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* ⇒ (*'a* ⇒ *ennreal*) ⇒ *bool* **where**
is-subprob-density M f ≡ (*f* ∈ *borel-measurable M*) ∧ *space M* ≠ {} ∧
(∀ *x*∈*space M*. *f x* ≥ *0*) ∧ (∫⁺*x*. *f x* ∂*M*) ≤ *1*

lemma *is-subprob-densityI*[*intro*]:
⟦*f* ∈ *borel-measurable M*; ⋀*x*. *x* ∈ *space M* ⟹ *f x* ≥ *0*; *space M* ≠ {}; (∫⁺*x*. *f x* ∂*M*) ≤ *1*⟧
⟹ *is-subprob-density M f*
⟨*proof*⟩

lemma *is-subprob-densityD*[*dest*]:
is-subprob-density M f ⟹ *f* ∈ *borel-measurable M*
is-subprob-density M f ⟹ *x* ∈ *space M* ⟹ *f x* ≥ *0*
is-subprob-density M f ⟹ *space M* ≠ {}
is-subprob-density M f ⟹ (∫⁺*x*. *f x* ∂*M*) ≤ *1*
⟨*proof*⟩

1.2 Measure spaces with densities

definition *has-density* :: *'a measure* ⇒ *'a measure* ⇒ (*'a* ⇒ *ennreal*) ⇒ *bool* **where**
has-density M N f ⟷ (*f* ∈ *borel-measurable N*) ∧ *space N* ≠ {} ∧ *M* = *density N f*

lemma *has-densityI*[*intro*]:
⟦*f* ∈ *borel-measurable N*; *M* = *density N f*; *space N* ≠ {}⟧ ⟹ *has-density M N f*
⟨*proof*⟩

lemma *has-densityD*:
assumes *has-density M N f*
shows *f* ∈ *borel-measurable N M* = *density N f space N* ≠ {}
⟨*proof*⟩

lemma *has-density-sets*: *has-density M N f* ⟹ *sets M* = *sets N*
⟨*proof*⟩

lemma *has-density-space*: *has-density M N f* ⟹ *space M* = *space N*
⟨*proof*⟩

lemma *has-density-emeasure*:
has-density M N f ⟹ *X* ∈ *sets M* ⟹ *emeasure M X* = ∫⁺*x*. *f x* ∗ *indicator X x* ∂*N*
⟨*proof*⟩

lemma *nn-integral-cong'*: (⋀*x*. *x* ∈ *space N* =*simp*=> *f x* = *g x*) ⟹ (∫⁺*x*. *f x* ∂*N*) = (∫⁺*x*. *g x* ∂*N*)
⟨*proof*⟩

lemma *has-density-emeasure-space*:
  has-density M N f $\Longrightarrow$ *emeasure M* (*space M*) = ($\int^+ x.$ *f x* ∂N)
⟨*proof*⟩

lemma *has-density-emeasure-space'*:
  has-density M N f $\Longrightarrow$ *emeasure* (*density N f*) (*space* (*density N f*)) = $\int^+ x.$ *f x* ∂N
⟨*proof*⟩

lemma *has-density-imp-is-subprob-density*:
  ⟦*has-density M N f*; ($\int^+ x.$ *f x* ∂N) = *1*⟧ $\Longrightarrow$ *is-subprob-density N f*
⟨*proof*⟩

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

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*⟩

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*
⟨*proof*⟩

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'*
⟨*proof*⟩

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*
⟨*proof*⟩

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

lemma *has-subprob-density-imp-subprob-space*[*dest*]:
  is-subprob-density M f $\Longrightarrow$ *subprob-space* (*density M f*)
⟨*proof*⟩

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

lemma *subprob-space-density-not-empty*: *subprob-space* (*density M f*) $\Longrightarrow$ *space M* $\neq$ {}
⟨*proof*⟩

lemma *has-subprob-densityI*:
⟦*f* ∈ *borel-measurable N*; *M* = *density N f*; *subprob-space M*⟧ $\Longrightarrow$ *has-subprob-density M N f*
⟨*proof*⟩

lemma *has-subprob-densityI'*:
assumes *f* ∈ *borel-measurable N space N* ≠ {}
M = *density N f* (∫⁺*x*. *f x* ∂*N*) ≤ *1*
shows *has-subprob-density M N f*
⟨*proof*⟩

lemma *has-subprob-densityD*:
assumes *has-subprob-density M N f*
shows *f* ∈ *borel-measurable N* ⋀*x*. *x* ∈ *space N* $\Longrightarrow$ *f x* ≥ *0 M* = *density N f* *subprob-space M*
⟨*proof*⟩

lemma *has-subprob-density-measurable*[*measurable-dest*]:
has-subprob-density M N f $\Longrightarrow$ *f* ∈ *N* →$_M$ *borel*
⟨*proof*⟩

lemma *has-subprob-density-imp-has-density*:
has-subprob-density M N f $\Longrightarrow$ *has-density M N f* ⟨*proof*⟩

lemma *has-subprob-density-equal-on-space*:
assumes *has-subprob-density M N f* ⋀*x*. *x* ∈ *space N* $\Longrightarrow$ *f x* = *g x*
shows *has-subprob-density M N g*
⟨*proof*⟩

lemma *has-subprob-density-cong*:
assumes ⋀*x*. *x* ∈ *space N* $\Longrightarrow$ *f x* = *g x*
shows *has-subprob-density M N f* = *has-subprob-density M N g*
⟨*proof*⟩

lemma *has-subprob-density-dens-AE*:
⟦*AE y in N*. *f y* = *f' y*; *f'* ∈ *borel-measurable N*;
⋀*x*. *x* ∈ *space M* $\Longrightarrow$ *f' x* ≥ *0*; *has-subprob-density M N f*⟧
$\Longrightarrow$ *has-subprob-density M N f'*
⟨*proof*⟩

1.4 Parametrized probability densities

definition

has-parametrized-subprob-density M N R f ≡
(∀ *x* ∈ *space M*. *has-subprob-density* (*N x*) *R* (*f x*)) ∧ *case-prod f* ∈ *borel-measurable* (*M* ⨂$_M$ *R*)

lemma *has-parametrized-subprob-densityI*:
assumes ⋀*x*. *x* ∈ *space M* ⟹ *N x* = *density R* (*f x*)
assumes ⋀*x*. *x* ∈ *space M* ⟹ *subprob-space* (*N x*)
assumes *case-prod f* ∈ *borel-measurable* (*M* ⨂$_M$ *R*)
shows *has-parametrized-subprob-density M N R f*
⟨*proof*⟩

lemma *has-parametrized-subprob-densityD*:
assumes *has-parametrized-subprob-density M N R f*
shows ⋀*x*. *x* ∈ *space M* ⟹ *N x* = *density R* (*f x*)
and ⋀*x*. *x* ∈ *space M* ⟹ *subprob-space* (*N x*)
and [*measurable-dest*]: *case-prod f* ∈ *borel-measurable* (*M* ⨂$_M$ *R*)
⟨*proof*⟩

lemma *has-parametrized-subprob-density-integral*:
assumes *has-parametrized-subprob-density M N R f x* ∈ *space M*
shows (∫$^+$*y*. *f x y* ∂*R*) ≤ *1*
⟨*proof*⟩

lemma *has-parametrized-subprob-density-cong*:
assumes ⋀*x*. *x* ∈ *space M* ⟹ *N x* = *N′ x*
shows *has-parametrized-subprob-density M N R f* = *has-parametrized-subprob-density M N′ R f*
⟨*proof*⟩

lemma *has-parametrized-subprob-density-dens-AE*:
assumes ⋀*x*. *x* ∈ *space M* ⟹ *AE y in R*. *f x y* = *f′ x y*
case-prod f′ ∈ *borel-measurable* (*M* ⨂$_M$ *R*)
has-parametrized-subprob-density M N R f
shows *has-parametrized-subprob-density M N R f′*
⟨*proof*⟩

1.5 Density in the Giry monad

lemma *emeasure-bind-density*:
assumes *space M* ≠ {} ⋀*x*. *x* ∈ *space M* ⟹ *has-density* (*f x*) *N* (*g x*)
f ∈ *measurable M* (*subprob-algebra N*) *X* ∈ *sets N*
shows *emeasure* (*M* ⤜ *f*) *X* = ∫$^+$*x*. ∫$^+$*y*. *g x y* ∗ *indicator X y* ∂*N* ∂*M*
⟨*proof*⟩

lemma *bind-density*:
assumes *sigma-finite-measure M sigma-finite-measure N*
space M ≠ {} ⋀*x*. *x* ∈ *space M* ⟹ *has-density* (*f x*) *N* (*g x*)
and [*measurable*]: *case-prod g* ∈ *borel-measurable* (*M* ⨂$_M$ *N*) *f* ∈ *measurable M* (*subprob-algebra N*)

shows $(M \gg\!= f) = \mathit{density}\ N\ (\lambda y.\ \int^+ x.\ g\ x\ y\ \partial M)$
⟨*proof*⟩

lemma *bind-has-density*:
assumes *sigma-finite-measure M sigma-finite-measure N*
$\mathit{space}\ M \neq \{\}\ \bigwedge x.\ x \in \mathit{space}\ M \Longrightarrow \mathit{has\text{-}density}\ (f\ x)\ N\ (g\ x)$
$\mathit{case\text{-}prod}\ g \in \mathit{borel\text{-}measurable}\ (M \bigotimes_M N)$
$f \in \mathit{measurable}\ M\ (\mathit{subprob\text{-}algebra}\ N)$
shows $\mathit{has\text{-}density}\ (M \gg\!= f)\ N\ (\lambda y.\ \int^+ x.\ g\ x\ y\ \partial M)$
⟨*proof*⟩

lemma *bind-has-density'*:
assumes *sfM*: *sigma-finite-measure M*
and *sfR*: *sigma-finite-measure R*
and *not-empty*: $\mathit{space}\ M \neq \{\}$ **and** *dens-M*: *has-density M N δM*
and *dens-f*: $\bigwedge x.\ x \in \mathit{space}\ M \Longrightarrow \mathit{has\text{-}density}\ (f\ x)\ R\ (\delta f\ x)$
and *Mδf*: $\mathit{case\text{-}prod}\ \delta f \in \mathit{borel\text{-}measurable}\ (N \bigotimes_M R)$
and *Mf*: $f \in \mathit{measurable}\ N\ (\mathit{subprob\text{-}algebra}\ R)$
shows $\mathit{has\text{-}density}\ (M \gg\!= f)\ R\ (\lambda y.\ \int^+ x.\ \delta M\ x * \delta f\ x\ y\ \partial N)$
⟨*proof*⟩

lemma *bind-has-subprob-density*:
assumes *subprob-space M sigma-finite-measure N*
$\mathit{space}\ M \neq \{\}\ \bigwedge x.\ x \in \mathit{space}\ M \Longrightarrow \mathit{has\text{-}density}\ (f\ x)\ N\ (g\ x)$
$\mathit{case\text{-}prod}\ g \in \mathit{borel\text{-}measurable}\ (M \bigotimes_M N)$
$f \in \mathit{measurable}\ M\ (\mathit{subprob\text{-}algebra}\ N)$
shows $\mathit{has\text{-}subprob\text{-}density}\ (M \gg\!= f)\ N\ (\lambda y.\ \int^+ x.\ g\ x\ y\ \partial M)$
⟨*proof*⟩

lemma *bind-has-subprob-density'*:
assumes *has-subprob-density M N δM space R* $\neq \{\}$ *sigma-finite-measure R*
$\bigwedge x.\ x \in \mathit{space}\ M \Longrightarrow \mathit{has\text{-}subprob\text{-}density}\ (f\ x)\ R\ (\delta f\ x)$
$\mathit{case\text{-}prod}\ \delta f \in \mathit{borel\text{-}measurable}\ (N \bigotimes_M R)\ f \in \mathit{measurable}\ N\ (\mathit{subprob\text{-}algebra}\ R)$
shows $\mathit{has\text{-}subprob\text{-}density}\ (M \gg\!= f)\ R\ (\lambda y.\ \int^+ x.\ \delta M\ x * \delta f\ x\ y\ \partial N)$
⟨*proof*⟩

lemma *null-measure-has-subprob-density*:
$\mathit{space}\ M \neq \{\} \Longrightarrow \mathit{has\text{-}subprob\text{-}density}\ (\mathit{null\text{-}measure}\ M)\ M\ (\lambda\text{-}.\ 0)$
⟨*proof*⟩

lemma *emeasure-has-parametrized-subprob-density*:
assumes *has-parametrized-subprob-density M N R f*
assumes $x \in \mathit{space}\ M\ X \in \mathit{sets}\ R$
shows $\mathit{emeasure}\ (N\ x)\ X = \int^+ y.\ f\ x\ y * \mathit{indicator}\ X\ y\ \partial R$
⟨*proof*⟩

lemma *emeasure-count-space-density-singleton*:

assumes $x \in A$ *has-density* M (*count-space* A) f
shows *emeasure* M $\{x\} = f\ x$
⟨*proof*⟩

lemma *subprob-count-space-density-le-1*:
assumes *has-subprob-density* M (*count-space* A) $f\ x \in A$
shows $f\ x \leq 1$
⟨*proof*⟩

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* $M\ N\ \delta$
(**is** *has-density* *?M'* *?N'* *?δ'* $\longleftrightarrow$ *has-density* $M\ N\ \delta$)
⟨*proof*⟩

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 $(\delta \circ f) \longleftrightarrow$ *has-density* M (*embed-measure* $N\ f$) δ
⟨*proof*⟩

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 $(\delta \circ f)$
⟨*proof*⟩

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 $(\delta \circ f)$
⟨*proof*⟩

end

2 Measure Space Transformations

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

lemma *not-top-le-1-ennreal*[*simp*]: $\neg$ *top* $\leq$ (*1*::*ennreal*)
⟨*proof*⟩

lemma *range-int*: *range int* = $\{n.\ n \geq 0\}$
⟨*proof*⟩

lemma *range-exp*: *range* (*exp* :: *real* ⇒ *real*) = {*x*. *x* > *0*}
⟨*proof*⟩

lemma *Int-stable-Icc*: *Int-stable* (*range* (λ(*a*, *b*). {*a* .. *b*::*real*}))
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *count-space-uminus*:
count-space UNIV = *distr* (*count-space UNIV*) (*count-space UNIV*) (*uminus* :: ('*a* :: *ring* ⇒ -))
⟨*proof*⟩

lemma *count-space-plus*:
count-space UNIV = *distr* (*count-space UNIV*) (*count-space UNIV*) ((+) (*c* :: ('*a* :: *ring*)))
⟨*proof*⟩

