A library for reasoning on randomized algorithms in Coq Version 4

Christine Paulin-Mohring

November 21, 2009

1 Introduction

This library forms a basis for reasoning on randomised algorithms in the proof assistant Coq [6]. The source files are available from the author homepage (http://www.lri.fr/~paulin). They can be compiled using Coq version V8.2pl1.

The theoretical basis of this work is a joint work with Philippe Audebaud and is described in [1, 2]. It is developed in the framework of the project SCALP : ANR-07-SESU-010.

As proposed by Kozen [3, 4], we interpret probabilistic programs as measures; 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 prove a general scheme for combining 2 different randomized expressions in order to get a better result. We study an example of partial termination.

With respect to previous versions of this library, we use the mechanism of type classes introduced by N. Oury and M. Sozeau [5] in order to represent the order structure and the cpo structure on a given type. It also defines simple automatic tactics for reasoning on measures.

The library is composed of the following files :

Prelude A few preliminary definition, in particular primitives for reasoning classically in Coq are defined.

Cpo The definition of structures for ordered sets and ω-complete partial orders (a monotonic sequence has a least-upper bound). We define the type $O_1 \stackrel{m}{\to} O_2$ of monotonic functions and define the fixpoint-construction for monotonic functionals.

Utheory An axiomatisation of the interval $[0,1]$. The type $[0,1]$ is given a cpo structure: we have the predicates $\leq$ and $==$, a least element 0 and a least-upper bound on all monotonic sequences of elements of $[0,1]$. The primitive operations are bounded addition $(x,y) \mapsto \min(x+y,1)$, multiplication $(x,y) \mapsto x \times y$, division $(x,y) \mapsto \min(x/y,1)$ and an inverse function $x \mapsto 1-x$ as well as a function which associates $\frac{1}{n+1}$ to each integer n.

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 $(\alpha \to [0,1]) \stackrel{m}{\to} [0,1]$ 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 $(\alpha \to [0,1]) \stackrel{m}{\to} [0,1]$ that enjoys extra properties such as stability with respect to basic operations, continuity. 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 $(\alpha \to [0,1]) \stackrel{m}{\to} [0,1]$ and proofs that this function enjoys the stability properties of measures. These properties are :

$$\begin{array}{rll} \mu(f_1 + f_2) & = \mu(f_1) + \mu(f_2) & \text{when } f_1 \leq 1 - f_2 \\ \mu(k \times f) & = k \times \mu(f) & \\ \mu(1 - f) & \leq 1 - \mu(f) & \\ \mu(\mathrm{lub}_n(f\, n)) & \leq \mathrm{lub}_n(\mu(f\, n)) & \text{when } f \text{ is a monotonic sequence of functions} \end{array}$$

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 probabilistic choice and conditional constructions 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 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).

Bernoulli Construction of a bernoulli distribution from the flip distribution. We consider the program

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

We prove that the probability of `bernouilli`(p) to answer `true` is exactly p. We use this distribution in order to simulate a binomial distribution such that $\mathsf{Pr}((\texttt{binomial}\ p\ n) = k) = C_k^n p^k (1-p)^{n-k}$.

Cover A definition of what it means for a function f to be the characteristic function of a predicate P. Defines also characteristic functions for decidable predicates. This file contains a proof of the principle :

$$\frac{1 \leq [a](\mathbb{I}_P) \qquad \forall x, (P\ x) \Rightarrow k \leq [b](f)}{k \leq [\textsf{let}\ x = a\ \textsf{in}\ b](f)}$$

This file uses the library `Sets.v` which define sets as predicates and finite sets with an inductive definition.

Ycart Evaluation of probability of termination for a program adapted from an example due to B. Ycart, parameterized by a function K of type `nat` $\to$ `nat` and a natural number N.

```
let rec ycart n = if random N < K n then n else ycart (n+1)
```

Probability of termination of (`ycart` n) is shown to be equal to $\prod_{k=n}^{\infty}(1 - Fk)$, with $F\, n = K\, n/(1 + N)$.

Contents

2 Misc.v: Preliminaries

```
Require Export Setoid.
```

2.1 Definition of iterator *compn*

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

```
Fixpoint compn (A:Type)(f:A → A → A) (x:A) (u:nat → A) (n:nat) {struct n}: A :=
  match n with O ⇒ x | (S p) ⇒ f (u p) (compn f x u p) end.
```

```
Lemma comp0 : ∀ (A:Type) (f:A → A → A) (x:A) (u:nat → A), compn f x u 0 = x.
```

```
Lemma compS : ∀ (A:Type) (f:A → A → A) (x:A) (u:nat → A) (n:nat),
              compn f x u (S n) = f (u n) (compn f x u n).
```

2.2 Reducing if constructs

Lemma if_then : $\forall$ (P:Prop) (b:{ P }+{ $\neg$ P})(A:Type)(p q:A),
$P \rightarrow$ (if b then p else q) = p.

Lemma if_else : $\forall$ (P:Prop) (b:{ P }+{ $\neg$ P})(A:Type)(p q:A),
$\neg$ $P \rightarrow$ (if b then p else q) = q.

2.3 Classical reasoning

Definition class (A:Prop) := $\neg$ $\neg$ $A \rightarrow A$.

Lemma class_neg : $\forall$ A:Prop, class ($\neg$ A).

Lemma class_false : class **False**.
Hint Resolve class_neg class_false.

Definition orc (A B:Prop) := $\forall$ C:Prop, class $C \rightarrow (A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C$.

Lemma orc_left : $\forall$ A B:Prop, $A \rightarrow$ orc A B.

Lemma orc_right : $\forall$ A B:Prop, $B \rightarrow$ orc A B.

Hint Resolve orc_left orc_right.

Lemma class_orc : $\forall$ A B, class (orc A B).

Implicit Arguments class_orc [].

Lemma orc_intro : $\forall$ A B, ($\neg$ $A \rightarrow \neg$ $B \rightarrow$ **False**) $\rightarrow$ orc A B.

Lemma class_and : $\forall$ A B, class $A \rightarrow$ class $B \rightarrow$ class ($A \wedge B$).

Lemma excluded_middle : $\forall$ A, orc A ($\neg$ A).

Definition exc (A :Type)(P:A $\rightarrow$ Prop) :=
$\forall$ C:Prop, class $C \rightarrow$ ($\forall$ x:A, P $x \rightarrow C$) $\rightarrow C$.

Lemma exc_intro : $\forall$ (A :Type)(P:A $\rightarrow$ Prop) (x:A), P $x \rightarrow$ exc P.

Lemma class_exc : $\forall$ (A :Type)(P:A $\rightarrow$ Prop), class (exc P).

Lemma exc_intro_class : $\forall$ (A:Type) (P:A $\rightarrow$ Prop), (($\forall$ x, $\neg$ P x) $\rightarrow$ **False**) $\rightarrow$ exc P.

Lemma not_and_elim_left : $\forall$ A B, $\neg$ ($A \wedge B$) $\rightarrow A \rightarrow$ ~B.

Lemma not_and_elim_right : $\forall$ A B, $\neg$ ($A \wedge B$) $\rightarrow B \rightarrow$ ~A.

Hint Resolve class_orc class_and class_exc excluded_middle.

Lemma class_double_neg : $\forall$ P Q: Prop, class $Q \rightarrow (P \rightarrow Q) \rightarrow \neg \neg P \rightarrow Q$.

2.4 Extensional equality

Definition feq A B (f g : $A \rightarrow B$) := $\forall$ x, f $x = g$ x.

Lemma feq_refl : $\forall$ A B (f:A->B), feq f f.

Lemma feq_sym : $\forall$ A B (f g : $A \rightarrow B$), feq f $g \rightarrow$ feq g f.

Lemma feq_trans : $\forall$ A B (f g h: $A \rightarrow B$), feq f $g \rightarrow$ feq g $h \rightarrow$ feq f h.

Hint Resolve feq_refl.
Hint Immediate feq_sym.
Hint Unfold feq.

Add *Parametric Relation* (A B : Type) : ($A \rightarrow B$) (feq (A:=A) (B:=B))
reflexivity *proved* by (feq_refl (A:=A) (B:=B))
symmetry *proved* by (feq_sym (A:=A) (B:=B))
transitivity *proved* by (feq_trans (A:=A) (B:=B))
as *feq_rel*.

3 Ccpo.v: Specification and properties of a cpo

```
Require Export Arith.
Require Export Omega.
Require Export Coq.Classes.SetoidTactics.
Require Export Coq.Classes.SetoidClass.
Require Export Coq.Classes.Morphisms.
Open Local Scope signature_scope.
```

3.1 Ordered type

```
Definition eq_rel {A} (E1 E2:relation A) := ∀ x y, E1 x y ↔ E2 x y.
Class Order {A} (E:relation A) (R:relation A) :=
  {reflexive :> Reflexive R;
   order_eq : ∀ x y, R x y ∧ R y x ↔ E x y;
   transitive :> Transitive R }.
Instance OrderEqRefl '{Order A E R} : Reflexive E.
Save.
Instance OrderEqSym '{Order A E R} : Symmetric E.
Save.
Instance OrderEqTrans '{Order A E R} : Transitive E.
Save.
Instance OrderEquiv '{Order A E R} : Equivalence E.
Save.
Class ord A :=
   { Oeq : relation A;
     Ole : relation A;
     order_rel :> Order Oeq Ole }.
Instance OrdSetoid '(o:ord A) : Setoid A.
Defined.
Add Parametric Relation {A} {o:ord A} : A (@Oeq _ o)
reflexivity proved by OrderEqRefl
symmetry proved by OrderEqSym
transitivity proved by OrderEqTrans
as Oeq_setoid.
Infix "≤" := Ole.
Infix "==" := Oeq : type_scope.
Lemma Ole_refl_eq : ∀ {O} {o:ord O} (x y:O), x == y → x ≤ y.
Hint Immediate @Ole_refl_eq.
Lemma Ole_refl_eq_inv : ∀ {O} {o:ord O} (x y:O), x == y → y ≤ x.
Hint Immediate @Ole_refl_eq_inv.
Lemma Ole_trans : ∀ {O} {o:ord O} (x y z:O), x ≤ y → y ≤ z → x ≤ z.
Lemma Ole_refl : ∀ {O} {o:ord O} (x:O), x ≤ x.
Hint Resolve @Ole_refl.
Add Parametric Relation {A} {o:ord A} : A (@Ole _ o)
reflexivity proved by Ole_refl
transitivity proved by Ole_trans
as Ole_setoid.
```

```
Lemma Ole_antisym : ∀ {O} {o:ord O} (x y:O), x ≤ y → y ≤ x → x==y.
Hint Immediate @Ole_antisym.

Lemma Oeq_refl : ∀ {O} {o:ord O} (x:O), x == x.
Hint Resolve @Oeq_refl.

Lemma Oeq_refl_eq : ∀ {O} {o:ord O} (x y:O), x=y → x == y.
Hint Resolve @Oeq_refl_eq.

Lemma Oeq_sym : ∀ {O} {o:ord O} (x y:O), x == y → y == x.

Lemma Oeq_le : ∀ {O} {o:ord O} (x y:O), x == y → x ≤ y.

Lemma Oeq_le_sym : ∀ {O} {o:ord O} (x y:O), x == y → y ≤ x.

Hint Resolve @Oeq_le.
Hint Immediate @Oeq_sym @Oeq_le_sym.

Lemma Oeq_trans
   : ∀ {O} {o:ord O} (x y z:O), x == y → y == z → x == z.
Hint Resolve @Oeq_trans.

Instance Ole_eq_compat_iff `(o:ord A) :
           Morphism (Oeq (A:=A) ==> Oeq (A:=A) ==> iff) (Ole (ord:=o)).
Save.
```

Equivalence of orders

```
Definition eq_ord {O} (o1 o2:ord O) := eq_rel (Ole (ord:=o1)) (Ole (ord:=o2)).

Lemma eq_ord_equiv : ∀ {O} (o1 o2:ord O), eq_ord o1 o2 →
      eq_rel (Oeq (ord:=o1)) (Oeq (ord:=o2)).

Lemma Ole_eq_compat :
     ∀ {O} {o:ord O} (x1 x2 : O),
       x1 == x2 → ∀ x3 x4 : O, x3 == x4 → x1 ≤ x3 → x2 ≤ x4.

Lemma Ole_eq_right : ∀ {O} {o:ord O} (x y z: O),
            x ≤ y → y == z → x ≤ z.

Lemma Ole_eq_left : ∀ {O} {o:ord O} (x y z: O),
            x == y → y ≤ z → x ≤ z.

Add Parametric Morphism `{o:ord A} : (Oeq (A:=A))
       with signature Oeq ⟹ Oeq ⟹ iff as Oeq_iff_morphism.
Qed.

Add Parametric Morphism `{o:ord A} : (Ole (A:=A))
       with signature Oeq ⟹ Oeq ⟹ iff as Ole_iff_morphism.
Qed.

Add Parametric Morphism `{o:ord A} : (Ole (A:=A))
       with signature Ole –> Ole ⟹ Basics.impl as Ole_impl_morphism.
Qed.
```

3.1.1 Dual order

- Iord x y := y <= x

```
Definition Iord : ∀ O {o:ord O}, ord O.
Defined.

Implicit Arguments Iord [[o]].
```

