A Verified Compiler for Probability Density Functions

Manuel Eberl, Johannes Hölzl and Tobias Nipkow

May 26, 2024

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

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* $M \neq \{\} \wedge$
$(\forall x \in$ *space* $M.\ f\ x \geq 0) \wedge (\int^+ x.\ f\ x\ \partial M) \leq 1$

lemma *is-subprob-densityI*[*intro*]:
$[\![f \in$ *borel-measurable* M; $\bigwedge x.\ x \in$ *space* $M \Longrightarrow f\ x \geq 0$; *space* $M \neq \{\}$; $(\int^+ x.\ f\ x\ \partial M) \leq 1]\!]$
$\Longrightarrow$ *is-subprob-density* $M\ f$
unfolding *is-subprob-density-def* **by** *simp*

lemma *is-subprob-densityD*[*dest*]:
is-subprob-density $M\ f \Longrightarrow f \in$ *borel-measurable* M
is-subprob-density $M\ f \Longrightarrow x \in$ *space* $M \Longrightarrow f\ x \geq 0$
is-subprob-density $M\ f \Longrightarrow$ *space* $M \neq \{\}$
is-subprob-density $M\ f \Longrightarrow (\int^+ x.\ f\ x\ \partial M) \leq 1$
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*]:
$[\![f \in$ *borel-measurable* N; $M =$ *density* $N\ f$; *space* $N \neq \{\}]\!] \Longrightarrow$ *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* $M\ N\ f \Longrightarrow$ *sets* $M =$ *sets* N
unfolding *has-density-def* **by** *simp*

lemma *has-density-space*: *has-density* $M\ N\ f \Longrightarrow$ *space* $M =$ *space* N
unfolding *has-density-def* **by** *simp*

lemma *has-density-emeasure*:
has-density $M\ N\ f \Longrightarrow X \in$ *sets* $M \Longrightarrow$ *emeasure* $M\ X = \int^+ x.\ f\ x *$ *indicator* $X\ x\ \partial N$
unfolding *has-density-def* **by** (*simp-all add*: *emeasure-density*)

lemma *nn-integral-cong'*: $(\bigwedge x.\ x \in$ *space* N =*simp*=> $f\ x = g\ x) \Longrightarrow (\int^+ x.\ f\ x\ \partial N) = (\int^+ x.\ g\ x\ \partial N)$
by (*simp add*: *simp-implies-def cong*: *nn-integral-cong*)

lemma *has-density-emeasure-space*:
  has-density M N f $\Longrightarrow$ *emeasure M* (*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 $\Longrightarrow$ *emeasure* (*density N f*) (*space* (*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*:
  ⟦*has-density M N f*; ($\int^+ x.\ f\ x\ \partial N$) = *1*⟧ $\Longrightarrow$ *is-subprob-density N f*
 by (*auto dest*: *has-densityD*)

lemma *has-density-imp-is-subprob-density'*:
  ⟦*has-density M N f*; *prob-space M*⟧ $\Longrightarrow$ *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* $\bigwedge x.\ x \in$ *space N* $\Longrightarrow f\ x = g\ x$
 shows *has-density M N g*
proof
 from *assms* **show** $g \in$ *borel-measurable N*
  by (*subst measurable-cong*[*of - - f*]) (*auto dest*: *has-densityD*)
 with *assms* **show** *M* = *density N g*
  by (*subst density-cong*[*of - - f*]) (*auto dest*: *has-densityD*)
 from *assms*(*1*) **show** *space N* $\neq$ {} **by** (*rule has-densityD*)
qed

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

lemma *has-density-dens-AE*:
  ⟦*AE y in N*. $f\ y = f'\ y$; $f' \in$ *borel-measurable N*;
   $\bigwedge x.\ x \in$ *space M* $\Longrightarrow f'\ x \geq 0$; *has-density M N f*⟧
    $\Longrightarrow$ *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*:
  ⟦*is-subprob-density N f*; *M* = *density N f*⟧ $\Longrightarrow$ *has-density M N f*
 by (*rule has-densityI*) *auto*

lemma *has-subprob-density-imp-subprob-space'*:
  ⟦*has-density M N f*; *is-subprob-density N f*⟧ $\Longrightarrow$ *subprob-space M*

proof (*rule subprob-spaceI*)
assume *has-density M N f*
hence M = *density N f* **by** (*simp add*: *has-density-def*)
also from ‹*has-density M N f*› **have** *space* ... ≠ {} **by** (*simp add*: *has-density-def*)
finally show *space M* ≠ {} **.**
qed (*auto simp add*: *has-density-emeasure-space dest*: *has-densityD*)

lemma *has-subprob-density-imp-subprob-space*[*dest*]:
is-subprob-density M f ⟹ *subprob-space* (*density M f*)
by (*rule has-subprob-density-imp-subprob-space′*) *auto*

definition *has-subprob-density M N f* ≡ *has-density M N f* ∧ *subprob-space M*

lemma *subprob-space-density-not-empty*: *subprob-space* (*density M f*) ⟹ *space M* ≠ {}
by (*subst space-density*[*symmetric*], *subst subprob-space.subprob-not-empty*, *assumption*) *simp*

lemma *has-subprob-densityI*:
⟦*f* ∈ *borel-measurable N*; *M* = *density N f*; *subprob-space M*⟧ ⟹ *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* ∈ *borel-measurable N space N* ≠ {}
M = *density N f* ($\int^+ x.\ f\ x\ \partial N$) ≤ *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* ∈ *borel-measurable N* ⋀*x*. *x* ∈ *space N* ⟹ *f x* ≥ *0 M* = *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 ⟹ *f* ∈ $N \to_M$ *borel*
by (*auto dest*: *has-subprob-densityD*)

lemma *has-subprob-density-imp-has-density*:
has-subprob-density M N f ⟹ *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* $\bigwedge x.\ x \in$ *space N* $\Longrightarrow f\ x = g\ x$
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 $\bigwedge x.\ x \in$ *space N* $\Longrightarrow f\ x = g\ x$
shows *has-subprob-density M N f* = *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*:
$[\![$*AE y in N*. $f\ y = f'\ y$; $f' \in$ *borel-measurable N*;
$\bigwedge x.\ x \in$ *space M* $\Longrightarrow f'\ x \geq 0$; *has-subprob-density M N f*$]\!]$
$\Longrightarrow$ *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 $\equiv$
($\forall x \in$ *space M*. *has-subprob-density* (*N x*) *R* (*f x*)) $\wedge$ *case-prod f* $\in$ *borel-measurable* ($M \bigotimes_M R$)

lemma *has-parametrized-subprob-densityI*:
assumes $\bigwedge x.\ x \in$ *space M* $\Longrightarrow N\ x =$ *density R* (*f x*)
assumes $\bigwedge x.\ x \in$ *space M* $\Longrightarrow$ *subprob-space* (*N x*)
assumes *case-prod f* $\in$ *borel-measurable* ($M \bigotimes_M R$)
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 $\bigwedge x.\ x \in$ *space M* $\Longrightarrow N\ x =$ *density R* (*f x*)
and $\bigwedge x.\ x \in$ *space M* $\Longrightarrow$ *subprob-space* (*N x*)
and [*measurable-dest*]: *case-prod f* $\in$ *borel-measurable* ($M \bigotimes_M R$)
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* $x \in$ *space M*
shows ($\int^+ y.\ f\ x\ y\ \partial R$) ≤ 1
proof−
have ($\int^+ y.\ f\ x\ y\ \partial R$) = *emeasure* (*density R* (*f x*)) (*space* (*density R* (*f x*)))
using *assms*
by (*auto simp*: *emeasure-density cong*: *nn-integral-cong'* *dest*: *has-parametrized-subprob-densityD*)
also have *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 $\bigwedge x.\ x \in \mathit{space}\ M \Longrightarrow N\ x = N'\ x$
shows *has-parametrized-subprob-density* $M\ N\ R\ f$ = *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 $\bigwedge x.\ x \in \mathit{space}\ M \Longrightarrow AE\ y\ in\ R.\ f\ x\ y = f'\ x\ y$
case-prod $f' \in$ *borel-measurable* $(M \bigotimes_M R)$
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*: $x \in \mathit{space}\ M$
with *assms(3)* **have** *space* $(N\ x)$ = *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 *space* $M \neq \{\}$ $\bigwedge x.\ x \in \mathit{space}\ M \Longrightarrow$ *has-density* $(f\ x)\ N\ (g\ x)$
$f \in$ *measurable* M (*subprob-algebra* N) $X \in$ *sets* N
shows *emeasure* $(M \gg= f)\ X = \int^+ x.\ \int^+ y.\ g\ x\ y * \mathit{indicator}\ X\ y\ \partial N\ \partial M$
proof−
from *assms* **have** *emeasure* $(M \gg= f)\ X = \int^+ x.$ *emeasure* $(f\ x)\ X\ \partial M$
by (*intro emeasure-bind*)
also have ... $= \int^+ x.\ \int^+ y.\ g\ x\ y * \mathit{indicator}\ X\ y\ \partial N\ \partial M$ **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
space $M \neq \{\}$ $\bigwedge x.\ x \in \mathit{space}\ M \Longrightarrow$ *has-density* $(f\ x)\ N\ (g\ x)$
and [*measurable*]: *case-prod* $g \in$ *borel-measurable* $(M \bigotimes_M N)$ $f \in$ *measurable* M (*subprob-algebra* N)
shows $(M \gg= f)$ = *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* $(M \gg= f)$ = *sets* (*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** $X \in$ *sets* ($M \gg= f$)
with *eq* **have** [*measurable*]: $X \in$ *sets* N **by** *auto*
with *assms* **have** *emeasure* ($M \gg= f$) $X = \int^+ x.\ \int^+ y.\ g\ x\ y * $ *indicator* $X\ y\ \partial N\ \partial M$
by (*intro emeasure-bind-density*) *simp-all*
also from ‹$X \in$ *sets* N› **have** ... $= \int^+ y.\ \int^+ x.\ g\ x\ y *$ *indicator* $X\ y\ \partial M\ \partial N$
by (*intro sfNM.Fubini′*) *measurable*
also {
fix y **assume** $y \in$ *space* N
have ($\lambda x.\ g\ x\ y$) $=$ *case-prod* $g \circ (\lambda x.\ (x,\ y))$ **by** (*rule ext*) *simp*
also from ‹$y \in$ *space* N› **have** ... $\in$ *borel-measurable* M
by (*intro measurable-comp*[*OF - assms*(*5*)] *measurable-Pair2′*)
finally have ($\lambda x.\ g\ x\ y$) $\in$ *borel-measurable* M **.**
}
hence ... $= \int^+ y.\ (\int^+ x.\ g\ x\ y\ \partial M) *$ *indicator* $X\ y\ \partial N$
by (*intro nn-integral-cong nn-integral-multc*) *simp-all*
also from ‹$X \in$ *sets* N› **and** *assms* **have** ... $=$ *emeasure* (*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* ($M \gg= f$) $X =$ *emeasure* (*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
space $M \neq \{\}$ $\bigwedge x.\ x \in$ *space* $M \Longrightarrow$ *has-density* ($f\ x$) N ($g\ x$)
case-prod $g \in$ *borel-measurable* ($M \bigotimes_M N$)
$f \in$ *measurable* M (*subprob-algebra* N)
shows *has-density* ($M \gg= f$) N ($\lambda y.\ \int^+ x.\ g\ x\ y\ \partial M$)
proof
interpret *sigma-finite-measure* M **by** *fact*
show ($\lambda y.\ \int^+ x.\ g\ x\ y\ \partial M$) $\in$ *borel-measurable* N **using** *assms*
by (*intro borel-measurable-nn-integral*, *subst measurable-pair-swap-iff*) *simp*
show $M \gg= f =$ *density* N ($\lambda y.\ \int^+ x.\ g\ x\ y\ \partial M$)
by (*intro bind-density*) (*simp-all add*: *assms*)
from ‹*space* $M \neq \{\}$› **obtain** x **where** $x \in$ *space* M **by** *blast*
with *assms* **have** *has-density* ($f\ x$) N ($g\ x$) **by** *simp*
thus *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*: *space* $M \neq \{\}$ **and** *dens-M*: *has-density* $M\ N\ \delta M$
and *dens-f*: $\bigwedge x.\ x \in$ *space* $M \Longrightarrow$ *has-density* ($f\ x$) R ($\delta f\ x$)
and *Mδf*: *case-prod* $\delta f \in$ *borel-measurable* ($N \bigotimes_M R$)
and *Mf*: $f \in$ *measurable* N (*subprob-algebra* R)

shows *has-density* $(M \gg= f)$ R $(\lambda y. \int^{+}x. \delta M\ x * \delta f\ x\ y\ \partial N)$
proof−
from *dens-M* **have** *M-M*: *measurable* M = *measurable* N
by (*intro ext measurable-cong-sets*) (*auto dest*: *has-densityD*)
from *dens-M* **have** *M-MR*: *measurable* $(M \bigotimes_M R)$ = *measurable* $(N \bigotimes_M R)$
by (*intro ext measurable-cong-sets sets-pair-measure-cong*) (*auto dest*: *has-densityD*)
have *has-density* $(M \gg= f)$ 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: $y \in$ *space* R
have $(\lambda x. \delta f\ x\ y)$ = *case-prod* $\delta f \circ (\lambda x. (x,y))$ **by** (*rule ext*) (*simp add*: *o-def*)
also have ... $\in$ *borel-measurable* N **by** (*intro measurable-comp*[*OF - Mδf*] *measurable-Pair2′ A*)
finally have *M-δf′*: $(\lambda x. \delta f\ x\ y) \in$ *borel-measurable* N **.**

from *dens-M* **have** M = *density* N δM **by** (*auto dest*: *has-densityD*)
also from *dens-M* **have** $(\int^{+}x. \delta f\ x\ y\ \partial ...)$ = $\int^{+}x. \delta M\ x * \delta f\ x\ y\ \partial N$
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* $(M \gg= f)$ 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
space $M \neq \{\}$ $\bigwedge x. x \in$ *space* $M \Longrightarrow$ *has-density* $(f\ x)$ N $(g\ x)$
case-prod $g \in$ *borel-measurable* $(M \bigotimes_M N)$
$f \in$ *measurable* M (*subprob-algebra* N)
shows *has-subprob-density* $(M \gg= f)$ N $(\lambda y. \int^{+}x. g\ x\ y\ \partial M)$
proof (*unfold has-subprob-density-def*, *intro conjI*)
from *assms* **show** *has-density* $(M \gg= f)$ 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* $(M \gg= f)$ **by** (*intro subprob-space-bind*)
qed

lemma *bind-has-subprob-density′*:
assumes *has-subprob-density* M N δM *space* $R \neq \{\}$ *sigma-finite-measure* R
$\bigwedge x. x \in$ *space* $M \Longrightarrow$ *has-subprob-density* $(f\ x)$ R $(\delta f\ x)$
case-prod $\delta f \in$ *borel-measurable* $(N \bigotimes_M R)$ $f \in$ *measurable* N (*subprob-algebra* R)
shows *has-subprob-density* $(M \gg= f)$ 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* $(M \gg= f)$ 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* $(M \gg= f)$

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

lemma *null-measure-has-subprob-density*:
 space M ≠ {} ⟹ *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* ∈ *sets R*
 shows *emeasure* (*N x*) *X* = ∫⁺*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* ∈ *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* = ∫⁺*y*. *f x y* ∗ *indicator X y*
∂*R*
  by (*rule emeasure-density*)
 finally show *?thesis* **.**
qed

lemma *emeasure-count-space-density-singleton*:
 assumes *x* ∈ *A has-density M* (*count-space A*) *f*
 shows *emeasure M* {*x*} = *f x*
proof−
 from *has-densityD*[*OF assms*(*2*)] **have** *nonneg*: ⋀*x*. *x* ∈ *A* ⟹ *f x* ≥ *0* **by** *simp*
 from *assms* **have** *M*: *M* = *density* (*count-space A*) *f* **by** (*intro has-densityD*)
 from *assms* **have** *emeasure M* {*x*} = ∫⁺*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* ∈ *A*
 shows *f x* ≤ *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*: *inj f* **and** *inv*: $\bigwedge x.\ x \in space\ N \Longrightarrow f'\ (f\ x) = x$
shows *has-density* (*embed-measure M f*) (*embed-measure N f*) ($\delta \circ f'$) $\longleftrightarrow$ *has-density M N* δ
(**is** *has-density ?M′ ?N′ ?δ′* $\longleftrightarrow$ *has-density M N* δ)
proof
assume *dens*: *has-density ?M′ ?N′ ?δ′*
show *has-density M N* δ
proof
from *dens* **show** *space N* $\neq$ {} **by** (*auto simp*: *space-embed-measure dest*: *has-densityD*)
from *dens* **have** *Mδf′*: $\delta \circ f' \in$ *borel-measurable ?N′* **by** (*rule has-densityD*)
hence *Mδf′f*: $\delta \circ f' \circ f \in$ *borel-measurable N*
by (*rule-tac measurable-comp*, *rule-tac measurable-embed-measure2*[*OF inj*])
thus *Mδ*: $\delta \in$ *borel-measurable N* **by** (*simp cong*: *measurable-cong add*: *inv*)
from *dens* **have** *embed-measure M f* = *density* (*embed-measure N f*) ($\delta \circ f'$) **by** (*rule has-densityD*)
also have ... = *embed-measure* (*density N* ($\delta \circ f' \circ f$)) *f*
by (*simp only*: *density-embed-measure*[*OF inj Mδf′*])
also have *density N* ($\delta \circ f' \circ f$) = *density N* δ
by (*intro density-cong*[*OF Mδf′f Mδ*]) (*simp-all add*: *inv*)
finally show *M* = *density N* δ **by** (*simp add*: *embed-measure-eq-iff*[*OF inj*])
qed
next
assume *dens*: *has-density M N* δ
show *has-density ?M′ ?N′ ?δ′*
proof
from *dens* **show** *space ?N′* $\neq$ {} **by** (*auto simp*: *space-embed-measure dest*: *has-densityD*)
have *Mf′f*: ($\lambda x.\ f'\ (f\ x)$) $\in$ *measurable N N* **by** (*subst measurable-cong*[*OF inv*]) *simp-all*
from *dens* **have** *Mδ*: $\delta \in$ *borel-measurable N* **by** (*auto dest*: *has-densityD*)
from *Mf′f* **and** *dens* **show** *Mδf′*: $\delta \circ f' \in$ *borel-measurable* (*embed-measure N f*)
by (*intro measurable-comp*) (*erule measurable-embed-measure1*, *rule has-densityD*)
have *embed-measure M f* = *embed-measure* (*density N* δ) *f*
by (*simp only*: *has-densityD*[*OF dens*])
also from *inv* **and** *dens* **and** *measurable-comp*[*OF Mf′f Mδ*]
have *density N* δ = *density N* (*?δ′* $\circ f$)
by (*intro density-cong*[*OF Mδ*]) (*simp add*: *o-def*, *simp add*: *inv o-def*)
also have *embed-measure* (*density N* (*?δ′* $\circ f$)) *f* = *density* (*embed-measure N f*) ($\delta \circ f'$)
by (*simp only*: *density-embed-measure*[*OF inj Mδf′*, *symmetric*])
finally show *embed-measure M f* = *density* (*embed-measure N f*) ($\delta \circ f'$) **.**
qed
qed

lemma *has-density-embed-measure′*:
assumes *inj*: *inj f* **and** *inv*: $\bigwedge x.\ x \in space\ N \Longrightarrow f'\ (f\ x) = x$ **and**
sets-M: *sets M = sets (embed-measure N f)*
shows *has-density (distr M N f′) N (δ ∘ f)* ⟷ *has-density M (embed-measure N f) δ*
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′*: $\bigwedge x.\ x \in space\ M \Longrightarrow f\ (f'\ x) = x$
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) (λx. f (f′ x))*
by (*subst distr-cong*[*OF refl - inv′, of - M*]) (*simp-all add*: *sets-embed-measure inj sets-M*)
also have ... = *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*: $\bigwedge x.\ x \in space\ N \Longrightarrow f'\ (f\ x) = x$ **and**
has-density M (embed-measure N f) δ
shows *has-density (distr M N f′) N (δ ∘ 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*: $\bigwedge x.\ x \in space\ N \Longrightarrow f'\ (f\ x) = x$ **and**
has-subprob-density M (embed-measure N f) δ
shows *has-subprob-density (distr M N f′) N (δ ∘ f)*
proof (*unfold has-subprob-density-def*, *intro conjI*)
from *assms* **show** *has-density (distr M N f′) N (δ ∘ 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*]: ¬ *top* ≤ (*1*::*ennreal*)
by (*simp add*: *top-unique*)

lemma *range-int*: *range int* = {*n*. *n* ≥ *0*}
proof (*intro equalityI subsetI*)
fix *n* :: *int* **assume** *n* ∈ {*n*. *n* ≥ *0*}
hence *n* = *int* (*nat n*) **by** *simp*
thus *n* ∈ *range int* **by** *blast*
qed *auto*

lemma *range-exp*: *range* (*exp* :: *real* ⇒ *real*) = {*x*. *x* > *0*}
proof (*intro equalityI subsetI*)
fix *x* :: *real* **assume** *x* ∈ {*x*. *x* > *0*}
hence *x* = *exp* (*ln x*) **by** *simp*
thus *x* ∈ *range exp* **by** *blast*
qed *auto*

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

lemma *distr-mult-real*:
assumes *c* ≠ *0 has-density M lborel* (*f* :: *real* ⇒ *ennreal*)
shows *has-density* (*distr M borel* ((∗) *c*)) *lborel* (λ*x*. *f* (*x* / *c*) ∗ *inverse* (*abs c*))
(**is** *has-density ?M′ - ?f′*)
proof
from *assms*(*2*) **have** *M* = *density lborel f* **by** (*rule has-densityD*)
also from *assms* **have** *Mf*[*measurable*]: *f* ∈ *borel-measurable borel*
by (*auto dest*: *has-densityD*)
hence *distr* (*density lborel f*) *borel* ((∗) *c*) = *density lborel ?f′* (**is** *?M1* = *?M2*)
proof (*intro measure-eqI*)
fix *X* **assume** *X*[*measurable*]: *X* ∈ *sets* (*distr* (*density lborel f*) *borel* ((∗) *c*))
with *assms* **have** *emeasure ?M1 X* = ∫ ⁺*x*. *f x* ∗ *indicator X* (*c* ∗ *x*) ∂*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* ∗ *indicator X x* ∂*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* ⇒ *ennreal*)
shows *has-density* (*distr M borel uminus*) *lborel* (λ*x*. *f* (− *x*))
proof−
from *assms* **have** *has-density* (*distr M borel* ((∗) (− *1*))) *lborel*
(λ*x*. *f* (*x* / −*1*) ∗ *ennreal* (*inverse* (*abs* (−*1*))))
by (*intro distr-mult-real*) *simp-all*
also have (∗) (−*1*) = (*uminus* :: *real* ⇒ *real*) **by** (*intro ext*) *simp*
also have (λ*x*. *f* (*x* / −*1*) ∗ *ennreal* (*inverse* (*abs* (−*1*)))) = (λ*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* ⇒ *ennreal*)
shows *has-density* (*distr M borel* ((+) *c*)) *lborel* (λ*x*. *f* (*x* − *c*))
proof
from *assms* **have** *M* = *density lborel f* **by** (*rule has-densityD*)
also from *assms* **have** *Mf*[*measurable*]: *f* ∈ *borel-measurable borel*
by (*auto dest*: *has-densityD*)
hence *distr* (*density lborel f*) *borel* ((+) *c*) = *density lborel* (λ*x*. *f* (*x* − *c*)) (**is**
?M1 = *?M2*)
proof (*intro measure-eqI*)
fix *X* **assume** *X*: *X* ∈ *sets* (*distr* (*density lborel f*) *borel* ((+) *c*))
with *assms* **have** *emeasure ?M1 X* = $\int^{+}x.$ *f x* ∗ *indicator X* (*c* + *x*) ∂*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*) ∗ *indicator X x* ∂*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 *emeasure ?M1 X* = *emeasure ?M2 X* **.**
 qed *simp*
 finally show *distr M borel* ((+) *c*) = *density lborel* (λ*x*. *f* (*x* − *c*)) **.**
qed (*insert assms*, *auto dest*: *has-densityD*)

lemma *count-space-uminus*:
 count-space UNIV = *distr* (*count-space UNIV*) (*count-space UNIV*) (*uminus* :: (′*a* :: *ring* ⇒ -))
proof (*rule distr-bij-count-space*[*symmetric*])
 show *bij* (*uminus* :: ′*a* ⇒ ′*a*)
  by (*auto intro*!: *o-bij*[**where** *g*=*uminus*])
qed

lemma *count-space-plus*:
 count-space UNIV = *distr* (*count-space UNIV*) (*count-space UNIV*) ((+) (*c* :: (′*a* :: *ring*)))
 by (*rule distr-bij-count-space* [*symmetric*]) *simp*

lemma *distr-uminus-ring-count-space*:
 assumes *has-density M* (*count-space UNIV*) (*f* :: - :: *ring* ⇒ *ennreal*)
 shows *has-density* (*distr M* (*count-space UNIV*) *uminus*) (*count-space UNIV*) (λ*x*. *f* (− *x*))
proof
 from *assms* **have** *M* = *density* (*count-space UNIV*) *f* **by** (*rule has-densityD*)
 also have *distr* (*density* (*count-space UNIV*) *f*) (*count-space UNIV*) *uminus* =
   density (*count-space UNIV*)(λ*x*. *f* (− *x*)) (**is** *?M1* = *?M2*)
 proof (*intro measure-eqI*)
  fix *X* **assume** *X*: *X* ∈ *sets* (*distr* (*density* (*count-space UNIV*) *f*) (*count-space UNIV*) *uminus*)
  with *assms* **have** *emeasure ?M1 X* = $\int^+$*x*. *f x* ∗ *indicator X* (−*x*) ∂*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** ... = *emeasure ?M2 X*
   by (*subst count-space-uminus*) (*simp-all add*: *nn-integral-distr emeasure-density*)
  finally show *emeasure ?M1 X* = *emeasure ?M2 X* **.**
 qed *simp*
 finally show *distr M* (*count-space UNIV*) *uminus* = *density* (*count-space UNIV*) (λ*x*. *f* (− *x*)) **.**
qed (*insert assms*, *auto dest*: *has-densityD*)

lemma *distr-plus-ring-count-space*:
 assumes *has-density M* (*count-space UNIV*) (*f* :: - :: *ring* ⇒ *ennreal*)
 shows *has-density* (*distr M* (*count-space UNIV*) ((+) *c*)) (*count-space UNIV*) (λ*x*. *f* (*x* − *c*))
proof
 from *assms* **have** *M* = *density* (*count-space UNIV*) *f* **by** (*rule has-densityD*)

also have *distr* (*density* (*count-space UNIV*) *f*) (*count-space UNIV*) ((+) *c*) =
density (*count-space UNIV*)(λx. *f* (*x* − *c*)) (**is** *?M1* = *?M2*)
proof (*intro measure-eqI*)
fix *X* **assume** *X*: *X* ∈ *sets* (*distr* (*density* (*count-space UNIV*) *f*) (*count-space UNIV*) ((+) *c*))
with *assms* **have** *emeasure ?M1 X* = $\int^+ x$. *f x* ∗ *indicator X* (*c* + *x*) ∂*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** ... = *emeasure ?M2 X*
by (*subst count-space-plus*[*of* −*c*]) (*simp-all add*: *nn-integral-distr emeasure-density*)
finally show *emeasure ?M1 X* = *emeasure ?M2 X* **.**
qed *simp*
finally show *distr M* (*count-space UNIV*) ((+) *c*) = *density* (*count-space UNIV*) (λx. *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* ≤ *b* ⟹ *emeasure* (*distr* (*density lborel f*) *lborel h*) {*a*..*b*} =
emeasure (*density 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 *distr M borel h* = *density lborel g*
proof (*rule measure-eqI-generator-eq*[*OF Int-stable-Icc*, *of UNIV*])
{
fix *x* :: *real*
obtain *n* :: *nat* **where** *n* > *abs x* **using** *reals-Archimedean2* **by** *auto*
hence ∃ *n*::*nat*. *x* ∈ {−*real n*..*real n*} **by** (*intro exI*[*of* - *n*]) *auto*
}
thus ($\bigcup$ *i*::*nat*. {−*real i*..*real i*}) = *UNIV* **by** *blast*
next
fix *i* :: *nat*
from *subprob-space* **have** *emeasure* (*distr M borel h*) {−*real i*..*real i*} ≤ *1*
by (*intro subprob-space.subprob-emeasure-le-1*) (*auto dest*: *has-subprob-densityD*)
thus *emeasure* (*distr M borel h*) {− *real i*..*real i*} ≠ ∞ **by** *auto*
next
fix *X* :: *real set* **assume** *X* ∈ *range* (λ(*a*,*b*). {*a*..*b*})

then obtain *a b* **where** $X = \{a..b\}$ **by** *auto*
with *dens* **have** *emeasure* (*distr M lborel h*) *X* = *emeasure* (*density lborel g*) *X*
by (*cases* $a \leq b$) (*auto simp*: *measure-eq dest*: *has-subprob-densityD*)
also have *distr M lborel h* = *distr M borel h*
by (*rule distr-cong*) *auto*
finally show *emeasure* (*distr M borel h*) *X* = *emeasure* (*density 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 lborel f*
shows *has-subprob-density* (*distr M borel exp*) *lborel*
(λx. *if* $x > 0$ *then f* (*ln x*) $*$ *ennreal* (*inverse x*) *else 0*)
(**is** *has-subprob-density - - ?g*)
proof (*rule subprob-density-distr-real-eq*[*OF dens*])
from *dens* **have** [*measurable*]: $f \in$ *borel-measurable borel*
by (*auto dest*: *has-subprob-densityD*)

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

fix *a b* :: *real* **assume** $a \leq b$

let *?A* = λi. {*inverse* (*Suc i*) :: *real* ..}
let *?M1* = *distr* (*density lborel f*) *lborel exp* **and** *?M2* = *density lborel ?g*
{
fix *x* :: *real* **assume** $\forall$ *i*. *x* < *inverse* (*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*∈{*a..b*}. $x \leq 0$} ∪ ($\bigcup$*i*. *?A i* ∩ {*a..b*}) (**is** - = *?C* ∪ *?D*)
by (*auto simp*: *not-le*)
have *inv-le*: $\bigwedge$*x i*. $x \geq$ *inverse* (*real* (*Suc i*)) $\Longrightarrow \neg(x \leq 0)$
by (*subst not-le*, *erule dual-order.strict-trans1*, *simp*)
hence *emeasure ?M1* {*a..b*} = *emeasure ?M1 ?C* + *emeasure ?M1 ?D*
by (*subst decomp*, *intro plus-emeasure*[*symmetric*]) *auto*
also have *emeasure ?M1 ?C* = *0* **by** (*subst emeasure-distr*) *auto*
also have *0* = *emeasure ?M2 ?C*
by (*subst emeasure-density*, *simp*, *simp*, *rule sym*, *subst nn-integral-0-iff*) *auto*
also have *emeasure ?M1* ($\bigcup$*i*. *?A i* ∩ {*a..b*}) = (*SUP i*. *emeasure ?M1* (*?A i* ∩ {*a..b*}))
by (*rule SUP-emeasure-incseq*[*symmetric*])
(*auto simp*: *incseq-def max-def not-le dest*: *order.strict-trans1*)
also have $\bigwedge$*i*. *emeasure ?M1* (*?A i* ∩ {*a..b*}) = *emeasure ?M2* (*?A i* ∩ {*a..b*})
proof (*case-tac inverse* (*Suc i*) $\leq$ *b*)
fix *i* **assume** *True*: *inverse* (*Suc i*) $\leq$ *b*
let *?a* = *inverse* (*Suc i*)
from ‹$a \leq b$› **have** *A*: *?A i* ∩ {*a..b*} = {*max ?a a..b*} (**is** *?E* = *?F*) **by** *auto*
hence *emeasure ?M1 ?E* = *emeasure ?M1 ?F* **by** *simp*

also have *strict-mono-on* {*max* (*inverse* (*real* (*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 ‹*a* ≤ *b*› *True dens*
have *emeasure ?M1 ?F* = *emeasure* (*density lborel* (λ*x*. *f* (*ln x*) ∗ *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*[*symmetric*]
also have *emeasure* (*density lborel* (λ*x*. *f* (*ln x*) ∗ *inverse x*)) *?E* = *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 i* ∩ {*a..b*}) = *emeasure ?M2* (*?A i* ∩ {*a..b*}) **.**
qed *simp*
hence (*SUP i*. *emeasure ?M1* (*?A i* ∩ {*a..b*})) = (*SUP i*. *emeasure ?M2* (*?A i* ∩ {*a..b*})) **by** *simp*
also have ... = *emeasure ?M2* (⋃*i*. *?A i* ∩ {*a..b*})
by (*rule SUP-emeasure-incseq*)
(*auto simp*: *incseq-def max-def not-le dest*: *order.strict-trans1*)
also have *emeasure ?M2 ?C* + *emeasure ?M2 ?D* = *emeasure ?M2* (*?C* ∪ *?D*)
by (*rule plus-emeasure*) (*auto dest*: *inv-le simp del*: *of-nat-Suc*)
also note *decomp*[*symmetric*]
finally show *emeasure ?M1* {*a..b*} = *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* (*distr M borel* (λ*x*. − *inverse x*)) *lborel*
(λ*x*. *f* (−*inverse x*) ∗ *ennreal* (*inverse* (*x* ∗ *x*)))
(**is** *has-subprob-density - - ?g*)
proof (*rule subprob-density-distr-real-eq*[*OF dens*])
from *dens* **have** *Mf*[*measurable*]: *f* ∈ *borel-measurable borel* **by** (*auto dest*: *has-subprob-densityD*)
show *Mg*: *?g* ∈ *borel-measurable borel* **by** *measurable*

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

have *inv-le*: $\bigwedge x\ i.\ x \geq$ *inverse* (*real* (*Suc i*)) $\Longrightarrow \neg(x \leq 0)$
by (*subst not-le*, *erule dual-order.strict-trans1*, *simp*)
have $\bigwedge x.\ x > 0 \Longrightarrow \exists\, i.\ x \geq$ *inverse* (*Suc i*)
proof (*rule ccontr*)
fix *x* :: *real* **assume** $x > 0$ $\neg(\exists\, i.\ x \geq$ *inverse* (*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 ‹$x > 0$› **show** *False* **by** *simp*
qed
hence *A*: ($\bigcup i.$ *?A2 i*) = {*0*<..} **by** (*auto dest*: *inv-le simp del*: *of-nat-Suc*)
moreover have $\bigwedge x.\ x < 0 \Longrightarrow \exists\, i.\ x \leq -$*inverse* (*Suc i*)
proof (*rule ccontr*)
fix *x* :: *real* **assume** $x < 0$ $\neg(\exists\, i.\ x \leq -$*inverse* (*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 ‹$x < 0$› **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*: *UNIV* = ($\bigcup i.$ *?A1 i*) ∪ ($\bigcup i.$ *?A2 i*) ∪ {*0*} **by** (*subst A*, *subst B*) *force*
have *UN-Int-distrib*: $\bigwedge f\ A.$ ($\bigcup i.\ f\ i$) ∩ *A* = ($\bigcup i.\ f\ i$ ∩ *A*) **by** *blast*
have *decomp*: {*a*..*b*} = ($\bigcup i.$ *?A1 i* ∩ {*a*..*b*}) ∪ ($\bigcup i.$ *?A2 i* ∩ {*a*..*b*}) ∪ *?C* (**is** - = *?D* ∪ *?E* ∪ -)
by (*subst Int-UNIV-left*[*symmetric*], *simp only*: *C Int-Un-distrib2 UN-Int-distrib*)
(*simp split*: *if-split*)
have *emeasure ?M1* {*a*..*b*} = *emeasure ?M1 ?D* + *emeasure ?M1 ?E* + *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 ($\lambda x.\ -$ *inverse x*) −‘ {*0* :: *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* = *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* ∩ {*a*..*b*}) = (*SUP i. emeasure ?M1* (*?A1 i* ∩ {*a*..*b*}))
by (*rule SUP-emeasure-incseq*[*symmetric*])
(*auto simp*: *incseq-def max-def not-le dest*: *order.strict-trans1*)
also have $\bigwedge i.$ *emeasure ?M1* (*?A1 i* ∩ {*a*..*b*}) = *emeasure ?M2* (*?A1 i* ∩ {*a*..*b*})
proof (*case-tac* −*inverse* (*Suc i*) ≥ *a*)

fix *i* **assume** *True*: $-$*inverse* (*Suc i*) $\geq$ *a*
let *?a* = $-$*inverse* (*Suc i*)
from ‹$a \leq b$› **have** *A*: *?A1 i* $\cap$ {*a..b*} = {*a..min ?a b*} (**is** *?F* = *?G*) **by** *auto*
hence *emeasure ?M1 ?F* = *emeasure ?M1 ?G* **by** *simp*
also have *strict-mono-on* {*a..min ?a b*} (λx. $-$*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 ‹$a \leq b$› *True dens*
have *emeasure ?M1 ?G* = *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*[*symmetric*]
finally show *emeasure ?M1* (*?A1 i* $\cap$ {*a..b*}) = *emeasure ?M2* (*?A1 i* $\cap$ {*a..b*})
.
qed *simp*
hence (*SUP i. emeasure ?M1* (*?A1 i* $\cap$ {*a..b*})) = (*SUP i. emeasure ?M2* (*?A1 i* $\cap$ {*a..b*})) **by** *simp*
also have ... = *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 *emeasure ?M1* ($\bigcup$*i*. *?A2 i* $\cap$ {*a..b*}) = (*SUP i. 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*. *emeasure ?M1* (*?A2 i* $\cap$ {*a..b*}) = *emeasure ?M2* (*?A2 i* $\cap$ {*a..b*})
proof (*case-tac inverse* (*Suc i*) $\leq$ *b*)
fix *i* **assume** *True*: *inverse* (*Suc i*) $\leq$ *b*
let *?a* = *inverse* (*Suc i*)
from ‹$a \leq b$› **have** *A*: *?A2 i* $\cap$ {*a..b*} = {*max ?a a..b*} (**is** *?F* = *?G*) **by** *auto*
hence *emeasure ?M1 ?F* = *emeasure ?M1 ?G* **by** *simp*
also have *strict-mono-on* {*max ?a a..b*} (λx. $-$*inverse x*)
by (*rule strict-mono-onI*) (*auto dest*!: *inv-le simp*: *not-le simp del*: *of-nat-Suc*)
with ‹$a \leq b$› *True dens*
have *emeasure ?M1 ?G* = *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*[*symmetric*]
finally show *emeasure ?M1* (*?A2 i* $\cap$ {*a..b*}) = *emeasure ?M2* (*?A2 i* $\cap$ {*a..b*})
.
qed *simp*
hence (*SUP i. emeasure ?M1* (*?A2 i* $\cap$ {*a..b*})) = (*SUP i. emeasure ?M2* (*?A2 i* $\cap$ {*a..b*})) **by** *simp*
also have ... = *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 *emeasure ?M2 ?D* + *emeasure ?M2 ?E* + *emeasure ?M2 ?C* = *emeasure ?M2* {*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 *emeasure ?M1* {*a..b*} = *emeasure ?M2* {*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* (λ*x*. *f* (*inverse x*) ∗ *ennreal* (*inverse* (*x* ∗ *x*)))
proof (*unfold has-subprob-density-def*, *intro conjI*)
let *?g′* = (λ*x*. *f* (−*inverse x*) ∗ *ennreal* (*inverse* (*x* ∗ *x*)))
have *prob*: *has-subprob-density* (*distr M borel* (λ*x*. −*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* (λ*x*. −*inverse x*)) *lborel*
(λ*x*. *f* (−*inverse x*) ∗ *ennreal* (*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*
(λ*x*. *f* (*inverse x*) ∗ *ennreal* (*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* × *real*) ⇒ *ennreal*)
shows *has-density* (*distr M borel* (*case-prod* (+))) *lborel* (λ*z*. ∫ ⁺*x*. *f* (*x*, *z* − *x*) ∂*lborel*)
(**is** *has-density ?M′* - *?f′*)
proof
from *has-densityD*[*OF assms*] **have** *Mf*[*measurable*]: *f* ∈ *borel-measurable borel* **by** *simp*
show *Mf′*: (λ*z*. ∫ ⁺ *x*. *f* (*x*, *z* − *x*) ∂*lborel*) ∈ *borel-measurable lborel* **by** *measurable*

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** ... = $\int^+$*z*. *f z* ∗ *indicator* ((λ(*x*, *y*). *x* + *y*) −' *X*) *z* ∂(*lborel* $\bigotimes_M$ *lborel*)
by (*simp add*: *emeasure-density has-densityD*[*OF assms*]
measurable-sets-borel[*OF M-add*] *lborel-prod*)
also have ... = $\int^+$*x*. $\int^+$*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 ... = $\int^+$*x*. $\int^+$*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 ... = $\int^+$*x*. $\int^+$*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 ... = $\int^+$*x*. $\int^+$*z*. *f* (*x*, *z*−*x*) ∗ *indicator X z* ∂*lborel* ∂*lborel*
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also have ... = $\int^+$*z*. $\int^+$*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 ... = $\int^+$*z*. ($\int^+$*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*) × *'a*) ⇒ *ennreal*)
shows *has-density* (*distr M* (*count-space UNIV*) (*case-prod* (+))) (*count-space UNIV*)
(λ*z*. ∫⁺*x*. *f* (*x*, *z* − *x*) ∂*count-space UNIV*)
(**is** *has-density ?M' - ?f'*)
proof
let *?CS* = *count-space UNIV* :: *'a measure* **and** *?CSP* = *count-space UNIV* :: (*'a* × *'a*) *measure*
show *Mf'*: (λ*z*. ∫⁺ *x*. *f* (*x*, *z* − *x*) ∂*count-space UNIV*) ∈ *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-eqI*)
fix *X* :: *'a set* **assume** *X*: *X* ∈ *sets* (*distr M ?CS* (*case-prod* (+)))
hence *emeasure* (*distr M ?CS* (*case-prod* (+))) *X* = *emeasure M* ((λ(*x*, *y*). *x* + *y*) −‘ *X*)
by (*simp-all add*: *emeasure-distr*)
also from *X* **have** ... = ∫⁺*z*. *f z* ∗ *indicator* ((λ(*x*, *y*). *x* + *y*) −‘ *X*) *z* ∂(*?CS* ⨂$_M$ *?CS*)
by (*simp add*: *emeasure-density has-densityD*[*OF assms*(*2*)]
sets-M pair-measure-countable C)
also have ... = ∫⁺*x*. ∫⁺*y*. *f* (*x*, *y*) ∗ *indicator* ((λ(*x*, *y*). *x* + *y*) −‘ *X*) (*x*,*y*) ∂*?CS* ∂*?CS*
by (*rule nn-integral-fst*[*symmetric*]) (*simp add*: *pair-measure-countable C*)
also have ... = ∫⁺*x*. ∫⁺*y*. *f* (*x*, *y*) ∗ *indicator* ((λ(*x*, *y*). *x* + *y*) −‘ *X*) (*x*,*y*)
∂*distr ?CS ?CS* ((+) (−*x*)) ∂*?CS*
by (*rule nn-integral-cong*, *subst count-space-plus*) *simp*
also have ... = ∫⁺*x*. ∫⁺*z*. *f* (*x*, *z*−*x*) ∗ *indicator* ((λ(*x*, *y*). *x* + *y*) −‘ *X*) (*x*, *z*−*x*) ∂*?CS* ∂*?CS*
by (*rule nn-integral-cong*) (*simp-all add*: *nn-integral-distr*)
also have ... = ∫⁺*x*. ∫⁺*z*. *f* (*x*, *z*−*x*) ∗ *indicator X z* ∂*?CS* ∂*?CS*
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also have ... = ∫⁺*z*. ∫⁺*x*. *f* (*x*, *z*−*x*) ∗ *indicator X z* ∂*?CS* ∂*?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 ... = ∫⁺*z*. (∫⁺*x*. *f* (*x*, *z*−*x*) ∂*?CS*) ∗ *indicator X z* ∂*?CS*
by (*rule nn-integral-cong*) (*simp split*: *split-indicator*)
also have ... = *emeasure* (*density ?CS ?f'*) *X* **using** *X* **by** (*simp add*: *emeasure-density*)
finally show *emeasure* (*distr M ?CS* (*case-prod* (+))) *X* = *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* ≡ *BoolVal True*
abbreviation *FALSE* ≡ *BoolVal False*

type-synonym *vname* = *nat*

type-synonym *state* = *vname* ⇒ *val*

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

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

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

abbreviation *extract-pair* ≡ *extract-pair′ id id*
abbreviation *extract-real-pair* ≡ *extract-pair′ extract-real extract-real*
abbreviation *extract-int-pair* ≡ *extract-pair′ extract-int extract-int*

definition *RealPairVal* ≡ λ(*x*,*y*). <|*RealVal x*, *RealVal y*|>

definition *IntPairVal* ≡ λ(*x*,*y*). <|*IntVal x*, *IntVal y*|>

lemma *inj-RealPairVal*: *inj RealPairVal* **by** (*auto simp*: *RealPairVal-def intro*!: *injI*)
lemma *inj-IntPairVal*: *inj IntPairVal* **by** (*auto simp*: *IntPairVal-def intro*!: *injI*)

fun *val-type* :: *val* ⇒ *pdf-type* **where**
  val-type (*BoolVal b*) = *BOOL*
| *val-type* (*IntVal i*) = *INTEG*
| *val-type UnitVal* = *UNIT*
| *val-type* (*RealVal r*) = *REAL*
| *val-type* (<|*v1* , *v2*|>) = (*PRODUCT* (*val-type v1*) (*val-type v2*))

lemma *val-type-eq-REAL*: *val-type x* = *REAL* ⟷ *x* ∈ *RealVal*‘*UNIV*
  by (*cases x*) *auto*

lemma *val-type-eq-INTEG*: *val-type x* = *INTEG* ⟷ *x* ∈ *IntVal*‘*UNIV*
  by (*cases x*) *auto*

definition *type-universe t* = {*v*. *val-type v* = *t*}

lemma *type-universe-nonempty*[*simp*]: *type-universe t* ≠ {}
  by (*induction t*) (*auto intro*: *val-type.simps simp*: *type-universe-def*)

lemma *val-in-type-universe*[*simp*]:
    v ∈ *type-universe* (*val-type v*)
  by (*simp add*: *type-universe-def*)

lemma *BoolVal-in-type-universe*[*simp*]: *BoolVal v* ∈ *type-universe BOOL*
  by (*simp add*: *type-universe-def*)

lemma *IntVal-in-type-universe*[*simp*]: *IntVal v* ∈ *type-universe INTEG*
by (*simp add*: *type-universe-def*)

lemma *type-universe-type*[*simp*]:
w ∈ *type-universe t* ⟷ *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* = *type-universe t2*
shows *t1* = *t2*
proof−
from *type-universe-nonempty* **obtain** *v* **where** *A*: *v* ∈ *type-universe t1* **by** *blast*
hence *t1* = *val-type v* **by** *simp*
also from *A* **and** *assms* **have** *v* ∈ *type-universe t2* **by** *simp*
hence *val-type v* = *t2* **by** *simp*
finally show *?thesis* **.**
qed

lemma *type-universe-eq-iff*[*simp*]: *type-universe t1* = *type-universe t2* ⟷ *t1* = *t2*
by (*blast intro*: *type-universe-eq-imp-type-eq*)

primrec *stock-measure* :: *pdf-type* ⇒ *val measure* **where**
stock-measure UNIT = *count-space* {*UnitVal*}
| *stock-measure INTEG* = *count-space* (*range IntVal*)
| *stock-measure BOOL* = *count-space* (*range BoolVal*)
| *stock-measure REAL* = *embed-measure lborel RealVal*
| *stock-measure* (*PRODUCT t1 t2*) =
embed-measure (*stock-measure t1* $\bigotimes_M$ *stock-measure t2*) (λ(*a*, *b*). <|*a*, *b*|>)

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* = *UNIT* ⟹ *c* ∈ *space M*
assumes ⋀*b*. *t* = *BOOL* ⟹ *g b* ∈ *space M*
assumes ⋀*i*. *t* = *INTEG* ⟹ *h i* ∈ *space M*
assumes *t* = *REAL* ⟹ *j* ∈ *measurable borel M*
assumes ∗: ⋀*t1 t2*. *t* = *PRODUCT t1 t2* ⟹ *case-prod k* ∈ *measurable* (*stock-measure*

t1 $\bigotimes_M$ *stock-measure t2*) *M*
shows (λx. *case x of UnitVal* ⇒ *c* | *BoolVal b* ⇒ *g b* | *IntVal i* ⇒ *h i* | *RealVal r* ⇒ *j r*
| *PairVal y z* ⇒ *k y z*) ∈ *measurable t M*
proof (*cases t*)
case (*PRODUCT t1 t2*) **with** ∗[*of t1 t2*] **show** *?thesis*
by (*auto intro*!: *measurable-embed-measure1 simp*: *split-beta′*)
qed (*auto intro*!: *measurable-embed-measure1 assms*)

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

lemma *type-universe-stock-measure*[*measurable*]: *type-universe t* ∈ *sets* (*stock-measure t*)
using *sets.top*[*of stock-measure t*] **by** *simp*

lemma *inj-RealVal*[*simp*]: *inj RealVal* **by** (*auto intro*!: *inj-onI*)
lemma *inj-IntVal*[*simp*]: *inj IntVal* **by** (*auto intro*!: *inj-onI*)
lemma *inj-BoolVal*[*simp*]: *inj BoolVal* **by** (*auto intro*!: *inj-onI*)
lemma *inj-PairVal*[*simp*]: *inj* (λ(*x*, *y*). <| *x* , *y* |>) **by** (*auto intro*: *injI*)

lemma *measurable-PairVal*[*measurable*]:
fixes *t1 t2* :: *pdf-type*
shows *case-prod PairVal* ∈ *measurable* (*t1* $\bigotimes_M$ *t2*) (*PRODUCT t1 t2*)
using *measurable-embed-measure2*[*measurable*] **by** *simp*

lemma *measurable-RealVal*[*measurable*]: *RealVal* ∈ *measurable borel REAL*
using *measurable-embed-measure2*[*measurable*] **by** *simp*

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

lemma *nn-integral-RealVal*:
f ∈ *borel-measurable REAL* ⟹ ($\int^+ x$. *f x* ∂*REAL*) = ($\int^+ x$. *f* (*RealVal x*) ∂*lborel*)
unfolding *stock-measure.simps* **using** *measurable-embed-measure2*[*measurable*]
by (*subst embed-measure-eq-distr*, *simp-all add*: *nn-integral-distr*)

lemma *nn-integral-IntVal*: ($\int^+ x$. *f x* ∂*INTEG*) = ($\int^+ x$. *f* (*IntVal x*) ∂*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*:
$f \in$ *borel-measurable* (*PRODUCT t1 t2*) $\Longrightarrow$
($\int^+ x.\ f\ x\ \partial PRODUCT\ t1\ t2$) = ($\int^+ x.\ f\ (PairVal\ (fst\ x)\ (snd\ x))\ \partial(t1 \bigotimes_M t2)$)
unfolding *stock-measure.simps*
by (*subst nn-integral-embed-measure*) (*simp-all add*: *split-beta′ inj-on-def*)

lemma *BOOL-E*: $[\![$*val-type* v = *BOOL*; $\bigwedge b.\ v = BoolVal\ b \Longrightarrow P]\!] \Longrightarrow P$
by (*cases v*) *auto*

lemma *PROD-E*: $[\![$*val-type* v = *PRODUCT t1 t2* ;
$\bigwedge a\ b.$ *val-type* $a = t1 \Longrightarrow$ *val-type* $b = t2 \Longrightarrow v$ = <| a, b |> $\Longrightarrow P]\!] \Longrightarrow P$
by (*cases v*) *auto*

lemma *REAL-E*: $[\![$*val-type* v = *REAL*; $\bigwedge b.\ v = RealVal\ b \Longrightarrow P]\!] \Longrightarrow P$
by (*cases v*) *auto*

lemma *INTEG-E*: $[\![$*val-type* v = *INTEG*; $\bigwedge i.\ v = IntVal\ i \Longrightarrow P]\!] \Longrightarrow P$
by (*cases v*) *auto*

lemma *measurable-extract-pair′*[*measurable* (*raw*)]:
fixes *t1 t2* :: *pdf-type*
assumes [*measurable*]: $f \in$ *measurable t1 M*
assumes [*measurable*]: $g \in$ *measurable t2 N*
assumes *h*: $h \in$ *measurable K* (*PRODUCT t1 t2*)
shows ($\lambda x.$ *extract-pair′ f g* ($h\ x$)) $\in$ *measurable K* ($M \bigotimes_M N$)
by (*rule measurable-compose*[*OF h*[*unfolded stock-measure.simps*] *measurable-embed-measure1*])
(*simp add*: *split-beta′*)

lemma *measurable-extract-pair*[*measurable*]: *extract-pair* $\in$ *measurable* (*PRODUCT t1 t2*) ($t1 \bigotimes_M t2$)
by *measurable*

lemma *measurable-extract-real*[*measurable*]: *extract-real* $\in$ *measurable REAL borel*
apply *simp*
apply *measurable*
apply (*rule measurable-embed-measure1*)
apply *simp*
done

lemma *measurable-extract-int*[*measurable*]: *extract-int* $\in$ *measurable INTEG* (*count-space UNIV*)
by *simp measurable*

lemma *measurable-extract-int-pair*[*measurable*]:
extract-int-pair ∈ *measurable* (*PRODUCT INTEG INTEG*) (*count-space UNIV* $\bigotimes_M$ *count-space UNIV*)
by *measurable*

lemma *measurable-extract-real-pair*[*measurable*]:
extract-real-pair ∈ *measurable* (*PRODUCT REAL REAL*) (*borel* $\bigotimes_M$ *borel*)
by *measurable*

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

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

lemma *map-int-pair-measurable*[*measurable*]:
assumes *f*: *case-prod f* ∈ *measurable* (*count-space UNIV* $\bigotimes_M$ *count-space UNIV*) *M*
shows *map-int-pair f g* ∈ *measurable* (*PRODUCT INTEG INTEG*) *M*
proof (*subst measurable-cong*)
fix *w* **assume** *w* ∈ *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 (λ(*x*, *y*). *f x y*) ∘ *extract-int-pair* ∈ *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* ∈ *measurable* (*PRODUCT REAL REAL*) *M*
shows *map-int-pair f g* ∈ *measurable* (*PRODUCT REAL REAL*) *M*
proof (*subst measurable-cong*)
fix *w* **assume** *w* ∈ *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* ∈ *measurable* (*borel* $\bigotimes_M$ *borel*) *M*
shows *map-real-pair f g* ∈ *measurable* (*PRODUCT REAL REAL*) *M*
proof (*subst measurable-cong*)
fix *w* **assume** *w* ∈ *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 (λ(*x*, *y*). *f x y*) ∘ *extract-real-pair* ∈ *measurable* (*stock-measure* (*PRODUCT*

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

lemma *count-space-IntVal-prod*[*simp*]: *INTEG* $\bigotimes_M$ *INTEG* = *count-space* (*range IntVal* × *range IntVal*)
 by (*auto intro*!: *pair-measure-countable*)

lemma *count-space-BoolVal-prod*[*simp*]: *BOOL* $\bigotimes_M$ *BOOL* = *count-space* (*range BoolVal* × *range BoolVal*)
 by (*auto intro*!: *pair-measure-countable*)