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*⟩

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*⟩

lemma *subprob-density-distr-real-eq*:
assumes *dens*: *has-subprob-density M lborel f*
assumes *Mh*: *h* ∈ *borel-measurable borel*
assumes *Mg*: *g* ∈ *borel-measurable borel*

assumes *measure-eq*:
$\bigwedge a\ b.\ a \leq b \Longrightarrow$ *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*⟩

lemma *subprob-density-distr-real-exp*:
assumes *dens*: *has-subprob-density M lborel f*
shows *has-subprob-density* (*distr M borel exp*) *lborel*
(λ*x*. *if x* > *0 then f* (*ln x*) ∗ *ennreal* (*inverse x*) *else 0*)
(**is** *has-subprob-density - - ?g*)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

lemma *map-int-pair-REAL*[*simp*]: *map-int-pair f g* <| *RealVal i*, *RealVal j* |> = *g* <| *RealVal i*, *RealVal j* |>
⟨*proof*⟩

lemma *map-real-pair*[*simp*]: *map-real-pair f g* <| *RealVal i*, *RealVal j* |> = *f i j*
⟨*proof*⟩

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* ⟨*proof*⟩
lemma *inj-IntPairVal*: *inj IntPairVal* ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *val-type-eq-INTEG*: *val-type x* = *INTEG* ⟷ *x* ∈ *IntVal‘UNIV*
  ⟨*proof*⟩

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

lemma *type-universe-nonempty*[*simp*]: *type-universe t* ≠ {}
  ⟨*proof*⟩

lemma *val-in-type-universe*[*simp*]:
    v ∈ *type-universe* (*val-type v*)
  ⟨*proof*⟩

lemma *BoolVal-in-type-universe*[*simp*]: *BoolVal v* ∈ *type-universe BOOL*
  ⟨*proof*⟩

lemma *IntVal-in-type-universe*[*simp*]: *IntVal v* ∈ *type-universe INTEG*
  ⟨*proof*⟩

lemma *type-universe-type*[*simp*]:
    w ∈ *type-universe t* ⟷ *val-type w* = *t*
  ⟨*proof*⟩

lemma *type-universe-REAL*: *type-universe REAL* = *RealVal ‘ UNIV*
  ⟨*proof*⟩

lemma *type-universe-eq-imp-type-eq*:
  assumes *type-universe t1* = *type-universe t2*
  shows *t1* = *t2*
⟨*proof*⟩

lemma *type-universe-eq-iff*[*simp*]: *type-universe t1 = type-universe t2* $\longleftrightarrow$ *t1 = t2*
⟨*proof*⟩

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*)
⟨*proof*⟩

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 (λ*x*. *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*⟩

lemma *space-stock-measure*[*simp*]: *space* (*stock-measure t*) = *type-universe t*
⟨*proof*⟩

lemma *type-universe-stock-measure*[*measurable*]: *type-universe t* ∈ *sets* (*stock-measure t*)
⟨*proof*⟩

lemma *inj-RealVal*[*simp*]: *inj RealVal* ⟨*proof*⟩
lemma *inj-IntVal*[*simp*]: *inj IntVal* ⟨*proof*⟩
lemma *inj-BoolVal*[*simp*]: *inj BoolVal* ⟨*proof*⟩
lemma *inj-PairVal*[*simp*]: *inj* (λ(*x*, *y*). <| *x* , *y* |>) ⟨*proof*⟩

lemma *measurable-PairVal*[*measurable*]:
fixes *t1 t2* :: *pdf-type*
shows *case-prod PairVal* ∈ *measurable* (*t1* $\bigotimes_M$ *t2*) (*PRODUCT t1 t2*)
⟨*proof*⟩

lemma *measurable-RealVal*[*measurable*]: *RealVal* ∈ *measurable borel REAL*
⟨*proof*⟩

lemma *nn-integral-BoolVal*:
assumes $\bigwedge x.\ f\ (BoolVal\ x) \geq 0$
shows $(\int^+ x.\ f\ x\ \partial BOOL) = f\ (BoolVal\ True) + f\ (BoolVal\ False)$
⟨*proof*⟩

lemma *nn-integral-RealVal*:
$f \in$ *borel-measurable REAL* $\Longrightarrow$ $(\int^+ x.\ f\ x\ \partial REAL) = (\int^+ x.\ f\ (RealVal\ x)\ \partial lborel)$
⟨*proof*⟩

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

lemma *nn-integral-PairVal*:
$f \in$ *borel-measurable* $(PRODUCT\ t1\ t2) \Longrightarrow$
$(\int^+ x.\ f\ x\ \partial PRODUCT\ t1\ t2) = (\int^+ x.\ f\ (PairVal\ (fst\ x)\ (snd\ x))\ \partial(t1 \bigotimes_M t2))$
⟨*proof*⟩

lemma *BOOL-E*: $[\![val\text{-}type\ v = BOOL;\ \bigwedge b.\ v = BoolVal\ b \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *PROD-E*: $[\![val\text{-}type\ v = PRODUCT\ t1\ t2\ ;$
$\bigwedge a\ b.\ val\text{-}type\ a = t1 \Longrightarrow val\text{-}type\ b = t2 \Longrightarrow v = <|\ a,\ b\ |> \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *REAL-E*: $[\![val\text{-}type\ v = REAL;\ \bigwedge b.\ v = RealVal\ b \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *INTEG-E*: $[\![val\text{-}type\ v = INTEG;\ \bigwedge i.\ v = IntVal\ i \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

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 $(\lambda x.\ extract\text{-}pair'\ f\ g\ (h\ x)) \in measurable\ K\ (M \bigotimes_M N)$
⟨*proof*⟩

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

lemma *measurable-extract-real*[*measurable*]: *extract-real* $\in$ *measurable REAL borel*
⟨*proof*⟩

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

lemma *measurable-extract-int-pair*[*measurable*]:
extract-int-pair ∈ *measurable* (*PRODUCT INTEG INTEG*) (*count-space UNIV* ⨂$_M$ *count-space UNIV*)
⟨*proof*⟩

lemma *measurable-extract-real-pair*[*measurable*]:
extract-real-pair ∈ *measurable* (*PRODUCT REAL REAL*) (*borel* ⨂$_M$ *borel*)
⟨*proof*⟩

lemma *measurable-extract-real-pair'*[*measurable*]:
extract-real-pair ∈ *measurable* (*PRODUCT REAL REAL*) *borel*
⟨*proof*⟩

lemma *measurable-extract-bool*[*measurable*]: *extract-bool* ∈ *measurable BOOL* (*count-space UNIV*)
⟨*proof*⟩

lemma *map-int-pair-measurable*[*measurable*]:
assumes *f*: *case-prod f* ∈ *measurable* (*count-space UNIV* ⨂$_M$ *count-space UNIV*) *M*
shows *map-int-pair f g* ∈ *measurable* (*PRODUCT INTEG INTEG*) *M*
⟨*proof*⟩

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*⟩

lemma *map-real-pair-measurable*[*measurable*]:
assumes *f*: *case-prod f* ∈ *measurable* (*borel* ⨂$_M$ *borel*) *M*
shows *map-real-pair f g* ∈ *measurable* (*PRODUCT REAL REAL*) *M*
⟨*proof*⟩

lemma *count-space-IntVal-prod*[*simp*]: *INTEG* ⨂$_M$ *INTEG* = *count-space* (*range IntVal* × *range IntVal*)
⟨*proof*⟩

lemma *count-space-BoolVal-prod*[*simp*]: *BOOL* ⨂$_M$ *BOOL* = *count-space* (*range BoolVal* × *range BoolVal*)
⟨*proof*⟩

lemma *measurable-stock-measure-val-type*:
assumes *f* ∈ *measurable M* (*stock-measure t*) *x* ∈ *space M*
shows *val-type* (*f x*) = *t*
⟨*proof*⟩

lemma *singleton-in-stock-measure*[*simp*]: *val-type* $v = t \Longrightarrow \{v\} \in$ *sets* t
⟨*proof*⟩

lemma *emeasure-stock-measure-singleton-finite*[*simp*]:
emeasure (*stock-measure* (*val-type* v)) $\{v\} \neq \infty$
⟨*proof*⟩

3.2 Measures on states

definition *state-measure* :: *vname set* ⇒ (*vname* ⇒ *pdf-type*) ⇒ *state measure*
where
state-measure $V\ \Gamma \equiv \Pi_M\ y{\in}V.\ \Gamma\ y$

lemma *state-measure-nonempty*[*simp*]: *space* (*state-measure* $V\ \Gamma$) $\neq \{\}$
⟨*proof*⟩

lemma *space-state-measure*: *space* (*state-measure* $V\ \Gamma$) = ($\Pi_E\ y{\in}V.$ *type-universe* ($\Gamma\ y$))
⟨*proof*⟩

lemma *state-measure-var-type*:
$\sigma \in$ *space* (*state-measure* $V\ \Gamma$) $\Longrightarrow x \in V \Longrightarrow$ *val-type* ($\sigma\ x$) = $\Gamma\ x$
⟨*proof*⟩

lemma *merge-in-state-measure*:
$x \in$ *space* (*state-measure* $A\ \Gamma$) $\Longrightarrow y \in$ *space* (*state-measure* $B\ \Gamma$) $\Longrightarrow$
merge $A\ B\ (x, y) \in$ *space* (*state-measure* ($A{\cup}B$) Γ) ⟨*proof*⟩

lemma *measurable-merge-stock*[*measurable* (*raw*)]:
$f \in N \rightarrow_M$ *state-measure* $V\ \Gamma \Longrightarrow g \in N \rightarrow_M$ *state-measure* $V'\ \Gamma \Longrightarrow$
($\lambda x.$ *merge* $V\ V'$ ($f\ x, g\ x$)) $\in N \rightarrow_M$ *state-measure* ($V \cup V'$) Γ
⟨*proof*⟩

lemma *comp-in-state-measure*:
assumes $\sigma \in$ *space* (*state-measure* $V\ \Gamma$)
shows $\sigma \circ f \in$ *space* (*state-measure* ($f -`$ V) ($\Gamma \circ f$))
⟨*proof*⟩

lemma *sigma-finite-state-measure*[*intro*]:
finite $V \Longrightarrow$ *sigma-finite-measure* (*state-measure* $V\ \Gamma$) ⟨*proof*⟩

3.3 Equalities of measure embeddings

lemma *embed-measure-RealPairVal*:
stock-measure (*PRODUCT REAL REAL*) = *embed-measure lborel RealPairVal*
⟨*proof*⟩

lemma *embed-measure-IntPairVal*:
stock-measure (*PRODUCT INTEG INTEG*) = *count-space* (*range IntPairVal*)

⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *measurable-return-val*[*simp*]:
return-val ∈ *measurable* (*stock-measure t*) (*subprob-algebra* (*stock-measure t*))
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *bind-return-val″*:
assumes *f* ∈ *measurable* (*stock-measure* (*val-type x*)) (*subprob-algebra M*)
shows *return-val x* ⤜ *f* = *f x*
⟨*proof*⟩

lemma *bind-assoc-return-val*:
assumes *sets-M*: *sets M* = *sets* (*stock-measure t*)
assumes *Mf*: *f* ∈ *measurable* (*stock-measure t*) (*stock-measure t′*)
assumes *Mg*: *g* ∈ *measurable* (*stock-measure t′*) (*stock-measure t″*)
shows (*M* ⤜ (λ*x*. *return-val* (*f x*))) ⤜ (λ*x*. *return-val* (*g x*)) =
M ⤜ (λ*x*. *return-val* (*g* (*f x*)))
⟨*proof*⟩

lemma *bind-return-val-distr*:
assumes *sets-M*: *sets M* = *sets* (*stock-measure t*)
assumes *Mf*: *f* ∈ *measurable* (*stock-measure t*) (*stock-measure t′*)
shows *M* ⤜ *return-val* ∘ *f* = *distr M* (*stock-measure t′*) *f*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *lift-RealIntVal-Real*:
x ∈ *space* (*stock-measure REAL*) ⟹ *lift-RealIntVal f g x* = *lift-RealVal g x*
⟨*proof*⟩

lemma *lift-RealIntVal-Int*:
x ∈ *space* (*stock-measure INTEG*) ⟹ *lift-RealIntVal f g x* = *lift-IntVal f x*
⟨*proof*⟩

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

lemma *measurable-lift-RealVal*[*measurable*]:
assumes [*measurable*]: *f* ∈ *borel-measurable borel*
shows *lift-RealVal f* ∈ *measurable REAL REAL*
⟨*proof*⟩

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

lemma *measurable-lift-IntVal'*[*measurable*]: *lift-IntVal f* ∈ *measurable INTEG INTEG*
⟨*proof*⟩

lemma *split-apply*: (*case x of* (*a*, *b*) ⇒ *f a b*) *y* = (*case x of* (*a*, *b*) ⇒ *f a b y*)
⟨*proof*⟩

lemma *measurable-lift-Comp-RealVal*[*measurable*]:
assumes [*measurable*]: *Measurable.pred* (*borel* ⨂$_M$ *borel*) (*case-prod g*)
shows *lift-Comp f g* ∈ *measurable* (*PRODUCT REAL REAL*) *BOOL*
⟨*proof*⟩

lemma *measurable-lift-Comp-IntVal*[*simp*]:
lift-Comp f g ∈ *measurable* (*PRODUCT INTEG INTEG*) *BOOL*
⟨*proof*⟩

lemma *measurable-lift-RealIntVal-IntVal*[*simp*]: *lift-RealIntVal f g* ∈ *range IntVal* → *range IntVal*
⟨*proof*⟩

lemma *measurable-lift-RealIntVal-IntVal'*[*measurable*]:
lift-RealIntVal f g ∈ *measurable INTEG INTEG*
⟨*proof*⟩

lemma *measurable-lift-RealIntVal-RealVal*[*measurable*]:
assumes [*measurable*]: *g* ∈ *borel-measurable borel*
shows *lift-RealIntVal f g* ∈ *measurable REAL REAL*
⟨*proof*⟩

lemma *measurable-lift-RealIntVal2-IntVal*[*measurable*]:
lift-RealIntVal2 f g ∈ *measurable* (*PRODUCT INTEG INTEG*) *INTEG*
⟨*proof*⟩

lemma *measurable-lift-RealIntVal2-RealVal*[*measurable*]:
assumes [*measurable*]: *case-prod g* ∈ *borel-measurable* (*borel* ⨂$_M$ *borel*)
shows *lift-RealIntVal2 f g* ∈ *measurable* (*PRODUCT REAL REAL*) *REAL*
⟨*proof*⟩

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* (λ*x*. *δ* (*RealVal x*)) ∘ *extract-real*)
⟨*proof*⟩