3.1.2 Order on functions

```
Definition fun_ext A B (R:relation B) : relation (A → B) :=
                fun f g ⇒ ∀ x, R (f x) (g x).
Implicit Arguments fun_ext [B].
```

- ford f g := forall x, f x <= g x

```
Instance ford A O {o:ord O} : ord (A → O) :=
  {Oeq:=fun_ext A (Oeq (A:=O));Ole:=fun_ext A (Ole (A:=O))}.
Defined.
Lemma ford_le_elim : ∀ A O (o:ord O) (f g:A → O), f ≤ g → ∀ n, f n ≤ g n.
Hint Immediate ford_le_elim.
Lemma ford_le_intro : ∀ A O (o:ord O) (f g:A → O), ( ∀ n, f n ≤ g n ) → f ≤ g.
Hint Resolve ford_le_intro.
Lemma ford_eq_elim : ∀ A O (o:ord O) (f g:A → O), f == g → ∀ n, f n == g n.
Hint Immediate ford_eq_elim.
Lemma ford_eq_intro : ∀ A O (o:ord O) (f g:A → O), ( ∀ n, f n == g n ) → f == g.
Hint Resolve ford_eq_intro.
```

3.2 Monotonicity

3.2.1 Definition and properties

```
Class monotonic `{o1:ord O1} `{o2:ord O2} (f : O1 → O2) :=
       monotonic_def : ∀ x y, x ≤ y → f x ≤ f y.
Lemma monotonic_intro : ∀ `{o1:ord O1} `{o2:ord O2} (f : O1 → O2),
  (∀ x y, x ≤ y → f x ≤ f y) → monotonic f.
Hint Resolve @monotonic_intro.
Instance monotonic_morphism `{o1:ord O1} `{o2:ord O2}
             (f : O1 → O2) {m:monotonic f}
          : Morphism (Ole (A:=O1) ==>Ole (A:=O2)) f.
Save.
Class stable `{o1:ord O1} `{o2:ord O2} (f : O1 → O2) :=
       stable_def : ∀ x y, x == y → f x == f y.
Hint Unfold stable.
Lemma stable_intro : ∀ `{o1:ord O1} `{o2:ord O2} (f : O1 → O2),
  (∀ x y, x == y → f x == f y) → stable f.
Hint Resolve @stable_intro.
Instance stable_morphism `{o1:ord O1} `{o2:ord O2} (f : O1 → O2) {s:stable f}
          : Morphism (Oeq (A:=O1) ==>Oeq (A:=O2)) f.
Save.
Typeclasses Opaque monotonic stable.
Instance monotonic_stable `{o1:ord O1} `{o2:ord O2} (f : O1 → O2) {m:monotonic f}
          : stable f.
Save.
```

3.2.2 Type of monotonic functions

```
Record fmon `{o1:ord O1} `{o2:ord O2}:= mon
           {fmont :> O1 → O2;
```

```
        fmonotonic: monotonic fmont}.

Implicit Arguments mon [[O1] [o1] [O2] [o2] [fmonotonic]].
Implicit Arguments fmon [[o1] [o2]].

Hint Resolve @fmonotonic.

Notation "O1 -m-> O2" := (fmon O1 O2)
  (right associativity, at level 30) : O_scope.
Notation "O1 -- -m-> O2" := (fmon O1 (o1:=Iord O1) O2 )
  (right associativity, at level 30) : O_scope.
Notation "O1 -- -m-> -- O2" := (fmon O1 (o1:=Iord O1) O2 (o2:=Iord O2))
  (right associativity, at level 30) : O_scope.
Notation "O1 -m-> -- O2" := (fmon O1 O2 (o2:=Iord O2))
  (right associativity, at level 30) : O_scope.

Open Scope O_scope.

Lemma mon_simpl : ∀ `{o1:ord O1} `{o2:ord O2} (f:O1 → O2){mf: monotonic f} x,
      mon f x = f x.
Hint Resolve @mon_simpl.

Instance fstable `{o1:ord O1} `{o2:ord O2} (f:O1 -m-> O2) : stable f.
Save.

Hint Resolve @fstable.

Lemma fmon_le : ∀ `{o1:ord O1} `{o2:ord O2} (f:O1 -m-> O2) x y,
                  x ≤ y → f x ≤ f y.
Hint Resolve @fmon_le.

Lemma fmon_eq : ∀ `{o1:ord O1} `{o2:ord O2} (f:O1 -m-> O2) x y,
                  x == y → f x == f y.
Hint Resolve @fmon_eq.

Instance fmono O1 O2 {o1:ord O1} {o2:ord O2} : ord (O1 -m-> O2)
   := {Oeq := fun (f g : O1-m-> O2)⇒ ∀ x, f x == g x;
       Ole := fun (f g : O1-m-> O2)⇒ ∀ x, f x ≤ g x}.
Defined.

Lemma mon_le_compat : ∀ `{o1:ord O1} `{o2:ord O2} (f g:O1 → O2)
      {mf:monotonic f} {mg:monotonic g}, f ≤ g → mon f ≤ mon g.
Hint Resolve @ mon_le_compat.

Lemma mon_eq_compat : ∀ `{o1:ord O1} `{o2:ord O2} (f g:O1→ O2)
      {mf:monotonic f} {mg:monotonic g}, f == g → mon f == mon g.
Hint Resolve @ mon_eq_compat.

Add Parametric Morphism `{o1:ord O1} `{o2:ord O2}
       : (fmont (O1:=O1) (O2:=O2))
       with signature Oeq ⟹Oeq ⟹Oeq as fmont_eq_morphism.
Qed.
```

3.2.3 Monotonicity and dual order

```
Lemma Imonotonic `{o1:ord O1} `{o2:ord O2} (f:O1 → O2) {m:monotonic f}
         : monotonic (o1:=Iord O1) (o2:=Iord O2) f.

Hint Extern 2 (@monotonic _ (Iord _) _ (Iord _) _) ⇒ apply @Imonotonic
  : typeclass_instances.

Definition imon `{o1:ord O1} `{o2:ord O2} (f:O1 → O2) {m:monotonic f} : O1 -m-> -- O2 :=
                  mon (o1:=Iord O1) (o2:=Iord O2) f.
```

`Lemma` *imon_simpl* : ∀ ‘{*o1*:ord *O1*} ‘{*o2*:ord *O2*} (*f*:O1 → *O2*) {*m*:monotonic *f*} (*x*:O1), *imon f x* = *f x*.

- *Iord* (*A* → *U*) corresponds to *A* → *Iord U*

`Lemma` *Iord_app* {*A*} ‘{*o1*:ord *O1*} (*x*: *A*) : ((*A* → *O1*) – $\stackrel{m}{\to}$ – *O1*).

- *Imon f* uses f as monotonic function over the dual order.

`Definition` *Imon* : ∀ ‘{*o1*:ord *O1*} ‘{*o2*:ord *O2*}, (*O1* $\stackrel{m}{\to}$ *O2*) → (*O1* – $\stackrel{m}{\to}$ – *O2*).
`Defined`.

`Lemma` *Imon_simpl* : ∀ ‘{*o1*:ord *O1*} ‘{*o2*:ord *O2*} (*f*:O1 $\stackrel{m}{\to}$ *O2*)(**x**:O1),
Imon f x = *f x*.

3.2.4 Monotonic functions with 2 arguments

`Class` *monotonic2* ‘{*o1*: *ord O1*} ‘{*o2*: *ord O2*} ‘{*o3*:ord *O3*} (*f*:O1 → *O2* → *O3*) :=
monotonic2_intro : ∀ (*x y*:O1) (*z t*:O2), *x* ≤ *y* → *z* ≤ *t* → *f x* **z** ≤ *f y t*.

`Instance` *mon2_intro* ‘{*o1*: *ord O1*} ‘{*o2*: *ord O2*} ‘{*o3*:ord *O3*} (*f*:O1 → *O2* → *O3*)
{*m1*:monotonic *f*} {*m2*: ∀ *x*, *monotonic* (*f x*)} : *monotonic2 f* | 10.
`Save`.

`Lemma` *mon2_elim1* ‘{*o1*: *ord O1*} ‘{o2: *ord O2*} ‘{*o3*:ord *O3*} (*f*:O1 → *O2* → *O3*)
{*m*:monotonic2 *f*} : *monotonic f*.

`Lemma` *mon2_elim2* ‘{*o1*: *ord O1*} ‘{*o2*: *ord O2*} ‘{o3:ord *O3*} (f:O1 → *O2* → *O3*)
{*m*:monotonic2 *f*} : ∀ *x*, *monotonic* (*f x*).
`Hint Immediate` @*mon2_elim1* @*mon2_elim2*: *typeclass_instances*.

`Definition` *mon_comp* {*A*} ‘{*o1*: *ord O1*} ‘{*o2*: *ord O2*} (*f*:A → *O1* → *O2*) {*mf*:forall *x*, *monotonic* (*f x*)}
: *A* → *O1* $\stackrel{m}{\to}$ *O2* := `fun` *x* ⇒ *mon* (*f x*).

`Instance` *mon_fun_mon* ‘{*o1*: *ord O1*} ‘{*o2*: *ord O2*} ‘{*o3*:ord *O3*} (*f*:O1 → *O2* → *O3*)
{**m**:monotonic2 *f*} : *monotonic* (`fun` *x* ⇒ *mon* (*f x*)).
`Save`.

`Class` *stable2* ‘{*o1*: *ord O1*} ‘{*o2*: *ord O2*} ‘{*o3*:ord *O3*} (*f*:O1 → *O2* → *O3*) :=
stable2_intro : ∀ (*x y*:O1) (*z t*:O2), **x**==y → *z* == *t* → *f x z* == *f y t*.

`Instance` *monotonic2_stable2* ‘{*o1*: *ord O1*} ‘{*o2*: *ord O2*} ‘{*o3*:ord *O3*}
(*f*:O1 → *O2* → *O3*) {*m*:monotonic2 *f*} : *stable2 f*.
`Save`.

`Type`*classes Opaque monotonic2* stable2.

`Definition` *mon2* ‘{*o1*: *ord O1*} ‘{*o2*: *ord O2*} ‘{*o3*:ord *O3*} (*f*:O1 → *O2* → *O3*)
{*mf*:monotonic2 *f*} : *O1* $\stackrel{m}{\to}$ *O2* $\stackrel{m}{\to}$ *O3* := *mon* (`fun` *x* ⇒ *mon* (*f x*)).

`Lemma` *mon2_simpl* : ∀ ‘{*o1*: ord *O1*} ‘{*o2*: *ord O2*} ‘{*o3*:ord *O3*} (*f*:O1 → *O2* → **O3**)
{*mf*:monotonic2 *f*} **x** *y*, *mon2 f x y* = *f x y*.
`Hint Resolve` @*mon2_simpl*.

`Lemma` *mon2_le_compat* : ∀ ‘{*o1*: *ord O1*} ‘{*o2*: *ord O2*} ‘{*o3*:ord *O3*}
(*f g*:O1 → *O2* → *O3*) {*mf*: *monotonic2 f*} {*mg*:monotonic2 *g*},
f ≤ *g* → *mon2 f* ≤ *mon2 g*.

`Definition` *fun2* ‘{*o1*: *ord* **O1**} ‘{*o2*: *ord O2*} ‘{*o3*:ord *O3*} (*f*:O1 → *O2* $\stackrel{m}{\to}$ *O3*)
: *O1* → *O2* → *O3* := `fun` *x* ⇒ *f x*.

`Instance` *fmon2_mon* ‘{*o1*: *ord O1*} ‘{*o2*: *ord O2*} ‘{*o3*:ord *O3*} (*f*:O1 → *O2* $\stackrel{m}{\to}$ *O3*) :
∀ *x*:O1, *monotonic* (*fun2 f x*).
`Save`.

```
Instance fun2_monotonic `{o1: ord O1} `{o2: ord O2} `{o3:ord O3}
        (f:O1 → O2 -m-> O3) {mf:monotonic f} : monotonic (fun2 f).
Save.
Hint Resolve @fun2_monotonic.

Instance fmonotonic2 `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (f:O1 -m-> O2 -m-> O3)
        : monotonic2 (fun2 f).
Save.
Hint Resolve @fmonotonic2.

Definition mfun2 `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (f:O1 -m-> O2 -m-> O3)
   : O1 -m-> (O2 → O3) := mon (fun2 f).

Lemma mfun2_simpl : ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (f:O1 -m-> O2 -m-> O3) x y,
     mfun2 f x y = f x y.

Instance mfun2_mon `{o1:ord O1} `{o2:ord O2} `{o3:ord O3}
        (f:O1 -m-> O2 -m-> O3) x : monotonic (mfun2 f x).
Save.

Lemma mon2_fun2 : ∀ `{o1: ord O1} `{o2: ord O2} `{o3:ord O3}
     (f:O1 -m-> O2 -m-> O3), mon2 (fun2 f) == f.

Lemma fun2_mon2 : ∀ `{o1: ord O1} `{o2: ord O2} `{o3:ord O3}
      (f:O1 → O2 → O3) {mf:monotonic2 f} , fun2 (mon2 f) == f.
Hint Resolve @mon2_fun2 @fun2_mon2.

Instance fstable2 `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (f:O1 -m-> O2 -m-> O3)
                : stable2 (fun2 f).
Save.
Hint Resolve @fstable2.

Definition Imon2 : ∀ `{o1: ord O1} `{o2: ord O2} `{o3:ord O3},
     (O1 -m-> O2 -m-> O3) → (O1 -m-> O2 -m-> O3).
Defined.

Lemma Imon2_simpl : ∀ `{o1: ord O1} `{o2: ord O2} `{o3:ord O3}
      (f:O1 -m-> O2 -m-> O3) (x:O1) (y: O2),
      Imon2 f x y = f x y.

Lemma Imonotonic2 `{o1: ord O1} `{o2: ord O2} `{o3:ord O3}
      (f:O1 → O2 → O3){mf : monotonic2 f}
      : monotonic2 (o1:=Iord O1) (o2:=Iord O2) (o3:=Iord O3) f.

Hint Extern 2 (@monotonic2 _ (Iord _) _ (Iord _) _ (Iord _) _) ⇒ apply @Imonotonic2
  : typeclass_instances.

Definition imon2 `{o1: ord O1} `{o2: ord O2} `{o3:ord O3}
      (f:O1 → O2 → O3){mf : monotonic2 f} : O1 -m-> O2 -m-> O3 :=
      mon2 (o1:=Iord O1) (o2:=Iord O2) (o3:=Iord O3) f.

Lemma imon2_simpl : ∀ `{o1: ord O1} `{o2: ord O2} `{o3:ord O3}
      (f:O1 → O2 → O3){mf : monotonic2 f} (x:O1) (y:O2),
      imon2 f x y = f x y.
```

3.3 Sequences

3.3.1 Order on natural numbers

- *natO n m* = $n \leq m$

```
Instance natO : ord nat :=
    { Oeq := fun n m : nat ⇒ n = m;
```

Ole := `fun` $n\ m : nat \Rightarrow (n \leq m)$%nat}.
`Defined`.

