intro measurable-cexpr-sem'*)
auto
show $(\int^+ \sigma. \text{ennreal } (?g \sigma) \partial ?M) < \infty$ **using** *fin A* **by** (*simp add: top-unique less-top*)
fix σ **assume** $\sigma: \sigma \in \text{space } ?M$
from $x y$ **have** $\text{set } ?vs \cup (\{x,y\} \cup \text{set } vs') = \text{set } vs \cup \text{set } vs'$ **by** *auto*
thus $?g \sigma \geq 0$ **using** *merge-in-state-measure[OF \sigma \varrho]*
by (*intro nonneg-cexprD[OF invar(4)] simp-all*)
qed
from $x y$ **have** $B: (\text{set } ?vs \cup (\{x, y\} \cup \text{set } vs')) = \text{set } vs \cup \text{set } vs'$ **by** *auto*
have *nonneg: nonneg-cexpr* $(\text{set } [z \leftarrow vs . z \neq x \wedge z \neq y] \cup (\{x, y\} \cup \text{set } vs'))$
 $\Gamma \delta$
using *invar* **by** (*subst B*) *simp*

have *ennreal* $(\text{extract-real } (\text{cexpr-sem } (\text{case-nat } (\text{case-prod } \text{PairVal } z) \varrho) (\text{marg-dens2-cexpr } \Gamma \text{ vs } x \text{ y } \delta))) =$
 $\text{extract-real } (\text{cexpr-sem } ((\text{case-nat } <|\text{fst } z, \text{snd } z|> \varrho) (\text{Suc } x := \text{fst } z, \text{Suc } y := \text{snd } z) \circ \text{Suc}))$
 $(\text{integrate-vars } \Gamma \text{ ?vs } \delta)$
unfolding *marg-dens2-cexpr-def*
by (*simp add: cexpr-sem-cexpr-comp-aux cexpr-sem-map-vars split: prod.split*)
also have $((\text{case-nat } <|\text{fst } z, \text{snd } z|> \varrho) (\text{Suc } x := \text{fst } z, \text{Suc } y := \text{snd } z)) \circ \text{Suc} =$
 $\varrho(x := \text{fst } z, y := \text{snd } z)$ (**is** $?q1 = ?q2$) **by** (*intro ext*) (*simp split: nat.split*)
also have *ennreal* $(\text{extract-real } (\text{cexpr-sem } (\varrho(x := \text{fst } z, y := \text{snd } z))$
 $(\text{integrate-vars } \Gamma [z \leftarrow vs . z \neq x \wedge z \neq y] \delta))) =$
 $\int^+ xa. ?f (\text{merge } (\text{set } ?vs) (\{x, y\} \cup \text{set } vs')) (xa, \varrho(x := \text{fst } z, y := \text{snd } z))) \partial ?M$
using *invar* *assms* **by** (*intro cexpr-sem-integrate-vars'[OF \varrho' - - nonneg integrable]*) *auto*
also have $C: \text{set } ?vs = \text{set } vs - \{x, y\}$ **by** *auto*
have $(\int^+ xa. ?f (\text{merge } (\text{set } ?vs) (\{x, y\} \cup \text{set } vs')) (xa, \varrho(x := \text{fst } z, y := \text{snd } z))) \partial ?M =$
 $\text{marg-dens2 } (\text{dens-ctxt-}\alpha (vs, vs', \Gamma, \delta)) \text{ } x \text{ y } \varrho (\text{case-prod } \text{PairVal } z)$
unfolding *marg-dens2-def*
by (*subst A[symmetric], subst C, simp only: dens-ctxt-}\alpha-def prod.case*)
(auto intro!: nn-integral-cong split: prod.split)

finally show $\text{ennreal } (\text{extract-real } (\text{cexpr-sem } (\text{case-nat } (\text{case-prod PairVal } z)$
 $\varrho)$
 $(\text{marg-dens2-cexpr } \Gamma \text{ vs } x \text{ y } \delta))) =$
 $\text{marg-dens2 } (\text{dens-ctxt-}\alpha \text{ (vs, vs', } \Gamma, \delta)) \text{ x y } \varrho \text{ (case-prod PairVal}$
 $z)$.
qed
thus *?thesis* **by** (*subst stock-measure.simps, subst AE-embed-measure[OF inj-PairVal]*)
simp
qed

lemma *nonneg-cexpr-sem-marg-dens2*:
assumes *cdens-ctxt-invar vs vs' Γ δ*
assumes *x: $x \in \text{set vs}$ and y: $y \in \text{set vs}$ and $\varrho: \varrho \in \text{space } (\text{state-measure } (\text{set vs}^\wedge) \Gamma)$*
assumes *v: $v \in \text{type-universe } (\text{PRODUCT } (\Gamma \text{ x}) (\Gamma \text{ y}))$*
shows $\text{extract-real } (\text{cexpr-sem } (\text{case-nat } v \varrho) (\text{marg-dens2-cexpr } \Gamma \text{ vs } x \text{ y } \delta)) \geq 0$
proof –
from *v* **obtain** *a b* **where** *v'*: $v = \langle |a, b| \rangle$ $a \in \text{type-universe } (\Gamma \text{ x})$ $b \in \text{type-universe } (\Gamma \text{ y})$
by (*auto simp: val-type-eq-PRODUCT*)
let *?vs = filter* ($\lambda z. z \neq x \wedge z \neq y$) *vs*
note *invar = cdens-ctxt-invarD[OF assms(1)]*
have *A: ((case-nat v ϱ) (Suc x := fst (extract-pair v), Suc y := snd (extract-pair v))) \circ Suc =*
 $\varrho(x := \text{fst } (\text{extract-pair } v), y := \text{snd } (\text{extract-pair } v))$ **by** (*intro ext*)
auto
have *B: $\varrho(x := \text{fst } (\text{extract-pair } v), y := \text{snd } (\text{extract-pair } v))$*
 $\in \text{space } (\text{state-measure } (\text{set vs}' \cup \{x, y\}) \Gamma)$ **using** *x y v' ϱ*
by (*auto simp: space-state-measure split: if-split-asm*)
from *x y* **have** $\text{set } ?vs \cup (\text{set vs}' \cup \{x, y\}) = \text{set vs} \cup \text{set vs}'$ **by** *auto*
with *invar* **have** *nonneg-cexpr* ($\text{set } ?vs \cup (\text{set vs}' \cup \{x, y\})$) $\Gamma \delta$ **by** *simp*
thus *?thesis* **using** *assms invar(1–3) A unfolding marg-dens2-cexpr-def*
by (*auto simp: cexpr-sem-cexpr-comp-aux cexpr-sem-map-vars intro!: non-neg-cexpr-sem-integrate-vars[OF B]*)
qed

definition *branch-prob-cexpr* :: $\text{cdens-ctxt} \Rightarrow \text{cexpr}$ **where**
 $\text{branch-prob-cexpr} \equiv \lambda(\text{vs, vs}', \Gamma, \delta). \text{integrate-vars } \Gamma \text{ vs } \delta$

lemma *free-vars-branch-prob-cexpr[simp]*:
 $\text{free-vars } (\text{branch-prob-cexpr } (\text{vs, vs}', \Gamma, \delta)) = \text{free-vars } \delta - \text{set vs}$
unfolding *branch-prob-cexpr-def* **by** *simp*

lemma *cexpr-typing-branch-prob-cexpr[intro]*:
 $\Gamma \vdash_c \delta : \text{REAL} \Longrightarrow \Gamma \vdash_c \text{branch-prob-cexpr } (\text{vs, vs}', \Gamma, \delta) : \text{REAL}$
unfolding *branch-prob-cexpr-def*
by (*simp only: prod.case, rule cexpr-typing-integrate-vars*)

lemma *cepr-sem-branch-prob*:
assumes *cdens-ctxt-invar vs vs' Γ δ*
assumes ϱ : $\varrho \in \text{space } (\text{state-measure } (\text{set } vs') \Gamma)$
shows $\text{ennreal } (\text{extract-real } (\text{cepr-sem } \varrho (\text{branch-prob-cepr } (vs, vs', \Gamma, \delta)))) =$
 $\text{branch-prob } (\text{dens-ctxt-}\alpha (vs, vs', \Gamma, \delta)) \varrho$
proof –
note *invar = cdens-ctxt-invarD[OF assms(1)]*
interpret *density-context set vs set vs' Γ λσ. extract-real (cepr-sem σ δ)*
by (*rule density-context-α*) *fact*
have $\text{ennreal } (\text{extract-real } (\text{cepr-sem } \varrho (\text{branch-prob-cepr } (vs, vs', \Gamma, \delta)))) =$
 $\int^+ \sigma. \text{extract-real } (\text{cepr-sem } (\text{merge } (\text{set } vs) (\text{set } vs') (\sigma, \varrho)) \delta)$
 $\partial \text{state-measure } (\text{set } vs) \Gamma (\text{is } - = ?I)$
using *assms(2) invar unfolding branch-prob-cepr-def*
by (*simp only: prod.case, subst cepr-sem-integrate-vars'*)
(*auto intro!: cdens-ctxt-invar-imp-integrable assms*)
also have $\dots = \text{branch-prob } (\text{dens-ctxt-}\alpha (vs, vs', \Gamma, \delta)) \varrho$
using ϱ **unfolding** *dens-ctxt-α-def* **by** (*simp only: prod.case branch-prob-altdef[of*
 $\varrho]$)
finally show *?thesis .*
qed

lemma *subprob-imp-subprob-cepr*:
assumes *density-context V V' Γ (λσ. extract-real (cepr-sem σ δ))*
shows *subprob-cepr V V' Γ δ*
proof (*intro subprob-ceprI*)
interpret *density-context V V' Γ λσ. extract-real (cepr-sem σ δ)* **by** *fact*
fix ϱ **assume** ϱ : $\varrho \in \text{space } (\text{state-measure } V' \Gamma)$
let $?M = \text{dens-ctxt-measure } (V, V', \Gamma, \lambda\sigma. \text{extract-real } (\text{cepr-sem } \sigma \delta)) \varrho$
from ϱ **have** $(\int^+ x. \text{ennreal } (\text{extract-real } (\text{cepr-sem } (\text{merge } V V' (x, \varrho)) \delta)))$
 $\partial \text{state-measure } V \Gamma =$
 $\text{branch-prob } (V, V', \Gamma, \lambda\sigma. \text{extract-real } (\text{cepr-sem } \sigma \delta)) \varrho (\text{is } ?I$
 $= -)$
by (*subst branch-prob-altdef[symmetric] simp-all*)
also have $\dots = \text{emeasure } ?M (\text{space } ?M)$ **unfolding** *branch-prob-def* **by** *simp*
also have $\dots \leq 1$ **by** (*rule subprob-space.emeasure-space-le-1*) (*simp add: sub-*
prob-space-dens ρ)
finally show $?I \leq 1$.
qed

end

10 Concrete PDF Compiler

theory *PDF-Compiler*
imports *PDF-Compiler-Pred PDF-Target-Density-Contexts*
begin

inductive *expr-has-density-cepr* :: *cdens-ctxt* \Rightarrow *expr* \Rightarrow *cepr* \Rightarrow *bool*

$(\langle 1/\vdash_c / (- \Rightarrow / -) \rangle [50,0,50] 50)$ **where**

edc-val: $\text{countable-type } (val\text{-type } v) \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c \text{Val } v \Rightarrow$
 $\text{map-vars } Suc \text{ (branch-prob-cexpr } (vs, vs', \Gamma, \delta)) *_c \langle CVar 0 =_c$
 $CVal v \rangle_c$

| *edc-var*: $x \in \text{set } vs \Longrightarrow (vs, vs', \Gamma, \delta) \vdash_c \text{Var } x \Rightarrow \text{marg-dens-cexpr } \Gamma \text{ vs } x \delta$
| *edc-pair*: $x \in \text{set } vs \Longrightarrow y \in \text{set } vs \Longrightarrow x \neq y \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c \langle \text{Var } x, \text{Var } y \rangle \Rightarrow \text{marg-dens2-cexpr } \Gamma \text{ vs } x \text{ } y \delta$

| *edc-fail*: $(vs, vs', \Gamma, \delta) \vdash_c \text{Fail } t \Rightarrow CReal 0$
| *edc-let*: $([], vs @ vs', \Gamma, CReal 1) \vdash_c e \Rightarrow f \Longrightarrow$
 $(\text{shift-vars } vs, \text{map } Suc \text{ } vs', \text{the } (expr\text{-type } \Gamma \text{ } e) \cdot \Gamma,$
 $\text{map-vars } Suc \delta *_c f) \vdash_c e' \Rightarrow g \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c LET \text{ } e \text{ } IN \text{ } e' \Rightarrow \text{map-vars } (\lambda x. x - 1) g$

| *edc-rand*: $(vs, vs', \Gamma, \delta) \vdash_c e \Rightarrow f \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c \text{Random } dst \text{ } e \Rightarrow$
 $\int_c \text{map-vars } (\text{case-nat } 0 \text{ } (\lambda x. x + 2)) f *_c$
 $\text{dist-dens-cexpr } dst \text{ } (CVar 0) \text{ } (CVar 1) \text{ } \partial \text{dist-param-type } dst$

| *edc-rand-det*: $\text{randomfree } e \Longrightarrow \text{free-vars } e \subseteq \text{set } vs' \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c \text{Random } dst \text{ } e \Rightarrow$
 $\text{map-vars } Suc \text{ (branch-prob-cexpr } (vs, vs', \Gamma, \delta)) *_c$
 $\text{dist-dens-cexpr } dst \text{ (map-vars } Suc \text{ (expr-rf-to-cexpr } e)) \text{ } (CVar 0)$

| *edc-if-det*: $\text{randomfree } b \Longrightarrow$
 $(vs, vs', \Gamma, \delta *_c \langle \text{expr-rf-to-cexpr } b \rangle_c) \vdash_c e1 \Rightarrow f1 \Longrightarrow$
 $(vs, vs', \Gamma, \delta *_c \langle \neg_c \text{expr-rf-to-cexpr } b \rangle_c) \vdash_c e2 \Rightarrow f2 \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c IF \text{ } b \text{ } THEN \text{ } e1 \text{ } ELSE \text{ } e2 \Rightarrow f1 +_c f2$

| *edc-if*: $([], vs @ vs', \Gamma, CReal 1) \vdash_c b \Rightarrow f \Longrightarrow$
 $(vs, vs', \Gamma, \delta *_c \text{cexpr-subst-val } f \text{ } TRUE) \vdash_c e1 \Rightarrow g1 \Longrightarrow$
 $(vs, vs', \Gamma, \delta *_c \text{cexpr-subst-val } f \text{ } FALSE) \vdash_c e2 \Rightarrow g2 \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c IF \text{ } b \text{ } THEN \text{ } e1 \text{ } ELSE \text{ } e2 \Rightarrow g1 +_c g2$

| *edc-op-discr*: $(vs, vs', \Gamma, \delta) \vdash_c e \Rightarrow f \Longrightarrow \Gamma \vdash e : t \Longrightarrow$
 $\text{op-type } oper \text{ } t = \text{Some } t' \Longrightarrow \text{countable-type } t' \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c oper \text{ } \$\$ \text{ } e \Rightarrow$
 $\int_c \langle (oper \text{ } \$\$ \text{ } (CVar 0)) =_c CVar 1 \rangle_c *_c \text{map-vars } (\text{case-nat}$
 $0 \text{ } (\lambda x. x + 2)) f \text{ } \partial t$

| *edc-fst*: $(vs, vs', \Gamma, \delta) \vdash_c e \Rightarrow f \Longrightarrow \Gamma \vdash e : PRODUCT \text{ } t \text{ } t' \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c Fst \text{ } \$\$ \text{ } e \Rightarrow$
 $\int_c (\text{map-vars } (\text{case-nat } 0 \text{ } (\lambda x. x + 2))) f \text{ } \circ_c \langle CVar 1, CVar$
 $0 \rangle_c \text{ } \partial t'$

| *edc-snd*: $(vs, vs', \Gamma, \delta) \vdash_c e \Rightarrow f \Longrightarrow \Gamma \vdash e : PRODUCT \text{ } t \text{ } t' \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c Snd \text{ } \$\$ \text{ } e \Rightarrow$
 $\int_c (\text{map-vars } (\text{case-nat } 0 \text{ } (\lambda x. x + 2))) f \text{ } \circ_c \langle CVar 0, CVar$
 $1 \rangle_c \text{ } \partial t$

| *edc-neg*: $(vs, vs', \Gamma, \delta) \vdash_c e \Rightarrow f \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c \text{Minus } \$\$ \text{ } e \Rightarrow f \text{ } \circ_c (\lambda x. -_c x)$

| *edc-addc*: $(vs, vs', \Gamma, \delta) \vdash_c e \Rightarrow f \Longrightarrow \text{randomfree } e' \Longrightarrow \text{free-vars } e' \subseteq \text{set } vs'$
 \Longrightarrow
 $(vs, vs', \Gamma, \delta) \vdash_c \text{Add } \$\$ \text{ } \langle e, e' \rangle \Rightarrow$
 $f \text{ } \circ_c (\lambda x. x -_c \text{map-vars } Suc \text{ (expr-rf-to-cexpr } e'))$

| *edc-multc*: $(vs, vs', \Gamma, \delta) \vdash_c e \Rightarrow f \Longrightarrow c \neq 0 \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c \text{Mult } \text{\$}\$ \langle e, \text{Val } (\text{RealVal } c) \rangle \Rightarrow$
 $(f \circ_c (\lambda_c x. x *_c \text{CReal } (\text{inverse } c))) *_c \text{CReal } (\text{inverse } (\text{abs } c))$

| *edc-add*: $(vs, vs', \Gamma, \delta) \vdash_c e \Rightarrow f \Longrightarrow \Gamma \vdash e : \text{PRODUCT } t \ t \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c \text{Add } \text{\$}\$ e \Rightarrow$
 $\int_c (\text{map-vars } (\text{case-nat } 0 \ (\lambda x. x+2)) f \circ_c (\lambda_c x. \langle x, \text{CVar } 1 -_c$
 $x \rangle_c)) \ \partial t$

| *edc-inv*: $(vs, vs', \Gamma, \delta) \vdash_c e \Rightarrow f \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c \text{Inverse } \text{\$}\$ e \Rightarrow$
 $(f \circ_c (\lambda_c x. \text{inverse}_c x)) *_c (\lambda_c x. (\text{inverse}_c x) \hat{\ }_c \text{CInt } 2)$

| *edc-exp*: $(vs, vs', \Gamma, \delta) \vdash_c e \Rightarrow f \Longrightarrow$
 $(vs, vs', \Gamma, \delta) \vdash_c \text{Exp } \text{\$}\$ e \Rightarrow$
 $(\lambda_c x. \text{IF}_c \text{CReal } 0 <_c x \ \text{THEN } (f \circ_c \ln_c x) *_c \text{inverse}_c x \ \text{ELSE}$
 $\text{CReal } 0)$

code-pred *expr-has-density-cexpr* .