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

lemma *singleton-in-stock-measure*[*simp*]: *val-type v* = *t* ⟹ {*v*} ∈ *sets t*
proof (*induction v arbitrary*: *t*)
 case (*PairVal v1 v2*)
 have *A*: {<|*v1*, *v2*|>} = (λ(*v1*,*v2*). <|*v1*,*v2*|>) ‘ ({*v1*}×{*v2*}) **by** *simp*
 from *pair-measureI*[*OF PairVal.IH*, *OF refl refl*] *PairVal.prems*[*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*]:
  emeasure (*stock-measure* (*val-type v*)) {*v*} ≠ ∞
proof (*induction v*)
 case (*RealVal r*)
 have *A*: {*RealVal r*} = *RealVal* ‘ {*r*} **by** *simp*
 have *RealVal* ‘ {*r*} ∈ *sets* (*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 inj-RealVal inj-vimage-image-eq*) *simp*
next
 case (*PairVal v1 v2*)
  let *?M* = λ*x*. *stock-measure* (*val-type x*)
  interpret *sigma-finite-measure stock-measure* (*val-type v2*)
   by (*rule sigma-finite-stock-measure*)
  have *A*: {<|*v1*, *v2*|>} = (λ(*v1*,*v2*). <|*v1*,*v2*|>) ‘ ({*v1*}×{*v2*}) **by** *simp*
  have *B*: {*v1*}×{*v2*} ∈ *?M v1* $\bigotimes_M$ *?M v2*
   by (*intro pair-measureI singleton-in-stock-measure*) *simp-all*
  hence *emeasure* (*?M* (<|*v1*,*v2*|>)) {<|*v1*,*v2*|>} = *emeasure* (*?M v1*) {*v1*} ∗ *emeasure* (*?M v2*) {*v2*}
   by (*simp only*: *stock-measure.simps val-type.simps A emeasure-embed-measure-image inj-PairVal*
   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* ⇒ (*vname* ⇒ *pdf-type*) ⇒ *state measure*
where
state-measure V $\Gamma \equiv \Pi_M$ $y{\in}V$. Γ 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) = (Π_E $y{\in}V$. *type-universe* (Γ y))
by (*simp add*: *state-measure-def space-PiM PiE-eq-empty-iff*)

lemma *state-measure-var-type*:
$\sigma \in$ *space* (*state-measure* V Γ) $\Longrightarrow x \in V \Longrightarrow$ *val-type* (σ x) = Γ x
by (*auto simp*: *state-measure-def space-PiM dest*!: *PiE-mem*)

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

lemma *measurable-merge-stock*[*measurable* (*raw*)]:
$f \in N \rightarrow_M$ *state-measure* V $\Gamma \Longrightarrow g \in N \rightarrow_M$ *state-measure* V' $\Gamma \Longrightarrow$
(λx. *merge* V V' (f x, g x)) $\in N \rightarrow_M$ *state-measure* ($V \cup V'$) Γ
by (*auto simp*: *state-measure-def*)

lemma *comp-in-state-measure*:
assumes $\sigma \in$ *space* (*state-measure* V Γ)
shows $\sigma \circ f \in$ *space* (*state-measure* (f −‘ V) ($\Gamma \circ f$))
using *assms* **by** (*auto simp*: *state-measure-def space-PiM*)

lemma *sigma-finite-state-measure*[*intro*]:
finite $V \Longrightarrow$ *sigma-finite-measure* (*state-measure* V Γ) **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*]: (λ(x, y). <| x , y |>) $\circ$ (λ(x, y). (*RealVal* x, *RealVal* y)) = *RealPairVal*
unfolding *RealPairVal-def* **by** *auto*
have *stock-measure* (*PRODUCT REAL REAL*) =
embed-measure (*embed-measure lborel* (λ(x, y). (*RealVal* x, *RealVal* y))) (*case-prod PairVal*)

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*]: (λ(*x*, *y*). <| *x* , *y* |>) ‘ (*range IntVal* × *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* = *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 ∈ *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* ≠ {} *f* ∈ *measurable M* (*stock-measure t′*)
shows *M* ⤜ (λ*x*. *return-val* (*f x*)) = *distr M* (*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* ∈ *measurable* (*stock-measure t*) (*stock-measure t′*)
shows *return-val x* ⤜ (λ*x*. *return-val* (*f x*)) = *return-val* (*f x*)
proof−
have *return-val x* ⤜ (λ*x*. *return-val* (*f x*)) = *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** $f\ x \in$ *space* (*stock-measure* t')
by (*rule measurable-space*)
(*simp add*: *assms*(*1*) *type-universe-def del*: *type-universe-type*)
hence *return* (*stock-measure* t') $(f\ x)$ = *return-val* $(f\ x)$
by (*simp add*: *return-val-def*)
finally show *?thesis* **.**
qed

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

lemma *bind-assoc-return-val*:
assumes *sets-M*: *sets* M = *sets* (*stock-measure* t)
assumes *Mf*: $f \in$ *measurable* (*stock-measure* t) (*stock-measure* t')
assumes *Mg*: $g \in$ *measurable* (*stock-measure* t') (*stock-measure* t'')
shows ($M \gg=$ (λx. *return-val* $(f\ x)$)) $\gg=$ (λx. *return-val* $(g\ x)$) =
$M \gg=$ (λx. *return-val* $(g\ (f\ x))$)
proof−
have ($M \gg=$ (λx. *return-val* $(f\ x)$)) $\gg=$ (λx. *return-val* $(g\ x)$) =
$M \gg=$ (λx. *return-val* $(f\ x) \gg=$ (λx. *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 ... = $M \gg=$ (λx. *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*: *sets* M = *sets* (*stock-measure* t)
assumes *Mf*: $f \in$ *measurable* (*stock-measure* t) (*stock-measure* t')
shows $M \gg=$ *return-val* $\circ f$ = *distr* M (*stock-measure* t') f
proof−
have $M \gg=$ *return-val* $\circ f = M \gg=$ *return* (*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 ≡ λ *RealVal v* ⇒ *RealVal* (*f v*) | - ⇒ *RealVal* (*f 0*)
definition *lift-IntVal* **where**
  lift-IntVal f ≡ λ *IntVal v* ⇒ *IntVal* (*f v*) | - ⇒ *IntVal* (*f 0*)
definition *lift-RealIntVal* **where**
  lift-RealIntVal f g ≡ λ *IntVal v* ⇒ *IntVal* (*f v*) | *RealVal v* ⇒ *RealVal* (*g v*)

definition *lift-RealIntVal2* **where**
  lift-RealIntVal2 f g ≡
    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 ≡ *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 ∈ *space* (*stock-measure REAL*) ⟹ *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 ∈ *space* (*stock-measure INTEG*) ⟹ *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* ∈ *borel-measurable borel*
  shows *lift-RealVal f* ∈ *measurable REAL REAL*
  unfolding *lift-RealVal-def*
  by (*auto intro*!: *val-case-stock-measurable*)

lemma *measurable-lift-IntVal*[*simp*]: *lift-IntVal f* ∈ *range IntVal* → *range IntVal*

by (*auto simp*: *lift-IntVal-def*)

lemma *measurable-lift-IntVal′*[*measurable*]: *lift-IntVal f* ∈ *measurable INTEG INTEG*
unfolding *lift-IntVal-def*
by (*auto intro*!: *val-case-stock-measurable*)

lemma *split-apply*: (*case x of* (*a*, *b*) ⇒ *f a b*) *y* = (*case x of* (*a*, *b*) ⇒ *f a b y*)
by (*cases x*) *simp*

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

lemma *measurable-lift-Comp-IntVal*[*simp*]:
lift-Comp f g ∈ *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* ∈ *range IntVal* → *range IntVal*
by (*auto simp*: *embed-measure-count-space lift-RealIntVal-def*)

lemma *measurable-lift-RealIntVal-IntVal′*[*measurable*]:
lift-RealIntVal f g ∈ *measurable INTEG INTEG*
by (*auto simp*: *lift-RealIntVal-def intro*!: *val-case-stock-measurable*)

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

lemma *measurable-lift-RealIntVal2-IntVal*[*measurable*]:
lift-RealIntVal2 f g ∈ *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* ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*)
shows *lift-RealIntVal2 f g* ∈ *measurable* (*PRODUCT REAL REAL*) *REAL*
unfolding *lift-RealIntVal2-def* **by** *measurable*

lemma *distr-lift-RealVal*:
fixes *f*
assumes *Mf*[*measurable*]: *f* ∈ *borel-measurable borel*
assumes *pdens*: *has-subprob-density M* (*stock-measure REAL*) *δ*
assumes *dens′*: ⋀*M δ*. *has-subprob-density M lborel δ* ⟹ *has-density* (*distr M*

borel f) *lborel* (*g δ*)
defines *N* ≡ *distr M* (*stock-measure REAL*) (*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* = *distr M* (*stock-measure REAL*) (*lift-RealVal f*) **by** (*simp add*: *N-def*)
also have . . . = *distr M* (*stock-measure REAL*) (*RealVal* ∘ *f* ∘ *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 ... = *distr* (*distr* (*distr M lborel extract-real*) *borel f*) (*stock-measure 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*) *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 *distr* (*distr M lborel extract-real*) *borel f* = *density lborel* (*g* (*δ* ∘ *RealVal*))
by (*rule has-densityD*)
also have *distr* ... (*stock-measure REAL*) *RealVal* = *embed-measure* ... *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 ... = *density* (*embed-measure lborel RealVal*) (*g* (*λx*. *δ* (*RealVal x*)) ∘ *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* = *density* (*stock-measure REAL*) (*g* (*λx*. *δ* (*RealVal x*)) ∘ *extract-real*)
by (*simp add*: *stock-measure.simps*)

from *dens′′*[*unfolded o-def*, *THEN has-densityD*(*1*)] *measurable-extract-real*
show *g* (*λx*. *δ* (*RealVal x*)) ∘ *extract-real* ∈ *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′*: ⋀*M δ*. *has-density M* (*count-space UNIV*) *δ* ⟹

has-density (*distr M* (*count-space UNIV*) *f*) (*count-space UNIV*) (*g* δ)
defines $N \equiv$ *distr M* (*stock-measure INTEG*) (*lift-IntVal f*)
shows *has-density N* (*stock-measure INTEG*) (*g* ($\lambda x.$ δ (*IntVal x*)) $\circ$ *extract-int*)
proof (*rule has-densityI*)
let *?R* = *count-space UNIV* **and** *?S* = *count-space* (*range IntVal*)
have *Mf*: $f \in$ *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* $\circ$ *f* $\circ$ *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 INTEG*) *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 $\circ$ *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 $\circ$ *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* ($\lambda x.$ δ (*IntVal x*)) $\circ$ *extract-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* ($\lambda x.$ δ (*IntVal x*)) $\circ$ *extract-int*)
by (*simp add*: *embed-measure-count-space stock-measure.simps*)

from *dens''*[*unfolded o-def*]
show *g* ($\lambda x.$ δ (*IntVal x*)) $\circ$ *extract-int* $\in$ *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* $\in$ *borel-measurable borel*

assumes *pdens*: *has-subprob-density M* (*stock-measure* (*PRODUCT REAL REAL*)) δ
assumes *dens'*: $\bigwedge$*M* δ. *has-subprob-density M lborel* $\delta \Longrightarrow$ *has-density* (*distr M borel* (*case-prod f*)) *lborel* (*g* δ)
defines *N* $\equiv$ *distr M* (*stock-measure REAL*) (*lift-RealIntVal2 f' f*)
shows *has-density N* (*stock-measure REAL*) (*g* (λ*x*. δ (*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-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 ... = *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 ... = *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-all add*: *RealPairVal-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* ... (*stock-measure REAL*) *RealVal* = *embed-measure* ... *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 ... = *density* (*embed-measure lborel RealVal*) (*g* (λ*x*. δ (*RealPairVal x*)) $\circ$ *extract-real*)
using *dens''*[*unfolded o-def*]
by (*subst density-embed-measure'*, *simp*, *simp add*: *extract-real-def*)
(*erule has-densityD*, *simp add*: *o-def*)
finally show *N* = *density* (*stock-measure REAL*) (*g* (λ*x*. δ (*RealPairVal x*)) $\circ$ *extract-real*)
by (*simp add*: *stock-measure.simps*)

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

by (*intro measurable-comp*)
(*rule measurable-extract-real*, *subst measurable-lborel2*[*symmetric*], *erule has-densityD*)
qed (*subst space-stock-measure*, *simp*)

lemma *distr-lift-IntPairVal*:
fixes $f\ f'$
assumes *pdens*: *has-density M* (*stock-measure* (*PRODUCT INTEG INTEG*)) δ
assumes *dens'*: $\bigwedge M\ \delta.$ *has-density M* (*count-space UNIV*) $\delta \Longrightarrow$
has-density (*distr M* (*count-space UNIV*) (*case-prod f*))
(*count-space UNIV*) ($g\ \delta$)
defines $N \equiv$ *distr M* (*stock-measure INTEG*) (*lift-RealIntVal2* $f\ f'$)
shows *has-density N* (*stock-measure INTEG*) (g ($\lambda x.\ \delta$ (*IntPairVal x*)) $\circ$ *extract-int*)
proof (*rule has-densityI*)
let *?R* = *count-space UNIV* **and** *?S* = *count-space* (*range IntVal*)
and *?T* = *count-space* (*range IntPairVal*) **and** *?tp* = *PRODUCT INTEG INTEG*
have *Mf*: $f \in$ *measurable ?R ?R* **by** *simp*
have *MIV*: *IntVal* $\in$ *measurable ?R ?S* **by** *simp*
from *assms*(*1*) **have** *dens*: *has-density M* (*stock-measure ?tp*) δ
unfolding *has-subprob-density-def* **by** *simp*
from *dens* **have** *M* = *density* (*stock-measure ?tp*) δ **by** (*rule has-densityD*)
hence *sets-M*: *sets M* = *sets ?T* **by** (*subst embed-measure-IntPairVal*[*symmetric*]) *auto*
hence [*simp*]: *space M* = *space ?T* **by** (*rule sets-eq-imp-space-eq*)
from *sets-M* **have** [*simp*]: *measurable M* = *measurable* (*count-space* (*range IntPairVal*))
by (*intro ext measurable-cong-sets*) *simp-all*

have *N* = *distr M* (*stock-measure INTEG*) (*lift-RealIntVal2* $f\ f'$) **by** (*simp add*: *N-def*)

also have ... = *distr M* (*stock-measure INTEG*) (*IntVal* $\circ$ *case-prod f* $\circ$ *extract-int-pair*)
by (*intro distr-cong*) (*auto simp*: *lift-RealIntVal2-def space-embed-measure space-pair-measure IntPairVal-def*)
also have ... = *distr* (*distr* (*distr M* (*count-space UNIV*) *extract-int-pair*)
(*count-space UNIV*) (*case-prod f*)) (*stock-measure INTEG*) *IntVal*
apply (*subst distr-distr*[*of - ?R*, *symmetric*], *simp*, *simp*)
apply (*subst distr-distr*[*symmetric*], *subst stock-measure.simps*, *rule MIV*,
simp-all add: *assms*(*1*) *cong*: *distr-cong*)
done
also have *dens''*: *has-density* (*distr* (*distr M* (*count-space UNIV*) *extract-int-pair*) *?R* (*case-prod f*)) *?R*
(g ($\delta \circ$ *IntPairVal*)) **using** *inj-IntPairVal embed-measure-IntPairVal*
by (*intro dens' has-density-embed-measure''*)
(*insert dens*, *simp-all add*: *extract-int-def embed-measure-count-space IntPairVal-def split*: *prod.split*)

hence *distr* (*distr M* (*count-space UNIV*) *extract-int-pair*) *?R* (*case-prod f*) =
density ?R (*g* ($\delta \circ$ *IntPairVal*)) **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* ($\lambda x.$ δ (*IntPairVal x*)) $\circ$ *extract-int*)
using *dens''*[*unfolded o-def*]
by (*subst density-embed-measure'*, *simp*, *simp add*: *extract-int-def*)
(*erule has-densityD*, *simp add*: *o-def*)
finally show *N* = *density* (*stock-measure INTEG*) (*g* ($\lambda x.$ δ (*IntPairVal x*)) $\circ$ *extract-int*)
by (*simp add*: *embed-measure-count-space stock-measure.simps*)

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

end

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

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

4 Built-in Probability Distributions

4.1 Bernoulli

definition *bernoulli-density* :: *real* ⇒ *bool* ⇒ *ennreal* **where**
bernoulli-density p b = (*if p* ∈ {*0..1*} *then* (*if b then p else 1* − *p*) *else 0*)

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

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

lemma *measurable-bernoulli*[*measurable*]: *bernoulli* ∈ *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* × *real* ⇒ *real* ⇒ *ennreal* **where**
uniform-real-density ≡ λ(*a*,*b*) *x*. *ennreal* (*if a* < *b* ∧ *x* ∈ {*a..b*} *then inverse* (*b* − *a*) *else 0*)

definition *uniform-int-density* :: *int* × *int* ⇒ *int* ⇒ *ennreal* **where**
uniform-int-density ≡ λ(*a*,*b*) *x*. (*if x* ∈ {*a..b*} *then inverse* (*nat* (*b* − *a* + *1*)) *else 0*)

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

lemma *measurable-uniform-density-real*[*measurable*]:
(*case-prod uniform-real-density*) ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*)
proof−
have (*case-prod uniform-real-density*) =
(λ*x*. *uniform-real-density* (*fst* (*fst x*), *snd* (*fst x*)) (*snd x*))
by (*rule ext*) (*simp split*: *prod.split*)
also have ... ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*)
unfolding *uniform-real-density-def*
by (*simp only*: *prod.case*) (*simp add*: *borel-prod*[*symmetric*])
finally show *?thesis* **.**
qed

definition *uniform-int* :: *val* ⇒ *val measure* **where**
uniform-int = *map-int-pair* (λ*l u*. *density INTEG* (*uniform-int-density* (*l*,*u*) *o extract-int*)) (λ-. *undefined*)

definition *uniform-real* :: *val* ⇒ *val measure* **where**
uniform-real = *map-real-pair* (λ*l u*. *density REAL* (*uniform-real-density* (*l*,*u*) *o extract-real*)) (λ-. *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 ∈ *measurable* (*PRODUCT INTEG INTEG*) (*subprob-algebra INTEG*)
unfolding *uniform-int-def*
proof (*rule measurable measurable-subprob-algebra-density*)+
fix *x* :: *int* × *int*

show $integral^N$ *INTEG* (*uniform-int-density* (*fst x*, *snd x*) ∘ *extract-int*) ≤ *1*
proof *cases*
assume *fst x* ≤ *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*′:
(⋀*x*. *x* ∈ *space M* ⟹ *f x* = *g x*) ⟹ *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 ∈ *measurable* (*PRODUCT REAL REAL*) (*subprob-algebra REAL*)
unfolding *uniform-real-def*
proof (*rule measurable measurable-subprob-algebra-density*)+
fix *x* :: *real* × *real*
obtain *l u* **where** [*simp*]: *x* = (*l*, *u*)
by (*cases x*) *auto*
show (∫$^+$*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* ∗ *s*^{2})) / *sqrt* (*2* ∗ *pi* ∗ *s*^{2}) *else 0*)

lemma *measurable-gaussian-density*[*measurable*]:
case-prod gaussian-density ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*)
proof−
have *case-prod gaussian-density* =
(λ(*x*,*y*). (*if snd x* > *0 then exp* (−((*y* − *fst x*)^*2*) / (*2* ∗ *snd x*^*2*)) /
sqrt (*2* ∗ *pi* ∗ *snd x*^*2*) *else 0*))
unfolding *gaussian-density-def* **by** (*intro ext*) (*simp split*: *prod.split*)
also have ... ∈ *borel-measurable* (*borel* $\bigotimes_M$ *borel*)
by (*simp add*: *borel-prod*[*symmetric*])
finally show *?thesis* **.**
qed

definition *gaussian* :: *val* ⇒ *val measure* **where**
gaussian = *map-real-pair* (λ*m 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*) ∘ *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* ⇒ *int* ⇒ *ennreal* **where**
poisson-density′ rate k = *pmf* (*poisson-pmf rate*) (*nat k*) ∗ *indicator* ({*0* <..} × {*0*..}) (*rate*, *k*)

lemma *measurable-poisson-density′*[*measurable*]:
case-prod poisson-density′ ∈ *borel-measurable* (*borel* $\bigotimes_M$ *count-space UNIV*)
proof −
have *case-prod poisson-density′* =
(λ(*rate*, *k*). *rate* ^ *nat k* / *real-of-nat* (*fact* (*nat k*)) ∗ *exp* (−*rate*) ∗ *indicator* ({*0* <..} × {*0*..}) (*rate*, *k*))
by (*auto split*: *split-indicator simp*: *fun-eq-iff poisson-density′-def*)
then show *?thesis*
by *simp*
qed

definition *poisson* :: *val* ⇒ *val measure* **where**
poisson rate = *density INTEG* (*poisson-density′* (*extract-real rate*) *o extract-int*)

lemma *measurable-poisson*[*measurable*]: *poisson* ∈ *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 (∫⁺ *x*. *ennreal* (*r* ^ *nat x* ∗ *exp* (− *r*) ∗ *indicator* ({*0*<..} × {*0*..}) (*r*, *x*) / (*fact* (*nat x*))) ∂*count-space UNIV*)
= (∫⁺ *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.emeasure-space-1*[*of poisson-pmf r*]
by (*subst nn-integral-pmf′*) (*auto simp*: *inj-on-def*)
finally have (∫⁺ *x*. *ennreal* (*r* ^ *nat x* ∗ *exp* (− *r*) ∗ *indicator* ({*0*<..} × {*0*..}) (*r*, *x*) / (*fact* (*nat x*))) ∂*count-space UNIV*) = *1*
. }
then show *integral*^{N} *INTEG* (*poisson-density′ r* ∘ *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 ∪ (λx. 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-vars-expr-code b ∪ *free-vars-expr-code e1* ∪ *free-vars-expr-code e2*
| *free-vars-expr-code* (*Fail* -) = {}

lemma *free-vars-expr-code*[*code*]:
free-vars (*e*::*expr*) = *free-vars-expr-code e*
proof−
have $\bigwedge A.$ *Suc* −' *A* = (λ*x*. *x* − *1*) ' (*A* − {*0*}) **by** *force*
thus *?thesis* **by** (*induction e*) *simp-all*
qed

primrec *dist-param-type* **where**
dist-param-type Bernoulli = *REAL*
| *dist-param-type Poisson* = *REAL*
| *dist-param-type Gaussian* = *PRODUCT REAL REAL*
| *dist-param-type UniformInt* = *PRODUCT INTEG INTEG*
| *dist-param-type UniformReal* = *PRODUCT REAL REAL*

primrec *dist-result-type* **where**
dist-result-type Bernoulli = *BOOL*
| *dist-result-type UniformInt* = *INTEG*
| *dist-result-type UniformReal* = *REAL*
| *dist-result-type Poisson* = *INTEG*
| *dist-result-type Gaussian* = *REAL*

primrec *dist-measure* :: *pdf-dist* ⇒ *val* ⇒ *val measure* **where**
dist-measure Bernoulli = *bernoulli*
| *dist-measure UniformInt* = *uniform-int*
| *dist-measure UniformReal* = *uniform-real*
| *dist-measure Poisson* = *poisson*
| *dist-measure Gaussian* = *gaussian*

lemma *measurable-dist-measure*[*measurable*]:
dist-measure d ∈ *measurable* (*dist-param-type d*) (*subprob-algebra* (*dist-result-type d*))
by (*cases d*) *simp-all*

lemma *sets-dist-measure*[*simp*]:
val-type x = *dist-param-type dst* ⟹
sets (*dist-measure dst x*) = *sets* (*stock-measure* (*dist-result-type dst*))
by (*rule sets-kernel*[*OF measurable-dist-measure*]) *simp*

lemma *space-dist-measure*[*simp*]:
val-type x = *dist-param-type dst* ⟹
space (*dist-measure dst x*) = *type-universe* (*dist-result-type dst*)
by (*subst space-stock-measure*[*symmetric*]) (*intro sets-eq-imp-space-eq sets-dist-measure*)

primrec *dist-dens* :: *pdf-dist* ⇒ *val* ⇒ *val* ⇒ *ennreal* **where**
dist-dens Bernoulli x y = *bernoulli-density* (*extract-real x*) (*extract-bool y*)

| *dist-dens UniformInt x y* = *uniform-int-density* (*extract-int-pair x*) (*extract-int y*)
| *dist-dens UniformReal x y* = *uniform-real-density* (*extract-real-pair x*) (*extract-real y*)
| *dist-dens Gaussian x y* = *gaussian-density* (*extract-real-pair x*) (*extract-real y*)
| *dist-dens Poisson x y* = *poisson-density′* (*extract-real x*) (*extract-int y*)

lemma *measurable-dist-dens*[*measurable*]:
assumes *f* ∈ *measurable M* (*stock-measure* (*dist-param-type dst*)) (**is** - ∈ *measurable M ?N*)
assumes *g* ∈ *measurable M* (*stock-measure* (*dist-result-type dst*)) (**is** - ∈ *measurable M ?R*)
shows (λ*x*. *dist-dens dst* (*f x*) (*g x*)) ∈ *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 ∈ *type-universe* (*dist-param-type dst*) ⟹
has-density (*dist-measure dst v*) (*stock-measure* (*dist-result-type dst*)) (*dist-dens dst v*)
proof (*intro has-densityI*)
fix *v* **assume** *v* ∈ *type-universe* (*dist-param-type dst*)
thus *dist-measure dst v* = *density* (*stock-measure* (*dist-result-type dst*)) (*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 ∈ *type-universe* (*dist-param-type dst*) ⟹ *subprob-space* (*dist-measure dst v*)
using *subprob-space-kernel*[*OF measurable-dist-measure*, *of v dst*] **by** *simp*

lemma *dist-measure-has-subprob-density*:
v ∈ *type-universe* (*dist-param-type dst*) ⟹
has-subprob-density (*dist-measure dst v*) (*stock-measure* (*dist-result-type dst*)) (*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* ∈ *type-universe* (*dist-param-type dst*)
shows (∫ ⁺*u*. *dist-dens dst v u* ∂*stock-measure* (*dist-result-type dst*)) ≤ *1*
proof−
let *?M* = *density* (*stock-measure* (*dist-result-type dst*)) (*dist-dens dst v*)
from *assms* **have** (∫ ⁺*u*. *dist-dens dst v u* ∂*stock-measure* (*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* ... (*space* ...) ≤ *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* ⇒ *pdf-type* ⇒ *pdf-type option* **where**
op-type Add x =
(*case x of*
PRODUCT INTEG INTEG ⇒ *Some INTEG*
| *PRODUCT REAL REAL* ⇒ *Some REAL*
| *-* ⇒ *None*)
| *op-type Mult x* =
(*case x of*
PRODUCT INTEG INTEG ⇒ *Some INTEG*
| *PRODUCT REAL REAL* ⇒ *Some REAL*
| *-* ⇒ *None*)
| *op-type Minus x* =
(*case x of*
INTEG ⇒ *Some INTEG*
| *REAL* ⇒ *Some REAL*
| *-* ⇒ *None*)
| *op-type Equals x* =
(*case x of*
PRODUCT t1 t2 ⇒ *if t1* = *t2 then Some BOOL else None*
| *-* ⇒ *None*)
| *op-type Less x* =
(*case x of*
PRODUCT INTEG INTEG ⇒ *Some BOOL*
| *PRODUCT REAL REAL* ⇒ *Some BOOL*
| *-* ⇒ *None*)
| *op-type* (*Cast t*) *x* =
(*case* (*x*, *t*) *of*
(*BOOL*, *INTEG*) ⇒ *Some INTEG*
| (*BOOL*, *REAL*) ⇒ *Some REAL*
| (*INTEG*, *REAL*) ⇒ *Some REAL*
| (*REAL*, *INTEG*) ⇒ *Some INTEG*
| *-* ⇒ *None*)
| *op-type Or x* = (*case x of PRODUCT BOOL BOOL* ⇒ *Some BOOL* | *-* ⇒ *None*)
| *op-type And x* = (*case x of PRODUCT BOOL BOOL* ⇒ *Some BOOL* | *-* ⇒ *None*)
| *op-type Not x* = (*case x of BOOL* ⇒ *Some BOOL* | *-* ⇒ *None*)
| *op-type Inverse x* = (*case x of REAL* ⇒ *Some REAL* | *-* ⇒ *None*)

| *op-type Fact x* = (*case x of INTEG* ⇒ *Some INTEG* | - ⇒ *None*)
| *op-type Sqrt x* = (*case x of REAL* ⇒ *Some REAL* | - ⇒ *None*)
| *op-type Exp x* = (*case x of REAL* ⇒ *Some REAL* | - ⇒ *None*)
| *op-type Ln x* = (*case x of REAL* ⇒ *Some REAL* | - ⇒ *None*)
| *op-type Pi x* = (*case x of UNIT* ⇒ *Some REAL* | - ⇒ *None*)
| *op-type Pow x* = (*case x of*
      PRODUCT REAL INTEG ⇒ *Some REAL*
    | *PRODUCT INTEG INTEG* ⇒ *Some INTEG*
    | - ⇒ *None*)
| *op-type Fst x* = (*case x of PRODUCT t* - ⇒ *Some t* | - ⇒ *None*)
| *op-type Snd x* = (*case x of PRODUCT* - *t* ⇒ *Some t* | - ⇒ *None*)

5.3 Semantics

abbreviation (*input*) *de-bruijn-insert* (**infixr** · *65*) **where**
  de-bruijn-insert x f ≡ *case-nat x f*

inductive *expr-typing* :: *tyenv* ⇒ *expr* ⇒ *pdf-type* ⇒ *bool* ((*1*-/ ⊢/ (- :/ -)) [*50*,*0*,*50*] *50*) **where**
  et-var: Γ ⊢ *Var x* : Γ *x*
| *et-val*: Γ ⊢ *Val v* : *val-type v*
| *et-let*: Γ ⊢ *e1* : *t1* ⟹ *t1* · Γ ⊢ *e2* : *t2* ⟹ Γ ⊢ *LetVar e1 e2* : *t2*
| *et-op*: Γ ⊢ *e* : *t* ⟹ *op-type oper t* = *Some t′* ⟹ Γ ⊢ *Operator oper e* : *t′*
| *et-pair*: Γ ⊢ *e1* : *t1* ⟹ Γ ⊢ *e2* : *t2* ⟹ Γ ⊢ <*e1*, *e2*> : *PRODUCT t1 t2*
| *et-rand*: Γ ⊢ *e* : *dist-param-type dst* ⟹ Γ ⊢ *Random dst e* : *dist-result-type dst*
| *et-if*: Γ ⊢ *b* : *BOOL* ⟹ Γ ⊢ *e1* : *t* ⟹ Γ ⊢ *e2* : *t* ⟹ Γ ⊢ *IF b THEN e1 ELSE e2* : *t*
| *et-fail*: Γ ⊢ *Fail t* : *t*

lemma *expr-typing-cong′*:
  Γ ⊢ *e* : *t* ⟹ (⋀*x*. *x* ∈ *free-vars e* ⟹ Γ *x* = Γ′ *x*) ⟹ Γ′ ⊢ *e* : *t*
proof (*induction arbitrary*: Γ′ *rule*: *expr-typing.induct*)
  case (*et-let* Γ *e1 t1 e2 t2* Γ′)
  have Γ′ ⊢ *e1* : *t1* **using** *et-let.prems* **by** (*intro et-let.IH*(*1*)) *auto*
  moreover have *case-nat t1* Γ′ ⊢ *e2* : *t2*
    using *et-let.prems* **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*:
  (⋀*x*. *x* ∈ *free-vars e* ⟹ Γ *x* = Γ′ *x*) ⟹ Γ ⊢ *e* : *t* ⟷ Γ′ ⊢ *e* : *t*
  by (*intro iffI*) (*simp-all add*: *expr-typing-cong′*)

inductive-cases *expr-typing-valE*[*elim*]: Γ ⊢ *Val v* : *t*
inductive-cases *expr-typing-varE*[*elim*]: Γ ⊢ *Var x* : *t*
inductive-cases *expr-typing-letE*[*elim*]: Γ ⊢ *LetVar e1 e2* : *t*
inductive-cases *expr-typing-ifE*[*elim*]: Γ ⊢ *IfThenElse b e1 e2* : *t*
inductive-cases *expr-typing-opE*[*elim*]: Γ ⊢ *Operator oper e* : *t*
inductive-cases *expr-typing-pairE*[*elim*]: Γ ⊢ <*e1*, *e2*> : *t*

```
inductive-cases expr-typing-randE[elim]: Γ ⊢ Random dst e : t
inductive-cases expr-typing-failE[elim]: Γ ⊢ Fail t : t'

lemma expr-typing-unique:
  Γ ⊢ e : t ⟹ Γ ⊢ e : t' ⟹ 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 ⇒ expr ⇒ 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 ⇒ expr-type (case-nat t Γ) e2
       | None ⇒ None)
| expr-type Γ (Operator oper e) =
      (case expr-type Γ e of Some t ⇒ op-type oper t | None ⇒ None)
| expr-type Γ (<e1, e2>) =
      (case (expr-type Γ e1, expr-type Γ e2) of
         (Some t1, Some t2) ⇒ Some (PRODUCT t1 t2)
       | - ⇒ 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') ⇒ if t = t' then Some t else None
          | - ⇒ None) else None)
| expr-type Γ (Fail t) = Some t

lemma expr-type-Some-iff: expr-type Γ e = Some t ⟷ Γ ⊢ e : t
  apply rule
  apply (induction e arbitrary: Γ t,
        auto intro!: expr-typing.intros split: option.split-asm if-split-asm) []
  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* ⇒ *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* ∧ *countable-type t2*)

lemma *countable-type-countable*[*dest*]:
countable-type t ⟹ *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 ⟹ *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* ∈ *?M1*
with *assms* **have** *emeasure ?M2 X* = $\int^+$ *x. indicator* {*v*} *x* ∗ *indicator X x* ∂*count-space* (*type-universe* (*val-type v*))
by (*simp add*: *return-val-def emeasure-density countable-type-imp-count-space*)
also have (λ*x. indicator* {*v*} *x* ∗ *indicator X x* :: *ennreal*) = (λ*x. indicator* (*X* ∩ {*v*}) *x*)
by (*rule ext*, *subst Int-commute*) (*simp split*: *split-indicator*)
also have *nn-integral ?M3* ... = *emeasure ?M3* (*X* ∩ {*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 emeasure ?M1 X = emeasure ?M2 X **..**
qed (simp add: return-val-def)

5.4 Semantics

definition bool-to-int :: bool ⇒ int **where**
bool-to-int b = (if b then 1 else 0)

lemma measurable-bool-to-int[measurable]:
bool-to-int ∈ measurable (count-space UNIV) (count-space UNIV)
by (rule measurable-count-space)

definition bool-to-real :: bool ⇒ real **where**
bool-to-real b = (if b then 1 else 0)

lemma measurable-bool-to-real[measurable]:
bool-to-real ∈ borel-measurable (count-space UNIV)
by (rule borel-measurable-count-space)

definition safe-ln :: real ⇒ real **where**
safe-ln x = (if x > 0 then ln x else 0)

lemma safe-ln-gt-0[simp]: x > 0 ⟹ safe-ln x = ln x
by (simp add: safe-ln-def)

lemma borel-measurable-safe-ln[measurable]: safe-ln ∈ borel-measurable borel
unfolding safe-ln-def[abs-def] **by** simp

definition safe-sqrt :: real ⇒ real **where**
safe-sqrt x = (if x ≥ 0 then sqrt x else 0)

lemma safe-sqrt-ge-0[simp]: x ≥ 0 ⟹ safe-sqrt x = sqrt x
by (simp add: safe-sqrt-def)

lemma borel-measurable-safe-sqrt[measurable]: safe-sqrt ∈ borel-measurable borel
unfolding safe-sqrt-def[abs-def] **by** simp

fun op-sem :: pdf-operator ⇒ val ⇒ val **where**
op-sem Add = lift-RealIntVal2 (+) (+)
| op-sem Mult = lift-RealIntVal2 (∗) (∗)
| op-sem Minus = lift-RealIntVal uminus uminus
| op-sem Equals = (λ <|v1, v2|> ⇒ BoolVal (v1 = v2))
| op-sem Less = lift-Comp (<) (<)
| op-sem Or = (λ <|BoolVal a, BoolVal b|> ⇒ BoolVal (a ∨ b))
| op-sem And = (λ <|BoolVal a, BoolVal b|> ⇒ BoolVal (a ∧ b))
| op-sem Not = (λ BoolVal a ⇒ BoolVal (¬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
     }
```

| *expr-sem* σ (*Fail t*) = *null-measure* (*stock-measure t*)

lemma *expr-sem-pair-vars*: *expr-sem* σ <*Var x*, *Var y*> = *return-val* <|σ *x*, σ *y*|>
by (*simp add*: *return-val-def bind-return*[**where** *N*=*PRODUCT* (*val-type* (σ *x*)) (*val-type* (σ *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′* ⟹ *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*:
Γ ⊢ *w* : *t* ⟹ (∀ *x* ∈ *free-vars w*. *val-type* (σ *x*) = Γ *x*) ⟹
sets (*expr-sem* σ *w*) = *sets* (*stock-measure t*)
proof (*induction arbitrary*: σ *rule*: *expr-typing.induct*)
case (*et-var* Γ *x* σ)
thus *?case* **by** (*simp add*: *return-val-def*)
next
case (*et-val* Γ *v* σ)
thus *?case* **by** (*simp add*: *return-val-def*)
next
case (*et-let* Γ *e1 t1 e2 t2* σ)
hence *sets* (*expr-sem* σ *e1*) = *sets* (*stock-measure t1*) **by** *simp*
from *sets-eq-imp-space-eq*[*OF this*]
have *A*: *space* (*expr-sem* σ *e1*) = *type-universe t1* **by** (*simp add*:)
hence *B*: (*SOME x*. *x* ∈ *space* (*expr-sem* σ *e1*)) ∈ *space* (*expr-sem* σ *e1*) (**is** *?v* ∈ -)
unfolding *some-in-eq* **by** *simp*
with *A et-let* **have** *sets* (*expr-sem* (*case-nat ?v* σ) *e2*) = *sets* (*stock-measure t2*)
by (*intro et-let.IH*(*2*)) (*auto split*: *nat.split*)
with *B* **show** *sets* (*expr-sem* σ (*LetVar e1 e2*)) = *sets* (*stock-measure t2*)
by (*subst expr-sem.simps*, *subst bind-nonempty*) *auto*
next
case (*et-op* Γ *e t oper t′* σ)
from *et-op.IH*[*of* σ] **and** *et-op.prems*
have [*simp*]: *sets* (*expr-sem* σ *e*) = *sets* (*stock-measure t*) **by** *simp*
from *sets-eq-imp-space-eq*[*OF this*]
have [*simp*]: *space* (*expr-sem* σ *e*) = *type-universe t* **by** (*simp add*:)
have (*SOME x*. *x* ∈ *space* (*expr-sem* σ *e*)) ∈ *space* (*expr-sem* σ *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* Γ *e1 t1 e2 t2* σ)

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

lemma *space-expr-sem*:
$\Gamma \vdash w : t \Longrightarrow (\forall x \in$ *free-vars w*. *val-type* $(\sigma\ x) = \Gamma\ x)$
$\Longrightarrow$ *space* (*expr-sem* σ *w*) = *type-universe t*
by (*subst space-stock-measure*[*symmetric*]) (*intro sets-expr-sem sets-eq-imp-space-eq*)

lemma *measurable-expr-sem-eq*:
$\Gamma \vdash e : t \Longrightarrow \sigma \in$ *space* (*state-measure V* Γ) $\Longrightarrow$ *free-vars e* $\subseteq V \Longrightarrow$
measurable (*expr-sem* σ *e*) = *measurable* (*stock-measure t*)
by (*intro ext measurable-cong-sets sets-expr-sem*)
(*auto simp*: *state-measure-def space-PiM dest*: *PiE-mem*)

lemma *measurable-expr-semI*:
$\Gamma \vdash e : t \Longrightarrow \sigma \in$ *space* (*state-measure V* Γ) $\Longrightarrow$ *free-vars e* $\subseteq V \Longrightarrow$
$f \in$ *measurable* (*stock-measure t*) $M \Longrightarrow f \in$ *measurable* (*expr-sem* σ *e*) M
by (*subst measurable-expr-sem-eq*)

lemma *expr-sem-eq-on-vars*:
$(\bigwedge x.\ x \in$ *free-vars e* $\Longrightarrow \sigma_1\ x = \sigma_2\ x) \Longrightarrow$ *expr-sem* σ_1 *e* = *expr-sem* σ_2 *e*
proof (*induction e arbitrary*: $\sigma_1\ \sigma_2$)
case (*LetVar e1 e2* $\sigma 1$ $\sigma 2$)

from *LetVar.prems* **have** *A*: *expr-sem σ1 e1 = expr-sem σ2 e1* **by** (*rule LetVar.IH*(*1*)) *simp-all*
from *LetVar.prems* **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 σ1 σ2*)
from *Operator.IH*[*OF Operator.prems*] **show** *?case* **by** *simp*
next
case (*Pair e1 e2 σ1 σ2*)
from *Pair.prems* **have** *expr-sem σ1 e1 = expr-sem σ2 e1* **by** (*intro Pair.IH*) *auto*
moreover from *Pair.prems* **have** *expr-sem σ1 e2 = expr-sem σ2 e2* **by** (*intro Pair.IH*) *auto*
ultimately show *?case* **by** *simp*
next
case (*Random dst e σ1 σ2*)
from *Random.prems* **have** *A*: *expr-sem σ1 e = expr-sem σ2 e* **by** (*rule Random.IH*) *simp-all*
show *?case*
by (*subst* (*1 2*) *expr-sem.simps*, *subst A*) (*auto intro*!: *bind-cong*)
next
case (*IfThenElse b e1 e2 σ1 σ2*)
have *A*: *expr-sem σ1 b = expr-sem σ2 b*
expr-sem σ1 e1 = expr-sem σ2 e1
expr-sem σ1 e2 = expr-sem σ2 e2
using *IfThenElse.prems* **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* ∈ *borel-measurable M g* ∈ *borel-measurable M*
shows *Measurable.pred M* (λ*x*. *f x* = (*g x*::*real*))
proof −
have ∗: (λ*x*. *f x* = *g x*) = (λ*x*. *f x* − *g x* = *0*)
by *simp*
show *?thesis*
unfolding ∗ **by** *measurable*
qed

lemma *measurable-equals*:
(λ(*x*,*y*). *x* = *y*) ∈ *measurable* (*stock-measure t* $\bigotimes_M$ *stock-measure t*) (*count-space UNIV*)
proof (*induction t*)
case *REAL*
let *?f* = λ*x*. *extract-real* (*fst x*) = *extract-real* (*snd x*)
show *?case*

proof (*subst measurable-cong*)
fix *x* **assume** $x \in$ *space* (*stock-measure REAL* $\bigotimes_M$ *stock-measure REAL*)
thus (λ(*x*,*y*). *x* = *y*) *x* = *?f x*
by (*auto simp*: *space-pair-measure elim*!: *REAL-E*)
next
show *?f* $\in$ *measurable* (*stock-measure REAL* $\bigotimes_M$ *stock-measure REAL*) (*count-space UNIV*)
by *measurable*
qed
next
case (*PRODUCT t1 t2*)
let *?g* = λ(*x*,*y*). *x* = *y*
let *?f* = λ*x*. *?g* (*fst* (*extract-pair* (*fst x*)), *fst* (*extract-pair* (*snd x*))) $\wedge$
?g (*snd* (*extract-pair* (*fst x*)), *snd* (*extract-pair* (*snd x*)))
show *?case*
proof (*subst measurable-cong*)
fix *x* **assume** $x \in$ *space* (*stock-measure* (*PRODUCT t1 t2*) $\bigotimes_M$ *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*) $\bigotimes_M$ *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 $f \in$ *measurable M* (*stock-measure t*) $g \in$ *measurable M* (*stock-measure t*)
shows *Measurable.pred M* (λ*x*. *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* $\in$ *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=λ(*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* ⇒ *vname set* **where**
shift-var-set V = *insert 0* (*Suc* ' *V*)

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

lemma *shift-var-set-Suc*[*simp*]: *Suc x* ∈ *shift-var-set V* ⟷ *x* ∈ *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*) ∘ *case-nat 0* (λ*x*. *Suc* (*Suc x*)) = *case-nat x σ*
by (*intro ext*) (*simp split*: *nat.split*)

lemma *delete-var-1-vimage*[*simp*]:
case-nat 0 (λ*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* ∈ *measurable R N h* ∈ *measurable R* (Pi_M *V M*)
shows (λ*x*. *case-nat* (*g x*) (*h x*)) ∈ *measurable R* (Pi_M (*shift-var-set V*) (*case-nat N M*))
proof (*rule measurable-Pair-compose-split*[*OF - assms*])
have (λ(*t*,*f*). λ*x*∈*shift-var-set V*. *case-nat t f x*)
∈ *measurable* ($N \bigotimes_M$ *PiM V M*) (*PiM* (*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 ⋀*x f*. *f* ∈ *space* (*PiM V M*) ⟹ *x* ∉ *V* ⟹ *undefined* = *f x*
by (*rule sym*, *subst* (*asm*) *space-PiM*, *erule PiE-arb*)
hence *?P* ⟷ (λ(*t*,*f*). *case-nat t f*)
∈ *measurable* ($N \bigotimes_M$ *PiM V M*) (*PiM* (*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 ∈ measurable R (stock-measure t) h ∈ measurable R (state-measure
V Γ)
  shows (λx. case-nat (g x) (h x)) ∈
          measurable R (state-measure (shift-var-set V) (case-nat t Γ))
proof−
  have A: (λx. stock-measure (case-nat t Γ x)) =
              case-nat (stock-measure t) (λx. stock-measure (Γ 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 ∈ type-universe t1 σ ∈ space (state-measure V Γ)
  shows case-nat x σ ∈ space (state-measure (shift-var-set V) (case-nat t1 Γ))
  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:
    Suc −‘ A ⊆ V ⟹ A ⊆ 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 Γ ⊢ e : t and free-vars e ⊆ V
  shows (λσ. expr-sem σ e) ∈ measurable (state-measure V Γ)
                                      (subprob-algebra (stock-measure t))
using assms
proof (induction arbitrary: V rule: expr-typing.induct)
  case (et-var Γ x)
  have A: (λσ. expr-sem σ (Var x)) = return-val ∘ (λσ. σ 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 Γ v)
  thus ?case by (auto intro!: measurable-const subprob-space-return
                  simp: space-subprob-algebra return-val-def)
next
  case (et-let Γ e1 t1 e2 t2 V)
    have A: (λv. stock-measure (case-nat t1 Γ v)) =
              case-nat (stock-measure t1) (λv. stock-measure (Γ 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* Γ *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* Γ *e1 e2*)
have *inj* (λ(*a*,*b*). <|*a*, *b*|>) **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* Γ *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* Γ *b e1 t e2 V*)
let *?M* = λ*e t*. (λσ. *expr-sem* σ *e*) ∈
measurable (*state-measure V* Γ) (*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* Γ *t V*)
show *?case*
by (*auto intro*!: *measurable-subprob-algebra subprob-spaceI simp*:)
qed

5.6 Randomfree expressions

fun *randomfree* :: *expr* ⇒ *bool* **where**
randomfree (*Val* -) = *True*
| *randomfree* (*Var* -) = *True*
| *randomfree* (*Pair e1 e2*) = (*randomfree e1* ∧ *randomfree e2*)
| *randomfree* (*Operator - e*) = *randomfree e*
| *randomfree* (*LetVar e1 e2*) = (*randomfree e1* ∧ *randomfree e2*)
| *randomfree* (*IfThenElse b e1 e2*) = (*randomfree b* ∧ *randomfree e1* ∧ *randomfree*

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

primrec *expr-sem-rf* :: *state* ⇒ *expr* ⇒ *val* **where**
expr-sem-rf - (*Val v*) = *v*
| *expr-sem-rf* σ (*Var x*) = σ *x*
| *expr-sem-rf* σ (<*e1*, *e2*>) = <|*expr-sem-rf* σ *e1*, *expr-sem-rf* σ *e2*|>
| *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* · 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*]:
Γ ⊢ *e* : *t* ⟹ *randomfree e* ⟹ *free-vars e* ⊆ *V* ⟹
(λσ. *expr-sem-rf* σ *e*) ∈ *measurable* (*state-measure V* Γ) (*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* (λ(*x*,*y*). <|*x*, *y*|>) **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*: (λσ. *expr-sem-rf* σ *e1*) ∈ *measurable* (*state-measure V* Γ) (*stock-measure*
t1)
and *M2*: (λσ. *expr-sem-rf* σ *e2*) ∈ *measurable* (*state-measure* (*shift-var-set V*)
(*case-nat t1* Γ))
(*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 (λσ. *expr-sem-rf* σ (*LetVar e1 e2*)) =
(λσ. *expr-sem-rf* σ *e2*) ∘ (λ(σ,*y*). *case-nat y* σ) ∘ (λσ. (σ, *expr-sem-rf*
σ *e1*)) (**is** - = *?f*)
by (*intro ext*) *simp*
also have *?f* ∈ *measurable* (*state-measure V* Γ) (*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*:
Γ ⊢ *e* : *t* ⟹ *randomfree e* ⟹ *free-vars e* ⊆ *V* ⟹ σ ∈ *space* (*state-measure V* Γ) ⟹
return-val (*expr-sem-rf* σ *e*) = *expr-sem* σ *e*
proof (*induction arbitrary*: *V* σ *rule*: *expr-typing.induct*)
case (*et-val* Γ *v*)
thus *?case* **by** *simp*
next
case (*et-var* Γ *x*)
thus *?case* **by** *simp*
next
case (*et-pair* Γ *e1 t1 e2 t2 V* σ)
let *?M* = *state-measure V* Γ
from *et-pair.hyps* **and** *et-pair.prems*
have *e1*: *return-val* (*expr-sem-rf* σ *e1*) = *expr-sem* σ *e1* **and**
e2: *return-val* (*expr-sem-rf* σ *e2*) = *expr-sem* σ *e2*
by (*auto intro*!: *et-pair.IH*[*of V*])

from *e1* **and** *et-pair.prems* **have** *space* (*return-val* (*expr-sem-rf* σ *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* σ *e1*) = *t1 expr-sem-rf* σ *e1* ∈ *type-universe t1*
by (*auto simp add*: *return-val-def*)
from *e2* **and** *et-pair.prems* **have** *space* (*return-val* (*expr-sem-rf* σ *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* σ *e2*) = *t2 expr-sem-rf* σ *e2* ∈ *type-universe t2*
by (*auto simp add*: *return-val-def*)

have *expr-sem* σ (<*e1*, *e2*>) = *expr-sem* σ *e1* ⤜
(λ*v*. *expr-sem* σ *e2* ⤜ (λ*w*. *return-val* (<|*v*,*w*|>))) **by** *simp*
also have *expr-sem* σ *e1* = *return* (*stock-measure t1*) (*expr-sem-rf* σ *e1*)
using *e1* **by** (*simp add*: *et-pair.prems return-val-def A*)
also have ... ⤜ (λ*v*. *expr-sem* σ *e2* ⤜ (λ*w*. *return-val* (<|*v*,*w*|>))) =
... ⤜ (λ*v*. *return-val* (<|*v*, *expr-sem-rf* σ *e2*|>))
proof (*intro bind-cong refl*)
fix *v* **assume** *v* ∈ *space* (*return* (*stock-measure t1*) (*expr-sem-rf* σ *e1*))
hence *v*: *val-type v* = *t1 v* ∈ *type-universe t1* **by** (*simp-all add*:)
have *expr-sem* σ *e2* ⤜ (λ*w*. *return-val* (<|*v*,*w*|>)) =
return (*stock-measure t2*) (*expr-sem-rf* σ *e2*) ⤜ (λ*w*. *return-val* (<|*v*,*w*|>))

```
    using e2 by (simp add: et-pair.prems 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.prems have A: ?P b ?P e1 ?P e2 by ((intro et-if.IH[symmetric],
simp-all) [])+
  from et-if.prems 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.prems
  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.prems have e: return-val (expr-sem-rf σ e) = expr-sem σ e
  by (intro et-op.IH[of V]) auto
```

with *et-op.prems* **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.prems e*
have *expr-sem* σ (*Operator oper e*) =
return-val (*expr-sem-rf* σ *e*) $\gg=$ (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* $\Longrightarrow$ *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*) $\gg=$ (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*) $\gg=$ (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 ‹$\sigma \in$ *space ?M*› **by** *simp*
have *return-val* (*expr-sem-rf* σ *e1*) $\gg=$ (λv. *return-val* (*expr-sem-rf* (*case-nat v* σ) *e2*)) =
return-val (*expr-sem-rf ?*σ' *e2*) **using** ‹$\sigma \in$ *space ?M*›

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* $\sigma \in$ *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*$\in$*free-vars e* $\Longrightarrow$ $\sigma 1$ *x* = $\sigma 2$ *x*) $\Longrightarrow$ *expr-sem-rf* $\sigma 1$ *e* = *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* = *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*: *expr-sem-rf* $\sigma 1$ *e1* = *expr-sem-rf* $\sigma 2$ *e1* **by** (*intro LetVar.IH*) *auto*
{
fix *y* **assume** *y* $\in$ *free-vars e2*
hence *case-nat* (*expr-sem-rf* $\sigma 1$ *e1*) $\sigma 1$ *y* = *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* = *expr-sem-rf* $\sigma 2$ *e1 expr-sem-rf* $\sigma 1$ *e2* = *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* = *expr-sem-rf* $\sigma 2$ *b expr-sem-rf* $\sigma 1$ *e1* = *expr-sem-rf* $\sigma 2$ *e1*
expr-sem-rf $\sigma 1$ *e2* = *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*)]:
$i \in V \Longrightarrow (\lambda x.\ x\ i) \in measurable\ (state\text{-}measure\ V\ \Gamma)\ (\Gamma\ i)$
unfolding *state-measure-def* **by** *measurable*

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

lemma *measurable-case-nat-Suc-PiM*:
$(\lambda\sigma.\ \sigma \circ Suc) \in measurable\ (PiM\ (Suc\ `\ A)\ (case\text{-}nat\ M\ N))\ (PiM\ A\ N)$
proof−
have $(\lambda\sigma.\ \lambda x{\in}A.\ \sigma\ (Suc\ x)) \in measurable$
$(PiM\ (Suc\ `\ A)\ (case\text{-}nat\ M\ N))\ (PiM\ A\ (\lambda x.\ case\text{-}nat\ M\ N\ (Suc\ x)))$ (**is** *?A*)
by *measurable*
also have *?A* ⟷ *?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*:
$(\lambda\sigma.\ \sigma \circ Suc) \in measurable\ (state\text{-}measure\ (Suc\ `\ A)\ (case\text{-}nat\ t\ \Gamma))\ (state\text{-}measure\ A\ \Gamma)$
proof−
have $(\lambda\sigma.\ \lambda x{\in}A.\ \sigma\ (Suc\ x)) \in measurable$
$(state\text{-}measure\ (Suc\ `\ A)\ (case\text{-}nat\ t\ \Gamma))\ (state\text{-}measure\ A\ (\lambda i.\ case\text{-}nat\ t\ \Gamma$

(*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 $\sigma \in (\Pi_E\ x{\in}V.\ \textit{type-universe}\ (\Gamma\ x))$ and returns the common density of these variables.

type-synonym *dens-ctxt* = *vname set* × *vname set* × (*vname* ⇒ *pdf-type*) × (*state* ⇒ *ennreal*)
type-synonym *expr-density* = *state* ⇒ *val* ⇒ *ennreal*

definition *empty-dens-ctxt* :: *dens-ctxt* **where**
empty-dens-ctxt = ({}, {}, λ-. *undefined*, λ-. *1*)

definition *state-measure′*
:: *vname set* ⇒ *vname set* ⇒ (*vname* ⇒ *pdf-type*) ⇒ *state* ⇒ *state measure*
where
state-measure′ V V′ Γ ϱ =
distr (*state-measure V* Γ) (*state-measure* (*V*∪*V′*) Γ) (λσ. *merge V V′* (σ, ϱ))