`Lemma` $le\_Ole : \forall\ n$ **m**, $((n \leq m)$%nat$)$-> $n \leq m$.
`Hint Resolve` le_Ole.

`Lemma` $nat\_monotonic : \forall\ \{O\}\ \{o$:ord $O\}$
$(f$:nat $\to O)$, $(\forall\ n, f\ n \leq f\ (S\ n)) \to monotonic\ f$.
`Hint Resolve` @$nat\_monotonic$.

`Definition` fnatO_intro : $\forall\ \{O\}\ \{o$:ord $O\}\ (f$:nat $\to O)$, $(\forall\ n, f\ n \leq$ f $(S\ n)) \to nat \xrightarrow{m} O$.
`Defined`.

`Lemma` $fnatO\_elim : \forall\ \{O\}\ \{o$:ord $O\}\ (f$:nat $\xrightarrow{m} O)\ (n$:nat$), f\ n \leq f\ (S\ n)$.
`Hint Resolve` @$fnatO\_elim$.

- (mseq_lift_left f n) k = f (n+k)

`Definition` $seq\_lift\_left\ \{O\}\ (f$:nat $\to O)\ n :=$ `fun` $k \Rightarrow f\ (n$+k$)$%nat.

`Instance` $mon\_seq\_lift\_left$
: $\forall\ \{O\}\ \{o$:ord $O\}\ (f$:nat $\to O)\ \{m$:monotonic **f**$\}$, $monotonic\ (seq\_lift\_left\ f\ n)$.
`Save`.

`Definition` mseq_lift_left : $\forall\ \{O\}\ \{o$:ord $O\}\ (f$:nat $\xrightarrow{m} O)\ (n$:nat$), nat \xrightarrow{m} O$.
`Defined`.

`Lemma` $mseq\_lift\_left\_simpl : \forall\ \{O\}\ \{o$:ord $O\}\ (f$:nat $\xrightarrow{m}$ **O**$)\ (n\ k$:nat$)$,
$mseq\_lift\_left\ f\ n\ k = f\ (n$+k$)$%nat.

`Lemma` $mseq\_lift\_left\_le\_compat : \forall\ \{O\}\ \{o$:ord $O\}\ (f\ g$:nat $\xrightarrow{m} O)\ (n$:nat$)$,
$f \leq g \to mseq\_lift\_left\ f$ **n** $\leq mseq\_lift\_left\ g\ n$.
`Hint Resolve` @$mseq\_lift\_left\_le\_compat$.

`Add` $Parametric\ Morphism\ \{O\}\ \{o$:ord $O\}$: (@$mseq\_lift\_left$ _ o)
`with` $signature\ Oeq \Longrightarrow eq \Longrightarrow Oeq$
`as` $mseq\_lift\_left\_eq\_compat$.
`Save`.
`Hint Resolve` @$mseq\_lift\_left\_eq\_compat$.

`Add` $Parametric\ Morphism\ \{O\}\ \{o$:ord $O\}$: (@$seq\_lift\_left\ O$)
`with` $signature\ Oeq \Longrightarrow eq \Longrightarrow Oeq$
`as` $seq\_lift\_left\_eq\_compat$.
`Save`.
`Hint Resolve` @$seq\_lift\_left\_eq\_compat$.

- (mseq_lift_right f n) k = f (k+n)

`Definition` **seq_lift_right** $\{O\}\ (f$:nat $\to O)\ n :=$ `fun` $k \Rightarrow f\ (k$+n$)$%nat.

`Instance` $mon\_seq\_lift\_right$
: $\forall\ \{O\}\ \{o$:ord $O\}\ (f$:nat $\to O)\ \{m$:monotonic $f\}$, $monotonic\ (seq\_lift\_right\ f$ **n**$)$.
`Save`.

`Definition` $mseq\_lift\_right : \forall\ \{O\}\ \{o$:ord $O\}\ (f$:nat $\xrightarrow{m} O)\ (n$:nat$), nat \xrightarrow{m} O$.
`Defined`.

`Lemma` $mseq\_lift\_right\_simpl : \forall\ \{O\}\ \{o$:ord $O\}$ (**f**:nat $\xrightarrow{m} O)\ (n\ k$:nat$)$,
$mseq\_lift\_right\ f\ n\ k = f\ (k$+n$)$%nat.

`Lemma` $mseq\_lift\_right\_le\_compat : \forall\ \{O\}\ \{o$:ord $O\}\ (f\ g$:nat $\xrightarrow{m} O)\ (n$:nat$)$,
$f \leq g \to mseq\_lift\_right\ f\ n \leq mseq\_lift\_right\ g\ n$.
`Hint Resolve` @$mseq\_lift\_right\_le\_compat$.

`Add` $Parametric\ Morphism\ \{O\}\ \{o$:ord $O\}$: $(mseq\_lift\_right\ (o$:=o$))$

```
  with signature Oeq ==> eq ==> Oeq
  as mseq_lift_right_eq_compat.
Save.

Add Parametric Morphism {O} {o:ord O}: (@seq_lift_right O)
  with signature Oeq ==> eq ==> Oeq
  as seq_lift_right_eq_compat.
Save.
Hint Resolve @seq_lift_right_eq_compat.

Lemma mseq_lift_right_left : ∀ {O} {o:ord O} (f:nat -m-> O) n,
      mseq_lift_left f n == mseq_lift_right f n.
```

3.3.2 Monotonicity and functions

- (shift f x) n = f n x

```
Instance shift_mon_fun {A} `{o1:ord O1} `{o2:ord O2} (f:O1 -m-> (A → O2)) :
      ∀ x:A, monotonic (fun (y:O1) ⇒ f y x).
Save.

Definition shift {A} `{o1:ord O1} `{o2:ord O2} (f:O1 -m-> (A → O2)) : A → O1 -m-> O2
  := fun x ⇒ (mon (fun y ⇒ f y x)).

Infix "◇" := shift (at level 30, no associativity) : O_scope.

Lemma shift_simpl : ∀ {A} `{o1:ord O1} `{o2:ord O2} (f:O1 -m-> (A → O2)) x y,
      (f ◇ x) y = f y x.

Lemma shift_le_compat : ∀ {A} `{o1:ord O1} `{o2:ord O2} (f g:O1 -m-> (A → O2)),
            f ≤ g → shift f ≤ shift g.
Hint Resolve @shift_le_compat.

Add Parametric Morphism {A} `{o1:ord O1} `{o2:ord O2}
    : (shift (A:=A) (O1:=O1) (O2:=O2)) with signature Oeq ==> eq ==> Oeq
as shift_eq_compat.
Save.

Instance ishift_mon {A} `{o1:ord O1} `{o2:ord O2} (f:A → (O1 -m-> O2)) :
      monotonic (fun (y:O1) (x:A) ⇒ f x y).
Save.

Definition ishift {A} `{o1:ord O1} `{o2:ord O2} (f:A → (O1 -m-> O2)) : O1 -m-> (A → O2)
  := mon (fun (y:O1) (x:A) ⇒ f x y) (fmonotonic:=ishift_mon f).

Lemma ishift_simpl : ∀ {A} `{o1:ord O1} `{o2:ord O2} (f:A → (O1 -m-> O2)) x y,
      ishift f x y = f y x.

Lemma ishift_le_compat : ∀ {A} `{o1:ord O1} `{o2:ord O2} (f g:A → (O1 -m-> O2)),
            f ≤ g → ishift f ≤ ishift g.
Hint Resolve @ishift_le_compat.

Add Parametric Morphism {A} `{o1:ord O1} `{o2:ord O2}
    : (ishift (A:=A) (O1:=O1) (O2:=O2)) with signature Oeq ==> eq ==> Oeq
as ishift_eq_compat.
Save.

Instance shift_fun_mon `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (f:O1 -m-> (O2 → O3))
     {m:forall x, monotonic (f x)} : monotonic (shift f).
Save.

Instance shift_mon2 `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (f:O1 -m-> O2 -m-> O3)
     : monotonic2 (fun x y ⇒ f y x).
```

```
Save.
Hint Resolve @shift_mon_fun @shift_fun_mon @shift_mon2.
Definition mshift `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (f:O1 -m> O2 -m> O3)
    : O2 -m> O1 -m> O3 := mon2 (fun x y => f y x).
```

- id c = c

```
Definition id O {o:ord O} : O → O := fun x => x.
Instance mon_id : ∀ {O:Type} {o:ord O}, monotonic (id O).
Save.
```

- (cte c) n = c

```
Definition cte A `{o1:ord O1} (c:O1) : A → O1 := fun x => c.
Instance mon_cte : ∀ `{o1:ord O1} `{o2:ord O2} (c:O2), monotonic (cte O1 c).
Save.
Definition mseq_cte {O} {o:ord O} (c:O) : nat -m> O := mon (cte nat c).
Add Parametric Morphism `{o1:ord O1} `{o2:ord O2} : (@cte O1 O2 _)
  with signature Ole ==> Ole as cte_le_compat.
Save.
Add Parametric Morphism `{o1:ord O1} `{o2:ord O2} : (@cte O1 O2 _)
  with signature Oeq ==> Oeq as cte_eq_compat.
Save.
Instance mon_diag `{o1:ord O1} `{o2:ord O2}(f:O1 -m> (O1 -m> O2))
    : monotonic (fun x => f x x).
Save.
Hint Resolve @mon_diag.
Definition diag `{o1:ord O1} `{o2:ord O2}(f:O1 -m> (O1 -m> O2)) : O1 -m> O2
    := mon (fun x => f x x).
Lemma fmon_diag_simpl : ∀ `{o1:ord O1} `{o2:ord O2} (f:O1 -m> (O1 -m> O2)) (x:O1),
              diag f x = f x x.
Lemma diag_le_compat : ∀ `{o1:ord O1} `{o2:ord O2} (f g:O1 -m> (O1 -m> O2)),
             f ≤ g → diag f ≤ diag g.
Hint Resolve @diag_le_compat.
Add Parametric Morphism `{o1:ord O1} `{o2:ord O2} : (diag (O1:=O1) (O2:=O2))
   with signature Oeq ==> Oeq as diag_eq_compat.
Save.
Lemma diag_shift : ∀ `{o1:ord O1} `{o2:ord O2} (f: O1 -m> O1 -m> O2),
                    diag f == diag (mshift f).
Hint Resolve @diag_shift.
Lemma mshift_simpl : ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3}
      (h:O1 -m> O2 -m> O3) (x : O2) (y:O1), mshift h x y = h y x.
Lemma mshift_le_compat : ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3}
      (f g:O1 -m> O2 -m> O3), f ≤ g → mshift f ≤ mshift g.
Hint Resolve @mshift_le_compat.
Add Parametric Morphism `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} : (@mshift O1 _ O2 _ O3 _)
     with signature Oeq ==> Oeq as mshift_eq_compat.
Save.
Lemma mshift2_eq : ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (h : O1 -m> O2 -m> O3),
             mshift (mshift h) == h.
```