Auxiliary lemmas

lemma *cdens-ctxt-invar-insert*:

assumes *inv*: *cdens-ctxt-invar* *vs vs' Γ δ*

assumes *t* : $\Gamma \vdash e : t'$

assumes *free-vars*: *free-vars* *e* \subseteq *set vs* \cup *set vs'*

assumes *hd*: *dens-ctxt-α* ($\llbracket \cdot \rrbracket$, *vs @ vs', Γ, CReal 1*) $\vdash_d e \Rightarrow (\lambda x \ xa. \text{ennreal } (\text{eval-cexpr } f \ x \ xa))$

notes *invar* = *cdens-ctxt-invarD*[*OF inv*]

assumes *wf1*: *is-density-cexpr* ($\llbracket \cdot \rrbracket$, *vs @ vs', Γ, CReal 1*) *t' f*

shows *cdens-ctxt-invar* (*shift-vars vs*) (*map Suc vs'*) (*t' · Γ*) (*map-vars Suc δ *_c f*)

proof (*intro cdens-ctxt-invarI*)

show *t'*: *case-nat t' Γ* $\vdash_c \text{map-vars } \text{Suc } \delta *_c f : \text{REAL using } \text{invar } \text{wf1}$

by (*intro cet-op*[**where** *t = PRODUCT REAL REAL*])

(*auto intro!*: *cexpr-typing.intros cexpr-typing-map-vars simp: o-def dest: is-density-cexprD*)

let *?vs* = *shift-var-set* (*set vs*) **and** *?vs'* = *Suc ' set vs'* **and** *?Γ* = *case-nat t' Γ*
and

?δ = *insert-dens* (*set vs*) (*set vs'*) ($\lambda \sigma \ x. \text{ennreal } (\text{eval-cexpr } f \ \sigma \ x)$)

($\lambda x. \text{ennreal } (\text{extract-real } (\text{cexpr-sem } x \ \delta))$)

interpret *density-context* *set vs set vs' Γ λσ. extract-real (cexpr-sem σ δ)*

by (*rule density-context-α*[*OF inv*])

have *dc*: *density-context* $\{\}$ (*set vs* \cup *set vs'*) $\Gamma (\lambda-. 1)$

by (*rule density-context-empty*)

hence *dens*: *has-parametrized-subprob-density* (*state-measure* (*set vs* \cup *set vs'*) Γ)

($\lambda \rho. \text{dens-ctxt-measure } (\{\}, \text{set } vs \cup \text{set } vs', \Gamma, \lambda-. 1) \ \rho \ggg (\lambda \sigma. \text{expr-sem } \sigma \ e)$)

(*stock-measure t'*) ($\lambda \sigma \ x. \text{ennreal } (\text{eval-cexpr } f \ \sigma \ x)$)

using *hd free-vars* **by** (*intro expr-has-density-sound-aux*[*OF - t dc*])

```

    (auto simp: shift-var-set-def dens-ctxt- $\alpha$ -def simp: extract-real-def one-ennreal-def)
  from density-context.density-context-insert[OF density-context- $\alpha$ [OF inv] this]
  have density-context ?vs ?vs' ? $\Gamma$  ? $\delta$  .
  have dc: density-context (shift-var-set (set vs)) (Suc ' set vs') (case-nat t'  $\Gamma$ )
    ( $\lambda\sigma$ . extract-real (cexpr-sem  $\sigma$  (map-vars Suc  $\delta$  *c f)))
  proof (rule density-context-equiv)
    show density-context (shift-var-set (set vs)) (Suc ' set vs') (case-nat t'  $\Gamma$ ) ? $\delta$ 
  by fact
    show ( $\lambda x$ . ennreal (extract-real (cexpr-sem x (map-vars Suc  $\delta$  *c f))))
       $\in$  borel-measurable (state-measure (?vs  $\cup$  ?vs') ? $\Gamma$ )
    apply (rule measurable-compose[OF - measurable-ennreal], rule measurable-
  able-compose[OF - measurable-extract-real])
    apply (rule measurable-cexpr-sem[OF t'])
    apply (insert invar is-density-exprD[OF wf1], auto simp: shift-var-set-def)
    done
  next
  fix  $\sigma$  assume  $\sigma$ :  $\sigma \in$  space (state-measure (?vs  $\cup$  ?vs') ? $\Gamma$ )
  have [simp]: case-nat ( $\sigma$  0) ( $\lambda x$ .  $\sigma$  (Suc x)) =  $\sigma$  by (intro ext) (simp split:
  nat.split)
  from  $\sigma$  show insert-dens (set vs) (set vs') ( $\lambda\sigma$  x. ennreal (eval-cexpr f  $\sigma$  x))
    ( $\lambda x$ . ennreal (extract-real (cexpr-sem x  $\delta$ )))  $\sigma$  =
    ennreal (extract-real (cexpr-sem  $\sigma$  (map-vars Suc  $\delta$  *c f)))
  unfolding insert-dens-def using invar is-density-exprD[OF wf1]
  apply (subst ennreal-mult'[symmetric])
  apply (erule nonneg-cexprD)
  apply (rule measurable-space[OF measurable-remove-var[where t=t']])
  apply simp
  apply (subst cexpr-sem-Mult[of ? $\Gamma$  - - - ?vs  $\cup$  ?vs'])
  apply (auto intro!: cexpr-typing-map-vars ennreal-mult'[symmetric]
    simp: o-def shift-var-set-def eval-cexpr-def
    cexpr-sem-map-vars remove-var-def)
  done
qed

from subprob-imp-subprob-cexpr[OF this]
  show subprob-cexpr (set (shift-vars vs)) (set (map Suc vs')) (case-nat t'  $\Gamma$ )
    (map-vars Suc  $\delta$  *c f) by simp

have Suc - ' shift-var-set (set vs  $\cup$  set vs') = set vs  $\cup$  set vs'
  by (auto simp: shift-var-set-def)
moreover have nonneg-cexpr (shift-var-set (set vs  $\cup$  set vs')) (case-nat t'  $\Gamma$ ) f
  using wf1[THEN is-density-exprD-nonneg] by simp
ultimately show nonneg-cexpr (set (shift-vars vs)  $\cup$  set (map Suc vs')) (case-nat
  t'  $\Gamma$ ) (map-vars Suc  $\delta$  *c f)
  using invar is-density-exprD[OF wf1]
  by (intro nonneg-cexpr-Mult)
    (auto intro!: cexpr-typing-map-vars nonneg-cexpr-map-vars
    simp: o-def shift-var-set-def image-Un)
qed (insert invar is-density-exprD[OF wf1],

```

auto simp: shift-vars-def shift-var-set-def distinct-map intro!: cexpr-typing-map-vars)

lemma *cdens-ctxt-invar-insert-bool:*

assumes *dens: dens-ctxt- α* (\square , *vs* @ *vs'*, Γ , *CReal* 1) \vdash_d *b* \Rightarrow ($\lambda \rho$ *x*. *ennreal* (*eval-cexpr* *f* ρ *x*))

assumes *wf: is-density-cexpr* (\square , *vs* @ *vs'*, Γ , *CReal* 1) *BOOL* *f*

assumes *t: $\Gamma \vdash b : \text{BOOL}$* **and** *vars: free-vars b \subseteq set vs \cup set vs'*

assumes *invar: cdens-ctxt-invar vs vs' Γ δ*

shows *cdens-ctxt-invar vs vs' Γ ($\delta *_c$ cexpr-subst-val *f* (BoolVal *v*))*

proof (*intro cdens-ctxt-invarI nonneg-cexpr-Mult nonneg-cexpr-subst-val*)

note *invar' = cdens-ctxt-invarD[OF invar]* **and** *wf' = is-density-cexprD[OF wf]*

show $\Gamma \vdash_c \delta *_c$ *cexpr-subst-val f* (BoolVal *v*) : *REAL* **using** *invar' wf'*

by (*intro cet-op[where t = PRODUCT REAL REAL] cet-pair cexpr-typing-subst-val*)

simp-all

let *?M = $\lambda \rho$. dens-ctxt-measure* ($\{\}$, *set vs* \cup *set vs'*, Γ , $\lambda \cdot$. 1) $\rho \ggg$ ($\lambda \sigma$. *expr-sem* σ *b*)

have *dens': has-parametrized-subprob-density* (*state-measure* (*set vs* \cup *set vs'*) Γ) *?M*

(stock-measure BOOL) ($\lambda \sigma$ *v*. *ennreal* (*eval-cexpr* *f* σ *v*))

using *density-context- α [OF invar]* *t vars dens* **unfolding** *dens-ctxt- α -def*

by (*intro expr-has-density-sound-aux density-context.density-context-empty*)

(auto simp: extract-real-def one-ennreal-def)

thus *nonneg: nonneg-cexpr* (*shift-var-set* (*set vs* \cup *set vs'*)) (*case-nat* *BOOL* Γ) *f*

using *wf[THEN is-density-cexprD-nonneg]* **by** *simp*

show *subprob-cexpr* (*set vs*) (*set vs'*) Γ ($\delta *_c$ *cexpr-subst-val f* (BoolVal *v*))

proof (*intro subprob-cexprI*)

fix ρ **assume** ρ : $\rho \in$ *space* (*state-measure* (*set vs'*) Γ)

let *?eval = λe σ . extract-real* (*cexpr-sem* (*merge* (*set vs*) (*set vs'*) (σ , ρ)) *e*)

$\{$

fix σ **assume** σ : $\sigma \in$ *space* (*state-measure* (*set vs*) Γ)

have *A: ?eval* ($\delta *_c$ *cexpr-subst-val f* (BoolVal *v*)) $\sigma =$

*?eval δ σ * ?eval* (*cexpr-subst-val f* (BoolVal *v*)) σ **using** *wf' invar'*

σ ρ

by (*subst cexpr-sem-Mult[where $\Gamma = \Gamma$ and $V = \text{set vs} \cup \text{set vs}'$]*)

(auto intro: merge-in-state-measure simp: shift-var-set-def)

have *?eval δ $\sigma \geq 0$* **using** σ ρ *invar'*

by (*blast dest: nonneg-cexprD intro: merge-in-state-measure*)

moreover **have** *?eval* (*cexpr-subst-val f* (BoolVal *v*)) $\sigma \geq 0$ **using** σ ρ *nonneg*

by (*intro nonneg-cexprD nonneg-cexpr-subst-val*) (*auto intro: merge-in-state-measure*)

moreover **have** *B: ennreal* (*?eval* (*cexpr-subst-val f* (BoolVal *v*)) σ) =

ennreal (*eval-cexpr f* (*merge* (*set vs*) (*set vs'*) (σ , ρ)) (BoolVal *v*))

σ ρ)

(is - = ?f (BoolVal *v*)) **by** (*simp add: eval-cexpr-def*)

hence *ennreal* (*?eval* (*cexpr-subst-val f* (BoolVal *v*)) σ) ≤ 1

using σ ρ *dens' unfolding has-parametrized-subprob-density-def*

by (*subst B, intro subprob-count-space-density-le-1[of - - ?f]*)

(auto intro: merge-in-state-measure simp: stock-measure.simps)

ultimately **have** *?eval* ($\delta *_c$ *cexpr-subst-val f* (BoolVal *v*)) $\sigma \leq$ *?eval δ σ*

by (*subst A, intro mult-right-le-one-le*) *simp-all*
}
hence $(\int^+ \sigma. ?eval (\delta *_c cexpr\text{-subst}\text{-val } f (BoolVal\ v)) \sigma \partial state\text{-measure } (set\ vs) \Gamma) \leq$
 $(\int^+ \sigma. ?eval \delta \sigma \partial state\text{-measure } (set\ vs) \Gamma)$ **by** (*intro nn-integral-mono*)
(simp add: ennreal-leI)
also have $\dots \leq 1$ **using** *invar' ρ* **by** (*intro subprob-cexprD*)
finally show $(\int^+ \sigma. ?eval (\delta *_c cexpr\text{-subst}\text{-val } f (BoolVal\ v)) \sigma \partial state\text{-measure } (set\ vs) \Gamma) \leq 1$.
qed
qed (*insert cdens-ctxt-invarD[OF invar] is-density-exprD[OF wf],*
auto simp: shift-var-set-def)

lemma *space-state-measureD-shift*:
 $\sigma \in space (state\text{-measure } (shift\text{-var}\text{-set } V) (case\text{-nat } t \Gamma)) \implies$
 $\exists x \sigma'. x \in type\text{-universe } t \wedge \sigma' \in space (state\text{-measure } V \Gamma) \wedge \sigma = case\text{-nat } x \sigma'$
by (*intro exI[of - σ 0] exI[of - σ ∘ Suc]*)
(auto simp: fun-eq-iff PiE-iff space-state-measure extensional-def split: nat.split)

lemma *space-state-measure-shift-iff*:
 $\sigma \in space (state\text{-measure } (shift\text{-var}\text{-set } V) (case\text{-nat } t \Gamma)) \iff$
 $(\exists x \sigma'. x \in type\text{-universe } t \wedge \sigma' \in space (state\text{-measure } V \Gamma) \wedge \sigma = case\text{-nat } x \sigma')$
by (*auto dest!: space-state-measureD-shift*)

lemma *nonneg-cexprI-shift*:
assumes $\bigwedge x \sigma. x \in type\text{-universe } t \implies \sigma \in space (state\text{-measure } V \Gamma) \implies$
 $0 \leq extract\text{-real } (cexpr\text{-sem } (case\text{-nat } x \sigma) e)$
shows *nonneg-cexpr (shift-var-set V) (case-nat t Γ) e*
by (*auto intro!: nonneg-cexprI assms dest!: space-state-measureD-shift*)

lemma *nonneg-cexpr-shift-iff*:
 $nonneg\text{-cexpr } (shift\text{-var}\text{-set } V) (case\text{-nat } t \Gamma) (map\text{-vars } Suc\ e) \iff nonneg\text{-cexpr } V \Gamma e$
apply (*auto simp: cexpr-sem-map-vars o-def nonneg-cexpr-def space-state-measure-shift-iff*)
subgoal for σ
apply (*drule bspec[of - - case-nat (SOME x. x ∈ type-universe t) σ]*)
using *type-universe-nonempty[of t]*
unfolding *ex-in-conv[symmetric]*
apply (*auto intro!: case-nat-in-state-measure intro: someI*)
done
done

lemma *case-nat-case-nat*: $case\text{-nat } x\ n (case\text{-nat } y\ m\ i) = case\text{-nat } (case\text{-nat } x\ n\ y) (\lambda i'. case\text{-nat } x\ n (m\ i'))\ i$
by (*rule nat.case-distrib*)

lemma *nonneg-cexpr-shift-iff2*:

```

nonneg-cexpr (shift-var-set (shift-var-set V))
  (case-nat t1 (case-nat t2  $\Gamma$ )) (map-vars (case-nat 0 ( $\lambda x. \text{Suc } (\text{Suc } x)$ )) e)  $\longleftrightarrow$ 
  nonneg-cexpr (shift-var-set V) (case-nat t1  $\Gamma$ ) e
apply (auto simp: cexpr-sem-map-vars o-def nonneg-cexpr-def space-state-measure-shift-iff)
subgoal for x  $\sigma$ 
  apply (drule bspec[of - - case-nat x (case-nat (SOME x. x  $\in$  type-universe t2)
 $\sigma$ ))]
  using type-universe-nonempty[of t2]
  unfolding ex-in-conv[symmetric]
  apply (auto simp: case-nat-case-nat cong: nat.case-cong
    intro!: case-nat-in-state-measure intro: someI-ex someI)
  done
apply (erule bspec)
subgoal for x1 x2  $\sigma$ 
  by (auto simp add: space-state-measure-shift-iff fun-eq-iff split: nat.split
    intro!: exI[of - x1] exI[of -  $\sigma$ ])
done

```

```

lemma nonneg-cexpr-Add:
  assumes  $\Gamma \vdash_c e1 : \text{REAL}$   $\Gamma \vdash_c e2 : \text{REAL}$ 
  assumes free-vars e1  $\subseteq V$  free-vars e2  $\subseteq V$ 
  assumes N1: nonneg-cexpr V  $\Gamma$  e1 and N2: nonneg-cexpr V  $\Gamma$  e2
  shows nonneg-cexpr V  $\Gamma$  (e1 +c e2)
proof (rule nonneg-cexprI)
  fix  $\sigma$  assume  $\sigma$ :  $\sigma \in \text{space } (\text{state-measure } V \Gamma)$ 
  hence extract-real (cexpr-sem  $\sigma$  (e1 +c e2)) = extract-real (cexpr-sem  $\sigma$  e1) +
  extract-real (cexpr-sem  $\sigma$  e2)
  using assms by (subst cexpr-sem-Add[of  $\Gamma$  - - - V]) simp-all
  also have ...  $\geq 0$  using  $\sigma$  N1 N2 by (intro add-nonneg-nonneg nonneg-cexprD)
  finally show extract-real (cexpr-sem  $\sigma$  (e1 +c e2))  $\geq 0$  .
qed

```

```

lemma expr-has-density-cexpr-sound-aux:
  assumes  $\Gamma \vdash e : t$  ( $vs, vs', \Gamma, \delta$ )  $\vdash_c e \Rightarrow f$  cdens-ctxt-invar vs vs'  $\Gamma \delta$ 
  free-vars e  $\subseteq \text{set } vs \cup \text{set } vs'$ 
  shows dens-ctxt- $\alpha$  (vs, vs',  $\Gamma, \delta$ )  $\vdash_d e \Rightarrow \text{eval-cexpr } f \wedge \text{is-density-cexpr } (vs, vs', \Gamma, \delta)$ 
  t f
using assms(2,1,3,4)
proof (induction arbitrary: t rule: expr-has-density-cexpr.induct[split-format (complete)])

