Lot 4.2

Technologie de modélisation

Probabilités

Bibliothèques Coq pour des notions probabilistes élémentaires

Description :	Ce document décrit une bibliothèque Coq pour la modélisation et l'analyse de programmes probabilistes. Cette bibliothèque est disponible comme contribution au système Coq version 8.1. Elle contient en particulier une modélisation de l'intervalle [0,1]. L'approche pour représenter les constructions probabilistes est de les considérer comme des mesures sur l'espace des résultats τ (ie des fonctions de type $(\tau \to [0,1]) \to [0,1]$ qui satisfont des conditions de stabilité par rapport aux opérations de base sur $[0,1]$). L'interprétation des programmes comme mesures peut se voir comme une construction monadique. La bibliothèque prouve des principes de base permettant d'estimer la probabilité d'un programme de satisfaire une certaine propriété. Des exemples simples sont étudiés : marche aléatoire, distribution de Bernouilli ...
Auteur(s) :	Christine PAULIN-MOHRING
Référence :	AVERROES / Lot 4.2 / Fourniture 4 / V1.0
Date :	4 janvier 2006
Statut :	à valider
Version :	1.0

Réseau National des Technologies Logicielles
Projet subventionné par le Ministère de la Recherche et des Nouvelles Technologies

CRIL Technology, France Télécom R&D, INRIA-Futurs, LaBRI (Univ. de Bordeaux – CNRS), LIX (École Polytechnique, CNRS) LORIA, LRI (Univ. de Paris Sud – CNRS), LSV (ENS de Cachan – CNRS)

Historique

4 janvier 2006	V 0.1	version préliminaire
4 janvier 2006	V 1.0	mise au format averroes

Table des matières

1 Introduction

This library forms a basis for reasoning on randomised algorithms in the proof assistant COQ [4]. The source files are available as a Coq contribution (see `http://coq.inria.fr`).

As proposed by Kozen [1, 2], we interpret probabilistic programs as measure transformers ; the originality of our approach is to view this interpretation as a monadic transformation on functional programs. Using this semantics, we derive general rules for estimating the probability for a randomised algorithm to satisfy a given property. We apply this approach to the formal proof in COQ of properties of randomised algorithms. We study the example of a program implementing a Bernoulli distribution using a coin flip as a primitive. We prove the probabilistic termination of a linear random walk. We also extend this approach in order to measure probability of traces in a probabilistic transition system.

The library is composed of the following files :

Ubase An axiomatisation of the interval $[0, 1]$. The primitive operations are bounded addition $(x, y) \mapsto \min(x + y, 1)$, multiplication $(x, y) \mapsto x \times y$ and an inverse function $x \mapsto 1 - x$ as well as a function which associates $\frac{1}{n+1}$ to each integer n. We also introduce the predicates $\leq$ and $=$ and a least-upper bound on all sequences of elements of $[0, 1]$.

Uprop Derived operations and properties of operators on $[0, 1]$. We define the operations max, a bounded difference $(x, y) \mapsto \max(x - y, 0)$, the special operator $x\&y$ defined as $\max(x + y - 1, 0)$, the functions $(n, x) \mapsto x^n$, $(n, x) \mapsto nx$, with n an integer, the function $(f, n) \mapsto \Sigma_{i=0}^{i=n-1} f(i)$, the mean of two points $(x, y) \mapsto \frac{1}{2}x + \frac{1}{2}y$.

Monads Definition of the basic monad for randomized constructions, the type α is mapped to the type $(\alpha \to [0, 1]) \to [0, 1]$ of measure functions. We define the `unit` and `star` constructions and prove that they satisfy the basic monadic properties. A measure will be a function of type $(\alpha \to [0, 1]) \to [0, 1]$ that enjoys extra properties such as monotonicity, stability with respect to basic operations. We prove that functions produced by `unit` and `star` satisfy these extra properties under apropriate assumptions.

Probas Definition of a dependent type for distributions on a type α. A distribution on a type α is a record containing a function μ of type $(\alpha \to [0, 1]) \to [0, 1]$ and proofs that this function enjoys the stability properties of measures. These properties are :

$$\begin{aligned} \mu(f_1 + f_2) &= \mu(f_1) + \mu(f_2) \\ \mu(k \times f) &= k \times \mu(f) \\ \mu(1 - f) &\leq 1 - \mu(f) \\ \mu(f) &\leq \mu(g) \quad \text{when } f \leq g \quad (i.e.\ \forall x. f(x) \leq g(x)) \end{aligned}$$

We define the interpretation of specific random primitives : the distribution corresponding to a coin flip and the distribution corresponding to the random function which applied to n gives a number between 0 and n with probability $\frac{1}{n+1}$.

Prog Definition of randomized programs constructions. We define the conditional construction and a fixpoint operator obtained by iterating a monotonic functional. We introduce an axiomatic semantics for these randomized programs : let e be a randomized expression of type τ, p be an element of $[0, 1]$ and q be a function of type $\tau \to [0, 1]$, we define $p \leq [e](q)$ to be the property : the measure of q by the distribution associated to the expression e is not less than p. In the case where q is the characteristic function of a predicate Q, $p \leq [e](q)$ can be interpreted as "the probability for the result of the evaluation of e to satisfy Q is not less than p". In the particular case where q is the constant function equal to 1, the relation $p \leq [e](q)$ can be interpreted as "the probability for the evaluation of e to terminate is not less than p".

We derive inference rules for this relation.

IterFlip A proof of probabilistic termination for a random walk. We consider the program

```
let rec iter x = if flip() then iter (x+1) else x
```

We prove that the probability that this program terminates is 1.

Choice A proof of composition of two runs of a probabilistic program, when a choice can improve the quality of the result. Given two randomized expressions p_1 and p_2 of type τ and a function Q to be estimated, we consider a `choice` function such that the value of Q for `choice`(x,y) is not less than $Q(x) + Q(y)$. We prove that if p_i evaluates Q not less than k_i and terminates with probability 1 then the expression `choice`(p_1, p_2) evaluates Q not less than $k_1(1 - k_2) + k_2$ (which is greater than both k_1 and k_2 when k_1 and k_2 are not equal to 0).

Bernouilli Construction of a bernouilli distribution from the flip distribution. We consider the program

```
let rec bernouilli p = if flip () then
    if x < 1/2 then false else bernouilli (2*p-1)
    else if x < 1/2 then bernouilli (2 p) else true
```

We prove that the probability of `bernouilli`(p) to answer `true` is exactly p.

Transitions A probabilistic transition system is defined by a set of states and a probabilistic transition function which associates to a state a the probability to go to a state b. In our system it corresponds to a function from states to distribution on states. We use this function in order to define the corresponding distribution on paths of length k for a given integer k. This library uses a module `Nelist` defining non empty lists.

2 Ubase.v : Specification of U, interval $[0, 1]$

Require Export *Setoid.*
Set Implicit *Arguments.*

2.1 Preliminaries

2.1.1 Definition of iterator *comp*

comp f u n x is defined as $(f\ (u\ (n-1))..(f(u\ 0)\ x))$

Fixpoint *comp* $(A : Type)\ (f : A \to A \to A)\ (x : A)\ (u : nat \to A)\ (n : nat)\ \{struct\ n\} : A :=$
$match\ n\ with\ O \Rightarrow x \mid (S\ p) \Rightarrow f\ (u\ p)\ (comp\ f\ x\ u\ p)\ end.$

Lemma *comp0* : $\forall\ (A : Type)\ (f : A \to A \to A)\ (x : A)\ (u : nat \to A),\ comp\ f\ x\ u\ 0 = x.$

Lemma *compS* : $\forall\ (A : Type)\ (f : A \to A \to A)\ (x : A)\ (u : nat \to A)\ (n : nat),$
$comp\ f\ x\ u\ (S\ n) = f\ (u\ n)\ (comp\ f\ x\ u\ n).$

2.1.2 Monotonicity of sequences for an arbitrary relation

Definition *mon_seq* $(A : Type)\ (le : A \to A \to Prop)\ (f : nat \to A)$
$:= \forall\ n\ m,\ (n \le m) \to (le\ (f\ n)\ (f\ m)).$

Definition *decr_seq* $(A : Type)\ (le : A \to A \to Prop)\ (f : nat \to A)$
$:= \forall\ n\ m,\ (n \le m) \to (le\ (f\ m)\ (f\ n)).$

2.2 Specification of U

- Constants : 0 and 1
- Constructor : $Unth\ n (\equiv \frac{1}{n+1})$
- Operations : $x + y\ (\equiv min(x + y, 1))$, $x * y$, $inv\ x\ (\equiv 1 - x)$
- Relations : $x \leq y$, $x == y$

Module Type *Universe.*
Parameter *U* : *Type.*
Bind Scope *U_scope with U.*
Delimit Scope *U_scope with U.*

Parameters *Ueq Ule* : $U \to U \to$ *Prop.*
Parameters *U0 U1* : *U.*
Parameters *Uplus Umult* : $U \to U \to U$.
Parameter *Uinv* : $U \to U$.
Parameter *Unth* : $nat \to U$.

Infix "+" := *Uplus* : *U_scope.*
Infix "×" := *Umult* : *U_scope.*
Infix "==" := *Ueq* (*at level* 70) : *U_scope.*
Infix "≤" := *Ule* : *U_scope.*
Notation "[1-] *x*" := (*Uinv x*) (*at level* 35, *right associativity*) : *U_scope.*
Notation "0" := *U0* : *U_scope.*
Notation "1" := *U1* : *U_scope.*
Notation "[1/]1+ *n*" := (*Unth n*) (*at level* 35, *right associativity*) : *U_scope.*
Open Local Scope *U_scope.*

2.2.1 Properties

Hypothesis *Ueq_refl* : $\forall\ x : U,\ x == x$.
Hypothesis *Ueq_sym* : $\forall\ x\ y : U,\ x == y \to y == x$.

Hypothesis *Udiff_0_1* : $\neg 0 == 1$.
Hypothesis *Upos* : $\forall\ x : U,\ 0 \leq x$.
Hypothesis *Unit* : $\forall\ x : U,\ x \leq 1$.

Hypothesis *Uplus_sym* : $\forall\ x\ y : U,\ (x + y) == (y + x)$.
Hypothesis *Uplus_assoc* : $\forall\ x\ y\ z : U,\ (x + (y + z)) == (x + y + z)$.
Hypothesis *Uplus_zero_left* : $\forall\ x : U,\ (0 + x) == x$.

Hypothesis *Umult_sym* : $\forall\ x\ y : U,\ (x \times y) == (y \times x)$.
Hypothesis *Umult_assoc* : $\forall\ x\ y\ z : U,\ (x \times (y \times z)) == (x \times y \times z)$.
Hypothesis *Umult_one_left* : $\forall\ x : U,\ (1 \times x) == x$.

Hypothesis *Uinv_one* : [1-] $1 == 0$.
Hypothesis *Uinv_opp_left* : $\forall\ x$, [1-] $x + x == 1$.

Hypothesis : $1 - (x + y) + x = 1 - y$ holds when $x + y$ does not overflow
Hypothesis *Uinv_plus_left* : $\forall\ x\ y,\ y \leq$ [1-] $x \to$ [1-] $(x + y) + x ==$ [1-] y.

Hypothesis : $(x + y) \times z = x \times z + y \times z$ holds when $x + y$ does not overflow
Hypothesis *Udistr_plus_right* : $\forall\ x\ y\ z,\ x \leq$ [1-] $y \to (x + y) \times z == (x \times z + y \times z)$.

Hypothesis : $1 - (x \times y) = (1 - x) \times y + (1 - y)$
Hypothesis *Udistr_inv_right* : $\forall\ x\ y : U$, [1-] $(x \times y) ==$ ([1-] $x) \times y +$ [1-] y.

The relation $x \leq y$ is reflexive, transitive and anti-symmetric
Hypothesis *Ueq_le* : $\forall\ x\ y : U,\ x == y \to x \leq y$.

Hypothesis *Ule_trans* : $\forall\ x\ y\ z : U,\ (x \leq y) \to (y \leq z) \to (x \leq z)$.

Hypothesis *Ule_antisym* : $\forall$ *x y* :*U*, $(x \leq y) \rightarrow (y \leq x) \rightarrow (x == y)$.

Totality of the order

Hypothesis *Ueq_double_neg* : $\forall$ *x y* : *U*, $\neg \neg x == y \rightarrow x == y$.

Hypothesis *Ule_total* : $\forall$ *x y* : *U*, $(x \leq y) \vee (y \leq x)$.

The relation $x \leq y$ is compatible with operators
Hypothesis *Uplus_le_compat_right* : $\forall$ *x y z* :*U*, $x \leq y \rightarrow (x + z) \leq (y + z)$.

Hypothesis *Umult_le_compat_right* : $\forall$ *x y z* :*U*, $x \leq y \rightarrow (x \times z) \leq (y \times z)$.