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* ⇒ *vname* ⇒ *expr-density* **where**
marg-dens = (λ(*V*,*V′*,Γ,δ) *x* ϱ *v*. $\int^{+}$σ. δ (*merge V V′* (σ(*x* := *v*), ϱ)) ∂*state-measure* (*V*−{*x*}) Γ)

definition *marg-dens2* :: *dens-ctxt* ⇒ *vname* ⇒ *vname* ⇒ *expr-density* **where**
marg-dens2 ≡ (λ(*V*,*V′*,Γ,δ) *x y* ϱ *v*.
$\int^{+}$σ. δ (*merge V V′* (σ(*x* := *fst* (*extract-pair v*), *y* := *snd* (*extract-pair v*)), ϱ))
∂*state-measure* (*V*−{*x*,*y*}) Γ)

definition *dens-ctxt-measure* :: *dens-ctxt* ⇒ *state* ⇒ *state measure* **where**
dens-ctxt-measure ≡ λ(*V*,*V′*,Γ,δ) ϱ. *density* (*state-measure′ V V′* Γ ϱ) δ

definition *branch-prob* :: *dens-ctxt* ⇒ *state* ⇒ *ennreal* **where**
branch-prob 𝒴 ϱ = *emeasure* (*dens-ctxt-measure* 𝒴 ϱ) (*space* (*dens-ctxt-measure* 𝒴 ϱ))

lemma *dens-ctxt-measure-nonempty*[*simp*]:
space (*dens-ctxt-measure* 𝒴 ϱ) ≠ {}
unfolding *dens-ctxt-measure-def state-measure′-def* **by** (*cases* 𝒴) *simp*

lemma *sets-dens-ctxt-measure-eq*[*measurable-cong*]:
sets (*dens-ctxt-measure* (V, V',Γ,δ) ϱ) = *sets* (*state-measure* ($V \cup V'$) 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* ($V \cup V'$) 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* ($V \cup V'$) 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 = ($\lambda\varrho$ *y*. $\int^{+}$*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 $\in$ *measurable* (*state-measure* (*shift-var-set* V) (*case-nat t* Γ)) (*state-measure* V Γ)
proof−
have ($\lambda\sigma$. $\lambda x{\in}V$. σ (*Suc x*)) $\in$ *measurable*
(*state-measure* (*shift-var-set* V) (*case-nat t* Γ)) (*state-measure* V (λx. *case-nat t* Γ (*Suc x*)))
(**is** *?f* $\in$ *?M*)
unfolding *state-measure-def shift-var-set-def* **by** *measurable*
also have $\bigwedge x\, f$. $x \notin V \Longrightarrow f \in$ *space* (*state-measure* (*shift-var-set* V) (*case-nat t* Γ)) $\Longrightarrow$
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* $\longleftrightarrow$ *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 $\in$ *measurable* (*state-measure* A Γ) (*state-measure* (*Suc*‘A) (*case-nat t* Γ)) (**is** - $\in$ *?M*)
proof−
have ($\lambda\sigma$. $\lambda x{\in}$*Suc*‘A. *case-nat undefined* σ *x*) $\in$ *?M* (**is** *?f* $\in$ -)
unfolding *state-measure-def* **by** (*rule measurable-restrict*) *auto*
also have *?f* $\in$ *?M* $\longleftrightarrow$ *?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* ⇒ *vname set* ⇒ *expr-density* ⇒ (*state* ⇒ *ennreal*) ⇒ *state* ⇒ *ennreal* **where**
  insert-dens V V′ f δ ≡ *λσ. δ* (*remove-var σ*) ∗ *f* (*remove-var σ*) (*σ 0*)

definition *if-dens* :: (*state* ⇒ *ennreal*) ⇒ (*state* ⇒ *val* ⇒ *ennreal*) ⇒ *bool* ⇒ (*state* ⇒ *ennreal*) **where**
  if-dens δ f b ≡ *λσ. δ σ* ∗ *f σ* (*BoolVal b*)

definition *if-dens-det* :: (*state* ⇒ *ennreal*) ⇒ *expr* ⇒ *bool* ⇒ (*state* ⇒ *ennreal*) **where**
  if-dens-det δ e b ≡ *λσ. δ σ* ∗ (*if expr-sem-rf σ e* = *BoolVal b then 1 else 0*)

lemma *measurable-if-dens*:
  assumes [*measurable*]: *δ* ∈ *borel-measurable M*
  assumes [*measurable*]: *case-prod f* ∈ *borel-measurable* (*M* ⨂$_M$ *count-space* (*range BoolVal*))
  shows *if-dens δ f b* ∈ *borel-measurable M*
  unfolding *if-dens-def* **by** *measurable*

lemma *measurable-if-dens-det*:
  assumes *e*: Γ ⊢ *e* : *BOOL randomfree e free-vars e* ⊆ *V*
  assumes [*measurable*]: *δ* ∈ *borel-measurable* (*state-measure V* Γ)
  shows *if-dens-det δ e b* ∈ *borel-measurable* (*state-measure V* Γ)
unfolding *if-dens-det-def*
proof (*intro borel-measurable-times-ennreal assms measurable-If*)
  have {*x* ∈ *space* (*state-measure V* Γ). *expr-sem-rf x e* = *BoolVal b*} =
      (*λσ. expr-sem-rf σ e*) −‘ {*BoolVal b*} ∩ *space* (*state-measure V* Γ) **by** *auto*
  also have ... ∈ *sets* (*state-measure V* Γ)
    by (*rule measurable-sets*, *rule measurable-expr-sem-rf*[*OF e*]) *simp-all*
  finally show {*x* ∈ *space* (*state-measure V* Γ). *expr-sem-rf x e* = *BoolVal b*}
      ∈ *sets* (*state-measure V* Γ) **.**
qed *simp-all*

locale *density-context* =
  fixes *V V′* Γ *δ*
  assumes *subprob-space-dens*:
      ⋀*ϱ*. *ϱ* ∈ *space* (*state-measure V′* Γ) ⟹ *subprob-space* (*dens-ctxt-measure* (*V*,*V′*,Γ,*δ*) *ϱ*)
    and *finite-vars*[*simp*]:   *finite V finite V′*
    and *measurable-dens*[*measurable*]:
        δ ∈ *borel-measurable* (*state-measure* (*V* ∪ *V′*) Γ)
    and *disjoint*:   *V* ∩ *V′* = {}

begin

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

lemma *branch-prob-altdef*:
  assumes ϱ: $\varrho \in$ *space* (*state-measure* V' Γ)
  shows *branch-prob* $\mathcal{Y}$ $\varrho = \int^{+} x.\ \delta$ (*merge* V V' $(x,\ \varrho)$) ∂*state-measure* V Γ
proof−
  have *branch-prob* $\mathcal{Y}$ ϱ =
    $\int^{+} x.\ \delta$ (*merge* V V' $(x,\ \varrho)$) $*$ *indicator* (*space* (*state-measure* $(V \cup V')$ Γ))
      (*merge* V V' $(x,\ \varrho)$) ∂*state-measure* V Γ
    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** ... $= \int^{+} x.\ \delta$ (*merge* V V' $(x,\ \varrho)$) ∂*state-measure* V Γ
   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 $\mathcal{Y} \in$ *borel-measurable* (*state-measure* V' Γ)
proof−
  interpret *sigma-finite-measure state-measure* V Γ **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) $\in$ *borel-measurable* (*state-measure* V' Γ $\bigotimes_M$ *stock-measure* $(\Gamma\ x)$)
proof−
  interpret *sigma-finite-measure state-measure* $(V - \{x\})$ Γ
    unfolding *state-measure-def*
   by (*rule product-sigma-finite.sigma-finite*, *simp-all add*: *product-sigma-finite-def*)
  from *assms* **have** $V =$ *insert* x $(V - \{x\})$ **by** *blast*
  hence A: *PiM* V = *PiM* ... **by** *simp*
  show *?thesis* **unfolding** *marg-dens-def*
    by (*simp add*: *insert-absorb*)
qed

lemma *insert-Diff*: *insert* x $(A - B)$ = *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) $\in$
    borel-measurable (*state-measure* V' Γ $\bigotimes_M$ *stock-measure* (*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 = $ *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 $x \in V\ \varrho \in$ *space* (*state-measure* $V'\ \Gamma$)
  shows *marg-dens* $\mathcal{Y}\ x\ \varrho \in$ *borel-measurable* (*stock-measure* $(\Gamma\ x)$)
 using *assms* **by** (*intro measurable-Pair-compose-split*[*OF measurable-marg-dens'*])
simp-all

lemma *measurable-marg-dens2*:
  assumes $x \in V\ y \in V\ x \neq y\ \varrho \in$ *space* (*state-measure* $V'\ \Gamma$)
  shows *marg-dens2* $\mathcal{Y}\ x\ y\ \varrho \in$ *borel-measurable* (*stock-measure* (*PRODUCT* $(\Gamma$
$x)\ (\Gamma\ y))$)
 using *assms* **by** (*intro measurable-Pair-compose-split*[*OF measurable-marg-dens2'*])
simp-all

lemma *measurable-state-measure-component*:
    $x \in V \Longrightarrow (\lambda\sigma.\ \sigma\ x) \in$ *measurable* (*state-measure* $V\ \Gamma$) (*stock-measure* $(\Gamma\ x)$)
  unfolding *state-measure-def*
  by (*auto intro*!: *measurable-component-singleton*)

lemma *measurable-dens-ctxt-measure-component*:
   $x \in V \Longrightarrow (\lambda\sigma.\ \sigma\ x) \in$ *measurable* (*dens-ctxt-measure* $(V, V', \Gamma, \delta)\ \varrho$) (*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 *space* (*state-measure* $V\ \Gamma$) =
          $\{\sigma(x := y)\ |\sigma\ y.\ \sigma \in$ *space* (*state-measure* $(V - \{x\})\ \Gamma) \wedge y \in$ *type-universe*
$(\Gamma\ x)\}$
proof −
  from *assms* **have** *insert* $x\ (V - \{x\}) = V$ **by** *auto*
  hence *state-measure* $V\ \Gamma = Pi_M$ (*insert* $x\ (V - \{x\})$) ($\lambda y.$ *stock-measure* $(\Gamma\ y)$)
    unfolding *state-measure-def* **by** *simp*
  also have *space* ... $= \{\sigma(x := y)\ |\sigma\ y.\ \sigma \in$ *space* (*state-measure* $(V - \{x\})\ \Gamma) \wedge$
$y \in$ *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 Γ)
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* $\lambda y.$ *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* $A\ \Gamma) = (\int^+\sigma.\ f\ \sigma\ \partial\Pi_M\ v{\in}$*insert* $x\ (A-\{x\})$.
stock-measure $(\Gamma\ v))$
unfolding *state-measure-def* **by** *simp*
also have ... $= \int^+ y.\ \int^+\sigma.\ f\ (\sigma(x := y))\ \partial$*state-measure* $(A-\{x\})\ \Gamma\ \partial$*stock-measure*
$(\Gamma\ x)$
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* $(\Gamma\ x))]\!]$
$\Longrightarrow \sigma(x := y) \in$ *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 :: *val set* **assumes** $x \in V$ **and** [*measurable*]: $X \in$ *sets* (*stock-measure* $(\Gamma$
$x))$
assumes $\varrho \in$ *space* (*state-measure* $V'\ \Gamma$)
defines $X' \equiv (\lambda\sigma.\ \sigma\ x)$ −‘ $X \cap$ *space* (*state-measure* $V\ \Gamma$)
shows $(\int^+ y.$ *marg-dens* $\mathcal{Y}\ x\ \varrho\ y *$ *indicator* $X\ y\ \partial$*stock-measure* $(\Gamma\ x)) =$
$(\int^+\sigma.\ \delta$ (*merge* $V\ V'\ (\sigma,\varrho)) *$ *indicator* $X'\ \sigma\ \partial$*state-measure* $V\ \Gamma)$
proof−
from *assms* **have** [*simp*]: *insert* x $V = V$ **by** *auto*
interpret *product-sigma-finite* $\lambda y.$ *stock-measure* $(\Gamma\ y)$
unfolding *product-sigma-finite-def* **by** *auto*

have $(\int^+\sigma.\ \delta$ (*merge* $V\ V'\ (\sigma,\varrho)) *$ *indicator* $X'\ \sigma\ \partial$*state-measure* $V\ \Gamma) =$
$\int^+ y.\ \int^+ \sigma.\ \delta$ (*merge* $V\ V'\ (\sigma(x := y),\ \varrho)) *$ *indicator* $X'\ (\sigma(x := y))$
∂*state-measure* $(V-\{x\})\ \Gamma\ \partial$*stock-measure* $(\Gamma\ x)$ **using** *assms*(*1*−*3*)
by (*subst state-measure-integral-split*[*of x*]) (*auto simp*: X'-*def*)
also have ... $= \int^+ y.\ \int^+ \sigma.\ \delta$ (*merge* $V\ V'\ (\sigma(x := y),\ \varrho)) *$ *indicator* $X\ y$
∂*state-measure* $(V-\{x\})\ \Gamma\ \partial$*stock-measure* $(\Gamma\ x)$
by (*intro nn-integral-cong*)
(*auto simp*: X'-*def split*: *split-indicator dest*: *fun-upd-in-state-measure*)
also have ... $= (\int^+ y.$ *marg-dens* $\mathcal{Y}\ x\ \varrho\ y *$ *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 :: *val set*
assumes $x \in V$ $y \in V$ $x \neq y$ **and** [*measurable*]: $X \in$ *sets* (*stock-measure* (*PRODUCT* (Γ x) (Γ y)))
assumes $\varrho \in$ *space* (*state-measure* V' Γ)
defines $X' \equiv (\lambda\sigma.$ <|σ x, σ y|>) $-$' $X \cap$ *space* (*state-measure* V Γ)
shows ($\int^+ z$. *marg-dens2* $\mathcal{Y}$ x y ϱ z $*$ *indicator* X z ∂*stock-measure* (*PRODUCT* (Γ x) (Γ y))) =
($\int^+\sigma$. δ (*merge* V V' (σ,ϱ)) $*$ *indicator* X' σ ∂*state-measure* V Γ)
proof−
let *?M* = *stock-measure* (*PRODUCT* (Γ x) (Γ y))
let *?M'* = *stock-measure* (Γ x) $\bigotimes_M$ *stock-measure* (Γ y)
interpret *product-sigma-finite* λx. *stock-measure* (Γ x)
unfolding *product-sigma-finite-def* **by** *simp*
from *assms* **have** ($\int^+ z$. *marg-dens2* $\mathcal{Y}$ x y ϱ z $*$ *indicator* X z ∂*?M*) =
$\int^+ z$. *marg-dens2* $\mathcal{Y}$ x y ϱ (*case-prod PairVal* z) $*$ *indicator* X (*case-prod PairVal* z) ∂*?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 = *insert* y ($V-\{y\}$) **by** *blast*
from *assms* **have** B: *insert* x ($V-\{x,y\}$) = $V - \{y\}$ **by** *blast*
from *assms* **have** X'[*measurable*]: $X' \in$ *sets* (*state-measure* V Γ) **unfolding** X'*-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[*simp*]: *insert* y ($V - \{y\}$) = V *insert* x ($V - \{x, y\}$) = $V - \{y\}$ *insert* y V = V
and [*measurable*]: $x \in V - \{y\}$
using *assms* **by** *auto*

have ($\int^+\sigma$. δ (*merge* V V' (σ,ϱ)) $*$ *indicator* X' σ ∂*state-measure* V Γ) =
($\int^+\sigma$. δ (*merge* V V' (σ,ϱ)) $*$ *indicator* X' σ ∂*state-measure* (*insert* y (*insert* x ($V-\{x, y\}$))) Γ)
using *assms* **by** (*intro arg-cong2*[**where** *f=nn-integral*] *arg-cong2*[**where** *f=state-measure*]) *auto*
also have ... = $\int^+ w$. $\int^+ v$. $\int^+\sigma$. δ (*merge* V V' (σ(x := v, y := w), ϱ)) $*$ *indicator* X' (σ(x := v, y := w))
∂*state-measure* ($V - \{x, y\}$) Γ ∂*stock-measure* (Γ x) ∂*stock-measure* (Γ 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 $\bigwedge v\ w\ \sigma.\ v \in$ *space* (*stock-measure* (Γ *x*)) $\Longrightarrow$ $w \in$ *space* (*stock-measure* (Γ *y*))
$\Longrightarrow \sigma \in$ *space* (*state-measure* (*V*−{*x*,*y*}) Γ)
$\Longrightarrow \sigma(x := v,\ y := w) \in X' \longleftrightarrow$ <|*v*,*w*|> $\in X$
by (*simp add*: *X′-def space-state-measure PiE-iff extensional-def*)
hence *?I* = $\int^+ w.\ \int^+ v.\ \int^+ \sigma.\ \delta$ (*merge V V′* (σ(*x* := *v*, *y* := *w*), ϱ)) ∗ *indicator* *X* <|*v*,*w*|>
∂*state-measure* (*V* − {*x*,*y*}) Γ ∂*stock-measure* (Γ *x*) ∂*stock-measure* (Γ *y*)
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also from *assms*(*5*)
have ... = $\int^+ w.\ \int^+ v.$ ($\int^+ \sigma.\ \delta$ (*merge V V′* (σ(*x* := *v*,*y* := *w*), ϱ)) ∂*state-measure* (*V* − {*x*,*y*}) Γ)
∗ *indicator X* <|*v*,*w*|> ∂*stock-measure* (Γ *x*) ∂*stock-measure* (Γ *y*)
using *assms*
apply (*simp add*: *ennreal-mult″ ennreal-indicator*)
by (*intro nn-integral-cong nn-integral-multc*) (*simp-all add*:)
also have ... = $\int^+ w.\ \int^+ v.$ *marg-dens2* $\mathcal{Y}$ *x y* ϱ <|*v*,*w*|> ∗ *indicator X* <|*v*,*w*|>
∂*stock-measure* (Γ *x*) ∂*stock-measure* (Γ *y*)
by (*intro nn-integral-cong*) (*simp add*: *marg-dens2-def*)
also from *assms*(*4*)
have ... = $\int^+ z.$ *marg-dens2* $\mathcal{Y}$ *x y* ϱ (*case-prod PairVal z*) ∗ *indicator X* (*case-prod PairVal z*)
∂(*stock-measure* (Γ *x*) $\bigotimes_M$ *stock-measure* (Γ *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 ... = $\int^+ z.$ *marg-dens2* $\mathcal{Y}$ *x y* ϱ *z* ∗ *indicator X z* ∂*stock-measure* (*PRODUCT* (Γ *x*) (Γ *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 $x \in V$ $\varrho \in$ *space* (*state-measure* V' Γ)
shows *density* (*stock-measure* (Γ x)) (*marg-dens* $\mathcal{Y}$ x ϱ) =
distr (*dens-ctxt-measure* (V,V',Γ,δ) ϱ) (*stock-measure* (Γ x)) ($\lambda\sigma$. σ x)
(**is** *?M1* = *?M2*)
proof (*rule measure-eqI*)
fix X **assume** X: $X \in$ *sets ?M1*
let *?X′* = ($\lambda\sigma$. σ x) −‘ X ∩ *space* (*state-measure* V Γ)
let *?X″* = ($\lambda\sigma$. σ x) −‘ X ∩ *space* (*state-measure* ($V \cup V'$) Γ)
from X **have** *emeasure ?M1* X = $\int^+$ σ. δ (*merge* V V' (σ, ϱ)) ∗ *indicator ?X′* σ ∂*state-measure* V Γ
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** ... = $\int^+$ σ. δ (*merge* V V' (σ, ϱ)) ∗
indicator ?X″ (*merge* V V' (σ,ϱ)) ∂*state-measure* V Γ
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** ... = *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 *emeasure ?M1* X = *emeasure ?M2* X **.**
qed *simp*

lemma *density-marg-dens2-eq*:
assumes $x \in V$ $y \in V$ $x \neq y$ $\varrho \in$ *space* (*state-measure* V' Γ)
defines $M \equiv$ *stock-measure* (*PRODUCT* (Γ x) (Γ y))
shows *density* M (*marg-dens2* $\mathcal{Y}$ x y ϱ) =
distr (*dens-ctxt-measure* (V,V',Γ,δ) ϱ) M ($\lambda\sigma$. <|σ x,σ y|>) (**is** *?M1* = *?M2*)
proof (*rule measure-eqI*)
fix X **assume** X: $X \in$ *sets ?M1*
let *?X′* = ($\lambda\sigma$. <|σ x , σ y|>) −‘ X ∩ *space* (*state-measure* V Γ)
let *?X″* = ($\lambda\sigma$. <|σ x , σ y|>) −‘ X ∩ *space* (*state-measure* ($V \cup V'$) Γ)

from *assms* **have** *meas*[*measurable*]: (λσ. <|σ *x*,σ *y*|>) ∈ *measurable* (*state-measure* (*V* ∪ *V′*) Γ)
(*stock-measure* (*PRODUCT* (Γ *x*) (Γ *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* 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 ... = ∫⁺σ. δ (*merge V V′* (σ, ϱ)) ∗ *indicator ?X″* (*merge V V′* (σ, ϱ)) ∂*state-measure V* Γ
(**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 ⋀σ. σ∈*space* (*state-measure V* Γ) ⟹ *merge V V′* (σ, ϱ) ∈ *?X″* ⟷ σ ∈ *?X′*
by (*auto simp*: *space-state-measure merge-def PiE-iff extensional-def*)
hence *?I* = ∫⁺σ. δ (*merge V V′* (σ, ϱ)) ∗ *indicator ?X′* σ ∂*state-measure V* Γ
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also from *assms X* **have** ... = ∫⁺*z*. *marg-dens2* 𝒴 *x y* ϱ *z* ∗ *indicator X z* ∂*M* **unfolding** *M-def*
by (*subst marg-dens2-integral*) *simp-all*
also from *X* **have** ... = *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* ∈ *borel-measurable* (*state-measure* (*V* ∪ *V′*) Γ ⨂$_M$ *stock-measure t*)
shows *insert-dens V V′ f* δ
∈ *borel-measurable* (*state-measure* (*shift-var-set* (*V* ∪ *V′*)) (*case-nat t* Γ))
proof−
have (λσ. σ *0*) ∈ *measurable* (*state-measure* (*shift-var-set* (*V* ∪ *V′*)) (*case-nat t* Γ))
(*stock-measure* (*case-nat t* Γ *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 $\varrho \in$ *space* (*state-measure* V' Γ)
$f \in$ *borel-measurable* (*state-measure* ($V \cup V'$) Γ)
shows ($\int^+ x.$ *f x* ∂*dens-ctxt-measure* (V,V',Γ,δ) ϱ) =
$\int^+ x.$ δ (*merge V V'* (x, ϱ)) $*$ *f* (*merge V V'* (x, ϱ)) ∂*state-measure V* Γ
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* $\cup$ *Suc* ‘ V' = *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* ϱ
defines $M \equiv$ *dens-ctxt-measure* (*shift-var-set V*, *Suc*‘V', *case-nat t* Γ, *insert-dens V V' f* δ) ϱ
assumes *dens*: *has-parametrized-subprob-density* (*state-measure* ($V \cup V'$) Γ) *F* (*stock-measure t*) *f*
assumes ϱ: $\varrho \in$ *space* (*state-measure* (*Suc*‘V') (*case-nat t* Γ))
assumes *X*: $X \in$ *sets M*
shows *emeasure M X* =
$\int^+ x.$ *insert-dens V V' f* δ (*merge* (*shift-var-set V*) (*Suc* ‘ V') (x, ϱ)) $*$
indicator X (*merge* (*shift-var-set V*) (*Suc* ‘ V') (x, ϱ))
∂*state-measure* (*shift-var-set V*) (*case-nat t* Γ) (**is** - = *?I*)
proof−
note [*measurable*] = *has-parametrized-subprob-densityD*(*3*)[*OF dens*]
have [*measurable*]:
($\lambda\sigma$. *merge* (*shift-var-set V*) (*Suc* ‘ V') (σ, ϱ))
$\in$ *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*
by (*simp del*: *shift-var-set-Un add*: *shift-var-set-Un*[*symmetric*])
from *assms* **have** *emeasure M X* = ($\int^+ x.$ *indicator X x* ∂M)
by (*subst nn-integral-indicator*)
(*simp-all add*: *dens-ctxt-measure-def state-measure'-def*)
also have *MI*: *indicator X* $\in$ *borel-measurable*
(*state-measure* (*shift-var-set* ($V \cup V'$)) (*case-nat t* Γ))
using *X* **unfolding** *M-def dens-ctxt-measure-def state-measure'-def* **by** *simp*
have ($\int^+ x.$ *indicator X x* ∂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'*:
  $\varrho \in$ *space* (*state-measure* (*Suc* ‘ V') (*case-nat* t Γ)) $\Longrightarrow$
  ($\lambda\sigma$. *merge* V V' (σ, $\varrho \circ$ *Suc*)) $\in$ *measurable* (*state-measure* V Γ) (*state-measure* ($V \cup V'$) Γ)
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*:
  $\varrho \in$ *space* (*state-measure* (*Suc* ‘ V') (*case-nat* t Γ)) $\Longrightarrow$
  ($\lambda\sigma$. δ (*merge* V V' (σ, $\varrho \circ$ *Suc*))) $\in$ *borel-measurable* (*state-measure* V Γ)
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*: $\bigwedge i$. *sigma-finite-measure* (N i)
  assumes *Mf*: $f \in$ *borel-measurable* (Pi_M V N)
  shows ($\int^+ x$. f x ∂*distr* (Pi_M (*Suc*‘V) (*case-nat* M N)) (Pi_M V N) ($\lambda\sigma$. $\sigma \circ$ *Suc*)) =
  ($\int^+ x$. f x ∂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 (*insert* (*Suc* v) (*Suc* ‘ V)) (*case-nat* M N)
  let *?M4* = Pi_M (*insert* (*Suc* v) (*Suc* ‘ V)) (*case-nat* (*count-space* {}) N)
  let *?M4'* = Pi_M (*Suc* ‘ V) (*case-nat* (*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 ∂*?M1 ?V*) = $\int^+ x$. f ($x \circ$ *Suc*) ∂*?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** ... = $\int^+ x$. $\int^+ y$. f (x(*Suc* v := y) $\circ$ *Suc*) ∂N v ∂*?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 (λx y. x(*Suc* v := y) $\circ$ *Suc*) = (λx y. ($x \circ$ *Suc*)(v := y))

by (*intro ext*) (*simp add*: *o-def*)
also have *?M4'* = Pi_M (*Suc* ' *V*) (*case-nat M N*) **by** (*intro PiM-cong*) *auto*
also from *insert* **have** ($\int^+ x.\ \int^+ y.\ f\ ((x \circ Suc)(v := y))\ \partial N\ v\ \partial ...$) =
($\int^+ x.\ \int^+ y.\ f\ (x(v := y))\ \partial N\ v\ \partial ?M1\ V$)
by (*subst nn-integral-distr*)
(*simp-all add*: *borel-measurable-nn-integral measurable-case-nat-Suc-PiM*)
also from *insert* **have** ... = ($\int^+ x.\ \int^+ y.\ f\ (x(v := y))\ \partial N\ v\ \partial Pi_M\ \ V\ N$)
by (*intro insert*(*3*)) *measurable*
also from *insert* **have** ... = ($\int^+ x.\ f\ x\ \partial Pi_M\ \ ?V\ N$)
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 $\bigwedge i.$ *sigma-finite-measure* (*N i*)
shows *distr* (Pi_M (*Suc'V*) (*case-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-measure* (*Suc* ' *V*) (*case-nat t* Γ)) (*state-measure V* Γ) ($\lambda\sigma.\ \sigma \circ Suc$) =
state-measure V Γ (**is** *?M1* = *?M2*)
unfolding *state-measure-def*
apply (*subst* (*2*) *PiM-Suc*[*of* λ*x*. *stock-measure* (Γ *x*) *stock-measure t*, *symmetric*], *simp*)
apply (*intro distr-cong PiM-cong*)
apply (*simp-all split*: *nat.split*)
done

lemma *emeasure-dens-ctxt-measure-insert'*:
fixes *t f* ϱ
defines *M* ≡ *dens-ctxt-measure* (*shift-var-set V*, *Suc'V'*, *case-nat t* Γ, *insert-dens V V' f* δ) ϱ
assumes *dens*: *has-parametrized-subprob-density* (*state-measure* (*V*∪*V'*) Γ) *F* (*stock-measure t*) *f*
assumes ϱ: ϱ ∈ *space* (*state-measure* (*Suc'V'*) (*case-nat t* Γ))
assumes *X*: *X* ∈ *sets M*
shows *emeasure M X* = $\int^+\sigma.\ \delta$ (*merge V V'* (σ, ϱ ∘ *Suc*)) ∗ $\int^+ y.\ f$ (*merge V V'* (σ, ϱ ∘ *Suc*)) *y* ∗
indicator X (*merge* (*shift-var-set V*) (*Suc'V'*) (*case-nat y* σ, ϱ))
∂*stock-measure t* ∂*state-measure V* Γ (**is** - = *?I*)
proof−
let *?m* = λ*x y*. *merge* (*insert 0* (*Suc* ' *V*)) (*Suc* ' *V'*) (*x*(*0* := *y*), ϱ)
from *dens* **have** *Mf*:
case-prod f ∈ *borel-measurable* (*state-measure* (*V*∪*V'*) Γ $\bigotimes_M$ *stock-measure*

t)
  by (*rule has-parametrized-subprob-densityD*)
 note [*measurable*] = *Mf*[*unfolded state-measure-def*]
 have *meas-merge*: (λx. *merge* (*shift-var-set V*) (*Suc'V'*) (*x*, ϱ))
   $\in$ *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'*: (λx. *merge* (*shift-var-set V*) (*Suc ' V'*) (*case-nat* (*snd x*) (*fst x*), ϱ))
   $\in$ *measurable* (*state-measure V* Γ $\bigotimes_M$ *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*. δ (*merge V V'* (σ, ϱ $\circ$ *Suc*)) $*$ *f* (*merge V V'* (σ, ϱ $\circ$ *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* $\in$ *space* (*state-measure V* Γ)
     $\Longrightarrow$ (λy. *f* (*merge V V'* (*x*, ϱ $\circ$ *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*. *insert-dens V V' f* δ (*merge* (*shift-var-set V*) (*Suc ' V'*) (*x*, ϱ)) $*$
     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 ... = ∫⁺x. ∫⁺y. insert-dens V V′ f δ (?m x y) ∗
indicator X (?m x y) ∂stock-measure t ∂state-measure (Suc‘V) (case-nat t Γ)
(**is** - = ?I) **using** ϱ X 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-dens′], simp)
apply (simp-all add: measurable-compose[OF - borel-measurable-indicator] image-Un)
done
also have ⋀σ y. σ ∈ space (state-measure (Suc‘V) (case-nat t Γ)) ⟹
y ∈ space (stock-measure t) ⟹
(remove-var (merge (insert 0 (Suc ‘ V)) (Suc ‘ V′) (σ(0:=y), ϱ))) =
merge V V′ (σ ∘ Suc, ϱ ∘ Suc)
by (auto simp: merge-def remove-var-def)
hence ?I = ∫⁺σ. ∫⁺y. δ (merge V V′ (σ ∘ Suc, ϱ ∘ Suc)) ∗ f (merge V V′ (σ ∘ Suc, ϱ ∘ Suc)) y ∗
indicator X (?m σ y)
∂stock-measure t ∂state-measure (Suc‘V) (case-nat t Γ) (**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-eq: ⋀x y. ?m x y = merge (shift-var-set V) (Suc‘V′) (case-nat y (x ∘ Suc), ϱ)
by (intro ext) (auto simp add: merge-def shift-var-set-def split: nat.split)
have ?I = ∫⁺σ. ∫⁺y. δ (merge V V′ (σ, ϱ ∘ Suc)) ∗ f (merge V V′ (σ, ϱ ∘ Suc)) y ∗
indicator X (merge (shift-var-set V) (Suc‘V′) (case-nat y σ, ϱ))
∂stock-measure t ∂state-measure V Γ **using** ϱ 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 ... = ∫⁺σ. δ (merge V V′ (σ, ϱ ∘ Suc)) ∗ ∫⁺y. f (merge V V′ (σ, ϱ ∘ Suc)) y ∗
indicator X (merge (shift-var-set V) (Suc‘V′) (case-nat y σ, ϱ))
∂stock-measure t ∂state-measure V Γ **using** ϱ 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 δ*)
(**is** *density-context ?V ?V' ?Γ' ?δ'*)
unfolding *density-context-def*
proof (*intro allI conjI impI*)
note *measurable-insert-dens*[*OF has-parametrized-subprob-densityD*(*3*)[*OF dens*]]
thus *insert-dens V V' f δ*
∈ *borel-measurable* (*state-measure* (*shift-var-set V* ∪ *Suc* ‘ *V'*) (*case-nat t* Γ))
unfolding *shift-var-set-def* **by** (*simp only*: *image-Un Un-insert-left*)
next
fix *ϱ* **assume** *ϱ*: *ϱ* ∈ *space* (*state-measure ?V' ?Γ'*)
hence *ϱ'*: *ϱ* ∘ *Suc* ∈ *space* (*state-measure V'* Γ)
by (*auto simp*: *state-measure-def space-PiM dest*: *PiE-mem*)
note *dens'* = *has-parametrized-subprob-densityD*[*OF dens*]
note *Mf*[*measurable*] = *dens'*(*3*)
have *M-merge*: (*λx. merge* (*shift-var-set V*) (*Suc* ‘ *V'*) (*x*, *ϱ*))
∈ *measurable* (Pi_M (*insert 0* (*Suc* ‘ *V*)) (*λy. stock-measure* (*case-nat t* Γ *y*)))
(*state-measure* (*shift-var-set* (*V* ∪ *V'*)) (*case-nat t* Γ))
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'*,*?Γ'*,*?δ'*) *ϱ*) (**is** *subprob-space ?M*)
proof (*rule subprob-spaceI*)
interpret *product-sigma-finite* (*λy. stock-measure* (*case y of 0* ⇒ *t* | *Suc x* ⇒ Γ *x*))
by (*simp add*: *product-sigma-finite-def*)
have *Suc-state-measure*:
⋀*x. x* ∈ *space* (*state-measure* (*Suc* ‘ *V*) (*case-nat t* Γ)) ⟹
merge V V' (*x* ∘ *Suc*, *ϱ* ∘ *Suc*) ∈ *space* (*state-measure* (*V* ∪ *V'*) Γ)
using *ϱ*
by (*intro merge-in-state-measure*) (*auto simp*: *state-measure-def space-PiM dest*: *PiE-mem*)

have *S*[*simp*]: ⋀*x X. Suc x* ∈ *Suc* ‘ *X* ⟷ *x* ∈ *X* **by** (*rule inj-image-mem-iff*) *simp*
let *?M* = *dens-ctxt-measure* (*?V*,*?V'*,*?Γ'*,*?δ'*) *ϱ*
from *ϱ* **have** ⋀*σ. σ* ∈ *space* (*state-measure ?V ?Γ'*) ⟹ *merge ?V ?V'* (*σ*, *ϱ*) ∈ *space ?M*
by (*auto simp*: *dens-ctxt-measure-def state-measure'-def simp del*: *shift-var-set-Un intro*!: *merge-in-state-measure*)

hence *emeasure ?M* (*space ?M*) =
$\int^+\sigma$. *insert-dens V V' f* δ (*merge ?V ?V'* (σ, ϱ)) ∂*state-measure ?V ?Γ'*
by (*subst emeasure-dens-ctxt-measure-insert*[*OF dens* ϱ], *simp*, *intro nn-integral-cong*)
(*simp split*: *split-indicator*)
also have ... = $\int^+\sigma$. *insert-dens V V' f* δ (*merge ?V ?V'* (σ, ϱ))
∂*state-measure* (*insert 0* (*Suc* ‘ *V*)) *?Γ'*
by (*simp add*: *shift-var-set-def*)
also have ... = $\int^+\sigma$. $\int^+x$. *insert-dens V V' f* δ (*merge ?V ?V'* (σ(*0* := *x*), ϱ))
∂*stock-measure t* ∂*state-measure* (*Suc* ‘ *V*) *?Γ'*
unfolding *state-measure-def* **using** *M-merge*
by (*subst product-nn-integral-insert*) *auto*
also have ... = $\int^+\sigma$. $\int^+x$. δ (*remove-var* (*merge ?V ?V'* (σ(*0*:=*x*), ϱ))) ∗
f (*remove-var* (*merge ?V ?V'* (σ(*0*:=*x*), ϱ))) *x*
∂*stock-measure t* ∂*state-measure* (*Suc* ‘ *V*) *?Γ'* (**is** - = *?I*)
by (*intro nn-integral-cong*) (*auto simp*: *insert-dens-def merge-def shift-var-set-def*)
also have $\bigwedge\sigma$ *x*. *remove-var* (*merge ?V ?V'* (σ(*0*:=*x*), ϱ)) = *merge V V'* (σ ∘ *Suc*, ϱ ∘ *Suc*)
by (*intro ext*) (*auto simp*: *remove-var-def merge-def shift-var-set-def o-def*)
hence *?I* = $\int^+\sigma$. $\int^+x$. δ (*merge V V'* (σ ∘ *Suc*, ϱ ∘ *Suc*)) ∗ *f* (*merge V V'* (σ ∘ *Suc*, ϱ ∘ *Suc*)) *x*
∂*stock-measure t* ∂*state-measure* (*Suc* ‘ *V*) *?Γ'* **by** *simp*
also have ... = $\int^+\sigma$. δ (*merge V V'* (σ ∘ *Suc*, ϱ ∘ *Suc*)) ∗
($\int^+x$. *f* (*merge V V'* (σ ∘ *Suc*, ϱ ∘ *Suc*)) *x* ∂*stock-measure t*)
∂*state-measure* (*Suc* ‘ *V*) *?Γ'* (**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** σ: $\sigma \in$ *space* (*state-measure* (*Suc* ‘ *V*) *?Γ'*)
let *?σ'* = *merge V V'* (σ ∘ *Suc*, ϱ ∘ *Suc*)
let *?N* = *density* (*stock-measure t*) (*f ?σ'*)
have ($\int^+x$. *f* (*merge V V'* (σ ∘ *Suc*, ϱ ∘ *Suc*)) *x* ∂*stock-measure t*) = *emeasure ?N* (*space ?N*)
using *dens'*(*3*) *Suc-state-measure*[*OF* σ]
by (*simp-all cong*: *nn-integral-cong' add*: *emeasure-density*)
also have *?N* = *F ?σ'* **by** (*subst dens'*) (*simp-all add*: *Suc-state-measure* σ)
also have *subprob-space* (*F ?σ'*) **by** (*rule dens'*) (*simp-all add*: *Suc-state-measure* σ)
hence *emeasure* (*F ?σ'*) (*space* (*F ?σ'*)) ≤ *1* **by** (*rule subprob-space.emeasure-space-le-1*)
finally have ($\int^+x$. *f* (*merge V V'* (σ ∘ *Suc*, ϱ ∘ *Suc*)) *x* ∂*stock-measure t*) ≤ *1* **.**
}
hence *?I* ≤ $\int^+\sigma$. δ (*merge V V'* (σ ∘ *Suc*, ϱ ∘ *Suc*)) ∗ *1* ∂*state-measure* (*Suc* ‘ *V*) *?Γ'*
by (*intro nn-integral-mono mult-left-mono*) (*simp-all add*: *Suc-state-measure*)
also have ... = $\int^+\sigma$. δ (*merge V V'* (σ, ϱ ∘ *Suc*))

∂*distr* (*state-measure* (*Suc* ‘ *V*) *?Γ′*) (*state-measure* *V* Γ) (λσ. σ ∘ *Suc*)
(**is** - = *nn-integral ?N* -)
using ϱ **by** (*subst nn-integral-distr*) (*simp-all add*: *measurable-case-nat-Suc merge-Suc-aux*)
also have *?N* = *state-measure V* Γ **by** (*rule distr-state-measure-Suc*)
also have ($\int^+$σ. δ (*merge V V′* (σ, ϱ ∘ *Suc*)) ∂*state-measure V* Γ) =
($\int^+$σ. *1* ∂*dens-ctxt-measure* 𝒴 (ϱ ∘ *Suc*)) (**is** - = *nn-integral ?N* -)
by (*subst nn-integral-dens-ctxt-measure*) (*simp-all add*: ϱ′)
also have ... = ($\int^+$σ. *indicator* (*space ?N*) σ ∂*?N*)
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also have ... = *emeasure ?N* (*space ?N*) **by** *simp*
also have ... ≤ *1* **by** (*simp-all add*: *subprob-space.emeasure-space-le-1 subprob-space-dens* ϱ′)
finally show *emeasure ?M* (*space ?M*) ≤ *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 ϱ: ϱ ∈ *space* (*state-measure V′* Γ)
assumes *meas-M*: *M* ∈ *measurable* (*state-measure* (*V*∪*V′*) Γ) (*subprob-algebra* (*stock-measure t*))
assumes *meas-f*[*measurable*]: *case-prod f* ∈ *borel-measurable* (*state-measure* (*V*∪*V′*) Γ $\bigotimes_M$ *stock-measure t*)
assumes *has-dens*: ⋀ϱ. ϱ ∈ *space* (*state-measure* (*V*∪*V′*) Γ) ⟹
has-subprob-density (*M* ϱ) (*stock-measure t*) (*f* ϱ)
shows *do* {σ ← *dens-ctxt-measure* (*V*,*V′*,Γ,δ) ϱ;
y ← *M* σ;
return (*state-measure* (*shift-var-set* (*V* ∪ *V′*)) (*case-nat t* Γ)) (*case-nat y* σ)} =
dens-ctxt-measure (*shift-var-set V*, *Suc*‘*V′*, *case-nat t* Γ, *insert-dens V V′ f* δ)
(*case-nat undefined* ϱ)
(**is** *bind ?N* (λ-. *bind* - (λ-. *return ?R* -)) = *dens-ctxt-measure* (*?V*,*?V′*,*?Γ′*,*?δ′*) -)
proof (*intro measure-eqI*)
let *?lhs* = *?N* ⪢= (λσ . *M* σ ⪢= (λ*y*. *return ?R* (*case-nat y* σ)))
let *?rhs* = *dens-ctxt-measure* (*?V*,*?V′*,*?Γ′*,*?δ′*) (*case-nat undefined* ϱ)

have *meas-f′*: ⋀*M g h*. *g* ∈ *measurable M* (*state-measure* (*V*∪*V′*) Γ) ⟹
h ∈ *measurable M* (*stock-measure t*) ⟹
(λ*x*. *f* (*g x*) (*h x*)) ∈ *borel-measurable M* **by** *measurable*
have *t*: *t* = *?Γ′* *0* **by** *simp*

have *nonempty*: *space ?N* ≠ {}
by (*auto simp*: *dens-ctxt-measure-def state-measure′-def state-measure-def space-PiM PiE-eq-empty-iff*)
have *meas-N-eq*: *measurable ?N* = *measurable* (*state-measure* (*V*∪*V′*) Γ)

by (*intro ext measurable-cong-sets*) (*auto simp*: *dens-ctxt-measure-def state-measure′-def*)
have *meas-M′*: $M \in$ *measurable ?N* (*subprob-algebra* (*stock-measure t*))
by (*subst meas-N-eq*) (*rule meas-M*)
have *meas-N′*: $\bigwedge R$. *measurable* (*?N* $\bigotimes_M$ *R*) = *measurable* (*state-measure* ($V \cup V'$) $\Gamma \bigotimes_M R$)
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*: $\bigwedge \varrho$. $\varrho \in$ *space ?N* $\Longrightarrow$ *measurable* ($M\ \varrho$) = *measurable* (*stock-measure t*)
by (*intro ext measurable-cong-sets sets-kernel*[*OF meas-M′*]) *simp-all*
have *meas-rhs*: $\bigwedge M$. *measurable M ?rhs* = *measurable M ?R*
by (*intro ext measurable-cong-sets*) (*simp-all add*: *dens-ctxt-measure-def state-measure′-def*)
have *subprob-algebra-rhs*: *subprob-algebra ?rhs* = *subprob-algebra* (*state-measure* (*shift-var-set* ($V \cup V'$)) *?Γ′*)
unfolding *dens-ctxt-measure-def state-measure′-def* **by** (*intro subprob-algebra-cong*) *auto*
have *nonempty′*: $\bigwedge \varrho$. $\varrho \in$ *space ?N* $\Longrightarrow$ *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*: $\bigwedge x$. $x \in$ *space* (*state-measure V Γ*) $\Longrightarrow$
merge V V′ (x, ϱ) $\in$ *space* (*dens-ctxt-measure* $\mathcal{Y}$ ϱ)
by (*simp add*: *space-dens-ctxt-measure merge-in-state-measure* ϱ)

have *sets ?lhs* = *sets* (*state-measure* (*shift-var-set* ($V \cup V'$)) *?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*]:
($\lambda\sigma$. *merge V V′* (σ, ϱ)) $\in$ *measurable* (*state-measure V Γ*) (*state-measure* ($V \cup V'$) Γ)
using ϱ **unfolding** *state-measure-def* **by** − *measurable*

fix X **assume** $X \in$ *sets ?lhs*
hence X: $X \in$ *sets ?rhs* **by** (*simp add*: *sets-eq*)
hence *emeasure ?lhs X* = $\int^+\sigma$. *emeasure* ($M\ \sigma \gg\!=$ (λy. *return ?R* (*case-nat y* σ))) X ∂*?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** ... =
$\int^+ x$. δ (*merge V V′* (x, ϱ)) $*$ *emeasure* (M (*merge V V′* (x, ϱ)) $\gg\!=$
(λy. *return ?R* (*case-nat y* (*merge V V′* (x, ϱ))))) X ∂*state-measure V* Γ
apply (*subst nn-integral-dens-ctxt-measure*[*OF* ϱ])
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** ... = ∫⁺*x*. δ (*merge V V′* (*x*, ϱ)) ∗
∫⁺*y*. *indicator X* (*case-nat y* (*merge V V′* (*x*, ϱ)))
∂*M* (*merge V V′* (*x*, ϱ)) ∂*state-measure V* Γ (**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 ⋀*x*. *x* ∈ *space* (*state-measure V* Γ) ⟹
M (*merge V V′* (*x*, ϱ)) = *density* (*stock-measure t*) (*f* (*merge V V′* (*x*, ϱ)))
by (*intro has-subprob-densityD*[*OF has-dens*]) (*simp add*: *merge-in-state-measure* ϱ)
hence *?I* = ∫⁺*x*. δ (*merge V V′* (*x*, ϱ)) ∗
∫⁺*y*. *indicator X* (*case-nat y* (*merge V V′* (*x*, ϱ)))
∂*density* (*stock-measure t*) (*f* (*merge V V′* (*x*, ϱ))) ∂*state-measure V* Γ
by (*intro nn-integral-cong*) *simp*
also have ... = ∫⁺*x*. δ (*merge V V′* (*x*, ϱ)) ∗
∫⁺*y*. *f* (*merge V V′* (*x*, ϱ)) *y* ∗ *indicator X* (*case-nat y* (*merge V V′* (*x*, ϱ)))
∂*stock-measure t* ∂*state-measure V* Γ (**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* ϱ ∘ *Suc* = ϱ **by** (*intro ext*) *simp*
have *B*: ⋀*x y*. *x* ∈ *space* (*state-measure V* Γ) ⟹ *y* ∈ *space* (*stock-measure t*) ⟹
(*case-nat y* (*merge V V′* (*x*, ϱ))) =
(*merge* (*shift-var-set V*) (*Suc* ‘ *V′*) (*case-nat y x*, *case-nat undefined* ϱ))
by (*intro ext*) (*auto simp add*: *merge-def shift-var-set-def split*: *nat.split*)
have *C*: ⋀*x*. *x* ∈ *space* (*state-measure V* Γ) ⟹
(∫⁺*y*. *f* (*merge V V′* (*x*, ϱ)) *y* ∗ *indicator X* (*case-nat y* (*merge V V′* (*x*,ϱ))) ∂*stock-measure t*) =
∫⁺*y*. *f* (*merge V V′* (*x*, ϱ)) *y* ∗ *indicator X* (*merge* (*shift-var-set V*) (*Suc*‘*V′*)
(*case-nat y x*,*case-nat undefined* ϱ)) ∂*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' Γ (*if-dens* δ f b)
unfolding *density-context-def*
proof (*intro allI conjI impI subprob-spaceI*)
note D = *has-parametrized-subprob-densityD*[*OF assms*]
from $D(3)$ **show** M: *if-dens* δ f b $\in$ *borel-measurable* (*state-measure* ($V \cup V'$) Γ)
by (*intro measurable-if-dens*) *simp-all*

fix ϱ **assume** ϱ: $\varrho \in$ *space* (*state-measure* V' Γ)
hence [*measurable*]: ($\lambda\sigma$. *merge* V V' (σ, ϱ)) $\in$
measurable (*state-measure* V Γ) (*state-measure* ($V \cup V'$) Γ)
unfolding *state-measure-def* **by** *simp*

{
fix σ **assume** $\sigma \in$ *space* (*state-measure* V Γ)
with ϱ **have** $\sigma\varrho$: *merge* V V' (σ, ϱ) $\in$ *space* (*state-measure* ($V \cup V'$) Γ)
by (*intro merge-in-state-measure*)
with *assms* **have** *has-subprob-density* (M (*merge* V V' (σ, ϱ))) (*count-space* (*range BoolVal*))
(f (*merge* V V' (σ, ϱ)))
unfolding *has-parametrized-subprob-density-def* **by** *auto*
with $\sigma\varrho$ **have** f (*merge* V V' (σ, ϱ)) (*BoolVal* b) $\leq$ *1* δ (*merge* V V' (σ, ϱ)) $\geq$ *0*
by (*auto intro*: *subprob-count-space-density-le-1*)
} **note** *dens-props* = *this*

from ϱ **interpret** *subprob-space dens-ctxt-measure* $\mathcal{Y}$ ϱ **by** (*rule subprob-space-dens*)
let *?M* = *dens-ctxt-measure* (V, V', Γ, *if-dens* δ f b) ϱ
have *emeasure ?M* (*space ?M*) =
$\int^+ x$. *if-dens* δ f b (*merge* V V' (x, ϱ)) ∂*state-measure* V Γ
using M ϱ **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** ... $\leq$ $\int^+ x$. δ (*merge* V V' (x, ϱ)) $*$ *1* ∂*state-measure* V Γ
unfolding *if-dens-def* **using** *dens-props*
by (*intro nn-integral-mono mult-left-mono*) *simp-all*
also from ϱ **have** ... = *branch-prob* $\mathcal{Y}$ ϱ **by** (*simp add*: *branch-prob-altdef*)
also have ... = *emeasure* (*dens-ctxt-measure* $\mathcal{Y}$ ϱ) (*space* (*dens-ctxt-measure* $\mathcal{Y}$ ϱ))
by (*simp add*: *branch-prob-def*)
also have ... $\leq$ *1* **by** (*rule emeasure-space-le-1*)
finally show *emeasure ?M* (*space ?M*) $\leq$ *1* **.**
qed (*insert disjoint*, *auto*)

lemma *density-context-if-dens-det*:
assumes e: $\Gamma \vdash e$: *BOOL randomfree* e *free-vars* $e \subseteq V \cup V'$
shows *density-context* V V' Γ (*if-dens-det* δ e b)

unfolding *density-context-def*
proof (*intro allI conjI impI subprob-spaceI*)
  from *assms* **show** *M*: *if-dens-det* δ *e b* $\in$ *borel-measurable* (*state-measure* ($V \cup V'$) Γ)
    by (*intro measurable-if-dens-det*) *simp-all*

  fix ϱ **assume** ϱ: $\varrho \in$ *space* (*state-measure* V' Γ)
  hence [*measurable*]: ($\lambda\sigma$. *merge* V V' (σ, ϱ)) $\in$
      measurable (*state-measure* V Γ) (*state-measure* ($V \cup V'$) Γ)
    unfolding *state-measure-def* **by** *simp*

  from ϱ **interpret** *subprob-space dens-ctxt-measure* $\mathcal{Y}$ ϱ **by** (*rule subprob-space-dens*)
  let *?M* = *dens-ctxt-measure* (V, V', Γ, *if-dens-det* δ *e b*) ϱ
  have *emeasure ?M* (*space ?M*) =
      $\int^+ x$. *if-dens-det* δ *e b* (*merge* V V' (x, ϱ)) ∂*state-measure* V Γ
    using *M* ϱ **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** ... $\le \int^+ x$. δ (*merge* V V' (x, ϱ)) $*$ *1* ∂*state-measure* V Γ
    unfolding *if-dens-det-def*
   by (*intro nn-integral-mono mult-left-mono*) (*simp-all add*: *merge-in-state-measure*)
  also from ϱ **have** ... = *branch-prob* $\mathcal{Y}$ ϱ **by** (*simp add*: *branch-prob-altdef*)
  also have ... = *emeasure* (*dens-ctxt-measure* $\mathcal{Y}$ ϱ) (*space* (*dens-ctxt-measure* $\mathcal{Y}$ ϱ))
    by (*simp add*: *branch-prob-def*)
  also have ... $\le$ *1* **by** (*rule emeasure-space-le-1*)
  finally show *emeasure ?M* (*space ?M*) $\le$ *1* **.**
qed (*insert disjoint assms*, *auto intro*: *measurable-if-dens-det*)

lemma *density-context-empty*[*simp*]: *density-context* {} ($V \cup V'$) Γ (λ-. *1*)
unfolding *density-context-def*
proof (*intro allI conjI impI subprob-spaceI*)
  fix ϱ **assume** ϱ: $\varrho \in$ *space* (*state-measure* ($V \cup V'$) Γ)
  let *?M* = *dens-ctxt-measure* ({},$V \cup V'$,Γ,λ-. *1*) ϱ
  from ϱ **have** $\bigwedge\sigma$. *merge* {} ($V \cup V'$) (σ,ϱ) = ϱ
    by (*intro ext*) (*auto simp*: *merge-def state-measure-def space-PiM*)
  with ϱ **show** *emeasure ?M* (*space ?M*) $\le$ *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 $\varrho \in$ *space* (*state-measure* V' Γ) *subprob-space N*
  shows *dens-ctxt-measure* $\mathcal{Y}$ ϱ $\gg\!=$ (λ-. *N*) = *density N* (λ-. *branch-prob* $\mathcal{Y}$ ϱ) (**is** *?M1* = *?M2*)
proof (*rule measure-eqI*)
  have [*simp*]: *sets ?M1* = *sets N* **by** (*auto simp*: *space-subprob-algebra assms*)
  thus *sets ?M1* = *sets ?M2* **by** *simp*

fix X **assume** X: $X \in$ *sets ?M1*
with *assms* **have** *emeasure ?M1* X = *emeasure N* X ∗ *branch-prob* $\mathcal{Y}$ ϱ
unfolding *branch-prob-def* **by** (*subst emeasure-bind-const′*) (*auto simp*: *subprob-space-dens*)
also from X **have** *emeasure N X* = $\int^{+}x$. *indicator X x* ∂*N* **by** *simp*
also from X **have** ... ∗ *branch-prob* $\mathcal{Y}$ ϱ = $\int^{+}x$. *branch-prob* $\mathcal{Y}$ ϱ ∗ *indicator X x* ∂*N*
by (*subst nn-integral-cmult*) (*auto simp*: *branch-prob-def field-simps*)
also from X **have** ... = *emeasure ?M2 X* **by** (*simp add*: *emeasure-density*)
finally show *emeasure ?M1 X* = *emeasure ?M2 X* **.**
qed

lemma *nn-integral-dens-ctxt-measure-restrict*:
assumes $\varrho \in$ *space* (*state-measure* V' Γ) *f* $\varrho \geq 0$
assumes *f* ∈ *borel-measurable* (*state-measure* V' Γ)
shows ($\int^{+}x$. *f* (*restrict x* V') ∂*dens-ctxt-measure* $\mathcal{Y}$ ϱ) = *branch-prob* $\mathcal{Y}$ ϱ ∗ *f* ϱ
proof−
have ($\int^{+}x$. *f* (*restrict x* V') ∂*dens-ctxt-measure* (V,V',Γ,δ) ϱ) =
$\int^{+}$ *x*. δ (*merge V* V' (*x*, ϱ)) ∗ *f* (*restrict* (*merge V* V' (*x*, ϱ)) V') ∂*state-measure V* Γ
(**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* ∈ *space* (*state-measure V* Γ) ⟹ *restrict* (*merge V* V' (*x*, ϱ)) V' = ϱ
by (*intro ext*) (*auto simp*: *restrict-def merge-def state-measure-def space-PiM dest*: *PiE-mem*)
hence *?I* = $\int^{+}$ *x*. δ (*merge V* V' (*x*, ϱ)) ∗ *f* ϱ ∂*state-measure V* Γ
by (*intro nn-integral-cong*) *simp*
also have ... = ($\int^{+}x$. *f* ϱ ∂*dens-ctxt-measure* (V,V',Γ,δ) ϱ)
by (*subst nn-integral-dens-ctxt-measure*) (*simp-all add*: *assms*)
also have ... = *f* ϱ ∗ *branch-prob* $\mathcal{Y}$ ϱ
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 Γ ⊢ *oper* $$ *e* : *t′ free-vars e* ⊆ *V* ∪ V' ϱ ∈ *space* (*state-measure* V' Γ)
defines *M* ≡ *dens-ctxt-measure* (V,V',Γ,δ) ϱ
shows *M* ⤜ (λσ. *expr-sem* σ (*oper* $$ *e*)) =
distr (*M* ⤜ (λσ. *expr-sem* σ *e*)) (*stock-measure t′*) (*op-sem oper*)
proof−
from *assms*(*1*) **obtain** *t* **where** *t1*: Γ ⊢ *e* : *t* **and** *t2*: *op-type oper t* = *Some t′* **by** *auto*

let *?N* = *stock-measure t* **and** *?R* = *subprob-algebra* (*stock-measure t′*)