- (f@g) x = f (g x)

```
Instance monotonic_comp `{o1:ord O1} `{o2:ord O2} `{o3:ord O3}
  (f:O2 → O3){mf : monotonic f} (g:O1 → O2){mg:monotonic g} : monotonic (fun x ⇒ f (g x)).
Save.
Hint Resolve @monotonic_comp.

Instance monotonic_comp_mon `{o1:ord O1} `{o2:ord O2} `{o3:ord O3}
  (f:O2 -m> O3)(g:O1 -m> O2) : monotonic (fun x ⇒ f (g x)).
Save.
Hint Resolve @monotonic_comp_mon.

Definition comp `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (f:O2 -m> O3) (g:O1 -m> O2)
      : O1 -m> O3 := mon (fun x ⇒ f (g x)).

Infix "@" := comp (at level 35) : O_scope.

Lemma comp_simpl : ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3}
      (f:O2 -m> O3) (g:O1 -m> O2) (x:O1), (f@g) x = f (g x).

Add Parametric Morphism `{o1:ord O1} `{o2:ord O2} `{o3:ord O3}: (@comp O1 _ O2 _ O3 _)
    with signature Ole ++> Ole ++> Ole
    as comp_le_compat.
Save.

Hint Immediate @comp_le_compat.

Add Parametric Morphism `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} : (@comp O1 _ O2 _ O3 _)
    with signature Oeq ==>Oeq ==>Oeq
    as comp_eq_compat.
Save.

Hint Immediate @comp_eq_compat.
```

- (f@2 g) h x = f (g x) (h x)

```
Instance mon_app2 `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} `{o4:ord O4}
      (f:O2 → O3 → O4) (g:O1 → O2) (h:O1 → O3)
      {mf:monotonic2 f}{mg:monotonic g} {mh:monotonic h}
      : monotonic (fun x ⇒ f (g x) (h x)).
Save.

Instance mon_app2_mon `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} `{o4:ord O4}
      (f:O2 -m> O3 -m> O4) (g:O1 -m> O2) (h:O1 -m> O3)
      : monotonic (fun x ⇒ f (g x) (h x)).
Save.

Definition app2 `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} `{o4:ord O4}
        (f:O2 -m> O3 -m> O4) (g:O1 -m> O2) (h:O1 -m> O3) : O1 -m> O4
        := mon (fun x ⇒ f (g x) (h x)).

Infix "@2" := app2 (at level 70) : O_scope.

Add Parametric Morphism `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} `{o4:ord O4}:
        (@app2 O1 _ O2 _ O3 _ O4 _)
    with signature Ole ++> Ole ++> Ole ++> Ole
    as app2_le_compat.
Save.

Hint Immediate @app2_le_compat.

Add Parametric Morphism `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} `{o4:ord O4}:
        (@app2 O1 _ O2 _ O3 _ O4 _)
```

```
    with signature Oeq ⟹Oeq ⟹Oeq ⟹Oeq
    as app2_eq_compat.
Save.

Hint Immediate @app2_eq_compat.

Lemma app2_simpl :
    ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} `{o4:ord O4}
          (f:O2 -m-> O3 -m-> O4) (g:O1 -m-> O2) (h:O1 -m-> O3) (x:O1),
    (f@2 g) h x = f (g x) (h x).

Lemma comp_monotonic_right :
      ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (f: O2 -m-> O3) (g1 g2:O1 -m-> O2),
            g1≤ g2 → f @ g1 ≤ f @ g2.
Hint Resolve @comp_monotonic_right.

Lemma comp_monotonic_left :
      ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} (f1 f2: O2 -m-> O3) (g:O1 -m-> O2),
            f1≤ f2 → f1 @ g ≤ f2 @ g.
Hint Resolve @comp_monotonic_left.

Instance comp_monotonic2 : ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3},
            monotonic2 (@comp O1 _ O2 _ O3 _).
Save.
Hint Resolve @comp_monotonic2.

Definition fcomp `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} :
   (O2 -m-> O3) -m-> (O1 -m-> O2) -m-> (O1 -m-> O3) := mon2 (@comp O1 _ O2 _ O3 _).

Implicit Arguments fcomp [[o1] [o2] [o3]].

Lemma fcomp_simpl : ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3}
      (f:O2 -m-> O3) (g:O1 -m-> O2), fcomp _ _ _ f g = f@g.

Definition fcomp2 `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} `{o4:ord O4} :
        (O3 -m-> O4) -m-> (O1 -m-> O2 -m-> O3) -m-> (O1 -m-> O2 -m-> O4):=
        (fcomp O1 (O2 -m-> O3) (O2 -m-> O4))@(fcomp O2 O3 O4).

Implicit Arguments fcomp2 [[o1] [o2] [o3] [o4]].

Lemma fcomp2_simpl : ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3} `{o4:ord O4}
      (f:O3 -m-> O4) (g:O1 -m-> O2 -m-> O3) (x:O1)(y:O2), fcomp2 _ _ _ _ f g x y = f (g x y).

Lemma fmon_le_compat2 : ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3}
      (f: O1 -m-> O2 -m-> O3) (x y:O1) (z t:O2), x<=y → z <=t → f x z ≤ f y t.
Hint Resolve fmon_le_compat2.

Lemma fmon_cte_comp : ∀ `{o1:ord O1} `{o2:ord O2} `{o3:ord O3}
      (c:O3)(f:O1 -m-> O2), (mon (cte O2 c)) @ f == mon (cte O1 c).
```

3.4 Basic operators of omega-cpos

- Constant : 0
- lub : limit of monotonic sequences

3.4.1 Definition of cpos

```
Class cpo `{o:ord D} : Type := mk_cpo
  {D0 : D; lub: ∀ (f:nat -m-> D), D;
   Dbot : ∀ x:D, D0 ≤ x;
   le_lub : ∀ (f : nat -m-> D) (n:nat), f n ≤ lub f;
```

$lub\_le : \forall\ (f : nat \xrightarrow{m} D)\ (x{:}\mathrm{D}),\ (\forall\ n, f\ n \leq x) \to lub\ f \leq x\}.$

```
Implicit Arguments cpo [[o]].

Notation "0" := D0 : O_scope.

Hint Resolve @Dbot @le_lub @lub_le.

Definition mon_ord_equiv : ∀ `{o:ord D1} `{o1:ord D2} {o2:ord D2},
      eq_ord o1 o2 → fmon D1 D2 (o2:=o2) → fmon D1 D2 (o2:=o1).
Defined.

Lemma mon_ord_equiv_simpl : ∀ `{o:ord D1} `{o1:ord D2} {o2:ord D2}
      (H:eq_ord o1 o2) (f:fmon D1 D2 (o2:=o2)) (x:D1),
      mon_ord_equiv H f x = f x.

Definition cpo_ord_equiv `{o1:ord D} (o2:ord D)
       : eq_ord o1 o2 → cpo (o:=o1) D → cpo (o:=o2) D.
Defined.
```

3.4.2 Least upper bounds

```
Add Parametric Morphism `{c:cpo D} : (lub (cpo:=c))
             with signature Ole ++> Ole as lub_le_compat.
Save.
Hint Resolve @lub_le_compat.

Add Parametric Morphism `{c:cpo D}: (lub (cpo:=c))
      with signature Oeq ==>Oeq as lub_eq_compat.
Save.
Hint Resolve @lub_eq_compat.

Notation "'mlub' f" := (lub (mon f)) (at level 60) : O_scope .

Lemma mlub_le_compat : ∀ `{c:cpo D} (f g:nat → D) {mf:monotonic f} {mg:monotonic g},
                f ≤ g → mlub f ≤ mlub g.
Hint Resolve @mlub_le_compat.

Lemma mlub_eq_compat : ∀ `{c:cpo D} (f g:nat → D) {mf:monotonic f} {mg:monotonic g},
                f == g → mlub f == mlub g.
Hint Resolve @mlub_eq_compat.

Lemma le_mlub : ∀ `{c:cpo D} (f:nat → D) {m:monotonic f} (n:nat), f n ≤ mlub f.

Lemma mlub_le : ∀ `{c:cpo D}(f:nat → D) {m:monotonic f}(x:D), (∀ n, f n ≤ x) → mlub f ≤ x.
Hint Resolve @le_mlub @mlub_le.

Instance lub_mon `{c:cpo D} : monotonic lub.
Save.

Definition Lub `{c:cpo D} : (nat -m-> D) -m-> D := mon lub.

Instance monotonic_lub_comp {O} {o:ord O} `{c:cpo D} (f:O → nat → D){mf:monotonic2 f}:
         monotonic (fun x ⇒ mlub (f x)).
Save.

Lemma lub_cte : ∀ `{c:cpo D} (d:D), mlub (cte nat d) == d.

Hint Resolve @lub_cte.

Lemma mlub_lift_right : ∀ `{c:cpo D} (f:nat -m-> D) n,
      lub f == mlub (seq_lift_right f n).
Hint Resolve @mlub_lift_right.

Lemma mlub_lift_left : ∀ `{c:cpo D} (f:nat -m-> D) n,
      lub f == mlub (seq_lift_left f n).
Hint Resolve @mlub_lift_left.
```

```
Lemma lub_lift_right : ∀ `{c:cpo D} (f:nat -m-> D) n,
      lub f == lub (mseq_lift_right f n).
Hint Resolve @lub_lift_right.

Lemma lub_lift_left : ∀ `{c:cpo D} (f:nat -m-> D) n,
      lub f == lub (mseq_lift_left f n).
Hint Resolve @lub_lift_left.

Lemma lub_le_lift : ∀ `{c:cpo D} (f g:nat -m-> D)
      (n:nat), (∀ k, n ≤ k → f k ≤ g k) → lub f ≤ lub g.

Lemma lub_eq_lift : ∀ `{c:cpo D} (f g:nat -m-> D) {m:monotonic f} {m':monotonic g}
      (n:nat), (∀ k, n ≤ k → f k == g k) → lub f == lub g.
```

- (lub_fun h) x = lub_n (h n x)

```
Definition lub_fun {A} `{c:cpo D} (h : nat -m-> (A → D)) : A → D
                := fun x ⇒ mlub (h ◇ x).

Instance lub_shift_mon {O} {o:ord O} `{c:cpo D} (h : nat -m-> (O -m-> D))
          : monotonic (fun (x:O) ⇒ lub (mshift h x)).
Save.
Hint Resolve @lub_shift_mon.
```

3.4.3 Functional cpos

```
Instance fcpo {A: Type} `(c:cpo D) : cpo (A → D) :=
   {D0 := fun x:A ⇒ (0:D);
    lub := fun f ⇒ lub_fun f}.
Defined.

Lemma fcpo_lub_simpl : ∀ {A} `{c:cpo D} (h:nat -m-> (A → D))(x:A),
      (lub h) x = mlub (h ◇ x).

Lemma lub_ishift : ∀ {A} `{c:cpo D} (h:A → (nat -m-> D)),
       lub (ishift h) == fun x ⇒ lub (h x).
```

3.5 Cpo of monotonic functions

```
Instance fmon_cpo {O} {o:ord O} `{c:cpo D} : cpo (O -m-> D) :=
     { D0 := mon (cte O (0:D));
       lub := fun h:nat -m-> (O -m-> D) ⇒ mon (fun (x:O) ⇒ lub (cpo:=c) (mshift h x))}.
Defined.

Lemma fmon_lub_simpl : ∀ {O} {o:ord O} `{c:cpo D}
      (h:nat -m-> (O -m-> D))(x:O), (lub h) x = lub (mshift h x).
Hint Resolve @fmon_lub_simpl.

Instance mon_fun_lub : ∀ {O} {o:ord O} `{c:cpo D}
         (h:nat -m-> (O → D)) {mh:forall n, monotonic (h n)}, monotonic (lub h).
Save.
```

Link between lubs on ordinary functions and monotonic functions

```
Lemma lub_mon_fcpo : ∀ {O} {o:ord O} `{c:cpo D} (h:nat -m-> (O -m-> D)),
      lub h == mon (lub (mfun2 h)).

Lemma lub_fcpo_mon : ∀ {O} {o:ord O} `{c:cpo D} (h:nat -m-> (O → D))
     {mh:forall x, monotonic (h x)}, lub h == lub (mon2 h).

Lemma double_lub_diag : ∀ `{c:cpo D} (h : nat -m-> nat -m-> D),
```

$lub\ (lub\ h) == lub\ (diag\ h)$.
Hint Resolve @*double_lub_diag*.

Lemma *double_lub_shift* : $\forall$ `{*c*:cpo D} $(h : nat \xrightarrow{m} nat \xrightarrow{m} D)$,
$lub\ (lub\ h) == lub\ (lub\ (mshift\ h))$.
Hint Resolve @*double_lub_shift*.

3.6 Continuity

Lemma *lub_comp_le* :
$\forall$ `{*c1*:cpo *D1*} `{*c2*:cpo *D2*} (*f*:D1 $\xrightarrow{m}$ *D2*) $(h : nat \xrightarrow{m} D1)$,
$lub\ (f\ @\ h) \leq f\ (lub\ h)$.
Hint Resolve @**lub_comp_le**.

Lemma *lub_app2_le* : $\forall$ `{*c1*:cpo *D1*} `{*c2*:cpo *D2*} `{*c3*:cpo *D3*}
(*F*:D1 $\xrightarrow{m}$ *D2* $\xrightarrow{m}$ *D3*) $(f : nat \xrightarrow{m} D1)$ (*g*: $nat \xrightarrow{m} D2$),
$lub\ ((F\ @2\ f)\ g) \leq F\ (lub\ \mathbf{f})\ (lub\ g)$.
Hint Resolve @lub_app2_le.

Class *continuous* `{**c1**:cpo *D1*} `{*c2*:cpo *D2*} (*f*:D1 $\xrightarrow{m}$ *D2*) :=
cont_intro : $\forall\ (h : nat \xrightarrow{m} D1), f\ (lub\ h) \leq lub\ (f\ @\ h)$.