Hypothesis *Uinv_le_compat* : $\forall$ *x y* :*U*, $x \leq y \rightarrow$ [1-] $y \leq$ [1-] x.

Properties of simplification in case there is no overflow
Hypothesis *Uplus_le_simpl_right* : $\forall$ *x y z*, $z \leq$ [1-] $x \rightarrow (x + z) \leq (y + z) \rightarrow x \leq y$.

Hypothesis *Umult_le_simpl_left* : $\forall$ *x y z* : *U*, $\neg 0 == z \rightarrow (z \times x) \leq (z \times y) \rightarrow x \leq y$.

Hypothesis *Unth* $\frac{1}{n+1} == 1 - n \times \frac{1}{n+1}$
Hypothesis *Unth_prop* : $\forall$ *n*, [1/]1+*n* == [1-](*comp Uplus* 0 (*fun k* $\Rightarrow$ [1/]1+*n*) *n*).

2.2.2 Archimedian property

Hypothesis *archimedian* : $\forall$ *x*, $\neg x == 0 \rightarrow \exists$ *n*, [1/]1+$n \leq x$.

2.2.3 Least upper bound, corresponds to limit for increasing sequences

Variable *lub* : (*nat* $\rightarrow$ *U*) $\rightarrow$ *U*.

Hypothesis *le_lub* : $\forall$ (*f* : *nat* $\rightarrow$ *U*) (*n* :*nat*), (*f n*) $\leq$ (*lub f*).
Hypothesis *lub_le* : $\forall$ (*f* : *nat* $\rightarrow$ *U*) (*x* :*U*), ($\forall$ *n*, *f n* $\leq x$) $\rightarrow$ (*lub f*) $\leq x$.

2.2.4 Stability properties of lubs with respect to + and ×

Hypothesis *lub_eq_plus_cte_right* : $\forall$ (*f* :*nat*$\rightarrow$*U*) (*k* :*U*), *lub* (*fun n* $\Rightarrow$ (*f n*) + *k*) == (*lub f*) + *k*.

Hypothesis *lub_eq_mult* : $\forall$ (*k* :*U*) (*f* :*nat*$\rightarrow$*U*), *lub* (*fun n* $\Rightarrow$ (*k* $\times$ (*f n*))) == *k* $\times$ *lub f*.

End *Universe*.

3 Uprop.v : Properties of operators on [0,1]

Set Implicit *Arguments*.
Require Export *Ubase*.
Require Export *Arith*.
Require Export *Omega*.
Module *Univ_prop* (*Univ* :*Universe*).
Import *Univ*.

Hint *Resolve Ueq_refl*.
Hint *Resolve Upos Unit Udiff_0_1 Unth_prop Ueq_le*.
Hint *Resolve Uplus_sym Uplus_assoc Umult_sym Umult_assoc*.
Hint *Resolve Uinv_one Uinv_opp_left Uinv_plus_left*.
Hint *Resolve Uplus_zero_left Umult_one_left Udistr_plus_right Udistr_inv_right*.
Hint *Resolve Uplus_le_compat_right Umult_le_compat_right Uinv_le_compat*.
Hint *Resolve lub_le le_lub lub_eq_mult lub_eq_plus_cte_right*.

Hint *Resolve Ule_total.*
Hint Immediate *Ueq_sym Ule_antisym Ueq_double_neg.*
Open Scope *nat_scope.*
Open Scope *U_scope.*

3.1 Direct consequences of axioms

Lemma *Ule_0_1* : $0 \le 1$.

Lemma *Ule_refl* : $\forall\ x : U, (x \le x)$.
Hint *Resolve Ule_refl.*

3.2 Properties of $==$ derived from properties of $\le$

Lemma *Ueq_trans* : $\forall\ x\ y\ z : U,\ x == y \to y == z \to x == z$.
Hint *Resolve Ueq_trans.*

Lemma *Uplus_eq_compat_right* : $\forall\ x\ y\ z : U,\ x == y \to (x + z) == (y + z)$.

Hint *Resolve Uplus_eq_compat_right.*

Lemma *Uplus_eq_compat_left* : $\forall\ x\ y\ z : U,\ x == y \to (z + x) == (z + y)$.

Lemma *Umult_eq_compat_right* : $\forall\ x\ y\ z : U,\ x == y \to (x \times z) == (y \times z)$.
Hint *Resolve Umult_eq_compat_right.*

Lemma *Umult_eq_compat_left* : $\forall\ x\ y\ z : U,\ x == y \to (z \times x) == (z \times y)$.

Hint *Resolve Uplus_eq_compat_left Umult_eq_compat_left.*

Lemma *Uinv_opp_right* : $\forall\ x,\ x +$ [1-] $x == 1$.
Hint *Resolve Uinv_opp_right.*

3.3 *U* is a setoid

Lemma *Usetoid* : *Setoid_Theory U Ueq.*

Add *Setoid U Ueq Usetoid as U_setoid.*

Add *Morphism Uplus* : *Uplus_eq_compat.*

Add *Morphism Umult* : *Umult_eq_compat.*

Add *Morphism Uinv* : *Uinv_eq_compat.*

Add *Morphism Ule* : *Ule_eq_compat_iff.*

Lemma *Ule_eq_compat* :
$\forall\ x1\ x2 : U,\ x1 == x2 \to \forall\ x3\ x4 : U,\ x3 == x4 \to x1 \le x3 \to x2 \le x4$.

3.4 Definition and properties of $x < y$

Definition *Ult* $(r1\ r2 : U)$: *Prop* := $\neg\ (r2 \le r1)$.

Infix "<" := *Ult* : *U_scope.*

Hint *Unfold Ult.*

Add *Morphism Ult* : *Ult_eq_compat_iff.*

Lemma *Ult_eq_compat* :
$\forall\ x1\ x2 : U,\ x1 == x2 \to \forall\ x3\ x4 : U,\ x3 == x4 \to x1 < x3 \to x2 < x4$.

3.4.1 Properties of $x \leq y$

Lemma *Ule_double_neg* : $\forall\ x\ y$, ~~$x \leq y \rightarrow x \leq y$.

Lemma *Ule_zero_eq* : $\forall\ x$, $x \leq 0 \rightarrow x == 0$.

Lemma *Uge_one_eq* : $\forall\ x$, $1 \leq x \rightarrow x == 1$.

Hint Immediate *Ule_zero_eq Uge_one_eq.*

3.4.2 Properties of $x < y$

Lemma *Ult_neq* : $\forall\ x\ y : U$, $x < y \rightarrow \neg x == y$.

Lemma *Ult_neq_rev* : $\forall\ x\ y : U$, $x < y \rightarrow \neg y == x$.

Lemma *Ult_le* : $\forall\ x\ y : U$, $x < y \rightarrow x \leq y$.

Lemma *Ule_diff_lt* : $\forall\ x\ y\ :\ U$, $x \leq y \rightarrow \neg x{=}{=}y \rightarrow x < y$.

Hint Immediate *Ult_neq Ult_neq_rev Ult_le.*
Hint *Resolve Ule_diff_lt.*

Lemma *Ult_neq_zero* : $\forall\ x$, $\neg 0 == x \rightarrow 0 < x$.

Hint *Resolve Ult_neq_zero.*

3.5 Properties of $+$ and $\times$

Lemma *Udistr_plus_left* : $\forall\ x\ y\ z$, $y \leq$ ([1-] z) $\rightarrow (x \times (y + z)) == (x \times y + x \times z)$.

Lemma *Udistr_inv_left* : $\forall\ x\ y$, [1-] $(x \times y) == (x \times$ ([1-] y)) + [1-] x.

Hint *Resolve Uinv_eq_compat Udistr_plus_left Udistr_inv_left.*

Lemma *Uplus_perm2* : $\forall\ x\ y\ z : U$, $x + (y + z) == y + (x + z)$.

Lemma *Umult_perm2* : $\forall\ x\ y\ z : U$, $x \times (y \times z) == y \times (x \times z)$.

Lemma *Uplus_perm3* : $\forall\ x\ y\ z\ :\ U$, $(x + (y + z)) == z + (x + y)$.

Lemma *Umult_perm3* : $\forall\ x\ y\ z\ :\ U$, $(x \times (y \times z)) == z \times (x \times y)$.

Hint *Resolve Uplus_perm2 Umult_perm2 Uplus_perm3 Umult_perm3.*

Lemma *Uplus_le_compat_left* : $\forall\ x\ y\ z : U$, $(x \leq y) \rightarrow (z + x \leq z + y)$.

Hint *Resolve Uplus_le_compat_left.*

Lemma *Uplus_le_compat* : $\forall\ x\ y\ z\ t : U$, $(x \leq y) \rightarrow (z \leq t) \rightarrow (x + z \leq y + t)$.
Hint Immediate *Uplus_le_compat.*

Lemma *Uplus_zero_right* : $\forall\ x : U$, $(x + 0) == x$.
Hint *Resolve Uplus_zero_right.*

Lemma *Uinv_zero* : [1-] $0 == 1$.
Hint *Resolve Uinv_zero.*

Lemma *Uinv_inv* : $\forall\ x\ :\ U$, [1-] ([1-] x) $== x$.
Hint *Resolve Uinv_inv.*

Lemma *Uinv_simpl* : $\forall\ x\ y\ :\ U$, ([1-] x) == [1-] $y \rightarrow x == y$.

Hint Immediate *Uinv_simpl.*

3.6 More properties on $+$ and $\times$ and *Uinv*

Lemma *Umult_le_compat_left* : $\forall\ x\ y\ z\ :\ U,\ x \leq y \rightarrow (z \times x) \leq (z \times y)$.

Hint *Resolve Umult_le_compat_left.*

Lemma *Umult_one_right* : $\forall\ x\ :U,\ (x \times 1) == x$.
Hint *Resolve Umult_one_right.*

Lemma *Uplus_eq_simpl_right* :
$\forall\ x\ y\ z\ :U,\ (z \leq ([1\text{-}]\ x)) \rightarrow (z \leq ([1\text{-}]\ y)) \rightarrow ((x + z) == (y + z)) \rightarrow (x == y)$.

Lemma *Ule_plus_right* : $\forall\ x\ y,\ (x \leq x + y)$.

Lemma *Ule_plus_left* : $\forall\ x\ y,\ (y \leq x + y)$.
Hint *Resolve Ule_plus_right Ule_plus_left.*

Lemma *Ule_mult_right* : $\forall\ x\ y,\ (x \times y \leq x\ )$.

Lemma *Ule_mult_left* : $\forall\ x\ y,\ (x \times y \leq y)$.
Hint *Resolve Ule_mult_right Ule_mult_left.*

Lemma *Uinv_le_perm_right* : $\forall\ x\ y\ :U,\ (x \leq ([1\text{-}]\ y)) \rightarrow y \leq ([1\text{-}]\ x)$.
Hint *Resolve Uinv_le_perm_right.*

Lemma *Uinv_le_perm_left* : $\forall\ x\ y\ :U,\ (([1\text{-}]\ x) \leq y) \rightarrow ([1\text{-}]\ y) \leq x$.
Hint *Resolve Uinv_le_perm_left.*

Lemma *Uinv_eq_perm_left* : $\forall\ x\ y\ :U,\ (x == ([1\text{-}]\ y)) \rightarrow [1\text{-}]\ x == y$.
Hint Immediate *Uinv_eq_perm_left.*

Lemma *Uinv_eq_perm_right* : $\forall\ x\ y\ :U,\ ([1\text{-}]\ x == y) \rightarrow x == ([1\text{-}]\ y)$.

Hint Immediate *Uinv_eq_perm_right.*

Lemma *Uinv_plus_right* : $\forall\ x\ y,\ y \leq ([1\text{-}]\ x) \rightarrow [1\text{-}]\ (x + y) + y == [1\text{-}]\ x$.
Hint *Resolve Uinv_plus_right.*

Lemma *Uplus_eq_simpl_left* :
$\forall\ x\ y\ z\ :U,\ (x \leq ([1\text{-}]\ y)) \rightarrow (x \leq ([1\text{-}]\ z\ )) \rightarrow ((x + y) == (x + z)) \rightarrow (y == z)$.

Lemma *Uplus_eq_zero_left* : $\forall\ x\ y\ :U,\ (x \leq [1\text{-}]\ y) \rightarrow (x + y) == y \rightarrow x == 0$.

Lemma *Uinv_le_trans* : $\forall\ x\ y\ z\ t,\ (x \leq [1\text{-}]\ y) \rightarrow z{\leq}x \rightarrow t{\leq}y \rightarrow z{<=}[1\text{-}]\ t$.

3.7 Disequality

Lemma *neq_sym* : $\forall\ x\ y,\ \neg x{==}y \rightarrow \neg y{==}x$.
Hint Immediate *neq_sym.*

Lemma *Uinv_neq_compat* : $\forall\ x\ y,\ \neg x == y \rightarrow \neg\ [1\text{-}]\ x == [1\text{-}]\ y$.