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* ≡ *distr M* (*stock-measure INTEG*) (*lift-IntVal f*)
shows *has-density N* (*stock-measure INTEG*) (*g* (λ*x*. *δ* (*IntVal x*)) ∘ *extract-int*)
⟨*proof*⟩

lemma *distr-lift-RealPairVal*:
fixes *f f' g*
assumes *Mf*[*measurable*]: *case-prod f* ∈ *borel-measurable borel*
assumes *pdens*: *has-subprob-density M* (*stock-measure* (*PRODUCT REAL REAL*)) *δ*
assumes *dens'*: ⋀*M δ*. *has-subprob-density M lborel δ* ⟹ *has-density* (*distr M*

borel (*case-prod f*)) *lborel* (*g δ*)
defines *N* ≡ *distr M* (*stock-measure REAL*) (*lift-RealIntVal2 f′ f*)
shows *has-density N* (*stock-measure REAL*) (*g* (λ*x*. *δ* (*RealPairVal x*)) ∘ *extract-real*)
⟨*proof*⟩

lemma *distr-lift-IntPairVal*:
fixes *f f′*
assumes *pdens*: *has-density M* (*stock-measure* (*PRODUCT INTEG INTEG*)) *δ*
assumes *dens′*: ⋀*M δ*. *has-density M* (*count-space UNIV*) *δ* ⟹
has-density (*distr M* (*count-space UNIV*) (*case-prod f*))
(*count-space UNIV*) (*g δ*)
defines *N* ≡ *distr M* (*stock-measure INTEG*) (*lift-RealIntVal2 f f′*)
shows *has-density N* (*stock-measure INTEG*) (*g* (λ*x*. *δ* (*IntPairVal x*)) ∘ *extract-int*)
⟨*proof*⟩

end

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

lemma *measurable-subprob-algebra-density*:
assumes *sigma-finite-measure N*
assumes *space N* ≠ {}
assumes [*measurable*]: *case-prod f* ∈ *borel-measurable* (*M* ⨂*M* *N*)
assumes ⋀*x*. *x* ∈ *space M* ⟹ (∫⁺*y*. *f x y* ∂*N*) ≤ *1*
shows (λ*x*. *density N* (*f x*)) ∈ *measurable M* (*subprob-algebra N*)
⟨*proof*⟩

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* ⨂*M* *count-space UNIV*)
⟨*proof*⟩

lemma *measurable-bernoulli*[*measurable*]: *bernoulli* ∈ *measurable REAL* (*subprob-algebra BOOL*)
⟨*proof*⟩

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* ⨂$_M$ *count-space* *UNIV*) ⨂$_M$ *count-space* *UNIV*)
⟨*proof*⟩

lemma *measurable-uniform-density-real*[*measurable*]:
(*case-prod* *uniform-real-density*) ∈ *borel-measurable* (*borel* ⨂$_M$ *borel*)
⟨*proof*⟩

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* ≤ *i* ∧ *i* ≤ *b* *then* *v* *else* *0*) = (*v*::*real*) ∗ *indicator* {*a* .. *b*} *i*
⟨*proof*⟩

lemma *measurable-uniform-int*[*measurable*]:
uniform-int ∈ *measurable* (*PRODUCT* *INTEG* *INTEG*) (*subprob-algebra* *INTEG*)
⟨*proof*⟩

lemma *density-cong′*:
(⋀*x*. *x* ∈ *space* *M* ⟹ *f* *x* = *g* *x*) ⟹ *density* *M* *f* = *density* *M* *g*
⟨*proof*⟩

lemma *measurable-uniform-real*[*measurable*]:
uniform-real ∈ *measurable* (*PRODUCT* *REAL* *REAL*) (*subprob-algebra* *REAL*)
⟨*proof*⟩

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* ⨂$_M$ *borel*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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* ⨂$_M$ *count-space UNIV*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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 ⟨proof⟩
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⟩


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*))
⟨*proof*⟩

lemma *sets-dist-measure*[*simp*]:
val-type x = *dist-param-type dst* ⟹
sets (*dist-measure dst x*) = *sets* (*stock-measure* (*dist-result-type dst*))
⟨*proof*⟩

lemma *space-dist-measure*[*simp*]:
val-type x = *dist-param-type dst* ⟹
space (*dist-measure dst x*) = *type-universe* (*dist-result-type dst*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *subprob-space-dist-measure*:
$v \in$ *type-universe* (*dist-param-type dst*) $\Longrightarrow$ *subprob-space* (*dist-measure dst v*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

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*⟩

lemma *expr-typing-cong*:
(⋀*x*. *x* ∈ *free-vars e* ⟹ Γ *x* = Γ' *x*) ⟹ Γ ⊢ *e* : *t* ⟷ Γ' ⊢ *e* : *t*
⟨*proof*⟩

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

lemma *expr-typing-unique*:
$\Gamma \vdash e : t \Longrightarrow \Gamma \vdash e : t' \Longrightarrow t = t'$
⟨*proof*⟩

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*
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *countable-type-imp-count-space*:
countable-type t $\Longrightarrow$ *stock-measure t* = *count-space* (*type-universe t*)
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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* $\Longrightarrow$ *safe-ln x* = *ln x*
⟨*proof*⟩

lemma *borel-measurable-safe-ln*[*measurable*]: *safe-ln* ∈ *borel-measurable borel*
⟨*proof*⟩

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* $\Longrightarrow$ *safe-sqrt x* = *sqrt x*
⟨*proof*⟩

lemma *borel-measurable-safe-sqrt*[*measurable*]: *safe-sqrt* ∈ *borel-measurable borel*
⟨*proof*⟩

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|>*
⟨*proof*⟩

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′
⟨*proof*⟩

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*⟩

lemma *space-expr-sem*:
Γ ⊢ w : t ⟹ (∀ x ∈ free-vars w. val-type (σ x) = Γ x)
⟹ space (expr-sem σ w) = type-universe t
⟨*proof*⟩

lemma *measurable-expr-sem-eq*:
Γ ⊢ e : t ⟹ σ ∈ space (state-measure V Γ) ⟹ free-vars e ⊆ V ⟹
measurable (expr-sem σ e) = measurable (stock-measure t)
⟨*proof*⟩

lemma *measurable-expr-semI*:
Γ ⊢ e : t ⟹ σ ∈ space (state-measure V Γ) ⟹ free-vars e ⊆ V ⟹
f ∈ measurable (stock-measure t) M ⟹ f ∈ measurable (expr-sem σ e) M
⟨*proof*⟩

lemma *expr-sem-eq-on-vars*:
(⋀x. x∈free-vars e ⟹ σ₁ x = σ₂ x) ⟹ expr-sem σ₁ e = expr-sem σ₂ e
⟨*proof*⟩

5.5 Measurability

lemma *borel-measurable-eq*[*measurable (raw)*]:
assumes [*measurable*]: *f ∈ borel-measurable M g ∈ borel-measurable M*
shows *Measurable.pred M (λx. f x = (g x::real))*
⟨*proof*⟩

lemma *measurable-equals*:
(λ(*x*,*y*). *x* = *y*) ∈ *measurable* (*stock-measure t* ⨂$_M$ *stock-measure t*) (*count-space UNIV*)
⟨*proof*⟩

lemma *measurable-equals-stock-measure*[*measurable* (*raw*)]:
assumes *f* ∈ *measurable M* (*stock-measure t*) *g* ∈ *measurable M* (*stock-measure t*)
shows *Measurable.pred M* (λ*x*. *f x* = *g x*)
⟨*proof*⟩

lemma *measurable-op-sem*:
assumes *op-type oper t* = *Some t′*
shows *op-sem oper* ∈ *measurable* (*stock-measure t*) (*stock-measure t′*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *shift-var-set-Suc*[*simp*]: *Suc x* ∈ *shift-var-set V* ⟷ *x* ∈ *V*
⟨*proof*⟩

lemma *case-nat-update-0*[*simp*]: (*case-nat x σ*)(*0* := *y*) = *case-nat y σ*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *measurable-case-nat′*[*measurable*]:
assumes *g* ∈ *measurable R* (*stock-measure t*) *h* ∈ *measurable R* (*state-measure V* Γ)
shows (λ*x*. *case-nat* (*g x*) (*h x*)) ∈
measurable R (*state-measure* (*shift-var-set V*) (*case-nat t* Γ))

⟨*proof*⟩

lemma *case-nat-in-state-measure*[*intro*]:
assumes $x \in$ *type-universe t1* $\sigma \in$ *space* (*state-measure V* Γ)
shows *case-nat x* $\sigma \in$ *space* (*state-measure* (*shift-var-set V*) (*case-nat t1* Γ))
⟨*proof*⟩

lemma *subset-shift-var-set*:
Suc −‘ $A \subseteq V \Longrightarrow A \subseteq$ *shift-var-set V*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*⟩

lemma *expr-sem-rf-sound*:

$\Gamma \vdash e : t \Longrightarrow$ *randomfree* $e \Longrightarrow$ *free-vars* $e \subseteq V \Longrightarrow \sigma \in$ *space* (*state-measure* V Γ) $\Longrightarrow$
return-val (*expr-sem-rf* σ e) = *expr-sem* σ e
⟨*proof*⟩

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*⟩

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*⟩

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-measure* V Γ) (Γ i)
⟨*proof*⟩

lemma *measurable-dens-ctxt-fun-upd*[*measurable* (*raw*)]:
$f \in N \to_M$ *state-measure* V' $\Gamma \Longrightarrow V = V' \cup \{x\} \Longrightarrow$
$g \in N \to_M$ *stock-measure* (Γ x) $\Longrightarrow$
$(\lambda\omega.\ (f\ \omega)(x := g\ \omega)) \in N \to_M$ *state-measure* V Γ
⟨*proof*⟩

lemma *measurable-case-nat-Suc-PiM*:
$(\lambda\sigma.\ \sigma \circ Suc) \in$ *measurable* (*PiM* (*Suc* ‘ A) (*case-nat* M N)) (*PiM* A N)
⟨*proof*⟩

lemma *measurable-case-nat-Suc*:
$(\lambda\sigma.\ \sigma \circ Suc) \in$ *measurable* (*state-measure* (*Suc* ‘ A) (*case-nat* t Γ)) (*state-measure* A Γ)
⟨*proof*⟩

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 $\sigma \in (\Pi_E\ x \in V.$ *type-universe* (Γ 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*. ∫ +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*.
∫ +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*) ≠ {}
⟨*proof*⟩

lemma *sets-dens-ctxt-measure-eq*[*measurable-cong*]:
sets (*dens-ctxt-measure* (*V*,*V′*,Γ,δ) *ϱ*) = *sets* (*state-measure* (*V*∪*V′*) Γ)
⟨*proof*⟩

lemma *measurable-dens-ctxt-measure-eq*:
measurable (*dens-ctxt-measure* (*V*,*V′*,Γ,δ) *ϱ*) = *measurable* (*state-measure* (*V*∪*V′*) Γ)
⟨*proof*⟩

lemma *space-dens-ctxt-measure*:
space (*dens-ctxt-measure* (*V*,*V′*,Γ,δ) *ϱ*) = *space* (*state-measure* (*V*∪*V′*) Γ)
⟨*proof*⟩

definition *apply-dist-to-dens* :: *pdf-dist* ⇒ (*state* ⇒ *val* ⇒ *ennreal*) ⇒ (*state* ⇒ *val* ⇒ *ennreal*) **where**
apply-dist-to-dens dst f = (λϱ *y*. ∫⁺*x*. *f* ϱ *x* ∗ *dist-dens dst x y* ∂*stock-measure* (*dist-param-type dst*))

definition *remove-var* :: *state* ⇒ *state* **where**
remove-var σ = (λ*x*. σ (*Suc x*))

lemma *measurable-remove-var*[*measurable*]:
remove-var ∈ *measurable* (*state-measure* (*shift-var-set V*) (*case-nat t* Γ)) (*state-measure V* Γ)
⟨*proof*⟩

lemma *measurable-case-nat-undefined*[*measurable*]:
case-nat undefined ∈ *measurable* (*state-measure A* Γ) (*state-measure* (*Suc*‘*A*) (*case-nat t* Γ)) (**is** - ∈ *?M*)
⟨*proof*⟩

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*
⟨*proof*⟩

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)
⟨*proof*⟩

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*]:
    $\delta \in$ *borel-measurable* (*state-measure* $(V \cup V')$ Γ)
  and *disjoint*: $V \cap V' = \{\}$
begin

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

lemma *branch-prob-altdef*:
  assumes ϱ: $\varrho \in$ *space* (*state-measure* V' Γ)
  shows *branch-prob* $\mathcal{Y}$ $\varrho = \int^+ x.$ δ (*merge* V V' (x, ϱ)) ∂*state-measure* V Γ
⟨*proof*⟩

lemma *measurable-branch-prob*[*measurable*]:
  branch-prob $\mathcal{Y} \in$ *borel-measurable* (*state-measure* V' Γ)
⟨*proof*⟩

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* (Γ x))
⟨*proof*⟩

lemma *insert-Diff*: *insert* x $(A - B)$ = *insert* x A $-$ $(B - \{x\})$
  ⟨*proof*⟩

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* (Γ x) (Γ y)))
⟨*proof*⟩

lemma *measurable-marg-dens*:
  assumes $x \in V$ $\varrho \in$ *space* (*state-measure* V' Γ)
  shows *marg-dens* $\mathcal{Y}$ x $\varrho \in$ *borel-measurable* (*stock-measure* (Γ x))
  ⟨*proof*⟩

lemma *measurable-marg-dens2*:
  assumes $x \in V$ $y \in V$ $x \neq y$ $\varrho \in$ *space* (*state-measure* V' Γ)
  shows *marg-dens2* $\mathcal{Y}$ x y $\varrho \in$ *borel-measurable* (*stock-measure* (*PRODUCT* (Γ x) (Γ y)))
  ⟨*proof*⟩

lemma *measurable-state-measure-component*:
  $x \in V \Longrightarrow (\lambda\sigma.\ \sigma\ x) \in$ *measurable* (*state-measure* V Γ) (*stock-measure* (Γ x))
  ⟨*proof*⟩

lemma *measurable-dens-ctxt-measure-component*:
$x \in V \Longrightarrow (\lambda\sigma.\ \sigma\ x) \in$ *measurable* (*dens-ctxt-measure* (V,V',Γ,δ) ϱ) (*stock-measure* (Γ x))
⟨*proof*⟩

lemma *space-dens-ctxt-measure-dens-ctxt-measure'*:
assumes $x \in V$
shows *space* (*state-measure* V Γ) =
$\{\sigma(x := y) \mid \sigma\ y.\ \sigma \in$ *space* (*state-measure* ($V - \{x\}$) Γ) $\wedge\ y \in$ *type-universe* (Γ x)$\}$
⟨*proof*⟩