```

```

  case (edc-val v vs vs'  $\Gamma \delta$ )
  from edc-val.prem have [simp]: t = val-type v by auto
  note invar = cdens-ctxt-invarD[OF edc-val.prem(2)]
  let ?e1 = map-vars Suc (branch-prob-cexpr (vs, vs',  $\Gamma, \delta$ )) and ?e2 =  $\langle \text{CVar } 0$ 
  =c CVal v  $\rangle_c$ 
  have ctype1: case-nat t  $\Gamma \vdash_c ?e1 : \text{REAL}$  and ctype2: case-nat t  $\Gamma \vdash_c ?e2$ :
  REAL using invar
  by (auto intro!: cexpr-typing.intros cexpr-typing-map-vars simp: o-def)
  hence ctype: case-nat t  $\Gamma \vdash_c ?e1 *_c ?e2 : \text{REAL}$  by (auto intro!: cexpr-typing.intros)

```

```

{
  fix  $\varrho$   $x$  assume  $x: x \in \text{type-universe } (\text{val-type } v)$ 
    and  $\varrho: \varrho \in \text{space } (\text{state-measure } (\text{set } \text{vs}') \Gamma)$ 
    hence  $\text{case-nat } x \varrho \in \text{space } (\text{state-measure } (\text{shift-var-set } (\text{set } \text{vs}')) (\text{case-nat } (\text{val-type } v) \Gamma))$ 
    by (rule case-nat-in-state-measure)
    hence  $\text{ennreal } (\text{eval-cexpr } (?e1 *_c ?e2) \varrho x) =$ 
       $\text{ennreal } (\text{extract-real } (\text{cexpr-sem } (\text{case-nat } x \varrho)$ 
         $(\text{map-vars } \text{Suc } (\text{branch-prob-cexpr } (\text{vs}, \text{vs}', \Gamma, \delta)))) *$ 
         $\text{ennreal } (\text{extract-real } (\text{RealVal } (\text{bool-to-real } (x = v)))) (\text{is } - = ?a * ?b)$ 
      )
    using invar unfolding eval-cexpr-def
    apply (subst ennreal-mult''[symmetric])
    apply (simp add: bool-to-real-def)
    apply (subst cexpr-sem-Mult[of case-nat t  $\Gamma$  - - shift-var-set (set vs')])
    apply (insert invar ctype1 ctype2)
    apply (auto simp: shift-var-set-def)
    done
    also have  $?a = \text{branch-prob } (\text{dens-ctxt-}\alpha (\text{vs}, \text{vs}', \Gamma, \delta)) \varrho$ 
    by (subst cexpr-sem-map-vars, subst cexpr-sem-branch-prob) (simp-all add: o-def  $\varrho$  edc-val.premis)
    also have  $?b = \text{indicator } \{v\} x$ 
    by (simp add: extract-real-def bool-to-real-def split: split-indicator)
    finally have  $\text{ennreal } (\text{eval-cexpr } (?e1 *_c ?e2) \varrho x) =$ 
       $\text{branch-prob } (\text{dens-ctxt-}\alpha (\text{vs}, \text{vs}', \Gamma, \delta)) \varrho * \text{indicator } \{v\} x .$ 
} note  $e = \text{this}$ 

have  $\text{meas}: (\lambda(\sigma, x). \text{ennreal } (\text{eval-cexpr } (?e1 *_c ?e2) \sigma x))$ 
   $\in \text{borel-measurable } (\text{state-measure } (\text{set } \text{vs}') \Gamma \otimes_M \text{stock-measure } (\text{val-type } v))$ 
apply (subst measurable-split-conv, rule measurable-compose[OF - measurable-ennreal])
apply (subst measurable-split-conv[symmetric], rule measurable-eval-cexpr)
apply (insert ctype invar, auto simp: shift-var-set-def)
done

have  $*$ :  $\text{Suc } - ' \text{shift-var-set } (\text{set } \text{vs}') = \text{set } \text{vs}' \text{ case-nat } (\text{val-type } v) \Gamma \circ \text{Suc} = \Gamma$ 
by (auto simp: shift-var-set-def)

have  $\text{nn}$ :  $\text{nonneg-cexpr } (\text{shift-var-set } (\text{set } \text{vs}')) (\text{case-nat } t \Gamma)$ 
   $(\text{map-vars } \text{Suc } (\text{branch-prob-cexpr } (\text{vs}, \text{vs}', \Gamma, \delta)) *_c \langle \text{CVar } 0 =_c \text{CVal } v \rangle_c)$ 
using invar ctype1 ctype2
by (fastforce intro!: nonneg-cexpr-Mult nonneg-indicator nonneg-cexpr-map-vars
  cexpr-typing.intros nonneg-cexpr-sem-integrate-vars'
  simp: branch-prob-cexpr-def *)

show  $?case$  unfolding dens-ctxt- $\alpha$ -def
apply (simp only: prod.case, intro conjI)
apply (rule hd-AE[OF hd-val et-val AE-I2])

```

```

apply (insert edc-val, simp-all add: e dens-ctxt- $\alpha$ -def meas) [4]
apply (intro is-density-exprI)
using ctype
apply simp
apply (insert invar nn, auto simp: shift-var-set-def)
done
next

case (edc-var x vs vs'  $\Gamma$   $\delta$  t)
hence t: t =  $\Gamma$  x by auto
note invar = cdens-ctxt-invarD[OF edc-var.prem(2)]
from invar have ctype: case-nat t  $\Gamma \vdash_c$  marg-dens-cexpr  $\Gamma$  vs x  $\delta$  : REAL by
(auto simp: t)

show ?case unfolding dens-ctxt- $\alpha$ -def
proof (simp only: prod.case, intro conjI is-density-exprI, rule hd-AE[OF hd-var
edc-var.prem(1)])
show case-nat t  $\Gamma \vdash_c$  marg-dens-cexpr  $\Gamma$  vs x  $\delta$  : REAL by fact
next
show free-vars (marg-dens-cexpr  $\Gamma$  vs x  $\delta$ )  $\subseteq$  shift-var-set (set vs')
using edc-var.prem(2) by (rule free-vars-marg-dens-cexpr)
next
have free-vars: free-vars (marg-dens-cexpr  $\Gamma$  vs x  $\delta$ )  $\subseteq$  shift-var-set (set vs')
using edc-var.prem(2) by (rule free-vars-marg-dens-cexpr)
show ( $\lambda(\rho, y).$  ennreal (eval-cexpr (marg-dens-cexpr  $\Gamma$  vs x  $\delta$ )  $\rho$  y))
 $\in$  borel-measurable (state-measure (set vs')  $\Gamma \otimes_M$  stock-measure t)
apply (subst measurable-split-conv, rule measurable-compose[OF - measurable-
ennreal])
apply (subst measurable-split-conv[symmetric], rule measurable-eval-cexpr)
apply (insert ctype free-vars, auto simp: shift-var-set-def)
done
next
fix  $\rho$  assume  $\rho \in$  space (state-measure (set vs')  $\Gamma$ )
hence AE y in stock-measure t.
marg-dens (dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )) x  $\rho$  y =
ennreal (eval-cexpr (marg-dens-cexpr  $\Gamma$  vs x  $\delta$ )  $\rho$  y)
using edc-var unfolding eval-cexpr-def by (subst t, subst eq-commute, intro
cexpr-sem-marg-dens)
thus AE y in stock-measure t.
marg-dens (set vs, set vs',  $\Gamma$ ,  $\lambda x.$  ennreal (extract-real (cexpr-sem x  $\delta$ )))
x  $\rho$  y =
ennreal (eval-cexpr (marg-dens-cexpr  $\Gamma$  vs x  $\delta$ )  $\rho$  y)
by (simp add: dens-ctxt- $\alpha$ -def)
next
show x  $\in$  set vs
by (insert edc-var.prem edc-var.hyps, auto simp: eval-cexpr-def intro!: non-
neg-cexpr-sem-marg-dens)
show nonneg-cexpr (shift-var-set (set vs')) (case-nat t  $\Gamma$ ) (marg-dens-cexpr  $\Gamma$ 
vs x  $\delta$ )

```

```

    by (intro nonneg-cexprI-shift nonneg-cexpr-sem-marg-dens[OF edc-var.prem(2)]
        ⟨x ∈ set vs⟩)
      (auto simp: t)
  qed
next
case (edc-pair x vs y vs' Γ δ t)
hence t[simp]: t = PRODUCT (Γ x) (Γ y) by auto
note invar = cdens-ctxt-invarD[OF edc-pair.prem(2)]
from invar have ctype: case-nat t Γ ⊢c marg-dens2-cexpr Γ vs x y δ : REAL by
auto
from edc-pair.prem have vars: free-vars (marg-dens2-cexpr Γ vs x y δ) ⊆
shift-var-set (set vs')
using free-vars-marg-dens2-cexpr by simp

show ?case unfolding dens-ctxt-α-def
proof (simp only: prod.case, intro conjI is-density-exprI, rule hd-AE[OF hd-pair
edc-pair.prem(1)])
  fix ρ assume ρ: ρ ∈ space (state-measure (set vs') Γ)
  show AE z in stock-measure t.
    marg-dens2 (set vs, set vs', Γ, λx. ennreal (extract-real (cexpr-sem x
δ))) x y ρ z =
      ennreal (eval-cexpr (marg-dens2-cexpr Γ vs x y δ) ρ z)
  using cexpr-sem-marg-dens2[OF edc-pair.prem(2) edc-pair.hyps ρ] unfolding
eval-cexpr-def
  by (subst t, subst eq-commute) (simp add: dens-ctxt-α-def)
next
show nonneg-cexpr (shift-var-set (set vs')) (case-nat t Γ) (marg-dens2-cexpr Γ
vs x y δ)
  by (intro nonneg-cexprI-shift nonneg-cexpr-sem-marg-dens2[OF edc-pair.prem(2)]
      ⟨x ∈ set vs⟩ ⟨y ∈ set vs⟩)
    auto
  qed (insert edc-pair invar ctype vars, auto simp: dens-ctxt-α-def)
next
case (edc-fail vs vs' Γ δ t t')
hence [simp]: t = t' by auto
have ctype: case-nat t' Γ ⊢c CReal 0 : REAL
  by (subst val-type.simps[symmetric]) (rule cexpr-typing.intros)
thus ?case by (auto simp: dens-ctxt-α-def eval-cexpr-def extract-real-def
zero-ennreal-def[symmetric] hd-fail
intro!: is-density-exprI nonneg-cexprI)
next
case (edc-let vs vs' Γ e f δ e' g t)
then obtain t' where t1: Γ ⊢ e : t' and t2: case-nat t' Γ ⊢ e' : t by auto
note invar = cdens-ctxt-invarD[OF edc-let.prem(2)]
from t1 have t1': the (expr-type Γ e) = t' by (auto simp: expr-type-Some-iff[symmetric])
have dens1: dens-ctxt-α ([], vs @ vs', Γ, CReal 1) ⊢d e ⇒
  (λx xa. ennreal (eval-cexpr f x xa)) and

```

```

wf1: is-density-expr ([], vs @ vs', Γ, CReal 1) t' f
using edc-let.IH(1)[OF t1] edc-let.premis by (auto dest: cdens-ctxt-invar-empty)

have invf: cdens-ctxt-invar (shift-vars vs) (map Suc vs') (case-nat t' Γ) (map-vars
Suc δ *c f)
using edc-let.premis edc-let.hyps dens1 wf1 invar
by (intro cdens-ctxt-invar-insert[OF - t1]) (auto simp: dens-ctxt-α-def)

let ?Y = (shift-vars vs, map Suc vs', case-nat t' Γ, map-vars Suc δ *c f)
have set (shift-vars vs) ∪ set (map Suc vs') = shift-var-set (set vs ∪ set vs')
by (simp add: shift-var-set-def image-Un)
hence dens-ctxt-α (shift-vars vs, map Suc vs', case-nat t' Γ, map-vars Suc δ *c
f) ⊢d
    e' ⇒ (λx xa. ennreal (eval-cexpr g x xa)) ∧
    is-density-expr (shift-vars vs, map Suc vs', case-nat t' Γ, map-vars Suc δ *c
f) t g
using invf t2 edc-let.premis subset-shift-var-set
by (simp only: t1 [symmetric], intro edc-let.IH(2)) simp-all
hence dens2: dens-ctxt-α ?Y ⊢d e' ⇒ (λx xa. ennreal (eval-cexpr g x xa)) and
    wf2: is-density-expr (shift-vars vs, map Suc vs', case-nat t' Γ, map-vars Suc
δ *c f) t g
by simp-all

have cexpr-eq: cexpr-sem (case-nat x ρ ∘ (λx. x - Suc 0)) g =
    cexpr-sem (case-nat x (case-nat undefined ρ)) g for x ρ
using is-density-exprD[OF wf2]
by (intro cexpr-sem-eq-on-vars) (auto split: nat.split simp: shift-var-set-def)

have [simp]: ∧σ. case-nat (σ 0) (λx. σ (Suc x)) = σ by (intro ext) (simp split:
nat.split)
hence (shift-var-set (set vs), Suc ' set vs', case-nat t' Γ,
    insert-dens (set vs) (set vs') (λx xa. ennreal (eval-cexpr f x xa))
    (λx. ennreal (extract-real (cexpr-sem x δ))))
    ⊢d e' ⇒ (λa aa. ennreal (eval-cexpr g a aa)) using dens2
apply (simp only: dens-ctxt-α-def prod.case set-shift-vars set-map)
apply (erule hd-dens-ctxt-cong)
apply (insert invar is-density-exprD[OF wf1])
unfolding insert-dens-def
apply (subst ennreal-mult' [symmetric])
apply (erule nonneg-cexprD)
apply (rule measurable-space[OF measurable-remove-var [where t=t]])
apply simp
apply (simp add: shift-var-set-def image-Un)
apply (subst cexpr-sem-Mult [of case-nat t' Γ])
apply (auto intro!: cexpr-typing-map-vars simp: o-def shift-var-set-def image-Un
    cexpr-sem-map-vars insert-dens-def eval-cexpr-def remove-var-def)
done

hence dens-ctxt-α (vs, vs', Γ, δ) ⊢d LET e IN e' ⇒