{
fix *x* **assume** $x \in$ *space* (*stock-measure t*)
with *t1 assms*(*2*,*3*) **have** *val-type x* = *t*
by (*auto simp*: *state-measure-def space-PiM dest*: *PiE-mem*)
hence *return-val* (*op-sem oper x*) = *return* (*stock-measure t′*) (*op-sem oper x*)
unfolding *return-val-def* **by** (*subst op-sem-val-type*) (*simp-all add*: *t2*)
} **note** *return-op-sem* = *this*

from *assms* **and** *t1* **have** *M-e*: ($\lambda\sigma$. *expr-sem* σ *e*) $\in$ *measurable M* (*subprob-algebra* (*stock-measure t*))
by (*simp add*: *M-def measurable-dens-ctxt-measure-eq measurable-expr-sem*)
from *return-op-sem*
have *M-cong*: (λx. *return-val* (*op-sem oper x*)) $\in$ *measurable ?N ?R* $\longleftrightarrow$
(λx. *return* (*stock-measure t′*) (*op-sem oper x*)) $\in$ *measurable ?N ?R*
by (*intro measurable-cong*) *simp*
have *M-ret*: (λx. *return-val* (*op-sem oper x*)) $\in$ *measurable* (*stock-measure t*) *?R*
by (*subst M-cong*, *intro measurable-compose*[*OF measurable-op-sem*[*OF t2*]] *return-measurable*)

from *M-e* **have** [*simp*]: *sets* (*M* $\gg\!=$ ($\lambda\sigma$. *expr-sem* σ *e*)) = *sets* (*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-op*: *op-sem oper* $\in$ *measurable* (*M* $\gg\!=$ ($\lambda\sigma$. *expr-sem* σ *e*)) (*stock-measure t′*)
by (*auto intro*!: *measurable-op-sem t2*)
have [*simp*]: *space* (*M* $\gg\!=$ ($\lambda\sigma$. *expr-sem* σ *e*)) = *space* (*stock-measure t*)
by (*rule sets-eq-imp-space-eq*) *simp*

from *M-e* **and** *M-ret* **have** *M* $\gg\!=$ ($\lambda\sigma$. *expr-sem* σ (*oper* $$ *e*)) =
(*M* $\gg\!=$ ($\lambda\sigma$. *expr-sem* σ *e*)) $\gg\!=$ (λx. *return-val* (*op-sem oper x*))
unfolding *M-def* **by** (*subst expr-sem.simps*, *intro bind-assoc*[*symmetric*]) *simp-all*
also have ... = (*M* $\gg\!=$ ($\lambda\sigma$. *expr-sem* σ *e*)) $\gg\!=$ (λx. *return* (*stock-measure t′*) (*op-sem oper x*))
by (*intro bind-cong refl*) (*simp add*: *return-op-sem*)
also have ... = *distr* (*M* $\gg\!=$ ($\lambda\sigma$. *expr-sem* σ *e*)) (*stock-measure t′*) (*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 $\bigwedge\sigma.\ \sigma \in$ *space* (*state-measure* $(V \cup V')\ \Gamma) \Longrightarrow \delta\ \sigma = \delta'\ \sigma$
assumes [*simp*, *measurable*]: $\delta' \in$ *borel-measurable* (*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** ϱ: $\varrho \in$ *space* (*state-measure* $V'\ \Gamma$)
let *?M* = *dens-ctxt-measure* $(V,\ V',\ \Gamma,\ \delta')\ \varrho$
let *?N* = *dens-ctxt-measure* $(V,\ V',\ \Gamma,\ \delta)\ \varrho$
from ϱ **have** *emeasure ?M* (*space ?M*) $= \int^+ x.\ \delta'$ (*merge* $V\ V'\ (x,\ \varrho)$) ∂*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** ... $= \int^+ x.\ \delta$ (*merge* $V\ V'\ (x,\ \varrho)$) ∂*state-measure* $V\ \Gamma$
by (*intro nn-integral-cong*, *subst assms*(*1*)) (*simp-all add*: *merge-in-state-measure*)
also from ϱ **have** ... = *branch-prob* $(V, V', \Gamma, \delta)\ \varrho$ **by** (*simp add*: *branch-prob-altdef*)
also have ... = *emeasure ?N* (*space ?N*) **by** (*simp add*: *branch-prob-def*)
also from ϱ **have** ... $\leq$ *1* **by** (*intro subprob-space.emeasure-space-le-1 subprob-space-dens*)
finally show *emeasure ?M* (*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 a expression to a probability density function of its distribution. The density is a HOL function of type *val* ⇒ *ennreal*.

inductive *expr-has-density* :: *dens-ctxt* ⇒ *expr* ⇒ (*state* ⇒ *val* ⇒ *ennreal*) ⇒ *bool*
((*1*- $\vdash_d$/ (- ⇒/ -)) [*50*,*0*,*50*] *50*) **where**
hd-AE: $[\![(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f;\ \Gamma \vdash e : t;$
$\bigwedge\varrho.\ \varrho \in$ *space* (*state-measure* $V'\ \Gamma$) $\Longrightarrow$
AE x *in stock-measure* $t.\ f\ \varrho\ x = f'\ \varrho\ x$;
case-prod $f' \in$ *borel-measurable* (*state-measure* $V'\ \Gamma \bigotimes_M$ *stock-measure* t)$]\!]$
$\Longrightarrow (V, V', \Gamma, \delta) \vdash_d e \Rightarrow f'$
| *hd-dens-ctxt-cong*:

(V, V′,Γ,δ) $\vdash_d$ e ⇒ f ⟹ (⋀σ. σ ∈ space (state-measure (V ∪ V′) Γ) ⟹ δ σ = δ′ σ)
⟹ (V, V′,Γ,δ′) $\vdash_d$ e ⇒ f
| hd-val: countable-type (val-type v) ⟹
(V, V′,Γ,δ) $\vdash_d$ Val v ⇒ (λϱ x. branch-prob (V, V′,Γ,δ) ϱ ∗ indicator {v} x)
| hd-var: x ∈ V ⟹ (V, V′,Γ,δ) $\vdash_d$ Var x ⇒ marg-dens (V, V′,Γ,δ) x
| hd-let: ⟦({}, V∪V′,Γ,λ-. 1) $\vdash_d$ e1 ⇒ f;
(shift-var-set V,Suc‘V′,the(expr-type Γ e1) · Γ,insert-dens V V′ f δ) $\vdash_d$ e2 ⇒ g⟧
⟹ (V, V′,Γ,δ) $\vdash_d$ LetVar e1 e2 ⇒ (λϱ. g (case-nat undefined ϱ))
| hd-rand: (V, V′,Γ,δ) $\vdash_d$ e ⇒ f ⟹ (V, V′,Γ,δ) $\vdash_d$ Random dst e ⇒ apply-dist-to-dens dst f
| hd-rand-det: randomfree e ⟹ free-vars e ⊆ V′ ⟹
(V, V′,Γ,δ) $\vdash_d$ Random dst e ⇒
(λϱ x. branch-prob (V, V′,Γ,δ) ϱ ∗ dist-dens dst (expr-sem-rf ϱ e) x)
| hd-fail: (V, V′,Γ,δ) $\vdash_d$ Fail t ⇒ (λ- -. 0)
| hd-pair: x ∈ V ⟹ y ∈ V ⟹ x ≠ y ⟹ (V, V′,Γ,δ) $\vdash_d$ <Var x, Var y> ⇒ marg-dens2 (V, V′,Γ,δ) x y
| hd-if: ⟦({}, V∪V′,Γ,λ-. 1) $\vdash_d$ b ⇒ f;
(V, V′,Γ,if-dens δ f True) $\vdash_d$ e1 ⇒ g1; (V, V′,Γ,if-dens δ f False) $\vdash_d$ e2 ⇒ g2⟧
⟹ (V, V′,Γ,δ) $\vdash_d$ IF b THEN e1 ELSE e2 ⇒ (λϱ x. g1 ϱ x + g2 ϱ x)
| hd-if-det: ⟦randomfree b; (V, V′,Γ,if-dens-det δ b True) $\vdash_d$ e1 ⇒ g1;
(V, V′,Γ,if-dens-det δ b False) $\vdash_d$ e2 ⇒ g2⟧
⟹ (V, V′,Γ,δ) $\vdash_d$ IF b THEN e1 ELSE e2 ⇒ (λϱ x. g1 ϱ x + g2 ϱ x)
| hd-fst: (V, V′,Γ,δ) $\vdash_d$ e ⇒ f ⟹
(V, V′,Γ,δ) $\vdash_d$ Fst $$ e ⇒
(λϱ x. $\int^+$y. f ϱ <|x,y|> ∂stock-measure (the (expr-type Γ (Snd $$ e))))
| hd-snd: (V, V′,Γ,δ) $\vdash_d$ e ⇒ f ⟹
(V, V′,Γ,δ) $\vdash_d$ Snd $$ e ⇒
(λϱ y. $\int^+$x. f ϱ <|x,y|> ∂stock-measure (the (expr-type Γ (Fst $$ e))))
| hd-op-discr: countable-type (the (expr-type Γ (oper $$ e))) ⟹ (V, V′,Γ,δ) $\vdash_d$ e ⇒ f ⟹
(V, V′,Γ,δ) $\vdash_d$ oper $$ e ⇒ (λϱ y. $\int^+$x. (if op-sem oper x = y then 1 else 0) ∗ f ϱ x
∂stock-measure (the (expr-type Γ e)))
| hd-neg: (V, V′,Γ,δ) $\vdash_d$ e ⇒ f ⟹
(V, V′,Γ,δ) $\vdash_d$ Minus $$ e ⇒ (λσ x. f σ (op-sem Minus x))
| hd-addc: (V, V′,Γ,δ) $\vdash_d$ e ⇒ f ⟹ randomfree e′ ⟹ free-vars e′ ⊆ V′ ⟹
(V, V′,Γ,δ) $\vdash_d$ Add $$ <e, e′> ⇒
(λϱ x. f ϱ (op-sem Add <|x, expr-sem-rf ϱ (Minus $$ e′)|>))
| hd-multc: (V, V′,Γ,δ) $\vdash_d$ e ⇒ f ⟹ val-type c = REAL ⟹ c ≠ RealVal 0 ⟹
(V, V′,Γ,δ) $\vdash_d$ Mult $$ <e, Val c> ⇒
(λϱ x. f ϱ (op-sem Mult <|x, op-sem Inverse c|>) ∗
inverse (abs (extract-real c)))

| *hd-exp*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow$
$(V, V', \Gamma, \delta) \vdash_d$ *Exp* $$ $e \Rightarrow$
($\lambda\sigma$ *x. if extract-real x* > *0 then*
f σ (*lift-RealVal safe-ln x*) ∗ *inverse* (*extract-real x*) *else 0*)
| *hd-inv*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow$
$(V, V', \Gamma, \delta) \vdash_d$ *Inverse* $$ $e \Rightarrow$ ($\lambda\sigma$ *x. f* σ (*op-sem Inverse x*) ∗
inverse (*extract-real x*) ^ *2*)
| *hd-add*: $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f \Longrightarrow$
$(V, V', \Gamma, \delta) \vdash_d$ *Add* $$ $e \Rightarrow$ ($\lambda\sigma$ *z.* $\int^+$*x. f* σ <|*x*, *op-sem Add* <|*z*,
op-sem Minus x|>|>
∂*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* ≡ *stock-measure* (*PRODUCT t1 t2*)
defines *N'* ≡ *stock-measure t1*
defines *fst'* ≡ *op-sem Fst*
defines *f'* ≡ λ*x.* $\int^+$*y. f* <|*x*,*y*|> ∂*stock-measure t2*
assumes *dens*: *has-subprob-density M N f*
shows *has-subprob-density* (*distr M N' 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* = *measurable N*
by (*intro ext measurable-cong-sets*) (*auto dest*: *has-subprob-densityD*)
hence *meas-fst*: *fst'* ∈ *measurable M N'* **unfolding** *fst'-def*
by (*subst op-sem.simps*) (*simp add*: *N'-def N-def M-M*)
thus *subprob-space* (*distr M N' fst'*)
by (*rule subprob-space-distr*) (*simp-all add*: *N'-def*)

interpret *sigma-finite-measure stock-measure t2* **by** *simp*
have *f*[*measurable*]: *f* ∈ *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'* ∈ *borel-measurable N'* **unfolding** *f'-def N'-def*
by *measurable*

let *?M1* = *distr M N' fst'* **and** *?M2* = *density N' f'*
show *sets ?M1* = *sets ?M2* **by** *simp*
fix *X* **assume** *X* ∈ *sets ?M1*
hence *X*: *X* ∈ *sets N' X* ∈ *sets N'* **by** (*simp-all add*: *N'-def*)
then have [*measurable*]: *X* ∈ *sets* (*stock-measure t1*)
by (*simp add*: *N'-def*)

from *meas-fst* **and** *X*(*1*) **have** *emeasure ?M1 X* = *emeasure M* (*fst'* −‘ *X* ∩

space M)
  by (*rule emeasure-distr*)
 also from *dens* **have** *M*: *M* = *density N f* **by** (*rule has-subprob-densityD*)
 from *this* **and** *meas-fst* **have** *meas-fst′*: *fst′* ∈ *measurable N N′* **by** *simp*
 with *dens* **and** *X* **have** *emeasure M* (*fst′* −‘ *X* ∩ *space M*) =
   ∫⁺*x*. *f x* ∗ *indicator* (*fst′* −‘ *X* ∩ *space N*) *x* ∂*N*
  by (*subst* (*1 2*) *M*, *subst space-density*, *subst emeasure-density*)
   (*erule has-subprob-densityD*, *erule measurable-sets*, *simp*, *simp*)
 also have *N* = *distr* (*N′* ⨂ₘ *stock-measure t2*) *N* (*case-prod PairVal*) (**is** - =
?N)
  unfolding *N-def N′-def stock-measure.simps* **by** (*rule embed-measure-eq-distr*)
(*simp add*: *inj-PairVal*)
 hence ⋀*f*. *nn-integral N f* = *nn-integral ... f* **by** *simp*
 also from *dens* **and** *X*
  have (∫⁺*x*. *f x* ∗ *indicator* (*fst′* −‘ *X* ∩ *space N*) *x* ∂*?N*) =
   ∫⁺*x*. *f* (*case-prod PairVal x*) ∗ *indicator* (*fst′* −‘ *X* ∩ *space N*) (*case-prod*
PairVal x)
    ∂(*N′* ⨂ₘ *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 ... = ∫⁺*x*. ∫⁺*y*. *f* <|*x*,*y*|> ∗ *indicator* (*fst′* −‘ *X* ∩ *space N*) <|*x*, *y*|>
∂*stock-measure t2* ∂*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*: ⋀*x y*. *x* ∈ *space N′* ⟹ *y* ∈ *space* (*stock-measure t2*) ⟹
   indicator (*fst′* −‘ *X* ∩ *space N*) <|*x*, *y*|> = *indicator X x*
  by (*auto split*: *split-indicator simp*: *fst′-def N-def*
   space-embed-measure space-pair-measure N′-def)
 have *?I* = ∫⁺*x*. ∫⁺*y*. *f* <|*x*,*y*|> ∗ *indicator X x* ∂*stock-measure t2* ∂*N′* (**is** -
= *?I*)
  by (*intro nn-integral-cong*) (*simp add*: *A*)
 also have *A*: ⋀*x*. *x* ∈ *space N′* ⟹ (λ*y*. *f* <|*x*,*y*|>) = *f* ∘ *case-prod PairVal* ∘
(λ*y*. (*x*,*y*))
  by (*intro ext*) *simp*
 from *dens* **have** *?I* = ∫⁺*x*. (∫⁺*y*. *f* <|*x*,*y*|> ∂*stock-measure t2*) ∗ *indicator X*
x ∂*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** ... = *emeasure ?M2 X* **unfolding** *f′-def*
  by (*rule emeasure-density*[*symmetric*])
 finally show *emeasure ?M1 X* = *emeasure ?M2 X* **.**
qed

lemma *has-subprob-density-distr-Snd*:
 fixes *t1 t2 f*
 defines *N* ≡ *stock-measure* (*PRODUCT t1 t2*)
 defines *N′* ≡ *stock-measure t2*

defines *snd′* ≡ *op-sem Snd*
defines *f′* ≡ λ*y*. ∫⁺*x*. *f* <|*x*,*y*|> ∂*stock-measure t1*
assumes *dens*: *has-subprob-density M N f*
shows *has-subprob-density* (*distr M N′ snd′*) *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* = *measurable N*
by (*intro ext measurable-cong-sets*) (*auto dest*: *has-subprob-densityD*)
hence *meas-snd*: *snd′* ∈ *measurable M N′* **unfolding** *snd′-def*
by (*subst op-sem.simps*) (*simp add*: *N′-def N-def M-M*)
thus *subprob-space* (*distr M N′ snd′*)
by (*rule subprob-space-distr*) (*simp-all add*: *N′-def*)

interpret *t1*: *sigma-finite-measure stock-measure t1* **by** *simp*
have *A*: (λ(*x*, *y*). *f* <| *x* , *y* |>) = *f* ∘ *case-prod PairVal*
by (*intro ext*) (*simp add*: *o-def split*: *prod.split*)
have *f*[*measurable*]: *f* ∈ *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′* ∈ *borel-measurable N′* **unfolding** *f′-def N′-def*
by *measurable*

interpret *N′*: *sigma-finite-measure N′*
unfolding *N′-def* **by** (*rule sigma-finite-stock-measure*)

interpret *N′-t1*: *pair-sigma-finite t1 N′* **proof qed**

let *?M1* = *distr M N′ snd′* **and** *?M2* = *density N′ f′*
show *sets ?M1* = *sets ?M2* **by** *simp*
fix *X* **assume** *X* ∈ *sets ?M1*
hence *X*: *X* ∈ *sets N′ X* ∈ *sets N′* **by** (*simp-all add*: *N′-def*)
then have [*measurable*]: *X* ∈ *sets* (*stock-measure t2*)
by (*simp add*: *N′-def*)

from *meas-snd* **and** *X*(*1*) **have** *emeasure ?M1 X* = *emeasure M* (*snd′* −‘ *X* ∩ *space M*)
by (*rule emeasure-distr*)
also from *dens* **have** *M*: *M* = *density N f* **by** (*rule has-subprob-densityD*)
from *this* **and** *meas-snd* **have** *meas-snd′*: *snd′* ∈ *measurable N N′* **by** *simp*
with *dens* **and** *X* **have** *emeasure M* (*snd′* −‘ *X* ∩ *space M*) =
∫⁺*x*. *f x* ∗ *indicator* (*snd′* −‘ *X* ∩ *space N*) *x* ∂*N*
by (*subst* (*1 2*) *M*, *subst space-density*, *subst emeasure-density*)
(*erule has-subprob-densityD*, *erule measurable-sets*, *simp*, *simp*)
also have *N* = *distr* (*stock-measure t1* $\bigotimes_M$ *N′*) *N* (*case-prod PairVal*) (**is** - = *?N*)
unfolding *N-def N′-def stock-measure.simps* **by** (*rule embed-measure-eq-distr*) (*simp add*: *inj-PairVal*)
hence ⋀*f*. *nn-integral N f* = *nn-integral* ... *f* **by** *simp*
also from *dens* **and** *X*
have (∫⁺*x*. *f x* ∗ *indicator* (*snd′* −‘ *X* ∩ *space N*) *x* ∂*?N*) =

∫ +x. f (case-prod PairVal x) ∗ indicator (snd′ −‘ X ∩ space N) (case-prod PairVal x)
∂(stock-measure t1 ⨂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 ... = ∫ +y. ∫ +x. f <|x,y|> ∗ indicator (snd′ −‘ X ∩ space N) <|x, y|> ∂stock-measure t1 ∂N′
(**is** . . . = ?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: ⋀x y. x ∈ space N′ ⟹ y ∈ space (stock-measure t1) ⟹
indicator (snd′ −‘ X ∩ space N) <|y, x|> = indicator X x
by (auto split: split-indicator simp: snd′-def N-def
space-embed-measure space-pair-measure N′-def)
have ?I = ∫ +y. ∫ +x. f <|x,y|> ∗ indicator X y ∂stock-measure t1 ∂N′ (**is** - = ?I)
by (intro nn-integral-cong) (simp add: A)
also have A: ⋀y. y ∈ space N′ ⟹ (λx. f <|x,y|>) = f ∘ case-prod PairVal ∘ (λx. (x,y))
by (intro ext) simp
from dens **have** ?I = ∫ +y. (∫ +x. f <|x,y|> ∂stock-measure t1) ∗ indicator X y ∂N′
by (intro nn-integral-cong nn-integral-multc) (auto simp: N′-def)
also from meas-f′ **and** X(2) **have** ... = emeasure ?M2 X **unfolding** f′-def
by (rule emeasure-density[symmetric])
finally show emeasure ?M1 X = emeasure ?M2 X **.**
qed

lemma dens-ctxt-measure-empty-bind:
assumes ϱ ∈ space (state-measure V′ Γ)
assumes f[measurable]: f ∈ measurable (state-measure V′ Γ) (subprob-algebra N)
shows dens-ctxt-measure ({},V′,Γ,λ-. 1) ϱ ⤜ f = f ϱ (**is** bind ?M - = ?R)
proof (intro measure-eqI)
from assms **have** nonempty: space ?M ≠ {}
by (auto simp: dens-ctxt-measure-def state-measure′-def state-measure-def space-PiM)
moreover have meas: measurable ?M = measurable (state-measure V′ Γ)
by (intro ext measurable-cong-sets) (auto simp: dens-ctxt-measure-def state-measure′-def)
moreover from assms **have** [simp]: sets (f ϱ) = sets N
by (intro sets-kernel[OF assms(2)])
ultimately show sets-eq: sets (?M ⤜ f) = 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** [simp]: ⋀σ. merge {} V′ (σ,ϱ) = ϱ
by (intro ext) (auto simp: merge-def state-measure-def space-PiM)

fix X **assume** X: X ∈ sets (?M ⤜ f)

hence *emeasure* (*?M* $\gg=$ *f*) *X* = $\int^+ x$. *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 ... = $\int^+$(*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* $\gg=$ *f*) *X* = *emeasure* (*f* ϱ) *X* **.**
qed

lemma (**in** *density-context*) *bind-dens-ctxt-measure-cong*:
assumes *fg*: $\bigwedge\sigma$. ($\bigwedge x$. $x \in V' \Longrightarrow \sigma\ x = \varrho\ x$) $\Longrightarrow f\ \sigma = g\ \sigma$
assumes ϱ[*measurable*]: $\varrho \in$ *space* (*state-measure* V' Γ)
assumes *Mf*[*measurable*]: *f* $\in$ *measurable* (*state-measure* ($V \cup V'$) Γ) (*subprob-algebra N*)
assumes *Mg*[*measurable*]: *g* $\in$ *measurable* (*state-measure* ($V \cup V'$) Γ) (*subprob-algebra N*)
defines $M \equiv$ *dens-ctxt-measure* (V,V',Γ,δ) ϱ
shows $M \gg= f = M \gg= g$
proof −
have [*measurable*]: ($\lambda\sigma$. *merge V V'* (σ, ϱ)) $\in$ *measurable* (*state-measure V* Γ) (*state-measure* ($V \cup V'$) 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* $\subseteq V \cup V'$ **and** *vars2*: *free-vars e'* $\subseteq V'$
assumes ϱ: $\varrho \in$ *space* (*state-measure* V' Γ)
defines $M \equiv$ *dens-ctxt-measure* (V,V',Γ,δ) ϱ

defines *v* ≡ *expr-sem-rf ϱ e′*
shows *M* ≫= (λσ. *expr-sem* σ (*oper* $$ <e, e′>)) =
distr (*M* ≫= (λσ. *expr-sem* σ *e*)) (*stock-measure tr*) (λ*w*. *op-sem oper*
<|*w*,*v*|>)
proof−
from *assms* **have** *vars1′*: ⋀σ. σ ∈ *space M* ⟹ ∀ *x*∈*free-vars e*. *val-type* (σ *x*)
= Γ *x*
and *vars2′*: ⋀σ. σ ∈ *space M* ⟹ ∀ *x*∈*free-vars e′*. *val-type* (σ *x*) = Γ
x
by (*auto simp*: *M-def space-dens-ctxt-measure state-measure-def space-PiM*
dest: *PiE-mem*)
have *Me*: (λσ. *expr-sem* σ *e*) ∈
measurable (*state-measure* (*V* ∪ *V′*) Γ) (*subprob-algebra* (*stock-measure*
t))
by (*rule measurable-expr-sem*[*OF t1 vars1*])

from *assms* **have** *e′*: ⋀σ. σ ∈ *space M* ⟹ *expr-sem* σ *e′* = *return-val* (*expr-sem-rf*
σ *e′*)
by (*intro expr-sem-rf-sound*[*symmetric*]) (*auto simp*: *M-def space-dens-ctxt-measure*)
from *assms* **have** *vt-e′*: ⋀σ. σ ∈ *space M* ⟹ *val-type* (*expr-sem-rf* σ *e′*) = *t′*
by (*intro val-type-expr-sem-rf*) (*auto simp*: *M-def space-dens-ctxt-measure*)

let *?tt′* = *PRODUCT t t′*
{
fix σ **assume** σ: σ ∈ *space M*
with *vars2* **have** [*simp*]: *measurable* (*expr-sem* σ *e′*) = *measurable* (*stock-measure*
t′)
by (*intro measurable-expr-sem-eq*[*OF t2*, *of* - *V*∪*V′*]) (*auto simp*: *M-def*
space-dens-ctxt-measure)
from σ **have** [*simp*]: *space* (*expr-sem* σ *e*) = *space* (*stock-measure t*)
space (*expr-sem* σ *e′*) = *space* (*stock-measure t′*)
using *space-expr-sem*[*OF t1 vars1′*[*OF* σ]] *space-expr-sem*[*OF t2 vars2′*[*OF*
σ]] **by** *simp-all*
have *expr-sem* σ *e* ≫= (λ*x*. *expr-sem* σ *e′* ≫= (λ*y*. *return-val* <| *x* , *y* |>)) =
expr-sem σ *e* ≫= (λ*x*. *return-val* (*expr-sem-rf* σ *e′*) ≫= (λ*y*. *return-val*
<| *x* , *y* |>))
by (*intro bind-cong refl*, *subst e′*[*OF* σ]) *simp*
also have ... = *expr-sem* σ *e* ≫= (λ*x*. *return-val* <|*x* , *expr-sem-rf* σ *e′*|>)
using σ *vars2*
by (*intro bind-cong refl*, *subst bind-return-val′*[*of* - *t′* - *?tt′*])
(*auto simp*: *vt-e′ M-def space-dens-ctxt-measure*
intro!: *measurable-PairVal*)
finally have *expr-sem* σ *e* ≫= (λ*x*. *expr-sem* σ *e′* ≫= (λ*y*. *return-val* <|*x*,*y*|>))
=
expr-sem σ *e* ≫= (λ*x*. *return-val* <|*x*, *expr-sem-rf* σ *e′*|>) **.**
}
hence *M* ≫= (λσ. *expr-sem* σ (*oper* $$ <e, e′>)) =
M ≫= (λσ. (*expr-sem* σ *e* ≫= (λ*x*. *return-val* <|*x*, *expr-sem-rf* σ *e′*|>))
≫= (λ*x*. *return-val* (*op-sem oper x*))) (**is** - = *?T*)

by (*intro bind-cong refl*) (*simp only*: *expr-sem.simps*)
also have [*measurable*]: ⋀σ. σ ∈ *space M* ⟹ *expr-sem-rf* σ *e'* ∈ *space t'*
by (*simp add*: *type-universe-def vt-e' del*: *type-universe-type*)
note [*measurable*] = *measurable-op-sem*[*OF t3*]
hence *?T* = *M* ⤜ (λσ. *expr-sem* σ *e* ⤜ (λ*x*. *return-val* (*op-sem oper* <|*x*, *expr-sem-rf* σ *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*: ⋀σ. (⋀*x*. *x* ∈ *V'* ⟹ σ *x* = ϱ *x*) ⟹ *expr-sem-rf* σ *e'* = *expr-sem-rf* ϱ *e'*
using *vars2* **by** (*intro expr-sem-rf-eq-on-vars*) *auto*
have [*measurable*]: (λσ. *expr-sem-rf* σ *e'*) ∈ *measurable* (*state-measure* (*V* ∪ *V'*) Γ) (*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*: *expr-sem-rf* ϱ *e'* ∈ *space* (*stock-measure t'*)
using *val-type-expr-sem-rf*[*OF t2 rf vars2* ϱ]
by (*simp add*: *type-universe-def del*: *type-universe-type*)
hence *?T* = *M* ⤜ (λσ. *expr-sem* σ *e* ⤜ (λ*x*. *return-val* (*op-sem oper* <|*x*, *expr-sem-rf* ϱ *e'*|>)))
using ϱ **unfolding** *M-def*
by (*intro bind-dens-ctxt-measure-cong*, *subst eq*) (*simp*, *simp*, *simp*, *measurable*)
also have ... = (*M* ⤜ (λσ. *expr-sem* σ *e*)) ⤜
return-val ∘ (λ*x*. *op-sem oper* <|*x*, *expr-sem-rf* ϱ *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 ... = *distr* (*M* ⤜ (λσ. *expr-sem* σ *e*)) (*stock-measure tr*)
(λ*x*. *op-sem oper* <|*x*, *expr-sem-rf* ϱ *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* ∈ *borel-measurable* (*state-measure V'* Γ ⨂$_M$ *stock-measure t*)
assumes *t-disj*: *t* = *REAL* ∨ *t* = *INTEG* **and** *t*: Γ ⊢ *e'* : *t*
assumes *rf*: *randomfree e'* **and** *vars*: *free-vars e'* ⊆ *V'*
defines *f'* ≡ (λϱ *x*. *f* ϱ (*op-sem Add* <|*x*, *expr-sem-rf* ϱ (*Minus* $$ *e'*)|>))
shows *case-prod f'* ∈ *borel-measurable* (*state-measure V'* Γ ⨂$_M$ *stock-measure t*)
proof (*insert t-disj*, *elim disjE*)
assume *A*: *t* = *REAL*
from *A* **and** *t* **have** *t'*: Γ ⊢ *e'* : *REAL* **by** *simp*
with *rf vars* **have** *vt-e'*:
⋀ϱ. ϱ ∈ *space* (*state-measure V'* Γ) ⟹ *val-type* (*expr-sem-rf* ϱ *e'*) = *REAL*

by (*intro val-type-expr-sem-rf*) *simp-all*
let *?f′* = $\lambda\sigma$ *x*. *let c* = *expr-sem-rf* σ *e′*
in f σ (*RealVal* (*extract-real x* − *extract-real c*))
note *Mf*[*unfolded A*, *measurable*] **and** *rf*[*measurable*] **and** *vars*[*measurable*] **and** *t*[*unfolded A*, *measurable*]
have *case-prod ?f′* ∈ *borel-measurable* (*state-measure V′* Γ $\bigotimes_M$ *stock-measure t*)
unfolding *Let-def A* **by** *measurable*
also have *case-prod ?f′* ∈ *borel-measurable* (*state-measure V′* Γ $\bigotimes_M$ *stock-measure t*) ⟷
case-prod f′ ∈ *borel-measurable* (*state-measure V′* Γ $\bigotimes_M$ *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* = *INTEG*
with *t* **have** *t′*: Γ ⊢ *e′* : *INTEG* **by** *simp*
with *rf vars* **have** *vt-e′*:
⋀ϱ. ϱ ∈ *space* (*state-measure V′* Γ) ⟹ *val-type* (*expr-sem-rf* ϱ *e′*) = *INTEG*
by (*intro val-type-expr-sem-rf*) *simp-all*
let *?f′* = $\lambda\sigma$ *x*. *let c* = *expr-sem-rf* σ *e′*
in f σ (*IntVal* (*extract-int x* − *extract-int c*))
note *Mf*[*unfolded A*, *measurable*] **and** *rf*[*measurable*] **and** *vars*[*measurable*] **and** *t*[*unfolded A*, *measurable*]
have *Mdiff*: *case-prod* ((−) :: *int* ⇒ -) ∈
measurable (*count-space UNIV* $\bigotimes_M$ *count-space UNIV*) (*count-space UNIV*) **by** *simp*
have *case-prod ?f′* ∈ *borel-measurable* (*state-measure V′* Γ $\bigotimes_M$ *stock-measure t*)
unfolding *Let-def A* **by** *measurable*
also have *case-prod ?f′* ∈ *borel-measurable* (*state-measure V′* Γ $\bigotimes_M$ *stock-measure t*) ⟷
case-prod f′ ∈ *borel-measurable* (*state-measure V′* Γ $\bigotimes_M$ *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 ϱ: ϱ ∈ *space* (*state-measure V′* Γ)
defines *M* ≡ *dens-ctxt-measure* 𝒴 ϱ
assumes *Mf*[*measurable*]: *f* ∈ *measurable M* (*subprob-algebra* (*stock-measure BOOL*))

assumes *Mg*[*measurable*]: $g \in$ *measurable M* (*subprob-algebra R*)
assumes *Mh*[*measurable*]: $h \in$ *measurable M* (*subprob-algebra R*)
assumes *densf*: *has-parametrized-subprob-density* (*state-measure* $(V \cup V')$ Γ)
f (*stock-measure BOOL*) δf
assumes *densg*: *has-parametrized-subprob-density* (*state-measure* V' Γ)
($\lambda\varrho$. *dens-ctxt-measure* ($V, V', \Gamma, \lambda\sigma.\ \delta\ \sigma * \delta f\ \sigma$ (*BoolVal True*))
$\varrho \gg= g$) *R* δg
assumes *densh*: *has-parametrized-subprob-density* (*state-measure* V' Γ)
($\lambda\varrho$. *dens-ctxt-measure* ($V, V', \Gamma, \lambda\sigma.\ \delta\ \sigma * \delta f\ \sigma$ (*BoolVal False*))
$\varrho \gg= h$) *R* δh
defines $P \equiv \lambda b.\ b =$ *BoolVal True*
shows $M \gg= (\lambda x.\ f\ x \gg= (\lambda b.$ *if P b then g x else h x*$))$ = *density R* $(\lambda x.\ \delta g\ \varrho$ $x + \delta h\ \varrho\ x)$
(**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** $X \in$ *sets ?lhs*
hence *X*: $X \in$ *sets R* **by** (*simp only*: *sets-lhs*)
from *Mf* **have** [*simp*]: $\bigwedge x.\ x \in$ *space M* $\Longrightarrow$ *sets* (*f x*) = *sets* (*stock-measure BOOL*)
by (*rule sets-kernel*)
note [*simp*] = *sets-eq-imp-space-eq*[*OF this*]
from *has-parametrized-subprob-densityD*(*3*)[*OF densf*]
have $M\delta f$[*measurable*]: $(\lambda(x, y).\ \delta f\ x\ y)$
$\in$ *borel-measurable* (*state-measure* $(V \cup V')$ $\Gamma \bigotimes_M$ *stock-measure 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*: *BoolVal True* $\in$ *space* (*stock-measure BOOL*)
BoolVal False $\in$ *space* (*stock-measure BOOL*) **by** *auto*
from *Mg* **have** *Mg'*[*measurable*]: $g \in$ *measurable* (*state-measure* $(V \cup V')$ Γ) (*subprob-algebra R*)
by (*simp add*: *M-def measurable-dens-ctxt-measure-eq*)
from *Mh* **have** *Mh'*[*measurable*]: $h \in$ *measurable* (*state-measure* $(V \cup V')$ Γ) (*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*) δf
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*)+)
[]
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 $\lambda\sigma.$ 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 $\lambda\sigma.$ 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* ($M \gg= (\lambda x.\ f\ x \gg= (\lambda b.$ *if* $P\ b$ *then* $g\ x$ *else* $h\ x$))) X =
$\int^+ x.$ *emeasure* ($f\ x \gg= (\lambda b.$ *if* $P\ b$ *then* $g\ x$ *else* $h\ x$)) X ∂M **using** X
by (*subst emeasure-bind*[*of* - - R], *simp add*: *M-def*, *intro measurable-bind*[*OF Mf*], *measurable*)
also have ... = $\int^+ x.\ \int^+ b.$ *emeasure* (*if* $P\ b$ *then* $g\ x$ *else* $h\ x$) X $\partial f\ x$ ∂M
by (*intro nn-integral-cong*) (*simp-all add*: X *emeasure-bind*[**where** N=R])
also have ... = $\int^+ x.\ \int^+ b.$ *emeasure* (*if* $P\ b$ *then* $g\ x$ *else* $h\ x$) X $*$ δf x b ∂*stock-measure BOOL* ∂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 ... = $\int^+ x.$ *emeasure* ($g\ x$) X $*$ δf x (*BoolVal True*) +
emeasure ($h\ x$) X $*$ δf x (*BoolVal False*) ∂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 ... = ($\int^+ x.$ *emeasure* ($g\ x$) X $*$ δf x (*BoolVal True*) ∂M) +
($\int^+ x.$ *emeasure* ($h\ x$) X $*$ δf x (*BoolVal False*) ∂M) **using** X
using *has-parametrized-subprob-densityD*[*OF densf'*] *BoolVal-in-space*
by (*intro nn-integral-add*) (*auto simp*:)
also have ($\int^+ x.$ *emeasure* ($g\ x$) X $*$ δf x (*BoolVal True*) ∂M) =
$\int^+ x.$ δ (*merge* V V' (x, ϱ)) $*$ δf (*merge* V V' (x, ϱ)) (*BoolVal True*) $*$
(*emeasure* (g (*merge* V V' (x, ϱ)))) X ∂*state-measure* V Γ
using X *has-parametrized-subprob-densityD*[*OF densf*] *BoolVal-in-space* **unfolding** *M-def*
by (*subst nn-integral-dens-ctxt-measure*) (*simp-all add*: ϱ *mult-ac*)
also have ... = *emeasure* (*density* R (δg ϱ)) X **using** ϱ X
apply (*subst dc-True.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 densg*], *simp-all*)
done
also have ($\int^+ x.$ *emeasure* ($h\ x$) X $*$ δf x (*BoolVal False*) ∂M) =
$\int^+ x.$ δ (*merge* V V' (x, ϱ)) $*$ δf (*merge* V V' (x, ϱ)) (*BoolVal False*) $*$
(*emeasure* (h (*merge* V V' (x, ϱ)))) X ∂*state-measure* V Γ
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 ... = *emeasure* (*density R* (*δh ϱ*)) *X* **using** *ϱ 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* (*δg ϱ*)) *X* + *emeasure* (*density R* (*δh ϱ*)) *X* =
emeasure (*density R* (*λx. δg ϱ x* + *δh ϱ x*)) *X* **using** *X ϱ*
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 *ϱ*: *ϱ* ∈ *space* (*state-measure V′* Γ)
defines *M* ≡ *dens-ctxt-measure* $\mathcal{Y}$ *ϱ*
defines *P* ≡ *λb. f b* = *BoolVal True* **and** *P′* ≡ *λb. f b* = *BoolVal False*
assumes *dc1*: *density-context V V′* Γ (*λσ. δ σ* ∗ (*if P σ then 1 else 0*))
assumes *dc2*: *density-context V V′* Γ (*λσ. δ σ* ∗ (*if P′ σ then 1 else 0*))
assumes *Mf*[*measurable*]: *f* ∈ *measurable M* (*stock-measure BOOL*)
assumes *Mg*[*measurable*]: *g* ∈ *measurable M* (*subprob-algebra R*)
assumes *Mh*[*measurable*]: *h* ∈ *measurable M* (*subprob-algebra R*)
assumes *densg*: *has-parametrized-subprob-density* (*state-measure V′* Γ)
(*λϱ. dens-ctxt-measure* (*V*,*V′*,Γ,*λσ. δ σ* ∗ (*if P σ then 1 else 0*)) *ϱ* ≫= *g*) *R δg*
assumes *densh*: *has-parametrized-subprob-density* (*state-measure V′* Γ)
(*λϱ. dens-ctxt-measure* (*V*,*V′*,Γ,*λσ. δ σ* ∗ (*if P′ σ then 1 else 0*)) *ϱ* ≫= *h*) *R δh*
shows *M* ≫= (*λx. if P x then g x else h x*) = *density R* (*λx. δg ϱ x* + *δh ϱ 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* ∈ *measurable* (*state-measure* (*V* ∪ *V′*) Γ) (*subprob-algebra R*)
by (*simp add*: *M-def measurable-dens-ctxt-measure-eq*)
from *Mh* **have** *Mh′*[*measurable*]: *h* ∈ *measurable* (*state-measure* (*V* ∪ *V′*) Γ) (*subprob-algebra R*)
by (*simp add*: *M-def measurable-dens-ctxt-measure-eq*)
have [*simp*]: ⋀*x. x* ∈ *space M* ⟹ *sets* (*g x*) = *sets R*

by (*rule sets-kernel*[*OF Mg*])
have [*simp*]: $\bigwedge x.\ x \in$ *space M* $\Longrightarrow$ *sets* (*h x*) = *sets R*
by (*rule sets-kernel*[*OF Mh*])
have [*simp*]: *sets M* = *sets* (*state-measure* ($V \cup V'$) Γ)
by (*simp add*: *M-def dens-ctxt-measure-def state-measure'-def*)
then have [*measurable-cong*]: *sets* (*state-measure* ($V \cup V'$) Γ) = *sets M* **..**
have [*simp*]: *range BoolVal* = {*BoolVal True*, *BoolVal False*} **by** *auto*

fix *X* **assume** $X \in$ *sets ?lhs*
hence *X*[*measurable*]: $X \in$ *sets R* **by** (*simp only*: *sets-lhs*)

interpret *dc-True*: *density-context V V'* Γ $\lambda\sigma.\ \delta\ \sigma * $ (*if P* σ *then 1 else 0*) **by** *fact*
interpret *dc-False*: *density-context V V'* Γ $\lambda\sigma.\ \delta\ \sigma * $ (*if P'* σ *then 1 else 0*) **by** *fact*

have *emeasure* (*M* $\gg=$ ($\lambda x.$ *if P x then g x else h x*)) *X* =
$\int^+ x.$ (*if P x then emeasure* (*g x*) *X else emeasure* (*h x*) *X*) ∂M **using** *X*
by (*subst emeasure-bind*[*of - - R*], *simp add*: *M-def*, *measurable*)
(*intro nn-integral-cong*, *simp*)
also have ... = $\int^+ x.$ (*if P x then 1 else 0*) $*$ *emeasure* (*g x*) *X* +
(*if P' x then 1 else 0*) $*$ *emeasure* (*h x*) *X* ∂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 ... = ($\int^+ x.$ (*if P x then 1 else 0*) $*$ *emeasure* (*g x*) *X* ∂M) +
($\int^+ x.$ (*if P' x then 1 else 0*) $*$ *emeasure* (*h x*) *X* ∂M) **using** *X*
by (*intro nn-integral-add*) (*simp-all add*:)
also have ... = ($\int^+ y.\ \delta g\ \varrho\ y *$ *indicator X y* ∂R) + ($\int^+ y.\ \delta h\ \varrho\ y *$ *indicator X y* ∂R)
unfolding *M-def* **using** ϱ *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 ... = *emeasure* (*density R* ($\lambda x.\ \delta g\ \varrho\ x + \delta h\ \varrho\ x$)) *X* **using** *X* ϱ
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*]:
$(\lambda x.\ restrict\ x\ V') \in measurable\ (state\text{-}measure\ (V \cup V')\ \Gamma)\ (state\text{-}measure\ V'\ \Gamma)$
by (*simp add*: *state-measure-def*)

lemma *expr-has-density-sound-op*:
assumes *dens-ctxt*: $density\text{-}context\ V\ V'\ \Gamma\ \delta$
assumes *dens*: $has\text{-}parametrized\text{-}subprob\text{-}density\ (state\text{-}measure\ V'\ \Gamma)$
$(\lambda \varrho.\ dens\text{-}ctxt\text{-}measure\ (V,V',\Gamma,\delta)\ \varrho \gg= (\lambda \sigma.\ expr\text{-}sem\ \sigma\ e))$ $(stock\text{-}measure\ t)\ f$
assumes *Mg*: $case\text{-}prod\ g \in borel\text{-}measurable\ (state\text{-}measure\ V'\ \Gamma \bigotimes_M stock\text{-}measure\ t')$
assumes *dens'*: $\bigwedge M\ \varrho.\ has\text{-}subprob\text{-}density\ M\ (stock\text{-}measure\ t)\ (f\ \varrho) \Longrightarrow$
$has\text{-}density\ (distr\ M\ (stock\text{-}measure\ t')\ (op\text{-}sem\ oper))$
$(stock\text{-}measure\ t')\ (g\ \varrho)$
assumes *t1*: $\Gamma \vdash e : t$ **and** *t2* : $op\text{-}type\ oper\ t = Some\ t'$
assumes *free-vars*: $free\text{-}vars\ (oper\ \$\$\ e) \subseteq V \cup V'$
shows $has\text{-}parametrized\text{-}subprob\text{-}density\ (state\text{-}measure\ V'\ \Gamma)$
$(\lambda \varrho.\ dens\text{-}ctxt\text{-}measure\ (V,V',\Gamma,\delta)\ \varrho \gg= (\lambda \sigma.\ expr\text{-}sem\ \sigma\ (oper\ \$\$\ e)))$ $(stock\text{-}measure\ t')\ g$
proof−
interpret $density\text{-}context\ V\ V'\ \Gamma\ \delta$ **by** *fact*
show *?thesis* **unfolding** *has-parametrized-subprob-density-def*
proof (*intro conjI ballI impI*)
show $case\text{-}prod\ g \in borel\text{-}measurable\ (state\text{-}measure\ V'\ \Gamma \bigotimes_M stock\text{-}measure\ t')$ **by** *fact*

fix ϱ **assume** ϱ: $\varrho \in space\ (state\text{-}measure\ V'\ \Gamma)$
let $?M = dens\text{-}ctxt\text{-}measure\ (V,V',\Gamma,\delta)\ \varrho$
have *Me*: $(\lambda \sigma.\ expr\text{-}sem\ \sigma\ e) \in measurable\ ?M\ (subprob\text{-}algebra\ (stock\text{-}measure\ t))$
by (*subst measurable-dens-ctxt-measure-eq*)
(*insert assms t1*, *auto intro*!: *measurable-expr-sem*)
from *dens* **and** ϱ **have** *dens*: $has\text{-}subprob\text{-}density\ (?M \gg= (\lambda \sigma.\ expr\text{-}sem\ \sigma\ e))$ $(stock\text{-}measure\ t)\ (f\ \varrho)$
unfolding *has-parametrized-subprob-density-def* **by** *auto*
have $has\text{-}subprob\text{-}density\ (distr\ (?M \gg= (\lambda \sigma.\ expr\text{-}sem\ \sigma\ e))\ (stock\text{-}measure\ t')\ (op\text{-}sem\ oper))$
$(stock\text{-}measure\ t')\ (g\ \varrho)$ (**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** $has\text{-}density\ ?N\ (stock\text{-}measure\ t')\ (g\ \varrho)$

by (*intro dens'*) (*simp add*: *has-subprob-density-def*)
qed
also from *assms* **and** ϱ
have *?N* = *?M* $\gg=$ ($\lambda\sigma$. *expr-sem* σ (*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' Γ δ *free-vars* $e \subseteq V \cup V'$
shows *has-parametrized-subprob-density* (*state-measure* V' Γ)
($\lambda\varrho$. *do* {$\sigma \leftarrow$ *dens-ctxt-measure* (V,V',Γ,δ) ϱ; *expr-sem* σ e})
(*stock-measure* t)
($\lambda\varrho$ x. f ϱ x)
using *assms*
proof (*induction arbitrary*: *t rule*: *expr-has-density.induct*[*split-format* (*complete*)])
case (*hd-AE* V V' Γ δ e f t f' t')
from ‹$\Gamma \vdash e : t'$› **and** ‹$\Gamma \vdash e : t$› **have** *t*[*simp*]: $t' = t$
by (*rule expr-typing-unique*)
have *has-parametrized-subprob-density* (*state-measure* V' Γ)
($\lambda\varrho$. *dens-ctxt-measure* (V, V', Γ, δ) $\varrho \gg= (\lambda\sigma.$ *expr-sem* σ 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' Γ δ e f δ' t)
interpret *dc'*: *density-context* V V' Γ δ' **by** *fact*
from *hd-dens-ctxt-cong.hyps* **and** *dc'.measurable-dens*
have [*simp*]: $\delta \in$ *borel-measurable* (*state-measure* $(V \cup V')$ Γ)
by (*erule-tac subst*[*OF measurable-cong*, *rotated*]) *simp*
hence *density-context* V V' Γ δ
by (*intro density-context-equiv*[*OF hd-dens-ctxt-cong.hyps*(*2*)[*symmetric*]])
(*insert hd-dens-ctxt-cong.prems hd-dens-ctxt-cong.hyps*, *simp-all*)
hence *has-parametrized-subprob-density* (*state-measure* V' Γ)
($\lambda\varrho$. *dens-ctxt-measure* (V, V', Γ, δ) $\varrho \gg= (\lambda\sigma.$ *expr-sem* σ e)) (*stock-measure*
t) *f* (**is** *?P*)
using *hd-dens-ctxt-cong.prems hd-dens-ctxt-cong.hyps*
by (*intro hd-dens-ctxt-cong.IH*) *simp-all*
also have $\bigwedge\sigma$. $\sigma \in$ *space* (*state-measure* V' Γ) $\Longrightarrow$
dens-ctxt-measure (V, V', Γ, δ') σ = *dens-ctxt-measure* (V, V', Γ, δ)
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* = *val-type v* **by** *auto*
  interpret *density-context V V′ Γ δ* **by** *fact*
  show *?case*
  proof (*rule has-parametrized-subprob-densityI*)
    show (λ(ϱ, *y*). *branch-prob* (*V*,*V′*,Γ,δ) ϱ ∗ *indicator* {*v*} *y*) ∈
            borel-measurable (*state-measure V′* Γ ⨂$_M$ *stock-measure t*)
      by (*subst measurable-split-conv*)
       (*auto intro*!: *measurable-compose*[*OF measurable-snd borel-measurable-indicator*]
                 borel-measurable-times-ennreal)
    fix ϱ **assume** ϱ: ϱ ∈ *space* (*state-measure V′* Γ)
    have *return-probspace*: *prob-space* (*return-val v*) **unfolding** *return-val-def*
      by (*simp add*: *prob-space-return*)
    thus *subprob-space* (*dens-ctxt-measure* (*V*,*V′*,Γ,δ) ϱ ⤜ (λσ. *expr-sem* σ (*Val v*))) **using** ϱ
      by (*auto simp*: *return-val-def*
              intro!: *measurable-compose*[*OF measurable-const return-measurable*] *subprob-space-bind*
                    subprob-space-dens hd-val.prems)
    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* 𝒴 ϱ ⤜ (λσ. *expr-sem* σ (*Val v*)) =
            density (*stock-measure t*) (λ*x*. *branch-prob* 𝒴 ϱ ∗ *indicator* {*v*} *x*)
        by (*subst expr-sem.simps*, *subst dens-ctxt-measure-bind-const*, *insert return-probspace*)
         (*auto simp*: *return-val-def return-count-space-eq-density* ϱ
                density-density-eq field-simps intro!: *prob-space-imp-subprob-space*)
  qed

next
  case (*hd-var x V V′ Γ δ t*)
  hence *t*: *t* = Γ *x* **by** *auto*
  interpret *density-context V V′ Γ δ* **by** *fact*
  from *hd-var* **have** *x* ∈ *V* **by** *simp*
  show *?case*
  proof (*rule has-parametrized-subprob-densityI*)
    fix ϱ **assume** ϱ: ϱ ∈ *space* (*state-measure V′* Γ)
    have *subprob-space* (*dens-ctxt-measure* 𝒴 ϱ ⤜ (λσ. *return* (*stock-measure t*) (σ *x*)))
      (**is** *subprob-space* (*?M* ⤜ *?f*)) **using** *hd-var* ϱ
      by (*intro subprob-space-bind*)
        (*auto simp*: *return-val-def t intro*!: *subprob-space-bind subprob-space-dens*
                measurable-compose[*OF measurable-dens-ctxt-measure-component return-measurable*])
    also from *hd-var.hyps* **have** *?M* ⤜ *?f* = *?M* ⤜ (λσ. *return-val* (σ *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* ⋙ (*λσ. expr-sem σ* (*Var x*))) **by** *simp*

from *hd-var* **interpret** *dcm*: *subprob-space dens-ctxt-measure 𝒴 ϱ*
by (*intro subprob-space-dens ϱ*)
let *?M1* = *dens-ctxt-measure 𝒴 ϱ* ⋙ (*λσ. expr-sem σ* (*Var x*))
let *?M2* = *density* (*stock-measure t*) (*λv. marg-dens 𝒴 x ϱ v*)
have ∀ *σ*∈*space* (*dens-ctxt-measure 𝒴 ϱ*). *val-type* (*σ x*) = *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 𝒴 ϱ* ⋙ (*return* (*stock-measure t*) ∘ (*λσ. σ x*))
by (*intro bind-cong-All*) (*simp add*: *return-val-def*)
also have ... = *distr* (*dens-ctxt-measure 𝒴 ϱ*) (*stock-measure t*) (*λσ. σ 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* ‹*x* ∈ *V*›]
by (*simp add*: *t hd-var.prems ϱ*)
finally show *?M1* = *?M2* **.**
qed (*auto intro*!: *measurable-marg-dens′ simp*: *hd-var t*)

next
case (*hd-let V V′* Γ *e1 f δ e2 g t*)
let *?t* = *the* (*expr-type* Γ *e1*)
let *?*Γ*′* = *case-nat ?t* Γ **and** *?δ′* = *insert-dens V V′ f δ*
let *?𝒴′* = (*shift-var-set V*, *Suc‘V′*, *?*Γ*′*, *?δ′*)
from *hd-let.prems* **have** *t1*: Γ ⊢ *e1* : *?t* **and** *t2*: *?*Γ*′* ⊢ *e2* : *t*
by (*auto simp*: *expr-type-Some-iff*[*symmetric*] *split*: *option.split-asm*)
interpret *dc*: *density-context V V′* Γ *δ* **by** *fact*

show *?case* **unfolding** *has-parametrized-subprob-density-def*
proof (*intro ballI conjI*)
have *density-context* {} (*V* ∪ *V′*) Γ (*λa. 1*) **by** (*rule dc.density-context-empty*)
moreover note *hd-let.prems*
ultimately have *has-parametrized-subprob-density* (*state-measure* (*V* ∪ *V′*) Γ)
(*λϱ. dens-ctxt-measure* ({}, *V*∪*V′*,Γ,*λa. 1*) *ϱ* ⋙ (*λσ. expr-sem σ e1*))
(*stock-measure ?t*) *f* (**is** *?P*)
by (*intro hd-let.IH*(*1*)) (*auto intro*!: *t1*)
also have *?P* ⟷ *has-parametrized-subprob-density* (*state-measure* (*V* ∪ *V′*) Γ)
(*λσ. expr-sem σ e1*) (*stock-measure ?t*) *f* **using** *hd-let.prems*
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*∪*V′*) Γ)
(*λϱ. expr-sem ϱ e1*) (*stock-measure ?t*) *f* **.**
have *g*: *has-parametrized-subprob-density* (*state-measure* (*Suc‘V′*) *?*Γ*′*)