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* (Γ x))
⟨*proof*⟩

lemma *fun-upd-in-state-measure*:
$[\![\sigma \in$ *space* (*state-measure* A Γ); $y \in$ *space* (*stock-measure* (Γ x))$]\!]$
$\Longrightarrow \sigma(x := y) \in$ *space* (*state-measure* (*insert* x A) Γ)
⟨*proof*⟩

lemma *marg-dens-integral*:
fixes X :: *val set* **assumes** $x \in V$ **and** [*measurable*]: $X \in$ *sets* (*stock-measure* (Γ x))
assumes $\varrho \in$ *space* (*state-measure* V' Γ)
defines $X' \equiv (\lambda\sigma.\ \sigma\ x) -` X \cap$ *space* (*state-measure* V Γ)
shows $(\int^+\ y.$ *marg-dens* $\mathcal{Y}$ x ϱ $y * $ *indicator* X $y\ \partial$*stock-measure* (Γ x)) =
$(\int^+\sigma.\ \delta$ (*merge* V V' (σ,ϱ)) $*$ *indicator* X' $\sigma\ \partial$*state-measure* V Γ)
⟨*proof*⟩

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 $X' \equiv (\lambda\sigma.\ <|\sigma\ x, \sigma\ y|>) -` X \cap$ *space* (*state-measure* V Γ)
shows $(\int^+z.$ *marg-dens2* $\mathcal{Y}$ x y ϱ $z * $ *indicator* X $z\ \partial$*stock-measure* (*PRODUCT* (Γ x) (Γ y))) =
$(\int^+\sigma.\ \delta$ (*merge* V V' (σ,ϱ)) $*$ *indicator* X' $\sigma\ \partial$*state-measure* V Γ)
⟨*proof*⟩

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*⟩

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*⟩

lemma *measurable-insert-dens*[*measurable*]:
assumes *Mf*[*measurable*]: *case-prod* $f \in$ *borel-measurable* (*state-measure* ($V \cup V'$) $\Gamma \bigotimes_M$ *stock-measure* t)
shows *insert-dens* V V' f δ
$\in$ *borel-measurable* (*state-measure* (*shift-var-set* ($V \cup V'$)) (*case-nat* t Γ))
⟨*proof*⟩

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
⟨*proof*⟩

lemma *shift-var-set-Un*[*simp*]: *shift-var-set* $V \cup$ *Suc* ‘ V' = *shift-var-set* ($V \cup V'$)
⟨*proof*⟩

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*⟩

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*

$(V \cup V')\ \Gamma)$
⟨*proof*⟩

lemma *merge-Suc-aux*:
$\varrho \in$ *space* (*state-measure* (*Suc* ‘ V') (*case-nat* $t\ \Gamma$)) $\Longrightarrow$
(*λσ*. δ (*merge* $V\ V'$ (σ, $\varrho \circ$ *Suc*))) $\in$ *borel-measurable* (*state-measure* $V\ \Gamma$)
⟨*proof*⟩

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 ($\int^+ x$. $f\ x\ \partial distr\ (Pi_M\ (Suc`V)\ (case\text{-}nat\ M\ N))\ (Pi_M\ V\ N)\ (\lambda\sigma.\ \sigma \circ Suc)) =$
$(\int^+ x.\ f\ x\ \partial Pi_M\ V\ N)$
(**is** *nn-integral* (*?M1* V) - = -)
⟨*proof*⟩

lemma *PiM-Suc*:
assumes $\bigwedge i$. *sigma-finite-measure* ($N\ i$)
shows *distr* ($Pi_M\ (Suc`V)\ (case\text{-}nat\ M\ N))\ (Pi_M\ V\ N)\ (\lambda\sigma.\ \sigma \circ Suc) = Pi_M\ V\ N$ (**is** *?M1* = *?M2*)
⟨*proof*⟩

lemma *distr-state-measure-Suc*:
distr (*state-measure* (*Suc* ‘ V) (*case-nat* $t\ \Gamma$)) (*state-measure* $V\ \Gamma$) (*λσ*. $\sigma \circ$ *Suc*) =
state-measure $V\ \Gamma$ (**is** *?M1* = *?M2*)
⟨*proof*⟩

lemma *emeasure-dens-ctxt-measure-insert′*:
fixes $t\ f\ \varrho$
defines $M \equiv$ *dens-ctxt-measure* (*shift-var-set* V, *Suc`V′*, *case-nat* $t\ \Gamma$, *insert-dens* $V\ V'\ f\ \delta$) 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* $t\ \Gamma$))
assumes X: $X \in$ *sets* M
shows *emeasure* $M\ X = \int^+ \sigma.\ \delta$ (*merge* $V\ V'$ (σ, $\varrho \circ$ *Suc*)) $*$ $\int^+ y.\ f$ (*merge* $V\ V'$ (σ, $\varrho \circ$ *Suc*)) y $*$
indicator X (*merge* (*shift-var-set* V) (*Suc`V′*) (*case-nat* $y\ \sigma$, ϱ))
∂*stock-measure* $t\ \partial$*state-measure* $V\ \Gamma$ (**is** - = *?I*)
⟨*proof*⟩

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* $t\ \Gamma$) (*insert-dens* $V\ V'\ f\ \delta$)

(**is** *density-context ?V ?V′ ?Γ′ ?δ′*)
⟨*proof*⟩

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* ⨂$_M$ *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*⟩

lemma *density-context-if-dens*:
assumes *has-parametrized-subprob-density* (*state-measure* (*V* ∪ *V′*) *Γ*) *M* (*count-space* (*range BoolVal*)) *f*
shows *density-context V V′ Γ* (*if-dens δ f b*)
⟨*proof*⟩

lemma *density-context-if-dens-det*:
assumes *e*: *Γ* ⊢ *e* : *BOOL randomfree e free-vars e* ⊆ *V* ∪ *V′*
shows *density-context V V′ Γ* (*if-dens-det δ e b*)
⟨*proof*⟩

lemma *density-context-empty*[*simp*]: *density-context* {} (*V*∪*V′*) *Γ* (λ-. *1*)
⟨*proof*⟩

lemma *dens-ctxt-measure-bind-const*:
assumes *ϱ* ∈ *space* (*state-measure V′ Γ*) *subprob-space N*
shows *dens-ctxt-measure* 𝒴 *ϱ* ⤜ (λ-. *N*) = *density N* (λ-. *branch-prob* 𝒴 *ϱ*) (**is** *?M1* = *?M2*)
⟨*proof*⟩

lemma *nn-integral-dens-ctxt-measure-restrict*:
assumes *ϱ* ∈ *space* (*state-measure V′ Γ*) *f ϱ* ≥ *0*
assumes *f* ∈ *borel-measurable* (*state-measure V′ Γ*)
shows (∫⁺*x*. *f* (*restrict x V′*) ∂*dens-ctxt-measure* 𝒴 *ϱ*) = *branch-prob* 𝒴 *ϱ* ∗ *f ϱ*

⟨*proof*⟩

lemma *expr-sem-op-eq-distr*:
assumes $\Gamma \vdash$ *oper* \$\$ *e* : *t′ free-vars e* $\subseteq V \cup V'$ $\varrho \in$ *space* (*state-measure V′* Γ)
defines $M \equiv$ *dens-ctxt-measure* (V,V',Γ,δ) ϱ
shows $M \gg\!= (\lambda\sigma.$ *expr-sem* σ (*oper* \$\$ *e*)) =
distr ($M \gg\!= (\lambda\sigma.$ *expr-sem* σ *e*)) (*stock-measure t′*) (*op-sem oper*)
⟨*proof*⟩

end

lemma *density-context-equiv*:
assumes $\bigwedge\sigma.\ \sigma \in$ *space* (*state-measure* ($V \cup V'$) Γ) $\Longrightarrow \delta\ \sigma = \delta'\ \sigma$
assumes [*simp*, *measurable*]: $\delta' \in$ *borel-measurable* (*state-measure* ($V \cup V'$) Γ)
assumes *density-context V V′* Γ δ
shows *density-context V V′* Γ δ'
⟨*proof*⟩

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',Γ,δ) $\vdash_d$ *e* ⇒ *f*; $\Gamma \vdash$ *e* : *t*;
$\bigwedge\varrho.\ \varrho \in$ *space* (*state-measure V′* Γ) $\Longrightarrow$
AE x in stock-measure t. f ϱ *x* = *f′* ϱ *x*;
case-prod f′ ∈ *borel-measurable* (*state-measure V′* $\Gamma \bigotimes_M$ *stock-measure t*)⟧
$\Longrightarrow$ (V,V',Γ,δ) $\vdash_d$ *e* ⇒ *f′*
| *hd-dens-ctxt-cong*:
(V,V',Γ,δ) $\vdash_d$ *e* ⇒ *f* $\Longrightarrow$ ($\bigwedge\sigma.\ \sigma \in$ *space* (*state-measure* ($V \cup V'$) Γ) $\Longrightarrow \delta\ \sigma = \delta'\ \sigma$)
$\Longrightarrow$ (V,V',Γ,δ') $\vdash_d$ *e* ⇒ *f*
| *hd-val*: *countable-type* (*val-type v*) $\Longrightarrow$
(V,V',Γ,δ) $\vdash_d$ *Val v* ⇒ ($\lambda\varrho$ *x. branch-prob* (V,V',Γ,δ) ϱ ∗ *indicator* {*v*} *x*)

| *hd-var*: $x \in V \Longrightarrow (V,V',\Gamma,\delta) \vdash_d Var\ x \Rightarrow marg\text{-}dens\ (V,V',\Gamma,\delta)\ x$
| *hd-let*: $[\![(\{\},V \cup V',\Gamma,\lambda\text{-}.\ 1) \vdash_d e1 \Rightarrow f;$
$(shift\text{-}var\text{-}set\ V, Suc`V', the(expr\text{-}type\ \Gamma\ e1) \cdot \Gamma, insert\text{-}dens\ V\ V'\ f\ \delta) \vdash_d e2 \Rightarrow g]\!]$
$\Longrightarrow (V,V',\Gamma,\delta) \vdash_d LetVar\ e1\ e2 \Rightarrow (\lambda\varrho.\ g\ (case\text{-}nat\ undefined\ \varrho))$
| *hd-rand*: $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f \Longrightarrow (V,V',\Gamma,\delta) \vdash_d Random\ dst\ e \Rightarrow apply\text{-}dist\text{-}to\text{-}dens\ dst\ f$
| *hd-rand-det*: $randomfree\ e \Longrightarrow free\text{-}vars\ e \subseteq V' \Longrightarrow$
$(V,V',\Gamma,\delta) \vdash_d Random\ dst\ e \Rightarrow$
$(\lambda\varrho\ x.\ branch\text{-}prob\ (V,V',\Gamma,\delta)\ \varrho * dist\text{-}dens\ dst\ (expr\text{-}sem\text{-}rf\ \varrho\ e)\ x)$
| *hd-fail*: $(V,V',\Gamma,\delta) \vdash_d Fail\ t \Rightarrow (\lambda\text{-}\ \text{-}.\ 0)$
| *hd-pair*: $x \in V \Longrightarrow y \in V \Longrightarrow x \neq y \Longrightarrow (V,V',\Gamma,\delta) \vdash_d <Var\ x,\ Var\ y> \Rightarrow marg\text{-}dens2\ (V,V',\Gamma,\delta)\ x\ y$
| *hd-if*: $[\![(\{\},V \cup V',\Gamma,\lambda\text{-}.\ 1) \vdash_d b \Rightarrow f;$
$(V,V',\Gamma,if\text{-}dens\ \delta\ f\ True) \vdash_d e1 \Rightarrow g1;\ (V,V',\Gamma,if\text{-}dens\ \delta\ f\ False) \vdash_d e2 \Rightarrow g2]\!]$
$\Longrightarrow (V,V',\Gamma,\delta) \vdash_d IF\ b\ THEN\ e1\ ELSE\ e2 \Rightarrow (\lambda\varrho\ x.\ g1\ \varrho\ x + g2\ \varrho\ x)$
| *hd-if-det*: $[\![randomfree\ b;\ (V,V',\Gamma,if\text{-}dens\text{-}det\ \delta\ b\ True) \vdash_d e1 \Rightarrow g1;$
$(V,V',\Gamma,if\text{-}dens\text{-}det\ \delta\ b\ False) \vdash_d e2 \Rightarrow g2]\!]$
$\Longrightarrow (V,V',\Gamma,\delta) \vdash_d IF\ b\ THEN\ e1\ ELSE\ e2 \Rightarrow (\lambda\varrho\ x.\ g1\ \varrho\ x + g2\ \varrho\ x)$
| *hd-fst*: $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f \Longrightarrow$
$(V,V',\Gamma,\delta) \vdash_d Fst\ \$\$\ e \Rightarrow$
$(\lambda\varrho\ x.\ \int^+ y.\ f\ \varrho <|x,y|> \partial stock\text{-}measure\ (the\ (expr\text{-}type\ \Gamma\ (Snd\ \$\$\ e))))$
| *hd-snd*: $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f \Longrightarrow$
$(V,V',\Gamma,\delta) \vdash_d Snd\ \$\$\ e \Rightarrow$
$(\lambda\varrho\ y.\ \int^+ x.\ f\ \varrho <|x,y|> \partial stock\text{-}measure\ (the\ (expr\text{-}type\ \Gamma\ (Fst\ \$\$\ e))))$
| *hd-op-discr*: $countable\text{-}type\ (the\ (expr\text{-}type\ \Gamma\ (oper\ \$\$\ e))) \Longrightarrow (V,V',\Gamma,\delta) \vdash_d e \Rightarrow f \Longrightarrow$
$(V,V',\Gamma,\delta) \vdash_d oper\ \$\$\ e \Rightarrow (\lambda\varrho\ y.\ \int^+ x.\ (if\ op\text{-}sem\ oper\ x = y\ then\ 1\ else\ 0) * f\ \varrho\ x$
$\partial stock\text{-}measure\ (the\ (expr\text{-}type\ \Gamma\ e)))$
| *hd-neg*: $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f \Longrightarrow$
$(V,V',\Gamma,\delta) \vdash_d Minus\ \$\$\ e \Rightarrow (\lambda\sigma\ x.\ f\ \sigma\ (op\text{-}sem\ Minus\ x))$
| *hd-addc*: $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f \Longrightarrow randomfree\ e' \Longrightarrow free\text{-}vars\ e' \subseteq V' \Longrightarrow$
$(V,V',\Gamma,\delta) \vdash_d Add\ \$\$\ <e,\ e'> \Rightarrow$
$(\lambda\varrho\ x.\ f\ \varrho\ (op\text{-}sem\ Add\ <|x,\ expr\text{-}sem\text{-}rf\ \varrho\ (Minus\ \$\$\ e')|>))$
| *hd-multc*: $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f \Longrightarrow val\text{-}type\ c = REAL \Longrightarrow c \neq RealVal\ 0 \Longrightarrow$
$(V,V',\Gamma,\delta) \vdash_d Mult\ \$\$\ <e,\ Val\ c> \Rightarrow$
$(\lambda\varrho\ x.\ f\ \varrho\ (op\text{-}sem\ Mult\ <|x,\ op\text{-}sem\ Inverse\ c|>) *$
$inverse\ (abs\ (extract\text{-}real\ c)))$
| *hd-exp*: $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f \Longrightarrow$
$(V,V',\Gamma,\delta) \vdash_d Exp\ \$\$\ e \Rightarrow$
$(\lambda\sigma\ x.\ if\ extract\text{-}real\ x > 0\ then$
$f\ \sigma\ (lift\text{-}RealVal\ safe\text{-}ln\ x) * inverse\ (extract\text{-}real\ x)\ else\ 0)$
| *hd-inv*: $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f \Longrightarrow$
$(V,V',\Gamma,\delta) \vdash_d Inverse\ \$\$\ e \Rightarrow (\lambda\sigma\ x.\ f\ \sigma\ (op\text{-}sem\ Inverse\ x) *$

inverse (*extract-real x*) ^ *2*)
| *hd-add*: (*V*, *V'*,Γ,δ) ⊢$_d$ *e* ⇒ *f* ⟹
(*V*, *V'*,Γ,δ) ⊢$_d$ *Add* $$ *e* ⇒ (λσ *z*. ∫$^+$*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*. ∫$^+$*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*⟩