Type*classes Opaque continuous*.

Lemma *continuous_eq_compat* : $\forall$ `{*c1*:cpo *D1*} `{*c2*:cpo *D2*}(*f g*:D1 $\xrightarrow{m}$ *D2*),
$f == g \rightarrow continuous\ f \rightarrow continuous\ g$.

Add *Parametric Morphism* `{*c1*:cpo *D1*} `{*c2*:cpo *D2*} : (@*continuous D1 _ _ D2 _ _*)
with *signature Oeq* $\Longrightarrow$*iff*
as *continuous_eq_compat_iff*.
Save.

Lemma *lub_comp_eq* :
$\forall$ `{*c1*:cpo *D1*} `{*c2*:cpo *D2*} (*f*:D1 $\xrightarrow{m}$ *D2*) $(h : nat \xrightarrow{m} D1)$,
$continuous\ f \rightarrow f\ (lub\ h) == lub\ (\mathbf{f}\ @\ h)$.
Hint Resolve @*lub_comp_eq*.

- mon0 x == 0

Instance *cont0* `{*c1*:cpo *D1*} `{*c2*:cpo *D2*} : *continuous* (*mon* (*cte D1* (0:D2))).
Save.
Implicit Arguments *cont0* [].

- double_app f g n m = f m (g n)

Definition *double_app* `{*o1*:ord *O1*} `{*o2*:ord *O2*} `{*o3*:ord *O3*} `{*o4*: *ord O4*}
(*f*:O1 $\xrightarrow{m}$ *O3* $\xrightarrow{m}$ *O4*) (*g*:O2 $\xrightarrow{m}$ *O3*)
: *O2* $\xrightarrow{m}$ (*O1* $\xrightarrow{m}$ *O4*) := *mon* ((*mshift f*) @ *g*).

3.6.1 Continuity

Class *continuous2* `{*c1*:cpo *D1*} `{*c2*:cpo *D2*} `{*c3*:cpo *D3*} (*F*:D1 $\xrightarrow{m}$ *D2* $\xrightarrow{m}$ *D3*) :=
continuous2_intro : $\forall\ (f : nat \xrightarrow{m} D1)\ (g$:nat $\xrightarrow{m} D2)$,
$F\ (lub\ f)\ (lub\ g) \leq lub\ ((F\ @2\ f)\ g)$.

Lemma *continuous2_app* : $\forall$ `{*c1*:cpo *D1*} `{*c2*:cpo *D2*} `{*c3*:cpo *D3*}
$(F : D1 \xrightarrow{m} D2 \xrightarrow{m} D3)$ {*cF*:continuous2 *F*} (*k*:D1), *continuous* (*F k*).

Type*classes Opaque continuous2*.

Lemma *continuous2_eq_compat* :

$\forall$ '{$c1$:cpo D1} '{$c2$:cpo $D2$} '{$c3$:cpo $D3$} (f g : $D1 \xrightarrow{m} D2 \xrightarrow{m} D3$),
$f == g \to$ *continuous2* $f \to$ *continuous2* g.

Lemma *continuous2_continuous* : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} '{$c3$:cpo $D3$}
($F : D1 \xrightarrow{m}$ D2 $\xrightarrow{m} D3$), *continuous2* F $\to$ *continuous* F.
Hint Immediate @*continuous2_continuous*.

Lemma *continuous2_left* : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} '{$c3$:cpo $D3$}
($F : D1 \xrightarrow{m} D2 \xrightarrow{m} D3$) ($h$:nat $\xrightarrow{m} D1$) (x:D2),
continuous $F \to F$ (*lub* h) $x \leq$ **lub** (*mshift* (F @ h) x).

Lemma *continuous2_right* : $\forall$ '{c1:cpo $D1$} '{$c2$:cpo $D2$} '{$c3$:cpo $D3$}
($F : D1 \xrightarrow{m} D2 \xrightarrow{m} D3$) ($x$:D1)($h$:nat $\xrightarrow{m} D2$),
continuous2 $F \to F$ x (*lub* h) $\leq$ *lub* (F x @ h).

Lemma *continuous_continuous2* : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} '{$c3$:cpo $D3$}
(F : D1 $\xrightarrow{m} D2 \xrightarrow{m} D3$) ($cFr$: $\forall$ k:D1, *continuous* (F k)) (cF: *continuous* F),
continuous2 F.

Hint Resolve @*continuous2_app* @*continuous2_continuous* @*continuous_continuous2*.

Lemma **lub_app2_eq** : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} '{$c3$:cpo $D3$}
($F : D1 \xrightarrow{m} D2 \xrightarrow{m} D3$) {$cFr$:forall k:D1, *continuous* (F k)} {cF : *continuous* F},
$\forall$ (f:nat $\xrightarrow{m} D1$) (g:nat $\xrightarrow{m} D2$),
F (*lub* f) (*lub* g) == *lub* ((F@2 f) g).

Lemma *lub_cont2_app2_eq* : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} '{$c3$:cpo $D3$}
($F : D1 \xrightarrow{m} D2 \xrightarrow{m} D3$){$cF$: *continuous2* F},
$\forall$ (f:nat $\xrightarrow{m} D1$) (g:nat $\xrightarrow{m} D2$),
F (*lub* f) (*lub* g) == *lub* ((F@2 f) g).

Lemma *mshift_continuous2* : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} '{$c3$:cpo $D3$}
($F : D1 \xrightarrow{m} D2 \xrightarrow{m} D3$), *continuous2* $F \to$ *continuous2* (*mshift* F).
Hint Resolve @*mshift_continuous2*.

Lemma *monotonic_sym* : $\forall$ '{$o1$:ord $D1$} '{$o2$:ord $D2$} ($F : D1 \to D1 \to D2$),
($\forall$ x y, F x y == F y x) $\to$ ($\forall$ k:D1, *monotonic* (F k)) $\to$ *monotonic* F.
Hint Immediate @*monotonic_sym*.

Lemma *monotonic2_sym* : $\forall$ '{$o1$:ord $D1$} '{$o2$:ord $D2$} ($F : D1 \to D1 \to D2$),
($\forall$ x y, F x y == F y x) $\to$ ($\forall$ k:D1, *monotonic* (F k)) $\to$ *monotonic2* F.
Hint Immediate @*monotonic2_sym*.

Lemma *continuous_sym* : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} ($F : D1 \xrightarrow{m} D1 \xrightarrow{m} D2$),
($\forall$ x y, F x y == F y x) $\to$ ($\forall$ k:D1, *continuous* (F k)) $\to$ *continuous* F.

Lemma *continuous2_sym* : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} (F : $D1$ -m>D1 -m>D2),
($\forall$ x y, F x y == F y x) $\to$ ($\forall$ k, *continuous* (F k)) $\to$ *continuous2* F.
Hint Resolve @*continuous2_sym*.

- continuity is preserved by composition

Lemma *continuous_comp* : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} '{$c3$:cpo $D3$}
(f:D2 $\xrightarrow{m} D3$)(g:D1 $\xrightarrow{m} D2$), *continuous* $f \to$ *continuous* $g \to$ *continuous* (*mon* (f@g)).
Hint Resolve @*continuous_comp*.

Lemma *continuous2_comp* : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} '{c3:cpo $D3$} '{$c4$:cpo $D4$}
(f:D1 $\xrightarrow{m} D2$)(g:D2 $\xrightarrow{m} D3 \xrightarrow{m} D4$),
continuous $f \to$ *continuous2* $g \to$ *continuous2* (g @ f).
Hint Resolve @*continuous2_comp*.

Lemma *continuous2_comp2* : $\forall$ '{$c1$:cpo $D1$} '{$c2$:cpo $D2$} '{$c3$:cpo $D3$} '{$c4$:cpo $D4$}
(f:D3 $\xrightarrow{m} D4$)(g:D1 $\xrightarrow{m} D2 \xrightarrow{m} D3$),

continuous f → *continuous2 g* → continuous2 (*fcomp2 D1 D2 D3 D4 f g*).
Hint Resolve @**continuous2_comp2**.

3.7 Cpo of continuous functions

Instance *lub_continuous* '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} '{*c3*:cpo *D3*}
(*f*:nat $\xrightarrow{m}$ (*D1* $\xrightarrow{m}$ *D2*)) {*cf*:forall *n*, *continuous* (*f n*)}
: *continuous* (*lub f*).
Save.

Record *fcont* '{*c1*:cpo *D1*} '{*c2*:cpo *D2*}: Type
:= *cont* {*fcontm* :> *D1* $\xrightarrow{m}$ *D2*; *fcontinuous* : *continuous fcontm*}.
Hint Resolve @*fcontinuous*.
Implicit Arguments *fcont* [[*o*][*c1*] [*o0*][*c2*]].
Implicit Arguments *cont* [[*D1*][*o*][*c1*] [*D2*][*o0*][*c2*] [*fcontinuous*]].

Infix "-c>" := *fcont* (at *level* 30, *right associativity*) : *O_scope*.

Definition *fcont_fun* '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} (*f*:D1 -c> *D2*) : *D1* → *D2* := fun *x* ⇒ *f x*.

Instance *fcont_ord* '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} : *ord* (*D1*-c>D2)
:= {*Oeq* := fun *f g* ⇒ ∀ x, *f x* == *g x*; *Ole* := fun *f g* ⇒ ∀ *x*, *f x* ≤ *g x*}.
Defined.

Lemma *fcont_le_intro* : ∀ '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} (*f g* : *D1* -c> *D2*),
(∀ *x*, *f x* ≤ *g x*) → *f* ≤ *g*.

Lemma *fcont_le_elim* : ∀ '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} (*f g* : *D1* -c> *D2*),
f ≤ *g* → ∀ *x*, *f x* ≤ *g x*.

Lemma *fcont_eq_intro* : ∀ '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} (*f g* : *D1* -c> *D2*),
(∀ *x*, *f x* == *g x*) → *f* == *g*.

Lemma *fcont_eq_elim* : ∀ '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} (*f g* : *D1* -c> *D2*),
f == *g* → ∀ *x*, *f x* == *g x*.

Lemma *fcont_le* : ∀ '{*c1*:cpo *D1*} '{*c2*:cpo D2} (*f* : *D1* -c> *D2*) (*x y* : *D1*),
x ≤ *y* → *f x* ≤ *f y*.
Hint Resolve @*fcont_le*.

Lemma *fcont_eq* : ∀ '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} (*f* : *D1* -c> *D2*) (*x y* : *D1*),
x == *y* → *f x* == *f y*.
Hint Resolve @*fcont_eq*.

Definition *fcont0 D1* '{*c1*:cpo *D1*} *D2* '{*c2*:cpo D2} : *D1* -c> *D2* := *cont* (*mon* (*cte D1* (0:D2))).

Instance *fcontm_monotonic* : ∀ '{*c1*:cpo *D1*} '{*c2*:cpo *D2*},
monotonic (*fcontm* (*D1*:=D1) (*D2*:=D2)).
Save.

Definition *Fcontm D1* '{*c1*:cpo *D1*} *D2* '{*c2*:cpo *D2*} : (**D1** -c> *D2*) $\xrightarrow{m}$ (*D1* $\xrightarrow{m}$ *D2*) :=
mon (*fcontm* (D1:=D1) (*D2*:=D2)).

Instance *fcont_lub_continuous* :
∀ '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} (*f*:nat $\xrightarrow{m}$ (*D1* -c> *D2*)),
continuous (*lub* (**D**:=D1 $\xrightarrow{m}$ *D2*) (*Fcontm D1 D2* @ *f*)).
Save.

Definition *fcont_lub* '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} : (*nat* $\xrightarrow{m}$ (*D1* -c> *D2*)) → *D1* -c> *D2* :=
fun *f* ⇒ *cont* (lub (D:=D1 $\xrightarrow{m}$ *D2*) (*Fcontm D1 D2* @ *f*)).