$(\lambda\varrho.$ *dens-ctxt-measure* $?\mathcal{Y}'\ \varrho \gg= (\lambda\sigma.$ *expr-sem* σ *e2*$))$ (*stock-measure* *t*) *g*
using *hd-let.prems 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 $(\lambda(\varrho, x).\ g$ (*case-nat undefined* ϱ) $x) \in$ *borel-measurable* (*state-measure* $V'\ \Gamma \bigotimes_M$ *stock-measure t*)
by *simp*

fix ϱ **assume** ϱ: $\varrho \in$ *space* (*state-measure* $V'\ \Gamma$)
let *?M* = *dens-ctxt-measure* $(V,\ V',\ \Gamma,\ \delta)\ \varrho$ **and**
?N = *state-measure* (*shift-var-set* $(V \cup V')$) $?\Gamma'$
have *M-dcm*: *measurable ?M* = *measurable* (*state-measure* $(V \cup V')\ \Gamma$)
by (*intro ext measurable-cong-sets*)
(*auto simp*: *dens-ctxt-measure-def state-measure-def state-measure′-def*)
have *M-dcm′*: $\bigwedge N$. *measurable* $(?M \bigotimes_M N)$ = *measurable* (*state-measure* $(V \cup V')\ \Gamma \bigotimes_M N$)
by (*intro ext measurable-cong-sets*)
(*auto simp*: *dens-ctxt-measure-def state-measure-def state-measure′-def*)
have $?M \gg= (\lambda\sigma.$ *expr-sem* σ (*LetVar e1 e2*)) =
do $\{\sigma \leftarrow ?M;\ y \leftarrow$ *expr-sem* σ *e1*; *return ?N* (*case-nat* $y\ \sigma$)$\} \gg= (\lambda\sigma.$ *expr-sem* σ *e2*)
(**is** - = *bind ?R* -)
using *hd-let.prems 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.prems*
have $(\lambda\sigma.$ *expr-sem* σ *e1*) $\in$
measurable (*state-measure* $(V \cup V')\ \Gamma$) (*subprob-algebra* (*stock-measure* *?t*))
by (*intro measurable-expr-sem*) *auto*
hence *?R* = *dens-ctxt-measure* $?\mathcal{Y}'$ (*case-nat undefined* ϱ) **using** *hd-let.prems hd-let.hyps f* ϱ
by (*intro dc.dens-ctxt-measure-insert*) (*auto simp*: *has-parametrized-subprob-density-def*)
also have *case-nat undefined* $\varrho \in$ *space* (*state-measure* (*Suc*‘V') $?\Gamma'$)
by (*rule measurable-space*[*OF measurable-case-nat-undefined* ϱ])
with *g* **have** *has-subprob-density* (*dens-ctxt-measure* $?\mathcal{Y}'$ (*case-nat undefined* ϱ) $\gg=$
$(\lambda\sigma.$ *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* $(?M \gg= (\lambda\sigma.$ *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*]: Γ ⊢ *e* : *dist-param-type dst randomfree e free-vars e* ⊆ *V'*
by *auto*

interpret *density-context V V' Γ δ* **by** *fact*
from *hd-rand-det* **have** *t*: *t* = *dist-result-type dst* **by** *auto*

{
fix ϱ **assume** ϱ: ϱ ∈ *space* (*state-measure V' Γ*)
let *?M* = *dens-ctxt-measure* (*V*,*V'*,Γ,δ) ϱ **and** *?t* = *dist-param-type dst*
have *?M* ⤜ (λσ. *expr-sem* σ (*Random dst e*)) =
?M ⤜ (λσ. *return-val* (*expr-sem-rf* σ *e*) ⤜ *dist-measure 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*: ⋀σ. σ ∈ *space ?M* ⟹ *val-type* (*expr-sem-rf* σ *e*) = *?t*
by (*intro val-type-expr-sem-rf*) (*auto simp*: *dens-ctxt-measure-def state-measure'-def*)
hence *?N* = *?M* ⤜ (λσ. *return* (*stock-measure ?t*) (*expr-sem-rf* σ *e*) ⤜ *dist-measure 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* ⤜ (λσ. *dist-measure dst* (*expr-sem-rf* σ *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* ⤜ (λσ. *expr-sem* σ (*Random dst e*)) =
?M ⤜ (λσ. *dist-measure dst* (*expr-sem-rf* σ *e*)) **.**
} **note** *A* = *this*

have *has-parametrized-subprob-density* (*state-measure V' Γ*)
(λϱ. *dens-ctxt-measure* 𝒴 ϱ ⤜ (λσ. *dist-measure dst* (*expr-sem-rf* σ *e*)))
(*stock-measure t*) (λϱ *x*. *branch-prob* 𝒴 ϱ ∗ *dist-dens dst* (*expr-sem-rf* ϱ *e*) *x*)
proof (*unfold has-parametrized-subprob-density-def*, *intro conjI ballI*)
show *M*: (λ(ϱ, *v*). *branch-prob* 𝒴 ϱ ∗ *dist-dens dst* (*expr-sem-rf* ϱ *e*) *v*)
∈ *borel-measurable* (*state-measure V' Γ* $\bigotimes_M$ *stock-measure t*)
by (*subst t*) *measurable*

fix ϱ **assume** ϱ: ϱ ∈ *space* (*state-measure V' Γ*)
let *?M* = *dens-ctxt-measure* (*V*,*V'*,Γ,δ) ϱ **and** *?t* = *dist-param-type dst*
have *?M* ⤜ (λσ. *expr-sem* σ (*Random dst e*)) =
?M ⤜ (λσ. *return-val* (*expr-sem-rf* σ *e*) ⤜ *dist-measure 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*: $\bigwedge\sigma.\ \sigma \in$ *space ?M* $\Longrightarrow$ *val-type* (*expr-sem-rf* σ *e*) = *?t*
by (*intro val-type-expr-sem-rf*) (*auto simp*: *dens-ctxt-measure-def state-measure′-def*)
hence *?N* = *?M* $\gg=$ ($\lambda\sigma$. *return* (*stock-measure ?t*) (*expr-sem-rf* σ *e*) $\gg=$ *dist-measure 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* $\gg=$ ($\lambda\sigma$. *dist-measure dst* (*expr-sem-rf* σ *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* $\gg=$ ($\lambda\sigma$. *dist-measure dst* (*expr-sem-rf* σ *e*))) (*stock-measure t*)
(λv. $\int^{+}\sigma$. *dist-dens dst* (*expr-sem-rf* (*restrict* σ V') *e*) *v* ∂*?M*)
(**is** *has-subprob-density ?N ?R ?f*)
proof (*rule bind-has-subprob-density*)
show *space ?M* $\neq$ {} **unfolding** *dens-ctxt-measure-def state-measure′-def state-measure-def*
by (*auto simp*: *space-PiM PiE-eq-empty-iff*)
show ($\lambda\sigma$. *dist-measure dst* (*expr-sem-rf* σ *e*)) $\in$ *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 $\lambda(x, y)$. *dist-dens dst* (*expr-sem-rf* (*restrict x* V') *e*) *y*)
$\in$ *borel-measurable* (*?M* $\bigotimes_M$ *stock-measure t*)
unfolding *t* **by** *measurable*
fix σ **assume** σ: $\sigma \in$ *space ?M*
hence σ': *restrict* σ $V' \in$ *space* (*state-measure* V' Γ)
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* σ V') *e* = *expr-sem-rf* σ *e*
by (*intro expr-sem-rf-eq-on-vars*) *auto*
from *hd-rand-det* **have** *val-type* (*expr-sem-rf* (*restrict* σ V') *e*) = *dist-param-type dst*
by (*auto intro*!: *val-type-expr-sem-rf*[*OF - - - σ'*])
also note *restr*
finally have *has-density* (*dist-measure dst* (*expr-sem-rf* σ *e*)) (*stock-measure t*)
(*dist-dens dst* (*expr-sem-rf* σ *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* σ *e*)) (*stock-measure t*)

($dist\text{-}dens\ dst\ (expr\text{-}sem\text{-}rf\ (restrict\ \sigma\ V')\ e)$) **by** (*simp add*: *restr*)
qed (*insert* ϱ, *auto intro*!: *subprob-space-dens*)
moreover have *val-type* ($expr\text{-}sem\text{-}rf\ \varrho\ e$) = *dist-param-type dst* **using** *hd-rand-det* ϱ
by (*intro val-type-expr-sem-rf*) *auto*
hence $expr\text{-}sem\text{-}rf\ \varrho\ e \in type\text{-}universe\ (dist\text{-}param\text{-}type\ dst)$
by (*simp add*: *type-universe-def del*: *type-universe-type*)
ultimately show *has-subprob-density* ($?M \gg= (\lambda\sigma.\ dist\text{-}measure\ dst\ (expr\text{-}sem\text{-}rf\ \sigma\ e))$)
($stock\text{-}measure\ t$) ($\lambda v.\ branch\text{-}prob\ \mathcal{Y}\ \varrho * dist\text{-}dens\ dst\ (expr\text{-}sem\text{-}rf\ \varrho\ e)\ v$)
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*: $t\ \varrho$)
done
qed
with *A* **show** *?case* **by** (*subst has-parametrized-subprob-density-cong*) (*simp-all add*: *A*)

next
case ($hd\text{-}rand\ V\ V'\ \Gamma\ \delta\ e\ f\ dst\ t$)
let $?t = dist\text{-}param\text{-}type\ dst$
from *hd-rand.prems* **have** $t1$: $\Gamma \vdash e : ?t$ **and** $t2$: $t = dist\text{-}result\text{-}type\ dst$ **by** *auto*
interpret *density-context* $V\ V'\ \Gamma\ \delta$ **by** *fact*
have *dens*[*measurable*]: *has-parametrized-subprob-density* ($state\text{-}measure\ V'\ \Gamma$)
($\lambda\varrho.\ dens\text{-}ctxt\text{-}measure\ (V,\ V',\ \Gamma,\ \delta)\ \varrho \gg= (\lambda\sigma.\ expr\text{-}sem\ \sigma\ e)$) ($stock\text{-}measure\ ?t$) *f*
using *hd-rand.prems* **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*) $\in$
borel-measurable ($state\text{-}measure\ V'\ \Gamma \bigotimes_M stock\text{-}measure\ t$)
unfolding *apply-dist-to-dens-def t2* **by** *measurable*

fix ϱ **assume** ϱ: $\varrho \in space\ (state\text{-}measure\ V'\ \Gamma)$
let $?M = dens\text{-}ctxt\text{-}measure\ (V,\ V',\ \Gamma,\ \delta)\ \varrho$
have *meas-M*: *measurable* $?M$ = *measurable* ($state\text{-}measure\ (V \cup V')\ \Gamma$)
by (*intro ext measurable-cong-sets*) (*auto simp*: *dens-ctxt-measure-def state-measure'-def*)
from *hd-rand* **have** *Me*: $(\lambda\sigma.\ expr\text{-}sem\ \sigma\ e) \in measurable\ ?M\ (subprob\text{-}algebra\ (stock\text{-}measure\ ?t))$
by (*subst meas-M*, *intro measurable-expr-sem*[*OF t1*]) *auto*
hence $?M \gg= (\lambda\sigma.\ expr\text{-}sem\ \sigma\ (Random\ dst\ e)) = (?M \gg= (\lambda\sigma.\ expr\text{-}sem\ \sigma\ e)) \gg= dist\text{-}measure\ dst$
(**is** - = $?N$)
by (*subst expr-sem.simps*, *intro bind-assoc*[*OF Me*, *symmetric*])
(*insert hd-rand*, *auto intro*!: *measurable-dist-measure*)
also have *space* $?M \neq \{\}$

```
      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 ⪢= (λσ. expr-sem σ (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)
          (λ-. null-measure (stock-measure t)) (stock-measure t') (λ- -. 0) (is ?P)
    using hd-fail by (auto simp: has-parametrized-subprob-density-def
                        intro!: null-measure-has-subprob-density)
  also have ?P ⟷ ?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.prems have t: t = PRODUCT (Γ x) (Γ y) by auto

  have meas: (λσ. <|σ x, σ y|>) ∈ measurable (state-measure (V∪V') Γ) ?R
    using hd-pair unfolding t state-measure-def by simp

  have has-parametrized-subprob-density (state-measure V' Γ)
          (λϱ. distr (dens-ctxt-measure (V, V', Γ, δ) ϱ) ?R (λσ. <|σ x, σ y|>))
          (stock-measure t) (marg-dens2 𝒴 x y)
  proof (rule has-parametrized-subprob-densityI)
    fix ϱ assume ϱ: ϱ ∈ space (state-measure V' Γ)
    let ?M = dens-ctxt-measure (V, V', Γ, δ) ϱ
     from hd-pair.hyps ϱ show distr ?M ?R (λσ. <|σ x , σ y |>) = density ?R
(marg-dens2 𝒴 x y ϱ)
      by (subst (1 2) t, rule density-marg-dens2-eq[symmetric])
    from ϱ interpret subprob-space ?M by (rule subprob-space-dens)
    show subprob-space (distr (dens-ctxt-measure (V,V',Γ,δ) ϱ) ?R (λσ. <|σ x ,σ
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 (λϱ. *distr* (*dens-ctxt-measure* (V, V′, Γ, δ) ϱ) *?R* (λσ. <|σ x, σ y|>)) =
(λϱ. *dens-ctxt-measure* (V, V′, Γ, δ) ϱ ⤜ (λσ. *return-val* <|σ x, σ y|>))
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.prems* **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)
(λϱ. *dens-ctxt-measure* ({},V∪V′,Γ,λa. 1) ϱ ⤜ (λσ. *expr-sem* σ b)) (*stock-measure BOOL*) f
(**is** *?P*) **using** *hd-if.prems tb* **by** (*intro hd-if.IH*(1)) *auto*
also have *?P* ⟷ *has-parametrized-subprob-density* (*state-measure* (V ∪ V′) Γ)
(λσ. *expr-sem* σ b) (*stock-measure BOOL*) f (**is** - = *?P*) **using** *hd-if.prems*
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* = λϱ. *dens-ctxt-measure* (V,V′,Γ,δ) ϱ
let *?M′* = λb 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)
(λϱ. *?M′ True* ϱ ⤜ (λσ. *expr-sem* σ e1)) (*stock-measure* t) g1 **using** *hd-if.prems*
by (*intro hd-if.IH*(2)[*OF t1 dc′*]) *simp*
have *g2*[*measurable*]: *has-parametrized-subprob-density* (*state-measure* V′ Γ)
(λϱ. *?M′ False* ϱ ⤜ (λσ. *expr-sem* σ e2)) (*stock-measure* t) g2 **using** *hd-if.prems*
by (*intro hd-if.IH*(3)[*OF t2 dc′*]) *simp*

show *?case*
proof (*rule has-parametrized-subprob-densityI*)
show (λ(ϱ, x). g1 ϱ x + g2 ϱ x) ∈ *borel-measurable* (*state-measure* V′ Γ $\bigotimes_M$ *stock-measure* t)
by *measurable*
fix ϱ **assume** ϱ: ϱ ∈ *space* (*state-measure* V′ Γ)
show *subprob-space* (*?M* ϱ ⤜ (λσ. *expr-sem* σ (*IF* b *THEN* e1 *ELSE* e2)))
using ϱ *hd-if.prems*
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*

```
            simp del: expr-sem.simps)
    show ?M ϱ ⤜ (λσ. expr-sem σ (IF b THEN e1 ELSE e2)) =
            density (stock-measure t) (λx. g1 ϱ x + g2 ϱ x) using ϱ hd-if.prems f
g1 g2
    by (subst expr-sem.simps, intro dc.emeasure-bind-if-dens-ctxt-measure)
      (auto simp: measurable-dens-ctxt-measure-eq if-dens-def
            simp del: stock-measure.simps intro!: measurable-expr-sem)
  qed

next
  case (hd-if-det b V V' Γ δ e1 g1 e2 g2 t)
  interpret dc: density-context V V' Γ δ by fact
  from hd-if-det.prems ‹randomfree b›
  have tb[measurable (raw)]: Γ ⊢ b : BOOL and [measurable (raw)]: randomfree b
    and t1[measurable (raw)]: Γ ⊢ e1 : t
    and t2[measurable (raw)]: Γ ⊢ e2 : t
    and fv-b[measurable (raw)]: free-vars b ⊆ V ∪ V'
    and fv-e1[measurable (raw)]: free-vars e1 ⊆ V ∪ V'
    and fv-e2[measurable (raw)]: free-vars e2 ⊆ V ∪ V' by auto
  let ?M = λϱ. dens-ctxt-measure (V,V',Γ,δ) ϱ
  let ?M' = λx ϱ. dens-ctxt-measure (V,V',Γ,if-dens-det δ b x) ϱ
  let ?N = λϱ. ?M ϱ⤜ (λσ. if expr-sem-rf σ b = BoolVal True then expr-sem σ
e1 else expr-sem σ e2)

  from hd-if-det.hyps hd-if-det.prems tb
    have dc': ⋀x. density-context V V' Γ (if-dens-det δ b x)
    by (intro dc.density-context-if-dens-det) simp-all
  have g1[measurable]: has-parametrized-subprob-density (state-measure V' Γ)
           (λϱ. ?M' True ϱ ⤜ (λσ. expr-sem σ e1)) (stock-measure t) g1 using
hd-if-det.prems
    by (intro hd-if-det.IH(1)[OF]) (simp-all add: dc' t1)
  have g2[measurable]: has-parametrized-subprob-density (state-measure V' Γ)
           (λϱ. ?M' False ϱ ⤜ (λσ. expr-sem σ e2)) (stock-measure t) g2 using
hd-if-det.prems
    by (intro hd-if-det.IH(2)[OF]) (simp-all add: dc' t2)

  note val-type-expr-sem-rf[OF tb, of V ∪ V', simp]