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*⟩

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*⟩

lemma (**in** *density-context*) *bind-dens-ctxt-measure-cong*:
assumes *fg*: ⋀σ. (⋀*x*. *x* ∈ *V'* ⟹ σ *x* = ϱ *x*) ⟹ *f* σ = *g* σ
assumes ϱ[*measurable*]: ϱ ∈ *space* (*state-measure V'* Γ)
assumes *Mf*[*measurable*]: *f* ∈ *measurable* (*state-measure* (*V* ∪ *V'*) Γ) (*subprob-algebra N*)
assumes *Mg*[*measurable*]: *g* ∈ *measurable* (*state-measure* (*V* ∪ *V'*) Γ) (*subprob-algebra N*)
defines *M* ≡ *dens-ctxt-measure* (*V*, *V'*,Γ,δ) ϱ
shows *M* ⤜ *f* = *M* ⤜ *g*
⟨*proof*⟩

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 $v \equiv$ *expr-sem-rf* ϱ *e'*
shows $M \gg\!= (\lambda\sigma.$ *expr-sem* σ (*oper* $$ <*e*, *e'*>)) =
distr ($M \gg\!= (\lambda\sigma.$ *expr-sem* σ *e*)) (*stock-measure tr*) ($\lambda w.$ *op-sem oper* <|*w*,*v*|>)
⟨*proof*⟩

lemma *addc-density-measurable*:
assumes *Mf*: *case-prod f* $\in$ *borel-measurable* (*state-measure V' Γ* $\bigotimes_M$ *stock-measure t*)
assumes *t-disj*: $t = REAL \vee t = INTEG$ **and** *t*: $\Gamma \vdash e' : t$
assumes *rf*: *randomfree e'* **and** *vars*: *free-vars e'* $\subseteq V'$
defines $f' \equiv (\lambda\varrho\ x.\ f\ \varrho$ (*op-sem Add* <|*x*, *expr-sem-rf* ϱ (*Minus* $$ *e'*)|>))
shows *case-prod f'* $\in$ *borel-measurable* (*state-measure V' Γ* $\bigotimes_M$ *stock-measure t*)
⟨*proof*⟩

lemma (**in** *density-context*) *emeasure-bind-if-dens-ctxt-measure*:
assumes *ϱ*: $\varrho \in$ *space* (*state-measure V' Γ*)
defines $M \equiv$ *dens-ctxt-measure* $\mathcal{Y}$ ϱ
assumes *Mf*[*measurable*]: $f \in$ *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',Γ,$\lambda\sigma.\ \delta\ \sigma * \delta f\ \sigma$ (*BoolVal True*)) $\varrho \gg\!= g$) *R* δg
assumes *densh*: *has-parametrized-subprob-density* (*state-measure V' Γ*) ($\lambda\varrho.$ *dens-ctxt-measure* (V,V',Γ,$\lambda\sigma.\ \delta\ \sigma * \delta f\ \sigma$ (*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*⟩

lemma (**in** *density-context*) *emeasure-bind-if-det-dens-ctxt-measure*:
fixes *f*
assumes *ϱ*: $\varrho \in$ *space* (*state-measure V' Γ*)
defines $M \equiv$ *dens-ctxt-measure* $\mathcal{Y}$ ϱ

defines $P \equiv \lambda b.\ f\ b = \mathit{BoolVal\ True}$ **and** $P' \equiv \lambda b.\ f\ b = \mathit{BoolVal\ False}$
assumes *dc1*: *density-context* $V\ V'\ \Gamma\ (\lambda\sigma.\ \delta\ \sigma * (\mathit{if}\ P\ \sigma\ \mathit{then\ 1\ else\ 0}))$
assumes *dc2*: *density-context* $V\ V'\ \Gamma\ (\lambda\sigma.\ \delta\ \sigma * (\mathit{if}\ P'\ \sigma\ \mathit{then\ 1\ else\ 0}))$
assumes *Mf*[*measurable*]: $f \in \mathit{measurable}\ M\ (\mathit{stock\text{-}measure\ BOOL})$
assumes *Mg*[*measurable*]: $g \in \mathit{measurable}\ M\ (\mathit{subprob\text{-}algebra}\ R)$
assumes *Mh*[*measurable*]: $h \in \mathit{measurable}\ M\ (\mathit{subprob\text{-}algebra}\ R)$
assumes *densg*: *has-parametrized-subprob-density* (*state-measure* $V'\ \Gamma$) ($\lambda\varrho.$ *dens-ctxt-measure* $(V,V',\Gamma,\lambda\sigma.\ \delta\ \sigma * (\mathit{if}\ P\ \sigma\ \mathit{then\ 1\ else\ 0}))$ $\varrho \gg\!= g)\ R\ \delta g$
assumes *densh*: *has-parametrized-subprob-density* (*state-measure* $V'\ \Gamma$) ($\lambda\varrho.$ *dens-ctxt-measure* $(V,V',\Gamma,\lambda\sigma.\ \delta\ \sigma * (\mathit{if}\ P'\ \sigma\ \mathit{then\ 1\ else\ 0}))$ $\varrho \gg\!= h)\ R\ \delta h$
shows $M \gg\!= (\lambda x.\ \mathit{if}\ P\ x\ \mathit{then}\ g\ x\ \mathit{else}\ h\ x) = \mathit{density}\ R\ (\lambda x.\ \delta g\ \varrho\ x + \delta h\ \varrho\ x)$
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

7.3 Soundness proof

lemma *restrict-state-measure*[*measurable*]:
$(\lambda x.\ \mathit{restrict}\ x\ V') \in \mathit{measurable}\ (\mathit{state\text{-}measure}\ (V \cup V')\ \Gamma)\ (\mathit{state\text{-}measure}\ V'\ \Gamma)$
⟨*proof*⟩

lemma *expr-has-density-sound-op*:
assumes *dens-ctxt*: *density-context* $V\ V'\ \Gamma\ \delta$
assumes *dens*: *has-parametrized-subprob-density* (*state-measure* $V'\ \Gamma$) ($\lambda\varrho.$ *dens-ctxt-measure* $(V,V',\Gamma,\delta)\ \varrho \gg\!= (\lambda\sigma.\ \mathit{expr\text{-}sem}\ \sigma\ e)$) (*stock-measure* t) f
assumes *Mg*: *case-prod* $g \in$ *borel-measurable* (*state-measure* $V'\ \Gamma \bigotimes_M$ *stock-measure* t')
assumes *dens'*: $\bigwedge M\ \varrho.$ *has-subprob-density* M (*stock-measure* t) $(f\ \varrho) \Longrightarrow$ *has-density* (*distr* M (*stock-measure* t') (*op-sem oper*)) (*stock-measure* t') $(g\ \varrho)$
assumes *t1*: $\Gamma \vdash e : t$ **and** *t2* : *op-type oper* t = *Some* t'
assumes *free-vars*: *free-vars* (*oper* $ e) $\subseteq V \cup V'$
shows *has-parametrized-subprob-density* (*state-measure* $V'\ \Gamma$) ($\lambda\varrho.$ *dens-ctxt-measure* $(V,V',\Gamma,\delta)\ \varrho \gg\!= (\lambda\sigma.\ \mathit{expr\text{-}sem}\ \sigma\ (\mathit{oper}\ \$\$\ e))$) (*stock-measure* t') g
⟨*proof*⟩

lemma *expr-has-density-sound-aux*:
assumes $(V,V',\Gamma,\delta) \vdash_d e \Rightarrow f\ \Gamma \vdash e : t$
density-context $V\ V'\ \Gamma\ \delta$ *free-vars* $e \subseteq V \cup V'$
shows *has-parametrized-subprob-density* (*state-measure* $V'\ \Gamma$) ($\lambda\varrho.$ *do* $\{\sigma \leftarrow$ *dens-ctxt-measure* $(V,V',\Gamma,\delta)\ \varrho;$ *expr-sem* $\sigma\ e\}$) (*stock-measure* t) ($\lambda\varrho\ x.\ f\ \varrho\ x$)
⟨*proof*⟩

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-measure* $V'\ \Gamma) \Longrightarrow x \in$ *space* (*stock-measure* t) $\Longrightarrow f\ \varrho\ x = f'\ \varrho\ x$
shows $(V, V', \Gamma, \delta) \vdash_d e \Rightarrow f'$
⟨*proof*⟩

lemma *prob-space-empty-dens-ctxt*[*simp*]:
prob-space (*dens-ctxt-measure* ({},{},Γ,(λ-. *1*)) (λ-. *undefined*))
⟨*proof*⟩

lemma *branch-prob-empty-ctxt*[*simp*]: *branch-prob* ({},{},Γ,(λ-. *1*)) (λ-. *undefined*) = *1*
⟨*proof*⟩

lemma *expr-has-density-sound*:
assumes ({},{},Γ,(λ-. *1*)) $\vdash_d e \Rightarrow f\ \Gamma \vdash e : t$ *free-vars* $e = \{\}$
shows *has-subprob-density* (*expr-sem* σ e) (*stock-measure* t) (f (λ-. *undefined*))
⟨*proof*⟩

end

8 Target Language Syntax and Semantics

theory *PDF-Target-Semantics*
imports *PDF-Semantics*
begin

datatype *cexpr* =
CVar vname
| *CVal val*
| *CPair cexpr cexpr* (<-, ->$_c$ [*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* ⇒ *cexpr*) ⇒ *cexpr* (**binder** λ_c *10*) **where**
cexpr-fun f ≡ *f* (*CVar 0*)
abbreviation *cexpr-Add* (**infixl** $+_c$ *65*) **where**
cexpr-Add a b ≡ *Add* $$$_c$ <*a*, *b*>$_c$
abbreviation *cexpr-Minus* ($-_c$ - [*81*] *80*) **where**
cexpr-Minus a ≡ *Minus* $$$_c$ *a*
abbreviation *cexpr-Sub* (**infixl** $-_c$ *65*) **where**
cexpr-Sub a b ≡ *a* $+_c$ $-_c$*b*
abbreviation *cexpr-Mult* (**infixl** $*_c$ *70*) **where**
cexpr-Mult a b ≡ *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** $\widehat{}_c$ *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**
  $\pi_c \equiv Pi\ \$\$_c\ (CVal\ UnitVal)$

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* ∂*t*) = *Suc* −‘ *free-vars-cexpr e*

instance ⟨*proof*⟩
end

inductive *cexpr-typing* :: *tyenv* ⇒ *cexpr* ⇒ *pdf-type* ⇒ *bool* ((*1-*/ $\vdash_c$/ (- :/ -))
[*50*,*0*,*50*] *50*) **where**

cet-val: $\Gamma \vdash_c \mathit{CVal}\ v$: *val-type* v
| *cet-var*: $\Gamma \vdash_c \mathit{CVar}\ x : \Gamma\ x$
| *cet-pair*: $\Gamma \vdash_c e1 : t1 \Longrightarrow \Gamma \vdash_c e2 : t2 \Longrightarrow \Gamma \vdash_c <e1, e2>_c : \mathit{PRODUCT}\ t1\ t2$
| *cet-op*: $\Gamma \vdash_c e : t \Longrightarrow$ *op-type oper* $t = \mathit{Some}\ t' \Longrightarrow \Gamma \vdash_c \mathit{oper}\ \$\$_c\ e : t'$
| *cet-if*: $\Gamma \vdash_c b : \mathit{BOOL} \Longrightarrow \Gamma \vdash_c e1 : t \Longrightarrow \Gamma \vdash_c e2 : t$
$\Longrightarrow \Gamma \vdash_c \mathit{IF}_c\ b\ \mathit{THEN}\ e1\ \mathit{ELSE}\ e2 : t$
| *cet-int*: $t \cdot \Gamma \vdash_c e : \mathit{REAL} \Longrightarrow \Gamma \vdash_c \int_c e\ \partial t : \mathit{REAL}$

lemma *cet-val'*: $t =$ *val-type* $v \Longrightarrow \Gamma \vdash_c \mathit{CVal}\ v : t$
⟨*proof*⟩

lemma *cet-var'*: $t = \Gamma\ x \Longrightarrow \Gamma \vdash_c \mathit{CVar}\ x : t$
⟨*proof*⟩

lemma *cet-not*: $\Gamma \vdash_c e : \mathit{BOOL} \Longrightarrow \Gamma \vdash_c \neg_c\ e : \mathit{BOOL}$
⟨*proof*⟩

lemma *cet-and*: $\Gamma \vdash_c e1 : \mathit{BOOL} \Longrightarrow \Gamma \vdash_c e2 : \mathit{BOOL} \Longrightarrow \Gamma \vdash_c e1 \wedge_c e2 : \mathit{BOOL}$
and
cet-or: $\Gamma \vdash_c e1 : \mathit{BOOL} \Longrightarrow \Gamma \vdash_c e2 : \mathit{BOOL} \Longrightarrow \Gamma \vdash_c e1 \vee_c e2 : \mathit{BOOL}$
⟨*proof*⟩

lemma *cet-minus-real*: $\Gamma \vdash_c e : \mathit{REAL} \Longrightarrow \Gamma \vdash_c -_c e : \mathit{REAL}$ **and**
cet-inverse: $\Gamma \vdash_c e : \mathit{REAL} \Longrightarrow \Gamma \vdash_c \mathit{inverse}_c\ e : \mathit{REAL}$ **and**
cet-sqrt: $\Gamma \vdash_c e : \mathit{REAL} \Longrightarrow \Gamma \vdash_c \mathit{sqrt}_c\ e : \mathit{REAL}$ **and**
cet-exp: $\Gamma \vdash_c e : \mathit{REAL} \Longrightarrow \Gamma \vdash_c \mathit{exp}_c\ e : \mathit{REAL}$ **and**
cet-ln: $\Gamma \vdash_c e : \mathit{REAL} \Longrightarrow \Gamma \vdash_c \mathit{ln}_c\ e : \mathit{REAL}$
⟨*proof*⟩