Lemma *Uinv_neq_simpl* : $\forall\ x\ y,\ \neg\ [1\text{-}]\ x == [1\text{-}]\ y \rightarrow \neg x == y$.

Hint *Resolve Uinv_neq_compat.*
Hint Immediate *Uinv_neq_simpl.*

Lemma *Uinv_neq_left* : $\forall\ x\ y,\ \neg x == [1\text{-}]\ y \rightarrow \neg\ [1\text{-}]\ x == y$.

Lemma *Uinv_neq_right* : $\forall\ x\ y,\ \neg\ [1\text{-}]\ x == y \rightarrow \neg x == [1\text{-}]\ y$.

3.7.1 Properties of <

Lemma *Ult_antirefl* : $\forall\ x : U,\ \neg x\ <\ x$.

Lemma *Ult_0_1* : $(0 < 1)$.

Lemma *Ule_lt_trans* : $\forall\ x\ y\ z : U,\ x \leq y \rightarrow y < z \rightarrow x < z$.

Lemma *Ult_le_trans* : $\forall\ x\ y\ z : U,\ x < y \rightarrow y \leq z \rightarrow x < z$.

Lemma *Ult_trans* : $\forall\ x\ y\ z : U,\ x < y \rightarrow y < z \rightarrow x < z$.

Hint *Resolve Ult_0_1 Ult_antirefl.*

Lemma *Uplus_neq_zero_left* : $\forall\ x\ y,\ \neg 0 == x \rightarrow \neg 0 == x+y$.

Lemma *Uplus_neq_zero_right* : $\forall\ x\ y,\ \neg 0 == y \rightarrow \neg 0 == x+y$.

Lemma *not_Ult_le* : $\forall\ x\ y,\ \neg x < y \rightarrow y \leq x$.

Lemma *Ule_not_lt* : $\forall\ x\ y,\ x \leq y \rightarrow \neg y < x$.

Hint Immediate *not_Ult_le Ule_not_lt.*

Theorem *Uplus_le_simpl_left* : $\forall\ x\ y\ z : U,\ z \leq [1\text{-}]\ x \rightarrow z + x \leq z + y \rightarrow x \leq y$.

Lemma *Uplus_lt_compat_right* : $\forall\ x\ y\ z : U,\ z \leq [1\text{-}]\ y \rightarrow x < y \rightarrow (x + z) < (y + z)$.

Lemma *Uplus_lt_compat_left* : $\forall\ x\ y\ z : U,\ z \leq [1\text{-}]\ y \rightarrow x < y \rightarrow (z + x) < (z + y)$.

Hint *Resolve Uplus_lt_compat_right Uplus_lt_compat_left.*

Lemma *Uplus_lt_compat* :
$\forall\ x\ y\ z\ t : U,\ z \leq [1\text{-}]\ x \rightarrow t \leq [1\text{-}]\ y \rightarrow x < y \rightarrow z < t \rightarrow (x + z) < (y + t)$.

Hint Immediate *Uplus_lt_compat.*

Lemma *Uplus_lt_simpl_left* : $\forall\ x\ y\ z : U,\ z \leq [1\text{-}]\ y \rightarrow (z + x) < (z + y) \rightarrow x < y$.

Lemma *Uplus_lt_simpl_right* : $\forall\ x\ y\ z : U,\ z \leq [1\text{-}]\ y \rightarrow (x + z) < (y + z) \rightarrow x < y$.

Lemma *Uplus_one_le* : $\forall\ x\ y,\ x + y == 1 \rightarrow [1\text{-}]\ y \leq x$.
Hint Immediate *Uplus_one_le.*

Theorem *Uplus_eq_zero* : $\forall\ x,\ x \leq [1\text{-}]\ x \rightarrow (x + x) == x \rightarrow x == 0$.

Lemma *Umult_zero_left* : $\forall\ x,\ 0 \times x == 0$.
Hint *Resolve Umult_zero_left.*

Lemma *Umult_zero_right* : $\forall\ x,\ (x \times 0) == 0$.
Hint *Resolve Uplus_eq_zero Umult_zero_right.*

3.7.2 Compatibility of operations with respect to order.

Lemma *Umult_le_simpl_right* : $\forall\ x\ y\ z,\ \neg 0 == z \rightarrow (x \times z) \leq (y \times z) \rightarrow x \leq y$.
Hint *Resolve Umult_le_simpl_right.*

Lemma *Umult_simpl_right* : $\forall\ x\ y\ z,\ \neg 0 == z \rightarrow (x \times z) == (y \times z) \rightarrow x == y$.

Lemma *Umult_simpl_left* : $\forall\ x\ y\ z,\ \neg 0 == x \rightarrow (x \times y) == (x \times z) \rightarrow y == z$.

Lemma *Umult_lt_compat_right* : $\forall\ x\ y\ z,\ \neg 0 == z \rightarrow x < y \rightarrow (x \times z) < (y \times z)$.

Lemma *Umult_lt_compat_left* : $\forall\ x\ y\ z,\ \neg 0 == z \rightarrow x < y \rightarrow (z \times x) < (z \times y)$.

Lemma *Umult_lt_simpl_right* : $\forall\ x\ y\ z,\ \neg 0 == z \rightarrow (x \times z) < (y \times z) \rightarrow x < y$.

Lemma *Umult_lt_simpl_left* : $\forall\ x\ y\ z,\ \neg 0 == z \rightarrow (z \times x) < (z \times y) \rightarrow x < y$.

Hint *Resolve Umult_lt_compat_left Umult_lt_compat_right.*

Lemma *Umult_zero_simpl_right* : $\forall\ x\ y,\ 0 == x \times y \to \neg 0 == x \to 0 == y.$

Lemma *Umult_zero_simpl_left* : $\forall\ x\ y,\ 0 == x \times y \to \neg 0 == y \to 0 == x.$

Lemma *Umult_neq_zero* : $\forall\ x\ y,\ \neg 0 == x \to \neg 0 == y \to \neg 0 == x \times y.$
Hint *Resolve Umult_neq_zero.*

3.7.3 More Properties

Lemma *Uplus_one* : $\forall\ x\ y,$ [1-] $x \le y \to x + y == 1.$
Hint *Resolve Uplus_one.*

Lemma *Uplus_one_right* : $\forall\ x,\ x + 1 == 1.$

Lemma *Uplus_one_left* : $\forall\ x : U,\ 1 + x == 1.$
Hint *Resolve Uplus_one_right Uplus_one_left.*

Lemma *Uinv_mult_simpl* : $\forall\ x\ y\ z\ t,\ x \le$ [1-] $y \to (x \times z) \le$ [1-] $(y \times t).$
Hint *Resolve Uinv_mult_simpl.*

Lemma *Umult_inv_plus* : $\forall\ x\ y,\ x \times$ [1-] $y + y == x + y \times$ [1-] $x.$
Hint *Resolve Umult_inv_plus.*

Lemma *Umult_inv_plus_le* : $\forall\ x\ y\ z,\ y \le z \to x \times$ [1-] $y + y \le x \times$ [1-] $z + z.$
Hint *Resolve Umult_inv_plus_le.*

3.8 Definition of x^n

Fixpoint *Uexp* $(x : U)$ $(n : nat)$ $\{struct\ n\}$: U :=
    match n with $O \Rightarrow 1 \mid (S\ p) \Rightarrow x \times$ *Uexp x p end.*

3.9 Definition and properties of $x \& y$

A conjonction operation which coincides with min and mult on 0 and 1, see Morgan & McIver

Definition *Uesp* $(x\ y : U)$:= [1-] ([1-] x + [1-] y).

Infix "&" := *Uesp* (*left associativity, at level* 40) : *U_scope.*

Lemma *Uinv_plus_esp* : $\forall\ x\ y,$ [1-] $(x + y) ==$ [1-] x & [1-] $y.$
Hint *Resolve Uinv_plus_esp.*

Lemma *Uinv_esp_plus* : $\forall\ x\ y,$ [1-] $(x \ \&\ y) ==$ [1-] x + [1-] $y.$
Hint *Resolve Uinv_esp_plus.*

Lemma *Uesp_sym* : $\forall\ x\ y : U,\ x \ \&\ y == y \ \&\ x.$

Lemma *Uesp_one_right* : $\forall\ x : U,\ x \ \&\ 1 == x.$

Lemma *Uesp_zero* : $\forall\ x\ y,\ x \le$ [1-] $y \to x \ \&\ y == 0.$

Hint *Resolve Uesp_sym Uesp_one_right Uesp_zero.*

Lemma *Uesp_zero_right* : $\forall\ x : U,\ x \ \&\ 0 == 0.$

Lemma *Uesp_zero_left* : $\forall\ x : U,\ 0 \ \&\ x == 0.$

Hint *Resolve Uesp_zero_right Uesp_zero_left.*

3.10 Definition and properties of $x - y$

Definition *Uminus* $(x\ y : U)$:= [1-] ([1-] $x + y$).

Infix "-" := *Uminus* : *U_scope*.

Lemma *Uminus_le_compat_left* : $\forall\ x\ y\ z,\ x \leq y \rightarrow x - z \leq y - z$.

Lemma *Uminus_le_compat_right* : $\forall\ x\ y\ z,\ y \leq z \rightarrow x - z \leq x - y$.

Hint *Resolve Uminus_le_compat_left Uminus_le_compat_right.*

Add *Morphism Uminus* : *Uminus_eq_compat.*
Hint Immediate *Uminus_eq_compat.*

Lemma *Uminus_zero_right* : $\forall\ x,\ x - 0 == x$.

Lemma *Uminus_one_left* : $\forall\ x$, $1 - x ==$ [1-] x.

Lemma *Uminus_le_zero* : $\forall\ x\ y,\ x \leq y \rightarrow x - y == 0$.

Hint *Resolve Uminus_zero_right Uminus_one_left Uminus_le_zero.*

Lemma *Uminus_plus_simpl* : $\forall\ x\ y,\ y \leq x \rightarrow (x - y) + y == x$.

Lemma *Uminus_plus_zero* : $\forall\ x\ y,\ x \leq y \rightarrow (x - y) + y == y$.

Hint *Resolve Uminus_plus_simpl Uminus_plus_zero.*

Lemma *Uesp_minus_distr_left* : $\forall\ x\ y\ z,\ (x\ \&\ y) - z == (x - z)\ \&\ y$.

Lemma *Uesp_minus_distr_right* : $\forall\ x\ y\ z,\ (x\ \&\ y) - z == x\ \&\ (y - z)$.

Hint *Resolve Uesp_minus_distr_left Uesp_minus_distr_right.*

Lemma *Uesp_minus_distr* : $\forall\ x\ y\ z\ t,\ (x\ \&\ y) - (z + t) == (x - z)\ \&\ (y - t)$.
Hint *Resolve Uesp_minus_distr.*

3.11 Definition and properties of max

Definition *max* $(x\ y : U) : U := (x - y) + y$.

Lemma *max_le_right* : $\forall\ x\ y : U,\ y \leq x \rightarrow (max\ x\ y) == x$.

Lemma *max_le_left* : $\forall\ x\ y : U,\ x \leq y \rightarrow (max\ x\ y) == y$.

Hint *Resolve max_le_right max_le_left.*

Lemma *max_le_case* : $\forall\ x\ y : U,\ (max\ x\ y) == x \lor (max\ x\ y) == y$.

Add *Morphism max* : *max_eq_compat.*

3.12 Other properties

Lemma *Uplus_minus_simpl_right* : $\forall\ x\ y$, $y \leq$ [1-] $x \rightarrow (x + y) - y == x$.
Hint *Resolve Uplus_minus_simpl_right.*

Lemma *Uplus_minus_simpl_left* : $\forall\ x\ y$, $y \leq$ [1-] $x \rightarrow (x + y) - x == y$.

Lemma *Uminus_assoc_left* : $\forall\ x\ y\ z,\ (x - y) - z == x - (y + z)$.

Hint *Resolve Uminus_assoc_left.*

Lemma *Uminus_le_perm_left* : $\forall\ x\ y\ z,\ y \leq x \rightarrow x - y \leq z \rightarrow x \leq z + y$.

Lemma *Uplus_le_perm_left* : $\forall\ x\ y\ z,\ y \leq x \rightarrow x \leq y + z \rightarrow x - y \leq z$.

Lemma *Uminus_eq_perm_left* : $\forall$ x y z, $y \leq x \rightarrow x$ - $y == z \rightarrow x == z + y$.

Lemma *Uplus_eq_perm_left* : $\forall$ x y z, $y \leq$ [1-] $z \rightarrow x == y + z \rightarrow x$ - $y == z$.

Hint *Resolve Uminus_le_perm_left Uminus_eq_perm_left.*
Hint *Resolve Uplus_le_perm_left Uplus_eq_perm_left.*

Lemma *Uminus_le_perm_right* : $\forall$ x y z, $z \leq y \rightarrow x \leq y$ - $z \rightarrow x + z \leq y$.