  have has-parametrized-subprob-density (state-measure V' Γ) ?N
          (stock-measure t) (λa b. g1 a b + g2 a b) (is ?P)
  proof (rule has-parametrized-subprob-densityI)
    show (λ(ϱ, x). g1 ϱ x + g2 ϱ x) ∈ borel-measurable (state-measure V' Γ ⨂M
stock-measure t)
      by measurable
    fix ϱ assume ϱ: ϱ ∈ space (state-measure V' Γ)
    show subprob-space (?N ϱ)
      using ϱ hd-if-det.prems hd-if-det.hyps t1 t2
     by (intro subprob-space-bind[of - - stock-measure t]) (auto simp add: dc.subprob-space-dens)
    show ?N ϱ = density (stock-measure t) (λx. g1 ϱ x + g2 ϱ x)
```

using ϱ *hd-if-det.prems g1 g2 dc′ hd-if-det.prems* **unfolding** *if-dens-det-def*
by (*intro dc.emeasure-bind-if-det-dens-ctxt-measure*)
(*simp-all add*: *space-dens-ctxt-measure*)
qed
also from *hd-if-det.prems 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*=*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.prems* **obtain** t' **where** *t*: $\Gamma \vdash e : PRODUCT\ t\ t'$
by (*elim expr-typing-opE*) (*auto split*: *pdf-type.split-asm*)
hence $\Gamma \vdash$ *Snd* $$ *e* : t' **by** (*intro expr-typing.intros*) *auto*
hence *t2*: *the* (*expr-type* Γ (*Snd* $$ *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* $V'\ \Gamma$)
($\lambda\varrho$. *dens-ctxt-measure* (V, V', Γ, δ) $\varrho \gg\!= (\lambda\sigma.$ *expr-sem* σ *e*)) *?N f*
by (*intro hd-fst.IH*) (*insert hd-fst.prems hd-fst.hyps t*, *auto*)

let *?f* = $\lambda\varrho\ x.\ \int^+ y.\ f\ \varrho$ <|x,y|> ∂*stock-measure t′*
have *has-parametrized-subprob-density* (*state-measure* $V'\ \Gamma$)
($\lambda\varrho$. *dens-ctxt-measure* (V, V', Γ, δ) $\varrho \gg\!= (\lambda\sigma.$ *expr-sem* σ (*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* $V'\ \Gamma \bigotimes_M$ *stock-measure t*)
by *measurable*

fix ϱ **assume** ϱ: $\varrho \in$ *space* (*state-measure* $V'\ \Gamma$)
let *?M* = *dens-ctxt-measure* (V, V', Γ, δ) ϱ
from *dens* **and** ϱ **have** *has-subprob-density* (*?M* $\gg\!= (\lambda\sigma.$ *expr-sem* σ *e*)) *?N* (*f* ϱ)
unfolding *has-parametrized-subprob-density-def* **by** *auto*
hence *has-subprob-density* (*distr* (*?M* $\gg\!= (\lambda\sigma.$ *expr-sem* σ *e*)) (*stock-measure t*) (*op-sem Fst*))
(*stock-measure t*) (*?f* ϱ) (**is** *has-subprob-density ?R - -*)
by (*intro has-subprob-density-distr-Fst*) *simp*
also from *hd-fst.prems* **and** ϱ **have** *?R* = *?M* $\gg\!= (\lambda\sigma.$ *expr-sem* σ (*Fst* $$ *e*))
by (*intro expr-sem-op-eq-distr*[*symmetric*]) *simp-all*
finally show *has-subprob-density ...* (*stock-measure t*) (*?f* ϱ) **.**
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.prems* **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)
(λϱ. *dens-ctxt-measure* (*V*, *V′*, Γ, δ) ϱ ⤜ (λσ. *expr-sem* σ *e*)) *?N f*
by (*intro hd-snd.IH*) (*insert hd-snd.prems hd-snd.hyps t*, *auto*)

let *?f* = λϱ *y*. $\int^+$ *x*. *f* ϱ <|*x*,*y*|> ∂*stock-measure t*
have *has-parametrized-subprob-density* (*state-measure V′* Γ)
(λϱ. *dens-ctxt-measure* (*V*, *V′*, Γ, δ) ϱ ⤜ (λσ. *expr-sem* σ (*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* ∈ *borel-measurable* (*state-measure V′* Γ $\bigotimes_M$ *stock-measure t′*)
by *measurable*

fix ϱ **assume** ϱ: ϱ ∈ *space* (*state-measure V′* Γ)
let *?M* = *dens-ctxt-measure* (*V*, *V′*, Γ, δ) ϱ
from *dens* **and** ϱ **have** *has-subprob-density* (*?M* ⤜ (λσ. *expr-sem* σ *e*)) *?N* (*f* ϱ)
unfolding *has-parametrized-subprob-density-def* **by** *auto*
hence *has-subprob-density* (*distr* (*?M* ⤜ (λσ. *expr-sem* σ *e*)) (*stock-measure t′*) (*op-sem Snd*))
(*stock-measure t′*) (*?f* ϱ) (**is** *has-subprob-density ?R - -*)
by (*intro has-subprob-density-distr-Snd*) *simp*
also from *hd-snd.prems* **and** ϱ **have** *?R* = *?M* ⤜ (λσ. *expr-sem* σ (*Snd* $$ *e*))
by (*intro expr-sem-op-eq-distr*[*symmetric*]) *simp-all*
finally show *has-subprob-density* ... (*stock-measure t′*) (*?f* ϱ) **.**
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.prems* **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)
(λϱ. *dens-ctxt-measure* (*V*, *V′*, Γ, δ) ϱ ⤜ (λσ. *expr-sem* σ *e*))
(*stock-measure t*) *f*

by (*intro hd-op-discr.IH*) (*insert hd-op-discr.prems hd-op-discr.hyps t1*, *auto*)
from *hd-op-discr t1* **have** *expr-type* Γ *e* = *Some t* **and** *expr-type* Γ (*oper* $$ *e*) = *Some t′*
by (*simp-all add*: *expr-type-Some-iff*[*symmetric*])
hence *t1′*: *the* (*expr-type* Γ *e*) = *t* **and** *t2′*: *the* (*expr-type* Γ (*oper* $$ *e*)) = *t′* **by** *auto*
with *hd-op-discr* **have** *countable*: *countable-type t′* **by** *simp*

have *A*: *has-parametrized-subprob-density* (*state-measure V′* Γ)
(λϱ. *distr* (*dens-ctxt-measure* (*V*, *V′*, Γ, δ) ϱ ≫= (λσ. *expr-sem* σ *e*))
(*stock-measure t′*) (*op-sem oper*))
(*stock-measure t′*)
(λ*a b*. ∫⁺*x*. (*if op-sem oper x* = *b then 1 else 0*) ∗ *f a x* ∂*stock-measure t*)
proof (*intro has-parametrized-subprob-densityI*)
let *?f* = λϱ *y*. ∫⁺*x*. (*if op-sem oper x* = *y then 1 else 0*) ∗ *f* ϱ *x* ∂*stock-measure t*
note *sigma-finite-measure.borel-measurable-nn-integral*[*OF sigma-finite-stock-measure*, *measurable*]
show *case-prod ?f* ∈ *borel-measurable* (*state-measure V′* Γ ⨂$_M$ *stock-measure t′*)
using *measurable-op-sem*[*OF t2*] **by** *measurable*

fix ϱ **assume** ϱ: ϱ ∈ *space* (*state-measure V′* Γ)
let *?M* = *dens-ctxt-measure* (*V*, *V′*, Γ, δ) ϱ
let *?N* = *?M* ≫= (λσ. *expr-sem* σ *e*)

from *dens* **and** ϱ **have** *dens′*: *has-subprob-density ?N* (*stock-measure t*) (*f* ϱ)
unfolding *has-parametrized-subprob-density-def* **by** *auto*
from *hd-op-discr.prems t1*
have *M-e*: (λσ. *expr-sem* σ *e*) ∈ *measurable ?M* (*subprob-algebra* (*stock-measure t*))
by (*auto simp*: *measurable-dens-ctxt-measure-eq intro*!: *measurable-expr-sem*)
from *M-e* **have** *meas-N*: *measurable ?N* = *measurable* (*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*)
(*auto dest*: *has-subprob-densityD intro*!: *measurable-op-sem simp*: *meas-N*)

from *countable* **have** *count-space*: *stock-measure t′* = *count-space* (*type-universe t′*)
by (*rule countable-type-imp-count-space*)
from *dens′* **have** *?N* = *density* (*stock-measure t*) (*f* ϱ) **by** (*rule has-subprob-densityD*)
also {
fix *x* **assume** *x* ∈ *type-universe t*
with *M-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*: $\bigwedge\varrho.\ \varrho \in$ *space* (*state-measure V′* Γ) $\Longrightarrow$
distr (*dens-ctxt-measure* (*V*,*V′*,Γ,δ) $\varrho \gg\!=$ (λσ. *expr-sem* σ *e*))
(*stock-measure t′*) (*op-sem oper*) =
dens-ctxt-measure (*V*,*V′*,Γ,δ) $\varrho \gg\!=$ (λσ. *expr-sem* σ (*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′* Γ δ *e f t′*)
from *hd-neg.prems* **obtain** *t* **where** *t1*: Γ ⊢ *e* : *t* **and** *t2*: *op-type Minus t* = *Some t′* **by** *auto*
have *dens*: *has-parametrized-subprob-density* (*state-measure V′* Γ)
(λϱ. *dens-ctxt-measure* (*V*, *V′*, Γ, δ) $\varrho \gg\!=$ (λσ. *expr-sem* σ *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* ∈ *measurable* (*stock-measure t′*) (*stock-measure t*)
by (*simp split*: *pdf-type.split-asm*)
from *dens* **have** *Mf*[*measurable*]: *case-prod f* ∈ *borel-measurable* (*state-measure V′* Γ $\bigotimes_M$ *stock-measure t*)
by (*blast dest*: *has-parametrized-subprob-densityD*)
show (λ(ϱ, *x*). *f* ϱ (*op-sem Minus x*))
∈ *borel-measurable* (*state-measure V′* Γ $\bigotimes_M$ *stock-measure t′*) **by** *simp*

fix *M* ϱ **assume** *dens′*: *has-subprob-density M* (*stock-measure t*) (*f* ϱ)
hence *space-M*: *space M* = *space* (*stock-measure t*) **by** (*auto dest*: *has-subprob-densityD*)
from *t2* **have** *t-disj*: (*t* = *REAL* ∧ *t′* = *REAL*) ∨ (*t* = *INTEG* ∧ *t′* = *INTEG*)
by (*auto split*: *pdf-type.split-asm*)
thus *has-density* (*distr M* (*stock-measure t′*) (*op-sem Minus*))
(*stock-measure t′*) (λ*x*. *f* ϱ (*op-sem 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′*)
((λ*x*. *f* ϱ (*RealVal* (−*x*))) ∘ *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'*) (($\lambda x.$ *f* ϱ (*RealVal* ($-x$))) $\circ$ *extract-real*) $\longleftrightarrow$ *has-density* ... (*stock-measure t'*) ($\lambda x.$ *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'*)
(($\lambda x.$ *f* ϱ (*IntVal* ($-x$))) $\circ$ *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'*) (($\lambda x.$ *f* ϱ (*IntVal* ($-x$))) $\circ$ *extract-int*) $\longleftrightarrow$ *has-density* ... (*stock-measure t'*) ($\lambda x.$ *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.prems* **have** *t1*: $\Gamma \vdash e$: *REAL* **and** *t2*: t' = *REAL*
by (*auto split*: *pdf-type.split-asm*)
have *dens*[*measurable*]: *has-parametrized-subprob-density* (*state-measure* V' Γ) ($\lambda\varrho.$ *dens-ctxt-measure* (V, V', Γ, δ) $\varrho \gg\!= (\lambda\sigma.$ *expr-sem* σ e)) (*stock-measure REAL*) *f*
by (*intro hd-exp.IH*) (*insert hd-exp.prems 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* $\in$ *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* $\bigotimes_M$ *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* $\bigotimes_M$ *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.prems* **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.prems 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* $\bigotimes_M$ *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* $\bigotimes_M$ *stock-measure t′*) **by** (*simp add*: *t2*)
fix *M ϱ* **assume** *dens′*: *has-subprob-density M* (*stock-measure REAL*) (*f ϱ*)

hence *space-M*: *space M* = *space* (*stock-measure REAL*) **by** (*auto dest*: *has-subprob-densityD*)
have *has-density* (*distr M* (*stock-measure t'*) (*lift-RealVal inverse*)) (*stock-measure t'*)
((λ*x*. *f* ϱ (*RealVal* (*inverse x*)) ∗ *ennreal* (*inverse* (*x* ∗ *x*)))
∘ *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* ϱ
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.prems* **have** *t1*: Γ ⊢ *e* : *t* **and** *t2*: Γ ⊢ *e'* : *t* **and** *t-disj*: *t* = *REAL* ∨ *t* = *INTEG*
by (*elim expr-typing-opE*, (*auto split*: *pdf-type.split-asm*)[])+
hence *t4*: *op-type Add* (*PRODUCT t t*) = *Some t* **by** *auto*
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 t*) *f*
by (*rule hd-addc.IH*) (*insert hd-addc.prems 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.prems hd-addc.hyps*
show *case-prod ?f* ∈ *borel-measurable* (*state-measure V'* Γ $\bigotimes_M$ *stock-measure t*)
by (*intro addc-density-measurable has-parametrized-subprob-densityD*[*OF dens*]) *auto*

fix ϱ **assume** ϱ: ϱ ∈ *space* (*state-measure V'* Γ)
let *?M* = *dens-ctxt-measure* (*V*, *V'*, Γ, δ) ϱ
let *?v1* = *extract-int* (*expr-sem-rf* ϱ *e'*) **and** *?v2* = *extract-real* (*expr-sem-rf* ϱ *e'*)
from *dens* **and** ϱ **have** *dens'*: *has-subprob-density* (*?M* ≫= (λσ. *expr-sem* σ *e*)) (*stock-measure t*) (*f* ϱ)
unfolding *has-parametrized-subprob-density-def has-subprob-density-def* **by** *auto*

have *Me*: (λσ. *expr-sem* σ *e*) ∈

measurable (*state-measure* ($V \cup V'$) Γ) (*subprob-algebra* (*stock-measure* *t*))
using *t1 hd-addc.prems* **by** (*intro measurable-expr-sem*) *simp-all*
from *hd-addc.prems 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* ⤜ ($\lambda\sigma$. *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.prems* **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* ⤜ ($\lambda\sigma$. *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* ⤜ ($\lambda\sigma$. *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* ⤜ ($\lambda\sigma$. *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 ϱ) **.**
qed
also have ?N′ = distr (?M ≫= (λσ. expr-sem σ e)) (stock-measure t)
(λw. op-sem Add <|w, expr-sem-rf ϱ e′|>) **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 ... = ?N ϱ
using hd-addc.prems hd-addc.hyps t-disj ϱ
by (intro bin-op-randomfree-restructure[OF t1 t2, symmetric]) auto
finally show has-density (?N ϱ) (stock-measure t) (?f ϱ) **.**
qed

next

case (hd-multc V V′ Γ δ e f c t)
interpret density-context V V′ Γ δ **by** fact
from hd-multc.prems hd-multc.hyps
have t1: Γ ⊢ e : REAL **and** t2: val-type c = REAL **and** t: t = REAL
by (elim expr-typing-opE expr-typing-valE expr-typing-pairE,
(auto split: pdf-type.split-asm) [])+
have t4: op-type Mult (PRODUCT REAL REAL) = Some REAL **by** simp
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
by (rule hd-multc.IH) (insert hd-multc.prems 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 = stock-measure t **and** ?MP = stock-measure (PRODUCT t t)
have M-mult[measurable]: (op-sem Mult) ∈ measurable ?MP ?MR **by** (simp add: measurable-op-sem t)
show case-prod ?f ∈ borel-measurable (state-measure V′ Γ $\bigotimes_M$ stock-measure t)
by measurable (insert t2, auto simp: t val-type-eq-REAL lift-RealVal-def)

fix ϱ **assume** ϱ: ϱ ∈ space (state-measure V′ Γ)
let ?M = dens-ctxt-measure (V, V′, Γ, δ) ϱ
from dens **and** ϱ **have** dens′: has-subprob-density (?M ≫= (λσ. expr-sem σ e)) (stock-measure t) (f ϱ)
unfolding has-parametrized-subprob-density-def has-subprob-density-def **by** auto

have Me: (λσ. expr-sem σ e) ∈
measurable (state-measure (V ∪ V′) Γ) (subprob-algebra (stock-measure REAL))

using *t1 hd-multc.prems* **by** (*intro measurable-expr-sem*) *simp-all*
have *space-e*: *space* (*?M* $\gg=$ ($\lambda\sigma$. *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.prems* **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* $\gg=$ ($\lambda\sigma$. *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*))) $\circ$ *extract-real*
from *hd-multc.hyps* **have** *extract-real c* $\neq$ *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* $\gg=$ ($\lambda\sigma$. *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.prems 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.prems hd-add.hyps*
have *t1*: $\Gamma \vdash$ *e* : *PRODUCT t t* **and** *t2*: *op-type Add* (*PRODUCT t t*) = *Some*
t **and**
t-disj: *t* = *REAL* $\vee$ *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)
($\lambda\varrho$. *dens-ctxt-measure* (V,V',Γ,δ) ϱ $\gg=$ ($\lambda\sigma$. *expr-sem* σ *e*))
(*stock-measure ?tp*) *f*
by (*rule hd-add.IH*) (*insert hd-add.prems t1 t2 t-disj*, *auto*)
from *hd-add.prems 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'* = $\lambda\varrho$ *z*. $\int^{+}$ *x*. *f* ϱ <|*x* , *op-sem Add* <|*z*, *op-sem Minus x*|>|> ∂*stock-measure t*
have *case-prod ?f'* $\in$ *borel-measurable* (*state-measure* V' Γ $\bigotimes_M$ *stock-measure* *t*)
by *measurable*
also have *case-prod ?f'* $\in$ *borel-measurable* (*state-measure* V' Γ $\bigotimes_M$ *stock-measure* *t*) $\longleftrightarrow$
case-prod ?f $\in$ *borel-measurable* (*state-measure* V' Γ $\bigotimes_M$ *stock-measure t*)
by (*intro measurable-cong*) (*auto simp*: *space-pair-measure*)
finally show *case-prod ?f* $\in$ *borel-measurable* (*state-measure* V' Γ $\bigotimes_M$ *stock-measure* *t*) **.**

fix *M* ϱ **assume** *dens'*: *has-subprob-density M* (*stock-measure* (*PRODUCT t* *t*)) (*f* ϱ)
hence *Mf*[*measurable*]: *f* ϱ $\in$ *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''* = (λ*z*. $\int^{+}$*x*. *f* ϱ (*RealPairVal* (*x*, *z* $-$ *x*)) ∂*lborel*) $\circ$ *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''* = (λ*z*. $\int^{+}$ *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 $\bigwedge$*z*. *val-type z* = *REAL* $\Longrightarrow$
(λ*x*. *f* ϱ <|*x*, *op-sem Add* <|*z*, *op-sem Minus x*|>|>) $\in$ *borel-measurable* (*stock-measure REAL*)
by (*rule Mf*[*THEN measurable-compose-rev*]) (*simp add*: *t*)

hence *has-density* (*distr M* (*stock-measure t*) (*op-sem Add*)) (*stock-measure t*) *?f''* $\longleftrightarrow$
has-density (*distr M* (*stock-measure t*) (*op-sem Add*)) (*stock-measure t*) (*?f* ϱ)
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*: $t = INTEG$
let $?f'' = (\lambda z.\ \int^+ x.\ f\ \varrho\ (IntPairVal\ (x,\ z - x))\ \partial count\text{-}space\ UNIV) \circ extract\text{-}int$
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-IntPairVal*)
(*simp-all add*: *has-subprob-density-imp-has-density distr-convolution-ring-count-space*)
also have $?f'' = (\lambda z.\ \int^+ x.\ f\ \varrho\ (IntPairVal\ (x,\ extract\text{-}int\ z - x))\ \partial count\text{-}space\ UNIV)$
(**is** - = *?f''*)
by (*auto simp add*: *t space-embed-measure extract-int-def*)
also have *has-density* (*distr M* (*stock-measure t*) (*op-sem Add*)) (*stock-measure t*) *?f''* $\longleftrightarrow$
has-density (*distr M* (*stock-measure t*) (*op-sem Add*)) (*stock-measure t*) (*?f* ϱ)
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 $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f$ *density-context* $V\ V'\ \Gamma\ \delta$ $\Gamma \vdash e : t$ *free-vars* $e \subseteq V \cup V'$
assumes $\bigwedge\varrho\ x.\ \varrho \in space\ (state\text{-}measure\ V'\ \Gamma) \Longrightarrow x \in space\ (stock\text{-}measure\ t) \Longrightarrow f\ \varrho\ x = f'\ \varrho\ x$
shows $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f'$
proof (*rule hd-AE*[*OF assms*(*1*,*3*) *AE-I2*[*OF assms*(*5*)]])
note *dens* = *expr-has-density-sound-aux*[*OF assms*(*1*,*3*,*2*,*4*)]
note *dens'* = *has-parametrized-subprob-densityD*[*OF this*]
show $(\lambda(\varrho, x).\ f'\ \varrho\ x) \in borel\text{-}measurable\ (state\text{-}measure\ V'\ \Gamma \bigotimes_M stock\text{-}measure\ t)$
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* ({},{},Γ,(λ-. *1*)) (λ-. *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* ({},{},Γ,(λ-. *1*)) (λ-. *undefined*) = *1*
unfolding *branch-prob-def* **by** (*subst prob-space.emeasure-space-1*) *simp-all*

lemma *expr-has-density-sound*:
assumes ({},{},Γ,(λ-. *1*)) $\vdash_d$ *e* ⇒ *f* Γ ⊢ *e* : *t free-vars e* = {}
shows *has-subprob-density* (*expr-sem σ e*) (*stock-measure t*) (*f* (λ-. *undefined*))
proof−
let *?M* = *dens-ctxt-measure* ({},{},Γ,λ-. *1*) (λ-. *undefined*)
have *density-context* {} {} Γ (λ-. *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* {} Γ)
(λϱ. *dens-ctxt-measure* ({},{},Γ,λ-. *1*) ϱ ⤜ (λσ. *expr-sem σ e*))
(*stock-measure t*) *f*
by *blast*
also have *state-measure* {} Γ = *count-space* {λ-. *undefined*}
by (*rule measure-eqI*) (*simp-all add*: *state-measure-def PiM-empty emeasure-density*)
finally have *has-subprob-density* (*?M* ⤜ (λσ. *expr-sem σ e*))
(*stock-measure t*) (*f* (λ-. *undefined*))
unfolding *has-parametrized-subprob-density-def* **by** *simp*
also from *assms* **have** (λσ. *expr-sem σ e*) ∈ *measurable* (*state-measure* {} Γ)
(*subprob-algebra* (*stock-measure t*))
by (*intro measurable-expr-sem*) *auto*
hence *?M* ⤜ (λσ. *expr-sem σ e*) = *expr-sem* (λ-. *undefined*) *e*
by (*intro dens-ctxt-measure-empty-bind*) (*auto simp*: *state-measure-def PiM-empty*)
also from *assms* **have** ... = *expr-sem σ 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 cexpr cexpr* $(<\text{-}, \text{-}>_c\ [0,\ 0]\ 1000)$
| *COperator pdf-operator cexpr* (**infixl** $\$\$_c$ *999*)
| *CIf cexpr cexpr cexpr* (IF_c - *THEN* - *ELSE* - $[0,\ 0,\ 10]$ *10*)
| *CIntegral cexpr pdf-type* ($\int_c$ - ∂- $[61]$ *110*)

abbreviation (*input*) *cexpr-fun* :: (*cexpr* $\Rightarrow$ *cexpr*) $\Rightarrow$ *cexpr* (**binder** λ_c *10*)
where
cexpr-fun f $\equiv$ *f* (*CVar 0*)
abbreviation *cexpr-Add* (**infixl** $+_c$ *65*) **where**
cexpr-Add a b $\equiv$ *Add* $\$\$_c$ $<a,\ b>_c$
abbreviation *cexpr-Minus* ($-_c$ - $[81]$ *80*) **where**
cexpr-Minus a $\equiv$ *Minus* $\$\$_c$ *a*
abbreviation *cexpr-Sub* (**infixl** $-_c$ *65*) **where**
cexpr-Sub a b $\equiv$ $a +_c -_c b$
abbreviation *cexpr-Mult* (**infixl** $*_c$ *70*) **where**
cexpr-Mult a b $\equiv$ *Mult* $\$\$_c$ $<a,\ b>_c$
abbreviation $inverse_c$ *e* $\equiv$*Inverse* $\$\$_c$ *e*
abbreviation *cexpr-Div* (**infixl** $'/_c$ *70*) **where**
cexpr-Div a b $\equiv$ $a *_c inverse_c\ b$
abbreviation $fact_c$ *e* $\equiv$ *Fact* $\$\$_c$ *e*
abbreviation $sqrt_c$ *e* $\equiv$ *Sqrt* $\$\$_c$ *e*
abbreviation exp_c *e* $\equiv$ *Exp* $\$\$_c$ *e*
abbreviation ln_c *e* $\equiv$ *Ln* $\$\$_c$ *e*
abbreviation fst_c *e* $\equiv$ *Fst* $\$\$_c$ *e*
abbreviation snd_c *e* $\equiv$ *Snd* $\$\$_c$ *e*
abbreviation *cexpr-Pow* (**infixl** $\hat{}_c$ *75*) **where**
cexpr-Pow a b $\equiv$ *Pow* $\$\$_c$ $<a,\ b>_c$
abbreviation *cexpr-And* (**infixl** $\wedge_c$ *35*) **where**
cexpr-And a b $\equiv$ *And* $\$\$_c$ $<a,\ b>_c$
abbreviation *cexpr-Or* (**infixl** $\vee_c$ *30*) **where**
cexpr-Or a b $\equiv$ *Or* $\$\$_c$ $<a,\ b>_c$
abbreviation *cexpr-Not* ($\neg_c$ - $[40]$ *40*) **where**
cexpr-Not a $\equiv$ *Not* $\$\$_c$ *a*
abbreviation *cexpr-Equals* (**infixl** $=_c$ *70*) **where**
cexpr-Equals a b $\equiv$ *Equals* $\$\$_c$ $<a,\ b>_c$
abbreviation *cexpr-Less* (**infixl** $<_c$ *70*) **where**
cexpr-Less a b $\equiv$ *Less* $\$\$_c$ $<a,\ b>_c$
abbreviation *cexpr-LessEq* (**infixl** $\leq_c$ *70*) **where**
cexpr-LessEq a b $\equiv$ $a =_c b \vee_c a <_c b$
abbreviation *cexpr-RealCast* ($\langle\text{-}\rangle_c$ $[0]$ *90*) **where**
cexpr-RealCast a $\equiv$ *Cast REAL* $\$\$_c$ *a*
abbreviation *CReal* **where**
CReal x $\equiv$ *CVal* (*RealVal x*)
abbreviation *CInt* **where**
CInt x $\equiv$ *CVal* (*IntVal x*)
abbreviation π_c **where**
π_c $\equiv$ *Pi* $\$\$_c$ (*CVal UnitVal*)

instantiation *cexpr* :: *expr*
begin

primrec *free-vars-cexpr* :: *cexpr* ⇒ *vname set* **where**
free-vars-cexpr (*CVar x*) = {*x*}
| *free-vars-cexpr* (*CVal* -) = {}
| *free-vars-cexpr* ($oper\ \$\$_c\ e$) = *free-vars-cexpr e*
| *free-vars-cexpr* ($<e1,\ e2>_c$) = *free-vars-cexpr e1* ∪ *free-vars-cexpr e2*
| *free-vars-cexpr* (IF_c *b THEN e1 ELSE e2*) =
free-vars-cexpr b ∪ *free-vars-cexpr e1* ∪ *free-vars-cexpr e2*
| *free-vars-cexpr* ($\int_c e\ \partial t$) = *Suc* −‘ *free-vars-cexpr e*

instance ..
end

inductive *cexpr-typing* :: *tyenv* ⇒ *cexpr* ⇒ *pdf-type* ⇒ *bool* ((*1*-/ $\vdash_c$/ (- :/ -)) [*50*,*0*,*50*] *50*) **where**
cet-val: $\Gamma \vdash_c$ *CVal v*: *val-type v*
| *cet-var*: $\Gamma \vdash_c$ *CVar x* : Γ *x*
| *cet-pair*: $\Gamma \vdash_c e1 : t1 \Longrightarrow \Gamma \vdash_c e2 : t2 \Longrightarrow \Gamma \vdash_c <e1,\ e2>_c : PRODUCT\ t1\ t2$
| *cet-op*: $\Gamma \vdash_c e : t \Longrightarrow$ *op-type oper t* = *Some* $t' \Longrightarrow \Gamma \vdash_c oper\ \$\$_c\ e : t'$
| *cet-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-int*: $t \cdot \Gamma \vdash_c e : REAL \Longrightarrow \Gamma \vdash_c \int_c e\ \partial t : REAL$

lemma *cet-val′*: *t* = *val-type v* $\Longrightarrow \Gamma \vdash_c$ *CVal v* : *t*
by (*simp add*: *cet-val*)

lemma *cet-var′*: *t* = Γ *x* $\Longrightarrow \Gamma \vdash_c$ *CVar x* : *t*
by (*simp add*: *cet-var*)

lemma *cet-not*: $\Gamma \vdash_c e : BOOL \Longrightarrow \Gamma \vdash_c \neg_c\ e : BOOL$
by (*intro cet-op*[**where** *t* = *BOOL*] *cet-pair*, *simp*, *simp*)

lemma *cet-and*: $\Gamma \vdash_c e1 : BOOL \Longrightarrow \Gamma \vdash_c e2 : BOOL \Longrightarrow \Gamma \vdash_c e1 \wedge_c e2 : BOOL$
and
cet-or: $\Gamma \vdash_c e1 : BOOL \Longrightarrow \Gamma \vdash_c e2 : BOOL \Longrightarrow \Gamma \vdash_c e1 \vee_c e2 : BOOL$
by (*intro cet-op*[**where** *t* = *PRODUCT BOOL BOOL*] *cet-pair*, *simp*, *simp*, *simp*)+

lemma *cet-minus-real*: $\Gamma \vdash_c e : REAL \Longrightarrow \Gamma \vdash_c -_c e : REAL$ **and**
cet-inverse: $\Gamma \vdash_c e : REAL \Longrightarrow \Gamma \vdash_c inverse_c\ e : REAL$ **and**
cet-sqrt: $\Gamma \vdash_c e : REAL \Longrightarrow \Gamma \vdash_c sqrt_c\ e : REAL$ **and**
cet-exp: $\Gamma \vdash_c e : REAL \Longrightarrow \Gamma \vdash_c exp_c\ e : REAL$ **and**
cet-ln: $\Gamma \vdash_c e : REAL \Longrightarrow \Gamma \vdash_c ln_c\ e : REAL$
by (*rule cet-op*[**where** *t* = *REAL*], *simp*, *simp*)+

lemma *cet-pow-real*: $\Gamma \vdash_c e1 : REAL \Longrightarrow \Gamma \vdash_c e2 : INTEG \Longrightarrow \Gamma \vdash_c e1\ \hat{}_c\ e2 :$ *REAL*

by (*intro cet-op*[**where** $t = PRODUCT\ REAL\ INTEG$] *cet-pair*) *simp-all*

lemma *cet-add-real*: $\Gamma \vdash_c e1 : REAL \Longrightarrow \Gamma \vdash_c e2 : REAL \Longrightarrow \Gamma \vdash_c e1 +_c e2 : REAL$ **and**
cet-mult-real: $\Gamma \vdash_c e1 : REAL \Longrightarrow \Gamma \vdash_c e2 : REAL \Longrightarrow \Gamma \vdash_c e1 *_c e2 : REAL$ **and**
cet-less-real: $\Gamma \vdash_c e1 : REAL \Longrightarrow \Gamma \vdash_c e2 : REAL \Longrightarrow \Gamma \vdash_c e1 <_c e2 : BOOL$
by (*intro cet-op*[**where** $t = PRODUCT\ REAL\ REAL$] *cet-pair*, *simp*, *simp*, *simp*)+

lemma *cet-eq*: $\Gamma \vdash_c e1 : t \Longrightarrow \Gamma \vdash_c e2 : t \Longrightarrow \Gamma \vdash_c e1 =_c e2 : BOOL$
by (*intro cet-op*[**where** $t = PRODUCT\ t\ t$] *cet-pair*, *simp*, *simp*, *simp*)+

lemma *cet-less-eq-real*: $\Gamma \vdash_c e1 : REAL \Longrightarrow \Gamma \vdash_c e2 : REAL \Longrightarrow \Gamma \vdash_c e1 \leq_c e2 : BOOL$
by (*intro cet-less-real cet-or cet-eq*)

lemma *cet-minus-int*: $\Gamma \vdash_c e : INTEG \Longrightarrow \Gamma \vdash_c -_c e : INTEG$
by (*rule cet-op*[**where** $t = INTEG$], *simp*, *simp*)+

lemma *cet-add-int*: $\Gamma \vdash_c e1 : INTEG \Longrightarrow \Gamma \vdash_c e2 : INTEG \Longrightarrow \Gamma \vdash_c e1 +_c e2 : INTEG$ **and**
cet-mult-int: $\Gamma \vdash_c e1 : INTEG \Longrightarrow \Gamma \vdash_c e2 : INTEG \Longrightarrow \Gamma \vdash_c e1 *_c e2 : INTEG$ **and**
cet-less-int: $\Gamma \vdash_c e1 : INTEG \Longrightarrow \Gamma \vdash_c e2 : INTEG \Longrightarrow \Gamma \vdash_c e1 <_c e2 : BOOL$
by (*intro cet-op*[**where** $t = PRODUCT\ INTEG\ INTEG$] *cet-pair*, *simp*, *simp*, *simp*)+

lemma *cet-less-eq-int*: $\Gamma \vdash_c e1 : INTEG \Longrightarrow \Gamma \vdash_c e2 : INTEG \Longrightarrow \Gamma \vdash_c e1 \leq_c e2 : BOOL$
by (*intro cet-less-int cet-or cet-eq*)

lemma *cet-sub-int*: $\Gamma \vdash_c e1 : INTEG \Longrightarrow \Gamma \vdash_c e2 : INTEG \Longrightarrow \Gamma \vdash_c e1 -_c e2 : INTEG$
by (*intro cet-minus-int cet-add-int*)

lemma *cet-fst*: $\Gamma \vdash_c e : PRODUCT\ t\ t' \Longrightarrow \Gamma \vdash_c fst_c\ e : t$ **and**
cet-snd: $\Gamma \vdash_c e : PRODUCT\ t\ t' \Longrightarrow \Gamma \vdash_c snd_c\ e : t'$
by (*erule cet-op*, *simp*)+

lemma *cet-cast-real*: $\Gamma \vdash_c e : BOOL \Longrightarrow \Gamma \vdash_c \langle e \rangle_c : REAL$
by (*intro cet-op*[**where** $t = BOOL$]) *simp-all*

lemma *cet-cast-real-int*: $\Gamma \vdash_c e : INTEG \Longrightarrow \Gamma \vdash_c \langle e \rangle_c : REAL$
by (*intro cet-op*[**where** $t = INTEG$]) *simp-all*

lemma *cet-sub-real*: $\Gamma \vdash_c e1 : REAL \Longrightarrow \Gamma \vdash_c e2 : REAL \Longrightarrow \Gamma \vdash_c e1 -_c e2 :$

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*]: $\Gamma \vdash_c$ <*e1*, *e2*>$_c$: *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* ⇒ *cexpr* ⇒ *pdf-type option* **where**
cexpr-type - (*CVal v*) = *Some* (*val-type v*)
| *cexpr-type* Γ (*CVar x*) = *Some* (Γ *x*)
| *cexpr-type* Γ (<*e1*, *e2*>$_c$) = (*case* (*cexpr-type* Γ *e1*, *cexpr-type* Γ *e2*) *of*
(*Some t1*, *Some t2*) ⇒ *Some* (*PRODUCT t1 t2*)
| - ⇒ *None*)
| *cexpr-type* Γ (*oper* \$\$$_c$ *e*) = (*case cexpr-type* Γ *e of*
Some t ⇒ *op-type oper t*
| - ⇒ *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′*) ⇒ *if t* = *t′ then Some t else None*
| - ⇒ *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* ⟷ $\Gamma \vdash_c e : t$
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 $\Gamma \vdash_c e : t$ $\bigwedge x.\ x \in$ *free-vars e* ⟹ Γ *x* = Γ′ *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 $\bigwedge x.\ x \in$ *free-vars e* $\Longrightarrow$ *case-nat t* $\Gamma\ x =$ *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 $\bigwedge x.\ x \in$ *free-vars e* $\Longrightarrow \Gamma\ x = \Gamma'\ x$
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 $<e1,\ e2>_c = <|$*cexpr-sem* σ *e1*, *cexpr-sem* σ *e2*$|>$
| *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* ($\int x.$ *extract-real* (*cexpr-sem* $(x \cdot \sigma)$ *e*) ∂(*stock-measure t*))

definition *cexpr-equiv* :: *cexpr* $\Rightarrow$ *cexpr* $\Rightarrow$ *bool* **where**
cexpr-equiv e1 e2 $\equiv \forall \sigma.$ *cexpr-sem* σ *e1* = *cexpr-sem* σ *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* $\subseteq V$ $\sigma \in$ *space* (*state-measure V* Γ)
shows *val-type* (*cexpr-sem* σ *e*) = *t*
using *assms* **by** (*induction arbitrary*: σ *V rule*: *cexpr-typing.induct*)
(*auto intro*: *state-measure-var-type op-sem-val-type*)

lemma *cexpr-sem-eq-on-vars*:
assumes $\bigwedge x.\ x \in$ *free-vars e* $\Longrightarrow \sigma\ x = \sigma'\ x$
shows *cexpr-sem* σ *e* = *cexpr-sem* σ' *e*
using *assms*
proof (*induction e arbitrary*: $\sigma\ \sigma'$)
case (*CPair e1 e2* $\sigma\ \sigma'$)
from *CPair.prems* **show** *?case* **by** (*auto intro*!: *CPair.IH*)
next
case (*COperator oper e* $\sigma\ \sigma'$)
from *COperator.prems* **show** *?case* **by** (*auto simp*: *COperator.IH*[*of* $\sigma\ \sigma'$])
next
case (*CIf b e1 e2* $\sigma\ \sigma'$)
from *CIf.prems* **show** *?case* **by** (*auto simp*: *CIf.IH*[*of* $\sigma\ \sigma'$])

next
  case (*CIntegral e t σ σ'*)
  have *cexpr-sem σ* ($\int_c$ *e ∂t*) = *RealVal* ($\int$ *x. extract-real* (*cexpr-sem* (*case-nat x σ*) *e*) *∂stock-measure t*)
    by *simp*
  also from *CIntegral.prems* **have** *A*: (*λv. cexpr-sem* (*case-nat v σ*) *e*) = (*λv. 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 ∂t*)
    by *simp*
  finally show *?case* **.**
qed *simp-all*

definition *eval-cexpr* :: *cexpr ⇒ state ⇒ val ⇒ 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* ⟹ *free-vars e* ⊆ *V* ⟹
    (*λσ. cexpr-sem σ e*) ∈ *measurable* (*state-measure V Γ*) (*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* = (Π_M *x∈V. stock-measure* (*Γ x*)) $\bigotimes_M$ *stock-measure t*
  let *?N* = *embed-measure lborel RealVal*
  have ∗[*measurable*]: (*λa. cexpr-sem a e*) ∈ *measurable* (*state-measure* (*shift-var-set V*) (*case-nat t Γ*)) *REAL*
    using *cet-int.prems 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* : *REAL*
  assumes *free-vars e* ⊆ *shift-var-set V*
  shows *case-prod* (*eval-cexpr e*) ∈ *borel-measurable* (*state-measure V Γ* $\bigotimes_M$ *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* : *REAL* $\Gamma \vdash_c$ *e2* : *REAL*
  assumes *σ* ∈ *space* (*state-measure V Γ*) *free-vars e1* ⊆ *V free-vars e2* ⊆ *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 $\sigma \in$ *space* (*state-measure V* Γ) *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*)

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* ∂t) = $\int_c$ *map-vars* (*case-nat 0* (λx. *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* (λx. *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*:
$\Gamma \circ f \vdash_c e : t \Longrightarrow \Gamma \vdash_c$ *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* (λx. *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* ($\lambda x.$ *Suc* (f x))) e) =
cexpr-sem (*case-nat* x $\sigma \circ$ *case-nat* *0* ($\lambda x.$ *Suc* (f x))) e
by (*rule CIntegral.IH*)
also have *case-nat* x $\sigma \circ$ *case-nat* *0* ($\lambda x.$ *Suc* (f x)) = *case-nat* x ($\lambda a.$ 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* ($\lambda x.$ *Suc* (f x))) e) =
cexpr-sem (*case-nat* x ($\lambda a.$ σ (f a))) e .
}
thus *?case* **by** *simp*
qed *simp-all*

definition *insert-var* :: *vname* $\Rightarrow$ (*vname* $\Rightarrow$ $'a$) $\Rightarrow$ $'a$ $\Rightarrow$ *vname* $\Rightarrow$ $'a$ **where**
insert-var v f x w $\equiv$ *if* $w = v$ *then* x *else if* $w > v$ *then* f ($w - 1$) *else* f w

lemma *insert-var-0*[*simp*]: *insert-var* *0* f x = *case-nat* x f
by (*intro ext*) (*simp add*: *insert-var-def split*: *nat.split*)

Substitutes expression e for variable x in e'.

primrec *cexpr-subst* :: *vname* $\Rightarrow$ *cexpr* $\Rightarrow$ *cexpr* $\Rightarrow$ *cexpr* **where**
cexpr-subst - - (*CVal* v) = *CVal* v
| *cexpr-subst* x e (*CVar* y) = *insert-var* x *CVar* e y
| *cexpr-subst* x e <*e1*, *e2*>$_c$ = <*cexpr-subst* x e *e1*, *cexpr-subst* x e *e2*>$_c$
| *cexpr-subst* x e (*oper* \$\$$_c$ e') = *oper* \$\$$_c$ (*cexpr-subst* x e e')
| *cexpr-subst* x e (*IF*$_c$ b *THEN e1 ELSE e2*) =
(*IF*$_c$ *cexpr-subst* x e b *THEN cexpr-subst* x e *e1* *ELSE cexpr-subst* x e *e2*)
| *cexpr-subst* x e ($\int_c$ e' ∂t) = ($\int_c$ *cexpr-subst* (*Suc* x) (*map-vars Suc* e) e' ∂t)

lemma *cexpr-sem-cexpr-subst-aux*:
cexpr-sem σ (*cexpr-subst* x e e') = *cexpr-sem* (*insert-var* x σ (*cexpr-sem* σ e)) e'
proof (*induction* e' *arbitrary*: x e σ)
case (*CIntegral* e' t x e σ)
have A: $\bigwedge y.$ *insert-var* (*Suc* x) (*case-nat* y σ) (*cexpr-sem* σ e) =
case-nat y (*insert-var* x σ (*cexpr-sem* σ e))
by (*intro ext*) (*simp add*: *insert-var-def split*: *nat.split*)
show *?case* **by** (*simp add*: *o-def A cexpr-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 *cexpr-sem-cexpr-subst*:
cexpr-sem σ (*cexpr-subst* *0* e e') = *cexpr-sem* (*case-nat* (*cexpr-sem* σ e) σ) e'
using *cexpr-sem-cexpr-subst-aux* **by** *simp*

lemma *cexpr-typing-subst-aux*:
assumes *insert-var* x Γ t $\vdash_c$ e' : t' Γ $\vdash_c$ e : t

shows $\Gamma \vdash_c$ *cexpr-subst x e e′ : t′*
using *assms*
proof (*induction e′ arbitrary*: *x* Γ *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′* = *REAL* **using** *CIntegral.prems*(*1*) **by** *auto*
have *case-nat t′′* (*insert-var x* Γ *t*) $\vdash_c$ *e′* : *t′* **using** *CIntegral.prems*(*1*) **by** *auto*
also have *case-nat t′′* (*insert-var x* Γ *t*) = *insert-var* (*Suc x*) (*case-nat t′′* Γ) *t*
by (*intro ext*) (*simp add*: *insert-var-def split*: *nat.split*)
finally have *insert-var* (*Suc x*) (*case-nat t′′* Γ) *t* $\vdash_c$ *e′* : *t′* **.**
moreover from *CIntegral.prems*(*2*) **have** *case-nat t′′* Γ $\vdash_c$ *map-vars Suc e* : *t*
by (*intro cexpr-typing-map-vars*) (*simp add*: *o-def*)
ultimately have *case-nat t′′* Γ $\vdash_c$ *cexpr-subst* (*Suc x*) (*map-vars 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 Γ $\vdash_c$ *e* : *t case-nat t* Γ $\vdash_c$ *e′* : *t′*
shows Γ $\vdash_c$ *cexpr-subst 0 e e′* : *t′*
using *cexpr-typing-subst-aux assms* **by** *simp*

lemma *free-vars-cexpr-subst-aux*:
free-vars (*cexpr-subst x e e′*) ⊆ (λ*y*. *if y* ≥ *x then y* + *1 else y*) −‘ *free-vars e′* ∪ *free-vars e*
(**is** *free-vars* - ⊆ *?f x* −‘ - ∪ -)
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 *free-vars* (*cexpr-subst x e* (∫ $_c$ *e′* ∂*t*)) ⊆
Suc −‘ (*?f* (*Suc x*) −‘ *free-vars e′*) ∪
Suc −‘ (*free-vars* (*map-vars Suc e*)) (**is** - ⊆ *?A* ∪ *?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' ∂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' ∂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* (σ(*x* := *cexpr-sem* σ *e*)) e'
proof (*induction* e' *arbitrary*: *x e* σ)
case (*CIntegral* e' *t x e* σ)
have $\bigwedge$*y*. (*case-nat y* σ)(*Suc x* := *cexpr-sem* (*case-nat y* σ) (*map-vars Suc e*)) =
case-nat y (σ(*x* := *cexpr-sem* σ *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** $\circ_c$ *55*) **where**
cexpr-comp b a $\equiv$ *cexpr-comp-aux 0 a b*

lemma *cexpr-typing-cexpr-comp-aux*:
assumes Γ(*x* := *t1*) $\vdash_c$ e' : *t2* Γ $\vdash_c$ *e* : *t1*
shows Γ $\vdash_c$ *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*) []
next
case *CPair*
thus *?case* **by** (*elim cexpr-typing-pairE*) (*auto intro*!: *cexpr-typing.intros*) []
next

case (*CIntegral* e' *t* Γ *e x t2*)
from *CIntegral.prems* **have** [*simp*]: $t2 = REAL$ **by** *auto*
from *CIntegral.prems* **have** *case-nat t* $(\Gamma(x := t1)) \vdash_c e' : REAL$ **by** (*elim cexpr-typing-intE*)
also have *case-nat t* $(\Gamma(x := t1)) =$ (*case-nat t* Γ)(*Suc x* $:= t1$)
by (*intro ext*) (*simp split*: *nat.split*)
finally have ... $\vdash_c e' : REAL$ **.**
thus $\Gamma \vdash_c$ *cexpr-comp-aux x e* $(\int_c e' \partial t) : t2$
by (*auto intro*!: *cexpr-typing.intros CIntegral.IH cexpr-typing-map-vars*
simp: *o-def CIntegral.prems*)
qed (*auto intro*!: *cexpr-typing.intros*)

lemma *cexpr-typing-cexpr-comp*[*intro*]:
assumes *case-nat t1* $\Gamma \vdash_c g : t2$
assumes *case-nat t2* $\Gamma \vdash_c f : t3$
shows *case-nat t1* $\Gamma \vdash_c f \circ_c g : t3$
proof (*unfold cexpr-comp-def*, *intro cexpr-typing-cexpr-comp-aux*)
have (*case-nat t1* Γ)($0 := t2$) = *case-nat t2* Γ
by (*intro ext*) (*simp split*: *nat.split*)
with *assms* **show** (*case-nat t1* Γ)($0 := t2$) $\vdash_c f : t3$ **by** *simp*
qed (*insert assms*)

lemma *free-vars-cexpr-comp-aux*:
free-vars (*cexpr-comp-aux x e* e') $\subseteq$ (*free-vars* $e' - \{x\}$) $\cup$ *free-vars e*
proof (*induction* e' *arbitrary*: *x e*)
case (*CIntegral* e' *t x e*)
note *IH* = *CIntegral.IH*[*of Suc x map-vars Suc e*]
have *free-vars* (*cexpr-comp-aux x e* $(\int_c e' \partial t)$) =
Suc −‘ *free-vars* (*cexpr-comp-aux* (*Suc x*) (*map-vars Suc e*) e') **by** *simp*
also have ... $\subseteq$ *Suc* −‘ (*free-vars* $e' - \{Suc\ x\}$ $\cup$ *free-vars* (*map-vars Suc e*))
by (*rule vimage-mono*, *rule CIntegral.IH*)
also have ... $\subseteq$ *free-vars* $(\int_c e' \partial t) - \{x\}$ $\cup$ *free-vars e*
by (*auto simp add*: *vimage-Diff vimage-image-eq*)
finally show *?case* **.**
qed (*simp*, *blast?*)+

lemma *free-vars-cexpr-comp*:
free-vars (*cexpr-comp e* e') $\subseteq$ (*free-vars e* $- \{0\}$) $\cup$ *free-vars* e'
by (*simp add*: *free-vars-cexpr-comp-aux cexpr-comp-def*)

lemma *free-vars-cexpr-comp'*:
free-vars (*cexpr-comp e* e') $\subseteq$ *free-vars e* $\cup$ *free-vars* e'
using *free-vars-cexpr-comp* **by** *blast*

lemma *cexpr-sem-cexpr-comp*:
cexpr-sem σ $(f \circ_c g)$ = *cexpr-sem* (σ($0 :=$ *cexpr-sem* σ *g*)) *f*
unfolding *cexpr-comp-def* **by** (*simp add*: *cexpr-sem-cexpr-comp-aux*)

lemma *eval-cexpr-comp*:
eval-cexpr ($f \circ_c g$) σ *x* = *eval-cexpr f* σ (*cexpr-sem* (*case-nat x* σ) *g*)
proof−
have (*case-nat x* σ)(*0* := *cexpr-sem* (*case-nat x* σ) *g*) = *case-nat* (*cexpr-sem* (*case-nat x* σ) *g*) σ
by (*intro ext*) (*auto split*: *nat.split*)
thus *?thesis* **by** (*simp add*: *eval-cexpr-def cexpr-sem-cexpr-comp*)
qed

primrec *cexpr-subst-val-aux* :: *nat* ⇒ *cexpr* ⇒ *val* ⇒ *cexpr* **where**
cexpr-subst-val-aux - (*CVal v*) - = *CVal v*
| *cexpr-subst-val-aux x* (*CVar y*) *v* = *insert-var x CVar* (*CVal v*) *y*
| *cexpr-subst-val-aux x* (IF_c *b THEN e1 ELSE e2*) *v* =
(IF_c *cexpr-subst-val-aux x b v THEN cexpr-subst-val-aux x e1 v ELSE cexpr-subst-val-aux x e2 v*)
| *cexpr-subst-val-aux x* (*oper* $\$\$_c$ *e*) *v* = *oper* $\$\$_c$ (*cexpr-subst-val-aux x e v*)
| *cexpr-subst-val-aux x* <*e1*, *e2*>$_c$ *v* = <*cexpr-subst-val-aux x e1 v*, *cexpr-subst-val-aux x e2 v*>$_c$
| *cexpr-subst-val-aux x* ($\int_c$ *e* ∂*t*) *v* = $\int_c$ *cexpr-subst-val-aux* (*Suc x*) *e v* ∂*t*

lemma *cexpr-subst-val-aux-eq-cexpr-subst*:
cexpr-subst-val-aux x e v = *cexpr-subst x* (*CVal v*) *e*
by (*induction e arbitrary*: *x*) *simp-all*

definition *cexpr-subst-val* :: *cexpr* ⇒ *val* ⇒ *cexpr* **where**
cexpr-subst-val e v ≡ *cexpr-subst-val-aux 0 e v*

lemma *cexpr-sem-cexpr-subst-val*[*simp*]:
cexpr-sem σ (*cexpr-subst-val e v*) = *cexpr-sem* (*case-nat v* σ) *e*
by (*simp add*: *cexpr-subst-val-def cexpr-subst-val-aux-eq-cexpr-subst cexpr-sem-cexpr-subst*)

lemma *cexpr-typing-subst-val*[*intro*]:
case-nat t Γ $\vdash_c$ *e* : *t′* ⟹ *val-type v* = *t* ⟹ Γ $\vdash_c$ *cexpr-subst-val e v* : *t′*
by (*auto simp*: *cexpr-subst-val-def cexpr-subst-val-aux-eq-cexpr-subst intro*!: *cet-val′*)

lemma *free-vars-cexpr-subst-val-aux*:
free-vars (*cexpr-subst-val-aux x e v*) = (λ*y*. *if y* ≥ *x then Suc y else y*) −‘ *free-vars e*
by (*induction e arbitrary*: *x*) (*auto simp*: *insert-var-def split*: *if-split-asm*)

lemma *free-vars-cexpr-subst-val*[*simp*]:
free-vars (*cexpr-subst-val e v*) = *Suc* −‘ *free-vars e*
by (*simp add*: *cexpr-subst-val-def free-vars-cexpr-subst-val-aux*)

8.2 Nonnegative expressions

definition *nonneg-cexpr V* Γ *e* ≡
∀σ ∈ *space* (*state-measure V* Γ). *extract-real* (*cexpr-sem* σ *e*) ≥ *0*

lemma *nonneg-cexprI*:
  $(\bigwedge\sigma.\ \sigma \in \textit{space}\ (\textit{state-measure}\ V\ \Gamma) \Longrightarrow \textit{extract-real}\ (\textit{cexpr-sem}\ \sigma\ e) \geq 0) \Longrightarrow$ $\textit{nonneg-cexpr}\ V\ \Gamma\ e$
  unfolding *nonneg-cexpr-def* **by** *simp*

lemma *nonneg-cexprD*:
  $\textit{nonneg-cexpr}\ V\ \Gamma\ e \Longrightarrow \sigma \in \textit{space}\ (\textit{state-measure}\ V\ \Gamma) \Longrightarrow \textit{extract-real}$ $(\textit{cexpr-sem}\ \sigma\ e) \geq 0$
  unfolding *nonneg-cexpr-def* **by** *simp*

lemma *nonneg-cexpr-map-vars*:
  assumes $\textit{nonneg-cexpr}\ (f\ -`\ V)\ (\Gamma \circ f)\ e$
  shows $\textit{nonneg-cexpr}\ V\ \Gamma\ (\textit{map-vars}\ f\ e)$
  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 $\textit{nonneg-cexpr}\ V\ \Gamma\ e\ V \subseteq V'\ \textit{free-vars}\ e \subseteq V$
  shows $\textit{nonneg-cexpr}\ V'\ \Gamma\ e$
proof (*intro nonneg-cexprI*)
  fix σ **assume** $\sigma \in \textit{space}\ (\textit{state-measure}\ V'\ \Gamma)$
  with *assms*(*2*) **have** $\textit{restrict}\ \sigma\ V \in \textit{space}\ (\textit{state-measure}\ V\ \Gamma)$
    by (*auto simp*: *state-measure-def space-PiM restrict-def*)
  from *nonneg-cexprD*[*OF assms*(*1*) *this*] **have** $\textit{extract-real}\ (\textit{cexpr-sem}\ (\textit{restrict}\ \sigma$ $V)\ e) \geq 0$ **.**
  also have $\textit{cexpr-sem}\ (\textit{restrict}\ \sigma\ V)\ e = \textit{cexpr-sem}\ \sigma\ e$ **using** *assms*(*3*)
    by (*intro cexpr-sem-eq-on-vars*) *auto*
  finally show $\textit{extract-real}\ (\textit{cexpr-sem}\ \sigma\ e) \geq 0$ **.**
qed

lemma *nonneg-cexpr-Mult*:
  assumes $\Gamma \vdash_c e1 : \textit{REAL}\ \Gamma \vdash_c e2 : \textit{REAL}$
  assumes $\textit{free-vars}\ e1 \subseteq V\ \textit{free-vars}\ e2 \subseteq V$
  assumes *N1*: $\textit{nonneg-cexpr}\ V\ \Gamma\ e1$ **and** *N2*: $\textit{nonneg-cexpr}\ V\ \Gamma\ e2$
  shows $\textit{nonneg-cexpr}\ V\ \Gamma\ (e1 *_c e2)$
proof (*rule nonneg-cexprI*)
  fix σ **assume** σ: $\sigma \in \textit{space}\ (\textit{state-measure}\ V\ \Gamma)$
  hence $\textit{extract-real}\ (\textit{cexpr-sem}\ \sigma\ (e1 *_c e2)) = \textit{extract-real}\ (\textit{cexpr-sem}\ \sigma\ e1) *$ $\textit{extract-real}\ (\textit{cexpr-sem}\ \sigma\ e2)$
    using *assms* **by** (*subst cexpr-sem-Mult*[*of* Γ - - - V]) *simp-all*
  also have ... ≥ 0 **using** σ *N1 N2* **by** (*intro mult-nonneg-nonneg nonneg-cexprD*)
  finally show $\textit{extract-real}\ (\textit{cexpr-sem}\ \sigma\ (e1 *_c e2)) \geq 0$ **.**
qed

lemma *nonneg-indicator*:
  assumes $\Gamma \vdash_c e : \textit{BOOL}\ \textit{free-vars}\ e \subseteq V$
  shows $\textit{nonneg-cexpr}\ V\ \Gamma\ (\langle e\rangle_c)$
proof (*intro nonneg-cexprI*)
  fix ϱ **assume** $\varrho \in \textit{space}\ (\textit{state-measure}\ V\ \Gamma)$

with *assms* **have** *val-type* (*cexpr-sem* ϱ *e*) = *BOOL* **by** (*rule val-type-cexpr-sem*)
thus *extract-real* (*cexpr-sem* ϱ ($\langle e \rangle_c$)) $\geq$ *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* (Γ(*x* := *t1*)) *e* **and** *x*:*x* $\in$ *V*
assumes *t2*: Γ(*x*:=*t1*) $\vdash_c$ *e* : *t2* **and** *t1*: Γ $\vdash_c$ *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: $\sigma \in$ *space* (*state-measure* *V* Γ)
have *extract-real* (*cexpr-sem* σ (*cexpr-comp-aux x f e*)) =
extract-real (*cexpr-sem* (σ(*x* := *cexpr-sem* σ *f*)) *e*)
by (*simp add*: *cexpr-sem-cexpr-comp-aux*)
also from *val-type-cexpr-sem*[*OF t1 vars* σ] **have** *cexpr-sem* σ *f* $\in$ *type-universe* *t1* **by** *auto*
with σ *x* **have** σ(*x* := *cexpr-sem* σ *f*) $\in$ *space* (*state-measure* *V* (Γ(*x* := *t1*)))
by (*auto simp*: *state-measure-def space-PiM shift-var-set-def split*: *if-split-asm*)
hence *extract-real* (*cexpr-sem* (σ(*x* := *cexpr-sem* σ *f*)) *e*) $\geq$ *0*
by(*intro nonneg-cexprD*[*OF assms*(*1*)])
finally show *extract-real* (*cexpr-sem* σ (*cexpr-comp-aux x f e*)) $\geq$ *0* **.**
qed

lemma *nonneg-cexpr-comp*:
assumes *nonneg-cexpr* (*shift-var-set* *V*) (*case-nat* *t2* Γ) *e*
assumes *case-nat* *t1* Γ $\vdash_c$ *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: $\sigma \in$ *space* (*state-measure* (*shift-var-set* *V*) (*case-nat* *t1* Γ))
have *extract-real* (*cexpr-sem* σ (*e* $\circ_c$ *f*)) = *extract-real* (*cexpr-sem* (σ(*0* := *cexpr-sem* σ *f*)) *e*)
by (*simp add*: *cexpr-sem-cexpr-comp*)
also from *val-type-cexpr-sem*[*OF assms*(*2*,*3*) σ] **have** *cexpr-sem* σ *f* $\in$ *type-universe* *t2* **by** *auto*
with σ **have** σ(*0* := *cexpr-sem* σ *f*) $\in$ *space* (*state-measure* (*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* (σ(*0* := *cexpr-sem* σ *f*)) *e*) $\geq$ *0*
by(*intro nonneg-cexprD*[*OF assms*(*1*)])
finally show *extract-real* (*cexpr-sem* σ (*e* $\circ_c$ *f*)) $\geq$ *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: $\sigma \in$ *space* (*state-measure* *V* Γ)
moreover from *assms*(*2*) **have** *v* $\in$ *type-universe* *t* **by** *auto*
ultimately show *extract-real* (*cexpr-sem* σ (*cexpr-subst-val* *e* *v*)) $\geq$ *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* ∂*t*)
proof (*intro nonneg-cexprI*)
fix σ **assume** σ: σ ∈ *space* (*state-measure V* Γ)
have *extract-real* (*cexpr-sem* σ ($\int_c$ *e* ∂*t*)) = ∫ *x*. *extract-real* (*cexpr-sem* (*case-nat x* σ) *e*) ∂*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* ∂*t*)) ≥ *0* **.**
qed

Subprobability density expressions

definition *subprob-cexpr V V′* Γ *e* ≡
∀ ϱ ∈ *space* (*state-measure V′* Γ).
($\int^+$σ. *extract-real* (*cexpr-sem* (*merge V V′* (σ, ϱ)) *e*) ∂*state-measure V* Γ) ≤ *1*

lemma *subprob-cexprI*:
assumes ⋀ϱ. ϱ ∈ *space* (*state-measure V′* Γ) ⟹
($\int^+$σ. *extract-real* (*cexpr-sem* (*merge V V′* (σ, ϱ)) *e*) ∂*state-measure V* Γ) ≤ *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 ⋀ϱ. ϱ ∈ *space* (*state-measure V′* Γ) ⟹
($\int^+$σ. *extract-real* (*cexpr-sem* (*merge V V′* (σ, ϱ)) *e*) ∂*state-measure V* Γ) ≤ *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* ∪ *V′*) Γ *e1*
assumes *t1*: Γ $\vdash_c$ *e1* : *REAL* **and** *t2*: Γ $\vdash_c$ *e2* : *BOOL*
assumes *vars1*: *free-vars e1* ⊆ *V* ∪ *V′* **and** *vars2*: *free-vars e2* ⊆ *V* ∪ *V′*
shows *subprob-cexpr V V′* Γ (*e1* $*_c$ $\langle e2\rangle_c$)
proof (*intro subprob-cexprI*)
fix ϱ **assume** ϱ: ϱ ∈ *space* (*state-measure V′* Γ)
from *t2* **have** *t2′*: Γ $\vdash_c$ $\langle e2\rangle_c$: *REAL* **by** (*rule cet-op*) *simp-all*
from *vars2* **have** *vars2′*: *free-vars* ($\langle e2\rangle_c$) ⊆ *V* ∪ *V′* **by** *simp*
let *?eval* = λσ *e*. *extract-real* (*cexpr-sem* (*merge V V′* (σ, ϱ)) *e*)
have ($\int^+$σ. *?eval* σ (*e1* $*_c$ $\langle e2\rangle_c$) ∂*state-measure V* Γ) =
($\int^+$σ. *?eval* σ *e1* ∗ *?eval* σ ($\langle e2\rangle_c$) ∂*state-measure V* Γ)

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** σ: $\sigma \in$ *space* (*state-measure V* Γ)
with ϱ **have** *val-type* (*cexpr-sem* (*merge V V′* (σ,ϱ)) *e2*) = *BOOL*
by (*intro val-type-cexpr-sem*[*OF t2 vars2*] *merge-in-state-measure*)
hence *?eval* σ ($\langle e2\rangle_c$) $\in$ {*0*,*1*}
by (*cases cexpr-sem* (*merge V V′* (σ,ϱ)) *e2*) (*auto simp*: *extract-real-def bool-to-real-def*)
moreover have *?eval* σ *e1* $\geq$ *0* **using** *nonneg* ϱ σ
by (*auto intro*!: *nonneg-cexprD merge-in-state-measure*)
ultimately have *?eval* σ *e1* $*$ *?eval* σ ($\langle e2\rangle_c$) $\leq$ *?eval* σ *e1*
by (*intro mult-right-le-one-le*) *auto*
}
hence ($\int^+\sigma$. *?eval* σ *e1* $*$ *?eval* σ ($\langle e2\rangle_c$) ∂*state-measure V* Γ) $\leq$
($\int^+\sigma$. *?eval* σ *e1* ∂*state-measure V* Γ)
by (*intro nn-integral-mono*) (*simp add*: *ennreal-leI*)
also from *subprob* **and** ϱ **have** ... $\leq$ *1* **by** (*rule subprob-cexprD*)
finally show ($\int^+\sigma$. *?eval* σ (*e1* $*_c$ $\langle e2\rangle_c$) ∂*state-measure V* Γ) $\leq$ *1* **.**
qed

lemma *measurable-cexpr-sem′*:
assumes ϱ: $\varrho \in$ *space* (*state-measure V′* Γ)
assumes *e*: Γ $\vdash_c$ *e* : *REAL free-vars e* $\subseteq$ *V* $\cup$ *V′*
shows ($\lambda\sigma$. *extract-real* (*cexpr-sem* (*merge V V′* (σ, ϱ)) *e*))
$\in$ *borel-measurable* (*state-measure V* Γ)
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 (λ(*x*,*y*). *y*(*v* := *x*)) $\in$ *measurable* (*stock-measure* (Γ *v*) $\bigotimes_M$ *state-measure V* Γ)
(*state-measure* (*insert v V*) Γ)
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 ϱ: $\varrho \in$ *space* (*state-measure V′* Γ)