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

lemma *cet-add-real*: $\Gamma \vdash_c e1 : \mathit{REAL} \Longrightarrow \Gamma \vdash_c e2 : \mathit{REAL} \Longrightarrow \Gamma \vdash_c e1 +_c e2 : \mathit{REAL}$ **and**
cet-mult-real: $\Gamma \vdash_c e1 : \mathit{REAL} \Longrightarrow \Gamma \vdash_c e2 : \mathit{REAL} \Longrightarrow \Gamma \vdash_c e1 *_c e2 : \mathit{REAL}$ **and**
cet-less-real: $\Gamma \vdash_c e1 : \mathit{REAL} \Longrightarrow \Gamma \vdash_c e2 : \mathit{REAL} \Longrightarrow \Gamma \vdash_c e1 <_c e2 : \mathit{BOOL}$
⟨*proof*⟩

lemma *cet-eq*: $\Gamma \vdash_c e1 : t \Longrightarrow \Gamma \vdash_c e2 : t \Longrightarrow \Gamma \vdash_c e1 =_c e2 : \mathit{BOOL}$
⟨*proof*⟩

lemma *cet-less-eq-real*: $\Gamma \vdash_c e1 : \mathit{REAL} \Longrightarrow \Gamma \vdash_c e2 : \mathit{REAL} \Longrightarrow \Gamma \vdash_c e1 \leq_c e2 : \mathit{BOOL}$
⟨*proof*⟩

lemma *cet-minus-int*: $\Gamma \vdash_c e : \mathit{INTEG} \Longrightarrow \Gamma \vdash_c -_c e : \mathit{INTEG}$

⟨*proof*⟩

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: $\Gamma \vdash_c e1 : INTEG \Longrightarrow \Gamma \vdash_c e2 : INTEG \Longrightarrow \Gamma \vdash_c e1 <_c e2 : BOOL$
⟨*proof*⟩

lemma *cet-less-eq-int*: $\Gamma \vdash_c e1 : INTEG \Longrightarrow \Gamma \vdash_c e2 : INTEG \Longrightarrow \Gamma \vdash_c e1 \leq_c e2 : BOOL$
⟨*proof*⟩

lemma *cet-sub-int*: $\Gamma \vdash_c e1 : INTEG \Longrightarrow \Gamma \vdash_c e2 : INTEG \Longrightarrow \Gamma \vdash_c e1 -_c e2 : INTEG$
⟨*proof*⟩

lemma *cet-fst*: $\Gamma \vdash_c e : PRODUCT\ t\ t' \Longrightarrow \Gamma \vdash_c fst_c\ e : t$ **and**
cet-snd: $\Gamma \vdash_c e : PRODUCT\ t\ t' \Longrightarrow \Gamma \vdash_c snd_c\ e : t'$
⟨*proof*⟩

lemma *cet-cast-real*: $\Gamma \vdash_c e : BOOL \Longrightarrow \Gamma \vdash_c \langle e \rangle_c : REAL$
⟨*proof*⟩

lemma *cet-cast-real-int*: $\Gamma \vdash_c e : INTEG \Longrightarrow \Gamma \vdash_c \langle e \rangle_c : REAL$
⟨*proof*⟩

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

lemma *cet-pi*: $\Gamma \vdash_c \pi_c : REAL$
⟨*proof*⟩

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 (∫$_c$ *e* ∂*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* ⟷ Γ ⊢$_c$ *e* : *t*
⟨*proof*⟩

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

lemma *cexpr-typing-cong′*:
assumes Γ ⊢$_c$ *e* : *t* ⋀*x*. *x* ∈ *free-vars e* ⟹ Γ *x* = Γ′ *x*
shows Γ′ ⊢$_c$ *e* : *t*
⟨*proof*⟩

lemma *cexpr-typing-cong*:
assumes ⋀*x*. *x* ∈ *free-vars e* ⟹ Γ *x* = Γ′ *x*
shows Γ ⊢$_c$ *e* : *t* ⟷ Γ′ ⊢$_c$ *e* : *t*
⟨*proof*⟩

primrec *cexpr-sem* :: *state* ⇒ *cexpr* ⇒ *val* **where**
cexpr-sem σ (*CVal v*) = *v*
| *cexpr-sem* σ (*CVar x*) = σ *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 (∫$_c$ *e* ∂*t*) = *RealVal* (∫ *x*. *extract-real* (*cexpr-sem* (*x* · σ) *e*) ∂(*stock-measure t*))

definition *cexpr-equiv* :: *cexpr* ⇒ *cexpr* ⇒ *bool* **where**
cexpr-equiv e1 e2 ≡ ∀σ. *cexpr-sem* σ *e1* = *cexpr-sem* σ *e2*

lemma *cexpr-equiv-commute*: *cexpr-equiv e1 e2* ⟷ *cexpr-equiv e2 e1*
⟨*proof*⟩

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) $= t$
⟨*proof*⟩

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
⟨*proof*⟩

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*]:
$\Gamma \vdash_c e : t \Longrightarrow$ *free-vars* $e \subseteq V \Longrightarrow$
$(\lambda\sigma.$ *cexpr-sem* σ $e) \in$ *measurable* (*state-measure* V Γ) (*stock-measure* t)
⟨*proof*⟩

lemma *measurable-eval-cexpr*[*measurable*]:
assumes *case-nat* t $\Gamma \vdash_c e : REAL$
assumes *free-vars* $e \subseteq$ *shift-var-set* V
shows *case-prod* (*eval-cexpr* e) $\in$ *borel-measurable* (*state-measure* V $\Gamma \bigotimes_M$ *stock-measure* t)
⟨*proof*⟩

lemma *cexpr-sem-Add*:
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 $e2$)
⟨*proof*⟩

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 $e2$)
⟨*proof*⟩

8.1 General functions on Expressions

Transform variable names in an expression.

primrec *map-vars* :: (*vname* ⇒ *vname*) ⇒ *cexpr* ⇒ *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* (λ*x*. *Suc* (*f x*))) *e* ∂*t*

lemma *free-vars-map-vars*[*simp*]:
free-vars (*map-vars f e*) = *f* ‘ *free-vars e*
⟨*proof*⟩

lemma *cexpr-typing-map-vars*:
Γ ∘ *f* ⊢$_c$ *e* : *t* ⟹ Γ ⊢$_c$ *map-vars f e* : *t*
⟨*proof*⟩

lemma *cexpr-sem-map-vars*:
cexpr-sem σ (*map-vars f e*) = *cexpr-sem* (*σ* ∘ *f*) *e*
⟨*proof*⟩

definition *insert-var* :: *vname* ⇒ (*vname* ⇒ *'a*) ⇒ *'a* ⇒ *vname* ⇒ *'a* **where**
insert-var v f x w ≡ *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*
⟨*proof*⟩

Substitutes expression e for variable x in e'.

primrec *cexpr-subst* :: *vname* ⇒ *cexpr* ⇒ *cexpr* ⇒ *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*⟩

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'*
⟨*proof*⟩

lemma *cexpr-typing-subst-aux*:
assumes *insert-var x* Γ *t* ⊢$_c$ *e'* : *t'* Γ ⊢$_c$ *e* : *t*
shows Γ ⊢$_c$ *cexpr-subst x e e'* : *t'*
⟨*proof*⟩

lemma *cexpr-typing-subst*[*intro*]:
assumes Γ ⊢$_c$ *e* : *t case-nat t* Γ ⊢$_c$ *e'* : *t'*

shows $\Gamma \vdash_c$ *cexpr-subst 0 e e'* : *t'*
⟨*proof*⟩

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*⟩

lemma *free-vars-cexpr-subst*:
free-vars (*cexpr-subst 0 e e'*) ⊆ *Suc* −‘ *free-vars e'* ∪ *free-vars e*
⟨*proof*⟩

primrec *cexpr-comp-aux* :: *vname* ⇒ *cexpr* ⇒ *cexpr* ⇒ *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* (∫$_c$ *e' ∂t*) = (∫$_c$ *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*⟩

definition *cexpr-comp* (**infixl** ∘$_c$ *55*) **where**
cexpr-comp b a ≡ *cexpr-comp-aux 0 a b*

lemma *cexpr-typing-cexpr-comp-aux*:
assumes Γ(*x* := *t1*) ⊢$_c$ *e'* : *t2* Γ ⊢$_c$ *e* : *t1*
shows Γ ⊢$_c$ *cexpr-comp-aux x e e'* : *t2*
⟨*proof*⟩

lemma *cexpr-typing-cexpr-comp*[*intro*]:
assumes *case-nat t1* Γ ⊢$_c$ *g* : *t2*
assumes *case-nat t2* Γ ⊢$_c$ *f* : *t3*
shows *case-nat t1* Γ ⊢$_c$ *f* ∘$_c$ *g* : *t3*
⟨*proof*⟩

lemma *free-vars-cexpr-comp-aux*:
free-vars (*cexpr-comp-aux x e e'*) ⊆ (*free-vars e'* − {*x*}) ∪ *free-vars e*

⟨*proof*⟩

lemma *free-vars-cexpr-comp*:
free-vars (*cexpr-comp e e'*) ⊆ (*free-vars e* − {*0*}) ∪ *free-vars e'*
⟨*proof*⟩

lemma *free-vars-cexpr-comp'*:
free-vars (*cexpr-comp e e'*) ⊆ *free-vars e* ∪ *free-vars e'*
⟨*proof*⟩

lemma *cexpr-sem-cexpr-comp*:
cexpr-sem σ (*f* ∘$_c$ *g*) = *cexpr-sem* (*σ*(*0* := *cexpr-sem σ g*)) *f*
⟨*proof*⟩

lemma *eval-cexpr-comp*:
eval-cexpr (*f* ∘$_c$ *g*) *σ x* = *eval-cexpr f σ* (*cexpr-sem* (*case-nat x σ*) *g*)
⟨*proof*⟩

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* (∫$_c$ *e ∂t*) *v* = ∫$_c$ *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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *cexpr-typing-subst-val*[*intro*]:
case-nat t Γ ⊢$_c$ *e* : *t'* ⟹ *val-type v* = *t* ⟹ Γ ⊢$_c$ *cexpr-subst-val e v* : *t'*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *free-vars-cexpr-subst-val*[*simp*]:
free-vars (*cexpr-subst-val e v*) = *Suc* −‘ *free-vars e*
⟨*proof*⟩

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*:
(⋀σ. σ ∈ *space* (*state-measure V* Γ) ⟹ *extract-real* (*cexpr-sem* σ *e*) ≥ *0*) ⟹ *nonneg-cexpr V* Γ *e*
⟨*proof*⟩

lemma *nonneg-cexprD*:
nonneg-cexpr V Γ *e* ⟹ σ ∈ *space* (*state-measure V* Γ) ⟹ *extract-real* (*cexpr-sem* σ *e*) ≥ *0*
⟨*proof*⟩

lemma *nonneg-cexpr-map-vars*:
assumes *nonneg-cexpr* (*f* −‘ *V*) (Γ ∘ *f*) *e*
shows *nonneg-cexpr V* Γ (*map-vars f e*)
⟨*proof*⟩

lemma *nonneg-cexpr-subset*:
assumes *nonneg-cexpr V* Γ *e V* ⊆ *V' free-vars e* ⊆ *V*
shows *nonneg-cexpr V'* Γ *e*
⟨*proof*⟩

lemma *nonneg-cexpr-Mult*:
assumes Γ ⊢$_c$ *e1* : *REAL* Γ ⊢$_c$ *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*⟩

lemma *nonneg-indicator*:
assumes Γ ⊢$_c$ *e* : *BOOL free-vars e* ⊆ *V*
shows *nonneg-cexpr V* Γ (⟨*e*⟩$_c$)
⟨*proof*⟩

lemma *nonneg-cexpr-comp-aux*:
assumes *nonneg*: *nonneg-cexpr V* (Γ(*x* := *t1*)) *e* **and** *x*:*x* ∈ *V*
assumes *t2*: Γ(*x*:=*t1*) ⊢$_c$ *e* : *t2* **and** *t1*: Γ ⊢$_c$ *f* : *t1* **and** *vars*: *free-vars f* ⊆ *V*
shows *nonneg-cexpr V* Γ (*cexpr-comp-aux x f e*)
⟨*proof*⟩

lemma *nonneg-cexpr-comp*:
assumes *nonneg-cexpr* (*shift-var-set V*) (*case-nat t2* Γ) *e*

assumes *case-nat t1* $\Gamma \vdash_c f$: *t2 free-vars* $f \subseteq$ *shift-var-set* V
shows *nonneg-cexpr* (*shift-var-set* V) (*case-nat t1* Γ) ($e \circ_c f$)
⟨*proof*⟩

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*⟩

lemma *nonneg-cexpr-int*:
assumes *nonneg-cexpr* (*shift-var-set* V) (*case-nat* t Γ) e
shows *nonneg-cexpr* V Γ ($\int_c e\ \partial t$)
⟨*proof*⟩