```

$(\lambda \varrho x. \text{ennreal } (\text{eval-cexpr } g \text{ (case-nat undefined } \varrho) x))$
unfolding *dens-ctxt- α -def*
by (*simp only: prod.case, intro hd-let*[**where** $f = \lambda x xa. \text{ennreal } (\text{eval-cexpr } f x xa)$])
(insert dens1 dens2, simp-all add: dens-ctxt- α -def extract-real-def one-ennreal-def t1')
hence *dens-ctxt- α* (vs, vs', Γ, δ) $\vdash_d LET e IN e' \Rightarrow$
 $(\lambda \varrho x. \text{ennreal } (\text{eval-cexpr } (\text{map-vars } (\lambda x. x - 1) g) \varrho x))$
proof (*simp only: dens-ctxt- α -def prod.case, erule-tac hd-cong*[*OF - - edc-let.prem*s(1,3)])
fix ϱx **assume** $\varrho: \varrho \in \text{space } (\text{state-measure } (\text{set } vs') \Gamma)$
and $x: x \in \text{space } (\text{stock-measure } t)$
have $\text{eval-cexpr } (\text{map-vars } (\lambda x. x - 1) g) \varrho x =$
 $\text{extract-real } (\text{cexpr-sem } (\text{case-nat } x \varrho \circ (\lambda x. x - \text{Suc } 0))) g$
unfolding *eval-cexpr-def* **by** (*simp add: cexpr-sem-map-vars*)
also note *cexpr-eq*[*of* $x \varrho$]
finally show $\text{ennreal } (\text{eval-cexpr } g \text{ (case-nat undefined } \varrho) x) =$
 $\text{ennreal } (\text{eval-cexpr } (\text{map-vars } (\lambda x. x - 1) g) \varrho x)$
by (*simp add: eval-cexpr-def*)
qed (*simp-all add: density-context- α* [*OF edc-let.prem*s(2)])
moreover have *is-density-expr* (vs, vs', Γ, δ) t (*map-vars* ($\lambda x. x - 1$) g)
proof (*intro is-density-exprI*)
note $wf = \text{is-density-exprD}$ [*OF wf2*]
show *case-nat* $t \Gamma \vdash_c \text{map-vars } (\lambda x. x - 1) g : REAL$
by (*rule cexpr-typing-map-vars, rule cexpr-typing-cong'*[*OF wf*(1)])
(insert wf(2), auto split: nat.split simp: shift-var-set-def)
from $wf(2)$ **show** *free-vars* (*map-vars* ($\lambda x. x - 1$) g)
 $\subseteq \text{shift-var-set } (\text{set } vs')$
by (*auto simp: shift-var-set-def*)
next
show *nonneg-cexpr* (*shift-var-set* ($\text{set } vs'$)) (*case-nat* $t \Gamma$) (*map-vars* ($\lambda x. x - 1$) g)
apply (*intro nonneg-cexprI-shift*)
apply (*simp add: cexpr-sem-map-vars cexpr-eq*)
apply (*rule nonneg-cexprD*[*OF wf2*][*THEN is-density-exprD-nonneg*])
apply (*auto simp: space-state-measure PiE-iff extensional-def split: nat.splits*)
done
qed
ultimately show *?case* **by** (*rule conjI*)
next
case (*edc-rand* $vs vs' \Gamma \delta e f \text{dst } t'$)
define t **where** $t = \text{dist-param-type } \text{dst}$
note $\text{invar} = \text{cdens-ctxt-invarD}$ [*OF edc-rand.prem*s(2)]
from *edc-rand* **have** $t1: \Gamma \vdash e : t$ **and** $t2: t' = \text{dist-result-type } \text{dst}$ **by** (*auto simp: t-def*)
have $\text{dens: dens-ctxt-}\alpha$ (vs, vs', Γ, δ) $\vdash_d e \Rightarrow (\lambda x xa. \text{ennreal } (\text{eval-cexpr } f x xa))$
and
 $wf: \text{is-density-expr } (vs, vs', \Gamma, \delta) t f$ **using** *edc-rand t1 t2* **by** *auto*

```

from wf have tf: case-nat t  $\Gamma \vdash_c f : REAL$  and varsf: free-vars f  $\subseteq$  shift-var-set
(set vs')
  unfolding is-density-expr-def by simp-all
  let ?M = ( $\lambda \rho$ . dens-ctxt-measure (dens-ctxt- $\alpha$  (vs,vs', $\Gamma,\delta$ ))  $\rho \gg \equiv$  ( $\lambda \sigma$ . expr-sem
 $\sigma$  e))
  have dens': has-parametrized-subprob-density (state-measure (set vs')  $\Gamma$ ) ?M
(stock-measure t)
    ( $\lambda \rho$  x. ennreal (eval-cexpr f  $\rho$  x)) using dens t1 edc-rand.premis
  by (simp-all add: dens-ctxt- $\alpha$ -def expr-has-density-sound-aux density-context- $\alpha$ )

let ?shift = case-nat 0 ( $\lambda x$ . Suc (Suc x))
let ?e1 = map-vars ?shift f
let ?e2 = dist-dens-cexpr dst (CVar 0) (CVar 1)
let ?e = ( $\int_c$  ?e1 *c ?e2  $\partial t$ )
have [simp]:  $\bigwedge t t' \Gamma$ . case-nat t (case-nat t'  $\Gamma$ )  $\circ$  ?shift = case-nat t  $\Gamma$ 
  by (intro ext) (simp split: nat.split add: o-def)
have te1: case-nat t (case-nat t'  $\Gamma$ )  $\vdash_c$  ?e1 : REAL using tf
  by (auto intro!: cexpr-typing.intros cexpr-typing-dist-dens-cexpr cet-var'
cexpr-typing-map-vars simp: t-def t2)
have te2: case-nat t (case-nat t'  $\Gamma$ )  $\vdash_c$  ?e2 : REAL
  by (intro cexpr-typing-dist-dens-cexpr cet-var') (simp-all add: t-def t2)
have te: case-nat t'  $\Gamma \vdash_c$  ?e : REAL using te1 te2
  by (intro cet-int cet-op[where t = PRODUCT REAL REAL] cet-pair) (simp-all
add: t2 t-def)
have vars-e1: free-vars ?e1  $\subseteq$  shift-var-set (shift-var-set (set vs'))
  using varsf by (auto simp: shift-var-set-def)
have (case-nat 0 ( $\lambda x$ . Suc (Suc x)) - ' shift-var-set (shift-var-set (set vs')) =
shift-var-set (set vs') by (auto simp: shift-var-set-def split: nat.split-asm)
have nonneg-e1: nonneg-cexpr (shift-var-set (shift-var-set (set vs'))) (case-nat t
(case-nat t'  $\Gamma$ )) ?e1
  by (auto intro!: nonneg-cexprI wf[THEN is-density-exprD-nonneg, THEN non-
neg-cexprD] case-nat-in-state-measure
dest!: space-state-measureD-shift simp: cexpr-sem-map-vars)
have vars-e2: free-vars ?e2  $\subseteq$  shift-var-set (shift-var-set (set vs'))
  by (intro order.trans[OF free-vars-dist-dens-cexpr]) (auto simp: shift-var-set-def)
have nonneg-e2: nonneg-cexpr (shift-var-set (shift-var-set (set vs')))
(case-nat t (case-nat t'  $\Gamma$ )) ?e2
  by (intro nonneg-dist-dens-cexpr cet-var') (auto simp: t2 t-def shift-var-set-def)

let ?f =  $\lambda \rho$  x.  $\int^+ y$ . ennreal (eval-cexpr f  $\rho$  y) * dist-dens dst y  $\times$  stock-measure
t
  let ?M = ( $\lambda \rho$ . dens-ctxt-measure (dens-ctxt- $\alpha$  (vs,vs', $\Gamma,\delta$ ))  $\rho \gg \equiv$  ( $\lambda \sigma$ . expr-sem
 $\sigma$  (Random dst e)))
  have dens': dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )  $\vdash_d$  Random dst e  $\Rightarrow$  ?f using dens
  by (simp only: dens-ctxt- $\alpha$ -def prod.case t-def hd-rand[unfolded apply-dist-to-dens-def])
  hence dens'': has-parametrized-subprob-density (state-measure (set vs')  $\Gamma$ ) ?M
(stock-measure t') ?f
  using edc-rand.premis invar
  by (simp only: dens-ctxt- $\alpha$ -def prod.case, intro expr-has-density-sound-aux)

```

```

(auto intro!: density-context- $\alpha$ )

{
  fix  $\rho$  assume  $\rho: \rho \in \text{space } (\text{state-measure } (\text{set } \text{vs}') \Gamma)$ 
  fix  $x$  assume  $x: x \in \text{type-universe } t'$ 
  fix  $y$  assume  $y: y \in \text{type-universe } t$ 
  let  $?q'' = \text{case-nat } y (\text{case-nat } x \rho)$  and  $?T'' = \text{case-nat } t (\text{case-nat } t' \Gamma)$ 
  let  $?V'' = \text{shift-var-set } (\text{shift-var-set } (\text{set } \text{vs}'))$ 
  have  $q'': ?q'' \in \text{space } (\text{state-measure } (\text{shift-var-set } (\text{shift-var-set } (\text{set } \text{vs}')))) ?T''$ 
  using  $\rho x y$  by (intro case-nat-in-state-measure) simp-all
  have  $A: \text{extract-real } (\text{cexpr-sem } ?q'' (?e1 *_c ?e2)) =$ 
     $\text{extract-real } (\text{cexpr-sem } ?q'' ?e1) * \text{extract-real } (\text{cexpr-sem } ?q'' ?e2)$ 
  by (rule cexpr-sem-Mult[OF te1 te2  $q''$  vars-e1 vars-e2])
  also have  $\dots \geq 0$  using nonneg-e1 nonneg-e2  $q''$ 
  by (blast intro: mult-nonneg-nonneg dest: nonneg-cexprD)
  finally have  $B: \text{extract-real } (\text{cexpr-sem } ?q'' (?e1 *_c ?e2)) \geq 0$  .
  note  $A$ 
  hence  $\text{eval-cexpr } f \rho y * \text{dist-dens } \text{dst } y x = \text{extract-real } (\text{cexpr-sem } ?q'' (?e1$ 
 $*_c ?e2))$ 
  using  $q''$ 
  apply (subst  $A$ )
  apply (subst ennreal-mult'')
  using nonneg-e2
  apply (erule nonneg-cexprD)
  apply (subst cexpr-sem-dist-dens-cexpr[OF ?T'' - - - ?V''])
  apply (force simp: cexpr-sem-map-vars eval-cexpr-def t2 t-def intro!: cet-var')+
  done
  note  $B$ 
} note  $e1e2 = \text{this}$ 

{
  fix  $\rho$  assume  $\rho: \rho \in \text{space } (\text{state-measure } (\text{set } \text{vs}') \Gamma)$ 
  have  $AE x$  in stock-measure  $t'$ .
     $\text{apply-dist-to-dens } \text{dst } (\lambda \rho x. \text{ennreal } (\text{eval-cexpr } f \rho x)) \rho x = \text{eval-cexpr}$ 
 $?e \rho x$ 
  proof (rule AE-mp[OF - AE-I2[OF impI]])
    from has-parametrized-subprob-density-integral[OF dens''  $\rho$ ]
    have  $(\int^+ x. ?f \rho x \partial \text{stock-measure } t') \neq \infty$  by auto
    thus  $AE x$  in stock-measure  $t'$ .  $?f \rho x \neq \infty$ 
    using has-parametrized-subprob-densityD(3)[OF dens''  $\rho$ ]
    by (intro nn-integral-PInf-AE) simp-all
  next
  fix  $x$  assume  $x: x \in \text{space } (\text{stock-measure } t')$  and finite:  $?f \rho x \neq \infty$ 
  let  $?q' = \text{case-nat } x \rho$ 
  have  $q': ?q' \in \text{space } (\text{state-measure } (\text{shift-var-set } (\text{set } \text{vs}')) (\text{case-nat } t' \Gamma))$ 
  using  $\rho x$  by (intro case-nat-in-state-measure) simp-all
  hence  $*$ :  $(\int^+ y. \text{ennreal } (\text{eval-cexpr } f \rho y) * \text{dist-dens } \text{dst } y x \partial \text{stock-measure}$ 
 $t) =$ 

$$\int^+ y. \text{extract-real } (\text{cexpr-sem } (\text{case-nat } y ?q') (?e1 *_c ?e2))$$


```

```

∂stock-measure t (is - = ?I)
  using ρ x by (intro nn-integral-cong) (simp add: e1e2)
  also from * and finite have finite': ?I < ∞ by (simp add: less-top)
  have ?I = ennreal (eval-cexpr ?e ρ x) using ρ' te te1 te2 vars-e1 vars-e2
nonneg-e1 nonneg-e2
  unfolding eval-cexpr-def
  by (subst cexpr-sem-integral-nonneg[OF finite'])
    (auto simp: eval-cexpr-def t2 t-def intro!: nonneg-cexpr-Mult)
  finally show apply-dist-to-dens dst (λρ x. ennreal (eval-cexpr f ρ x)) ρ x =
    ennreal (eval-cexpr ?e ρ x)
  unfolding apply-dist-to-dens-def by (simp add: t-def)
qed
} note AE-eq = this

have meas: (λ(ρ, x). ennreal (eval-cexpr ?e ρ x))
  ∈ borel-measurable (state-measure (set vs') Γ ⊗M stock-measure t')
  apply (subst measurable-split-conv, rule measurable-compose[OF - measurable-ennreal])
  apply (subst measurable-split-conv[symmetric], rule measurable-eval-cexpr[OF te])
  apply (insert vars-e1 vars-e2, auto simp: shift-var-set-def)
  done
show ?case
proof (intro conjI is-density-exprI, simp only: dens-ctxt-α-def prod.case,
  rule hd-AE[OF hd-rand edc-rand.premis(1)])
  from dens show (set vs, set vs', Γ, λx. ennreal (extract-real (cexpr-sem x δ)))
  ⊢d
    e ⇒ (λx xa. ennreal (eval-cexpr f x xa))
  unfolding dens-ctxt-α-def by simp
next
  have nonneg-cexpr (shift-var-set (set vs')) (case-nat t' Γ) (∫c ?e1 *c ?e2 ∂t)
  by (intro nonneg-cexpr-int nonneg-cexpr-Mult nonneg-dist-dens-cexpr te1 te2
vars-e1 vars-e2 nonneg-e1)
    (auto simp: t-def t2 intro!: cet-var')
  then show nonneg-cexpr (shift-var-set (set vs')) (case-nat t' Γ)
    (∫c map-vars (case-nat 0 (λx. x + 2)) f *c ?e2 ∂dist-param-type dst)
  by (simp add: t-def)
qed (insert AE-eq meas te vars-e1 vars-e2, auto simp: t-def t2 shift-var-set-def)

next
case (edc-rand-det e vs' vs Γ δ dst t')
define t where t = dist-param-type dst
note invar = cdens-ctxt-invarD[OF edc-rand-det.premis(2)]
from edc-rand-det have t1: Γ ⊢ e : t and t2: t' = dist-result-type dst by (auto
simp: t-def)
let ?e1 = map-vars Suc (branch-prob-cexpr (vs, vs', Γ, δ)) and
  ?e2 = dist-dens-cexpr dst (map-vars Suc (expr-rf-to-cexpr e)) (CVar 0)
have ctype1: case-nat t' Γ ⊢c ?e1 : REAL
  using invar by (auto intro!: cexpr-typing-map-vars simp: o-def)

```

```

have vars2': free-vars (map-vars Suc (expr-rf-to-cexpr e))  $\subseteq$  shift-var-set (set vs')
unfolding shift-var-set-def using free-vars-expr-rf-to-cexpr edc-rand-det.hyps
by auto
have vars2: free-vars ?e2  $\subseteq$  shift-var-set (free-vars e)
unfolding shift-var-set-def using free-vars-expr-rf-to-cexpr edc-rand-det.hyps
by (intro order.trans[OF free-vars-dist-dens-cexpr]) auto
have ctype2: case-nat t'  $\Gamma \vdash_c$  ?e2 : REAL using t1 edc-rand-det.hyps
by (intro cexpr-typing-dist-dens-cexpr cexpr-typing-map-vars)
    (auto simp: o-def t-def t2 intro!: cet-var')

have nonneg-e2: nonneg-cexpr (shift-var-set (set vs')) (case-nat t'  $\Gamma$ ) ?e2
using t1 ‹randomfree e› free-vars-expr-rf-to-cexpr[of e] edc-rand-det.hyps
apply (intro nonneg-dist-dens-cexpr cexpr-typing-map-vars)
apply (auto simp add: o-def t-def t2 intro!: cet-var')
done

have nonneg-e1: nonneg-cexpr (shift-var-set (set vs')) (case-nat t'  $\Gamma$ ) ?e1
using invar
by (auto simp add: branch-prob-cexpr-def nonneg-cexpr-shift-iff intro!: non-
neg-cexpr-sem-integrate-vars')

{
  fix  $\varrho$  x
  assume x: x  $\in$  type-universe t' and  $\varrho$ :  $\varrho \in$  space (state-measure (set vs')  $\Gamma$ )
  hence  $\varrho'$ : case-nat x  $\varrho \in$  space (state-measure (shift-var-set (set vs')) (case-nat
t'  $\Gamma$ ))
  by (rule case-nat-in-state-measure)
  hence eval-cexpr (?e1 *c ?e2)  $\varrho$  x =
    ennreal (extract-real (cexpr-sem (case-nat x  $\varrho$ )
      (map-vars Suc (branch-prob-cexpr (vs, vs',  $\Gamma$ ,  $\delta$ )))) *
    ennreal (extract-real (cexpr-sem (case-nat x  $\varrho$ ) ?e2)) (is - = ?a * ?b)
  using invar
  apply (subst ennreal-mult''[symmetric])
  apply (rule nonneg-cexprD[OF nonneg-e2])
  apply simp
  unfolding eval-cexpr-def
  apply (subst cexpr-sem-Mult[of case-nat t'  $\Gamma$  - - - shift-var-set (set vs')])
  apply (insert invar ctype1 vars2 ctype2 edc-rand-det.hyps(2))
  apply (auto simp: shift-var-set-def)
  done
  also have ?a = branch-prob (dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ ))  $\varrho$  (is - = ?c)
  by (subst cexpr-sem-map-vars, subst cexpr-sem-branch-prob) (simp-all add:
o-def  $\varrho$  edc-rand-det.premis)
  also have ?b = dist-dens dst (expr-sem-rf  $\varrho$  e) x (is - = ?d) using t1
edc-rand-det.hyps
  by (subst cexpr-sem-dist-dens-cexpr[of case-nat t'  $\Gamma$ ], insert  $\varrho'$  vars2')
    (auto intro!: cexpr-typing-map-vars cet-var')
    (simp: o-def t-def t2 cexpr-sem-map-vars cexpr-sem-expr-rf-to-cexpr)

```

```

finally have A: ennreal (eval-cexpr (?e1 *c ?e2) ρ x) = ?c * ?d .
} note A = this

have meas: (λ(ρ, x). ennreal (eval-cexpr (?e1 *c ?e2) ρ x))
  ∈ borel-measurable (state-measure (set vs') Γ ⊗M stock-measure t')
using ctype1 ctype2 vars2 invar edc-rand-det.hyps
by (subst measurable-split-conv, intro measurable-compose[OF - measurable-ennreal],
  subst measurable-split-conv[symmetric], intro measurable-eval-cexpr)
  (auto intro!: cexpr-typing.intros simp: shift-var-set-def)
from ctype1 ctype2 vars2 invar edc-rand-det.hyps
have wf: is-density-expr (vs, vs', Γ, δ) t' (?e1 *c ?e2)
proof (intro is-density-exprI)
  show nonneg-cexpr (shift-var-set (set vs')) (case-nat t' Γ) (?e1 *c ?e2)
  using invar(2)
  order-trans[OF free-vars-expr-rf-to-cexpr[OF ‹randomfree e›] ‹free-vars e ⊆
set vs'›]
  by (intro nonneg-cexpr-Mult ctype1 ctype2 nonneg-e2 nonneg-e1
  free-vars-dist-dens-cexpr[THEN order-trans])
  (auto simp: intro: order-trans)
qed (auto intro!: cexpr-typing.intros simp: shift-var-set-def)
show ?case using edc-rand-det.prem1 edc-rand-det.hyps meas wf A
apply (intro conjI, simp add: dens-ctxt-α-def)
apply (intro hd-AE[OF hd-rand-det[OF edc-rand-det.hyps] edc-rand-det.prem1(1)
AE-I2])
apply (simp-all add: dens-ctxt-α-def)
done

next
case (edc-if-det b vs vs' Γ δ e1 f1 e2 f2 t)
hence tb: Γ ⊢ b : BOOL and t1: Γ ⊢ e1 : t and t2: Γ ⊢ e2 : t by auto
from edc-if-det have b: randomfree b free-vars b ⊆ set vs ∪ set vs' by simp-all
note invar = cdens-ctxt-invarD[OF edc-if-det.prem1(2)]

let ?ind1 = ‹expr-rf-to-cexpr b›c and ?ind2 = ‹¬c expr-rf-to-cexpr b›c
have tind1: Γ ⊢c ?ind1 : REAL and tind2: Γ ⊢c ?ind2 : REAL
using edc-if-det.hyps tb by (auto intro!: cexpr-typing.intros)
have tδ1: Γ ⊢c δ *c ?ind1 : REAL and tδ2: Γ ⊢c δ *c ?ind2 : REAL
using invar(3) edc-if-det.hyps tb by (auto intro!: cexpr-typing.intros)
have nonneg-ind1: nonneg-cexpr (set vs ∪ set vs') Γ ?ind1 and
  nonneg-ind2: nonneg-cexpr (set vs ∪ set vs') Γ ?ind2
using tind1 tind2 edc-if-det.hyps tb
by (auto intro!: nonneg-cexprI simp: cexpr-sem-expr-rf-to-cexpr bool-to-real-def
extract-real-def
  dest: val-type-expr-sem-rf[OF tb b] elim!: BOOL-E split: if-split)
have subprob1: subprob-cexpr (set vs) (set vs') Γ (δ *c ?ind1) and
  subprob2: subprob-cexpr (set vs) (set vs') Γ (δ *c ?ind2)
using invar tb edc-if-det.hyps edc-if-det.prem1 free-vars-expr-rf-to-cexpr[OF
edc-if-det.hyps(1)]
by (auto intro!: subprob-indicator cet-op)