assumes *e*: Γ $\vdash_c$ *e* : *REAL free-vars e* $\subseteq$ *insert v V* $\cup$ *V′*
assumes *int*: *integrable* (*state-measure* (*insert v V*) Γ)
($\lambda\sigma$. *extract-real* (*cexpr-sem* (*merge* (*insert v V*) *V′* (σ, ϱ)) *e*))
(**is** *integrable - ?f′*)

shows *AE x in stock-measure* (Γ *v*).
integrable (*state-measure V* Γ)
(λσ. *extract-real* (*cexpr-sem* (*merge V* (*insert v V'*) (σ, ϱ(*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* ∈ *borel-measurable ?M* **using** ϱ *e disjoint*
by (*intro AE-I2 measurable-cexpr-sem'*)
(*auto simp*: *state-measure-def space-PiM dest*: *PiE-mem split*: *if-split-asm*)

let *?f''* = λ*x* σ. *extract-real* (*cexpr-sem* (*merge* (*insert v V*) *V'* (σ(*v* := *x*), ϱ)) *e*)
{
fix *x* σ **assume** *x* ∈ *space ?N* σ ∈ *space ?M*
hence *merge* (*insert v V*) *V'* (σ(*v* := *x*), ϱ) = *merge V* (*insert v V'*) (σ, ϱ(*v* := *x*))
using *disjoint* **by** (*intro ext*) (*simp add*: *merge-def split*: *if-split-asm*)
hence *?f'' x* σ = *?f x* σ **by** *simp*
} **note** *f''-eq-f* = *this*

interpret *product-sigma-finite* (λ*v*. *stock-measure* (Γ *v*))
by (*simp add*: *product-sigma-finite-def*)
interpret *sigma-finite-measure state-measure V* Γ
by (*rule sigma-finite-state-measure*[*OF fin*])

from *int* **have** (∫⁺σ. *ennreal* (*?f'* σ) ∂*state-measure* (*insert v V*) Γ) ≠ ∞
by (*simp add*: *real-integrable-def*)
also have (∫⁺σ. *ennreal* (*?f'* σ) ∂*state-measure* (*insert v V*) Γ) =
∫⁺*x*. ∫⁺σ. *ennreal* (*?f'' x* σ) ∂*?M* ∂*?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*] *measurable-cexpr-sem'*[*unfolded state-measure-def*])
finally have *AE x in ?N*. (∫⁺σ. *ennreal* (*?f'' x* σ) ∂*?M*) ≠ ∞ (**is** *?P*) **using** *e disjoint*
by (*intro nn-integral-PInf-AE*)
(*auto simp*: *measurable-split-conv intro*!: *borel-measurable-nn-integral measurable-compose*[*OF - measurable-ennreal*]
measurable-compose[*OF - measurable-cexpr-sem'*[*OF* ϱ]])
moreover have ⋀*x*. *x* ∈ *space ?N* ⟹ (∫⁺σ. *ennreal* (*?f'' x* σ) ∂*?M*) = (∫⁺σ. *ennreal* (*?f x* σ) ∂*?M*)
by (*intro nn-integral-cong*) (*simp add*: *f''-eq-f*)
hence *?P* ⟷ (*AE x in ?N*. (∫⁺σ. *ennreal* (*?f x* σ) ∂*?M*) ≠ ∞) **by** (*intro AE-cong*) *simp*
ultimately show *AE x in ?N*. (∫⁺σ. *ennreal* (*?f x* σ) ∂*?M*) ≠ ∞ **by** *simp*

from *int* **have** (∫⁺σ. *ennreal* (−*?f'* σ) ∂*state-measure* (*insert v V*) Γ) ≠ ∞
by (*simp add*: *real-integrable-def*)
also have (∫⁺σ. *ennreal* (−*?f'* σ) ∂*state-measure* (*insert v V*) Γ) =
∫⁺*x*. ∫⁺σ. *ennreal* (−*?f'' x* σ) ∂*?M* ∂*?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* σ) ∂*?M*) $\neq \infty$ (**is** *?P*) **using** *e disjoint*
by (*intro nn-integral-PInf-AE*)
(*auto simp*: *measurable-split-conv intro*!: *borel-measurable-nn-integral measurable-compose*[*OF* - *measurable-ennreal*]
measurable-compose[*OF* - *measurable-cexpr-sem*′[*OF* ϱ]] *borel-measurable-uminus*)
moreover have $\bigwedge$*x*. *x* $\in$ *space ?N* $\Longrightarrow$ ($\int^+\sigma$. *ennreal* (−*?f*′′ *x* σ) ∂*?M*) = ($\int^+\sigma$. *ennreal* (−*?f x* σ) ∂*?M*)
by (*intro nn-integral-cong*) (*simp add*: *f*′′*-eq-f*)
hence *?P* $\longleftrightarrow$ (*AE x in ?N*. ($\int^+\sigma$. *ennreal* (−*?f x* σ) ∂*?M*) $\neq \infty$) **by** (*intro AE-cong*) *simp*
ultimately show *AE x in ?N*. ($\int^+\sigma$. *ennreal* (−*?f x* σ) ∂*?M*) $\neq \infty$ **by** *simp*
qed

definition *cdens-ctxt-invar* :: *vname list* ⇒ *vname list* ⇒ *tyenv* ⇒ *cexpr* ⇒ *bool*
where
cdens-ctxt-invar vs vs′ Γ δ ≡
distinct (*vs* @ *vs*′) ∧
free-vars δ ⊆ *set* (*vs* @ *vs*′) ∧
Γ $\vdash_c$ δ : *REAL* ∧
nonneg-cexpr (*set vs* ∪ *set vs*′) Γ δ ∧
subprob-cexpr (*set vs*) (*set vs*′) Γ δ

lemma *cdens-ctxt-invarI*:
⟦*distinct* (*vs* @ *vs*′); *free-vars* δ ⊆ *set* (*vs* @ *vs*′); Γ $\vdash_c$ δ : *REAL*;
nonneg-cexpr (*set vs* ∪ *set vs*′) Γ δ;
subprob-cexpr (*set vs*) (*set vs*′) Γ δ ⟧ ⟹
cdens-ctxt-invar vs vs′ Γ δ
by (*simp add*: *cdens-ctxt-invar-def*)

lemma *cdens-ctxt-invarD*:
assumes *cdens-ctxt-invar vs vs*′ Γ δ
shows *distinct* (*vs* @ *vs*′) *free-vars* δ ⊆ *set* (*vs* @ *vs*′) Γ $\vdash_c$ δ : *REAL*
nonneg-cexpr (*set vs* ∪ *set vs*′) Γ δ *subprob-cexpr* (*set vs*) (*set vs*′) Γ δ
using *assms* **by** (*simp-all add*: *cdens-ctxt-invar-def*)

lemma *cdens-ctxt-invar-empty*:
assumes *cdens-ctxt-invar vs vs*′ Γ δ
shows *cdens-ctxt-invar* [] (*vs* @ *vs*′) Γ (*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: ϱ ∈ *space* (*state-measure* (*set vs′*) Γ)
shows *integrable* (*state-measure* (*set vs*) Γ)
(λσ. *extract-real* (*cexpr-sem* (*merge* (*set vs*) (*set vs′*) (σ, ϱ)) δ)) (**is** *integrable ?M ?f*)
unfolding *integrable-iff-bounded*
proof (*intro conjI*)
note *invar* = *cdens-ctxt-invarD*[*OF assms*(*1*)]
show *?f* ∈ *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*: ⋀σ. σ ∈ *space ?M* ⟹ *?f* σ ≥ *0*
using ‹*nonneg-cexpr* (*set vs* ∪ *set vs′*) Γ δ›
by (*rule nonneg-cexprD*, *intro merge-in-state-measure*[*OF - ϱ*])
with ‹*subprob-cexpr* (*set vs*) (*set vs′*) Γ δ› **and** ϱ
show ($\int^{+}$σ. *ennreal* (*norm* (*?f* σ)) ∂*?M*) < ∞ **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* ⇒ *cexpr* **where**
expr-rf-to-cexpr (*Val v*) = *CVal v*
| *expr-rf-to-cexpr* (*Var x*) = *CVar x*
| *expr-rf-to-cexpr* <*e1*, *e2*> = <*expr-rf-to-cexpr e1*, *expr-rf-to-cexpr e2*>$_c$
| *expr-rf-to-cexpr* (*oper* $$ *e*) = *oper* $$$_c$ (*expr-rf-to-cexpr e*)
| *expr-rf-to-cexpr* (*IF b THEN e1 ELSE e2*) =
(*IF*$_c$ *expr-rf-to-cexpr b THEN expr-rf-to-cexpr e1 ELSE expr-rf-to-cexpr e2*)
| *expr-rf-to-cexpr* (*LET e1 IN e2*) =
cexpr-subst 0 (*expr-rf-to-cexpr e1*) (*expr-rf-to-cexpr e2*)
| *expr-rf-to-cexpr* (*Random - -*) = *undefined*
| *expr-rf-to-cexpr* (*Fail -*) = *undefined*

lemma *cexpr-sem-expr-rf-to-cexpr*:
randomfree e ⟹ *cexpr-sem* σ (*expr-rf-to-cexpr e*) = *expr-sem-rf* σ *e*
by (*induction e arbitrary*: σ) (*auto simp*: *cexpr-sem-cexpr-subst*)

lemma *cexpr-typing-expr-rf-to-cexpr*[*intro*]:
assumes Γ ⊢ *e* : *t randomfree e*
shows Γ ⊢$_c$ *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 $\Longrightarrow$ *free-vars* (*expr-rf-to-cexpr e*) $\subseteq$ *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 = (IF_c *CReal 0* $\leq_c$ *p* $\wedge_c$ *p* $\leq_c$ *CReal 1 THEN*
IF_c *x THEN p ELSE CReal 1* $-_c$ *p*
ELSE CReal 0)
| *dist-dens-cexpr UniformInt p x* = (IF_c fst_c *p* $\leq_c$ snd_c *p* $\wedge_c$ fst_c *p* $\leq_c$ *x* $\wedge_c$ *x* $\leq_c$ snd_c *p THEN*
$inverse_c$ ($\langle snd_c$ *p* $-_c$ fst_c *p* $+_c$ *CInt 1*$\rangle_c$) *ELSE CReal 0*)
| *dist-dens-cexpr UniformReal p x* = (IF_c fst_c *p* $<_c$ snd_c *p* $\wedge_c$ fst_c *p* $\leq_c$ *x* $\wedge_c$ *x* $\leq_c$ snd_c *p THEN*
$inverse_c$ (snd_c *p* $-_c$ fst_c *p*) *ELSE CReal 0*)
| *dist-dens-cexpr Gaussian p x* = (IF_c *CReal 0* $<_c$ snd_c *p THEN*
exp_c ($-_c$((*x* $-_c$ fst_c *p*)$\widehat{\ }_c$*CInt 2* $/_c$ (*CReal 2* $*_c$ snd_c *p*$\widehat{\ }_c$*CInt 2*))) $/_c$
$sqrt_c$ (*CReal 2* $*_c$ π_c $*_c$ snd_c *p* $\widehat{\ }_c$ *CInt 2*) *ELSE CReal 0*)
| *dist-dens-cexpr Poisson p x* = (IF_c *CReal 0* $<_c$ *p* $\wedge_c$ *CInt 0* $\leq_c$ *x THEN*
p $\widehat{\ }_c$ *x* $/_c$ $\langle fact_c\ x\rangle_c$ $*_c$ exp_c ($-_c$ *p*) *ELSE CReal 0*)

lemma *free-vars-dist-dens-cexpr*:
free-vars (*dist-dens-cexpr dst e1 e2*) $\subseteq$ *free-vars e1* $\cup$ *free-vars e2*
by (*subst dist-dens-cexpr-def*, *cases dst*) *simp-all*

lemma *cexpr-typing-dist-dens-cexpr*:
assumes $\Gamma \vdash_c$ *e1* : *dist-param-type dst* $\Gamma \vdash_c$ *e2* : *dist-result-type dst*
shows $\Gamma \vdash_c$ *dist-dens-cexpr dst e1 e2* : *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* = *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* ⟷ *x* ∈ *BoolVal‘UNIV*
by (*cases x*) *auto*

lemma *val-type-eq-INTEG*: *val-type x* = *INTEG* ⟷ *x* ∈ *IntVal‘UNIV*
by (*cases x*) *auto*

lemma *val-type-eq-PRODUCT*: *val-type x* = *PRODUCT t1 t2* ⟷
(∃ *a b*. *val-type a* = *t1* ∧ *val-type b* = *t2* ∧ *x* = <| *a*, *b* |>)
by (*cases x*) *auto*

lemma *cexpr-sem-dist-dens-cexpr-nonneg*:
assumes $\Gamma \vdash_c$ *e1* : *dist-param-type dst* $\Gamma \vdash_c$ *e2* : *dist-result-type dst*
assumes *free-vars e1* ⊆ *V free-vars e2* ⊆ *V*
assumes σ ∈ *space* (*state-measure V* Γ)
shows *ennreal* (*extract-real* (*cexpr-sem* σ (*dist-dens-cexpr dst e1 e2*))) =
dist-dens dst (*cexpr-sem* σ *e1*) (*cexpr-sem* σ *e2*) ∧
0 ≤ *extract-real* (*cexpr-sem* σ (*dist-dens-cexpr dst e1 e2*))
proof−
from *val-type-cexpr-sem*[*OF assms*(*1*,*3*,*5*)] **and** *val-type-cexpr-sem*[*OF assms*(*2*,*4*,*5*)]
have *cexpr-sem* σ *e1* ∈ *space* (*stock-measure* (*dist-param-type dst*)) **and**
cexpr-sem σ *e2* ∈ *space* (*stock-measure* (*dist-result-type dst*))
by (*auto simp*: *type-universe-def simp del*: *type-universe-type*)
thus *?thesis*
by (*subst dist-dens-cexpr-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 *cexpr-sem-dist-dens-cexpr*:
assumes $\Gamma \vdash_c$ *e1* : *dist-param-type dst* $\Gamma \vdash_c$ *e2* : *dist-result-type dst*
assumes *free-vars e1* ⊆ *V free-vars e2* ⊆ *V*
assumes σ ∈ *space* (*state-measure V* Γ)

shows *ennreal* (*extract-real* (*cexpr-sem* σ (*dist-dens-cexpr dst e1 e2*))) =
dist-dens dst (*cexpr-sem* σ *e1*) (*cexpr-sem* σ *e2*)
using *cexpr-sem-dist-dens-cexpr-nonneg*[*OF assms*] **by** *simp*

lemma *nonneg-dist-dens-cexpr*:
assumes $\Gamma \vdash_c$ *e1* : *dist-param-type dst* $\Gamma \vdash_c$ *e2* : *dist-result-type dst*
assumes *free-vars e1* $\subseteq$ *V free-vars e2* $\subseteq$ *V*
shows *nonneg-cexpr V* Γ (*dist-dens-cexpr dst e1 e2*)
proof (*intro nonneg-cexprI*)
fix σ **assume** ϱ: $\sigma \in$ *space* (*state-measure V* Γ)
from *cexpr-sem-dist-dens-cexpr-nonneg*[*OF assms this*]
show *0* $\leq$ *extract-real* (*cexpr-sem* σ (*dist-dens-cexpr dst e1 e2*))
by *simp*
qed

8.5 Integral expressions

definition *integrate-var* :: *tyenv* $\Rightarrow$ *vname* $\Rightarrow$ *cexpr* $\Rightarrow$ *cexpr* **where**
integrate-var Γ *v e* = $\int_c$ *map-vars* (λw. *if v* = *w then 0 else Suc w*) *e* ∂(Γ *v*)

definition *integrate-vars* :: *tyenv* $\Rightarrow$ *vname list* $\Rightarrow$ *cexpr* $\Rightarrow$ *cexpr* **where**
integrate-vars Γ = *foldr* (*integrate-var* Γ)

lemma *cexpr-sem-integrate-var*:
cexpr-sem σ (*integrate-var* Γ *v e*) =
RealVal ($\int$ *x*. *extract-real* (*cexpr-sem* (σ(*v* := *x*)) *e*) ∂*stock-measure* (Γ *v*))
proof−
let *?f* = (λw. *if v* = *w then 0 else Suc w*)
have *cexpr-sem* σ (*integrate-var* Γ *v e*) =
RealVal ($\int$ *x*. *extract-real* (*cexpr-sem* (*case-nat x* $\sigma \circ$ *?f*) *e*) ∂*stock-measure*
(Γ *v*))
by (*simp add*: *extract-real-def integrate-var-def cexpr-sem-map-vars*)
also have (λx. *case-nat x* $\sigma \circ$ *?f*) = (λx. σ(*v* := *x*))
by (*intro ext*) (*simp add*: *o-def split*: *if-split*)
finally show *?thesis* **.**
qed

lemma *cexpr-sem-integrate-var'*:
extract-real (*cexpr-sem* σ (*integrate-var* Γ *v e*)) =
($\int$ *x*. *extract-real* (*cexpr-sem* (σ(*v* := *x*)) *e*) ∂*stock-measure* (Γ *v*))
by (*subst cexpr-sem-integrate-var*, *simp add*: *extract-real-def*)

lemma *cexpr-typing-integrate-var*[*simp*]:
$\Gamma \vdash_c$ *e* : *REAL* $\Longrightarrow$ $\Gamma \vdash_c$ *integrate-var* Γ *v e* : *REAL*
unfolding *integrate-var-def*
by (*rule cexpr-typing.intros*, *rule cexpr-typing-map-vars*)
(*erule cexpr-typing-cong'*, *simp split*: *nat.split*)

lemma *cexpr-typing-integrate-vars*[*simp*]:

$\Gamma \vdash_c e : REAL \Longrightarrow \Gamma \vdash_c$ *integrate-vars* Γ *vs e* : *REAL*
by (*induction vs arbitrary*: *e*)
(*simp-all add*: *integrate-vars-def*)

lemma *free-vars-integrate-var*[*simp*]:
free-vars (*integrate-var* Γ *v e*) = *free-vars e* − {*v*}
by (*auto simp*: *integrate-var-def*)

lemma *free-vars-integrate-vars*[*simp*]:
free-vars (*integrate-vars* Γ *vs e*) = *free-vars e* − *set vs*
by (*induction vs arbitrary*: *e*) (*auto simp*: *integrate-vars-def*)

lemma (**in** *product-sigma-finite*) *product-integral-insert*′:
fixes *f* :: - ⇒ *real*
assumes *finite I i* ∉ *I integrable* (Pi_M (*insert i I*) *M*) *f*
shows $integral^L$ (Pi_M (*insert i I*) *M*) *f* = *LINT y*|*M i*. *LINT x*|Pi_M *I M*. *f* (*x*(*i* := *y*))
proof−
interpret *pair-sigma-finite M i* Pi_M *I M*
by (*simp-all add*: *sigma-finite assms pair-sigma-finite-def sigma-finite-measures*)
interpret *Mi*: *sigma-finite-measure M i*
by (*simp add*: *assms sigma-finite-measures*)
from *assms*(*3*) **have** *int*: *integrable* (*M i* $\bigotimes_M$ Pi_M *I M*) (λ(*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** $integral^L$ (Pi_M (*insert i I*) *M*) *f* = *LINT x*|Pi_M *I M*. *LINT y*|*M i*. *f* (*x*(*i* := *y*))
by (*rule product-integral-insert*)
also from *int* **have** ... = *LINT y*|*M i*. *LINT x*|Pi_M *I M*. *f* (*x*(*i* := *y*))
by (*rule Fubini-integral*)
finally show *?thesis* **.**
qed

lemma *cexpr-sem-integrate-vars*:
assumes ϱ: ϱ ∈ *space* (*state-measure V*′ Γ)
assumes *disjoint*: *distinct vs set vs* ∩ *V*′ = {}
assumes *integrable* (*state-measure* (*set vs*) Γ)
(λσ. *extract-real* (*cexpr-sem* (*merge* (*set vs*) *V*′ (σ, ϱ)) *e*))
assumes *e*: $\Gamma \vdash_c e$: *REAL free-vars e* ⊆ *set vs* ∪ *V*′
shows *extract-real* (*cexpr-sem* ϱ (*integrate-vars* Γ *vs e*)) =
∫ σ. *extract-real* (*cexpr-sem* (*merge* (*set vs*) *V*′ (σ, ϱ)) *e*) ∂*state-measure* (*set vs*) Γ
using *assms*

proof (*induction vs arbitrary*: ϱ *V'*)
case *Nil*
hence $\bigwedge v.$ (*if v* $\in$ *V' then* ϱ *v else undefined*) = ϱ *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* ϱ *V'*)
interpret *product-sigma-finite* $\lambda v.$ *stock-measure* (Γ *v*)
by (*simp add*: *product-sigma-finite-def*)
interpret *sigma-finite-measure state-measure* (*set vs*) Γ
by (*simp add*: *sigma-finite-state-measure*)
have ϱ': $\bigwedge x.$ *x* $\in$ *type-universe* (Γ *v*) $\Longrightarrow$ $\varrho(v := x)$ $\in$ *space* (*state-measure* (*insert v V'*) Γ)
using *Cons.prems*(*1*) **by** (*auto simp*: *state-measure-def space-PiM split*: *if-split-asm*)
have *extract-real* (*cexpr-sem* ϱ (*integrate-vars* Γ (*v* # *vs*) *e*)) =
$\int$ *x*. *extract-real* (*cexpr-sem* ($\varrho(v := x)$) (*integrate-vars* Γ *vs e*)) ∂*stock-measure* (Γ *v*)
(**is** - = *?I*) **by** (*simp add*: *integrate-vars-def cexpr-sem-integrate-var extract-real-def*)
also from *Cons.prems*(*4*) **have** *int*: *integrable* (*state-measure* (*insert v* (*set vs*)) Γ)
($\lambda\sigma.$ *extract-real* (*cexpr-sem* (*merge* (*insert v* (*set vs*)) *V'* (σ, ϱ)) *e*)) **by** *simp*
have *AE x in stock-measure* (Γ *v*).
extract-real (*cexpr-sem* ($\varrho(v := x)$) (*integrate-vars* Γ *vs e*)) =
$\int$ σ. *extract-real* (*cexpr-sem* (*merge* (*set vs*) (*insert v V'*) (σ, $\varrho(v := x)$)) *e*)
∂*state-measure* (*set vs*) Γ
apply (*rule AE-mp*[*OF - AE-I2*[*OF impI*]])
apply (*rule integrable-cexpr-projection*[*OF - - - - - - - int*])
apply (*insert Cons.prems*, *auto*) [*7*]
apply (*subst Cons.IH*, *rule* ϱ', *insert Cons.prems*, *auto*)
done
hence *?I* = $\int$ *x*. $\int$ σ. *extract-real* (*cexpr-sem* (*merge* (*set vs*) (*insert v V'*) (σ, $\varrho(v := x)$)) *e*)
∂*state-measure* (*set vs*) Γ ∂*stock-measure* (Γ *v*) **using** *Cons.prems*
apply (*intro integral-cong-AE*)
apply (*rule measurable-compose*[*OF measurable-Pair-compose-split*[*OF measurable-fun-upd-state-measure*[*of v V'* Γ]]])
apply (*simp*, *simp*, *simp*, *rule measurable-compose*[*OF - measurable-extract-real*])
apply (*rule measurable-cexpr-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-cexpr-sem*[*of* Γ *- - set vs* $\cup$ *insert v V'*]])
apply (*unfold state-measure-def*, *rule measurable-compose*[*OF - measurable-merge*])
apply *simp-all*
done
also have (λx σ. *merge* (*set vs*) (*insert v V'*) (σ, $\varrho(v := x)$)) =
(λx σ. *merge* (*set* (*v#vs*)) *V'* ($\sigma(v := x)$, ϱ))

using *Cons.prems* **by** (*intro ext*) (*auto simp*: *merge-def split*: *if-split*)
also have ($\int x.\ \int \sigma.$ *extract-real* (*cexpr-sem* (*merge* (*set* (*v#vs*)) V' (σ(*v* := *x*), ϱ)) *e*)
∂*state-measure* (*set vs*) Γ ∂*stock-measure* (Γ *v*)) =
$\int \sigma.$ *extract-real* (*cexpr-sem* (*merge* (*set* (*v#vs*)) V' (σ, ϱ)) *e*)
∂*state-measure* (*set* (*v#vs*)) Γ
using *Cons.prems* **unfolding** *state-measure-def*
by (*subst* (*2*) *set-simps*, *subst product-integral-insert'*) *simp-all*
finally show *?case* **.**
qed

lemma *cexpr-sem-integrate-vars'*:
assumes ϱ: $\varrho \in$ *space* (*state-measure* V' Γ)
assumes *disjoint*: *distinct vs set vs* $\cap$ V' = {}
assumes *nonneg*: *nonneg-cexpr* (*set vs* $\cup$ V') Γ *e*
assumes *integrable* (*state-measure* (*set vs*) Γ)
($\lambda\sigma.$ *extract-real* (*cexpr-sem* (*merge* (*set vs*) V' (σ, ϱ)) *e*))
assumes *e*: $\Gamma \vdash_c e$: *REAL free-vars e* $\subseteq$ *set vs* $\cup$ V'
shows *ennreal* (*extract-real* (*cexpr-sem* ϱ (*integrate-vars* Γ *vs e*))) =
$\int^{+}\sigma.$ *extract-real* (*cexpr-sem* (*merge* (*set vs*) V' (σ, ϱ)) *e*) ∂*state-measure* (*set vs*) Γ
proof−
from *assms* **have** *extract-real* (*cexpr-sem* ϱ (*integrate-vars* Γ *vs e*)) =
$\int \sigma.$ *extract-real* (*cexpr-sem* (*merge* (*set vs*) V' (σ, ϱ)) *e*) ∂*state-measure* (*set vs*) Γ
by (*intro cexpr-sem-integrate-vars*)
also have *ennreal* ... =
$\int^{+}\sigma.$ *extract-real* (*cexpr-sem* (*merge* (*set vs*) V' (σ, ϱ)) *e*) ∂*state-measure* (*set vs*) Γ
using *assms*
by (*intro nn-integral-eq-integral*[*symmetric*] *AE-I2*)
(*auto intro*!: *nonneg-cexprD merge-in-state-measure*)
finally show *?thesis* **.**
qed

lemma *nonneg-cexpr-sem-integrate-vars*:
assumes ϱ: $\varrho \in$ *space* (*state-measure* V' Γ)
assumes *disjoint*: *distinct vs set vs* $\cap$ V' = {}
assumes *nonneg*: *nonneg-cexpr* (*set vs* $\cup$ V') Γ *e*
assumes *e*: $\Gamma \vdash_c e$: *REAL free-vars e* $\subseteq$ *set vs* $\cup$ V'
shows *extract-real* (*cexpr-sem* ϱ (*integrate-vars* Γ *vs e*)) $\geq$ *0*
using *assms*
proof (*induction vs arbitrary*: ϱ V')
case *Nil*
hence $\bigwedge v.$ (*if* $v \in V'$ *then* ϱ *v else undefined*) = ϱ *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 nonneg-cexprD*)

next
case (*Cons v vs* ϱ V')
have ϱ': $\bigwedge x.\ x \in$ *type-universe* (Γ *v*) $\Longrightarrow$ $\varrho(v := x) \in$ *space* (*state-measure* (*insert v* V') Γ)
using *Cons.prems*(*1*) **by** (*auto simp*: *state-measure-def space-PiM split*: *if-split-asm*)
have *extract-real* (*cexpr-sem* ϱ (*integrate-vars* Γ (*v* # *vs*) *e*)) =
$\int$ *x. extract-real* (*cexpr-sem* ($\varrho(v := x)$) (*integrate-vars* Γ *vs e*)) ∂*stock-measure* (Γ *v*)
by (*simp add*: *integrate-vars-def cexpr-sem-integrate-var extract-real-def*)
also have ... $\geq$ *0*
by (*rule integral-nonneg-AE*, *rule AE-I2*, *subst Cons.IH*[*OF* ϱ']) (*insert Cons.prems*, *auto*)
finally show *extract-real* (*cexpr-sem* ϱ (*integrate-vars* Γ (*v* # *vs*) *e*)) $\geq$ *0* **.**
qed

lemma *nonneg-cexpr-sem-integrate-vars'*:
distinct vs $\Longrightarrow$ *set vs* $\cap$ V' = {} $\Longrightarrow$ *nonneg-cexpr* (*set vs* $\cup$ V') Γ *e* $\Longrightarrow$ $\Gamma \vdash_c e$: *REAL* $\Longrightarrow$
free-vars e $\subseteq$ *set vs* $\cup$ V' $\Longrightarrow$ *nonneg-cexpr* V' Γ (*integrate-vars* Γ *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*: ($\int^+ x$. *extract-real* (*cexpr-sem* (*case-nat x* σ) *e*) ∂*stock-measure t*) < ∞
assumes *nonneg*: *nonneg-cexpr* (*shift-var-set V*) (*case-nat t* Γ) *e*
assumes *t*: *case-nat t* $\Gamma \vdash_c e$: *REAL* **and** *vars*: *free-vars e* $\subseteq$ *shift-var-set V*
assumes ϱ: $\sigma \in$ *space* (*state-measure V* Γ)
shows *ennreal* (*extract-real* (*cexpr-sem* σ ($\int_c$ *e* ∂t))) =
$\int^+ x$. *extract-real* (*cexpr-sem* (*case-nat x* σ) *e*) ∂*stock-measure t*
proof−
let *?f* = λx. *extract-real* (*cexpr-sem* (*case-nat x* σ) *e*)
have *meas*: *?f* $\in$ *borel-measurable* (*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*: *integrable* (*stock-measure t*) *?f*
by (*auto intro*!: *integrableI-nonneg nonneg-cexprD case-nat-in-state-measure*[*OF - ϱ*])

have *extract-real* (*cexpr-sem* σ ($\int_c$ *e* ∂t)) =
$\int$ *x. extract-real* (*cexpr-sem* (*case-nat x* σ) *e*) ∂*stock-measure t*
by (*simp add*: *extract-real-def*)
also have *ennreal* ... = $\int^+ x$. *extract-real* (*cexpr-sem* (*case-nat x* σ) *e*) ∂*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*)
($\lambda\varrho$ *x*. $\int^+$*y*. *eval-cexpr f* (*case-nat x ϱ*) *y ∂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* : *REAL*
assumes *varsf*: *free-vars f* ⊆ *shift-var-set* (*shift-var-set V′*)
assumes *ϱ*: *ϱ* ∈ *space* (*state-measure V′ Γ*)
shows *AE x in stock-measure t*.
($\int^+$*y*. *eval-cexpr f* (*case-nat x ϱ*) *y ∂stock-measure t′*) = *ennreal* (*eval-cexpr* ($\int_c$ *f ∂t′*) *ϱ x*)
proof (*rule AE-mp*[*OF - AE-I2*[*OF impI*]])
interpret *sigma-finite-measure stock-measure t′* **by** *simp*
let *?f* = *λx*. $\int^+$*y*. *eval-cexpr f* (*case-nat x ϱ*) *y ∂stock-measure t′*
from *has-parametrized-subprob-density-integral*[*OF dens ϱ*]
have ($\int^+$*x*. *?f x ∂stock-measure t*) ≠ ∞ **by** (*auto simp*: *eval-cexpr-def top-unique*)
thus *AE x in stock-measure t*. *?f x* ≠ ∞ **using** *ϱ tf varsf* **by** (*intro nn-integral-PInf-AE*) *simp-all*
fix *x* **assume** *x*: *x* ∈ *space* (*stock-measure t*) **and** *finite*: *?f x* ≠ ∞
have *nonneg′*: *AE y in stock-measure t′*. *eval-cexpr f* (*case-nat x ϱ*) *y* ≥ *0*
unfolding *eval-cexpr-def* **using** *ϱ x*
by (*intro AE-I2 nonneg-cexprD*[*OF nonneg*]) (*auto intro*!: *case-nat-in-state-measure*)
hence *integrable* (*stock-measure t′*) (*λy*. *eval-cexpr f* (*case-nat x ϱ*) *y*)
using *x ϱ tf varsf finite* **by** (*intro integrableI-nonneg*) (*simp-all add*: *top-unique less-top*)
thus *?f x* = *ennreal* (*eval-cexpr* ($\int_c$ *f ∂t′*) *ϱ 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* × *vname list* × *tyenv* × *cexpr*

definition *dens-ctxt-α* :: *cdens-ctxt* ⇒ *dens-ctxt* **where**
dens-ctxt-α ≡ λ(*vs*,*vs′*,Γ,δ). (*set vs*, *set vs′*, Γ, *λσ*. *extract-real* (*cexpr-sem σ δ*))

definition *shift-vars* :: *nat list* ⇒ *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* ⇒ *pdf-type* ⇒ *cexpr* ⇒ *bool* **where**
is-density-expr ≡ λ(*vs*,*vs'*,Γ,δ) *t e*.
case-nat t Γ $\vdash_c$ *e* : *REAL* ∧
free-vars e ⊆ *shift-var-set* (*set vs'*) ∧
nonneg-cexpr (*shift-var-set* (*set vs'*)) (*case-nat t* Γ) *e*

lemma *is-density-exprI*:
case-nat t Γ $\vdash_c$ *e* : *REAL* ⟹
free-vars e ⊆ *shift-var-set* (*set vs'*) ⟹
nonneg-cexpr (*shift-var-set* (*set vs'*)) (*case-nat t* Γ) *e* ⟹
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* Γ $\vdash_c$ *e* : *REAL free-vars e* ⊆ *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 (λσ. *extract-real* (*cexpr-sem* σ δ))
proof (*unfold density-context-def*, *intro allI ballI conjI impI subprob-spaceI*)
show (λ*x*. *ennreal* (*extract-real* (*cexpr-sem x* δ)))
∈ *borel-measurable* (*state-measure* (*set vs* ∪ *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: ϱ ∈ *space* (*state-measure* (*set vs'*) Γ)
let *?M* = *dens-ctxt-measure* (*set vs*, *set vs'*, Γ, λ*x*. *ennreal* (*extract-real* (*cexpr-sem x* δ))) ϱ
from ϱ **have** (λσ. *merge* (*set vs*) (*set vs'*) (σ, ϱ))
∈ *measurable* (*state-measure* (*set vs*) Γ) (*state-measure* (*set vs* ∪ *set vs'*) Γ)
unfolding *state-measure-def* **by** *simp*
hence *emeasure ?M* (*space ?M*) =
$\int^+$*x*. *ennreal* (*extract-real* (*cexpr-sem* (*merge* (*set vs*) (*set vs'*) (*x*, ϱ))

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* ≤ *1* **unfolding** *subprob-cexpr-def* **by** *blast*
finally show *emeasure ?M* (*space ?M*) ≤ *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* ⇒ *vname list* ⇒ *vname* ⇒ *cexpr* ⇒ *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* ≠ *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*) ⊆ *shift-var-set* (*set vs′*)
proof−
have *free-vars* (*marg-dens-cexpr* Γ *vs x δ*) ⊆ *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** ... ⊆ *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* ⟹ *case-nat* (Γ *x*) $\Gamma \vdash_c$ *marg-dens-cexpr* Γ *vs x δ* : *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* ∈ *set vs* **and** *ϱ*: *ϱ* ∈ *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* ≠ *x*) *vs*
note *cdens-ctxt-invar-imp-integrable*[*OF assms*(*1*) *ϱ*]
moreover from *x* **have** *insert-eq*: *insert x* {*xa* ∈ *set vs. xa* ≠ *x*} = *set vs* **by** *auto*

ultimately have *integrable*:
AE v in stock-measure (Γ *x*).
integrable (*state-measure* (*set ?vs*) Γ)
(λσ. *extract-real* (*cexpr-sem* (*merge* (*set ?vs*) (*insert x* (*set vs'*)) (σ, ϱ(*x* := *v*))) δ))
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* ∈ *space* (*stock-measure* (Γ *x*))
assume *integrable*:
integrable (*state-measure* (*set ?vs*) Γ)
(λσ. *extract-real* (*cexpr-sem* (*merge* (*set ?vs*) (*insert x* (*set vs'*)) (σ, ϱ(*x* := *v*))) δ))

from *v* **and** ϱ **have** ϱ': (ϱ(*x* := *v*)) ∈ *space* (*state-measure* (*set* (*x*#*vs'*)) Γ)
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* ϱ ∘ (λ*y*. *if y* = *x then 0 else Suc y*))
(*integrate-vars* Γ [*y*←*vs* . *y* ≠ *x*] δ) (**is** - = *?A*)
unfolding *marg-dens-cexpr-def* **by** (*simp add*: *cexpr-sem-map-vars*)
also have ⋀*y*. *y* ∈ *free-vars* (*integrate-vars* Γ [*y*←*vs* . *y* ≠ *x*] δ)
⟹ (*case-nat v* ϱ ∘ (λ*y*. *if y* = *x then 0 else Suc y*)) *y* = (ϱ(*x* := *v*)) *y*
unfolding *o-def* **by** *simp*
hence *?A* = *cexpr-sem* (ϱ(*x* := *v*)) (*integrate-vars* Γ [*y*←*vs* . *y* ≠ *x*] δ) **by** (*rule cexpr-sem-eq-on-vars*)
also from *x* **have** *insert x* {*xa* ∈ *set vs*. *xa* ≠ *x*} ∪ *set vs'* = *set vs* ∪ *set vs'* **by** *auto*
hence *extract-real* (*cexpr-sem* (ϱ(*x* := *v*)) (*integrate-vars* Γ [*y*←*vs* . *y* ≠ *x*] δ)) =
∫ ⁺σ. *extract-real* (*cexpr-sem* (*merge* (*set ?vs*) (*insert x* (*set vs'*)) (σ, ϱ(*x* := *v*))) δ)
∂*state-measure* (*set ?vs*) Γ
using ϱ' *invar integrable* **by** (*subst cexpr-sem-integrate-vars'*) (*auto*)
also from *x* **have** (λσ. *merge* (*set ?vs*) (*insert x* (*set vs'*)) (σ, ϱ(*x* := *v*))) =
(λσ. *merge* (*set vs*) (*set vs'*) (σ(*x* := *v*), ϱ))
by (*intro ext*) (*auto simp*: *merge-def*)
also from *x* **have** *set ?vs* = *set vs* − {*x*} **by** *auto*
also have (∫ ⁺σ. *extract-real* (*cexpr-sem* (*merge* (*set vs*) (*set vs'*) (σ(*x* := *v*), ϱ)) δ)
∂*state-measure* (*set vs* − {*x*}) Γ) =
marg-dens (*dens-ctxt-α* (*vs*,*vs'*,Γ,δ)) *x* ϱ *v*
unfolding *marg-dens-def dens-ctxt-α-def* **by** *simp*
finally show *ennreal* (*extract-real* (*cexpr-sem* (λ*a*. *case a of 0* ⇒ *v* | *Suc a* ⇒ ϱ *a*)
(*marg-dens-cexpr* Γ *vs x* δ))) =
marg-dens (*dens-ctxt-α* (*vs*, *vs'*, Γ, δ)) *x* ϱ *v* **.**
qed

qed

lemma *nonneg-cexpr-sem-marg-dens*:
  assumes *cdens-ctxt-invar vs vs′* Γ *δ*
  assumes *x*: *x* ∈ *set vs* **and** *ϱ*: *ϱ* ∈ *space* (*state-measure* (*set vs′*) Γ)
  assumes *v*: *v* ∈ *type-universe* (Γ *x*)
  shows *extract-real* (*cexpr-sem* (*case-nat v ϱ*) (*marg-dens-cexpr* Γ *vs x δ*)) ≥ *0*
proof−
  note *invar* = *cdens-ctxt-invarD*[*OF assms*(*1*)]
  from *assms* **have** *ϱ*: *case-nat v ϱ* ∘ (λ*y*. *if y* = *x then 0 else Suc y*)
      ∈ *space* (*state-measure* (*set* (*x*#*vs′*)) Γ)
    by (*force simp*: *state-measure-def space-PiM o-def split*: *if-split-asm*)
  moreover from *x* **have** *insert x* {*xa* ∈ *set vs*. *xa* ≠ *x*} ∪ *set vs′* = *set vs* ∪ *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* ⇒ *vname list* ⇒ *vname* ⇒ *vname* ⇒ *cexpr* ⇒ *cexpr* **where**
  marg-dens2-cexpr Γ *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* (λ*z*. *z* ≠ *x* ∧ *z* ≠ *y*) *vs*) *e*))))

lemma *free-vars-marg-dens2-cexpr*:
  assumes *cdens-ctxt-invar vs vs′* Γ *δ*
  shows *free-vars* (*marg-dens2-cexpr* Γ *vs x y δ*) ⊆ *shift-var-set* (*set vs′*)
proof−
  have *free-vars* (*marg-dens2-cexpr* Γ *vs x y δ*) ⊆
      shift-var-set (*free-vars δ* − *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** ... ⊆ *shift-var-set* (*set vs′*)
    unfolding *shift-var-set-def* **by** *auto*
  finally show *?thesis* **.**
qed

lemma *cexpr-typing-marg-dens2-cexpr*[*intro*]:
  assumes Γ $\vdash_c$ *δ* : *REAL*
  shows *case-nat* (*PRODUCT* (Γ *x*) (Γ *y*)) Γ $\vdash_c$ *marg-dens2-cexpr* Γ *vs x y δ* : *REAL*
proof−

have *A*: (*case-nat* (*PRODUCT* (Γ *x*) (Γ *y*)) Γ) (*Suc x* := Γ *x*, *Suc y* := Γ *y*) ∘ *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* ∈ *set vs* **and** *y*: *y* ∈ *set vs* **and** *x* ≠ *y*
assumes *ϱ*: *ϱ* ∈ *space* (*state-measure* (*set vs′*) Γ)
shows *AE z in stock-measure* (*PRODUCT* (Γ *x*) (Γ *y*)).
ennreal (*extract-real* (*cexpr-sem* (*case-nat z ϱ*) (*marg-dens2-cexpr* Γ *vs x y δ*))) =
marg-dens2 (*dens-ctxt-α* (*vs*,*vs′*,Γ,*δ*)) *x y ϱ z*
proof−
note *invar* = *cdens-ctxt-invarD*[*OF assms*(*1*)]
let *?f* = *λx*. *ennreal* (*extract-real* (*cexpr-sem x δ*))
let *?vs* = *filter* (*λz*. *z* ≠ *x* ∧ *z* ≠ *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*: (*λσ*. *extract-real* (*cexpr-sem* (*merge* (*set vs*) (*set vs′*) (*σ*, *ϱ*)) *δ*))
∈ *borel-measurable* (*state-measure* (*set vs*) Γ) **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*: (*λ*(*σ*,*v*). *σ*(*y* := *v*)) ∈
measurable (*PiM* (*insert x* (*set vs*)) (*λx*. *stock-measure* (Γ *x*)) $\bigotimes_M$ *stock-measure* (Γ *y*))
(*PiM* (*insert y* (*insert x* (*set vs*))) (*λx*. *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* := *fst* (*fst xa*), *y* := *snd* (*fst xa*)))
∈ *measurable* ((*stock-measure* (Γ *x*) $\bigotimes_M$ *stock-measure* (Γ *y*)) $\bigotimes_M$
Pi_M (*set ?vs*) (*λy*. *stock-measure* (Γ *y*)))
(Pi_M (*set vs*) (*λy*. *stock-measure* (Γ *y*)))
by (*subst insert-eq′*[*symmetric*], *intro measurable-Pair-compose-split*[*OF measurable-add-dim*])
(*simp-all del*: *fun-upd-apply*)

from *x y* **have** *A*: *set vs* = {*x*, *y*} ∪ *set ?vs* **by** *auto*

have $(\int^{+}\sigma.$ *?f* (*merge* (*set vs*) (*set vs'*) $(\sigma, \varrho))$ ∂*state-measure* (*set vs*) $\Gamma) =$
$(\int^{+}\sigma'. \int^{+}\sigma.$ *?f* (*merge* (*set vs*) (*set vs'*) (*merge* $\{x, y\}$ (*set ?vs*) $(\sigma', \sigma), \varrho))$
∂*state-measure* (*set ?vs*) Γ ∂*state-measure* $\{x,y\}$ Γ) (**is** - = *?I*)
using *meas insert-eq* **unfolding** *state-measure-def*
by (*subst A*, *subst product-nn-integral-fold*) (*simp-all add*: *measurable-compose*[*OF* - *measurable-ennreal*])
also have $\bigwedge\sigma'\ \sigma.$ *merge* (*set vs*) (*set vs'*) (*merge* $\{x, y\}$ (*set ?vs*) $(\sigma', \sigma), \varrho) =$
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* (*merge* (*set vs*) (*set vs'*) $(\sigma(x := \sigma'\ x, y := \sigma'\ y), \varrho))$
∂*state-measure* (*set ?vs*) Γ ∂*state-measure* $\{x,y\}$ Γ) **by** *simp*
also have ... = $\int^{+}z. \int^{+}\sigma.$ *?f* (*merge* (*set vs*) (*set vs'*) $(\sigma(x :=$ *fst z*, $y :=$ *snd z*$), \varrho))$
∂*state-measure* (*set ?vs*) Γ ∂(*stock-measure* $(\Gamma\ x)$ $\bigotimes_M$ *stock-measure* $(\Gamma\ y))$
(**is** - = *?I*) **using** ‹$x \neq y$› *meas-upd2* ϱ *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* - *measurable-ennreal*]
measurable-compose[*OF* - *measurable-cexpr-sem'*] *measurable-Pair*)
finally have $(\int^{+}\sigma.$ *?f* (*merge* (*set vs*) (*set vs'*) $(\sigma, \varrho))$ ∂*state-measure* (*set vs*) $\Gamma) =$ *?I* **.**
moreover have $(\int^{+}\sigma.$ *?f* (*merge* (*set vs*) (*set vs'*) $(\sigma, \varrho))$ ∂*state-measure* (*set vs*) $\Gamma) \neq \infty$
using *cdens-ctxt-invar-imp-integrable*[*OF assms*(*1*) ϱ] **unfolding** *real-integrable-def* **by** *simp*
ultimately have *?I* $\neq \infty$ **by** *simp*
hence *AE z in stock-measure* $(\Gamma\ x)$ $\bigotimes_M$ *stock-measure* $(\Gamma\ y)$.
$(\int^{+}\sigma.$ *?f* (*merge* (*set vs*) (*set vs'*) $(\sigma(x :=$ *fst z*, $y :=$ *snd z*$), \varrho))$
∂*state-measure* (*set ?vs*) $\Gamma) \neq \infty$ (**is** *AE z in* -. *?P z*)
using *meas-upd2* ϱ *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* - *measurable-cexpr-sem'*]
measurable-Pair simp: *measurable-split-conv state-measure-def*)
hence *AE z in stock-measure* $(\Gamma\ x)$ $\bigotimes_M$ *stock-measure* $(\Gamma\ y)$.
ennreal (*extract-real* (*cexpr-sem* (*case-nat* (*case-prod PairVal z*) ϱ) (*marg-dens2-cexpr* Γ *vs x y* δ))) =
marg-dens2 (*dens-ctxt-*α (*vs*,*vs'*,Γ,δ)) *x y* ϱ (*case-prod PairVal z*)
proof (*rule AE-mp*[*OF* - *AE-I2*[*OF impI*]])
fix *z* **assume** *z*: $z \in$ *space* (*stock-measure* $(\Gamma\ x)$ $\bigotimes_M$ *stock-measure* $(\Gamma\ y))$
assume *fin*: *?P z*
have $\bigwedge\sigma.$ *merge* (*set vs*) (*set vs'*) $(\sigma(x :=$ *fst z*, $y :=$ *snd z*$), \varrho) =$
merge (*set ?vs*) ($\{x,y\} \cup$ *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* (*merge* (*set vs*) (*set vs'*) (σ(x := *fst z*, y := *snd z*), ϱ))
∂*state-measure* (*set ?vs*) Γ) =
($\int^{+}\sigma$. *?f* (*merge* (*set ?vs*) ({x,y} $\cup$ *set vs'*) (σ, ϱ(x := *fst z*, y := *snd z*)))
∂*state-measure* (*set ?vs*) Γ) (**is** - = $\int^{+}\sigma$. *ennreal* (*?g* σ) ∂*?M*)
by (*intro nn-integral-cong*) *simp*
have ϱ': ϱ(x := *fst z*, y := *snd z*) $\in$ *space* (*state-measure* ({x, y} $\cup$ *set vs'*) Γ)
using *z* ϱ **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* $\in$ *borel-measurable ?M* **using** *invar* ϱ' **by** (*intro measurable-cexpr-sem'*) *auto*
show ($\int^{+}\sigma$. *ennreal* (*?g* σ) ∂*?M*) < ∞ **using** *fin A* **by** (*simp add*: *top-unique less-top*)
fix σ **assume** σ: $\sigma \in$ *space ?M*
from *x y* **have** *set ?vs* $\cup$ ({x,y} $\cup$ *set vs'*) = *set vs* $\cup$ *set vs'* **by** *auto*
thus *?g* $\sigma \geq 0$ **using** *merge-in-state-measure*[*OF* σ ϱ']
by (*intro nonneg-cexprD*[*OF invar*(*4*)]) *simp-all*
qed
from *x y* **have** *B*: (*set ?vs* $\cup$ ({x, y} $\cup$ *set vs'*)) = *set vs* $\cup$ *set vs'* **by** *auto*
have *nonneg*: *nonneg-cexpr* (*set* [*z*←*vs* . $z \neq x \wedge z \neq y$] $\cup$ ({x, y} $\cup$ *set vs'*)) Γ δ
using *invar* **by** (*subst B*) *simp*

have *ennreal* (*extract-real* (*cexpr-sem* (*case-nat* (*case-prod PairVal z*) ϱ) (*marg-dens2-cexpr* Γ *vs x y* δ))) =
extract-real (*cexpr-sem* ((*case-nat* <|*fst z*, *snd z*|> ϱ) (*Suc x* := *fst z*, *Suc y* := *snd z*) $\circ$ *Suc*)
(*integrate-vars* Γ *?vs* δ))
unfolding *marg-dens2-cexpr-def*
by (*simp add*: *cexpr-sem-cexpr-comp-aux cexpr-sem-map-vars split*: *prod.split*)
also have ((*case-nat* <|*fst z*, *snd z*|> ϱ) (*Suc x* := *fst z*, *Suc y* := *snd z*)) $\circ$ *Suc* =
ϱ(x := *fst z*, y := *snd z*) (**is** *?*ϱ*1* = *?*ϱ*2*) **by** (*intro ext*) (*simp split*: *nat.split*)
also have *ennreal* (*extract-real* (*cexpr-sem* (ϱ(x := *fst z*, y := *snd z*))
(*integrate-vars* Γ [*z*←*vs* . $z \neq x \wedge z \neq y$] δ))) =
$\int^{+}$*xa*. *?f* (*merge* (*set ?vs*) ({x, y} $\cup$ *set vs'*) (*xa*, ϱ(x := *fst z*, y := *snd z*))) ∂*?M*
using *invar assms* **by** (*intro cexpr-sem-integrate-vars'*[*OF* ϱ' - - *nonneg integrable*]) *auto*
also have *C*: *set ?vs* = *set vs* $-$ {x, y} **by** *auto*
have ($\int^{+}$*xa*. *?f* (*merge* (*set ?vs*) ({x, y} $\cup$ *set vs'*) (*xa*, ϱ(x := *fst z*, y := *snd z*))) ∂*?M*) =
marg-dens2 (*dens-ctxt-*α (*vs*,*vs'*,Γ,δ)) *x y* ϱ (*case-prod PairVal z*)
unfolding *marg-dens2-def*
by (*subst A*[*symmetric*], *subst C*, *simp only*: *dens-ctxt-*α*-def prod.case*)
(*auto intro*!: *nn-integral-cong split*: *prod.split*)

finally show *ennreal* (*extract-real* (*cexpr-sem* (*case-nat* (*case-prod PairVal z*) ϱ)
(*marg-dens2-cexpr* Γ *vs x y* δ))) =
marg-dens2 (*dens-ctxt-α* (*vs*, *vs'*, Γ, δ)) *x y* ϱ (*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* ∈ *set vs* **and** *y*: *y* ∈ *set vs* **and** ϱ: ϱ ∈ *space* (*state-measure* (*set vs'*) Γ)
assumes *v*: *v* ∈ *type-universe* (*PRODUCT* (Γ *x*) (Γ *y*))
shows *extract-real* (*cexpr-sem* (*case-nat v* ϱ) (*marg-dens2-cexpr* Γ *vs x y* δ)) ≥ *0*
proof−
from *v* **obtain** *a b* **where** *v'*: *v* = <|*a*, *b*|> *a* ∈ *type-universe* (Γ *x*) *b* ∈ *type-universe* (Γ *y*)
by (*auto simp*: *val-type-eq-PRODUCT*)
let *?vs* = *filter* (λ*z*. *z* ≠ *x* ∧ *z* ≠ *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*))) ∘ *Suc* =
ϱ(*x* := *fst* (*extract-pair v*), *y* := *snd* (*extract-pair v*)) **by** (*intro ext*) *auto*
have *B*: ϱ(*x* := *fst* (*extract-pair v*), *y* := *snd* (*extract-pair v*))
∈ *space* (*state-measure* (*set vs'* ∪ {*x*,*y*}) Γ) **using** *x y v'* ϱ
by (*auto simp*: *space-state-measure split*: *if-split-asm*)
from *x y* **have** *set ?vs* ∪ (*set vs'* ∪ {*x*, *y*}) = *set vs* ∪ *set vs'* **by** *auto*
with *invar* **have** *nonneg-cexpr* (*set ?vs* ∪ (*set vs'* ∪ {*x*, *y*})) Γ δ **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*!: *nonneg-cexpr-sem-integrate-vars*[*OF B*])
qed

definition *branch-prob-cexpr* :: *cdens-ctxt* ⇒ *cexpr* **where**
branch-prob-cexpr ≡ λ(*vs*, *vs'*, Γ, δ). *integrate-vars* Γ *vs* δ

lemma *free-vars-branch-prob-cexpr*[*simp*]:
free-vars (*branch-prob-cexpr* (*vs*, *vs'*, Γ, δ)) = *free-vars* δ − *set vs*
unfolding *branch-prob-cexpr-def* **by** *simp*

lemma *cexpr-typing-branch-prob-cexpr*[*intro*]:
$\Gamma \vdash_c \delta : REAL \Longrightarrow \Gamma \vdash_c$ *branch-prob-cexpr* (*vs*,*vs'*,Γ,δ) : *REAL*
unfolding *branch-prob-cexpr-def*
by (*simp only*: *prod.case*, *rule cexpr-typing-integrate-vars*)

lemma *cexpr-sem-branch-prob*:
assumes *cdens-ctxt-invar vs vs'* Γ δ
assumes ϱ: $\varrho \in$ *space* (*state-measure* (*set vs'*) Γ)
shows *ennreal* (*extract-real* (*cexpr-sem* ϱ (*branch-prob-cexpr* (*vs*, *vs'*, Γ, δ)))) =
branch-prob (*dens-ctxt-*α (*vs*, *vs'*, Γ, δ)) ϱ
proof−
note *invar* = *cdens-ctxt-invarD*[*OF assms*(*1*)]
interpret *density-context set vs set vs'* Γ $\lambda\sigma$. *extract-real* (*cexpr-sem* σ δ)
by (*rule density-context-*α) *fact*
have *ennreal* (*extract-real* (*cexpr-sem* ϱ (*branch-prob-cexpr* (*vs*, *vs'*, Γ, δ)))) =
$\int^{+}\sigma$. *extract-real* (*cexpr-sem* (*merge* (*set vs*) (*set vs'*) (σ, ϱ)) δ)
∂*state-measure* (*set vs*) Γ (**is** - = *?I*)
using *assms*(*2*) *invar* **unfolding** *branch-prob-cexpr-def*
by (*simp only*: *prod.case*, *subst cexpr-sem-integrate-vars'*)
(*auto intro*!: *cdens-ctxt-invar-imp-integrable assms*)
also have ... = *branch-prob* (*dens-ctxt-*α (*vs*, *vs'*, Γ, δ)) ϱ
using ϱ **unfolding** *dens-ctxt-*α*-def* **by** (*simp only*: *prod.case branch-prob-altdef*[*of* ϱ])
finally show *?thesis* **.**
qed

lemma *subprob-imp-subprob-cexpr*:
assumes *density-context V V'* Γ ($\lambda\sigma$. *extract-real* (*cexpr-sem* σ δ))
shows *subprob-cexpr V V'* Γ δ
proof (*intro subprob-cexprI*)
interpret *density-context V V'* Γ $\lambda\sigma$. *extract-real* (*cexpr-sem* σ δ) **by** *fact*
fix ϱ **assume** ϱ: $\varrho \in$ *space* (*state-measure V'* Γ)
let *?M* = *dens-ctxt-measure* (*V*, *V'*, Γ, $\lambda\sigma$. *extract-real* (*cexpr-sem* σ δ)) ϱ
from ϱ **have** ($\int^{+}$ *x*. *ennreal* (*extract-real* (*cexpr-sem* (*merge V V'* (*x*, ϱ)) δ)) ∂*state-measure V* Γ) =
branch-prob (*V*, *V'*, Γ, $\lambda\sigma$. *extract-real* (*cexpr-sem* σ δ)) ϱ (**is** *?I* = -)
by (*subst branch-prob-altdef*[*symmetric*]) *simp-all*
also have ... = *emeasure ?M* (*space ?M*) **unfolding** *branch-prob-def* **by** *simp*
also have ... $\leq$ *1* **by** (*rule subprob-space.emeasure-space-le-1*) (*simp add*: *subprob-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-cexpr* :: *cdens-ctxt* $\Rightarrow$ *expr* $\Rightarrow$ *cexpr* $\Rightarrow$ *bool*

((1-/ $\vdash_c$/ (- $\Rightarrow$/ -)) [50,0,50] 50) **where**

edc-val: countable-type (val-type v) $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ Val v $\Rightarrow$
map-vars Suc (branch-prob-cexpr (vs, vs$'$, Γ, δ)) $*_c$ $\langle$CVar 0 $=_c$
CVal v$\rangle_c$
| edc-var: x $\in$ set vs $\Longrightarrow$ (vs, vs$'$, Γ, δ) $\vdash_c$ Var x $\Rightarrow$ marg-dens-cexpr Γ vs x δ
| edc-pair: x $\in$ set vs $\Longrightarrow$ y $\in$ set vs $\Longrightarrow$ x $\neq$ y $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ <Var x, Var y> $\Rightarrow$ marg-dens2-cexpr Γ vs x y δ
| edc-fail: (vs, vs$'$, Γ, δ) $\vdash_c$ Fail t $\Rightarrow$ CReal 0
| edc-let: ([], vs @ vs$'$, Γ, CReal 1) $\vdash_c$ e $\Rightarrow$ f $\Longrightarrow$
(shift-vars vs, map Suc vs$'$, the (expr-type Γ e) $\cdot$ Γ,
map-vars Suc δ $*_c$ f) $\vdash_c$ e$'$ $\Rightarrow$ g $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ LET e IN e$'$ $\Rightarrow$ map-vars (λx. x $-$ 1) g
| edc-rand: (vs, vs$'$, Γ, δ) $\vdash_c$ e $\Rightarrow$ f $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ Random dst e $\Rightarrow$
$\int_c$ map-vars (case-nat 0 (λx. x + 2)) f $*_c$
dist-dens-cexpr dst (CVar 0) (CVar 1) ∂dist-param-type dst
| edc-rand-det: randomfree e $\Longrightarrow$ free-vars e $\subseteq$ set vs$'$ $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ Random dst e $\Rightarrow$
map-vars Suc (branch-prob-cexpr (vs, vs$'$, Γ, δ)) $*_c$
dist-dens-cexpr dst (map-vars Suc (expr-rf-to-cexpr e)) (CVar 0)
| edc-if-det: randomfree b $\Longrightarrow$
(vs, vs$'$, Γ, δ $*_c$ $\langle$expr-rf-to-cexpr b$\rangle_c$) $\vdash_c$ e1 $\Rightarrow$ f1 $\Longrightarrow$
(vs, vs$'$, Γ, δ $*_c$ $\langle\neg_c$ expr-rf-to-cexpr b$\rangle_c$) $\vdash_c$ e2 $\Rightarrow$ f2 $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ IF b THEN e1 ELSE e2 $\Rightarrow$ f1 $+_c$ f2
| edc-if: ([], vs @ vs$'$, Γ, CReal 1) $\vdash_c$ b $\Rightarrow$ f $\Longrightarrow$
(vs, vs$'$, Γ, δ $*_c$ cexpr-subst-val f TRUE) $\vdash_c$ e1 $\Rightarrow$ g1 $\Longrightarrow$
(vs, vs$'$, Γ, δ $*_c$ cexpr-subst-val f FALSE) $\vdash_c$ e2 $\Rightarrow$ g2 $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ IF b THEN e1 ELSE e2 $\Rightarrow$ g1 $+_c$ g2
| edc-op-discr: (vs, vs$'$, Γ, δ) $\vdash_c$ e $\Rightarrow$ f $\Longrightarrow$ Γ $\vdash$ e : t $\Longrightarrow$
op-type oper t = Some t$'$ $\Longrightarrow$ countable-type t$'$ $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ oper $$ e $\Rightarrow$
$\int_c$ $\langle$(oper $$$_c$ (CVar 0)) $=_c$ CVar 1$\rangle_c$ $*_c$ map-vars (case-nat
0 (λx. x+2)) f ∂t
| edc-fst: (vs, vs$'$, Γ, δ) $\vdash_c$ e $\Rightarrow$ f $\Longrightarrow$ Γ $\vdash$ e : PRODUCT t t$'$ $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ Fst $$ e $\Rightarrow$
$\int_c$ (map-vars (case-nat 0 (λx. x + 2)) f $\circ_c$ <CVar 1, CVar
0>$_c$) ∂t$'$
| edc-snd: (vs, vs$'$, Γ, δ) $\vdash_c$ e $\Rightarrow$ f $\Longrightarrow$ Γ $\vdash$ e : PRODUCT t t$'$ $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ Snd $$ e $\Rightarrow$
$\int_c$ (map-vars (case-nat 0 (λx. x + 2)) f $\circ_c$ <CVar 0, CVar
1>$_c$) ∂t
| edc-neg: (vs, vs$'$, Γ, δ) $\vdash_c$ e $\Rightarrow$ f $\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ Minus $$ e $\Rightarrow$ f $\circ_c$ (λ_cx. $-_c$ x)
| edc-addc: (vs, vs$'$, Γ, δ) $\vdash_c$ e $\Rightarrow$ f $\Longrightarrow$ randomfree e$'$ $\Longrightarrow$ free-vars e$'$ $\subseteq$ set vs$'$
$\Longrightarrow$
(vs, vs$'$, Γ, δ) $\vdash_c$ Add $$ <e, e$'$> $\Rightarrow$
f $\circ_c$ (λ_cx. x $-_c$ map-vars Suc (expr-rf-to-cexpr e$'$))

| *edc-multc*: $(vs,\ vs',\ \Gamma,\ \delta) \vdash_c e \Rightarrow f \Longrightarrow c \neq 0 \Longrightarrow$
$(vs,\ vs',\ \Gamma,\ \delta) \vdash_c$ *Mult* \$\$ <*e*, *Val* (*RealVal c*)> $\Rightarrow$
$(f \circ_c (\lambda_c x.\ x *_c$ *CReal* (*inverse c*))) $*_c$ *CReal* (*inverse* (*abs c*))
| *edc-add*: $(vs,\ vs',\ \Gamma,\ \delta) \vdash_c e \Rightarrow f \Longrightarrow \Gamma \vdash e :$ *PRODUCT t t* $\Longrightarrow$
$(vs,\ vs',\ \Gamma,\ \delta) \vdash_c$ *Add* \$\$ $e \Rightarrow$
$\int_c$ (*map-vars* (*case-nat 0* ($\lambda x.\ x+2$)) $f \circ_c$ ($\lambda_c x.$ <*x*, *CVar 1* $-_c$ *x*>$_c$)) ∂t
| *edc-inv*: $(vs,\ vs',\ \Gamma,\ \delta) \vdash_c e \Rightarrow f \Longrightarrow$
$(vs,\ vs',\ \Gamma,\ \delta) \vdash_c$ *Inverse* \$\$ $e \Rightarrow$
$(f \circ_c (\lambda_c x.\ inverse_c\ x)) *_c (\lambda_c x.\ (inverse_c\ x)\ \widehat{}_c$ *CInt 2*)
| *edc-exp*: $(vs,\ vs',\ \Gamma,\ \delta) \vdash_c e \Rightarrow f \Longrightarrow$
$(vs,\ vs',\ \Gamma,\ \delta) \vdash_c$ *Exp* \$\$ $e \Rightarrow$
$(\lambda_c x.\ IF_c$ *CReal 0* $<_c x$ *THEN* $(f \circ_c ln_c\ x) *_c inverse_c\ x$ *ELSE* *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* ⊆ *set vs* ∪ *set vs'*
assumes *hd*: *dens-ctxt-α* ([], *vs* @ *vs'*, Γ, *CReal 1*) $\vdash_d e \Rightarrow$ (λ*x xa*. *ennreal* (*eval-cexpr f x xa*))
notes *invar* = *cdens-ctxt-invarD*[*OF inv*]
assumes *wf1*: *is-density-expr* ([], *vs* @ *vs'*, Γ, *CReal 1*) *t' f*
shows *cdens-ctxt-invar* (*shift-vars vs*) (*map Suc vs'*) ($t' \cdot \Gamma$) (*map-vars Suc* $\delta *_c f$)
proof (*intro cdens-ctxt-invarI*)
show *t'*: *case-nat t'* Γ $\vdash_c$ *map-vars Suc* $\delta *_c f$: *REAL* **using** *invar 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-exprD*)

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'*) (λσ *x*. *ennreal* (*eval-cexpr f* σ *x*))
(λ*x*. *ennreal* (*extract-real* (*cexpr-sem x* δ)))
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* ∪ *set vs'*) Γ (λ-. *1*)
by (*rule density-context-empty*)
hence *dens*: *has-parametrized-subprob-density* (*state-measure* (*set vs* ∪ *set vs'*) Γ)
(λϱ. *dens-ctxt-measure* ({}, *set vs* ∪ *set vs'*, Γ, λ-. *1*) ϱ ≫= (λσ. *expr-sem* σ *e*))
(*stock-measure t'*) (λσ *x*. *ennreal* (*eval-cexpr f* σ *x*))
using *hd free-vars* **by** (*intro expr-has-density-sound-aux*[*OF - t dc*])

(*auto simp*: *shift-var-set-def dens-ctxt-α-def simp*: *extract-real-def one-ennreal-def*)
from *density-context.density-context-insert*[*OF density-context-α*[*OF inv*] *this*]
have *density-context ?vs ?vs′ ?Γ ?δ* **.**
have *dc*: *density-context* (*shift-var-set* (*set vs*)) (*Suc ‘ set vs′*) (*case-nat t′ Γ*)
(*λσ. extract-real* (*cexpr-sem σ* (*map-vars Suc δ* $*_c$ *f*)))
proof (*rule density-context-equiv*)
show *density-context* (*shift-var-set* (*set vs*)) (*Suc ‘ set vs′*) (*case-nat t′ Γ*) *?δ*
by *fact*
show (*λx. ennreal* (*extract-real* (*cexpr-sem x* (*map-vars Suc δ* $*_c$ *f*))))
∈ *borel-measurable* (*state-measure* (*?vs* ∪ *?vs′*) *?Γ*)
apply (*rule measurable-compose*[*OF - measurable-ennreal*], *rule measurable-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 *σ* **assume** *σ*: *σ* ∈ *space* (*state-measure* (*?vs* ∪ *?vs′*) *?Γ*)
have [*simp*]: *case-nat* (*σ 0*) (*λx. σ* (*Suc x*)) = *σ* **by** (*intro ext*) (*simp split*: *nat.split*)
from *σ* **show** *insert-dens* (*set vs*) (*set vs′*) (*λσ x. ennreal* (*eval-cexpr f σ x*))
(*λx. ennreal* (*extract-real* (*cexpr-sem x δ*))) *σ* =
ennreal (*extract-real* (*cexpr-sem σ* (*map-vars Suc δ* $*_c$ *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 ?Γ - - - ?vs* ∪ *?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′ Γ*)
(*map-vars Suc δ* $*_c$ *f*) **by** *simp*

have *Suc* −‘ *shift-var-set* (*set vs* ∪ *set vs′*) = *set vs* ∪ *set vs′*
by (*auto simp*: *shift-var-set-def*)
moreover have *nonneg-cexpr* (*shift-var-set* (*set vs* ∪ *set vs′*)) (*case-nat t′ Γ*) *f*
using *wf1*[*THEN is-density-exprD-nonneg*] **by** *simp*
ultimately show *nonneg-cexpr* (*set* (*shift-vars vs*) ∪ *set* (*map Suc vs′*)) (*case-nat t′ Γ*) (*map-vars Suc δ* $*_c$ *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* ([], *vs* @ *vs'*, Γ, *CReal 1*) $\vdash_d$ *b* ⇒ (λϱ *x*. *ennreal* (*eval-cexpr f* ϱ *x*))
assumes *wf*: *is-density-expr* ([], *vs* @ *vs'*, Γ, *CReal 1*) *BOOL f*
assumes *t*: Γ ⊢ *b* : *BOOL* **and** *vars*: *free-vars b* ⊆ *set vs* ∪ *set vs'*
assumes *invar*: *cdens-ctxt-invar vs vs'* Γ δ
shows *cdens-ctxt-invar vs vs'* Γ (δ $*_c$ *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-exprD*[*OF wf*]
show Γ $\vdash_c$ δ $*_c$ *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* = λϱ. *dens-ctxt-measure* ({}, *set vs* ∪ *set vs'*, Γ, λ-. *1*) ϱ ⤜ (λσ. *expr-sem* σ *b*)
have *dens'*: *has-parametrized-subprob-density* (*state-measure* (*set vs* ∪ *set vs'*) Γ) *?M*
(*stock-measure BOOL*) (λσ *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* ∪ *set vs'*)) (*case-nat BOOL* Γ) *f*
using *wf*[*THEN is-density-exprD-nonneg*] **by** *simp*

show *subprob-cexpr* (*set vs*) (*set vs'*) Γ (δ $*_c$ *cexpr-subst-val f* (*BoolVal v*))
proof (*intro subprob-cexprI*)
fix ϱ **assume** ϱ: ϱ ∈ *space* (*state-measure* (*set vs'*) Γ)
let *?eval* = λ*e* σ. *extract-real* (*cexpr-sem* (*merge* (*set vs*) (*set vs'*) (σ, ϱ)) *e*)
{
fix σ **assume** σ : σ ∈ *space* (*state-measure* (*set vs*) Γ)
have *A*: *?eval* (δ $*_c$ *cexpr-subst-val f* (*BoolVal v*)) σ =
?eval δ σ ∗ *?eval* (*cexpr-subst-val f* (*BoolVal v*)) σ **using** *wf' invar'* σ ϱ
by (*subst cexpr-sem-Mult*[**where** Γ = Γ **and** *V* = *set vs* ∪ *set vs'*])
(*auto intro*: *merge-in-state-measure simp*: *shift-var-set-def*)