Instance *fcont_cpo* '{*c1*:cpo *D1*} '{*c2*:cpo *D2*} : *cpo* (*D1*-c> *D2*) :=
{*D0*:=fcont0 *D1 D2*; *lub*:=fcont_lub (*D1*:=D1) (*D2*:=D2)}.
Defined.

```
Definition fcont_app {O} {o:ord O} `{c1:cpo D1} `{c2:cpo D2} (f: O -m> D1 -c> D2) (x:D1) : O -m> D2
        := mshift (Fcontm D1 D2 @ f) x.

Infix "⟹" := fcont_app (at level 70) : O_scope.

Lemma fcont_app_simpl : ∀ {O} {o:ord O} `{c1:cpo D1} `{c2:cpo D2} (f: O -m> D1 -c> D2)(x:D1)(y:O),
            (f ⟹ x) y = f y x.

Instance ishift_continuous :
  ∀ {A:Type} `{c1:cpo D1} `{c2:cpo D2} (f: A → (D1 -c> D2)),
          continuous (ishift f).
Qed.

Definition fcont_ishift {A:Type} `{c1:cpo D1} `{c2:cpo D2} (f: A → (D1 -c> D2))
        : D1 -c> (A → D2) := cont _ (fcontinuous:=ishift_continuous f).

Instance mshift_continuous : ∀ {O} {o:ord O} `{c1:cpo D1} `{c2:cpo D2} (f: O -m> (D1 -c> D2)),
        continuous (mshift (Fcontm D1 D2 @ f)).
Save.

Definition fcont_mshift {O} {o:ord O} `{c1:cpo D1} `{c2:cpo D2} (f: O -m> (D1 -c> D2))
  : D1 -c> O -m> D2 := cont (mshift (Fcontm D1 D2 @ f)).

Lemma fcont_app_continuous :
       ∀ {O} {o:ord O} `{c1:cpo D1} `{c2:cpo D2} (f: O -m> D1 -c> D2) (h:nat -m> D1),
            f ⟹ (lub h) ≤ lub (D:=O -m> D2) ((fcont_mshift f) @ h).

Lemma fcont_lub_simpl : ∀ `{c1:cpo D1} `{c2:cpo D2} (h:nat -m> D1 -c> D2)(x:D1),
            lub h x = lub (h ⟹ x).

Instance cont_app_monotonic : ∀ `{o1:ord D1} `{c2:cpo D2} `{c3:cpo D3} (f:D1 -m> D2 -m> D3)
            (p:forall k, continuous (f k)),
            monotonic (O2:=D2 -c> D3) (fun (k:D1) ⇒ cont _ (fcontinuous:=p k)).
Qed.

Definition cont_app `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} (f:D1 -m> D2 -m> D3)
            (p:forall k, continuous (f k)) : D1 -m> (D2 -c> D3)
    := mon (fun k ⇒ cont (f k) (fcontinuous:=p k)).

Lemma cont_app_simpl :
∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}(f:D1 -m> D2 -m> D3)(p:forall k, continuous (f k))
        (k:D1), cont_app f p k = cont (f k).

Instance cont2_continuous `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} (f:D1 -m> D2 -m> D3)
           (p:continuous2 f) : continuous (cont_app f (continuous2_app f)).
Qed.

Definition cont2 `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} (f:D1 -m> D2 -m> D3)
           {p:continuous2 f} : D1 -c> (D2 -c> D3)
:= cont (cont_app f (continuous2_app f)).

Instance Fcontm_continuous `{c1:cpo D1} `{c2:cpo D2} : continuous (Fcontm D1 D2).
Save.
Hint Resolve @Fcontm_continuous.

Instance fcont_comp_continuous : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
    (f:D2 -c> D3) (g:D1 -c> D2), continuous (f @ g).
Save.

Definition fcont_comp `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} (f:D2 -c> D3) (g:D1 -c> D2)
  : D1 -c> D3 := cont (f @ g).

Infix "@_" := fcont_comp (at level 35) : O_scope.

Lemma fcont_comp_simpl : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
       (f:D2 -c> D3)(g:D1 -c> D2) (x:D1), (f @_ g) x = f (g x).
```

```
Lemma fcontm_fcont_comp_simpl : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
      (f:D2 -c> D3)(g:D1 -c> D2), fcontm (f @_ g) = f @ g.

Lemma fcont_comp_le_compat : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
      (f g : D2 -c> D3) (k l :D1 -c> D2),
      f ≤ g → k ≤ l → f @_ k ≤ g @_ l.
Hint Resolve @fcont_comp_le_compat.

Add Parametric Morphism `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
    : (@fcont_comp _ _ c1 _ _ c2 _ _ c3)
      with signature Ole ++> Ole ++> Ole as fcont_comp_le_morph.
Save.

Add Parametric Morphism `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
    : (@fcont_comp _ _ c1 _ _ c2 _ _ c3)
      with signature Oeq ==>Oeq ==>Oeq as fcont_comp_eq_compat.
Save.

Definition fcont_Comp D1 `{c1:cpo D1} D2 `{c2:cpo D2} D3 `{c3:cpo D3}
      : (D2 -c> D3) -m> (D1 -c> D2) -m> D1 -c> D3
      := mon2 _ (mf:=fcont_comp_le_compat (D1:=D1) (D2:=D2) (D3:=D3)).

Lemma fcont_Comp_simpl : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
                         (f:D2 -c> D3) (g:D1 -c> D2), fcont_Comp D1 D2 D3 f g = f @_ g.

Instance fcont_Comp_continuous2
   : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}, continuous2 (fcont_Comp D1 D2 D3).
Save.

Definition fcont_COMP D1 `{c1:cpo D1} D2 `{c2:cpo D2} D3 `{c3:cpo D3}
      : (D2 -c> D3) -c> (D1 -c> D2) -c> D1 -c> D3
      := cont2 (fcont_Comp D1 D2 D3).

Lemma fcont_COMP_simpl : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
        (f: D2 -c> D3) (g:D1 -c> D2),
        fcont_COMP D1 D2 D3 f g = f @_ g.

Definition fcont2_COMP D1 `{c1:cpo D1} D2 `{c2:cpo D2} D3 `{c3:cpo D3} D4 `{c4:cpo D4}
   : (D3 -c> D4) -c> (D1 -c> D2 -c> D3) -c> D1 -c> D2 -c> D4 :=
     (fcont_COMP D1 (D2 -c> D3) (D2 -c> D4)) @_ (fcont_COMP D2 D3 D4).

Definition fcont2_comp `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} `{c4:cpo D4}
           (f:D3 -c> D4)(F:D1 -c> D2 -c> D3) := fcont2_COMP D1 D2 D3 D4 f F.

Infix "@@_" := fcont2_comp (at level 35) : O_scope.

Lemma fcont2_comp_simpl : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} `{c4:cpo D4}
       (f:D3 -c> D4)(F:D1 -c> D2 -c> D3)(x:D1)(y:D2), (f @@_ F) x y = f (F x y).

Lemma fcont_le_compat2 : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} (f : D1 -c> D2 -c> D3)
    (x y : D1) (z t : D2), x ≤ y → z ≤ t → f x z ≤ f y t.
Hint Resolve @fcont_le_compat2.

Lemma fcont_eq_compat2 : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} (f : D1 -c> D2 -c> D3)
    (x y : D1) (z t : D2), x == y → z == t → f x z == f y t.
Hint Resolve @fcont_eq_compat2.

Lemma fcont_continuous : ∀ `{c1:cpo D1} `{c2:cpo D2} (f:D1 -c> D2)(h:nat-m> D1),
            f (lub h) ≤ lub (f @ h).
Hint Resolve @fcont_continuous.

Instance fcont_continuous2 : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
          (f:D1 -c> D2 -c> D3), continuous2 (Fcontm D2 D3 @ f).
Save.
Hint Resolve @fcont_continuous2.
```

```
Instance cshift_continuous2 : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
         (f:D1 -c> D2 -c> D3), continuous2 (mshift (Fcontm D2 D3 @ f)).
Save.
Hint Resolve @cshift_continuous2.

Definition cshift `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} (f:D1 -c> D2 -c> D3)
   : D2 -c> D1 -c> D3 := cont2 (mshift (Fcontm D2 D3 @ f)).

Lemma cshift_simpl : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
           (f:D1 -c> D2 -c> D3) (x:D2) (y:D1), cshift f x y = f y x.

Definition fcont_SEQ D1 `{c1:cpo D1} D2 `{c2:cpo D2} D3 `{c3:cpo D3}
   : (D1 -c> D2) -c> (D2 -c> D3) -c> D1 -c> D3 := cshift (fcont_COMP D1 D2 D3).

Lemma fcont_SEQ_simpl : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
        (f: D1 -c> D2) (g:D2 -c> D3), fcont_SEQ D1 D2 D3 f g = g @_ f.

Instance Id_mon : ∀ `{o1:ord O1}, monotonic (fun x:O1 ⇒ x).
Save.

Definition Id O1 {o1:ord O1} : O1 -m> O1 := mon (fun x ⇒ x).

Lemma Id_simpl : ∀ `{o1:ord O1} (x:O1), Id O1 x = x.
```

3.8 Fixpoints

```
Fixpoint iter_ `{c: cpo D} (f : D -m> D) n {struct n} : D
    := match n with O ⇒ 0 | S m ⇒ f (iter_ f m) end.

Lemma iter_incr : ∀ `{c: cpo D} (f : D -m> D) n, iter_ f n ≤ f (iter_ f n).
Hint Resolve @iter_incr.

Instance iter_mon : ∀ `{c: cpo D} (f : D -m> D), monotonic (iter_ f).
Save.

Definition iter `{c: cpo D} (f : D -m> D) : nat -m> D := mon (iter_ f).

Definition fixp `{c: cpo D} (f : D -m> D) : D := mlub (iter_ f).

Lemma fixp_le : ∀ `{c: cpo D} (f : D -m> D), fixp f ≤ f (fixp f).
Hint Resolve @fixp_le.

Lemma fixp_eq : ∀ `{c: cpo D} (f : D -m> D) {mf:continuous f},
       fixp f == f (fixp f).

Lemma fixp_inv : ∀ `{c: cpo D} (f : D -m> D) g, f g ≤ g → fixp f ≤ g.

Definition fixp_cte : ∀ `{c:cpo D} (d:D), fixp (mon (cte D d)) == d.
Save.
Hint Resolve @fixp_cte.

Lemma fixp_le_compat : ∀ `{c:cpo D} (f g : D -m> D),
       f ≤ g → fixp f ≤ fixp g.
Hint Resolve @fixp_le_compat.

Instance fixp_monotonic `{c:cpo D} : monotonic fixp.
Save.

Add Parametric Morphism `{c:cpo D} : (fixp (c:=c))
    with signature Oeq ==>Oeq as fixp_eq_compat.
Save.
Hint Resolve @fixp_eq_compat.

Definition Fixp D `{c:cpo D} : (D -m> D) -m> D := mon fixp.

Lemma Fixp_simpl : ∀ `{c:cpo D} (f:D-m>D), Fixp D f = fixp f.
```

```
Instance iter_monotonic `{c:cpo D} : monotonic iter.
Save.
Definition Iter D `{c:cpo D} : (D -m-> D) -m-> (nat -m-> D) := mon iter.
Lemma IterS_simpl : ∀ `{c:cpo D} f n, Iter D f (S n) = f (Iter D f n).
Lemma iterO_simpl : ∀ `{c:cpo D} (f: D-m-> D), iter f O = (0:D).
Lemma iterS_simpl : ∀ `{c:cpo D} f n, iter f (S n) = f (iter f n).
Lemma iter_continuous : ∀ `{c:cpo D} (h : nat -m-> (D -m-> D)),
       (∀ n, continuous (h n)) → iter (lub h) ≤ lub (mon iter @ h).
Hint Resolve @iter_continuous.
Lemma iter_continuous_eq : ∀ `{c:cpo D} (h : nat -m-> (D -m-> D)),
     (∀ n, continuous (h n)) → iter (lub h) == lub (mon iter @ h).
Lemma fixp_continuous : ∀ `{c:cpo D} (h : nat -m-> (D -m-> D)),
       (∀ n, continuous (h n)) → fixp (lub h) ≤ lub (mon fixp @ h).
Hint Resolve @fixp_continuous.
Lemma fixp_continuous_eq : ∀ `{c:cpo D} (h : nat -m-> (D -m-> D)),
       (∀ n, continuous (h n)) → fixp (lub h) == lub (mon fixp @ h).
```

3.8.1 Induction principle

```
Definition admissible `{c:cpo D}(P:D->Type) :=
          ∀ f : nat -m-> D, (∀ n, P (f n)) → P (lub f).
Lemma fixp_ind : ∀ `{c:cpo D}(F:D -m-> D)(P:D->Type),
       admissible P → P 0 → (∀ x, P x → P (F x)) → P (fixp F).

Ltac continuity cont Cont Hcont:=
  match goal with
 | ⊢ (Ole ?x1 (lub (mon (fun (n:nat) ⇒ cont (@?g n))))) ⇒
      let f := fresh "f" in (
           pose (f:=g); assert (monotonic f) ;
                                [auto | (transitivity (lub (Cont@(mon f))); [rewrite ← Hcont | auto])]
           )
end.
Ltac gen_monotonic :=
match goal with ⊢ context [(@mon _ _ _ _ ?f ?mf)] ⇒ generalize (mf:monotonic f)
end.
Ltac gen_monotonic1 f :=
match goal with ⊢ context [(@mon _ _ _ _ f ?mf)] ⇒ generalize (mf:monotonic f)
end.
```

4 Utheory.v: Specification of U, interval $[0, 1]$

```
Require Export Misc.
Require Export Ccpo.
Open Local Scope O_scope.
```

4.1 Basic operators 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.
Instance ordU : ord U.
Instance cpoU : cpo U.
Delimit Scope U_scope with U.
```

```
Parameters Uplus Umult Udiv: U → U → U.
Parameter Uinv : U → U.
Parameter Unth : nat → U.
```

```
Infix "+" := Uplus : U_scope.
Infix "×" := Umult : U_scope.
Infix "/" := Udiv : U_scope.
Notation "[1-] x" := (Uinv x) (at level 35, right associativity) : U_scope.
```

```
Notation "[1/]1+ n" := (Unth n) (at level 35, right associativity) : U_scope.
Open Local Scope U_scope.
```

```
Definition U1 : U := [1-] 0.
Notation "1" := U1 : U_scope.
```

4.2 Basic Properties