Lemma *Uplus_le_perm_right* : $\forall$ x y z, $z \leq$ [1-] $x \rightarrow x + z \leq y \rightarrow x \leq y$ - z.
Hint *Resolve Uminus_le_perm_right Uplus_le_perm_right.*

Lemma *Uminus_le_perm* : $\forall$ x y z, $z \leq y \rightarrow x \leq$ [1-] $z \rightarrow x \leq y$ - $z \rightarrow z \leq y$ - x.
Hint *Resolve Uminus_le_perm.*

Lemma *Uminus_eq_perm_right* : $\forall$ x y z, $z \leq y \rightarrow x == y$ - $z \rightarrow x + z == y$.
Hint *Resolve Uminus_eq_perm_right.*

Lemma *Uminus_zero_le* : $\forall$ x y, x - $y == 0 \rightarrow x \leq y$.

Lemma *Uminus_lt_non_zero* : $\forall$ x y, $x < y \rightarrow \neg y$ - $x == 0$.
Hint Immediate *Uminus_zero_le Uminus_lt_non_zero.*

Lemma *Ult_le_nth* : $\forall$ x y, $x < y \rightarrow (\exists$ n, $x \leq y$ - [1/]1+n).

Lemma *Uminus_distr_left* : $\forall$ x y z, (x - y) $\times$ z == ($x \times z$) - ($y \times z$).

Hint *Resolve Uminus_distr_left.*

Lemma *Uminus_distr_right* : $\forall$ x y z, $x \times$ (y - z) == ($x \times y$) - ($x \times z$).

Hint *Resolve Uminus_distr_right.*

Lemma *Uminus_assoc_right* : $\forall$ x y z, $y \leq x \rightarrow z \leq y \rightarrow x$ - (y - z) == (x - y) + z.

3.13 Definition and properties of generalized sums

Definition *sigma* (*alpha* : *nat* $\rightarrow$ U) (n :*nat*) := *comp Uplus* 0 *alpha n*.

Lemma *sigma0* : $\forall$ (*alpha* : *nat* $\rightarrow$ U), *sigma alpha O* = 0.

Lemma *sigmaS* : $\forall$ (*alpha* : *nat* $\rightarrow$ U) (n :*nat*), *sigma alpha* (S n) = (*alpha* n) + (*sigma alpha* n).

Lemma *sigma_incr* : $\forall$ (f : *nat* $\rightarrow$ U) (n m :*nat*), ($n \leq m$)%*nat* $\rightarrow$ (*sigma* f n) $\leq$ (*sigma* f m).

Hint *Resolve sigma_incr.*

Lemma *sigma_eq_compat* : $\forall$ (f g : *nat* $\rightarrow$ U) (n :*nat*),
($\forall$ k, ($k < n$)%*nat* $\rightarrow$ f k == g k) $\rightarrow$ (*sigma* f n) == (*sigma* g n).

Lemma *sigma_le_compat* : $\forall$ (f g : *nat* $\rightarrow$ U) (n :*nat*),
($\forall$ k, ($k < n$)%*nat* $\rightarrow$ f k $\leq$ g k) $\rightarrow$ (*sigma* f n) $\leq$ (*sigma* g n).

Hint *Resolve sigma_eq_compat sigma_le_compat.*

3.14 Properties of *Unth*

Lemma *Unth_zero* : [1/]1+O == 1.

Notation "[1/2]" := (*Unth* (S O)).

Lemma *Unth_one* : [1/2] == [1-] [1/2].

Hint *Resolve Unth_zero Unth_one.*

Lemma *Unth_one_plus* : [1/2] + [1/2] == 1.
Hint *Resolve Unth_one_plus.*

Lemma *Unth_not_null* : ∀ *n*, ¬ (0 == [1/]1+*n*).
Hint *Resolve Unth_not_null.*

Lemma *Unth_prop_sigma* : ∀ *n*, [1/]1+*n* == [1-] (*sigma* (*fun k* ⇒ [1/]1+*n*) *n*).
Hint *Resolve Unth_prop_sigma.*

Lemma *Unth_sigma_n* : ∀ *n* : *nat*, ¬ (1 == *sigma* (*fun k* ⇒ [1/]1+*n*) *n*).

Lemma *Unth_sigma_Sn* : ∀ *n* : *nat*, 1 == *sigma* (*fun k* ⇒ [1/]1+*n*) (*S n*).

Hint *Resolve Unth_sigma_n Unth_sigma_Sn.*

Lemma *Unth_decr* : ∀ *n*, [1/]1+(*S n*) < [1/]1+*n*.
Hint *Resolve Unth_decr.*

Lemma *Unth_anti_mon* :
∀ *n m*, (*n* ≤ *m*)%*nat* → [1/]1+*m* ≤ [1/]1+*n*.
Hint *Resolve Unth_anti_mon.*

Lemma *Unth_le_half* : ∀ *n*, [1/]1+(*S n*) ≤ [1/2].
Hint *Resolve Unth_le_half.*

3.14.1 Mean of two numbers : $\frac{1}{2}x + \frac{1}{2}y$

Definition *mean* (*x y* :*U*) := [1/2] × *x* + [1/2] × *y*.

Lemma *mean_eq* : ∀ *x* :*U*, *mean x x* ==*x*.

Lemma *mean_le_compat_right* : ∀ *x y z*, *y* ≤ *z* → *mean x y* ≤ *mean x z*.

Lemma *mean_le_compat_left* : ∀ *x y z*, *x* ≤ *y* → *mean x z* ≤ *mean y z*.

Hint *Resolve mean_eq mean_le_compat_left mean_le_compat_right.*

Lemma *mean_lt_compat_right* : ∀ *x y z*, *y* < *z* → *mean x y* < *mean x z*.

Lemma *mean_lt_compat_left* : ∀ *x y z*, *x* < *y* → *mean x z* < *mean y z*.

Hint *Resolve mean_eq mean_le_compat_left mean_le_compat_right.*
Hint *Resolve mean_lt_compat_left mean_lt_compat_right.*

Lemma *mean_le_up* : ∀ *x y*, *x* ≤ *y* → *mean x y* ≤ *y*.

Lemma *mean_le_down* : ∀ *x y*, *x* ≤ *y* → *x* ≤ *mean x y*.

Lemma *mean_lt_up* : ∀ *x y*, *x* < *y* → *mean x y* < *y*.

Lemma *mean_lt_down* : ∀ *x y*, *x* < *y* → *x* < *mean x y*.

Hint *Resolve mean_le_up mean_le_down mean_lt_up mean_lt_down.*

3.14.2 Properties of $\frac{1}{2}$

Lemma *le_half_inv* : ∀ *x*, *x* ≤ [1/2] → *x* ≤ [1-] *x*.

Hint Immediate *le_half_inv.*

Lemma *ge_half_inv* : ∀ *x*, [1/2] ≤ *x* → [1-] *x* ≤ *x*.

Hint Immediate *ge_half_inv.*

Lemma *Uinv_le_half_left* : ∀ *x*, *x* ≤ [1/2] → [1/2] ≤ [1-] *x*.

Lemma *Uinv_le_half_right* : ∀ x, [1/2] ≤ x → [1-] x ≤ [1/2].

Hint *Resolve Uinv_le_half_left Uinv_le_half_right.*

Lemma *half_twice* : ∀ x, (x ≤ [1/2]) → ([1/2]) × (x + x) == x.

Lemma *half_twice_le* : ∀ x, ([1/2]) × (x + x) ≤ x.

Lemma *Uinv_half* : ∀ x, ([1/2]) × ([1-] x) + ([1/2]) == [1-] (([1/2]) × x).

Lemma *half_esp* :
∀ x, ([1/2] ≤ x) → ([1/2]) × (x & x) + [1/2] == x.

Lemma *half_esp_le* : ∀ x, x ≤ ([1/2]) × (x & x) + [1/2].
Hint *Resolve half_esp_le.*

Lemma *half_le* : ∀ x y, y ≤ [1-] y → x ≤ y + y → ([1/2]) × x ≤ y.

Lemma *half_Unth* : ∀ n, ([1/2])*([1/]1+n) ≤ [1/]1+(S n).
Hint *Resolve half_le half_Unth.*

Lemma *half_exp* : ∀ n, *Uexp* ([1/2]) n == *Uexp* ([1/2]) (S n) + *Uexp* ([1/2]) (S n).

3.15 Density

Lemma *Ule_lt_lim* : ∀ x y, (∀ t, t < x → t ≤ y) → x ≤ y.

3.16 Properties of least upper bounds

Section *lubs.*

Lemma *lub_le_stable* : ∀ f g, (∀ n, f n ≤ g n) → *lub* f ≤ *lub* g.

Hint *Resolve lub_le_stable.*

Lemma *lub_eq_stable* : ∀ f g, (∀ n, f n == g n) → *lub* f == *lub* g.

Hint *Resolve lub_eq_stable.*

Lemma *lub_zero* : (*lub* (*fun* n ⇒ 0)) == 0.

Lemma *lub_un* : (*lub* (*fun* n ⇒ 1)) == 1.

Lemma *lub_cte* : ∀ c :U, (*lub* (*fun* n ⇒ c)) == c.

Hint *Resolve lub_zero lub_un lub_cte.*

Lemma *lub_eq_plus_cte_left* : ∀ (f : *nat* → U) (k :U), *lub* (*fun* n ⇒ k + (f n)) == k + (*lub* f).
Hint *Resolve lub_eq_plus_cte_left.*

Variables f g : *nat* → U.

Hypothesis *monf* : (*mon_seq Ule* f).
Hypothesis *mong* : (*mon_seq Ule* g).

Lemma *lub_lift* : ∀ n, (*lub* f) == (*lub* (*fun* k ⇒ f (n+k)%*nat*)).

Hint *Resolve lub_lift.*

Let *sum* := *fun* n ⇒ f n + g n.

Lemma *mon_sum* : *mon_seq Ule sum.*

Hint *Resolve mon_sum.*

Lemma *lub_eq_plus* : *lub* (*fun* n ⇒ (f n) + (g n)) == (*lub* f) + (*lub* g).

Hint *Resolve lub_eq_plus.*

Variables k : U.
Let *prod* := *fun* $n \Rightarrow k \times f\ n$.

Lemma *mon_prod* : *mon_seq Ule prod.*

Let *inv* := *fun* $n \Rightarrow$ [1-] $(g\ n)$.

Lemma *lub_inv* : $(\forall\ n, f\ n \leq inv\ n) \rightarrow lub\ f \leq$ [1-] $(lub\ g)$.

End *lubs.*

3.17 Properties of barycentre of two points

Section *Barycentre.*
Variables $a\ b$: U.
Hypothesis *sum_le_one* : $a \leq$ [1-] b.

Lemma *Uinv_bary* :
$\forall\ x\ y : U$, [1-] $(a \times x + b \times y) == a \times$ [1-] $x + b \times$ [1-] $y +$ [1-] $(a + b)$.
End *Barycentre.*

3.18 Properties of generalized sums *sigma*

Lemma *sigma_plus* :
$\forall\ (f\ g : nat \rightarrow U)\ (n : nat), (sigma\ (fun\ k \Rightarrow (f\ k) + (g\ k))\ n) == (sigma\ f\ n) + (sigma\ g\ n)$.

Definition *retract* $(f : nat \rightarrow U)\ (n : nat) := \forall\ k, (k < n)\%nat \rightarrow (f\ k) \leq$ [1-] $(sigma\ f\ k)$.

Lemma *retract0* : $\forall\ (f : nat \rightarrow U)\ (n : nat)$, *retract* $f\ O$.

Lemma *retract_pred* : $\forall\ (f : nat \rightarrow U)\ (n : nat)$, *retract* $f\ (S\ n) \rightarrow$ *retract* $f\ n$.

Lemma *retractS* : $\forall\ (f : nat \rightarrow U)\ (n : nat)$, *retract* $f\ (S\ n) \rightarrow f\ n \leq$ [1-] $(sigma\ f\ n)$.

Hint *Resolve retract0.*
Hint Immediate *retract_pred retractS.*

Lemma *sigma_mult* :
$\forall\ (f : nat \rightarrow U)\ n\ c$, *retract* $f\ n \rightarrow (sigma\ (fun\ k \Rightarrow c \times (f\ k))\ n) == c \times (sigma\ f\ n)$.
Hint *Resolve sigma_mult.*

Lemma *sigma_prod_maj* : $\forall\ (f\ g : nat \rightarrow U)\ n$,
$(sigma\ (fun\ k \Rightarrow (f\ k) \times (g\ k))\ n) \leq (sigma\ f\ n)$.