have *?eval* δ σ ≥ *0* **using** σ ϱ *invar'*
by (*blast dest*: *nonneg-cexprD intro*: *merge-in-state-measure*)
moreover have *?eval* (*cexpr-subst-val f* (*BoolVal v*)) σ ≥ *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*))
(**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* (δ $*_c$ *cexpr-subst-val f* (*BoolVal v*)) σ ≤ *?eval* δ σ

by (*subst A*, *intro mult-right-le-one-le*) *simp-all*
}
hence ($\int^+\sigma$. *?eval* ($\delta *_c$ *cexpr-subst-val f* (*BoolVal v*)) σ ∂*state-measure* (*set vs*) Γ) $\leq$
($\int^+\sigma$. *?eval* δ σ ∂*state-measure* (*set vs*) Γ) **by** (*intro nn-integral-mono*) (*simp add*: *ennreal-leI*)
also have ... $\leq$ *1* **using** *invar'* ϱ **by** (*intro subprob-cexprD*)
finally show ($\int^+\sigma$. *?eval* ($\delta *_c$ *cexpr-subst-val f* (*BoolVal v*)) σ ∂*state-measure* (*set vs*) Γ) $\leq$ *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-measure* (*shift-var-set V*) (*case-nat t* Γ)) $\Longrightarrow$
$\exists x\ \sigma'$. $x \in$ *type-universe t* $\wedge$ $\sigma' \in$ *space* (*state-measure V* Γ) $\wedge$ σ = *case-nat x* σ'
by (*intro exI*[*of* - σ *0*] *exI*[*of* - $\sigma \circ$ *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-measure* (*shift-var-set V*) (*case-nat t* Γ)) $\longleftrightarrow$
($\exists x\ \sigma'$. $x \in$ *type-universe t* $\wedge$ $\sigma' \in$ *space* (*state-measure V* Γ) $\wedge$ σ = *case-nat x* σ')
by (*auto dest*!: *space-state-measureD-shift*)

lemma *nonneg-cexprI-shift*:
assumes $\bigwedge x\ \sigma$. $x \in$ *type-universe t* $\Longrightarrow$ $\sigma \in$ *space* (*state-measure V* Γ) $\Longrightarrow$
0 $\leq$ *extract-real* (*cexpr-sem* (*case-nat x* σ) *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-cexpr (*shift-var-set V*) (*case-nat t* Γ) (*map-vars Suc e*) $\longleftrightarrow$ *nonneg-cexpr V* Γ *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*. $x \in$ *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-nat x n* (*case-nat y m i*) = *case-nat* (*case-nat x n y*) ($\lambda i'$. *case-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* Γ)) (*map-vars* (*case-nat 0* (λ*x. Suc* (*Suc x*))) *e*) ⟷
nonneg-cexpr (*shift-var-set V*) (*case-nat t1* Γ) *e*
apply (*auto simp*: *cexpr-sem-map-vars o-def nonneg-cexpr-def space-state-measure-shift-iff*)
subgoal for *x* σ
apply (*drule bspec*[*of - - case-nat x* (*case-nat* (*SOME x. x* ∈ *type-universe t2*) σ)])
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* σ
by (*auto simp add*: *space-state-measure-shift-iff fun-eq-iff split*: *nat.split*
intro!: *exI*[*of - x1*] *exI*[*of -* σ])
done

lemma *nonneg-cexpr-Add*:
assumes $\Gamma \vdash_c$ *e1* : *REAL* $\Gamma \vdash_c$ *e2* : *REAL*
assumes *free-vars e1* ⊆ *V free-vars e2* ⊆ *V*
assumes *N1*: *nonneg-cexpr V* Γ *e1* **and** *N2*: *nonneg-cexpr V* Γ *e2*
shows *nonneg-cexpr V* Γ (*e1* $+_c$ *e2*)
proof (*rule nonneg-cexprI*)
fix σ **assume** σ: σ ∈ *space* (*state-measure V* Γ)
hence *extract-real* (*cexpr-sem* σ (*e1* $+_c$ *e2*)) = *extract-real* (*cexpr-sem* σ *e1*) + *extract-real* (*cexpr-sem* σ *e2*)
using *assms* **by** (*subst cexpr-sem-Add*[*of* Γ *- - - V*]) *simp-all*
also have ... ≥ *0* **using** σ *N1 N2* **by** (*intro add-nonneg-nonneg nonneg-cexprD*)
finally show *extract-real* (*cexpr-sem* σ (*e1* $+_c$ *e2*)) ≥ *0* **.**
qed

lemma *expr-has-density-cexpr-sound-aux*:
assumes Γ ⊢ *e* : *t* (*vs*, *vs'*, Γ, δ) $\vdash_c$ *e* ⇒ *f cdens-ctxt-invar vs vs'* Γ δ
free-vars e ⊆ *set vs* ∪ *set vs'*
shows *dens-ctxt-α* (*vs*,*vs'*,Γ,δ) $\vdash_d$ *e* ⇒ *eval-cexpr f* ∧ *is-density-expr* (*vs*,*vs'*,Γ,δ) *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'* Γ δ)
from *edc-val.prems* **have** [*simp*]: *t* = *val-type v* **by** *auto*
note *invar* = *cdens-ctxt-invarD*[*OF edc-val.prems*(*2*)]
let *?e1* = *map-vars Suc* (*branch-prob-cexpr* (*vs*, *vs'*, Γ, δ)) **and** *?e2* = ⟨*CVar 0* $=_c$ *CVal v*⟩$_c$
have *ctype1*: *case-nat t* Γ $\vdash_c$ *?e1* : *REAL* **and** *ctype2*: *case-nat t* Γ $\vdash_c$ *?e2*: *REAL* **using** *invar*
by (*auto intro*!: *cexpr-typing.intros cexpr-typing-map-vars simp*: *o-def*)
hence *ctype*: *case-nat t* Γ $\vdash_c$ *?e1* $*_c$ *?e2* : *REAL* **by** (*auto intro*!: *cexpr-typing.intros*)

{
fix ϱ x **assume** x: $x \in$ *type-universe* (*val-type* v)
and ϱ: $\varrho \in$ *space* (*state-measure* (*set vs'*) Γ)
hence *case-nat* x $\varrho \in$ *space* (*state-measure* (*shift-var-set* (*set vs'*)) (*case-nat* (*val-type* v) Γ))
by (*rule case-nat-in-state-measure*)
hence *ennreal* (*eval-cexpr* (*?e1* $*_c$ *?e2*) ϱ x) =
ennreal (*extract-real* (*cexpr-sem* (*case-nat* x ϱ)
(*map-vars Suc* (*branch-prob-cexpr* (*vs*, *vs'*, Γ, δ))))) $*$
ennreal (*extract-real* (*RealVal* (*bool-to-real* (x = v)))) (**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* Γ - - - *shift-var-set* (*set vs'*)])
apply (*insert invar ctype1 ctype2*)
apply (*auto simp*: *shift-var-set-def*)
done
also have *?a* = *branch-prob* (*dens-ctxt-*α (*vs*,*vs'*,Γ,δ)) ϱ
by (*subst cexpr-sem-map-vars*, *subst cexpr-sem-branch-prob*) (*simp-all add*: *o-def* ϱ *edc-val.prems*)
also have *?b* = *indicator* $\{v\}$ x
by (*simp add*: *extract-real-def bool-to-real-def split*: *split-indicator*)
finally have *ennreal* (*eval-cexpr* (*?e1* $*_c$ *?e2*) ϱ x) =
branch-prob (*dens-ctxt-*α (*vs*,*vs'*,Γ,δ)) ϱ $*$ *indicator* $\{v\}$ x **.**
} **note** e = *this*

have *meas*: ($\lambda(\sigma, x)$. *ennreal* (*eval-cexpr* (*?e1* $*_c$ *?e2*) σ x))
$\in$ *borel-measurable* (*state-measure* (*set vs'*) Γ $\bigotimes_M$ *stock-measure* (*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 $*$: *Suc* −‘ *shift-var-set* (*set vs'*) = *set vs'* *case-nat* (*val-type* v) $\Gamma \circ$ *Suc* = Γ
by (*auto simp*: *shift-var-set-def*)

have *nn*: *nonneg-cexpr* (*shift-var-set* (*set vs'*)) (*case-nat t* Γ)
(*map-vars Suc* (*branch-prob-cexpr* (*vs*, *vs'*, Γ, δ)) $*_c$ $\langle$*CVar 0* $=_c$ *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-*α*-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-α-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′ Γ δ t*)
hence *t*: *t* = *Γ x* **by** *auto*
note *invar* = *cdens-ctxt-invarD*[*OF edc-var.prems*(*2*)]
from *invar* **have** *ctype*: *case-nat t Γ* $\vdash_c$ *marg-dens-cexpr Γ vs x δ* : *REAL* **by** (*auto simp*: *t*)

show *?case* **unfolding** *dens-ctxt-α-def*
proof (*simp only*: *prod.case*, *intro conjI is-density-exprI*, *rule hd-AE*[*OF hd-var edc-var.prems*(*1*)])
show *case-nat t Γ* $\vdash_c$ *marg-dens-cexpr Γ vs x δ* : *REAL* **by** *fact*
next
show *free-vars* (*marg-dens-cexpr Γ vs x δ*) ⊆ *shift-var-set* (*set vs′*)
using *edc-var.prems*(*2*) **by** (*rule free-vars-marg-dens-cexpr*)
next
have *free-vars*: *free-vars* (*marg-dens-cexpr Γ vs x δ*) ⊆ *shift-var-set* (*set vs′*)
using *edc-var.prems*(*2*) **by** (*rule free-vars-marg-dens-cexpr*)
show (*λ*(*ϱ*, *y*). *ennreal* (*eval-cexpr* (*marg-dens-cexpr Γ vs x δ*) *ϱ y*))
∈ *borel-measurable* (*state-measure* (*set vs′*) *Γ* $\bigotimes_M$ *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 *ϱ* **assume** *ϱ* ∈ *space* (*state-measure* (*set vs′*) *Γ*)
hence *AE y in stock-measure t*.
marg-dens (*dens-ctxt-α* (*vs*, *vs′*, *Γ*, *δ*)) *x ϱ y* =
ennreal (*eval-cexpr* (*marg-dens-cexpr Γ vs x δ*) *ϱ 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′*, *Γ*, *λx*. *ennreal* (*extract-real* (*cexpr-sem x δ*))) *x ϱ y* =
ennreal (*eval-cexpr* (*marg-dens-cexpr Γ vs x δ*) *ϱ y*)
by (*simp add*: *dens-ctxt-α-def*)
next
show *x* ∈ *set vs*
by (*insert edc-var.prems edc-var.hyps*, *auto simp*: *eval-cexpr-def intro*!: *nonneg-cexpr-sem-marg-dens*)
show *nonneg-cexpr* (*shift-var-set* (*set vs′*)) (*case-nat t Γ*) (*marg-dens-cexpr Γ vs x δ*)

by (*intro nonneg-cexprI-shift nonneg-cexpr-sem-marg-dens*[*OF edc-var.prems*(*2*) ‹$x \in$ *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.prems*(*2*)]
from *invar* **have** *ctype*: *case-nat t* Γ $\vdash_c$ *marg-dens2-cexpr* Γ *vs x y* δ : *REAL* **by** *auto*
from *edc-pair.prems* **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.prems*(*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, λ*x*. *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.prems*(*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.prems*(*2*) ‹$x \in$ *set vs*› ‹$y \in$*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 $\vdash_c$ *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.prems*(*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*) $\vdash_d$ *e* ⇒
(λ*x 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.prems* **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* $\delta *_c f$)
using *edc-let.prems edc-let.hyps dens1 wf1 invar*
by (*intro cdens-ctxt-invar-insert*[*OF - t1*]) (*auto simp*: *dens-ctxt-*α*-def*)

let *?$\mathcal{Y}$* = (*shift-vars vs*, *map Suc vs'*, *case-nat t'* Γ, *map-vars Suc* $\delta *_c f$)
have *set* (*shift-vars vs*) $\cup$ *set* (*map Suc vs'*) = *shift-var-set* (*set vs* $\cup$ *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* $\delta *_c f$) $\vdash_d$
e' $\Rightarrow$ (λ*x xa*. *ennreal* (*eval-cexpr g x xa*)) $\wedge$
is-density-expr (*shift-vars vs*, *map Suc vs'*, *case-nat t'* Γ, *map-vars Suc* $\delta *_c f$) *t g*
using *invf t2 edc-let.prems subset-shift-var-set*
by (*simp only*: *t1'*[*symmetric*], *intro edc-let.IH*(*2*)) *simp-all*
hence *dens2*: *dens-ctxt-*α *?$\mathcal{Y}$* $\vdash_d$ *e'* $\Rightarrow$ (λ*x 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* $\delta *_c f$) *t g*
by *simp-all*

have *cexpr-eq*: *cexpr-sem* (*case-nat x* $\varrho \circ$ (λ*x*. *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*]: $\bigwedge\sigma$. *case-nat* (σ *0*) (λ*x*. 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'*) (λ*x xa*. *ennreal* (*eval-cexpr f x xa*))
(λ*x*. *ennreal* (*extract-real* (*cexpr-sem x* δ))))
$\vdash_d$ *e'* $\Rightarrow$ (λ*a 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) $\vdash_d$ *LET e IN e'* $\Rightarrow$

(λϱ x. *ennreal* (*eval-cexpr g* (*case-nat undefined* ϱ) *x*))
unfolding *dens-ctxt-α-def*
by (*simp only*: *prod.case*, *intro hd-let*[**where** *f* = λ*x xa*. *ennreal* (*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'* ⇒
(λϱ x. *ennreal* (*eval-cexpr* (*map-vars* (λx. x − 1) *g*) ϱ *x*))
proof (*simp only*: *dens-ctxt-α-def prod.case*, *erule-tac hd-cong*[*OF* - - *edc-let.prems*(1,3)])
fix ϱ *x* **assume** ϱ: ϱ ∈ *space* (*state-measure* (*set vs'*) Γ)
and *x*: *x* ∈ *space* (*stock-measure t*)
have *eval-cexpr* (*map-vars* (λx. x − 1) *g*) ϱ *x* =
extract-real (*cexpr-sem* (*case-nat x* ϱ ∘ (λx. x − *Suc 0*)) *g*)
unfolding *eval-cexpr-def* **by** (*simp add*: *cexpr-sem-map-vars*)
also note *cexpr-eq*[*of x* ϱ]
finally show *ennreal* (*eval-cexpr g* (*case-nat undefined* ϱ) *x*) =
ennreal (*eval-cexpr* (*map-vars* (λx. x − 1) *g*) ϱ *x*)
by (*simp add*: *eval-cexpr-def*)
qed (*simp-all add*: *density-context-α*[*OF edc-let.prems*(2)])
moreover have *is-density-expr* (*vs*, *vs'*, Γ, δ) *t* (*map-vars* (λx. x − 1) *g*)
proof (*intro is-density-exprI*)
note *wf* = *is-density-exprD*[*OF wf2*]
show *case-nat t* Γ $\vdash_c$ *map-vars* (λx. 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* (λx. x − 1) *g*)
⊆ *shift-var-set* (*set vs'*)
by (*auto simp*: *shift-var-set-def*)
next
show *nonneg-cexpr* (*shift-var-set* (*set vs'*)) (*case-nat t* Γ) (*map-vars* (λx. 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'* Γ δ *e f dst t'*)
define *t* **where** *t* = *dist-param-type dst*
note *invar* = *cdens-ctxt-invarD*[*OF edc-rand.prems*(2)]
from *edc-rand* **have** *t1*: Γ ⊢ *e* : *t* **and** *t2*: *t'* = *dist-result-type dst* **by** (*auto simp*: *t-def*)

have *dens*: *dens-ctxt-α* (*vs*, *vs'*, Γ, δ) $\vdash_d$ *e* ⇒ (λ*x xa*. *ennreal* (*eval-cexpr f x xa*))
and
wf: *is-density-expr* (*vs*, *vs'*, Γ, δ) *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* = ($\lambda\varrho$. *dens-ctxt-measure* (*dens-ctxt-α* (*vs*,*vs'*,Γ,δ)) $\varrho \gg=$ ($\lambda\sigma$. *expr-sem* σ *e*))
have *dens'*: *has-parametrized-subprob-density* (*state-measure* (*set vs'*) Γ) *?M* (*stock-measure t*)
($\lambda\varrho$ *x*. *ennreal* (*eval-cexpr f* ϱ *x*)) **using** *dens t1 edc-rand.prems*
by (*simp-all add*: *dens-ctxt-α-def expr-has-density-sound-aux density-context-α*)

let *?shift* = *case-nat 0* (λx. *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* ∂t)
have [*simp*]: $\bigwedge t\ t'\ \Gamma$. *case-nat t* (*case-nat t'* Γ) $\circ$ *?shift* = *case-nat t* Γ
by (*intro ext*) (*simp split*: *nat.split add*: *o-def*)
have *te1*: *case-nat t* (*case-nat t'* Γ) $\vdash_c$ *?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'* Γ) $\vdash_c$ *?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* (λx. *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'* Γ)) *?e1*
by (*auto intro*!: *nonneg-cexprI wf*[*THEN is-density-exprD-nonneg*, *THEN nonneg-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'* Γ)) *?e2*
by (*intro nonneg-dist-dens-cexpr cet-var'*) (*auto simp*: *t2 t-def shift-var-set-def*)

let *?f* = $\lambda\varrho$ *x*. $\int^+ y$. *ennreal* (*eval-cexpr f* ϱ *y*) $*$ *dist-dens dst y x*∂*stock-measure t*
let *?M* = ($\lambda\varrho$. *dens-ctxt-measure* (*dens-ctxt-α* (*vs*,*vs'*,Γ,δ)) $\varrho \gg=$ ($\lambda\sigma$. *expr-sem* σ (*Random dst e*)))
have *dens'*: *dens-ctxt-α* (*vs*, *vs'*, Γ, δ) $\vdash_d$ *Random dst e* $\Rightarrow$ *?f* **using** *dens*
by (*simp only*: *dens-ctxt-α-def prod.case t-def hd-rand*[*unfolded apply-dist-to-dens-def*])
hence *dens''*: *has-parametrized-subprob-density* (*state-measure* (*set vs'*) Γ) *?M* (*stock-measure t'*) *?f*
using *edc-rand.prems invar*
by (*simp only*: *dens-ctxt-α-def prod.case*, *intro expr-has-density-sound-aux*)

(*auto intro*!: *density-context-α*)

{
fix ϱ **assume** ϱ: $\varrho \in$ *space* (*state-measure* (*set vs'*) Γ)
fix *x* **assume** *x*: $x \in$ *type-universe t'*
fix *y* **assume** *y*: $y \in$ *type-universe t*
let *?ϱ''* = *case-nat y* (*case-nat x ϱ*) **and** *?Γ''* = *case-nat t* (*case-nat t' Γ*)
let *?V''* = *shift-var-set* (*shift-var-set* (*set vs'*))
have *ϱ''*: *?ϱ''* ∈ *space* (*state-measure* (*shift-var-set* (*shift-var-set* (*set vs'*))) *?Γ''*)
using *ϱ x y* **by** (*intro case-nat-in-state-measure*) *simp-all*
have *A*: *extract-real* (*cexpr-sem ?ϱ''* (*?e1* $*_c$ *?e2*)) =
extract-real (*cexpr-sem ?ϱ'' ?e1*) ∗ *extract-real* (*cexpr-sem ?ϱ'' ?e2*)
by (*rule cexpr-sem-Mult*[*OF te1 te2 ϱ'' vars-e1 vars-e2*])
also have ... ≥ *0* **using** *nonneg-e1 nonneg-e2 ϱ''*
by (*blast intro*: *mult-nonneg-nonneg dest*: *nonneg-cexprD*)
finally have *B*: *extract-real* (*cexpr-sem ?ϱ''* (*?e1* $*_c$ *?e2*)) ≥ *0* **.**
note *A*
hence *eval-cexpr f ϱ y* ∗ *dist-dens dst y x* = *extract-real* (*cexpr-sem ?ϱ''* (*?e1* $*_c$ *?e2*))
using *ϱ''*
apply (*subst A*)
apply (*subst ennreal-mult''*)
using *nonneg-e2*
apply (*erule nonneg-cexprD*)
apply (*subst cexpr-sem-dist-dens-cexpr*[*of ?Γ''* - - - *?V''*])
apply (*force simp*: *cexpr-sem-map-vars eval-cexpr-def t2 t-def intro*!: *cet-var'*)+
done
note *this B*
} **note** *e1e2* = *this*

{
fix ϱ **assume** ϱ: $\varrho \in$ *space* (*state-measure* (*set vs'*) Γ)
have *AE x in stock-measure t'*.
apply-dist-to-dens dst (*λϱ x. ennreal* (*eval-cexpr f ϱ x*)) *ϱ x* = *eval-cexpr ?e ϱ x*
proof (*rule AE-mp*[*OF - AE-I2*[*OF impI*]])
from *has-parametrized-subprob-density-integral*[*OF dens'' ϱ*]
have ($\int^+$*x. ?f ϱ x ∂stock-measure t'*) ≠ ∞ **by** *auto*
thus *AE x in stock-measure t'. ?f ϱ x* ≠ ∞
using *has-parametrized-subprob-densityD*(*3*)[*OF dens''*] *ϱ*
by (*intro nn-integral-PInf-AE*) *simp-all*
next
fix *x* **assume** *x*: *x* ∈ *space* (*stock-measure t'*) **and** *finite*: *?f ϱ x* ≠ ∞
let *?ϱ'* = *case-nat x ϱ*
have *ϱ'*: *?ϱ'* ∈ *space* (*state-measure* (*shift-var-set* (*set vs'*)) (*case-nat t' Γ*))
using *ϱ x* **by** (*intro case-nat-in-state-measure*) *simp-all*
hence ∗: ($\int^+$*y. ennreal* (*eval-cexpr f ϱ y*) ∗ *dist-dens dst y x ∂stock-measure t*) =
$\int^+$*y. extract-real* (*cexpr-sem* (*case-nat y ?ϱ'*) (*?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* (λ*ϱ 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.prems*(*1*)])
from *dens* **show** (*set vs*, *set vs′*, Γ, λ*x*. *ennreal* (*extract-real* (*cexpr-sem x δ*))) ⊢$_d$
e ⇒ (λ*x 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* (λ*x*. *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.prems*(*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'* Γ) *?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'* Γ) *?e1*
using *invar*
by (*auto simp add*: *branch-prob-cexpr-def nonneg-cexpr-shift-iff intro*!: *nonneg-cexpr-sem-integrate-vars'*)

{
fix ϱ *x*
assume *x*: $x \in$ *type-universe t'* **and** ϱ: $\varrho \in$ *space* (*state-measure* (*set vs'*) Γ)
hence ϱ': *case-nat x* $\varrho \in$ *space* (*state-measure* (*shift-var-set* (*set vs'*)) (*case-nat t'* Γ))
by (*rule case-nat-in-state-measure*)
hence *eval-cexpr* (*?e1* $*_c$ *?e2*) ϱ *x* =
ennreal (*extract-real* (*cexpr-sem* (*case-nat x* ϱ)
(*map-vars Suc* (*branch-prob-cexpr* (*vs*, *vs'*, Γ, δ))))) $*$
ennreal (*extract-real* (*cexpr-sem* (*case-nat x* ϱ) *?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'* Γ - - - *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-*α (*vs*,*vs'*,Γ,δ)) ϱ (**is** - = *?c*)
by (*subst cexpr-sem-map-vars*, *subst cexpr-sem-branch-prob*) (*simp-all add*: *o-def* ϱ *edc-rand-det.prems*)
also have *?b* = *dist-dens dst* (*expr-sem-rf* ϱ *e*) *x* (**is** - = *?d*) **using** *t1 edc-rand-det.hyps*
by (*subst cexpr-sem-dist-dens-cexpr*[*of case-nat t'* Γ], *insert* ϱ' *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 $\bigotimes_M$ *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.prems 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.prems*(*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.prems*(*2*)]

let *?ind1* = $\langle$*expr-rf-to-cexpr b*$\rangle_c$ **and** *?ind2* = $\langle\neg_c$ *expr-rf-to-cexpr b*$\rangle_c$
have *tind1*: Γ $\vdash_c$ *?ind1* : *REAL* **and** *tind2*: Γ $\vdash_c$ *?ind2* : *REAL*
using *edc-if-det.hyps tb* **by** (*auto intro*!: *cexpr-typing.intros*)
have *tδ1*: Γ $\vdash_c$ δ $*_c$ *?ind1* : *REAL* **and** *tδ2*: Γ $\vdash_c$ 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.prems free-vars-expr-rf-to-cexpr*[*OF*
edc-if-det.hyps(*1*)]
by (*auto intro*!: *subprob-indicator cet-op*)

have *vars1*: *free-vars* ($\delta *_c$ *?ind1*) $\subseteq$ *set vs* $\cup$ *set vs'* **and**
vars2: *free-vars* ($\delta *_c$ *?ind2*) $\subseteq$ *set vs* $\cup$ *set vs'*
using *invar edc-if-det.hyps edc-if-det.prems free-vars-expr-rf-to-cexpr* **by** *auto*
have *inv1*: *cdens-ctxt-invar vs vs'* Γ ($\delta *_c$ *?ind1*)
using *invar edc-if-det.hyps edc-if-det.prems tind1 tδ1 subprob1 nonneg-ind1 vars1*
by (*intro cdens-ctxt-invarI nonneg-cexpr-Mult*) *auto*
have *inv2*: *cdens-ctxt-invar vs vs'* Γ ($\delta *_c$ *?ind2*)
using *invar edc-if-det.hyps edc-if-det.prems tind2 tδ2 subprob2 nonneg-ind2 vars2*
by (*intro cdens-ctxt-invarI nonneg-cexpr-Mult*) *auto*
have *dens1*: *dens-ctxt-α* (*vs*, *vs'*, Γ, $\delta *_c$ *?ind1*) $\vdash_d$ *e1* $\Rightarrow$ ($\lambda\varrho$ *x*. *eval-cexpr f1* ϱ *x*) **and**
wf1: *is-density-expr* (*vs*, *vs'*, Γ, $\delta *_c$ *?ind1*) *t f1*
using *edc-if-det.IH*(*1*)[*OF t1 inv1*] *edc-if-det.prems* **by** *auto*
have *dens2*: *dens-ctxt-α* (*vs*, *vs'*, Γ, $\delta *_c$ *?ind2*) $\vdash_d$ *e2* $\Rightarrow$ ($\lambda\varrho$ *x*. *eval-cexpr f2* ϱ *x*) **and**
wf2: *is-density-expr* (*vs*, *vs'*, Γ, $\delta *_c$ *?ind2*) *t f2*
using *edc-if-det.IH*(*2*)[*OF t2 inv2*] *edc-if-det.prems* **by** *auto*

show *?case*
proof (*rule conjI*, *simp only*: *dens-ctxt-α-def prod.case*, *rule hd-cong*[*OF hd-if-det*])
let *?$\mathcal{Y}$* = (*set vs*, *set vs'*, Γ, *if-dens-det* (λx. *ennreal* (*extract-real* (*cexpr-sem x* δ))) *b True*)
show *?$\mathcal{Y}$* $\vdash_d$ *e1* $\Rightarrow$ ($\lambda\varrho$ *x*. *eval-cexpr f1* ϱ *x*)
proof (*rule hd-dens-ctxt-cong*)
let *?δ* = $\lambda\sigma$. *ennreal* (*extract-real* (*cexpr-sem* σ ($\delta *_c$ *?ind1*)))
show (*set vs*, *set vs'*, Γ, *?δ*) $\vdash_d$ *e1* $\Rightarrow$ ($\lambda\varrho$ *x*. *ennreal* (*eval-cexpr f1* ϱ *x*))
using *dens1* **by** (*simp add*: *dens-ctxt-α-def*)
fix σ **assume** σ: $\sigma \in$ *space* (*state-measure* (*set vs* $\cup$ *set vs'*) Γ)
have *extract-real* (*cexpr-sem* σ ($\delta *_c$ *?ind1*)) =
extract-real (*cexpr-sem* σ δ) $*$ *extract-real* (*cexpr-sem* σ *?ind1*) **using** *invar vars1*
by (*subst cexpr-sem-Mult*[*OF invar*(*3*) *tind1* σ]) *simp-all*
also have *extract-real* (*cexpr-sem* σ *?ind1*) = (*if expr-sem-rf* σ *b* = *TRUE 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* ($\lambda\sigma$. *ennreal* (*extract-real* (*cexpr-sem* σ δ))) *b True* σ
by (*simp add*: *if-dens-det-def*)
qed
next
let *?$\mathcal{Y}$* = (*set vs*, *set vs'*, Γ, *if-dens-det* (λx. *ennreal* (*extract-real* (*cexpr-sem x* δ))) *b False*)
show *?$\mathcal{Y}$* $\vdash_d$ *e2* $\Rightarrow$ ($\lambda\varrho$ *x*. *eval-cexpr f2* ϱ *x*)
proof (*rule hd-dens-ctxt-cong*)
let *?δ* = $\lambda\sigma$. *ennreal* (*extract-real* (*cexpr-sem* σ ($\delta *_c$ *?ind2*)))

show (*set vs*, *set vs'*, Γ, *?δ*) $\vdash_d$ *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.prems 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.prems*(*2*)]

have *densb*: *dens-ctxt-α* ([], *vs* @ *vs'*, Γ, *CReal 1*) $\vdash_d$ *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.prems* **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.prems* **by** (*auto intro*!: *cdens-ctxt-invar-insert-bool*)
let *?δ1* = *cexpr-subst-val f TRUE* **and** *?δ2* = *cexpr-subst-val f FALSE*
have *tδ1*: $\Gamma \vdash_c \delta *_c$ *?δ1* : *REAL* **and** *tδ2*: $\Gamma \vdash_c \delta *_c$ *?δ2* : *REAL*
using *is-density-exprD*[*OF wfb*] *invar*
by (*auto intro*!: *cet-op*[**where** *t* = *PRODUCT REAL REAL*] *cet-pair*)
have *vars1*: *free-vars* ($\delta *_c$ *?δ1*) ⊆ *set vs* ∪ *set vs′* **and**
vars2: *free-vars* ($\delta *_c$ *?δ2*) ⊆ *set vs* ∪ *set vs′*
using *invar is-density-exprD*[*OF wfb*] **by** (*auto simp*: *shift-var-set-def*)
have *dens1*: *dens-ctxt-α* (*vs*, *vs′*, Γ, $\delta *_c$ *?δ1*) $\vdash_d$ *e1* ⇒ (*λx xa. ennreal* (*eval-cexpr g1 x xa*)) **and**
wf1: *is-density-expr* (*vs*, *vs′*, Γ, $\delta *_c$ *?δ1*) *t g1* **and**
dens2: *dens-ctxt-α* (*vs*, *vs′*, Γ, $\delta *_c$ *?δ2*) $\vdash_d$ *e2* ⇒ (*λx xa. ennreal* (*eval-cexpr g2 x xa*)) **and**
wf2: *is-density-expr* (*vs*, *vs′*, Γ, $\delta *_c$ *?δ2*) *t g2*
using *edc-if.IH*(*2*)[*OF t1 inv1*] *edc-if.IH*(*3*)[*OF t2 inv2*] *edc-if.prems* **by** *simp-all*

have *f-nonneg*[*simp*]: σ ∈ *space* (*state-measure* (*set vs* ∪ *set vs′*) Γ) ⟹
0 ≤ *extract-real* (*cexpr-sem* (*case-nat* (*BoolVal b*) σ) *f*) **for** *b σ*
using *wfb*[*THEN is-density-exprD-nonneg*] **by** (*rule nonneg-cexprD*) *auto*

let *?δ′* = *λσ. ennreal* (*extract-real* (*cexpr-sem σ δ*)) **and** *?f* = *λσ x. ennreal* (*eval-cexpr f σ x*)
show *?case*
proof (*rule conjI*, *simp only*: *dens-ctxt-α-def prod.case*, *rule hd-cong*[*OF hd-if*])
let *?𝒴* = (*set vs*, *set vs′*, Γ, *if-dens ?δ′ ?f True*)
show *?𝒴* $\vdash_d$ *e1* ⇒ (*λϱ x. eval-cexpr g1 ϱ x*)
proof (*rule hd-dens-ctxt-cong*)
let *?δ* = *λσ. ennreal* (*extract-real* (*cexpr-sem σ* ($\delta *_c$ *?δ1*)))
show (*set vs*, *set vs′*, Γ, *?δ*) $\vdash_d$ *e1* ⇒ (*λϱ x. ennreal* (*eval-cexpr g1 ϱ x*))
using *dens1* **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* ($\delta *_c$ *?δ1*)) =
extract-real (*cexpr-sem σ δ*) ∗ *extract-real* (*cexpr-sem σ ?δ1*)
using *invar vars1 is-density-exprD*[*OF wfb*] **by** (*subst cexpr-sem-Mult*[*OF invar*(*3*) - *σ*]) *auto*
also have ... = *if-dens ?δ′ ?f True σ* **unfolding** *if-dens-def* **by** (*simp add*: *eval-cexpr-def ennreal-mult′′ σ*)
finally show *?δ σ* = *if-dens ?δ′ ?f True σ* **by** (*simp add*: *if-dens-det-def*)
qed
next
let *?𝒴* = (*set vs*, *set vs′*, Γ, *if-dens ?δ′ ?f False*)
show *?𝒴* $\vdash_d$ *e2* ⇒ (*λϱ x. eval-cexpr g2 ϱ x*)
proof (*rule hd-dens-ctxt-cong*)
let *?δ* = *λσ. ennreal* (*extract-real* (*cexpr-sem σ* ($\delta *_c$ *?δ2*)))
show (*set vs*, *set vs′*, Γ, *?δ*) $\vdash_d$ *e2* ⇒ (*λϱ x. ennreal* (*eval-cexpr g2 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* ($\delta *_c$ *?δ2*)) =

extract-real (*cexpr-sem* σ δ) $*$ *extract-real* (*cexpr-sem* σ *?δ2*)
using *invar vars1 is-density-exprD*[*OF wfb*] **by** (*subst cexpr-sem-Mult*[*OF invar*(*3*) - σ]) *auto*
also have ... = *if-dens ?δ′ ?f False* σ **unfolding** *if-dens-def* **by** (*simp add*: *eval-cexpr-def ennreal-mult′′* σ)
finally show *?δ* σ = *if-dens ?δ′ ?f False* σ **by** (*simp add*: *if-dens-det-def*)
qed
next
fix ϱ *x* **assume** ϱ: $\varrho \in$ *space* (*state-measure* (*set vs′*) Γ) **and** *x* : *x* ∈ *space* (*stock-measure t*)
hence *eval-cexpr* (*g1* $+_c$ *g2*) ϱ *x* = *eval-cexpr g1* ϱ *x* + *eval-cexpr g2* 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 g1* ϱ *x 0* ≤ *eval-cexpr g2* 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 g1* ϱ *x*) + *ennreal* (*eval-cexpr g2* ϱ *x*) = *ennreal* (*eval-cexpr* (*g1* $+_c$ *g2*) ϱ *x*)
by *simp*
next
show *is-density-expr* (*vs*, *vs′*, Γ, δ) *t* (*g1* $+_c$ *g2*) **using** *wf1 wf2*
by (*auto simp*: *is-density-expr-def intro*!: *cet-op*[**where** *t* = *PRODUCT REAL REAL*] *cet-pair nonneg-cexpr-Add*)
next
show ({}, *set vs* ∪ *set vs′*, Γ, λ-. *1*) $\vdash_d$ *b* ⇒ (λσ *x*. *ennreal* (*eval-cexpr f* σ *x*))
using *densb* **unfolding** *dens-ctxt-α-def* **by** (*simp add*: *extract-real-def one-ennreal-def*)
qed (*insert edc-if.prems edc-if.hyps*, *auto intro*!: *density-context-α*)

next
case (*edc-op-discr vs vs′* Γ δ *e f t oper t′ t′′*)
let *?expr′* = ⟨(*oper* $$$_c$ (*CVar 0*)) $=_c$ *CVar 1*⟩$_c$ $*_c$ *map-vars* (*case-nat 0* (λ*x*. *x*+*2*)) *f*
let *?expr* = $\int_c$ *?expr′* ∂*t* **and** *?shift* = *case-nat 0* (λ*x*. *x* + *2*)
from *edc-op-discr.prems*(*1*) *edc-op-discr.hyps*
have *t*: Γ ⊢ *e* : *t* **by** (*elim expr-typing-opE*, *fastforce split*: *pdf-type.split-asm*)
with *edc-op-discr.prems*(*1*) **and** *edc-op-discr.hyps* **have** [*simp*]: *t′′* = *t′*
by (*intro expr-typing-unique*) (*auto intro*: *et-op*)
from *t* **and** *edc-op-discr.prems*(*1*)
have *the-t1*: *the* (*expr-type* Γ *e*) = *t* **and** *the-t2*: *the* (*expr-type* Γ (*oper* $$ *e*)) = *t′*
by (*simp-all add*: *expr-type-Some-iff*[*symmetric*])

from *edc-op-discr.prems edc-op-discr.IH*[*OF t*]
have *dens*: *dens-ctxt-α* (*vs*, *vs′*, Γ, δ) $\vdash_d$ *e* ⇒ (λ*x xa*. *ennreal* (*eval-cexpr f x xa*)) **and**
wf: *is-density-expr* (*vs*, *vs′*, Γ, δ) *t f* **by** *simp-all*

note *wf′* = *is-density-exprD*[*OF wf*]
have *ctype‴*: *case-nat t* (*case-nat t′* Γ) $\vdash_c$ (*oper* \$\$$_c$ (*CVar 0*)) $=_c$ *CVar 1* : *BOOL* **and**
ctype″: *case-nat t* (*case-nat t′* Γ) $\vdash_c$ ⟨(*oper* \$\$$_c$ (*CVar 0*)) $=_c$ *CVar 1*⟩$_c$: *REAL* **and**
ctype′: *case-nat t* (*case-nat t′* Γ) $\vdash_c$ *?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′* Γ $\vdash_c$ *?expr* : *REAL* **by** (*rule cet-int*)
have *vars′*: *free-vars ?expr′* ⊆ *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* ⊆ *shift-var-set* (*set vs′*) **by** (*auto split*: *nat.split-asm*)

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-cexpr* (*shift-var-set* (*set vs′*)) (*case-nat t* Γ) *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′* Γ)) *?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* = λϱ. *dens-ctxt-measure ?𝒴* ϱ ⤜ (λσ. *expr-sem* σ (*oper* \$\$ *e*))
let *?f* = λϱ *x y*. (*if op-sem oper y* = *x then 1 else 0*) ∗ *ennreal* (*eval-cexpr f* ϱ *y*)
have *?𝒴* $\vdash_d$ *oper* \$\$ *e* ⇒ (λϱ *x*. ∫$^+$*y*. *?f* ϱ *x y* ∂*stock-measure t*) **using** *dens t*
edc-op-discr.hyps
by (*subst the-t1*[*symmetric*], *intro hd-op-discr*)
(*simp-all add*: *dens-ctxt-α-def the-t1 expr-type-Some-iff*[*symmetric*])
hence *dens*: *?𝒴* $\vdash_d$ *oper* \$\$ *e* ⇒ (λϱ *x*. ∫$^+$*y*. *eval-cexpr ?expr′* (*case-nat x* ϱ) *y*
∂*stock-measure t*)
proof (*rule hd-cong*[*OF - - - - nn-integral-cong*])
fix ϱ *x y* **let** *?P* = λ*x M*. *x* ∈ *space M*
assume *A*: *?P* ϱ (*state-measure* (*set vs′*) Γ) *?P x* (*stock-measure t′*) *?P y*
(*stock-measure t*)
hence *val-type* (*cexpr-sem* (*case-nat y* ϱ) *f*) = *REAL* **using** *wf′* **by** (*intro*
val-type-cexpr-sem) *auto*
thus *?f* ϱ *x y* = *ennreal* (*eval-cexpr ?expr′* (*case-nat x* ϱ) *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.prems*, *auto intro*!: *density-context-α*)
hence *dens′*: *has-parametrized-subprob-density* (*state-measure* (*set vs′*) Γ) *?M*
(*stock-measure t′*)
(λϱ *x*. ∫$^+$*y*. *eval-cexpr ?expr′* (*case-nat x* ϱ) *y* ∂*stock-measure t*)
using *edc-op-discr.prems* **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** *ϱ*: $\varrho \in$ *space* (*state-measure* (*set vs′*) Γ)
let *?dens* = *λx*. $\int^+ y$. *eval-cexpr ?expr′* (*case-nat x ϱ*) *y ∂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* $\neq \infty$ **by** (*intro nn-integral-PInf-AE*) *auto*
next
fix *x* **assume** *x*: $x \in$ *space* (*stock-measure t′*) **and** *fin*: *?dens x* $\neq \infty$
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-map-vars simp*: *less-top*)
qed
next
show *nonneg-cexpr* (*shift-var-set* (*set vs′*)) (*case-nat t″* Γ) ($\int_c$ *?expr′ ∂t*)
using *nonneg* **by** (*intro nonneg-cexpr-int*) *simp*
qed (*insert vars ctype edc-op-discr.prems*, *auto*)

next
case (*edc-fst vs vs′* Γ *δ 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* = *case-nat 0* (*λx*. *x* + *2*)
have [*simp*]: ⋀*t t′*. *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*)
note *invar* = *cdens-ctxt-invarD*[*OF edc-fst.prems*(*2*)]
have *dens*: *dens-ctxt-α* (*vs*, *vs′*, Γ, *δ*) $\vdash_d$ *e* ⇒ (*λϱ x*. *ennreal* (*eval-cexpr f ϱ x*)) **and**
wf: *is-density-expr* (*vs*, *vs′*, Γ, *δ*) (*PRODUCT t t′*) *f* **using** *edc-fst* **by** *auto*
let *?M* = *λϱ*. *dens-ctxt-measure* (*set vs*, *set vs′*, Γ, *λϱ*. *ennreal* (*extract-real* (*cexpr-sem ϱ δ*))) *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*)

note *wf′* = *is-density-exprD*[*OF wf*]
let *?expr* = *map-vars ?shift f* $\circ_c$ <*CVar 1*, *CVar 0*>$_c$
have *ctype*: *case-nat t′* (*case-nat t* Γ) $\vdash_c$ *?expr* : *REAL*
using *wf′* **by** (*auto intro*!: *cexpr-typing.intros cexpr-typing-map-vars*)
have *vars*: *free-vars ?expr* ⊆ *shift-var-set* (*shift-var-set* (*set vs′*)) **using** *free-vars-cexpr-comp wf′*
by (*intro subset-shift-var-set*) (*force simp*: *shift-var-set-def*)

let *?M* = $\lambda\varrho$. *dens-ctxt-measure* (*set vs*, *set vs'*, Γ, $\lambda\varrho$. *ennreal* (*extract-real* (*cexpr-sem* ϱ δ))) ϱ
$\gg=$ ($\lambda\sigma$. *expr-sem* σ (*Fst* $$ *e*))
have *A*: $\bigwedge x\ y\ \varrho$. ((*case-nat x* (*case-nat y* ϱ))(*0* := <|*y*, *x*|>)) $\circ$ *?shift* = *case-nat* <|*y*, *x*|> ϱ
by (*intro ext*) (*simp split*: *nat.split add*: *o-def*)
have *dens'*: (*set vs*, *set vs'*, Γ, $\lambda\varrho$. *ennreal* (*extract-real* (*cexpr-sem* ϱ δ))) $\vdash_d$ *Fst* $$ *e* $\Rightarrow$
($\lambda\varrho$ *x*. ($\int^+$*y*. *eval-cexpr f* ϱ (<|*x*, *y*|>) ∂*stock-measure t'*)) (**is** *?Y* $\vdash_d$ - $\Rightarrow$ *?f*)
using *dens* **by** (*subst t'*[*symmetric*], *intro hd-fst*) (*simp add*: *dens-ctxt-α-def*)
hence *dens'*: *?Y* $\vdash_d$ *Fst* $$ *e* $\Rightarrow$ ($\lambda\varrho$ *x*. ($\int^+$*y*. *eval-cexpr ?expr* (*case-nat x* ϱ) *y* ∂*stock-measure t'*))
(**is** - $\vdash_d$ - $\Rightarrow$ *?f*) **by** (*rule hd-cong*, *intro density-context-α*, *insert edc-fst.prems 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.prems* **by** (*intro expr-has-density-sound-aux density-context-α*) *simp-all*

have $\bigwedge V$. *?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: $\varrho \in$ *space* (*state-measure* (*set vs'*) Γ)
thus *AE x in stock-measure t*. *?f* ϱ *x* = *ennreal* (*eval-cexpr* ($\int_c$ *?expr* $\partial t'$) 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)
($\int_c$ (*map-vars* (*case-nat 0* (λx. *x* + *2*)) *f* $\circ_c$ <*CVar 1*, *CVar 0*>$_c$) $\partial t'$)
using *nonneg'* **by** (*intro nonneg-cexpr-int*)
qed (*insert edc-fst.prems 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' 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* = *case-nat 0* ($\lambda x.$ *x* + *2*)
have [*simp*]: $\bigwedge t\ t'$. *case-nat t* (*case-nat t′* Γ) ∘ *case-nat 0* (λ*x*. *Suc* (*Suc x*)) = *case-nat t* Γ
by (*intro ext*) (*simp split*: *nat.split add*: *o-def*)
note *invar* = *cdens-ctxt-invarD*[*OF edc-snd.prems*(*2*)]
have *dens*: *dens-ctxt-α* (*vs*, *vs′*, Γ, δ) $\vdash_d$ *e* ⇒ (λϱ *x*. *ennreal* (*eval-cexpr f* ϱ *x*)) **and**
wf: *is-density-expr* (*vs*, *vs′*, Γ, δ) (*PRODUCT t t′*) *f* **using** *edc-snd* **by** *auto*
let *?M* = λϱ. *dens-ctxt-measure* (*set vs*, *set vs′*, Γ, λϱ. *ennreal* (*extract-real* (*cexpr-sem* ϱ δ))) 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*)

note *wf′* = *is-density-exprD*[*OF wf*]
let *?expr* = *map-vars ?shift f* $\circ_c$ <*CVar 0*, *CVar 1*>$_c$
have *ctype*: *case-nat t* (*case-nat t′* Γ) $\vdash_c$ *?expr* : *REAL*
using *wf′* **by** (*auto intro*!: *cexpr-typing.intros cexpr-typing-map-vars*)
have *vars*: *free-vars ?expr* ⊆ *shift-var-set* (*shift-var-set* (*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 (*Snd* $$ *e*))
have *A*: $\bigwedge x\ y\ \varrho$. ((*case-nat y* (*case-nat x* ϱ))(*0* := <|*y*, *x*|>)) ∘ *?shift* = *case-nat* <|*y*, *x*|> ϱ
by (*intro ext*) (*simp split*: *nat.split add*: *o-def*)
have *dens′*: (*set vs*, *set vs′*, Γ, λϱ. *ennreal* (*extract-real* (*cexpr-sem* ϱ δ))) $\vdash_d$ *Snd* $$ *e* ⇒
(λϱ *y*. ($\int^+$*x*. *eval-cexpr f* ϱ (<|*x*, *y*|>) ∂*stock-measure t*)) (**is** *?Y* $\vdash_d$ - ⇒ *?f*)
using *dens* **by** (*subst t′*[*symmetric*], *intro hd-snd*) (*simp add*: *dens-ctxt-α-def*)
hence *dens′*: *?Y* $\vdash_d$ *Snd* $$ *e* ⇒ (λϱ *y*. ($\int^+$*x*. *eval-cexpr ?expr* (*case-nat y* ϱ) *x* ∂*stock-measure t*))
(**is** - $\vdash_d$ - ⇒ *?f*) **by** (*rule hd-cong*, *intro density-context-α*, *insert edc-snd.prems 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-snd.prems* **by** (*intro expr-has-density-sound-aux density-context-α*) *simp-all*

have $\bigwedge V$. *?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: $\varrho \in$ *space* (*state-measure* (*set vs'*) Γ)
    thus *AE x in stock-measure t'*. *?f* ϱ *x* = *ennreal* (*eval-cexpr* ($\int_c$ *?expr* ∂*t*) ϱ *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'' Γ*) ($\int_c$ *?expr* ∂*t*)
      using *nonneg'* **by** (*intro nonneg-cexpr-int*) *simp*
  qed (*insert edc-snd.prems 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'* Γ δ *e f t*)
  from *edc-neg.prems*(*1*) **have** *t*: Γ ⊢ *e* : *t* **by** (*cases t*) (*auto split*: *pdf-type.split-asm*)
  from *edc-neg.prems*(*1*) **have** *t-disj*: *t* = *REAL* ∨ *t* = *INTEG*
    by (*cases t*) (*auto split*: *pdf-type.split-asm*)
  from *edc-neg.prems edc-neg.IH*[*OF t*]
    have *dens*: *dens-ctxt-α* (*vs*, *vs'*, Γ, δ) $\vdash_d$ *e* ⇒ (λ*x xa*. *ennreal* (*eval-cexpr f x xa*)) **and**
        wf: *is-density-expr* (*vs*, *vs'*, Γ, δ) *t f* **by** *simp-all*

  have *dens-ctxt-α* (*vs*, *vs'*, Γ, δ) $\vdash_d$ *Minus* $$ *e* ⇒ (λσ *x*. *ennreal* (*eval-cexpr f* σ (*op-sem Minus x*)))
    using *dens* **by** (*simp only*: *dens-ctxt-α-def prod.case*, *intro hd-neg*) *simp-all*
  also have (λσ *x*. *ennreal* (*eval-cexpr f* σ (*op-sem Minus x*))) =
        (λσ *x*. *ennreal* (*eval-cexpr* (*f* $\circ_c$ $-_c$ *CVar 0*) σ *x*))
    by (*intro ext*) (*auto simp*: *eval-cexpr-comp*)
  finally have *dens-ctxt-α* (*vs*, *vs'*, Γ, δ) $\vdash_d$ *Minus* $$ *e* ⇒
          (λσ *x*. *ennreal* (*eval-cexpr* (*f* $\circ_c$ $-_c$ *CVar 0*) σ *x*)) **.**
  moreover have *is-density-expr* (*vs*, *vs'*, Γ, δ) *t* (*f* $\circ_c$ $-_c$ *CVar 0*)
  proof (*intro is-density-exprI*)
    from *t-disj* **have** *t-minus*: *case-nat t* Γ $\vdash_c$ $-_c$ *CVar 0* : *t*
      by (*intro cet-op*[**where** *t* = *t*]) (*auto simp*: *cexpr-type-Some-iff*[*symmetric*])
    thus *case-nat t* Γ $\vdash_c$ *f* $\circ_c$ $-_c$ *CVar 0* : *REAL* **using** *is-density-exprD*(*1*)[*OF wf*]
      by (*intro cexpr-typing-cexpr-comp*[*of - - - t*])
    show *free-vars* (*f* $\circ_c$ $-_c$ *CVar 0*) ⊆ *shift-var-set* (*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* Γ) (*f* $\circ_c$ $-_c$ *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* $\circ_c$ (λ_c*x*. *x* $-_c$ *map-vars Suc* (*expr-rf-to-cexpr e'*))
from *edc-addc.prems*(*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.prems*(*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.prems edc-addc.IH*[*OF t1*]
have *dens*: *dens-ctxt-α* (*vs*, *vs'*, Γ, δ) $\vdash_d$ *e* ⇒ (λ*x 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 $\vdash_c$ *?expr* : *REAL* **using** *t1 t2 t3 t3' edc-addc.hyps edc-addc.prems*
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.prems 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.prems*(*1*)
apply (*cases*)
apply (*erule expr-typing.cases*)
apply (*auto split*: *pdf-type.splits*)
done

have *dens-ctxt-α* (*vs*, *vs'*, Γ, δ) $\vdash_d$ *Add* $$ <*e*, *e'*> ⇒
(λσ *x*. *ennreal* (*eval-cexpr f* σ (*op-sem Add* <|*x*, *expr-sem-rf* σ (*Minus* $$ *e'*)|>)))
(**is** *?𝒴* $\vdash_d$ - ⇒ *?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) $\vdash_d$ *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.prems 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'* Γ δ *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.prems*(*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.prems edc-multc.IH*[*OF t1*]
have *dens*: *dens-ctxt-*α (*vs*, *vs'*, Γ, δ) $\vdash_d$ *e* $\Rightarrow$ (λ*x xa*. *ennreal* (*eval-cexpr f x xa*)) **and**
wf: *is-density-expr* (*vs*, *vs'*, Γ, δ) *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.prems is-density-exprD*[*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.prems edc-multc.hyps 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*
hence *vars*: *free-vars ?expr* $\subseteq$ *shift-var-set* (*set vs'*) **by** *simp*

have *dens-ctxt-*α (*vs*, *vs'*, Γ, δ) $\vdash_d$ *Mult* \$\$ <*e*, *Val* (*RealVal c*)> $\Rightarrow$
($\lambda\sigma$ *x*. *ennreal* (*eval-cexpr f* σ (*op-sem Mult* <|*x*, *op-sem Inverse* (*RealVal c*)|>)) $*$
ennreal (*inverse* (*abs* (*extract-real* (*RealVal c*)))))
(**is** *?Y* $\vdash_d$ - $\Rightarrow$ *?f*) **using** *dens edc-multc.hyps*
by (*simp only*: *dens-ctxt-*α*-def prod.case*, *intro hd-multc*) *simp-all*
hence *dens-ctxt-*α (*vs*, *vs'*, Γ, δ) $\vdash_d$ *Mult* \$\$ <*e*, *Val* (*RealVal c*)> $\Rightarrow$
($\lambda\sigma$ *x*. *ennreal* (*eval-cexpr ?expr* σ *x*))
proof (*simp only*: *dens-ctxt-*α*-def prod.case*, *erule-tac hd-cong*)
fix ϱ *x* **assume** ϱ: $\varrho \in$ *space* (*state-measure* (*set vs'*) Γ) **and** *x*: *x* $\in$ *space* (*stock-measure REAL*)
hence *eval-cexpr ?expr* ϱ *x* =
extract-real (*cexpr-sem* (*case-nat x* ϱ) (*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* ϱ (*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* ϱ (*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.prems*, *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*]] *nonneg-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.prems*(*2*)]
  from *edc-add.prems* **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) $\vdash_d$ *e* ⇒ (λ*x 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* (λ*x*. *x* + *2*)
  let *?expr'* = *map-vars ?shift f* $\circ_c$ (λ_c*x*. <*x*, *CVar 1* $-_c$ *x*>$_c$)
  let *?expr* = $\int_c$ *?expr'* ∂*t*
  have [*simp*]: ⋀*t t'* Γ. *case-nat t* (*case-nat t'* Γ) ∘ *case-nat 0* (λ*x*. *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) $\vdash_c$ <*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) $\vdash_c$ *?expr'* : *REAL* **using** *wf'*
    by (*intro cexpr-typing-cexpr-comp cexpr-typing-map-vars*) *simp-all*
  hence *ctype*: *case-nat t* Γ $\vdash_c$ *?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* = $\lambda\varrho$. *dens-ctxt-measure ?Y* ϱ $\gg=$ ($\lambda\sigma$. *expr-sem* σ (*Add* $$ *e*))
let *?f* = $\lambda\varrho$ *x y*. *eval-cexpr f* ϱ <|*y*, *op-sem Add* <|*x*, *op-sem Minus y*|>|>
have *?Y* $\vdash_d$ *Add* $$ *e* $\Rightarrow$ ($\lambda\varrho$ *x*. $\int^+$*y*. *?f* ϱ *x y* ∂*stock-measure* (*val-type x*)) **using** *dens*
by (*intro hd-add*) (*simp add*: *dens-ctxt-*α*-def*)
hence *dens*: *?Y* $\vdash_d$ *Add* $$ *e* $\Rightarrow$ ($\lambda\varrho$ *x*. $\int^+$*y*. *eval-cexpr ?expr'* (*case-nat x* ϱ) *y* ∂*stock-measure t*)
by (*rule hd-cong*) (*insert edc-add.prems*, *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*. *eval-cexpr ?expr'* (*case-nat x* ϱ) *y* ∂*stock-measure t*)
using *edc-add.prems* **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** ϱ: $\varrho \in$ *space* (*state-measure* (*set vs'*) Γ)
let *?dens* = λ*x*. $\int^+$*y*. *eval-cexpr ?expr'* (*case-nat x* ϱ) *y* ∂*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* $\neq \infty$ **by** (*intro nn-integral-PInf-AE*) *auto*
next
fix *x* **assume** *x*: *x* $\in$ *space* (*stock-measure t*) **and** *fin*: *?dens x* $\neq \infty$
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 nonneg-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*] *nonneg-cexpr-map-vars*)
auto
qed (*insert vars ctype edc-add.prems*, *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* $\circ_c$ (λ_c*x*. *inverse*$_c$ *x*)) $*_c$ (λ_c*x*. (*inverse*$_c$ *x*) $\widehat{\ }_c$ *CInt 2*)

have *dens*: *dens-ctxt-*α (*vs*, *vs'*, Γ, δ) $\vdash_d$ *e* $\Rightarrow$ ($\lambda\varrho$ *x*. *ennreal* (*eval-cexpr f* ϱ *x*)) **and**
wf: *is-density-expr* (*vs*, *vs'*, Γ, δ) *REAL f* **using** *edc-inv t* **by** *simp-all*
note *wf'* = *is-density-exprD*[*OF wf*]
from *wf'* **have** *ctype*: *case-nat REAL* Γ $\vdash_c$ *?expr* : *REAL*
by (*auto intro*!: *cet-op-intros cexpr-typing-cexpr-comp cet-var' cet-val'*)
from *wf'* **have** *vars'*: *free-vars* (*f* $\circ_c$ (λ_c*x*. *inverse*$_c$ *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-exprI*, *simp only*: *dens-ctxt-*α*-def prod.case*, *rule hd-cong*[*OF hd-inv*])
fix ϱ *x* **assume** ϱ: $\varrho \in$ *space* (*state-measure* (*set vs'*) Γ)
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 ϱ **and** *wf'* **have** *val-type* (*cexpr-sem* (*case-nat* (*RealVal* (*inverse x'*)) ϱ) *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* ϱ (*op-sem Inverse x*)) $*$ *ennreal* ((*inverse* (*extract-real x*))^2) =
ennreal (*eval-cexpr ?expr* ϱ *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* Γ) (*inverse*$_c$ (*CVar 0*) $\widehat{\ }_c$ *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* Γ) *?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-*α *simp*: *dens-ctxt-*α*-def*)

next
case (*edc-exp 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-exp.prems*(*2*)]
let *?expr* = (λ_c*x*. *IF*$_c$ *CReal 0* $<_c$ *x THEN* (*f* $\circ_c$ *ln*$_c$ *x*) $*_c$ *inverse*$_c$ *x ELSE CReal 0*)

have *dens*: *dens-ctxt-α* (*vs*, *vs′*, Γ, δ) $\vdash_d$ *e* ⇒ (λϱ *x*. *ennreal* (*eval-cexpr f* ϱ *x*))
and
wf: *is-density-expr* (*vs*, *vs′*, Γ, δ) *REAL f* **using** *edc-exp t* **by** *simp-all*
note *wf′* = *is-density-exprD*[*OF wf*]
from *wf′* **have** *ctype*: *case-nat REAL* Γ $\vdash_c$ *?expr* : *REAL*
by (*auto intro*!: *cet-if cet-op-intros cet-var′ cet-val′ cexpr-typing-cexpr-comp*)
from *wf′* **have** *free-vars* (*f* $\circ_c$ ($\lambda_c x$. ln_c *x*)) ⊆ *shift-var-set* (*set vs′*)
by (*intro order.trans*[*OF free-vars-cexpr-comp*]) *auto*
hence *vars*: *free-vars ?expr* ⊆ *shift-var-set* (*set vs′*) **using** *free-vars-cexpr-comp*
by *simp*

show *?case*
proof (*intro conjI is-density-exprI*, *simp only*: *dens-ctxt-α-def prod.case*, *rule hd-cong*[*OF hd-exp*])
fix ϱ *x* **assume** ϱ: ϱ ∈ *space* (*state-measure* (*set vs′*) Γ)
and *x*: *x* ∈ *space* (*stock-measure REAL*)
from *x* **obtain** *x′* **where** [*simp*]: *x* = *RealVal x′* **by** (*auto simp*: *val-type-eq-REAL*)
from ϱ **and** *wf′* **have** *val-type* (*cexpr-sem* (*case-nat* (*RealVal* (*ln x′*)) ϱ) *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* ϱ (*lift-RealVal safe-ln x*))
*
ennreal (*inverse* (*extract-real x*)) *else 0*) = *ennreal* (*eval-cexpr ?expr* ϱ
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* Γ) *?expr*
proof (*rule nonneg-cexprI-shift*)
fix *x* σ **assume** *x* ∈ *type-universe t* **and** σ: σ ∈ *space* (*state-measure* (*set vs′*)
Γ)
then obtain *r* **where** *x* = *RealVal r*
by (*auto simp*: *val-type-eq-REAL*)
moreover note σ *nonneg-cexprD*[*OF is-density-exprD-nonneg*[*OF wf*], *of case-nat* (*RealVal* (*ln r*)) σ]
moreover have *val-type* (*cexpr-sem* (*case-nat* (*RealVal* (*ln r*)) σ) *f*) = *REAL*
using σ **by** (*intro val-type-cexpr-sem*[*OF wf′*(*1*,*2*)] *case-nat-in-state-measure*)
auto
ultimately show *0* ≤ *extract-real*
(*cexpr-sem* (*case-nat x* σ)
(IF_c *CReal 0* $<_c$ *CVar 0 THEN* (*f* $\circ_c$ 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.prems ctype vars dens*,
auto intro!: *density-context-α simp*: *dens-ctxt-α-def*)

qed

lemma *expr-has-density-cexpr-sound*:
assumes ([], [], Γ, *CReal 1*) $\vdash_c$ *e* ⇒ *f* Γ ⊢ *e* : *t free-vars e* = {}
shows *has-subprob-density* (*expr-sem σ e*) (*stock-measure t*) (λ*x*. *ennreal* (*eval-cexpr f σ x*))
∀ *x*∈*type-universe t*. *0* ≤ *extract-real* (*cexpr-sem* (*case-nat x σ*) *f*)
Γ′ *0* = *t* ⟹ Γ′ $\vdash_c$ *f* : *REAL*
free-vars f ⊆ {*0*}
proof−
have *dens-ctxt-α* ([], [], Γ, *CReal 1*) $\vdash_d$ *e* ⇒ (λϱ *x*. *ennreal* (*eval-cexpr f* ϱ *x*)) ∧
is-density-expr ([], [], 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* ([], [], Γ, *CReal 1*) $\vdash_d$ *e* ⇒ (λϱ *x*. *ennreal* (*eval-cexpr f* ϱ *x*))
and *wf*: *is-density-expr* ([], [], Γ, *CReal 1*) *t f* **using** *assms* **by** *blast+*

have *has-subprob-density* (*expr-sem σ e*) (*stock-measure t*)
(λ*x*. *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 ⋀*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* ⟷ *has-subprob-density* (*expr-sem σ e*) (*stock-measure t*)
(λ*x*. *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* ⊆ {*0*} **by** (*auto simp*: *shift-var-set-def*)

show ∀ *x*∈*type-universe t*. *0* ≤ *extract-real* (*cexpr-sem* (*case-nat x σ*) *f*)
proof
fix *v* **assume** *v*: *v* ∈ *type-universe t*
then have *0* ≤ *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 ‹*free-vars f* ⊆ {*0*}› **by** (*intro cexpr-sem-eq-on-vars*) *auto*
finally show *0* ≤ *extract-real* (*cexpr-sem* (*case-nat v σ*) *f*) **.**
qed

assume Γ' *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* ⇒ *pdf-type* ⇒ *cexpr* ⇒ *bool* (- : - $\Rightarrow_c$ - [*10*,*0*,*10*] *10*)
for *e t f* **where**
(λ-. *UNIT*) ⊢ *e* : *t* ⟹ *free-vars e* = {} ⟹
([], [], λ-. *UNIT*, *CReal 1*) $\vdash_c$ *e* ⇒ *f* ⟹
e : *t* $\Rightarrow_c$ *f*

code-pred *expr-compiles-to* **.**

lemma *expr-compiles-to-sound*:
assumes *e* : *t* $\Rightarrow_c$ *f*
shows *expr-sem σ e* = *density* (*stock-measure t*) (λ*x*. *ennreal* (*eval-cexpr f σ' x*))
∀ *x*∈*type-universe t*. *eval-cexpr f σ' x* ≥ *0*
Γ ⊢ *e* : *t*
t · $\Gamma' \vdash_c f$: *REAL*
free-vars f ⊆ {*0*}
proof−
let *?Γ* = λ-. *UNIT*
from *assms* **have** *A*: ([], [], *?Γ*, *CReal 1*) $\vdash_c$ *e* ⇒ *f* *?Γ* ⊢ *e* : *t* *free-vars e* = {}
by (*simp-all add*: *expr-compiles-to.simps*)
hence *expr-sem σ e* = *expr-sem σ' e* **by** (*intro expr-sem-eq-on-vars*) *simp*
with *expr-has-density-cexpr-sound*[*OF A*]
show *expr-sem σ e* = *density* (*stock-measure t*) (λ*x*. *ennreal* (*eval-cexpr f σ' x*))
∀ *x*∈*type-universe t*. *eval-cexpr f σ' x* ≥ *0*
t · $\Gamma' \vdash_c f$: *REAL*
free-vars f ⊆ {*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** (λ-. *UNIT*) ⊢ *e* : *t* **by** (*simp add*: *expr-compiles-to.simps*)
from *this* **and** *assms* **show** Γ ⊢ *e* : *t*
by (*subst expr-typing-cong*) (*auto simp*: *expr-compiles-to.simps*)
qed

11 Tests

values {(*t*, *f*) |*t f*. *Val* (*IntVal 42*) : *t* $\Rightarrow_c$ *f*}
values {(*t*, *f*) |*t f*. *Minus* $$ (*Val* (*IntVal 42*)) : *t* $\Rightarrow_c$ *f*}
values {(*t*, *f*) |*t f*. *Fst* $$ (*Val* <|*IntVal 13*, *IntVal 37*|>) : *t* $\Rightarrow_c$ *f*}
values {(*t*, *f*) |*t f*. *Random Bernoulli* (*Val* (*RealVal 0.5*)) : *t* $\Rightarrow_c$ *f*}
values {(*t*, *f*) |*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) |t f. cthulhu′ 42 : t ⇒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.