Subprobability density expressions

definition *subprob-cexpr* V V' Γ $e \equiv$
$\forall \varrho \in$ *space* (*state-measure* V' Γ).
($\int^+ \sigma$. *extract-real* (*cexpr-sem* (*merge* V V' (σ, ϱ)) e) ∂*state-measure* V Γ) ≤ *1*

lemma *subprob-cexprI*:
assumes $\bigwedge \varrho$. $\varrho \in$ *space* (*state-measure* V' Γ) ⟹
($\int^+ \sigma$. *extract-real* (*cexpr-sem* (*merge* V V' (σ, ϱ)) e) ∂*state-measure* V Γ) ≤ *1*
shows *subprob-cexpr* V V' Γ e
⟨*proof*⟩

lemma *subprob-cexprD*:
assumes *subprob-cexpr* V V' Γ e
shows $\bigwedge \varrho$. $\varrho \in$ *space* (*state-measure* V' Γ) ⟹
($\int^+ \sigma$. *extract-real* (*cexpr-sem* (*merge* V V' (σ, ϱ)) e) ∂*state-measure* V Γ) ≤ *1*
⟨*proof*⟩

lemma *subprob-indicator*:
assumes *subprob*: *subprob-cexpr* V V' Γ *e1* **and** *nonneg*: *nonneg-cexpr* ($V \cup V'$) Γ *e1*
assumes *t1*: $\Gamma \vdash_c$ *e1* : *REAL* **and** *t2*: $\Gamma \vdash_c$ *e2* : *BOOL*
assumes *vars1*: *free-vars e1* $\subseteq V \cup V'$ **and** *vars2*: *free-vars e2* $\subseteq V \cup V'$
shows *subprob-cexpr* V V' Γ (*e1* $*_c$ ⟨*e2*⟩$_c$)
⟨*proof*⟩

lemma *measurable-cexpr-sem'*:
assumes ϱ: $\varrho \in$ *space* (*state-measure* V' Γ)
assumes e: $\Gamma \vdash_c e$: *REAL free-vars* $e \subseteq V \cup V'$
shows ($\lambda\sigma$. *extract-real* (*cexpr-sem* (*merge* V V' (σ, ϱ)) e))
∈ *borel-measurable* (*state-measure* V Γ)
⟨*proof*⟩

lemma *measurable-fun-upd-state-measure*[*measurable*]:
assumes $v \notin V$
shows $(\lambda(x,y).\ y(v := x)) \in$ *measurable* (*stock-measure* $(\Gamma\ v) \bigotimes_M$ *state-measure* $V\ \Gamma$)
(*state-measure* (*insert* $v\ V$) Γ)
⟨*proof*⟩

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* $V'\ \Gamma$)
assumes *e*: $\Gamma \vdash_c e$: *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* $(\Gamma\ v)$.
integrable (*state-measure* $V\ \Gamma$)
($\lambda\sigma$. *extract-real* (*cexpr-sem* (*merge* V (*insert* $v\ V'$) $(\sigma, \varrho(v := x))$) *e*))
(**is** *AE x in ?N*. *integrable ?M* (*?f x*))
⟨*proof*⟩

definition *cdens-ctxt-invar* :: *vname list* ⇒ *vname list* ⇒ *tyenv* ⇒ *cexpr* ⇒ *bool*
where
cdens-ctxt-invar vs vs' Γ $\delta \equiv$
distinct (*vs* @ *vs'*) ∧
free-vars $\delta \subseteq$ *set* (*vs* @ *vs'*) ∧
$\Gamma \vdash_c \delta$: *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* $\delta \subseteq$ *set* (*vs* @ *vs'*); $\Gamma \vdash_c \delta$: *REAL*;
nonneg-cexpr (*set vs* ∪ *set vs'*) Γ δ;
subprob-cexpr (*set vs*) (*set vs'*) Γ δ ⟧ ⟹
cdens-ctxt-invar vs vs' Γ δ
⟨*proof*⟩

lemma *cdens-ctxt-invarD*:
assumes *cdens-ctxt-invar vs vs'* Γ δ
shows *distinct* (*vs* @ *vs'*) *free-vars* $\delta \subseteq$ *set* (*vs* @ *vs'*) $\Gamma \vdash_c \delta$: *REAL*
nonneg-cexpr (*set vs* ∪ *set vs'*) Γ δ *subprob-cexpr* (*set vs*) (*set vs'*) Γ δ
⟨*proof*⟩

lemma *cdens-ctxt-invar-empty*:
assumes *cdens-ctxt-invar vs vs'* Γ δ
shows *cdens-ctxt-invar* [] (*vs* @ *vs'*) Γ (*CReal 1*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *cexpr-typing-expr-rf-to-cexpr*[*intro*]:
assumes *Γ* ⊢ *e* : *t randomfree e*
shows *Γ* ⊢$_c$ *expr-rf-to-cexpr e* : *t*
⟨*proof*⟩

lemma *free-vars-expr-rf-to-cexpr*:
randomfree e ⟹ *free-vars* (*expr-rf-to-cexpr e*) ⊆ *free-vars e*
⟨*proof*⟩

8.4 Builtin density expressions

primrec *dist-dens-cexpr* :: *pdf-dist* ⇒ *cexpr* ⇒ *cexpr* ⇒ *cexpr* **where**
dist-dens-cexpr Bernoulli p x = (*IF*$_c$ *CReal 0* ≤$_c$ *p* ∧$_c$ *p* ≤$_c$ *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* ≤$_c$ *snd*$_c$ *p* ∧$_c$ *fst*$_c$ *p* ≤$_c$ *x* ∧$_c$ *x* ≤$_c$ *snd*$_c$ *p THEN*
inverse$_c$ (⟨*snd*$_c$ *p* −$_c$ *fst*$_c$ *p* +$_c$ *CInt 1*⟩$_c$) *ELSE CReal 0*)
| *dist-dens-cexpr UniformReal p x* = (*IF*$_c$ *fst*$_c$ *p* <$_c$ *snd*$_c$ *p* ∧$_c$ *fst*$_c$ *p* ≤$_c$ *x* ∧$_c$ *x* ≤$_c$ *snd*$_c$ *p THEN*

$inverse_c\ (snd_c\ p\ -_c\ fst_c\ p)\ ELSE\ CReal\ 0)$
| *dist-dens-cexpr Gaussian p x* = $(IF_c\ CReal\ 0 <_c snd_c\ p\ THEN$
$exp_c\ (-_c((x\ -_c\ fst_c\ p)\ \widehat{}_c CInt\ 2\ /_c\ (CReal\ 2\ *_c\ snd_c\ p\ \widehat{}_c CInt\ 2)))\ /_c$
$sqrt_c\ (CReal\ 2\ *_c\ \pi_c\ *_c\ snd_c\ p\ \widehat{}_c\ CInt\ 2)\ ELSE\ CReal\ 0)$
| *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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *val-type-eq-BOOL*: *val-type x* = *BOOL* $\longleftrightarrow$ *x* $\in$ *BoolVal'UNIV*
⟨*proof*⟩

lemma *val-type-eq-INTEG*: *val-type x* = *INTEG* $\longleftrightarrow$ *x* $\in$ *IntVal'UNIV*
⟨*proof*⟩

lemma *val-type-eq-PRODUCT*: *val-type x* = *PRODUCT t1 t2* $\longleftrightarrow$
($\exists$ *a b*. *val-type a* = *t1* $\wedge$ *val-type b* = *t2* $\wedge$ *x* = $\langle|$ *a*, *b* $|\rangle$)
⟨*proof*⟩

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* $\subseteq$ *V free-vars e2* $\subseteq$ *V*
assumes σ $\in$ *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*) $\wedge$
0 $\leq$ *extract-real* (*cexpr-sem* σ (*dist-dens-cexpr dst e1 e2*))
⟨*proof*⟩

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* $\subseteq$ *V free-vars e2* $\subseteq$ *V*
assumes σ $\in$ *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*)
⟨*proof*⟩

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*⟩

8.5 Integral expressions

definition *integrate-var* :: *tyenv* ⇒ *vname* ⇒ *cexpr* ⇒ *cexpr* **where**
integrate-var Γ v e = ∫ *c map-vars* (λ*w*. *if v* = *w then 0 else Suc w*) *e* ∂(*Γ v*)

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

lemma *cexpr-sem-integrate-var*:
cexpr-sem σ (*integrate-var Γ v e*) =
RealVal (∫ *x*. *extract-real* (*cexpr-sem* (*σ*(*v* := *x*)) *e*) ∂*stock-measure* (*Γ v*))
⟨*proof*⟩

lemma *cexpr-sem-integrate-var′*:
extract-real (*cexpr-sem σ* (*integrate-var Γ v e*)) =
(∫ *x*. *extract-real* (*cexpr-sem* (*σ*(*v* := *x*)) *e*) ∂*stock-measure* (*Γ v*))
⟨*proof*⟩

lemma *cexpr-typing-integrate-var*[*simp*]:
Γ ⊢*c e* : *REAL* ⟹ *Γ* ⊢*c integrate-var Γ v e* : *REAL*
⟨*proof*⟩

lemma *cexpr-typing-integrate-vars*[*simp*]:
Γ ⊢*c e* : *REAL* ⟹ *Γ* ⊢*c integrate-vars Γ vs e* : *REAL*
⟨*proof*⟩

lemma *free-vars-integrate-var*[*simp*]:
free-vars (*integrate-var Γ v e*) = *free-vars e* − {*v*}
⟨*proof*⟩

lemma *free-vars-integrate-vars*[*simp*]:
free-vars (*integrate-vars Γ vs e*) = *free-vars e* − *set vs*
⟨*proof*⟩

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*⟩

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*)

$(\lambda\sigma.\ extract\text{-}real\ (cexpr\text{-}sem\ (merge\ (set\ vs)\ V'\ (\sigma,\ \varrho))\ e))$
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*)) =
$\int \sigma.\ extract\text{-}real\ (cexpr\text{-}sem\ (merge\ (set\ vs)\ V'\ (\sigma,\ \varrho))\ e)\ \partial state\text{-}measure$
$(set\ vs)\ \Gamma$
⟨*proof*⟩

lemma *cexpr-sem-integrate-vars'*:
assumes ϱ: $\varrho \in space\ (state\text{-}measure\ V'\ \Gamma)$
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)
$(\lambda\sigma.\ extract\text{-}real\ (cexpr\text{-}sem\ (merge\ (set\ vs)\ V'\ (\sigma,\ \varrho))\ e))$
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*))) =
$\int^+ \sigma.\ extract\text{-}real\ (cexpr\text{-}sem\ (merge\ (set\ vs)\ V'\ (\sigma,\ \varrho))\ e)\ \partial state\text{-}measure$
$(set\ vs)\ \Gamma$
⟨*proof*⟩

lemma *nonneg-cexpr-sem-integrate-vars*:
assumes ϱ: $\varrho \in space\ (state\text{-}measure\ V'\ \Gamma)$
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*)) ≥ 0
⟨*proof*⟩

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* $V'\ \Gamma$ (*integrate-vars* Γ *vs* *e*)
⟨*proof*⟩

lemma *cexpr-sem-integral-nonneg*:
assumes *finite*: $(\int^+ x.\ extract\text{-}real\ (cexpr\text{-}sem\ (case\text{-}nat\ x\ \sigma)\ e)\ \partial stock\text{-}measure\ t) < \infty$
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: $\sigma \in space\ (state\text{-}measure\ V\ \Gamma)$
shows *ennreal* (*extract-real* (*cexpr-sem* σ $(\int_c e\ \partial t)$)) =
$\int^+ x.\ extract\text{-}real\ (cexpr\text{-}sem\ (case\text{-}nat\ x\ \sigma)\ e)\ \partial stock\text{-}measure\ t$
⟨*proof*⟩

lemma *has-parametrized-subprob-density-cexpr-sem-integral*:
assumes *dens*: *has-parametrized-subprob-density* (*state-measure* $V'\ \Gamma$) *M* (*stock-measure* *t*)
$(\lambda\varrho\ x.\ \int^+ y.\ eval\text{-}cexpr\ f\ (case\text{-}nat\ x\ \varrho)\ y\ \partial stock\text{-}measure\ t')$
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) ⊢$_c$ *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*.
(∫$^+$*y*. *eval-cexpr f* (*case-nat x ϱ*) *y* ∂*stock-measure t′*) = *ennreal* (*eval-cexpr* (∫$_c$ *f* ∂*t′*) *ϱ x*)
⟨*proof*⟩

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, λ*σ*. *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*)
⟨*proof*⟩

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 ⊢$_c$ *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 ⊢$_c$ *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*
⟨*proof*⟩

lemma *is-density-exprD*:
assumes *is-density-expr* (*vs*, *vs′*, Γ, *δ*) *t e*
shows *case-nat t* Γ ⊢$_c$ *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*
⟨*proof*⟩

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*⟩

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*⟩

lemma *cexpr-typing-marg-dens-cexpr*[*intro*]:
Γ ⊢$_c$ *δ* : *REAL* ⟹ *case-nat* (*Γ x*) *Γ* ⊢$_c$ *marg-dens-cexpr Γ vs x δ* : *REAL*
⟨*proof*⟩

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*⟩

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*⟩

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* (*λz. 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*⟩

lemma *cexpr-typing-marg-dens2-cexpr*[*intro*]:
assumes Γ ⊢$_c$ *δ* : *REAL*
shows *case-nat* (*PRODUCT* (Γ *x*) (Γ *y*)) Γ ⊢$_c$ *marg-dens2-cexpr Γ vs x y δ* : *REAL*
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

lemma *cexpr-typing-branch-prob-cexpr*[*intro*]:
Γ ⊢$_c$ *δ* : *REAL* ⟹ Γ ⊢$_c$ *branch-prob-cexpr* (*vs*,*vs′*,Γ,*δ*) : *REAL*
⟨*proof*⟩

lemma *cexpr-sem-branch-prob*:
assumes *cdens-ctxt-invar vs vs′ Γ δ*
assumes *ϱ*: *ϱ* ∈ *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*⟩

lemma *subprob-imp-subprob-cexpr*:

assumes *density-context V V′ Γ* (λσ. *extract-real* (*cexpr-sem σ δ*))
shows *subprob-cexpr V V′ Γ δ*
⟨*proof*⟩

end

10 Concrete PDF Compiler

theory *PDF-Compiler*
imports *PDF-Compiler-Pred PDF-Target-Density-Contexts*
begin

inductive *expr-has-density-cexpr* :: *cdens-ctxt* ⇒ *expr* ⇒ *cexpr* ⇒ *bool*
((*1-*/ ⊢$_c$/ (- ⇒/ -)) [*50*,*0*,*50*] *50*) **where**