```
Hypothesis Udiff_0_1 : ¬0 == 1.
```

```
Hypothesis Uplus_sym : ∀ x y:U, x + y == y + x.
Hypothesis Uplus_assoc : ∀ x y z:U, x + (y + z) == x + y + z.
Hypothesis Uplus_zero_left : ∀ x:U, 0 + x == x.
```

```
Hypothesis Umult_sym : ∀ x y:U, x × y == y × x.
Hypothesis Umult_assoc : ∀ x y z:U, x × (y × z) == x × y × z.
Hypothesis Umult_one_left : ∀ x:U, 1 × x == x.
```

```
Hypothesis Uinv_one : [1-] 1 == 0.
```

```
Hypothesis Umult_div : ∀ x y, ¬0 == y → x ≤ y → y × (x/y) == x.
Hypothesis Udiv_le_one : ∀ x y, ¬0 == y → y ≤ x → (x/y) == 1.
Hypothesis Udiv_by_zero : ∀ x y, 0 == y → (x/y) == 0.
```

- Property : $1 - (x + y) + x = 1 - y$ holds when $x + y$ does not overflow

```
Hypothesis Uinv_plus_left : ∀ x y, y ≤ [1-] x → [1-] (x + y) + x == [1-] y.
```

- Property : $(x + y) \times z = x \times z + y \times z$ holds when $x + y$ does not overflow

```
Hypothesis Udistr_plus_right : ∀ x y z, x ≤ [1-] y → (x + y) × z == x × z + y × z.
```

- Property : $1 - (x \times y) = (1 - x) \times y + (1 - y)$

```
Hypothesis Udistr_inv_right : ∀ x y:U, [1-] (x × y) == ([1-] x) × y + [1-] y.
```

- Totality of the order