```

```

have vars1: free-vars ( $\delta *_{\mathcal{C}} ?ind1$ )  $\subseteq$  set vs  $\cup$  set vs' and
  vars2: free-vars ( $\delta *_{\mathcal{C}} ?ind2$ )  $\subseteq$  set vs  $\cup$  set vs'
using invar edc-if-det.hyps edc-if-det.premis free-vars-expr-rf-to-cexpr by auto
have inv1: cdens-ctxt-invar vs vs'  $\Gamma$  ( $\delta *_{\mathcal{C}} ?ind1$ )
using invar edc-if-det.hyps edc-if-det.premis tind1 t $\delta$ 1 subprob1 nonneg-ind1
vars1
by (intro cdens-ctxt-invarI nonneg-cexpr-Mult) auto
have inv2: cdens-ctxt-invar vs vs'  $\Gamma$  ( $\delta *_{\mathcal{C}} ?ind2$ )
using invar edc-if-det.hyps edc-if-det.premis tind2 t $\delta$ 2 subprob2 nonneg-ind2
vars2
by (intro cdens-ctxt-invarI nonneg-cexpr-Mult) auto
have dens1: dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta *_{\mathcal{C}} ?ind1$ )  $\vdash_d e1 \Rightarrow (\lambda \varrho x. \text{eval-cexpr } f1 \ \varrho \ x)$  and
  wf1: is-density-expr (vs, vs',  $\Gamma$ ,  $\delta *_{\mathcal{C}} ?ind1$ ) t f1
using edc-if-det.IH(1)[OF t1 inv1] edc-if-det.premis by auto
have dens2: dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta *_{\mathcal{C}} ?ind2$ )  $\vdash_d e2 \Rightarrow (\lambda \varrho x. \text{eval-cexpr } f2 \ \varrho \ x)$  and
  wf2: is-density-expr (vs, vs',  $\Gamma$ ,  $\delta *_{\mathcal{C}} ?ind2$ ) t f2
using edc-if-det.IH(2)[OF t2 inv2] edc-if-det.premis by auto

show ?case
proof (rule conjI, simp only: dens-ctxt- $\alpha$ -def prod.case, rule hd-cong[OF hd-if-det])
  let ? $\mathcal{Y}$  = (set vs, set vs',  $\Gamma$ , if-dens-det ( $\lambda x. \text{ennreal (extract-real (cexpr-sem } x \ \delta))$ )) b True)
show ? $\mathcal{Y} \vdash_d e1 \Rightarrow (\lambda \varrho x. \text{eval-cexpr } f1 \ \varrho \ x)$ 
proof (rule hd-dens-ctxt-cong)
  let ? $\delta$  =  $\lambda \sigma. \text{ennreal (extract-real (cexpr-sem } \sigma \ (\delta *_{\mathcal{C}} ?ind1))$ )
show (set vs, set vs',  $\Gamma$ , ? $\delta$ )  $\vdash_d e1 \Rightarrow (\lambda \varrho x. \text{ennreal (eval-cexpr } f1 \ \varrho \ x))$ 
using dens1 by (simp add: dens-ctxt- $\alpha$ -def)
fix  $\sigma$  assume  $\sigma$ :  $\sigma \in \text{space (state-measure (set vs } \cup \text{ set vs') } \Gamma)$ 
have extract-real (cexpr-sem  $\sigma$  ( $\delta *_{\mathcal{C}} ?ind1$ )) =
  extract-real (cexpr-sem  $\sigma$   $\delta$ ) * extract-real (cexpr-sem  $\sigma$  ?ind1) using
invar vars1
by (subst cexpr-sem-Mult[OF invar(3) tind1  $\sigma$ ]) simp-all
also have extract-real (cexpr-sem  $\sigma$  ?ind1) = (if expr-sem-rf  $\sigma$  b = TRUE
then 1 else 0)
using edc-if-det.hyps val-type-expr-sem-rf[OF tb b  $\sigma$ ]
by (auto simp: cexpr-sem-expr-rf-to-cexpr extract-real-def bool-to-real-def
elim!: BOOL-E)
finally show ? $\delta$   $\sigma$  = if-dens-det ( $\lambda \sigma. \text{ennreal (extract-real (cexpr-sem } \sigma \ \delta))$ )
b True  $\sigma$ 
by (simp add: if-dens-det-def)
qed
next
let ? $\mathcal{Y}$  = (set vs, set vs',  $\Gamma$ , if-dens-det ( $\lambda x. \text{ennreal (extract-real (cexpr-sem } x \ \delta))$ )) b False)
show ? $\mathcal{Y} \vdash_d e2 \Rightarrow (\lambda \varrho x. \text{eval-cexpr } f2 \ \varrho \ x)$ 
proof (rule hd-dens-ctxt-cong)
  let ? $\delta$  =  $\lambda \sigma. \text{ennreal (extract-real (cexpr-sem } \sigma \ (\delta *_{\mathcal{C}} ?ind2))$ )

```

```

show (set vs, set vs', Γ, ?δ) ⊢d e2 ⇒ (λρ x. ennreal (eval-cexpr f2 ρ x))
  using dens2 by (simp add: dens-ctxt-α-def)
fix σ assume σ: σ ∈ space (state-measure (set vs ∪ set vs') Γ)
have extract-real (cexpr-sem σ (δ *c ?ind2)) =
  extract-real (cexpr-sem σ δ) * extract-real (cexpr-sem σ ?ind2) using
invar vars1
  by (subst cexpr-sem-Mult[OF invar(3) tind2 σ]) simp-all
also have extract-real (cexpr-sem σ ?ind2) = (if expr-sem-rf σ b = FALSE
then 1 else 0)
  using edc-if-det.hyps val-type-expr-sem-rf[OF tb b σ]
  by (auto simp: cexpr-sem-expr-rf-to-cexpr extract-real-def bool-to-real-def
elim!: BOOL-E)
  finally show ?δ σ = if-dens-det (λσ. ennreal (extract-real (cexpr-sem σ δ)))
b False σ
  by (simp add: if-dens-det-def)
qed
next
fix ρ x assume ρ: ρ ∈ space (state-measure (set vs') Γ) and x : x ∈ space
(stock-measure t)
  hence eval-cexpr (f1 +c f2) ρ x = eval-cexpr f1 ρ x + eval-cexpr f2 ρ x
  using wf1 wf2 unfolding eval-cexpr-def is-density-expr-def
  by (subst cexpr-sem-Add[where Γ = case-nat t Γ and V = shift-var-set (set
vs^)] auto
moreover have 0 ≤ eval-cexpr f1 ρ x 0 ≤ eval-cexpr f2 ρ x
  unfolding eval-cexpr-def
  using ρ x wf1 [THEN is-density-exprD-nonneg, THEN nonneg-cexprD] wf2 [THEN
is-density-exprD-nonneg, THEN nonneg-cexprD]
  unfolding space-state-measure-shift-iff by auto
  ultimately show ennreal (eval-cexpr f1 ρ x) + ennreal (eval-cexpr f2 ρ x) =
ennreal (eval-cexpr (f1 +c f2) ρ x)
  by simp
next
show is-density-expr (vs, vs', Γ, δ) t (f1 +c f2) using wf1 wf2
using wf1 [THEN is-density-exprD-nonneg] wf2 [THEN is-density-exprD-nonneg]
  by (auto simp: is-density-expr-def intro!: cet-op[where t = PRODUCT REAL
REAL] cet-pair nonneg-cexpr-Add)
qed (insert edc-if-det.premis edc-if-det.hyps, auto intro!: density-context-α)

next
case (edc-if vs vs' Γ b f δ e1 g1 e2 g2 t)
hence tb: Γ ⊢ b : BOOL and t1: Γ ⊢ e1 : t and t2: Γ ⊢ e2 : t by auto
note invar = cdens-ctxt-invarD[OF edc-if.premis(2)]

  have densb: dens-ctxt-α ([], vs @ vs', Γ, CReal 1) ⊢d b ⇒ (λρ b. ennreal
(eval-cexpr f ρ b)) and
    wfb: is-density-expr ([], vs @ vs', Γ, CReal 1) BOOL f
  using edc-if.IH(1)[OF tb] edc-if.premis by (simp-all add: cdens-ctxt-invar-empty)
have inv1: cdens-ctxt-invar vs vs' Γ (δ *c cexpr-subst-val f TRUE) and
  inv2: cdens-ctxt-invar vs vs' Γ (δ *c cexpr-subst-val f FALSE)

```

using *tb densb wfb edc-if.prem*s **by** (*auto intro!*: *cdens-ctxt-invar-insert-bool*)
let $?\delta 1 = \text{cexpr-subst-val } f \text{ TRUE}$ **and** $?\delta 2 = \text{cexpr-subst-val } f \text{ FALSE}$
have $t\delta 1: \Gamma \vdash_c \delta *_{\text{c}} ?\delta 1 : \text{REAL}$ **and** $t\delta 2: \Gamma \vdash_c \delta *_{\text{c}} ?\delta 2 : \text{REAL}$
using *is-density-exprD*[*OF wfb*] *invar*
by (*auto intro!*: *cet-op*[**where** $t = \text{PRODUCT REAL REAL}$] *cet-pair*)
have *vars1*: *free-vars* $(\delta *_{\text{c}} ?\delta 1) \subseteq \text{set } vs \cup \text{set } vs'$ **and**
vars2: *free-vars* $(\delta *_{\text{c}} ?\delta 2) \subseteq \text{set } vs \cup \text{set } vs'$
using *invar is-density-exprD*[*OF wfb*] **by** (*auto simp*: *shift-var-set-def*)
have *dens1*: *dens-ctxt- α* $(vs, vs', \Gamma, \delta *_{\text{c}} ?\delta 1) \vdash_d e1 \Rightarrow (\lambda x xa. \text{ennreal } (\text{eval-cexpr } g1 \ x \ xa))$ **and**
wf1: *is-density-expr* $(vs, vs', \Gamma, \delta *_{\text{c}} ?\delta 1) \ t \ g1$ **and**
dens2: *dens-ctxt- α* $(vs, vs', \Gamma, \delta *_{\text{c}} ?\delta 2) \vdash_d e2 \Rightarrow (\lambda x xa. \text{ennreal } (\text{eval-cexpr } g2 \ x \ xa))$ **and**
wf2: *is-density-expr* $(vs, vs', \Gamma, \delta *_{\text{c}} ?\delta 2) \ t \ g2$
using *edc-if.IH*(2)[*OF t1 inv1*] *edc-if.IH*(3)[*OF t2 inv2*] *edc-if.prem*s **by**
simp-all

have *f-nonneg*[*simp*]: $\sigma \in \text{space } (\text{state-measure } (\text{set } vs \cup \text{set } vs') \ \Gamma) \Longrightarrow$
 $0 \leq \text{extract-real } (\text{cexpr-sem } (\text{case-nat } (\text{BoolVal } b) \ \sigma) \ f)$ **for** $b \ \sigma$
using *wfb*[*THEN is-density-exprD-nonneg*] **by** (*rule nonneg-cexprD*) *auto*

let $?\delta' = \lambda \sigma. \text{ennreal } (\text{extract-real } (\text{cexpr-sem } \sigma \ \delta))$ **and** $?f = \lambda \sigma \ x. \text{ennreal } (\text{eval-cexpr } f \ \sigma \ x)$
show *?case*
proof (*rule conjI*, *simp only*: *dens-ctxt- α -def prod.case*, *rule hd-cong*[*OF hd-if*])
let $?Y = (\text{set } vs, \text{set } vs', \Gamma, \text{if-dens } ?\delta' \ ?f \ \text{True})$
show $?Y \vdash_d e1 \Rightarrow (\lambda \rho \ x. \text{eval-cexpr } g1 \ \rho \ x)$
proof (*rule hd-dens-ctxt-cong*)
let $?\delta = \lambda \sigma. \text{ennreal } (\text{extract-real } (\text{cexpr-sem } \sigma \ (\delta *_{\text{c}} ?\delta 1)))$
show $(\text{set } vs, \text{set } vs', \Gamma, ?\delta) \vdash_d e1 \Rightarrow (\lambda \rho \ x. \text{ennreal } (\text{eval-cexpr } g1 \ \rho \ x))$
using *dens1* **by** (*simp add*: *dens-ctxt- α -def*)
fix σ **assume** $\sigma: \sigma \in \text{space } (\text{state-measure } (\text{set } vs \cup \text{set } vs') \ \Gamma)$
have $\text{extract-real } (\text{cexpr-sem } \sigma \ (\delta *_{\text{c}} ?\delta 1)) =$
 $\text{extract-real } (\text{cexpr-sem } \sigma \ \delta) * \text{extract-real } (\text{cexpr-sem } \sigma \ ?\delta 1)$
using *invar vars1 is-density-exprD*[*OF wfb*] **by** (*subst cexpr-sem-Mult*[*OF invar*(3) - σ]) *auto*
also **have** $\dots = \text{if-dens } ?\delta' \ ?f \ \text{True} \ \sigma$ **unfolding** *if-dens-def* **by** (*simp add*: *eval-cexpr-def ennreal-mult''* σ)
finally **show** $?f \ \sigma = \text{if-dens } ?\delta' \ ?f \ \text{True} \ \sigma$ **by** (*simp add*: *if-dens-det-def*)
qed

next
let $?Y = (\text{set } vs, \text{set } vs', \Gamma, \text{if-dens } ?\delta' \ ?f \ \text{False})$
show $?Y \vdash_d e2 \Rightarrow (\lambda \rho \ x. \text{eval-cexpr } g2 \ \rho \ x)$
proof (*rule hd-dens-ctxt-cong*)
let $?\delta = \lambda \sigma. \text{ennreal } (\text{extract-real } (\text{cexpr-sem } \sigma \ (\delta *_{\text{c}} ?\delta 2)))$
show $(\text{set } vs, \text{set } vs', \Gamma, ?\delta) \vdash_d e2 \Rightarrow (\lambda \rho \ x. \text{ennreal } (\text{eval-cexpr } g2 \ \rho \ x))$
using *dens2* **by** (*simp add*: *dens-ctxt- α -def*)
fix σ **assume** $\sigma: \sigma \in \text{space } (\text{state-measure } (\text{set } vs \cup \text{set } vs') \ \Gamma)$
have $\text{extract-real } (\text{cexpr-sem } \sigma \ (\delta *_{\text{c}} ?\delta 2)) =$

```

      extract-real (cexpr-sem  $\sigma$   $\delta$ ) * extract-real (cexpr-sem  $\sigma$  ? $\delta$ 2)
    using invar vars1 is-density-exprD[OF wfb] by (subst cexpr-sem-Mult[OF
invar( $\beta$ ) -  $\sigma$ ]) auto
    also have ... = if-dens ? $\delta'$  ?f False  $\sigma$  unfolding if-dens-def by (simp add:
eval-cexpr-def ennreal-mult''  $\sigma$ )
    finally show ? $\delta$   $\sigma$  = if-dens ? $\delta'$  ?f False  $\sigma$  by (simp add: if-dens-det-def)
  qed
next
  fix  $\rho$   $x$  assume  $\rho$ :  $\rho \in \text{space (state-measure (set vs') \Gamma)}$  and  $x$ :  $x \in \text{space}$ 
(stock-measure  $t$ )
  hence eval-cexpr ( $g1 +_c g2$ )  $\rho$   $x$  = eval-cexpr  $g1$   $\rho$   $x$  + eval-cexpr  $g2$   $\rho$   $x$ 
  using wf1 wf2 unfolding eval-cexpr-def is-density-expr-def
  by (subst cexpr-sem-Add[where  $\Gamma = \text{case-nat } t \Gamma$  and  $V = \text{shift-var-set (set}$ 
vs $^{\wedge}$ )] auto
  moreover have  $0 \leq \text{eval-cexpr } g1 \rho x$  and  $0 \leq \text{eval-cexpr } g2 \rho x$ 
  unfolding eval-cexpr-def
  using  $\rho$   $x$  wf1 [THEN is-density-exprD-nonneg, THEN nonneg-cexprD] wf2 [THEN
is-density-exprD-nonneg, THEN nonneg-cexprD]
  unfolding space-state-measure-shift-iff by auto
  ultimately show ennreal (eval-cexpr  $g1$   $\rho$   $x$ ) + ennreal (eval-cexpr  $g2$   $\rho$   $x$ ) =
ennreal (eval-cexpr ( $g1 +_c g2$ )  $\rho$   $x$ )
  by simp
next
  show is-density-expr (vs, vs',  $\Gamma$ ,  $\delta$ )  $t$  ( $g1 +_c g2$ ) using wf1 wf2
  by (auto simp: is-density-expr-def intro!: cet-op[where  $t = \text{PRODUCT REAL}$ 
REAL] cet-pair nonneg-cexpr-Add)
next
  show ({}, set vs  $\cup$  set vs',  $\Gamma$ ,  $\lambda \cdot 1$ )  $\vdash_d b \Rightarrow (\lambda \sigma x. \text{ennreal (eval-cexpr } f \sigma x))$ 
  using densb unfolding dens-ctxt- $\alpha$ -def by (simp add: extract-real-def one-ennreal-def)
  qed (insert edc-if.premis edc-if.hyps, auto intro!: density-context- $\alpha$ )
next
  case (edc-op-discr vs vs'  $\Gamma$   $\delta$   $e$   $f$   $t$  oper  $t'$   $t''$ )
  let ?expr' =  $\langle (\text{oper } \text{\$}\text{\$}_c (\text{CVar } 0)) =_c \text{CVar } 1 \rangle_c *_c \text{map-vars (case-nat } 0 (\lambda x.
x+2)) f$ 
  let ?expr =  $\int_c ?expr' \partial t$  and ?shift = case-nat 0 ( $\lambda x. x + 2$ )
  from edc-op-discr.premis(1) edc-op-discr.hyps
  have  $t: \Gamma \vdash e : t$  by (elim expr-typing-opE, fastforce split: pdf-type.split-asm)
  with edc-op-discr.premis(1) and edc-op-discr.hyps have [simp]:  $t'' = t'$ 
  by (intro expr-typing-unique) (auto intro: et-op)
  from  $t$  and edc-op-discr.premis(1)
  have the-t1: the (expr-type  $\Gamma$   $e$ ) =  $t$  and the-t2: the (expr-type  $\Gamma$  (oper  $\text{\$}\text{\$}$   $e$ ))
=  $t'$ 
  by (simp-all add: expr-type-Some-iff[symmetric])

  from edc-op-discr.premis edc-op-discr.IH[OF  $t$ ]
  have dens: dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )  $\vdash_d e \Rightarrow (\lambda x xa. \text{ennreal (eval-cexpr } f x
xa))$  and
  wf: is-density-expr (vs, vs',  $\Gamma$ ,  $\delta$ )  $t$   $f$  by simp-all