Hint *Resolve sigma_prod_maj.*

Lemma *sigma_prod_le* : $\forall\ (f\ g : nat \rightarrow U)\ (c : U)$,
$(\forall\ k, (f\ k) \leq c) \rightarrow \forall\ n$, (*retract* $g\ n$) $\rightarrow (sigma\ (fun\ k \Rightarrow (f\ k) \times (g\ k))\ n) \leq c \times (sigma\ g\ n)$.

Lemma *sigma_prod_ge* : $\forall\ (f\ g : nat \rightarrow U)\ (c : U), (\forall\ k, c \leq (f\ k))$
$\rightarrow \forall\ n$, (*retract* $g\ n$) $\rightarrow c \times (sigma\ g\ n) \leq (sigma\ (fun\ k \Rightarrow (f\ k) \times (g\ k))\ n)$.

Hint *Resolve sigma_prod_maj sigma_prod_le sigma_prod_ge.*

Lemma *sigma_inv* : $\forall\ (f\ g : nat \rightarrow U)\ (n : nat)$, (*retract* $f\ n$) $\rightarrow$
[1-] $(sigma\ (fun\ k \Rightarrow (f\ k) \times (g\ k))\ n) == (sigma\ (fun\ k \Rightarrow (f\ k) \times$ [1-] $(g\ k))\ n) +$ [1-] $(sigma\ f\ n)$.

3.19 Product by an integer

Fixpoint *Nmult* $(n : nat)$ $(x : U)$ $\{struct\ n\}$: U :=
match n with O $\Rightarrow 0 \mid (S\ O) \Rightarrow x \mid S\ p \Rightarrow x + (Nmult\ p\ x)$ *end.*

Condition for n */ x to be exact
Definition *Nmult_def* $(n : nat)$ $(x : U)$:=
match n with O $\Rightarrow$ *True* $\mid (S\ O) \Rightarrow$ *True* $\mid S\ p \Rightarrow x \leq [1/]1{+}p$ *end.*

Lemma *Nmult_def_O* : $\forall$ x, *Nmult_def* O x.
Hint *Resolve Nmult_def_O.*

Lemma *Nmult_def_1* : $\forall$ x, *Nmult_def* $(S\ O)$ x.
Hint *Resolve Nmult_def_1.*

Lemma *Nmult_def_intro* : $\forall$ n x , $x \leq [1/]1{+}n \rightarrow$ *Nmult_def* $(S\ n)$ x.
Hint *Resolve Nmult_def_intro.*

Lemma *Nmult_def_Unth* : $\forall$ n , *Nmult_def* $(S\ n)$ $([1/]1{+}n)$.
Hint *Resolve Nmult_def_Unth.*

Lemma *Nmult_def_pred* : $\forall$ n x, *Nmult_def* $(S\ n)$ $x \rightarrow$ *Nmult_def* n x.

Hint Immediate *Nmult_def_pred.*

Lemma *Nmult_defS* : $\forall$ n x, *Nmult_def* $(S\ n)$ $x \rightarrow x \leq [1/]1{+}n$.
Hint Immediate *Nmult_defS.*

Infix "*/" := *Nmult* (*at level* 60) : *U_scope.*

Lemma *Nmult0* : $\forall$ $(x : U)$, $(O{*}/x) = 0$.

Lemma *Nmult1* : $\forall$ $(n : nat)$ $(x : U)$, $((S\ O){*}/x) = x$.

Lemma *NmultSS* : $\forall$ $(n : nat)$ $(x : U)$, $(S\ (S\ n)\ {*}/x) = x + (S\ n\ {*}/\ x)$.

Lemma *Nmult2* : $\forall$ $(x : U)$, $(2{*}/x) = x + x$.

Lemma *NmultS* : $\forall$ $(n : nat)$ $(x : U)$, $(S\ n\ {*}/\ x) == x + (n{*}/x)$.

Hint *Resolve Nmult1 NmultSS Nmult2 NmultS.*

Add *Morphism Nmult* : *Nmult_eq_compat.*
Hint *Resolve Nmult_eq_compat.*

Lemma *Nmult_eq_compat_right* : $\forall$ $(n\ m : nat)$ $(x : U)$, $(n = m)\%nat \rightarrow (n\ {*}/\ x) == (m\ {*}/\ x)$.
Hint *Resolve Nmult_eq_compat_right.*

Lemma *Nmult_le_compat_left* : $\forall$ n x y, $(x \leq y) \rightarrow (n\ {*}/\ x) \leq (n\ {*}/\ y)$.

Lemma *Nmult_le_compat_right* : $\forall$ n m x, $(n \leq m)\%nat \rightarrow (n\ {*}/\ x) \leq (m\ {*}/\ x)$.

Lemma *Nmult_sigma* : $\forall$ $(n : nat)$ $(x : U)$, $n\ {*}/\ x ==$ *sigma* (*fun* $k \Rightarrow x$) n.

Hint *Resolve Nmult_le_compat_left Nmult_le_compat_right*
Nmult_le_compat_right Nmult_sigma.

Lemma *Nmult_Unth_prop* : $\forall$ $n : nat$, $[1/]1{+}n == [1\text{-}]\ (n{*}/\ ([1/]1{+}n))$.
Hint *Resolve Nmult_Unth_prop.*

Lemma *Nmult_n_Unth* : $\forall$ $n : nat$, $(n\ {*}/\ [1/]1{+}n) == [1\text{-}]\ ([1/]1{+}n)$.

Lemma *Nmult_Sn_Unth* : $\forall$ $n : nat$, $(S\ n\ {*}/\ [1/]1{+}n) == 1$.

Hint *Resolve Nmult_n_Unth Nmult_Sn_Unth.*

Lemma *Nmult_ge_Sn_Unth* : $\forall$ n k, $(S\ n \leq k)\%nat \rightarrow (k\ {*}/\ [1/]1{+}n) == 1$.

Lemma *Nmult_le_n_Unth* : ∀ *n k*, $(k \leq n)\%nat \rightarrow (k$ */ [1/]1+$n) \leq$ [1-] ([1/]1+n).

Hint *Resolve Nmult_ge_Sn_Unth Nmult_le_n_Unth.*

Lemma *Nmult_Umult_assoc_left* : ∀ *n x y*, (*Nmult_def n x*) → (n*/(x×y)) == (n*/x)*y.

Hint *Resolve Nmult_Umult_assoc_left.*

Lemma *Nmult_Umult_assoc_right* : ∀ *n x y*, (*Nmult_def n y*) → (n*/(x×y)) == (x*(n*/y)).

Hint *Resolve Nmult_Umult_assoc_right.*

Lemma *plus_Nmult_distr* : ∀ *n m x*, (n + m */ x)== (n */ x) + (m */ x).

Lemma *Nmult_Uplus_distr* : ∀ *n x y*, (n */ x + y)== (n */ x) + (n */ y).

Lemma *Nmult_mult_assoc* : ∀ *n m x*, (n × m */ x)== (n */ (m */ x)).

Lemma *Nmult_Unth_simpl_left* : ∀ *n x*, (S n) */ ([1/]1+n × x) == x.

Lemma *Nmult_Unth_simpl_right* : ∀ *n x*, (S n) */ (x × [1/]1+n) == x.

Hint *Resolve Nmult_Unth_simpl_left Nmult_Unth_simpl_right.*

Lemma *Uinv_Nmult* : ∀ *k n*, [1-] (k */ [1/]1+n) == ((S n) - k) */ [1/]1+n.

Lemma *Nmult_neq_zero* : ∀ *n x*, ¬0==x → ¬0==S n */ x.
Hint *Resolve Nmult_neq_zero.*

Lemma *Nmult_le_simpl* : ∀ (n :*nat*) (x y :U),
(*Nmult_def* (S n) x) → (*Nmult_def* (S n) y) → (S n */ x) ≤ (S n */ y) → $x \leq y$.

Lemma *Nmult_Unth_le* : ∀ (*n1 n2 m1 m2* :*nat*),
(*n2* × S *n1* ≤ *m2* × S *m1*)%*nat* → (*n2* */ [1/]1+*m1* ≤ *m2* */ [1/]1+*n1*).

Lemma *Nmult_Unth_eq* :
∀ (*n1 n2 m1 m2* :*nat*),
(*n2* × S *n1*= *m2* × S *m1*)%*nat* → (*n2* */ [1/]1+*m1* == *m2* */ [1/]1+*n1*).

Hint *Resolve Nmult_Unth_le Nmult_Unth_eq.*

3.20 Tactic for simplification of goals

```
Ltac Usimpl :=
  match goal with
    ⊢ context [(Uplus 0 ?x)] ⇒ setoid_rewrite (Uplus_zero_left x)
  | ⊢ context [(Uplus ?x 0)] ⇒ setoid_rewrite (Uplus_zero_right x)
  | ⊢ context [(Uplus 1 ?x)] ⇒ setoid_rewrite (Uplus_one_left x)
  | ⊢ context [(Uplus ?x 1)] ⇒ setoid_rewrite (Uplus_one_right x)
  | ⊢ context [(Umult 0 ?x)] ⇒ setoid_rewrite (Umult_zero_left x)
  | ⊢ context [(Umult ?x 0)] ⇒ setoid_rewrite (Umult_zero_right x)
  | ⊢ context [(Umult 1 ?x)] ⇒ setoid_rewrite (Umult_one_left x)
  | ⊢ context [(Umult ?x 1)] ⇒ setoid_rewrite (Umult_one_right x)
  | ⊢ context [(Uminus 0 ?x)] ⇒ setoid_rewrite (Uminus_le_zero 0 x);
                                          [apply (Upos x)| idtac]
  | ⊢ context [(Uminus ?x 0)] ⇒ setoid_rewrite (Uminus_zero_right x)
  | ⊢ context [(Uminus ?x 1)] ⇒ setoid_rewrite (Uminus_le_zero x 1);
                                          [apply (Unit x)| idtac]
  | ⊢ context [([1-] ([1-] ?x))] ⇒ setoid_rewrite (Uinv_inv x)
  | ⊢ context [([1/]1+O)] ⇒ setoid_rewrite Unth_zero
  | ⊢ context [(Nmult 0 ?x)] ⇒ setoid_rewrite (Nmult0 x)
  | ⊢ context [(Nmult 1 ?x)] ⇒ setoid_rewrite (Nmult1 x)
```

| ⊢ (*Ule* (*Uplus* ?x ?y) (*Uplus* ?x ?z)) ⇒ *apply Uplus_le_compat_left*
| ⊢ (*Ule* (*Uplus* ?x ?z) (*Uplus* ?y ?z)) ⇒ *apply Uplus_le_compat_right*
| ⊢ (*Ule* (*Uplus* ?x ?z) (*Uplus* ?z ?y)) ⇒ *setoid_rewrite* (*Uplus_sym z y*) ;
apply Uplus_le_compat_right
| ⊢ (*Ule* (*Uplus* ?x ?y) (*Uplus* ?z ?x)) ⇒ *setoid_rewrite* (*Uplus_sym x y*) ;
apply Uplus_le_compat_right
| ⊢ (*Ueq* (*Uplus* ?x ?y) (*Uplus* ?x ?z)) ⇒ *apply Uplus_eq_compat_left*
| ⊢ (*Ueq* (*Uplus* ?x ?z) (*Uplus* ?y ?z)) ⇒ *apply Uplus_eq_compat_right*
| ⊢ (*Ueq* (*Uplus* ?x ?z) (*Uplus* ?z ?y)) ⇒ *setoid_rewrite* (*Uplus_sym z y*) ;
apply Uplus_eq_compat_right
| ⊢ (*Ueq* (*Uplus* ?x ?y) (*Uplus* ?z ?x)) ⇒ *setoid_rewrite* (*Uplus_sym x y*) ;
apply Uplus_eq_compat_right
| ⊢ (*Ule* (*Umult* ?x ?y) (*Umult* ?x ?z)) ⇒ *apply Umult_le_compat_left*
| ⊢ (*Ule* (*Umult* ?x ?z) (*Umult* ?y ?z)) ⇒ *apply Umult_le_compat_right*
| ⊢ (*Ule* (*Umult* ?x ?z) (*Umult* ?z ?y)) ⇒ *setoid_rewrite* (*Umult_sym z y*) ;
apply Umult_le_compat_right
| ⊢ (*Ule* (*Umult* ?x ?y) (*Umult* ?z ?x)) ⇒ *setoid_rewrite* (*Umult_sym x y*) ;
apply Umult_le_compat_right
| ⊢ (*Ueq* (*Umult* ?x ?y) (*Umult* ?x ?z)) ⇒ *apply Umult_eq_compat_left*
| ⊢ (*Ueq* (*Umult* ?x ?z) (*Umult* ?y ?z)) ⇒ *apply Umult_eq_compat_right*
| ⊢ (*Ueq* (*Umult* ?x ?z) (*Umult* ?z ?y)) ⇒ *setoid_rewrite* (*Umult_sym z y*) ;
apply Umult_eq_compat_right
| ⊢ (*Ueq* (*Umult* ?x ?y) (*Umult* ?z ?x)) ⇒ *setoid_rewrite* (*Umult_sym x y*) ;
apply Umult_eq_compat_right
end.