edc-val: *countable-type* (*val-type v*) ⟹
(*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *Val v* ⇒
map-vars Suc (*branch-prob-cexpr* (*vs*, *vs′*, Γ, *δ*)) ∗$_c$ ⟨*CVar 0* =$_c$ *CVal v*⟩$_c$
| *edc-var*: *x* ∈ *set vs* ⟹ (*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *Var x* ⇒ *marg-dens-cexpr* Γ *vs x δ*
| *edc-pair*: *x* ∈ *set vs* ⟹ *y* ∈ *set vs* ⟹ *x* ≠ *y* ⟹
(*vs*, *vs′*, Γ, *δ*) ⊢$_c$ <*Var x*, *Var y*> ⇒ *marg-dens2-cexpr* Γ *vs x y δ*
| *edc-fail*: (*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *Fail t* ⇒ *CReal 0*
| *edc-let*: ([], *vs* @ *vs′*, Γ, *CReal 1*) ⊢$_c$ *e* ⇒ *f* ⟹
(*shift-vars vs*, *map Suc vs′*, *the* (*expr-type* Γ *e*) · Γ,
map-vars Suc δ ∗$_c$ *f*) ⊢$_c$ *e′* ⇒ *g* ⟹
(*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *LET e IN e′* ⇒ *map-vars* (λ*x*. *x* − *1*) *g*
| *edc-rand*: (*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *e* ⇒ *f* ⟹
(*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *Random dst e* ⇒
∫$_c$ *map-vars* (*case-nat 0* (λ*x*. *x* + *2*)) *f* ∗$_c$
dist-dens-cexpr dst (*CVar 0*) (*CVar 1*) ∂*dist-param-type dst*
| *edc-rand-det*: *randomfree e* ⟹ *free-vars e* ⊆ *set vs′* ⟹
(*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *Random dst e* ⇒
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* ⟹
(*vs*, *vs′*, Γ, *δ* ∗$_c$ ⟨*expr-rf-to-cexpr b*⟩$_c$) ⊢$_c$ *e1* ⇒ *f1* ⟹
(*vs*, *vs′*, Γ, *δ* ∗$_c$ ⟨¬$_c$ *expr-rf-to-cexpr b*⟩$_c$) ⊢$_c$ *e2* ⇒ *f2* ⟹
(*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *IF b THEN e1 ELSE e2* ⇒ *f1* +$_c$ *f2*
| *edc-if*: ([], *vs* @ *vs′*, Γ, *CReal 1*) ⊢$_c$ *b* ⇒ *f* ⟹
(*vs*, *vs′*, Γ, *δ* ∗$_c$ *cexpr-subst-val f TRUE*) ⊢$_c$ *e1* ⇒ *g1* ⟹
(*vs*, *vs′*, Γ, *δ* ∗$_c$ *cexpr-subst-val f FALSE*) ⊢$_c$ *e2* ⇒ *g2* ⟹
(*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *IF b THEN e1 ELSE e2* ⇒ *g1* +$_c$ *g2*
| *edc-op-discr*: (*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *e* ⇒ *f* ⟹ Γ ⊢ *e* : *t* ⟹
op-type oper t = *Some t′* ⟹ *countable-type t′* ⟹
(*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *oper* $$ *e* ⇒
∫$_c$ ⟨(*oper* $$$_c$ (*CVar 0*)) =$_c$ *CVar 1*⟩$_c$ ∗$_c$ *map-vars* (*case-nat 0* (λ*x*. *x*+*2*)) *f* ∂*t*
| *edc-fst*: (*vs*, *vs′*, Γ, *δ*) ⊢$_c$ *e* ⇒ *f* ⟹ Γ ⊢ *e* : *PRODUCT t t′* ⟹

$(vs,\ vs',\ \Gamma,\ \delta) \vdash_c Fst\ \$\$\ e \Rightarrow$
$\int_c (map\text{-}vars\ (case\text{-}nat\ 0\ (\lambda x.\ x + 2))\ f \circ_c <CVar\ 1,\ CVar\ 0>_c)\ \partial t'$

| *edc-snd*: $(vs,\ vs',\ \Gamma,\ \delta) \vdash_c e \Rightarrow f \Longrightarrow \Gamma \vdash e : PRODUCT\ t\ t' \Longrightarrow$
$(vs,\ vs',\ \Gamma,\ \delta) \vdash_c Snd\ \$\$\ e \Rightarrow$
$\int_c (map\text{-}vars\ (case\text{-}nat\ 0\ (\lambda x.\ x + 2))\ f \circ_c <CVar\ 0,\ CVar\ 1>_c)\ \partial t$

| *edc-neg*: $(vs,\ vs',\ \Gamma,\ \delta) \vdash_c e \Rightarrow f \Longrightarrow$
$(vs,\ vs',\ \Gamma,\ \delta) \vdash_c Minus\ \$\$\ e \Rightarrow f \circ_c (\lambda_c x.\ -_c\ x)$

| *edc-addc*: $(vs,\ vs',\ \Gamma,\ \delta) \vdash_c e \Rightarrow f \Longrightarrow randomfree\ e' \Longrightarrow free\text{-}vars\ e' \subseteq set\ vs' \Longrightarrow$
$(vs,\ vs',\ \Gamma,\ \delta) \vdash_c Add\ \$\$\ <e,\ e'> \Rightarrow$
$f \circ_c (\lambda_c x.\ x\ -_c\ map\text{-}vars\ Suc\ (expr\text{-}rf\text{-}to\text{-}cexpr\ e'))$

| *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*: $(vs,\ vs',\ \Gamma,\ \delta) \vdash_c e \Rightarrow f \Longrightarrow \Gamma \vdash e : PRODUCT\ t\ t \Longrightarrow$
$(vs,\ vs',\ \Gamma,\ \delta) \vdash_c Add\ \$\$\ e \Rightarrow$
$\int_c (map\text{-}vars\ (case\text{-}nat\ 0\ (\lambda x.\ x+2))\ f \circ_c (\lambda_c x.\ <x,\ CVar\ 1\ -_c\ x>_c))\ \partial t$

| *edc-inv*: $(vs,\ vs',\ \Gamma,\ \delta) \vdash_c e \Rightarrow f \Longrightarrow$
$(vs,\ vs',\ \Gamma,\ \delta) \vdash_c Inverse\ \$\$\ e \Rightarrow$
$(f \circ_c (\lambda_c x.\ inverse_c\ x)) *_c (\lambda_c x.\ (inverse_c\ x)\ \hat{}_c\ CInt\ 2)$

| *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* ⟨*proof*⟩

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*: $free\text{-}vars\ e \subseteq set\ vs \cup set\ vs'$
assumes *hd*: $dens\text{-}ctxt\text{-}\alpha\ ([],\ vs\ @\ vs',\ \Gamma,\ CReal\ 1) \vdash_d e \Rightarrow (\lambda x\ xa.\ ennreal\ (eval\text{-}cexpr\ f\ x\ xa))$
notes *invar* = *cdens-ctxt-invarD*[*OF inv*]
assumes *wf1*: $is\text{-}density\text{-}expr\ ([],\ vs\ @\ vs',\ \Gamma,\ CReal\ 1)\ t'\ f$
shows $cdens\text{-}ctxt\text{-}invar\ (shift\text{-}vars\ vs)\ (map\ Suc\ vs')\ (t' \cdot \Gamma)\ (map\text{-}vars\ Suc\ \delta *_c f)$
⟨*proof*⟩

lemma *cdens-ctxt-invar-insert-bool*:
assumes *dens*: $dens\text{-}ctxt\text{-}\alpha\ ([],\ vs\ @\ vs',\ \Gamma,\ CReal\ 1) \vdash_d b \Rightarrow (\lambda\varrho\ x.\ ennreal\ (eval\text{-}cexpr\ f\ \varrho\ x))$
assumes *wf*: $is\text{-}density\text{-}expr\ ([],\ vs\ @\ vs',\ \Gamma,\ CReal\ 1)\ BOOL\ f$
assumes *t*: $\Gamma \vdash b : BOOL$ **and** *vars*: $free\text{-}vars\ b \subseteq set\ vs \cup set\ vs'$
assumes *invar*: $cdens\text{-}ctxt\text{-}invar\ vs\ vs'\ \Gamma\ \delta$

shows *cdens-ctxt-invar vs vs′ Γ* (*δ* $*_c$ *cexpr-subst-val f* (*BoolVal v*))
⟨*proof*⟩

lemma *space-state-measureD-shift*:
σ ∈ *space* (*state-measure* (*shift-var-set V*) (*case-nat t Γ*)) ⟹
∃ *x σ′*. *x* ∈ *type-universe t* ∧ *σ′* ∈ *space* (*state-measure V Γ*) ∧ *σ* = *case-nat x σ′*
⟨*proof*⟩

lemma *space-state-measure-shift-iff*:
σ ∈ *space* (*state-measure* (*shift-var-set V*) (*case-nat t Γ*)) ⟷
(∃ *x σ′*. *x* ∈ *type-universe t* ∧ *σ′* ∈ *space* (*state-measure V Γ*) ∧ *σ* = *case-nat x σ′*)
⟨*proof*⟩

lemma *nonneg-cexprI-shift*:
assumes ⋀*x σ*. *x* ∈ *type-universe t* ⟹ *σ* ∈ *space* (*state-measure V Γ*) ⟹
0 ≤ *extract-real* (*cexpr-sem* (*case-nat x σ*) *e*)
shows *nonneg-cexpr* (*shift-var-set V*) (*case-nat t Γ*) *e*
⟨*proof*⟩

lemma *nonneg-cexpr-shift-iff*:
nonneg-cexpr (*shift-var-set V*) (*case-nat t Γ*) (*map-vars Suc e*) ⟷ *nonneg-cexpr V Γ e*
⟨*proof*⟩

lemma *case-nat-case-nat*: *case-nat x n* (*case-nat y m i*) = *case-nat* (*case-nat x n y*) (*λi′*. *case-nat x n* (*m i′*)) *i*
⟨*proof*⟩

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* (*λx*. *Suc* (*Suc x*))) *e*) ⟷
nonneg-cexpr (*shift-var-set V*) (*case-nat t1 Γ*) *e*
⟨*proof*⟩

lemma *nonneg-cexpr-Add*:
assumes *Γ* ⊢$_c$ *e1* : *REAL Γ* ⊢$_c$ *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*⟩

lemma *expr-has-density-cexpr-sound-aux*:
assumes *Γ* ⊢ *e* : *t* (*vs*, *vs′*, *Γ*, *δ*) ⊢$_c$ *e* ⇒ *f cdens-ctxt-invar vs vs′ Γ δ*
free-vars e ⊆ *set vs* ∪ *set vs′*
shows *dens-ctxt-α* (*vs*,*vs′*,*Γ*,*δ*) ⊢$_d$ *e* ⇒ *eval-cexpr f* ∧ *is-density-expr* (*vs*,*vs′*,*Γ*,*δ*) *t f*
⟨*proof*⟩

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*⟩

inductive *expr-compiles-to* :: *expr* ⇒ *pdf-type* ⇒ *cexpr* ⇒ *bool* (- : - ⇒$_c$ - [*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* ⇒$_c$ *f*

code-pred *expr-compiles-to* ⟨*proof*⟩

lemma *expr-compiles-to-sound*:
assumes *e* : *t* ⇒$_c$ *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 · Γ′ $\vdash_c$ *f* : *REAL*
free-vars f ⊆ {*0*}
⟨*proof*⟩

11 Tests

values {(*t*, *f*) |*t f*. *Val* (*IntVal 42*) : *t* ⇒$_c$ *f*}
values {(*t*, *f*) |*t f*. *Minus* $$ (*Val* (*IntVal 42*)) : *t* ⇒$_c$ *f*}
values {(*t*, *f*) |*t f*. *Fst* $$ (*Val* <|*IntVal 13*, *IntVal 37*|>) : *t* ⇒$_c$ *f*}
values {(*t*, *f*) |*t f*. *Random Bernoulli* (*Val* (*RealVal 0.5*)) : *t* ⇒$_c$ *f*}
values {(*t*, *f*) |*t f*. *Add* $$ <*Val* (*IntVal 37*), *Minus* $$ (*Val* (*IntVal 13*))> : *t* ⇒$_c$ *f*}
values {(*t*, *f*) |*t f*. *LET Val* (*IntVal 13*) *IN LET Minus* $$ (*Val* (*IntVal 37*)) *IN*
Add $$ <*Var 0*, *Var 1*> : *t* ⇒$_c$ *f*}
values {(*t*, *f*) |*t f*. *IF Random Bernoulli* (*Val* (*RealVal 0.5*)) *THEN*
Random Bernoulli (*Val* (*RealVal 0.25*))
ELSE
Random Bernoulli (*Val* (*RealVal 0.75*)) : *t* ⇒$_c$ *f*}
values {(*t*, *f*) |*t f*. *LET Random Bernoulli* (*Val* (*RealVal 0.5*)) *IN*
IF Var 0 THEN
Random Bernoulli (*Val* (*RealVal 0.25*))
ELSE
Random Bernoulli (*Val* (*RealVal 0.75*)) : *t* ⇒$_c$ *f*}

```
values {(t, f) |t f. LET Random Gaussian <Val (RealVal 0), Val (RealVal 1)>
IN
                    LET Random Gaussian <Val (RealVal 0), Val (RealVal 1)> IN
                      Add $$ <Var 0, Var 1> : t ⇒c f}
values {(t, f) |t f. LET Random UniformInt <Val (IntVal 1), Val (IntVal 6)>
IN
                    LET Random UniformInt <Val (IntVal 1), Val (IntVal 6)> IN
                      Add $$ <Var 0, Var 1> : t ⇒c f}
```

```
values {(t, f) |t f. LET Random UniformReal <Val (RealVal 0), Val (RealVal
1)> IN
                     LET Random Bernoulli (Var 0) IN
                     IF Var 0 THEN Add $$ <Var 1, Val (RealVal 1)> ELSE Var
1 : t ⇒c f}
```

```
definition cthulhu skill ≡
  LET Random UniformInt (Val <|IntVal 1, IntVal 100|>)
   IN IF Less $$ <Val (IntVal skill), Var 0> THEN
       Val (IntVal skill)
     ELSE IF Or $$ <Less $$ <Var 0, Val (IntVal 6)>,
                  Less $$ <Mult $$ <Var 0, Val (IntVal 5)>,
                           Add $$ <Val (IntVal skill), Val (IntVal 1)> > > THEN
      Add $$ <IF Less $$ <Val (IntVal skill),
                        Random UniformInt <Val (IntVal 1), Val (IntVal 100)> >
THEN
               Random UniformInt <Val (IntVal 1), Val (IntVal 10)>
             ELSE
               Val (IntVal 0),
         Val (IntVal skill)>
     ELSE Val (IntVal skill)

definition cthulhu′ (skill :: int) =
LET Random UniformInt (Val <|IntVal 1, IntVal 100|>)
  IN IF Less $$ <Val (IntVal skill), Var 0> THEN
       Val (IntVal skill)
     ELSE IF Or $$ <Less $$ <Var 0, Val (IntVal 6)>,
                  Less $$ <Mult $$ <Var 0, Val (IntVal 5)>,
                           Add $$ <Val (IntVal skill), Val (IntVal 1)> > > THEN
```

```
  LET Random UniformInt (Val <|IntVal 1, IntVal 100|>)
  IN  Add $$ <IF Less $$ <Val (IntVal skill), Var 1 > THEN
                Random UniformInt (Val <|IntVal 1, IntVal 10|>)
              ELSE
                Val (IntVal 0),
        Val (IntVal skill)>
  ELSE Val (IntVal skill)
```

values $\{(t, f) \mid t\ f.\ cthulhu'\ 42 : t \Rightarrow_c f\}$

end

References

[1] S. Bhat, J. Borgström, A. D. Gordon, and C. Russo. Deriving probability density functions from probabilistic functional programs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2013.

[2] M. Eberl. A verified compiler for probability density functions. Master's thesis, Institut für Informatik, TU München, 2014. http://home.in.tum.de/~eberlm/pdfcompiler.pdf.