```

note $wf' = is-density-exprD[OF wf]$
have $ctype'''$: $case-nat\ t\ (case-nat\ t'\ \Gamma) \vdash_c\ (oper\ \$$_c\ (CVar\ 0)) =_c\ CVar\ 1 :$
BOOL **and**
 $ctype''$: $case-nat\ t\ (case-nat\ t'\ \Gamma) \vdash_c\ \langle (oper\ \$$_c\ (CVar\ 0)) =_c\ CVar\ 1 \rangle_c :$
REAL **and**
 $ctype'$: $case-nat\ t\ (case-nat\ t'\ \Gamma) \vdash_c\ ?expr' : REAL$ **using** wf' *edc-op-discr.hyps*
by $((intro\ cet-op-intros\ cexpr-typing-map-vars\ cet-var'\ cet-pair\ cet-eq,$
 $auto\ intro! : cet-op\ cet-var') [])+$
from $ctype'$ **have** $ctype$: $case-nat\ t'\ \Gamma \vdash_c\ ?expr : REAL$ **by** $(rule\ cet-int)$
have $vars'$: $free-vars\ ?expr' \subseteq shift-var-set\ (shift-var-set\ (set\ vs'))$ **using** wf'
by $(auto\ split : nat.split\ simp : shift-var-set-def)$
hence $vars$: $free-vars\ ?expr \subseteq shift-var-set\ (set\ vs')$ **by** $(auto\ split : nat.split-asm)$

let $?Y = (set\ vs,\ set\ vs',\ \Gamma,\ \lambda\ \varrho.\ ennreal\ (extract-real\ (cexpr-sem\ \varrho\ \delta)))$
let $?M = \lambda\ \varrho.\ dens-ctxt-measure\ ?Y\ \varrho \ggg (\lambda\ \sigma.\ expr-sem\ \sigma\ e)$
have $nonneg-cexpr$ $(shift-var-set\ (set\ vs'))\ (case-nat\ t\ \Gamma)\ f$
using $wf[THEN\ is-density-exprD-nonneg]$.
hence $nonneg$: $nonneg-cexpr\ (shift-var-set\ (shift-var-set\ (set\ vs')))$
 $(case-nat\ t\ (case-nat\ t'\ \Gamma))\ ?expr'$
using $wf'\ vars'\ ctype'''$ **by** $(intro\ nonneg-cexpr-Mult[OF\ ctype'']\ cexpr-typing-map-vars$
 $nonneg-cexpr-map-vars\ nonneg-indicator)$
 $(auto\ dest : nonneg-cexprD\ simp : extract-real-def$
bool-to-real-def)

let $?M = \lambda\ \varrho.\ dens-ctxt-measure\ ?Y\ \varrho \ggg (\lambda\ \sigma.\ expr-sem\ \sigma\ (oper\ \$$\ e))$
let $?f = \lambda\ \varrho\ x\ y.\ (if\ op-sem\ oper\ y = x\ then\ 1\ else\ 0) * ennreal\ (eval-cexpr\ f\ \varrho\ y)$
have $?Y \vdash_d\ oper\ \$$\ e \Rightarrow (\lambda\ \varrho\ x.\ \int^+ y.\ ?f\ \varrho\ x\ y\ \partial stock-measure\ t)$ **using** $dens\ t$
edc-op-discr.hyps
by $(subst\ the-t1[symmetric],\ intro\ hd-op-discr)$
 $(simp-all\ add : dens-ctxt-\alpha-def\ the-t1\ expr-type-Some-iff[symmetric])$
hence $dens$: $?Y \vdash_d\ oper\ \$$\ e \Rightarrow (\lambda\ \varrho\ x.\ \int^+ y.\ eval-cexpr\ ?expr'\ (case-nat\ x\ \varrho)\ y$
 $\partial stock-measure\ t)$
proof $(rule\ hd-cong[OF\ - - -\ nn-integral-cong])$
fix $\varrho\ x\ y$ **let** $?P = \lambda x\ M.\ x \in space\ M$
assume A : $?P\ \varrho\ (state-measure\ (set\ vs')\ \Gamma)\ ?P\ x\ (stock-measure\ t')\ ?P\ y$
 $(stock-measure\ t)$
hence $val-type\ (cexpr-sem\ (case-nat\ y\ \varrho)\ f) = REAL$ **using** wf' **by** $(intro$
 $val-type-cexpr-sem)\ auto$
thus $?f\ \varrho\ x\ y = ennreal\ (eval-cexpr\ ?expr'\ (case-nat\ x\ \varrho)\ y)$
by $(auto\ simp : eval-cexpr-def\ extract-real-def\ lift-RealIntVal2-def$
 $bool-to-real-def\ cexpr-sem-map-vars\ elim! : REAL-E)$
qed $(insert\ edc-op-discr.premis,\ auto\ intro! : density-context-\alpha)$
hence $dens'$: $has-parametrized-subprob-density\ (state-measure\ (set\ vs')\ \Gamma)\ ?M$
 $(stock-measure\ t')$
 $(\lambda\ \varrho\ x.\ \int^+ y.\ eval-cexpr\ ?expr'\ (case-nat\ x\ \varrho)\ y\ \partial stock-measure\ t)$
using $edc-op-discr.premis$ **by** $(intro\ expr-has-density-sound-aux\ density-context-\alpha)$
simp-all

show $?case$

proof (*intro conjI is-density-exprI, simp only: dens-ctxt- α -def prod.case, rule hd-AE[OF dens]*)
fix ϱ **assume** $\varrho: \varrho \in \text{space } (\text{state-measure } (\text{set } vs') \Gamma)$
let $?dens = \lambda x. \int^+ y. \text{eval-cexpr } ?expr' (\text{case-nat } x \varrho) y \partial \text{stock-measure } t$
show $AE \ x \ \text{in } \text{stock-measure } t'. ?dens \ x = \text{ennreal } (\text{eval-cexpr } ?expr \ \varrho \ x)$
proof (*rule AE-mp[OF - AE-I2[OF impI]]*)
from *has-parametrized-subprob-density-integral*[OF $dens' \ \varrho$] **and**
has-parametrized-subprob-densityD(\mathcal{B})[OF $dens'$] **and** ϱ
show $AE \ x \ \text{in } \text{stock-measure } t'. ?dens \ x \neq \infty$ **by** (*intro nn-integral-PInf-AE*)
auto
next
fix x **assume** $x: x \in \text{space } (\text{stock-measure } t')$ **and** *fin*: $?dens \ x \neq \infty$
thus $?dens \ x = \text{ennreal } (\text{eval-cexpr } ?expr \ \varrho \ x)$
using $\varrho \ \text{vars}' \ \text{ctype}' \ \text{ctype}' \ \text{nonneg}$ **unfolding** *eval-cexpr-def*
by (*subst cexpr-sem-integral-nonneg*) (*auto intro!: nonneg-cexpr-map-vars*)
simp: less-top)
qed
next
show *nonneg-cexpr* (*shift-var-set* (*set vs'*)) (*case-nat* $t'' \ \Gamma$) ($\int_c ?expr' \ \partial t$)
using *nonneg* **by** (*intro nonneg-cexpr-int*) *simp*
qed (*insert vars ctype edc-op-discr.prem, auto*)

next
case (*edc-fst vs vs' $\Gamma \ \delta \ e \ f \ t'' \ t' \ t$*)
hence [*simp*]: $t'' = t$ **by** (*auto intro!: expr-typing-unique et-op*)
from *edc-fst.hyps* **have** t' : *the* (*expr-type* Γ (*Snd* $\$ \$ \ e$)) = t'
by (*simp add: expr-type-Some-iff[symmetric]*)
let $?shift = \text{case-nat } 0 \ (\lambda x. x + 2)$
have [*simp*]: $\bigwedge t \ t'. \text{case-nat } t \ (\text{case-nat } t' \ \Gamma) \circ \text{case-nat } 0 \ (\lambda x. \text{Suc } (\text{Suc } x)) = \text{case-nat } t \ \Gamma$
by (*intro ext*) (*simp split: nat.split add: o-def*)
note *invar* = *cdens-ctxt-invarD*[OF *edc-fst.prem*(2)]
have *dens*: *dens-ctxt- α* (vs, vs', Γ, δ) $\vdash_a \ e \Rightarrow (\lambda \varrho \ x. \text{ennreal } (\text{eval-cexpr } f \ \varrho \ x))$
and
 $wf: \text{is-density-expr } (vs, vs', \Gamma, \delta) \ (\text{PRODUCT } t \ t') \ f$ **using** *edc-fst* **by** *auto*
let $?M = \lambda \varrho. \text{dens-ctxt-measure } (\text{set } vs, \text{set } vs', \Gamma, \lambda \varrho. \text{ennreal } (\text{extract-real } (\text{cexpr-sem } \varrho \ \delta))) \ \varrho$
 $\gg (\lambda \sigma. \text{expr-sem } \sigma \ e)$
have *nonneg*: *nonneg-cexpr* (*shift-var-set* (*set vs'*)) (*case-nat* (*PRODUCT* $t \ t'$) Γ) f
using wf **by** (*rule is-density-exprD-nonneg*)

note $wf' = \text{is-density-exprD}$ [OF wf]
let $?expr = \text{map-vars } ?shift \ f \circ_c < \text{CVar } 1, \text{CVar } 0 >_c$
have *ctype*: *case-nat* t' (*case-nat* $t \ \Gamma$) $\vdash_c \ ?expr : \text{REAL}$
using wf' **by** (*auto intro!: cexpr-typing.intros cexpr-typing-map-vars*)
have *vars*: *free-vars* $?expr \subseteq \text{shift-var-set } (\text{shift-var-set } (\text{set } vs'))$ **using** *free-vars-cexpr-comp wf'*
by (*intro subset-shift-var-set*) (*force simp: shift-var-set-def*)

```

let ?M = λρ. dens-ctxt-measure (set vs, set vs', Γ, λρ. ennreal (extract-real
(cexpr-sem ρ δ))) ρ
  ≫ (λσ. expr-sem σ (Fst $$ e))
have A: ∧x y ρ. ((case-nat x (case-nat y ρ))(0 := <|y, x|>)) ∘ ?shift = case-nat
<|y, x|> ρ
  by (intro ext) (simp split: nat.split add: o-def)
have dens': (set vs, set vs', Γ, λρ. ennreal (extract-real (cexpr-sem ρ δ))) ⊢d Fst
$$ e ⇒
  (λρ x. (∫+ y. eval-cexpr f ρ (<|x, y|>) ∂stock-measure t')) (is ?Y
⊢d - ⇒ ?f)
  using dens by (subst t'[symmetric], intro hd-fst) (simp add: dens-ctxt-α-def)
hence dens': ?Y ⊢d Fst $$ e ⇒ (λρ x. (∫+ y. eval-cexpr ?expr (case-nat x ρ) y
∂stock-measure t'))
  (is - ⊢d - ⇒ ?f) by (rule hd-cong, intro density-context-α, insert edc-fst.premis
A)
  (auto intro!: nn-integral-cong simp: eval-cexpr-def cexpr-sem-cexpr-comp
cexpr-sem-map-vars)
hence dens'': has-parametrized-subprob-density (state-measure (set vs') Γ) ?M
(stock-measure t) ?f
  using edc-fst.premis by (intro expr-has-density-sound-aux density-context-α)
simp-all

have ∧V. ?shift -' shift-var-set (shift-var-set V) = shift-var-set V
  by (auto simp: shift-var-set-def split: nat.split-asm)
hence nonneg': nonneg-cexpr (shift-var-set (shift-var-set (set vs'))) (case-nat t'
(case-nat t Γ)) ?expr
  by (auto intro!: nonneg-cexpr-comp nonneg-cexpr-map-vars nonneg cexpr-typing.intros
cet-var')
show ?case
proof (intro conjI is-density-exprI, simp only: dens-ctxt-α-def prod.case, rule
hd-AE[OF dens'])
  fix ρ assume ρ: ρ ∈ space (state-measure (set vs') Γ)
  thus AE x in stock-measure t. ?f ρ x = ennreal (eval-cexpr (∫c ?expr ∂t') ρ
x)
  using ctype vars edc-fst.hyps nonneg'
  by (intro has-parametrized-subprob-density-cexpr-sem-integral[OF dens'']) auto
next
show nonneg-cexpr (shift-var-set (set vs')) (case-nat t Γ)
  (∫c (map-vars (case-nat 0 (λx. x + 2)) f ∘c <CVar 1, CVar 0>c) ∂t')
  using nonneg' by (intro nonneg-cexpr-int)
qed (insert edc-fst.premis ctype vars, auto simp: measurable-split-conv
intro!: cet-int measurable-compose[OF - measurable-ennreal]
measurable-Pair-compose-split[OF measurable-eval-cexpr])

next
case (edc-snd vs vs' Γ δ e f t t')
hence [simp]: t'' = t' by (auto intro!: expr-typing-unique et-op)
from edc-snd.hyps have t': the (expr-type Γ (Fst $$ e)) = t
  by (simp add: expr-type-Some-iff[symmetric])

```

let $?shift = \text{case-nat } 0 (\lambda x. x + 2)$
have $[simp]: \bigwedge t t'. \text{case-nat } t (\text{case-nat } t' \Gamma) \circ \text{case-nat } 0 (\lambda x. \text{Suc } (\text{Suc } x)) = \text{case-nat } t \Gamma$
by $(\text{intro ext}) (\text{simp split: nat.split add: o-def})$
note $\text{invar} = \text{cdens-ctxt-invarD}[OF \text{edc-snd.prem}(2)]$
have $\text{dens}: \text{dens-ctxt-}\alpha (vs, vs', \Gamma, \delta) \vdash_d e \Rightarrow (\lambda \varrho. x. \text{ennreal } (\text{eval-cexpr } f \varrho x))$
and
 $wf: \text{is-density-cexpr } (vs, vs', \Gamma, \delta) (\text{PRODUCT } t t') f$ **using** edc-snd **by** auto
let $?M = \lambda \varrho. \text{dens-ctxt-measure } (\text{set } vs, \text{set } vs', \Gamma, \lambda \varrho. \text{ennreal } (\text{extract-real } (\text{cexpr-sem } \varrho \delta))) \varrho$
 $\gg (\lambda \sigma. \text{cexpr-sem } \sigma e)$
have $\text{nonneg}: \text{nonneg-cexpr } (\text{shift-var-set } (\text{set } vs')) (\text{case-nat } (\text{PRODUCT } t t') \Gamma) f$
using wf **by** $(\text{rule is-density-cexprD-nonneg})$

note $wf' = \text{is-density-cexprD}[OF wf]$
let $?cexpr = \text{map-vars } ?shift f \circ_c < \text{CVar } 0, \text{CVar } 1 >_c$
have $\text{ctype}: \text{case-nat } t (\text{case-nat } t' \Gamma) \vdash_c ?cexpr : \text{REAL}$
using wf' **by** $(\text{auto intro!: cexpr-typing.intros cexpr-typing-map-vars})$
have $\text{vars}: \text{free-vars } ?cexpr \subseteq \text{shift-var-set } (\text{shift-var-set } (\text{set } vs'))$ **using** $\text{free-vars-cexpr-comp } wf'$
by $(\text{intro subset-shift-var-set}) (\text{force simp: shift-var-set-def})$
let $?M = \lambda \varrho. \text{dens-ctxt-measure } (\text{set } vs, \text{set } vs', \Gamma, \lambda \varrho. \text{ennreal } (\text{extract-real } (\text{cexpr-sem } \varrho \delta))) \varrho$
 $\gg (\lambda \sigma. \text{cexpr-sem } \sigma (\text{Snd } \$\$ e))$
have $A: \bigwedge x y \varrho. ((\text{case-nat } y (\text{case-nat } x \varrho)) (0 := <|y, x|>)) \circ ?shift = \text{case-nat } <|y, x|> \varrho$
by $(\text{intro ext}) (\text{simp split: nat.split add: o-def})$
have $\text{dens}': (\text{set } vs, \text{set } vs', \Gamma, \lambda \varrho. \text{ennreal } (\text{extract-real } (\text{cexpr-sem } \varrho \delta))) \vdash_d \text{Snd } \$\$ e \Rightarrow$
 $(\lambda \varrho y. (\int^{+x}. \text{eval-cexpr } f \varrho (<|x, y|>) \partial \text{stock-measure } t)) (\text{is } ?\mathcal{Y} \vdash_d$
 $- \Rightarrow ?f)$
using dens **by** $(\text{subst } t'[\text{symmetric}], \text{intro hd-snd}) (\text{simp add: dens-ctxt-}\alpha\text{-def})$
hence $\text{dens}': ?\mathcal{Y} \vdash_d \text{Snd } \$\$ e \Rightarrow (\lambda \varrho y. (\int^{+x}. \text{eval-cexpr } ?cexpr (\text{case-nat } y \varrho) x \partial \text{stock-measure } t))$
 $(\text{is } - \vdash_d - \Rightarrow ?f)$ **by** $(\text{rule hd-cong, intro density-context-}\alpha, \text{insert edc-snd.prem } A)$
 $(\text{auto intro!: nn-integral-cong simp: eval-cexpr-def cexpr-sem-cexpr-comp cexpr-sem-map-vars})$
hence $\text{dens}'': \text{has-parametrized-subprob-density } (\text{state-measure } (\text{set } vs') \Gamma) ?M (\text{stock-measure } t') ?f$
using edc-snd.prem **by** $(\text{intro expr-has-density-sound-aux density-context-}\alpha) \text{simp-all}$