End *Univ_prop.*

4 Monads.v : Monads for randomized constructions

Set Implicit *Arguments.*
Require Export *Uprop.*

Module *Monad* (*Univ* :*Universe*).
Module *UP* := (*Univ_prop Univ*).

4.1 Definition of monadic operators

Definition *M* (*A* :*Type*) := (*A* → *U*) → *U*.

Definition *unit* (*A* :*Type*) (*x* :*A*) : *M A* := *fun f* ⇒ *f x*.

Definition *star* (*A B* :*Type*) (*a* :*M A*) (*F* :*A* → *M B*) : *M B* := *fun f* ⇒ *a* (*fun x* ⇒ *F x f*).

4.2 Properties of monadic operators

Lemma *law1* : ∀ (*A B* :*Type*) (*x* :*A*) (*F* :*A* → *M B*) (*f* :*B* → *U*), *star* (*unit x*) *F f* = *F x f*.

Lemma *law2* :
∀ (*A* :*Type*) (*a* :*M A*) (*f* :*A* → *U*), *star a* (*fun x* :*A* ⇒ *unit x*) *f* = *a* (*fun x* :*A* ⇒ *f x*).

Lemma *law3* :
∀ (*A B C* :*Type*) (*a* :*M A*) (*F* :*A*→ *M B*) (*G* :*B*→ *M C*)
(*f* :*C*→*U*), *star* (*star a F*) *G f* = *star a* (*fun x* :*A* ⇒ *star* (*F x*) *G*) *f*.

4.3 Properties of distributions

4.3.1 Extension to functions of basic operations

Definition *fle* $(A : Type)\ (f\ g : A \rightarrow U) : Prop := \forall\ x : A,\ (f\ x) \leq (g\ x)$.
Definition *feq* $(A : Type)\ (f\ g : A \rightarrow U) : Prop := \forall\ x : A,\ (f\ x) == (g\ x)$.
Hint *Unfold fle feq.*
Definition *fplus* $(A : Type)\ (f\ g : A \rightarrow U)\ (x : A) : U := (f\ x) + (g\ x)$.
Definition *finv* $(A : Type)\ (f : A \rightarrow U)\ (x : A) : U := Uinv\ (f\ x)$.
Definition *fmult* $(A : Type)\ (k : U)\ (f : A \rightarrow U)\ (x : A) : U := k \times (f\ x)$.
Definition *f_one* $(A : Type)\ (x : A) : U := U1$.
Definition *f_zero* $(A : Type)\ (x : A) : U := U0$.
Definition *f_cte* $(A : Type)\ (c : U)\ (x : A) : U := c$.

Implicit *Arguments f_one* [].
Implicit *Arguments f_zero* [].
Implicit *Arguments f_cte* [].

4.3.2 Expected properties of measures

Definition *monotonic* $(A : Type)\ (m : M\ A) : Prop := \forall\ f\ g : A \rightarrow U,\ fle\ f\ g \rightarrow (m\ f) \leq (m\ g)$.

Definition *stable_eq* $(A : Type)\ (m : M\ A) : Prop := \forall\ f\ g : A \rightarrow U,\ feq\ f\ g \rightarrow (m\ f) == (m\ g)$.

Definition *stable_inv* $(A : Type)\ (m : M\ A) : Prop := \forall\ f : A \rightarrow U,\ m\ (finv\ f) \leq Uinv\ (m\ f)$.

Definition *fplusok* $(A : Type)\ (f\ g : A \rightarrow U) := fle\ f\ (finv\ g)$.
Hint *Unfold fplusok.*

Definition *stable_plus* $(A : Type)\ (m : M\ A) : Prop :=$
$\forall\ f\ g : A \rightarrow U,\ fplusok\ f\ g \rightarrow m\ (fplus\ f\ g) == (m\ f) + (m\ g)$.

Definition *stable_mult* $(A : Type)\ (m : M\ A) : Prop :=$
$\forall\ (k : U)\ (f : A \rightarrow U),\ m\ (fmult\ k\ f) == Umult\ k\ (m\ f)$.

4.3.3 Monotonicity

Lemma *unit_monotonic* : $\forall\ (A : Type)\ (x : A),\ monotonic\ (unit\ x)$.

Lemma *star_monotonic* :
$\forall\ (A\ B : Type)\ (m : M\ A)\ (F : A \rightarrow M\ B)$,
$monotonic\ m \rightarrow (\forall\ a : A,\ monotonic\ (F\ a)) \rightarrow monotonic\ (star\ m\ F)$.

4.3.4 Stability for equality

Lemma *monotonic_stable_eq* : $\forall\ (A : Type)\ (m : M\ A),\ (monotonic\ m) \rightarrow (stable\_eq\ m)$.
Hint *Resolve monotonic_stable_eq.*

Lemma *unit_stable_eq* : $\forall\ (A : Type)\ (x : A),\ stable\_eq\ (unit\ x)$.

Lemma *star_stable_eq* :
$\forall\ (A\ B : Type)\ (m : M\ A)\ (F : A \rightarrow M\ B)$,
$stable\_eq\ m \rightarrow (\forall\ a : A,\ stable\_eq\ (F\ a)) \rightarrow stable\_eq\ (star\ m\ F)$.

4.3.5 Stability for inversion

Lemma *unit_stable_inv* : ∀ (*A* :*Type*) (*x* :*A*), *stable_inv* (*unit x*).

Lemma *star_stable_inv* : ∀ (*A B* :*Type*) (*m* :*M A*) (*F* :*A* → *M B*),
stable_inv m → *monotonic m*
→ (∀ *a* :*A*, *stable_inv* (*F a*)) → (∀ *a* :*A*, *monotonic* (*F a*))
→ *stable_inv* (*star m F*).

4.3.6 Stability for addition

Lemma *unit_stable_plus* : ∀ (*A* :*Type*) (*x* :*A*), *stable_plus* (*unit x*).

Lemma *star_stable_plus* :
∀ (*A B* :*Type*) (*m* :*M A*) (*F* :*A* → *M B*),
stable_plus m → *stable_eq m* →
(∀ *a* :*A*, ∀ *f g*, *fplusok f g* → (*F a f*) ≤ *Uinv* (*F a g*))
→ (∀ *a* :*A*, *stable_plus* (*F a*)) → *stable_plus* (*star m F*).

4.3.7 Stability for product

Lemma *unit_stable_mult* : ∀ (*A* :*Type*) (*x* :*A*), *stable_mult* (*unit x*).

Lemma *star_stable_mult* :
∀ (*A B* :*Type*) (*m* :*M A*) (*F* :*A* → *M B*),
stable_mult m → *stable_eq m* → (∀ *a* :*A*, *stable_mult* (*F a*)) → *stable_mult* (*star m F*).

End *Monad*.

5 Probas.v : Definition of the monad for distributions

Require Export *Uprop*.
Require Export *Monads*.
Set Implicit *Arguments*.
Module *Proba* (*Univ* :*Universe*).
Module *MP* := (*Monad Univ*).

5.1 Definition of distribution

Distributions are measure functions such that
- $\mu(1-f) \leq 1-\mu(f)$
- $\mu(f+g) = \mu(f)+\mu(g)$
- $\mu(k \times f) = k \times \mu(f)$
- $f \leq g \Rightarrow \mu(f) \leq \mu(g)$

Record *distr* (*A* :*Type*) : *Type* :=
{*mu* : *M A*;
mu_stable_inv : *stable_inv mu*;
mu_stable_plus : *stable_plus mu*;
mu_stable_mult : *stable_mult mu*;
mu_monotonic : *monotonic mu*}.

Hint *Resolve mu_stable_plus mu_stable_inv mu_stable_mult mu_monotonic*.

5.2 Properties of measures

Lemma *mu_stable_eq* : ∀ (*A* : *Type*)(*m* :(*distr A*)), *stable_eq* (*mu m*).
Hint *Resolve mu_stable_eq.*
Implicit *Arguments mu_stable_eq* [A].

Lemma *mu_zero* : ∀ (*A* : *Type*)(*m* :(*distr A*)),(*mu m* (*f_zero A*)) == 0.
Hint *Resolve mu_zero.*

Lemma *mu_one_inv* :
∀ (*A* : *Type*)(*m* :(*distr A*)),
mu m (*f_one A*) == 1 → ∀ *f*, *mu m* (*finv f*) == [1-] (*mu m f*).
Hint *Resolve mu_one_inv.*

Lemma *mu_cte* : ∀ (*A* : *Type*)(*m* :(*distr A*)) (*c* :*U*),
mu m (*f_cte A c*) == *c* × *mu m* (*f_one A*).
Hint *Resolve mu_cte.*

Lemma *mu_cte_le* : ∀ (*A* : *Type*)(*m* :(*distr A*)) (*c* :*U*),
mu m (*f_cte A c*) ≤ *c*.

Lemma *mu_cte_eq* : ∀ (*A* : *Type*)(*m* :(*distr A*)) (*c* :*U*),
mu m (*f_one A*) == 1 → *mu m* (*f_cte A c*) == *c*.

Hint *Resolve mu_cte_le mu_cte_eq.*

Lemma *mu_stable_mult_right* : ∀ (*A* : *Type*)(*m* :(*distr A*)) (*c* :*U*) (*f* : *A* → *U*),
mu m (*fun x* ⇒ (*f x*) × *c*) == (*mu m f*) × *c*.

5.3 Monadic operators for distributions

Definition *Munit* : ∀ *A* :*Type*, *A* → *distr A*.

Definition *Mlet* : ∀ *A B* :*Type*, (*distr A*) → (*A* → *distr B*) → *distr B*.

5.4 Operations on distributions

Definition *le_distr* (*A* :*Type*) (*m1 m2* :*distr A*) := ∀ *f*, (*mu m1 f*) ≤ (*mu m2 f*).

Definition *eq_distr* (*A* :*Type*) (*m1 m2* :*distr A*) := ∀ *f*, (*mu m1 f*) == (*mu m2 f*).

Lemma *eq_distr_antisym* : ∀ (*A* :*Type*) (*m1 m2* :*distr A*),
(*le_distr m1 m2*) → (*le_distr m2 m1*) → *eq_distr m1 m2*.

Lemma *le_distr_refl* : ∀ (*A* :*Type*) (*m* :*distr A*), *le_distr m m*.

Lemma *le_distr_trans* : ∀ (*A* :*Type*) (*m1 m2 m3* :*distr A*),
(*le_distr m1 m2*)->(*le_distr m2 m3*)->(*le_distr m1 m3*).

Hint *Resolve le_distr_refl.*
Hint *Unfold le_distr.*

Lemma *le_distr_gen* : ∀ (*A* :*Type*) (*m1 m2* :*distr A*),
(*le_distr m1 m2*) → ∀ *f g*, (*fle f g*) → (*mu m1 f*) ≤ (*mu m2 g*).

5.5 Properties of monadic operators

Lemma *Mlet_unit* : $\forall$ (*A B* :*Type*) (*x* :*A*) (*m* :*A* $\rightarrow$ *distr B*), *eq_distr* (*Mlet* (*Munit x*) *m*) (*m x*).

Lemma *M_ext* : $\forall$ (*A* :*Type*) (*m* :*distr A*), *eq_distr* (*Mlet m* (*fun x* $\Rightarrow$ (*Munit x*))) *m*.

Lemma *Mcomp* : $\forall$ (*A B C* :*Type*) (*m1* :(*distr A*)) (*m2* :*A* $\rightarrow$ *distr B*) (*m3* :*B* $\rightarrow$ *distr C*),
eq_distr (*Mlet* (*Mlet m1 m2*) *m3*) (*Mlet m1* (*fun x* :*A* $\Rightarrow$ (*Mlet* (*m2 x*) *m3*))).

Lemma *Mlet_mon* : $\forall$ (*A B* :*Type*) (*m1 m2* : *distr A*) (*f1 f2* : *A* $\rightarrow$ *distr B*),
le_distr m1 m2 $\rightarrow$ ($\forall$ *x*, *le_distr* (*f1 x*) (*f2 x*)) $\rightarrow$ *le_distr* (*Mlet m1 f1*) (*Mlet m2 f2*).

5.6 A specific distribution

Definition *distr_null* : $\forall$ *A* : *Type*, *distr A*.

Lemma *le_distr_null* : $\forall$ (*A* :*Type*) (*m* : *distr A*), (*le_distr* (*distr_null A*) *m*).
Hint *Resolve le_distr_null*.

5.7 Least upper bound of increasing sequences of distributions

Section *Lubs*.
Variable *A* : *Type*.
Variable *muf* : *nat* $\rightarrow$ (*distr A*).
Hypothesis *muf_mon* : $\forall$ *n m* :*nat*, ($n \leq m$)%*nat* $\rightarrow$ (*le_distr* (*muf n*) (*muf m*)).