have $\bigwedge V. ?shift -' \text{shift-var-set } (\text{shift-var-set } V) = \text{shift-var-set } V$
by $(\text{auto simp: shift-var-set-def split: nat.split-asm})$
hence $\text{nonneg}': \text{nonneg-cexpr } (\text{shift-var-set } (\text{shift-var-set } (\text{set } vs')) (\text{case-nat } t (\text{case-nat } t' \Gamma)) ?cexpr$
by $(\text{auto intro!: nonneg-cexpr-comp nonneg-cexpr-map-vars nonneg cexpr-typing.intros})$

```

cet-var')
show ?case
proof (intro conjI is-density-exprI , simp only: dens-ctxt- $\alpha$ -def prod.case, rule
hd-AE[OF dens'])
  fix  $\varrho$  assume  $\varrho$ :  $\varrho \in \text{space } (\text{state-measure } (\text{set } vs') \Gamma)$ 
  thus AE  $x$  in stock-measure  $t'$ . ?f  $\varrho$   $x = \text{ennreal } (\text{eval-cexpr } (\int_c ?\text{expr } \partial t) \varrho$ 
 $x)$ 
    using ctype vars edc-snd.hyps nonneg'
  by (intro has-parametrized-subprob-density-cexpr-sem-integral[OF dens']) auto
next
show nonneg-cexpr (shift-var-set (set vs')) (case-nat  $t'' \Gamma$ ) ( $\int_c ?\text{expr } \partial t$ )
  using nonneg' by (intro nonneg-cexpr-int) simp
qed (insert edc-snd.premis ctype vars, auto simp: measurable-split-conv
intro!: cet-int measurable-compose[OF - measurable-ennreal]
measurable-Pair-compose-split[OF measurable-eval-cexpr])

next
case (edc-neg vs vs'  $\Gamma$   $\delta$   $e$   $f$   $t$ )
from edc-neg.premis(1) have  $t$ :  $\Gamma \vdash e : t$  by (cases  $t$ ) (auto split: pdf-type.split-asm)
from edc-neg.premis(1) have  $t$ -disj:  $t = \text{REAL} \vee t = \text{INTEG}$ 
  by (cases  $t$ ) (auto split: pdf-type.split-asm)
from edc-neg.premis edc-neg.IH[OF  $t$ ]
  have dens: dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )  $\vdash_d e \Rightarrow (\lambda x xa. \text{ennreal } (\text{eval-cexpr } f x$ 
 $xa))$  and
  wf: is-density-expr (vs, vs',  $\Gamma$ ,  $\delta$ )  $t$   $f$  by simp-all

  have dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )  $\vdash_d \text{Minus } \$\$ e \Rightarrow (\lambda \sigma x. \text{ennreal } (\text{eval-cexpr } f \sigma$ 
 $(\text{op-sem } \text{Minus } x)))$ 
    using dens by (simp only: dens-ctxt- $\alpha$ -def prod.case, intro hd-neg) simp-all
  also have  $(\lambda \sigma x. \text{ennreal } (\text{eval-cexpr } f \sigma (\text{op-sem } \text{Minus } x))) =$ 
     $(\lambda \sigma x. \text{ennreal } (\text{eval-cexpr } (f \circ_c -_c \text{CVar } 0) \sigma x))$ 
    by (intro ext) (auto simp: eval-cexpr-comp)
  finally have dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )  $\vdash_d \text{Minus } \$\$ e \Rightarrow$ 
     $(\lambda \sigma x. \text{ennreal } (\text{eval-cexpr } (f \circ_c -_c \text{CVar } 0) \sigma x))$  .
  moreover have is-density-expr (vs, vs',  $\Gamma$ ,  $\delta$ )  $t$   $(f \circ_c -_c \text{CVar } 0)$ 
proof (intro is-density-exprI)
  from  $t$ -disj have  $t$ -minus: case-nat  $t \Gamma \vdash_c -_c \text{CVar } 0 : t$ 
  by (intro cet-op[where  $t = t$ ]) (auto simp: cexpr-type-Some-iff[symmetric])
  thus case-nat  $t \Gamma \vdash_c f \circ_c -_c \text{CVar } 0 : \text{REAL}$  using is-density-exprD(1)[OF
wf]
  by (intro cexpr-typing-cexpr-comp[of - - -  $t$ ])
  show free-vars  $(f \circ_c -_c \text{CVar } 0) \subseteq \text{shift-var-set } (\text{set } vs')$  using is-density-exprD(2)[OF
wf]
  by (intro order.trans[OF free-vars-cexpr-comp]) (auto simp: shift-var-set-def)
  show nonneg-cexpr (shift-var-set (set vs')) (case-nat  $t \Gamma$ )  $(f \circ_c -_c \text{CVar } 0)$ 
  using wf[THEN is-density-exprD-nonneg]  $t$ -disj
  by (intro nonneg-cexpr-comp)
  (auto intro!: cet-var' cet-minus-real cet-minus-int)
qed

```

ultimately show *?case* by (rule *conjI*)

next

case (*edc-addc vs vs' Γ δ e f e' t*)

let *?expr = f ◦_c (λ_cx. x -_c map-vars Suc (expr-rf-to-cexpr e'))*

from *edc-addc.premis(1)*

have *t1: Γ ⊢ e : t and t2: Γ ⊢ e' : t and t3: op-type Add (PRODUCT t t) = Some t*

by (*elim expr-typing-opE expr-typing-pairE, fastforce split: pdf-type.split-asm*)+

from *edc-addc.premis(1)* have *t-disj: t = REAL ∨ t = INTEG*

by (*cases t*) (*auto split: pdf-type.split-asm*)

hence *t3': op-type Minus t = Some t* by *auto*

from *edc-addc.premis edc-addc.IH[OF t1]*

have *dens: dens-ctxt-α (vs, vs', Γ, δ) ⊢_d e ⇒ (λx xa. ennreal (eval-cexpr f x xa)) and*

wf: is-density-expr (vs, vs', Γ, δ) t f by *simp-all*

hence *ctype: case-nat t Γ ⊢_c ?expr : REAL* using *t1 t2 t3 t3' edc-addc.hyps edc-addc.premis*

by (*intro cexpr-typing-cexpr-comp cet-op[where t = PRODUCT t t] cet-var'*)

(*auto intro!: cet-pair cexpr-typing-map-vars cet-var' cet-op dest: is-density-exprD simp: o-def*)

have *vars: free-vars ?expr ⊆ shift-var-set (set vs')* using *edc-addc.premis edc-addc.hyps*

using *free-vars-expr-rf-to-cexpr is-density-exprD[OF wf]*

by (*intro order.trans[OF free-vars-cexpr-comp subset-shift-var-set]*) *auto*

have *cet-e': Γ ⊢ e' : t*

using *edc-addc.premis(1)*

apply (*cases*)

apply (*erule expr-typing.cases*)

apply (*auto split: pdf-type.splits*)

done

have *dens-ctxt-α (vs, vs', Γ, δ) ⊢_d Add \$\$ <e, e'> ⇒*

(*λσ x. ennreal (eval-cexpr f σ (op-sem Add <|x, expr-sem-rf σ (Minus \$\$ e')|>)))*)

(*is ?Y ⊢_d - ⇒ ?f*) using *dens edc-addc.hyps*

by (*simp only: dens-ctxt-α-def prod.case, intro hd-addc*) *simp-all*

also have *?f = (λσ x. ennreal (eval-cexpr ?expr σ x))* using *edc-addc.hyps*

by (*intro ext*) (*auto simp: eval-cexpr-comp cexpr-sem-map-vars o-def cexpr-sem-expr-rf-to-cexpr*)

finally have *dens-ctxt-α (vs, vs', Γ, δ) ⊢_d Add \$\$ <e, e'> ⇒*

(*λσ x. ennreal (eval-cexpr ?expr σ x)*) .

moreover have *is-density-expr (vs, vs', Γ, δ) t ?expr* using *ctype vars*

proof (*intro is-density-exprI*)

show *nonneg-cexpr (shift-var-set (set vs')) (case-nat t Γ) ?expr*

using *t-disj edc-addc.hyps edc-addc.premis cet-e' free-vars-expr-rf-to-cexpr[of e']*

by (*intro nonneg-cexpr-comp[OF wf[THEN is-density-exprD-nonneg]]*)

(*auto intro!: cet-add-int cet-add-real cet-minus-int cet-minus-real cet-var' cexpr-typing-map-vars*)

```

      simp: o-def)
qed auto
ultimately show ?case by (rule conjI)

next
case (edc-multc vs vs'  $\Gamma$   $\delta$   $e$   $f$   $c$   $t$ )
let ?expr = (f  $\circ_c$  ( $\lambda_c x. x *_c CReal$  (inverse c))) *_c CReal (inverse (abs c))
from edc-multc.premis(1) edc-multc.hyps have t1:  $\Gamma \vdash e : REAL$  and [simp]: t
= REAL
by (elim expr-typing-opE expr-typing-pairE, force split: pdf-type.split-asm)+
from edc-multc.premis edc-multc.IH[OF t1]
have dens: dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )  $\vdash_d e \Rightarrow (\lambda x xa. ennreal (eval-cexpr f x
xa))$  and
wf: is-density-cexpr (vs, vs',  $\Gamma$ ,  $\delta$ ) REAL f by simp-all
have ctype': case-nat t  $\Gamma \vdash_c f \circ_c (\lambda_c x. x *_c CReal$  (inverse c)) : REAL
using t1 edc-multc.hyps edc-multc.premis is-density-cexprD[OF wf]
by (intro cexpr-typing-cexpr-comp)
(auto intro!: cet-pair cexpr-typing-map-vars cet-var' cet-val' cet-op-intros)
hence ctype: case-nat t  $\Gamma \vdash_c ?expr : REAL$ 
by (auto intro!: cet-op-intros cet-pair cet-val')
have vars': free-vars (f  $\circ_c$  ( $\lambda_c x. x *_c CReal$  (inverse c)))  $\subseteq$  shift-var-set (set vs')
using edc-multc.premis edc-multc.hyps free-vars-cexpr-rf-to-cexpr is-density-cexprD[OF
wf]
by (intro order.trans[OF free-vars-cexpr-comp subset-shift-var-set]) auto
hence vars: free-vars ?expr  $\subseteq$  shift-var-set (set vs') by simp

have dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )  $\vdash_d Mult$  $$  $\langle e, Val (RealVal c) \rangle \Rightarrow$ 
( $\lambda \sigma x. ennreal (eval-cexpr f \sigma (op-sem Mult <|x, op-sem Inverse (RealVal
c)|>))$  *
ennreal (inverse (abs (extract-real (RealVal c))))))
(is ? $\mathcal{Y} \vdash_d - \Rightarrow ?f$ ) using dens edc-multc.hyps
by (simp only: dens-ctxt- $\alpha$ -def prod.case, intro hd-multc) simp-all
hence dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )  $\vdash_d Mult$  $$  $\langle e, Val (RealVal c) \rangle \Rightarrow$ 
( $\lambda \sigma x. ennreal (eval-cexpr ?expr \sigma x)$ )
proof (simp only: dens-ctxt- $\alpha$ -def prod.case, erule-tac hd-cong)
fix  $\rho$  x assume  $\rho: \rho \in space$  (state-measure (set vs')  $\Gamma$ ) and  $x: x \in space$ 
(stock-measure REAL)
hence eval-cexpr ?expr  $\rho$  x =
extract-real (cexpr-sem (case-nat x  $\rho$ ) (f  $\circ_c CVar$  0 *_c CReal (inverse
c))) * inverse |c|
(is - = ?a * ?b) unfolding eval-cexpr-def
by (subst cexpr-sem-Mult[OF ctype' cet-val' - vars'])
(auto simp: extract-real-def simp del: stock-measure.simps)
also hence ?a = eval-cexpr f  $\rho$  (op-sem Mult <|x, op-sem Inverse (RealVal
c)|>)
by (auto simp: cexpr-sem-cexpr-comp eval-cexpr-def lift-RealVal-def lift-RealIntVal2-def)
finally show ennreal (eval-cexpr f  $\rho$  (op-sem Mult <|x, op-sem Inverse (RealVal
c)|>)) *
ennreal (inverse |extract-real (RealVal c)|) = ennreal (eval-cexpr

```

```

?expr ρ x)
  by (simp add: extract-real-def ennreal-mult')
qed (insert edc-multc.premis, auto intro!: density-context-α)
moreover have is-density-expr (vs, vs', Γ, δ) t ?expr using ctype vars
proof (intro is-density-exprI)
  show nonneg-cexpr (shift-var-set (set vs')) (case-nat t Γ) ?expr
  using is-density-exprD[OF wf] vars vars'
  by (intro nonneg-cexpr-comp[OF wf[THEN is-density-exprD-nonneg]] non-
neg-cexpr-Mult ctype')
  (auto intro!: nonneg-cexprI cet-var' cet-val' cet-op-intros)
qed auto
ultimately show ?case by (rule conjI)

next
case (edc-add vs vs' Γ δ e f t t')
note t = ⟨Γ ⊢ e : PRODUCT t t⟩
note invar = cdens-ctxt-invarD[OF edc-add.premis(2)]
from edc-add.premis and t have op-type Add (PRODUCT t t) = Some t'
  by (elim expr-typing-opE) (auto dest: expr-typing-unique)
hence [simp]: t' = t and t-disj: t = INTEG ∨ t = REAL by (auto split:
pdf-type.split-asm)

  have dens: dens-ctxt-α (vs, vs', Γ, δ) ⊢d e ⇒ (λx xa. ennreal (eval-cexpr f x xa))
and
  wf: is-density-expr (vs, vs', Γ, δ) (PRODUCT t t) f
  using edc-add by simp-all
note wf' = is-density-exprD[OF wf]
let ?Y = (set vs, set vs', Γ, λρ. ennreal (extract-real (cexpr-sem ρ δ)))
let ?M = λρ. dens-ctxt-measure ?Y ρ ≫ (λσ. expr-sem σ e)
have nonneg: nonneg-cexpr (shift-var-set (set vs')) (case-nat (PRODUCT t t) Γ)
f
  using wf by (rule is-density-exprD-nonneg)

let ?shift = case-nat 0 (λx. x + 2)
let ?expr' = map-vars ?shift f ◦c (λcx. <x, CVar 1 -c x>c)
let ?expr = ∫c ?expr' ∂t
have [simp]: ∧ t t' Γ. case-nat t (case-nat t' Γ) ◦ case-nat 0 (λx. Suc (Suc x)) =
case-nat t Γ
  by (intro ext) (simp split: nat.split add: o-def)
have ctype'': case-nat t (case-nat t Γ) ⊢c <CVar 0, CVar 1 -c CVar 0>c :
PRODUCT t t
  by (rule cet-pair, simp add: cet-var', rule cet-op[where t = PRODUCT t t],
rule cet-pair)
  (insert t-disj, auto intro!: cet-var' cet-op[where t = t])
hence ctype': case-nat t (case-nat t Γ) ⊢c ?expr' : REAL using wf'
  by (intro cexpr-typing-cexpr-comp cexpr-typing-map-vars) simp-all
hence ctype: case-nat t Γ ⊢c ?expr : REAL by (rule cet-int)
have vars': free-vars ?expr' ⊆ shift-var-set (shift-var-set (set vs')) using wf'
  by (intro order.trans[OF free-vars-cexpr-comp]) (auto split: nat.split simp:

```

shift-var-set-def
hence *vars*: *free-vars* *?expr* \subseteq *shift-var-set* (*set vs'*) **by** *auto*

let *?M* = $\lambda \varrho. \text{dens-ctxt-measure } ?\mathcal{Y} \varrho \gg= (\lambda \sigma. \text{expr-sem } \sigma (\text{Add } \$\$ e))$
let *?f* = $\lambda \varrho x y. \text{eval-cexpr } f \varrho <|y, \text{op-sem Add } <|x, \text{op-sem Minus } y|>|>$
have $?\mathcal{Y} \vdash_d \text{Add } \$\$ e \Rightarrow (\lambda \varrho x. \int^+ y. ?f \varrho x y \partial \text{stock-measure } (\text{val-type } x))$ **using**
dens
by (*intro hd-add*) (*simp add: dens-ctxt- α -def*)
hence *dens*: $?\mathcal{Y} \vdash_d \text{Add } \$\$ e \Rightarrow (\lambda \varrho x. \int^+ y. \text{eval-cexpr } ?\text{expr}' (\text{case-nat } x \varrho) y \partial \text{stock-measure } t)$
by (*rule hd-cong*) (*insert edc-add.premis, auto intro!: density-context- α nn-integral-cong*
simp: eval-cexpr-def cexpr-sem-cexpr-comp
cexpr-sem-map-vars)
hence *dens'*: *has-parametrized-subprob-density* (*state-measure* (*set vs'*) Γ) *?M*
(*stock-measure t*)
 $(\lambda \varrho x. \int^+ y. \text{eval-cexpr } ?\text{expr}' (\text{case-nat } x \varrho) y \partial \text{stock-measure } t)$
using *edc-add.premis* **by** (*intro expr-has-density-sound-aux density-context- α*)
simp-all

show *?case*
proof (*intro conjI is-density-exprI, simp only: dens-ctxt- α -def prod.case, rule*
hd-AE[OF dens])
fix ϱ **assume** $\varrho: \varrho \in \text{space } (\text{state-measure } (\text{set } vs') \Gamma)$
let *?dens* = $\lambda x. \int^+ y. \text{eval-cexpr } ?\text{expr}' (\text{case-nat } x \varrho) y \partial \text{stock-measure } t$
show *AE x in stock-measure t. ?dens x = ennreal (eval-cexpr ?expr ϱ x)*
proof (*rule AE-mp[OF - AE-I2[OF impI]]*)
from *has-parametrized-subprob-density-integral[OF dens' ϱ] and*
has-parametrized-subprob-densityD(3)[OF dens'] and ϱ
show *AE x in stock-measure t. ?dens x \neq ∞* **by** (*intro nn-integral-PInf-AE*)
auto

next
fix *x* **assume** $x: x \in \text{space } (\text{stock-measure } t)$ **and** *fin*: *?dens x \neq ∞*
thus *?dens x = ennreal (eval-cexpr ?expr ϱ x)*
using ϱ *vars' ctype' ctype'' nonneg unfolding eval-cexpr-def*
by (*subst cexpr-sem-integral-nonneg*) (*auto intro!: nonneg-cexpr-comp non-*
neg-cexpr-map-vars simp: less-top)

qed
next
show *nonneg-cexpr (shift-var-set (set vs')) (case-nat t' Γ) ?expr*
using *ctype'' nonneg*
by (*intro nonneg-cexpr-int nonneg-cexpr-comp[of - PRODUCT t t] non-*
neg-cexpr-map-vars)
auto