Definition *mu_lub_* : *M A* := *fun f* $\Rightarrow$ *lub* (*fun n* $\Rightarrow$ *mu* (*muf n*) *f*).

Definition *mu_lub* : *distr A*.

Lemma *mu_lub_le* : $\forall$ *n* :*nat*, *le_distr* (*muf n*) *mu_lub*.

Lemma *mu_lub_sup* : $\forall$ *m* :(*distr A*), ($\forall$ *n* :*nat*, *le_distr* (*muf n*) *m*) $\rightarrow$ *le_distr mu_lub m*.

End *Lubs*.

5.8 Distribution for *flip*

The distribution associated to *flip* () is $f \mapsto \frac{1}{2}f(true) + \frac{1}{2}f(false)$
Definition *flip* : (*M bool*) := *fun* (*f* : *bool* $\rightarrow$ *U*) $\Rightarrow$ [1/2] $\times$ (*f true*) + [1/2] $\times$ (*f false*).

Lemma *flip_stable_inv* : *stable_inv flip*.

Lemma *flip_stable_plus* : *stable_plus flip*.

Lemma *flip_stable_mult* : *stable_mult flip*.

Lemma *flip_monotonic* : *monotonic flip*.

Definition *ctrue* (*b* :*bool*) := *if b then* 1 *else* 0.
Definition *cfalse* (*b* :*bool*) := *if b then* 0 *else* 1.

Lemma *flip_ctrue* : *flip ctrue* == [1/2].

Lemma *flip_cfalse* : *flip cfalse* == [1/2].

Hint *Resolve flip_ctrue flip_cfalse*.

Definition *Flip* : (*distr bool*).

5.9 Uniform distribution beween 0 and n

Require *Arith*.

5.9.1 Definition of *fnth*

fnth n k is defined as $\frac{1}{n+1}$

Definition *fnth* (n :*nat*) : *nat* → U :=
fun k ⇒ ([1/]1+n).

5.9.2 Basic properties of *fnth*

Lemma *Unth_eq* : ∀ n, *Unth* n == [1-] (*sigma* (*fnth* n) n).
Hint *Resolve Unth_eq*.

Lemma *sigma_fnth_one* : ∀ n, *sigma* (*fnth* n) (S n) == 1.
Hint *Resolve sigma_fnth_one*.

Lemma *Unth_inv_eq* : ∀ n, [1-] ([1/]1+n) == *sigma* (*fnth* n) n.

Lemma *sigma_fnth_sup* : ∀ n m, ($m > n$) → *sigma* (*fnth* n) m == *sigma* (*fnth* n) (S n).

Lemma *sigma_fnth_le* : ∀ n m, (*sigma* (*fnth* n) m) ≤ (*sigma* (*fnth* n) (S n)).

Hint *Resolve sigma_fnth_le*.

fnth is a retract
Lemma *fnth_retract* : ∀ n :*nat*,(*retract* (*fnth* n) (S n)).

Implicit *Arguments fnth_retract* [].

5.9.3 Distribution for *random n*

The distribution associated to *random n* is $f \mapsto \Sigma_{i=0}^{n} \frac{f(i)}{n+1}$ we cannot factorize $\frac{1}{n+1}$ because of possible overflow

Definition *random* (n :*nat*) :*M nat* := *fun* (f :*nat*→U) ⇒ *sigma* (*fun* k ⇒ *Unth* n × f k) (S n).

5.9.4 Properties of *random*

Lemma *random_stable_inv* : ∀ n, *stable_inv* (*random* n).

Lemma *random_stable_plus* : ∀ n, *stable_plus* (*random* n).

Lemma *random_stable_mult* : ∀ n, *stable_mult* (*random* n).

Lemma *random_monotonic* : ∀ n, *monotonic* (*random* n).

Definition *Random* (n :*nat*) : (*distr nat*).

End *Proba*.

6 Prog.v : Composition of distributions

Require Export *Probas*.
Set Implicit *Arguments*.

Module *Rules* (*Univ* :*Universe*).
Module *PP* := (*Proba Univ*).

6.1 Conditional

Definition *Mif* $(A : Type)\ (b : distr\ bool)\ (m1\ m2 : distr\ A)$
$:= Mlet\ b\ (fun\ x : bool \Rightarrow if\ x\ then\ m1\ else\ m2).$

Lemma *Mif_mon* :
$\forall\ (A : Type)\ (b\ b' : distr\ bool)\ (m1\ m2\ m1'\ m2' : distr\ A),$
$le\_distr\ b\ b' \to le\_distr\ m1\ m1' \to le\_distr\ m2\ m2'$
$\to le\_distr\ (Mif\ b\ m1\ m2)\ (Mif\ b'\ m1'\ m2').$

6.2 Fixpoints

Section *Fixpoints.*

6.2.1 Hypotheses

Variables $A\ B : Type.$

Variable $F : (A \to (distr\ B)) \to A \to (distr\ B).$

Hypothesis *F_mon* :
$\forall\ f\ g : A \to (distr\ B),$
$(\forall\ x, (le\_distr\ (f\ x)\ (g\ x))) \to$
$(\forall\ x, (le\_distr\ (F\ f\ x)\ (F\ g\ x))).$

6.2.2 Iteration of the functional F from the 0-distribution

Fixpoint *iter* $(n : nat) : A \to (distr\ B)$
$:= match\ n\ with \mid O \Rightarrow fun\ x \Rightarrow (distr\_null\ B)$
$\mid S\ n \Rightarrow fun\ x \Rightarrow F\ (iter\ n)\ x$
end.

Definition *Flift* $(dn : A \to nat \to (distr\ B))\ (x : A)\ (n : nat) : (distr\ B)$
$:= (F\ (fun\ x \Rightarrow dn\ x\ n)\ x).$

Lemma *Flift_mon* :
$\forall\ dn : A \to nat \to (distr\ B),$
$(\forall\ (x : A)\ (n\ m : nat), (n \leq m)\%nat \to le\_distr\ (dn\ x\ n)\ (dn\ x\ m))$
$\to \forall\ (x : A)\ (n\ m : nat),$
$(n \leq m)\%nat \to le\_distr\ (Flift\ dn\ x\ n)\ (Flift\ dn\ x\ m).$

Hypothesis *F_continuous* :
$\forall$
$(dn : A \to nat \to (distr\ B))$
$(dnmon : \forall\ (x : A)\ (n\ m : nat), (n \leq m)\%nat \to le\_distr\ (dn\ x\ n)\ (dn\ x\ m))$
$(x : A),$
$(le\_distr\ (F\ (fun\ x \Rightarrow mu\_lub\ (dn\ x)\ (dnmon\ x))\ x)$
$(mu\_lub\ (Flift\ dn\ x)\ (Flift\_mon\ dn\ dnmon\ x))).$

Let *muf* $(x : A)\ (n : nat) := (iter\ n\ x).$

Lemma *muf_mon_succc* : $\forall\ (n : nat)\ (x : A), le\_distr\ (muf\ x\ n)\ (muf\ x\ (S\ n)).$

Lemma *muf_mon* : $\forall\ (x : A)\ (n\ m : nat), (n \leq m)\%nat \to le\_distr\ (muf\ x\ n)\ (muf\ x\ m).$

6.2.3 Definition

Definition *Mfix* $(x : A) := mu\_lub\ (fun\ n \Rightarrow iter\ n\ x)\ (muf\_mon\ x).$

6.2.4 Properties

Lemma *Mfix_le_iter* : ∀ (*x* : *A*) (*n* :*nat*), (*le_distr* (*iter n x*) (*Mfix x*)).
Hint *Resolve Mfix_le_iter*.

Lemma *Mfix_le* : ∀ *x* : *A*, (*le_distr* (*Mfix x*) (*F Mfix x*)).

Lemma *Mfix_sup* : ∀ *x* : *A*, (*le_distr* (*F Mfix x*) (*Mfix x*)).

Lemma *Mfix_eq* : ∀ *x* : *A*, (*eq_distr* (*Mfix x*) (*F Mfix x*)).

End *Fixpoints*.

7 Prog.v : An axiomatic semantics for randomized programs

7.1 Definition of correctness judgement

$p \leq [e](q)$ is defined as $p \leq \mu(e)(q)$

Definition *ok* (*A* :*Type*) (*p* :*U*) (*e* :*distr A*) (*q* :*A*→*U*) := *p* ≤ (*mu e q*).

Definition *okfun* (*A B* :*Type*)(*p* :*A*→*U*)(*e* :*A*→*distr B*)(*q* :*B*→*U*)
:= ∀ *x* :*A*, (*ok* (*p x*) (*e x*) *q*).

7.2 Stability properties

Lemma *ok_le_compat* :
∀ (*A* :*Type*) (*p p'* :*U*) (*e* :*distr A*) (*q q'* :*A*→*U*),
p' ≤ *p* → *fle q q'* → *ok p e q* → *ok p' e q'*.

Lemma *ok_eq_compat* :
∀ (*A* :*Type*) (*p p'* :*U*) (*e e'* :*distr A*) (*q q'* :*A*→*U*),
p' == *p* → (*feq q q'*) → *eq_distr e e'* → *ok p e q* → *ok p' e' q'*.

Lemma *okfun_le_compat* :
∀ (*A B* :*Type*) (*p p'* :*A* → *U*) (*e* :*A* → *distr B*) (*q q'* :*B*→*U*),
fle p' p → *fle q q'* → *okfun p e q* → *okfun p' e q'*.

Lemma *okfun_eq_compat* :
∀ (*A B* :*Type*) (*p p'* :*U*) (*e e'* :*distr A*) (*q q'* :*A*→*U*),
p' == *p* → (*feq q q'*) → *eq_distr e e'* → *ok p e q* → *ok p' e' q'*.

7.3 Basic rules

7.3.1 Rule for application

$$\frac{r \leq [a](p) \quad \forall x, p(x) \leq [f(x)](q)}{r \leq [f(a)](q)}$$

Lemma *apply_rule* :
∀ (*A B* :*Type*)(*a* :(*distr A*))(*f* :*A*→*distr B*)(*r* :*U*)(*p* :*A*→*U*)(*q* :*B*→*U*),
(*ok r a p*) → (*okfun p f q*) → (*ok r* (*Mlet a f*) *q*).

7.3.2 Rule for abstraction

Lemma *lambda_rule* :
∀ (*A B* :*Type*)(*f* :*A*→*distr B*)(*p* :*A*→*U*)(*q* :*B*→*U*),
(∀ *x* :*A*, (*ok* (*p x*) (*f x*) *q*)) → (*okfun p f q*).

7.3.3 Rule for conditional

$$\frac{p_1 \leq [e_1](q) \quad p_2 \leq [e_2](q)}{p_1 \times \mu(e_1)(1_{true}) + p_2 \times \mu(e_2)(1_{false}) \leq [if\ b\ then\ e_1\ else\ e_2](q)}$$

Lemma *ifok* : ∀ *f1 f2*, *fplusok* (*fmult f1 ctrue*) (*fmult f2 cfalse*).
Hint *Resolve ifok*.

Lemma *Mif_eq* :
∀ (*A* :*Type*)(*b* :(*distr bool*))(*f1 f2* :*distr A*)(*q* :*A*→*U*),
(*mu* (*Mif b f1 f2*) *q*) == (*mu f1 q*) × (*mu b ctrue*) + (*mu f2 q*) × (*mu b cfalse*).

Lemma *ifrule* :
∀ (*A* :*Type*)(*b* :(*distr bool*))(*f1 f2* :*distr A*)(*p1 p2* :*U*)(*q* :*A*→*U*),
(*ok p1 f1 q*)->(*ok p2 f2 q*)
->(*ok* ((*p1* × (*mu b ctrue*)) + (*p2* × (*mu b cfalse*))) (*Mif b f1 f2*) *q*).

7.3.4 Properties of *Flip*

Lemma *Flip_ctrue* : *mu Flip ctrue* == [1/2].

Lemma *Flip_cfalse* : *mu Flip cfalse* == [1/2].

7.3.5 Rule for fixpoints

with $\phi(x) = F(\phi)(x)$, p_i an increasing sequence of functions starting from 0

$$\frac{\forall f\ i, (\forall x, p_i(x) \leq [f](q)) \Rightarrow \forall x, p_{i+1}(x) \leq [F(f)(x)](q)}{\forall x, \bigcup_i p_i\ x \leq [\phi(x)](q)}$$

Section *Fixrule*.
Variables *A B* : *Type*.

Variable *F* : (*A* → (*distr B*)) → *A* → (*distr B*).

Hypothesis *F_mon* :
∀ *f g* : *A* → (*distr B*), (∀ *x*, (*le_distr* (*f x*) (*g x*))) → (∀ *x*, (*le_distr* (*F f x*) (*F g x*))).

Variable *p* : *A* → *nat* → *U*.
Hypothesis *pmon* : ∀ *x* :*A*, *mon_seq Ule* (*p x*).
Hypothesis *p0* : ∀ *x* :*A*, *p x O* == 0.
Variable *q* : *B* → *U*.
Lemma *fixrule* :
(∀ (*i* :*nat*) (*f* :*A*->(*distr B*)),
(*okfun* (*fun x* ⇒ (*p x i*)) *f q*)
→ *okfun* (*fun x* ⇒ (*p x* (*S i*))) (*fun x* ⇒ *F f x*) *q*)
→ *okfun* (*fun x* ⇒ *lub* (*p x*)) (*Mfix F F_mon*) *q*.

End *Fixrule*.

7.3.6 Rule for *flip*

Lemma *ok_flip* : ∀ *q* : *bool* → *U*, *ok* ([1/2] × *q true* + [1/2] × *q false*) *Flip q*.

End *Rules*.

8 Nelist.v : A general theory of non empty lists on Type

Set Implicit *Arguments*.
Set Strict Implicit.

Section *NELIST*.

Variable *A* : *Type*.

Inductive *nelist* : *Type* :=
singl : *A* → *nelist* | *add* : *A* → *nelist* → *nelist*.

Definition *hd* (*l* :*nelist*) : *A* :=
match l with (*singl a*) ⇒ *a* | (*add a _*) ⇒ *a end*.

Fixpoint *app* (*l m* : *nelist*) {*struct l*} : *nelist* :=
match l with (*singl a*) ⇒ *add a m* | (*add a l1*) ⇒ *add a* (*app l1 m*) *end*.

Fixpoint *rev_app* (*l m* : *nelist*) {*struct l*} : *nelist* :=
match l with (*singl a*) ⇒ *add a m* | (*add a l1*) ⇒ *rev_app l1* (*add a m*) *end*.

Definition *rev* (*l* :*nelist*) : *nelist* :=
match l with (*singl a*) ⇒ *l* | (*add a l1*) ⇒ *rev_app l1* (*singl a*) *end*.

Lemma *app_assoc* : ∀ *l1 l2 l3*, *app l1* (*app l2 l3*) = *app* (*app l1 l2*) *l3*.

Hint *Resolve app_assoc*.

Lemma *rev_app_rev* : ∀ *l m*, *rev_app l m* = *app* (*rev l*) *m*.

Hint *Resolve rev_app_rev*.

Lemma *rev_app_app_rev* : ∀ *l m*, *rev* (*rev_app l m*) = *app* (*rev m*) *l*.

Lemma *rev_rev* : ∀ *l*, *rev* (*rev l*) = *l*.

Lemma *rev_app_distr* : ∀ *l m*, *rev* (*app l m*) = *app* (*rev m*) (*rev l*).

Hint *Resolve rev_rev rev_app_distr*.

Lemma *hd_app* : ∀ *l m*, *hd* (*app l m*) = *hd l*.

Hint *Resolve hd_app*.

Lemma *hd_rev_add* : ∀ *a l*, *hd* (*rev* (*add a l*)) = *hd* (*rev l*).
Hint *Resolve hd_rev_add*.

End *NELIST*.

9 Transitions.v : Probabilistic Deterministic Transition System

Require Export *Prog*.
Set Implicit *Arguments*.

Module *PTS*(*Univ* :*Universe*).
Module *RP* := (*Rules Univ*).
Section *TRANSITIONS*.

Variable *A* : *Type*.

9.1 One step of probabilistic transition

Variable *step* : $A \to distr\ A$.

9.2 Extension to distributions on sequences of length k

Require Export *Nelist*.

Definition *add_step* (*start* : *distr* (*nelist* *A*)) : *M* (*nelist* *A*) :=
fun f ⇒ *mu start* (*fun l* ⇒ (*mu* (*step* (*hd l*)) (*fun x* ⇒ (*f* (*add x l*))))).

Lemma *add_step_stable_inv* :
∀ (*start* : *distr* (*nelist A*)), *stable_inv* (*add_step start*).

Lemma *add_step_stable_plus* :
∀ (*start* : *distr* (*nelist A*)), *stable_plus* (*add_step start*).

Lemma *add_step_stable_mult* :
∀ (*start* : *distr* (*nelist A*)), *stable_mult* (*add_step start*).

Lemma *add_step_monotonic* :
∀ (*start* : *distr* (*nelist A*)), *monotonic* (*add_step start*).

Definition *Add_step* : (*distr* (*nelist A*)) → (*distr* (*nelist A*)).

Definition of the measure
Fixpoint *path* (*k* :*nat*) (*s* : *A*) {*struct k*} : *distr* (*nelist A*) :=
match k with
O ⇒ *Munit* (*singl s*)
|(*S p*) ⇒ *Add_step* (*path p s*)
end.

The opposite view of composition starting from one step

Lemma *path_unfold* : ∀ *k s f*,
mu (*path* (*S k*) *s*) *f* == *mu* (*step s*) (*fun x* ⇒ *mu* (*path k x*) (*fun l* ⇒ *f* (*app l* (*singl s*)))).

End *TRANSITIONS*.

End *PTS*.

10 Bernouilli.v : Simulating Bernouilli distribution

Require Export *Prog*.
Set Implicit *Arguments*.

Module *Bernouilli* (*Univ* :*Universe*).
Module *RP* := (*Rules Univ*).

10.1 Program for computing a Bernouilli distribution

bernouilli p gives true with probability p and false with probability (1-p)

```
let rec bernouilli x = if flip then
        if x < 1/2 then false else bernouilli (2 p - 1)
        else if x < 1/2 then bernouilli (2 p) else true
```

Hypothesis *dec_demi* : ∀ *x* : *U*, {$x <$ *Unth* 1}+{*Unth* 1 $\leq x$}.

Definition *Fbern* (*f* : *U* → (*distr bool*)) (*p* :*U*) :=

Mif Flip
(*if dec_demi p then* (*Munit false*) *else* (*f* (*p* & *p*)))
(*if dec_demi p then* (*f* (*p* + *p*)) *else* (*Munit true*)).

Lemma *Fbern_mon* :
∀ *f g* : *U* → *distr bool*, (∀ *n*, *le_distr* (*f n*) (*g n*))
→ ∀ *n*, *le_distr* (*Fbern f n*) (*Fbern g n*).

Definition *bernouilli* : *U* → (*distr bool*) :=
Mfix Fbern Fbern_mon.

10.2 Properties of the Bernouilli program

10.2.1 Definition of the invariant

$q(p)(n) = p - \frac{1}{2^n}$

Definition *q* (*p* :*U*) (*n* :*nat*) := *p* - (*Uexp* (*Unth* 1) *n*).

Add *Morphism q* : *q_eq_compat*.

10.2.2 Properties of the invariant

Lemma *q_esp_S* : ∀ *p n*, *q* (*p* & *p*) *n* == *q p* (*S n*) & *q p* (*S n*).

Lemma *q_esp_le* : ∀ *p n*, (*q p* (*S n*)) ≤ (*Unth* 1) × (*q* (*p* & *p*) *n*) + (*Unth* 1).

Lemma *q_plus_eq* : ∀ *p n*, (*p* ≤ *Unth* 1) → (*q p* (*S n*)) == (*Unth* 1) × (*q* (*p* + *p*) *n*).

Lemma *q_0* : ∀ *p* :*U*, *q p* 0 == *U0*.

Lemma *p_le* : ∀ (*p* :*U*) (*n* :*nat*), *p* - (*Unth n*) ≤ *q p n*.

Hint *Resolve p_le*.

Lemma *lim_q_p* : ∀ *p*, *p* ≤ *lub* (*q p*).

Hint *Resolve lim_q_p*.

10.2.3 Proof of main results

Property : $\forall p, p \leq [\text{bernouilli } p](\mathbf{result} = \text{true})$

Definition *q1* (*b* :*bool*) := *if b then U1 else U0*.

Lemma *bernouilli_true* : *okfun* (*fun p* ⇒ *p*) *bernouilli q1*.

Property : $\forall p, 1 - p \leq [\text{bernouilli } p](\mathbf{result} = \text{false})$

Definition *q2* (*b* :*bool*) := *if b then U0 else U1*.

Lemma *bernouilli_false* : *okfun* (*fun p* ⇒ *Uinv p*) *bernouilli q2*.

Probability for the result of (bernouilli p) to be true is exactly p

Lemma *q1_q2_inv* : ∀ *b* :*bool*, *q1 b* == *Uinv* (*q2 b*).

Lemma *bernouilli_eq* : ∀ *p*, *mu* (*bernouilli p*) *q1* == *p*.

End *Bernouilli*.

11 Choice.v : An example of probabilistic choice

Require Export *Prog.*
Set Implicit *Arguments.*
Module *Choice* (*Univ* :*Universe*).
Module *RP* := (*Rules Univ*).

11.1 Definition of a probabilistic choice

We interpret the probabilistic program p which executes two probabilistic programs p_1 and p_2 and then make a choice between the two computed results.

```
let rec p () = let x = p1 () in let y = p2 () in choice x y
```

Section *CHOICE.*
Variable *A* : *Type.*
Variables *p1 p2* : *distr A.*
Variable *choice* : $A \to A \to A$.
Definition *p* : *distr A* := *Mlet p1* (*fun x* ⇒ *Mlet p2* (*fun y* ⇒ *Munit* (*choice x y*))).

11.2 Main result

We estimate the probability for p to satisfy Q given estimations for both p_1 and p_2.

11.2.1 Assumptions

We need extra properties on p_1, p_2 and *choice*.
- p_1 and p_2 terminate with probability 1
- Q value on *choice* is not less than the sum of values of Q on separate elements. If Q is a boolean function it means than if one of x or y satisfies Q then (*choice* x y) will also satisfy Q

Hypothesis *p1_terminates* : (*mu p1* (*f_one A*))==*U1.*
Hypothesis *p2_terminates* : (*mu p2* (*f_one A*))==*U1.*

Variable *Q* : $A \to U$.
Hypothesis *choiceok* : ∀ *x y*, $Q\ x + Q\ y \leq Q$ (*choice x y*).

11.2.2 Proof of estimation

$$\frac{k_1 \leq [p_1](Q) \quad k_2 \leq [p_2](Q)}{k_1(1-k_2) + k_2 \leq [p](Q)}$$

Lemma *choicerule* :
∀ *k1 k2*, *k1* ≤ *mu p1 Q* → *k2* ≤ *mu p2 Q*
→ (*k1* × (*Uinv k2*) + *k2*) ≤ *mu p Q.*

End *CHOICE.*
End *Choice.*

12 IterFlip.v : An example of probabilistic termination

Require Export *Prog.*
Set Implicit *Arguments.*

Module *IterFlip* (*Univ* :*Universe*).
Module *RP* := (*Rules Univ*).

12.1 Definition of a random walk

We interpret the probabilistic program

```
let rec iter x = if flip() then iter (x+1) else x
```

Require Import *ZArith*.

Definition *Fiter* $(f : Z \to (distr\ Z))\ (x :Z) :=$
$Mif\ Flip\ (f\ (Zsucc\ x))\ (Munit\ x)$.

Lemma *Fiter_mon* :
$\forall\ f\ g : Z \to distr\ Z, (\forall\ n, le\_distr\ (f\ n)\ (g\ n))$
$\to \forall\ n, le\_distr\ (Fiter\ f\ n)\ (Fiter\ g\ n)$.

Definition *iterflip* : $Z \to (distr\ Z)$:= *Mfix Fiter Fiter_mon*.

12.2 Main result

Probability for *iter* to terminate is 1

12.2.1 Auxiliary function p_n

Definition $p_n = 1 - \frac{1}{2^n}$

Fixpoint $p\ (n : nat) : U := match\ n\ with\ O \Rightarrow U0 \mid (S\ n) \Rightarrow [1/2] \times p\ n + [1/2]\ end$.

Lemma *p_eq* : $\forall\ n :nat, p\ n == Uinv\ (Uexp\ [1/2]\ n)$.
Hint *Resolve p_eq*.

Lemma *p_le* : $\forall\ n :nat, Uinv\ (Unth\ n) \leq p\ n$.

Hint *Resolve p_le*.

Lemma *lim_p_one* : $U1 \leq lub\ p$.

Hint *Resolve lim_p_one*.

12.2.2 Proof of probabilistic termination

Definition $q1\ (z :Z) := U1$.

Lemma *iterflip_term* : $okfun\ (fun\ k \Rightarrow U1)\ iterflip\ q1$.

End *IterFlip*.

Références

[1] Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 1981.

[2] Dexter Kozen. A probabilistic PDL. In *15th ACM Symposium on Theory of Computing*, 1983.

[3] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Monographs in Computer Science. Springer-Verlag, 2005.

[4] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.0*, April 2004. `http://coq.inria.fr`.