qed (*insert vars ctype edc-add.premis, auto*)

next
case (*edc-inv vs vs' Γ δ e f t*)
hence $t: \Gamma \vdash e : t$ **and** [*simp*]: *t = REAL*
by (*elim expr-typing-opE, force split: pdf-type.split-asm*)+

```

note invar = cdens-ctxt-invarD[OF edc-inv.prems(2)]
let ?expr = (f  $\circ_c$  ( $\lambda_c x.$  inversec x)) *c ( $\lambda_c x.$  (inversec x)  $\hat{c}$  CInt 2)

have dens: dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )  $\vdash_d$  e  $\Rightarrow$  ( $\lambda \varrho x.$  ennreal (eval-cexpr f  $\varrho$  x))
and
  wf: is-density-cexpr (vs, vs',  $\Gamma$ ,  $\delta$ ) REAL f using edc-inv t by simp-all
note wf' = is-density-cexprD[OF wf]
from wf' have ctype: case-nat REAL  $\Gamma \vdash_c$  ?expr : REAL
  by (auto intro!: cet-op-intros cexpr-typing-cexpr-comp cet-var' cet-val')
from wf' have vars': free-vars (f  $\circ_c$  ( $\lambda_c x.$  inversec x))  $\subseteq$  shift-var-set (set vs')
  by (intro order.trans[OF free-vars-cexpr-comp]) auto
hence vars: free-vars ?expr  $\subseteq$  shift-var-set (set vs') using free-vars-cexpr-comp
by simp

show ?case
proof (intro conjI is-density-cexprI, simp only: dens-ctxt- $\alpha$ -def prod.case, rule
hd-cong[OF hd-inv])
  fix  $\varrho$  x assume  $\varrho$ :  $\varrho \in$  space (state-measure (set vs')  $\Gamma$ )
    and x: x  $\in$  space (stock-measure REAL)
from x obtain x' where [simp]: x = RealVal x' by (auto simp: val-type-eq-REAL)
from  $\varrho$  and wf' have val-type (cexpr-sem (case-nat (RealVal (inverse x'))  $\varrho$ )
f) = REAL
  by (intro val-type-cexpr-sem[OF - - case-nat-in-state-measure ])
    (auto simp: type-universe-def simp del: type-universe-type)
thus ennreal (eval-cexpr f  $\varrho$  (op-sem Inverse x)) * ennreal ((inverse (extract-real
x))2) =
  ennreal (eval-cexpr ?expr  $\varrho$  x)
  by (auto simp: eval-cexpr-def lift-RealVal-def lift-RealIntVal2-def ennreal-mult''
extract-real-def cexpr-sem-cexpr-comp elim!: REAL-E)

next
have nonneg-cexpr (shift-var-set (set vs')) (case-nat REAL  $\Gamma$ ) (inversec (CVar
0))  $\hat{c}$  CInt 2)
by (auto intro!: nonneg-cexprI simp: space-state-measure-shift-iff val-type-eq-REAL
lift-RealVal-eq)
then show nonneg-cexpr (shift-var-set (set vs')) (case-nat t  $\Gamma$ ) ?expr
using wf'
by (intro nonneg-cexpr-Mult nonneg-cexpr-comp vars')
  (auto intro!: cet-op-intros cexpr-typing-cexpr-comp cet-var' cet-val')
qed (insert edc-inv.prems ctype vars dens,
auto intro!: density-context- $\alpha$  simp: dens-ctxt- $\alpha$ -def)

next
case (edc-exp vs vs'  $\Gamma$   $\delta$  e f t)
hence t:  $\Gamma \vdash e$  : t and [simp]: t = REAL
by (elim cexpr-typing-opE, force split: pdf-type.split-asm)+
note invar = cdens-ctxt-invarD[OF edc-exp.prems(2)]
let ?expr = ( $\lambda_c x.$  IFc CReal 0 <c x THEN (f  $\circ_c$  lnc x) *c inversec x ELSE
CReal 0)

```

```

have dens: dens-ctxt- $\alpha$  (vs, vs',  $\Gamma$ ,  $\delta$ )  $\vdash_d e \Rightarrow (\lambda \varrho x. \text{ennreal (eval-cexpr f } \varrho x))$ 
and
  wf: is-density-cexpr (vs, vs',  $\Gamma$ ,  $\delta$ ) REAL f using edc-exp t by simp-all
note wf' = is-density-cexprD[OF wf]
from wf' have ctype: case-nat REAL  $\Gamma \vdash_c ?\text{expr} : \text{REAL}$ 
  by (auto intro!: cet-if cet-op-intros cet-var' cet-val' cexpr-typing-cexpr-comp)
from wf' have free-vars (f  $\circ_c (\lambda_c x. \text{ln}_c x)$ )  $\subseteq$  shift-var-set (set vs')
  by (intro order.trans[OF free-vars-cexpr-comp]) auto
hence vars: free-vars ?expr  $\subseteq$  shift-var-set (set vs') using free-vars-cexpr-comp
by simp

show ?case
proof (intro conjI is-density-cexprI, simp only: dens-ctxt- $\alpha$ -def prod.case, rule
hd-cong[OF hd-exp])
  fix  $\varrho x$  assume  $\varrho$ :  $\varrho \in \text{space (state-measure (set vs') } \Gamma)$ 
    and  $x$ :  $x \in \text{space (stock-measure REAL)}$ 
from  $x$  obtain  $x'$  where [simp]:  $x = \text{RealVal } x'$  by (auto simp: val-type-eq-REAL)
from  $\varrho$  and wf' have val-type (cexpr-sem (case-nat (RealVal (ln  $x'$ ))  $\varrho$ ) f) =
REAL
  by (intro val-type-cexpr-sem[OF - - case-nat-in-state-measure ])
    (auto simp: type-universe-def simp del: type-universe-type)
thus (if 0 < extract-real x then ennreal (eval-cexpr f  $\varrho$  (lift-RealVal safe-ln x))
*
  ennreal (inverse (extract-real x)) else 0) = ennreal (eval-cexpr ?expr  $\varrho$ 
x)
  by (auto simp: eval-cexpr-def lift-RealVal-def lift-RealIntVal2-def lift-Comp-def
ennreal-mult''
  extract-real-def cexpr-sem-cexpr-comp elim!: REAL-E)

next
show nonneg-cexpr (shift-var-set (set vs')) (case-nat t  $\Gamma$ ) ?expr
proof (rule nonneg-cexprI-shift)
  fix  $x \sigma$  assume  $x \in \text{type-universe t}$  and  $\sigma$ :  $\sigma \in \text{space (state-measure (set vs') } \Gamma)$ 
then obtain  $r$  where  $x = \text{RealVal } r$ 
  by (auto simp: val-type-eq-REAL)
moreover note  $\sigma$  nonneg-cexprD[OF is-density-cexprD-nonneg[OF wf], of
case-nat (RealVal (ln  $r$ ))  $\sigma$ ]
moreover have val-type (cexpr-sem (case-nat (RealVal (ln  $r$ ))  $\sigma$ ) f) = REAL
using  $\sigma$  by (intro val-type-cexpr-sem[OF wf'(1,2)] case-nat-in-state-measure)
auto
ultimately show 0  $\leq$  extract-real
  (cexpr-sem (case-nat  $x \sigma$ )
  (IF $_c$  CReal 0 < $_c$  CVar 0 THEN (f  $\circ_c \text{ln}_c$  (CVar 0)) / $_c$  CVar 0
ELSE CReal 0))
  by (auto simp: lift-Comp-def lift-RealVal-eq cexpr-sem-cexpr-comp val-type-eq-REAL
case-nat-in-state-measure lift-RealIntVal2-def)

qed
qed (insert edc-exp.premis ctype vars dens,
auto intro!: density-context- $\alpha$  simp: dens-ctxt- $\alpha$ -def)

```

qed

lemma *expr-has-density-cexpr-sound*:

assumes (\square , \square , Γ , *CReal 1*) $\vdash_c e \Rightarrow f \Gamma \vdash e : t$ *free-vars e = {}*

shows *has-subprob-density (expr-sem σ e) (stock-measure t) (λx . ennreal (eval-cexpr f σ x))*

$\forall x \in \text{type-universe } t. 0 \leq \text{extract-real (cexpr-sem (case-nat x } \sigma) f)$

$\Gamma' 0 = t \Longrightarrow \Gamma' \vdash_c f : \text{REAL}$

free-vars f $\subseteq \{0\}$

proof –

have *dens-ctxt- α (\square , \square , Γ , *CReal 1*) $\vdash_d e \Rightarrow (\lambda \varrho x$. ennreal (eval-cexpr f ϱ x)) \wedge is-density-expr (\square , \square , Γ , *CReal 1*) t f using *assms**

by (*intro expr-has-density-cexpr-sound-aux assms cdens-ctxt-invarI nonneg-cexprI subprob-cexprI*)

(*auto simp: state-measure-def PiM-empty cexpr-type-Some-iff[symmetric] extract-real-def*)

hence *dens: dens-ctxt- α (\square , \square , Γ , *CReal 1*) $\vdash_d e \Rightarrow (\lambda \varrho x$. ennreal (eval-cexpr f ϱ x))*

and *wf: is-density-expr (\square , \square , Γ , *CReal 1*) t f using *assms by blast+**

have *has-subprob-density (expr-sem σ e) (stock-measure t)*

(λx . ennreal (eval-cexpr f (λ -. undefined) x)) (**is** ?P) **using** *dens assms*

by (*intro expr-has-density-sound*) (*auto simp: dens-ctxt- α -def extract-real-def one-ennreal-def*)

also have $\bigwedge x$. *cexpr-sem (case-nat x (λ -. undefined)) f = cexpr-sem (case-nat x σ) f*

using *is-density-exprD[OF wf]*

by (*intro cexpr-sem-eq-on-vars*) (*auto split: nat.split simp: shift-var-set-def*)

hence ?P \longleftrightarrow *has-subprob-density (expr-sem σ e) (stock-measure t)*
(λx . ennreal (eval-cexpr f σ x))

by (*intro has-subprob-density-cong*) (*simp add: eval-cexpr-def*)

finally show

from *is-density-exprD[OF wf]* **show** *vars: free-vars f $\subseteq \{0\}$ by (auto simp: shift-var-set-def)*

show $\forall x \in \text{type-universe } t. 0 \leq \text{extract-real (cexpr-sem (case-nat x } \sigma) f)$

proof

fix *v* **assume** *v: v \in type-universe t*

then have $0 \leq \text{extract-real (cexpr-sem (case-nat v (λ -. undefined)) f)$

by (*intro nonneg-cexprD[OF wf[THEN is-density-exprD-nonneg]] case-nat-in-state-measure*)
(*auto simp: space-state-measure*)

also have *cexpr-sem (case-nat v (λ -. undefined)) f = cexpr-sem (case-nat v σ)*

f

using $\langle \text{free-vars } f \subseteq \{0\} \rangle$ **by** (*intro cexpr-sem-eq-on-vars*) *auto*

finally show $0 \leq \text{extract-real (cexpr-sem (case-nat v } \sigma) f)$.

qed

assume $\Gamma' \ 0 = t$
thus $\Gamma' \vdash_c f : REAL$
by (*intro cexpr-typing-cong*'[*OF is-density-exprD*(1)[*OF wf*]])
 (*insert vars, auto split: nat.split*)
qed

inductive *expr-compiles-to* :: *expr* \Rightarrow *pdf-type* \Rightarrow *cexpr* \Rightarrow *bool* ($\langle - : - \Rightarrow_c - \rangle$
 [*10,0,10*] *10*)
for *e t f* **where**
 ($\lambda-. UNIT$) $\vdash e : t \Rightarrow free-vars\ e = \{\}$ \Rightarrow
 ($\langle \rangle, \langle \rangle, \lambda-. UNIT, CReal\ 1$) $\vdash_c e \Rightarrow f \Rightarrow$
 $e : t \Rightarrow_c f$

code-pred *expr-compiles-to* .

lemma *expr-compiles-to-sound*:

assumes $e : t \Rightarrow_c f$
shows *expr-sem* $\sigma\ e = density\ (stock-measure\ t)\ (\lambda x. ennreal\ (eval-cexpr\ f\ \sigma'\ x))$
 $\forall x \in type-universe\ t. eval-cexpr\ f\ \sigma'\ x \geq 0$
 $\Gamma \vdash e : t$
 $t \cdot \Gamma' \vdash_c f : REAL$
 $free-vars\ f \subseteq \{0\}$

proof –

let $\ ?\Gamma = \lambda-. UNIT$
from *assms* **have** $A : (\langle \rangle, \langle \rangle, ?\Gamma, CReal\ 1) \vdash_c e \Rightarrow f\ ?\Gamma \vdash e : t\ free-vars\ e = \{\}$
by (*simp-all add: expr-compiles-to.simps*)
hence *expr-sem* $\sigma\ e = expr-sem\ \sigma'\ e$ **by** (*intro expr-sem-eq-on-vars*) *simp*
with *expr-has-density-cexpr-sound*[*OF A*]
show *expr-sem* $\sigma\ e = density\ (stock-measure\ t)\ (\lambda x. ennreal\ (eval-cexpr\ f\ \sigma'\ x))$
 $\forall x \in type-universe\ t. eval-cexpr\ f\ \sigma'\ x \geq 0$
 $t \cdot \Gamma' \vdash_c f : REAL$
 $free-vars\ f \subseteq \{0\}$ **unfolding** *has-subprob-density-def has-density-def*
eval-cexpr-def

by (*auto intro!: nonneg-cexprD case-nat-in-state-measure*)
from *assms* **have** $(\lambda-. UNIT) \vdash e : t$ **by** (*simp add: expr-compiles-to.simps*)
from *this* **and** *assms* **show** $\Gamma \vdash e : t$
by (*subst expr-typing-cong*) (*auto simp: expr-compiles-to.simps*)
qed

11 Tests

values $\{(t, f) \mid t\ f.\ Val\ (IntVal\ 42) : t \Rightarrow_c f\}$
values $\{(t, f) \mid t\ f.\ Minus\ \$\$ (Val\ (IntVal\ 42)) : t \Rightarrow_c f\}$
values $\{(t, f) \mid t\ f.\ Fst\ \$\$ (Val\ <|IntVal\ 13, IntVal\ 37|>) : t \Rightarrow_c f\}$
values $\{(t, f) \mid t\ f.\ Random\ Bernoulli\ (Val\ (RealVal\ 0.5)) : t \Rightarrow_c f\}$
values $\{(t, f) \mid t\ f.\ Add\ \$\$ <Val\ (IntVal\ 37), Minus\ \$\$ (Val\ (IntVal\ 13))> : t \Rightarrow_c f\}$

```

values {(t, f) | t f. LET Val (IntVal 13) IN LET Minus $$ (Val (IntVal 37)) IN
    Add $$ <Var 0, Var 1> : t ⇒c f}
values {(t, f) | t f. IF Random Bernoulli (Val (RealVal 0.5)) THEN
    Random Bernoulli (Val (RealVal 0.25))
    ELSE
    Random Bernoulli (Val (RealVal 0.75)) : t ⇒c f}
values {(t, f) | t f. LET Random Bernoulli (Val (RealVal 0.5)) IN
    IF Var 0 THEN
    Random Bernoulli (Val (RealVal 0.25))
    ELSE
    Random Bernoulli (Val (RealVal 0.75)) : t ⇒c f}
values {(t, f) | t f. LET Random Gaussian <Val (RealVal 0), Val (RealVal 1)>
    IN
    LET Random Gaussian <Val (RealVal 0), Val (RealVal 1)> IN
    Add $$ <Var 0, Var 1> : t ⇒c f}
values {(t, f) | t f. LET Random UniformInt <Val (IntVal 1), Val (IntVal 6)>
    IN
    LET Random UniformInt <Val (IntVal 1), Val (IntVal 6)> IN
    Add $$ <Var 0, Var 1> : t ⇒c f}

values {(t, f) | t f. LET Random UniformReal <Val (RealVal 0), Val (RealVal
    1)> IN
    LET Random Bernoulli (Var 0) IN
    IF Var 0 THEN Add $$ <Var 1, Val (RealVal 1)> ELSE Var
    1 : t ⇒c f}

```

```

definition cthulhu skill ≡
    LET Random UniformInt (Val <|IntVal 1, IntVal 100|>)
    IN IF Less $$ <Val (IntVal skill), Var 0> THEN
    Val (IntVal skill)
    ELSE IF Or $$ <Less $$ <Var 0, Val (IntVal 6)>,
    Less $$ <Mult $$ <Var 0, Val (IntVal 5)>,
    Add $$ <Val (IntVal skill), Val (IntVal 1)> > > THEN
    Add $$ <IF Less $$ <Val (IntVal skill),
    Random UniformInt <Val (IntVal 1), Val (IntVal 100)> >
    THEN
    Random UniformInt <Val (IntVal 1), Val (IntVal 10)>
    ELSE

```

```

    Val (IntVal 0),
    Val (IntVal skill)>
  ELSE Val (IntVal skill)

```

definition *cthulhu'* (*skill* :: *int*) =

```

LET Random UniformInt (Val <|IntVal 1, IntVal 100|>)
  IN IF Less $$ <Val (IntVal skill), Var 0> THEN
    Val (IntVal skill)
  ELSE IF Or $$ <Less $$ <Var 0, Val (IntVal 6)>,
    Less $$ <Mult $$ <Var 0, Val (IntVal 5)>,
    Add $$ <Val (IntVal skill), Val (IntVal 1)> > > THEN
    LET Random UniformInt (Val <|IntVal 1, IntVal 100|>)
      IN Add $$ <IF Less $$ <Val (IntVal skill), Var 1 > THEN
        Random UniformInt (Val <|IntVal 1, IntVal 10|>)
      ELSE
        Val (IntVal 0),
        Val (IntVal skill)>
    ELSE Val (IntVal skill)

```

values $\{(t, f) \mid t.f. \text{ cthulhu}' 42 : t \Rightarrow_c f\}$

end

References

- [1] S. Bhat, J. Borgström, A. D. Gordon, and C. Russo. Deriving probability density functions from probabilistic functional programs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2013.
- [2] M. Eberl. A verified compiler for probability density functions. Master’s thesis, Institut für Informatik, TU München, 2014. <http://home.in.tum.de/~eberlm/pdfcompiler.pdf>.