```
Hypothesis Ule_class : ∀ x y : U, class (x ≤ y).
```

```
Hypothesis Ule_total : ∀ x y : U, orc (x ≤ y) (y ≤ x).
Implicit Arguments Ule_total [].
```

- The relation $x \leq y$ is compatible with operators

Instance Uplus_mon_right : $\forall\ x$, **monotonic** (*Uplus* x).
Instance Umult_mon_right : $\forall\ x$, **monotonic** (*Umult* x).

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, $\neg 0 == z \rightarrow z \times x \leq z \times y \rightarrow x \leq y$.

- Property of *Unth*: $\frac{1}{n+1} == 1 - n \times \frac{1}{n+1}$

Hypothesis *Unth_prop* : $\forall\ n$, [1/]1+n == [1-](compn *Uplus* 0 (fun $k \Rightarrow$ [1/]1+n) n).

- Archimedian property

Hypothesis *archimedian* : $\forall\ x$, $\neg 0 == x \rightarrow$ exc (fun $n \Rightarrow$ [1/]1+n $\leq x$).

- Stability properties of lubs with respect to + and $\times$

Hypothesis *Uplus_right_continuous* : $\forall\ k$, **continuous** (mon (*Uplus* k)).
Hypothesis *Umult_right_continuous* : $\forall\ k$, **continuous** (mon (*Umult* k)).

End UNIVERSE.

Declare Module *Univ*:Universe.
Export *Univ*.

Hint Resolve *Udiff_0_1 Unth_prop*.
Hint Resolve *Uplus_sym Uplus_assoc Umult_sym Umult_assoc*.
Hint Resolve *Uinv_one Uinv_plus_left Umult_div Udiv_le_one Udiv_by_zero*.
Hint Resolve *Uplus_zero_left Umult_one_left Udistr_plus_right Udistr_inv_right*.
Hint Resolve *Uplus_mon_right Umult_mon_right Uinv_le_compat*.
Hint Resolve *lub_le le_lub Uplus_right_continuous Umult_right_continuous*.
Hint Resolve *Ule_total Ule_class*.

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

Require Export Utheory.
Require Export Arith.
Require Export Omega.

Open Local Scope *U_scope*.

5.1 Direct consequences of axioms

Lemma *Uplus_le_compat_right* : $\forall\ x\ y\ z$:U, $y \leq z \rightarrow x + y \leq x + z$.
Hint Resolve *Uplus_le_compat_right*.

Instance *Uplus_mon2* : *monotonic2 Uplus*.
Save.
Hint Resolve *Uplus_mon2*.

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

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

Lemma *Uplus_eq_compat_left* : $\forall$ x $y\ z$:U, $x == y \rightarrow x + z == y + z$.
Hint Resolve *Uplus_eq_compat_left*.

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

Hint Resolve *Uplus_eq_compat_left Uplus_eq_compat_right*.

Add *Morphism Uplus* with *signature Oeq* $\Longrightarrow$ *Oeq* $\Longrightarrow$ *Oeq* as *Uplus_eq_compat*.
Qed.
Hint Immediate *Uplus_eq_compat*.

Add *Morphism Uinv* with *signature Oeq* $\Longrightarrow$ *Oeq* as *Uinv_eq_compat*.
Qed.
Hint Resolve *Uinv_eq_compat*.

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

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

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

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

Lemma *Unit* : $\forall\ x$:U, $x \leq 1$.
Hint Resolve *Unit*.

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

Lemma *Ueq_class* : $\forall\ x\ y$:U, *class* (x==y).

Lemma *Ueq_double_neg* : $\forall\ x\ y : U$, $\neg\ \neg\ (x == y) \to x == y$.
Hint Resolve *Ueq_class*.
Hint Immediate *Ueq_double_neg*.

Lemma *Ule_orc* : $\forall\ x\ y : U$, *orc* (x<=y) ($\neg\ x$<=y).
Implicit Arguments *Ule_orc* [].

Lemma *Ueq_orc* : $\forall\ x\ y$:U, *orc* (x==y) ($\neg\ x$==y).
Implicit Arguments *Ueq_orc* [].

Lemma *Upos* : $\forall\ x$:U, $0 \leq x$.

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

Hint Resolve *Upos Ule_0_1*.

5.2 Properties of == derived from properties of $\leq$

Definition *UPlus* : $U \xrightarrow{m} \mathsf{U} \xrightarrow{m} U :=$ *mon2 Uplus*.

Definition *UPlus_simpl* : $\forall$ x y, *UPlus* $x\ y = x$+y.
Save.

Instance *Uplus_continuous2* : *continuous2* (*mon2 Uplus*).
Save.

Hint Resolve *Uplus_continuous2*.

Lemma *Umult_le_compat_right* : $\forall\ x\ y\ z$:U, $y \leq z \to x \times y \leq x \times z$.
Hint Resolve *Umult_le_compat_right*.

Instance *Umult_mon2* : monotonic2 *Umult*.
Save.

Lemma *Umult_le_compat_left* : $\forall\ x\ y\ z$:U, $x \leq y \to x \times z \leq y \times z$.
Hint Resolve *Umult_le_compat_left*.

Lemma *Umult_le_compat* : $\forall\ x$ y $z\ t$, $x \leq y \to z \leq t \to x \times z \leq y \times t$.

Hint Immediate *Umult_le_compat*.

Definition *UMult* : $U \xrightarrow{m} U \xrightarrow{m} U := mon2\ Umult$.

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

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

Hint Resolve *Umult_eq_compat_left Umult_eq_compat_right*.

Definition *UMult_simpl* : $\forall\ x\ y$, $UMult\ x\ y = x$*y.
Save.

Instance *Umult_continuous2* : *continuous2* (*mon2 Umult*).
Save.
Hint Resolve *Umult_continuous2*.

5.3 U is a setoid

Add *Morphism Umult* with *signature Oeq* $\Longrightarrow$ *Oeq* $\Longrightarrow$ *Oeq*
as *Umult_eq_compat*.
Qed.

Hint Immediate Umult_eq_compat.

Instance *Uinv_mon* : *monotonic* (*o1*:=Iord *U*) *Uinv*.
Save.

Definition *UInv* : $U \xrightarrow{m} U :=$ *mon* (o1:=Iord *U*) *Uinv*.

Definition *UInv_simpl* : $\forall\ x$, *UInv* $x = [1\text{-}]x$.
Save.

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

5.4 Definition and properties of $x < y$

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

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

Hint Unfold *Ult*.

Add *Morphism Ult* with *signature Oeq* $\Longrightarrow$ *Oeq* $\Longrightarrow$ *iff* as *Ult_eq_compat_iff*.
Save.

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

Lemma *Ult_class* : $\forall\ x\ y$, *class* (x<y).
Hint Resolve *Ult_class*.

5.4.1 Properties of $x \leq y$

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

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

Hint Immediate *Ule_zero_eq Uge_one_eq*.

5.4.2 Properties of $x < y$

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

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

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

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

Lemma *Ule_diff_lt* : $\forall\ x\ y : U$, $x \leq y \rightarrow$ ˜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 *Ule_total Ult_neq_zero*.

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

Hint Immediate *Uinv_simpl*.

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

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

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

Lemma *Umult_one_right_eq* : $\forall\ x\ y$:U, $y == 1 \rightarrow x \times y == x$.
Hint Resolve *Umult_one_right_eq*.

Lemma *Umult_one_left_eq* : $\forall\ x\ y$:U, $x == 1 \rightarrow x \times y == y$.
Hint Resolve *Umult_one_left_eq*.

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

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

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-] $y \rightarrow y \leq$ [1-] x.

Hint Immediate *Uinv_le_perm_right.*

Lemma Uinv_le_perm_left : $\forall\ x\ y$:U, [1-] $x \leq y \rightarrow$ [1-] $y \leq x$.
Hint Immediate *Uinv_le_perm_left.*

Lemma *Uinv_le_simpl* : $\forall\ x\ y$:U, [1-] $x \leq$ [1-] $y \rightarrow y \leq x$.
Hint Immediate *Uinv_le_simpl.*

Lemma *Uinv_double_le_simpl_right* : $\forall\ x\ y$, x<=y $\rightarrow x \leq$ [1-][1-]y.
Hint Resolve *Uinv_double_le_simpl_right.*

Lemma *Uinv_double_le_simpl_left* : $\forall\ x\ y$, x<=y $\rightarrow$ [1-][1-]$x \leq y$.
Hint Resolve *Uinv_double_le_simpl_left.*

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

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

Hint Immediate *Uinv_eq_perm_right.*

Lemma *Uinv_eq_simpl* : $\forall\ x\ y$:U, [1-] $x ==$ [1-] $y \rightarrow x == y$.
Hint Immediate *Uinv_eq_simpl.*

Lemma *Uinv_double_eq_simpl_right* : $\forall\ x\ y$, x==y $\rightarrow x ==$ [1-][1-]y.
Hint Resolve *Uinv_double_eq_simpl_right.*

Lemma *Uinv_double_eq_simpl_left* : $\forall\ x\ y$, x==y $\rightarrow$ [1-][1-]$x == y$.
Hint Resolve *Uinv_double_eq_simpl_left.*

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

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

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

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

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

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

Hint Resolve *Uinv_plus_left_le Uinv_plus_right_le.*

5.7 Disequality

Lemma *neq_sym* : $\forall\ x\ y$:U, ~x==y $\rightarrow$ ~y==x.
Hint Immediate *neq_sym.*

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

Lemma Uinv_neq_simpl : $\forall\ x\ y$, $\neg$ [1-] $x ==$ [1-] $y \rightarrow$ ~x == y.

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

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

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

5.7.1 Properties of $<$

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

Hint Resolve *Ult_0_1 Ult_antirefl*.

Lemma *Ule_neq_zero* : $\forall\ (x\ y{:}\mathrm{U}), \neg 0 == \mathtt{x} \to x \leq y \to \neg\ 0 == y$.

Lemma *Uplus_neq_zero_left* : $\forall\ x\ y, \neg 0 == x \to \neg 0 == x{+}\mathrm{y}$.

Lemma *Uplus_neq_zero_right* : $\forall\ x\ y, \neg 0 == y \to \neg 0 == \mathtt{x}{+}\mathrm{y}$.

Lemma *not_Ult_le* : $\forall\ x\ y, \tilde{}\mathrm{x} < y \to y \leq \mathtt{x}$.

Lemma *Ule_not_lt* : $\forall\ x\ y, x \leq y \to \tilde{}\mathrm{y} < x$.

Hint Immediate *not_Ult_le Ule_not_lt*.

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

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

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

Hint Resolve *Uplus_lt_compat_right Uplus_lt_compat_left*.

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

Hint Immediate *Uplus_lt_compat*.

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

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

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

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

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*.

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

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

Lemma *Umult_zero_eq* : $\forall\ x\ y\ z, x == 0 \to x \times y == x \times z$.

5.7.2 Compatibility of operations with respect to order.

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

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

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

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

Lemma *Umult_lt_compat_right* : $\forall\ x\ y\ z, \neg 0 == z \to x < \mathtt{y} \to (z \times x) < (z \times y)$.

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

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

Hint Resolve *Umult_lt_compat_left Umult_lt_compat_right*.

Lemma *Umult_zero_simpl_right* : $\forall\ x\ y, 0 == x{*}\mathrm{y} \to \neg 0 == x \to 0 == y$.

Lemma *Umult_zero_simpl_left* : $\forall\ x\ y, 0 == x{*}\mathrm{y} \to \neg 0 == y \to 0 == x$.

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

Lemma *Umult_lt_zero* : $\forall\ x\ y, 0 < x \to 0 < y \to 0 < x{*}\mathrm{y}$.
Hint Resolve *Umult_lt_zero*.

Lemma *Umult_lt_compat* : $\forall\ x\ y\ z\ t, x < y \to z < t \to x \times z < y \times t$.

5.7.3 More Properties

Lemma *Uplus_one* : $\forall\ x\ y$, [1-] $x \leq y \rightarrow x + y == 1$.
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* : $\forall\ x\ y\ z\ t$, $x \leq$ [1-] $y \rightarrow (x \times z) \leq$ [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, $y \leq z \rightarrow x \times$ [1-] $y + y \leq x \times$ [1-] $z + z$.
Hint Resolve *Umult_inv_plus_le*.

Lemma *Uplus_lt_Uinv* : $\forall\ x\ y$, x+y $< 1 \rightarrow x \leq$ [1-] y.

Lemma *Uinv_lt_perm_left*: $\forall\ x\ y : U$, [1-] $x < y \rightarrow$ [1-] $y < x$.

Lemma *Uinv_lt_perm_right*: $\forall\ x\ y : U$, $x <$ [1-] $y \rightarrow y <$ [1-] x.

Hint Immediate Uinv_lt_perm_left *Uinv_lt_perm_right*.

Lemma *Uinv_lt_one* : $\forall\ x$, $0 < x \rightarrow$ [1-]$x < 1$.

Lemma *Uinv_lt_zero* : $\forall\ x$, $x < 1 \rightarrow 0 <$ [1-]x.

Hint Resolve Uinv_lt_one *Uinv_lt_zero*.

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

Lemma *Umult_lt_right* : $\forall\ p$ q, $p <1 \rightarrow 0 < q \rightarrow p \times q < q$.

Lemma *Umult_lt_left* : $\forall\ p\ q$, $0 < p \rightarrow q < 1 \rightarrow p \times q < p$.

Hint Resolve *Umult_lt_right Umult_lt_left*.

5.8 Definition of x^n

Fixpoint *Uexp* (x:U) (n:nat) {struct n} : $U :=$
  match n with $0 \Rightarrow 1 \mid (S\ p) \Rightarrow x \times$ *Uexp* $x\ p$ end.

Infix "^" := *Uexp* : *U_scope*.

Lemma *Uexp_1* : $\forall\ x$, x^1==x.

Lemma *Uexp_0* : $\forall\ x$, x^0==1.

Lemma *Uexp_zero* : $\forall\ n$, (0<n)%nat $\rightarrow$ 0^n==0.

Lemma *Uexp_one* : $\forall\ n$, 1^n==1.

Lemma *Uexp_le_compat_right* :
  $\forall\ x\ n\ m$, (n<=m)%nat $\rightarrow x$^m $\leq x$^n.

Lemma *Uexp_le_compat_left* : $\forall\ x\ y\ n$, x<=y $\rightarrow$ x^n $\leq$ y^n.
Hint Resolve *Uexp_le_compat_left Uexp_le_compat_right*.

Lemma *Uexp_le_compat* : $\forall\ x\ y$ ($n\ m$:nat), x<=y $\rightarrow$ n<=m $\rightarrow$ x^m $\leq$ y^n.

Instance *Uexp_mon2* : *monotonic2* (*o1*:=Iord U) (*o3*:=Iord U) *Uexp*.
Save.

Definition *UExp* : $U \xrightarrow{m} (nat \xrightarrow{m} - \ U) :=$ *mon2 Uexp*.

Add *Morphism Uexp* with *signature Oeq* $\Longrightarrow$*eq* $\Longrightarrow$*Oeq* as *Uexp_eq_compat*.
Save.

Lemma *Uexp_inv_S* : $\forall\ x\ n$, ([1-]x^(S n))==x*([1-]x^n)+[1-]x.

Lemma *Uexp_lt_compat* : ∀ *p q n*, (*O*<n)%nat->(p<q)->(p^n<q^n).

Hint Resolve *Uexp_lt_compat*.

Lemma *Uexp_lt_zero* : ∀ *p n*, (0<p)->(0<p^n).
Hint Resolve *Uexp_lt_zero*.

Lemma *Uexp_lt_one* : ∀ *p n*, (0<n)%nat->(p<1)->(p^n<1).
Hint Resolve *Uexp_lt_one*.

Lemma *Uexp_lt_antimon*: ∀ *p n m*, (*n*<m)%nat-> 0<p → *p* < 1 → *p*^m < *p*^n.
Hint Resolve *Uexp_lt_antimon*.

5.9 Properties of division

Lemma *Udiv_mult* : ∀ *x y*, ¬0==y → *x* ≤ *y* → (*x*/y)*y == *x*.
Hint Resolve *Udiv_mult*.

Lemma *Umult_div_le* : ∀ *x y*, *y* × (*x* / *y*) ≤ *x*.
Hint Resolve *Umult_div_le*.

Lemma *Udiv_mult_le* : ∀ *x y*, (*x*/y)*y ≤ *x*.
Hint Resolve *Udiv_mult_le*.

Lemma *Udiv_le_compat_left* : ∀ *x y z*, *x* ≤ *y* → x/z ≤ *y*/z.
Hint Resolve *Udiv_le_compat_left*.

Lemma *Udiv_eq_compat_left* : ∀ *x y z*, *x* == *y* → *x*/z == *y*/z.
Hint Resolve *Udiv_eq_compat_left*.

Lemma *Umult_div_le_left* : ∀ *x y z*, ¬0==y → *x**y<=z → *x* ≤ *z*/y.

Lemma *Udiv_le_compat_right* : ∀ *x y z*, ¬0==y → *y* ≤ *z* → *x*/z ≤ *x*/y.
Hint Resolve *Udiv_le_compat_right*.

Lemma *Udiv_eq_compat_right* : ∀ *x y z*, *y* == *z* → *x*/z == *x*/y.
Hint Resolve *Udiv_eq_compat_right*.

Add *Morphism Udiv* with *signature Oeq* ⟹*Oeq* ⟹*Oeq* as *Udiv_eq_compat*.
Save.

Add *Morphism Udiv* with *signature Ole* ++> *Oeq* ⟹*Ole* as *Udiv_le_compat*.
Save.

Lemma *Umult_div_eq* : ∀ *x y z*, ¬0 == *y* → *x**y == *z* → *x* == *z*/y.

Lemma *Umult_div_le_right* : ∀ *x y z*, *x* ≤ *y**z → *x*/z ≤ *y*.

Lemma Udiv_le : ∀ *x y*, ¬0==y → *x* ≤ *x*/y.

Lemma *Udiv_zero* : ∀ *x*, 0/x==0.
Hint Resolve *Udiv_zero*.

Lemma *Udiv_zero_eq* : ∀ *x y*, 0==x → *x*/y==0.
Hint Resolve *Udiv_zero_eq*.

Lemma *Udiv_one* : ∀ *x*, ¬0==x → *x*/1==x.
Hint Resolve *Udiv_one*.

Lemma *Udiv_refl* : ∀ *x*, ¬0==x → *x*/x==1.
Hint Resolve *Udiv_refl*.

Lemma *Umult_div_assoc* : ∀ *x y z*, *y* ≤ *z*→ (*x**y) / *z* == *x* × (*y*/z).

Lemma *Udiv_mult_assoc* : ∀ *x y z*, *x* ≤ *y**z → *x*/(y*z) == (*x*/y)/z.

Lemma *Udiv_inv* : ∀ *x y*, ¬0==y → [1-](*x*/y) ≤ ([1-]*x*)/y.

Lemma *Uplus_div_inv* : ∀ *x y z*, *x*+y ≤ *z* → *x*≤[1-]*y* → *x*/z ≤ [1-](*y*/z).
Hint Resolve *Uplus_div_inv*.

Lemma *Udiv_plus_le* : $\forall\ x\ y\ z,\ x/\mathrm{z} + y/\mathrm{z} \leq (x{+}\mathrm{y})/\mathrm{z}$.
Hint Resolve *Udiv_plus_le*.

Lemma *Udiv_plus* : $\forall\ x\ y\ z,\ (x{+}\mathrm{y})/\mathrm{z} == x/\mathrm{z} + y/\mathrm{z}$.
Hint Resolve *Udiv_plus*.

Instance *Udiv_mon* : $\forall\ k,\ monotonic\ (\mathtt{fun}\ x \Rightarrow (x/\mathrm{k}))$.
Save.

Definition *UDiv* $(k{:}\mathrm{U}) : U \xrightarrow{m} U := mon\ (\mathtt{fun}\ x \Rightarrow (x/\mathrm{k}))$.

Lemma *UDiv_simpl* : $\forall\ (k{:}\mathrm{U})\ x,\ UDiv\ k\ x = x/\mathrm{k}$.

5.10 Definition and properties of $x\&y$

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

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

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

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

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

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

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

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

Lemma Uesp_zero : $\forall\ x\ y,\ x \leq [1\text{-}]\ y \rightarrow x\ \&\ y == 0$.

Hint Resolve *Uesp_sym Uesp_one_right Uesp_one_left 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.

Add *Morphism Uesp* with *signature* $Oeq \Longrightarrow Oeq \Longrightarrow Oeq$ as *Uesp_eq_compat*.
Save.

Lemma *Uesp_le_compat* : $\forall\ x\ y\ z\ t,\ x{<=}\mathrm{y} \rightarrow z <=\mathrm{t} \rightarrow x\&\mathrm{z} \leq y\&\mathrm{t}$.

Hint Immediate *Uesp_le_compat Uesp_eq_compat*.

Lemma *Uesp_zero_one_mult_left* : $\forall\ x\ y,\ orc\ (x == 0)\ (x == 1) \rightarrow x\ \&\ y == x \times y$.

Lemma *Uesp_zero_one_mult_right* : $\forall\ x\ y,\ orc\ (y == 0)\ (y == 1) \rightarrow x\ \&\ y == x \times y$.

Hint Immediate *Uesp_zero_one_mult_left Uesp_zero_one_mult_right*.

Instance *Uesp_mon* : *monotonic2 Uesp*.
Save.

Definition *UEsp* : $U \xrightarrow{m} U \xrightarrow{m} U := mon2\ Uesp$.

Lemma *UEsp_simpl* : $\forall\ x\ y,\ UEsp\ x\ y = x\ \&\ y$.

Lemma *Uesp_le_left* : $\forall\ x\ y,\ x\ \&\ y \leq x$.

Lemma *Uesp_le_right* : $\forall\ x\ y,\ x\ \&\ y \leq y$.

Hint Resolve *Uesp_le_left Uesp_le_right*.

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

Lemma *Uesp_le_plus_inv* : $\forall\ x\ y,\ x \leq x\ \&\ y + [1\text{-}]\ y$.
Hint Resolve *Uesp_le_plus_inv*.

Lemma *Uplus_inv_le_esp* : $\forall\ x\ y\ z,\ x \leq y + ([1\text{-}]\ z) \rightarrow x\ \&\ z \leq y$.
Hint Immediate *Uplus_inv_le_esp*.

5.11 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*.

Lemma Uminus_le_compat : $\forall$ x y z t, $x \leq y \rightarrow t \leq$ z $\rightarrow x$ - $z \leq y$ - t.

Hint Immediate *Uminus_le_compat*.

Add *Morphism Uminus* with *signature Oeq* $\Longrightarrow$ *Oeq* $\Longrightarrow$ *Oeq* as Uminus_eq_compat.
Save.
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_zero_left* : $\forall$ x, 0 - $x == 0$.
Hint Resolve *Uminus_zero_left*.

Lemma *Uminus_eq* : $\forall$ x, x - $x == 0$.
Hint Resolve *Uminus_eq*.

Lemma *Uminus_le_left* : $\forall$ x y, x - $y \leq x$.

Hint Resolve *Uminus_le_left*.

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

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 *Uminus_plus_le* : $\forall$ x y, $x \leq (x$ - $y)$ + y.

Hint Resolve *Uminus_plus_le*.

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*.

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

Lemma *Uplus_esp_simpl* : $\forall$ x y, $(x$ - $(x$ & $y))$+y == x+y.
Hint Resolve *Uminus_esp_simpl_left Uplus_esp_simpl*.

Lemma *Uminus_esp_le_inv* : $\forall$ x y, x - $(x$ & $y) \leq$ [1-]y.

Hint Resolve *Uminus_esp_le_inv*.

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

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

5.12 Definition and properties of max

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

Lemma $max\_eq\_right : \forall\ x\ y : U,\ y \leq x \rightarrow max\ x\ y == x$.

Lemma $max\_eq\_left : \forall\ x\ y : U,\ x \leq y \rightarrow max\ x\ y == y$.

Hint Resolve $max\_eq\_right\ max\_eq\_left$.

Lemma $max\_eq\_case : \forall\ x\ y : U,\ orc\ (max\ x\ y == x)\ (max\ x\ y == y)$.

Add $Morphism\ max$ with $signature\ Oeq \Longrightarrow Oeq \Longrightarrow Oeq$ as max_eq_compat.
Save.

Lemma $max\_le\_right : \forall\ x\ y : U,\ x \leq max\ x\ y$.

Lemma $max\_le\_left : \forall\ x\ y : U,\ y \leq max\ x\ y$.

Hint Resolve $max\_le\_right\ max\_le\_left$.

Lemma $max\_le : \forall\ x\ y\ z : U,\ x \leq z \rightarrow y \leq z \rightarrow max\ x\ y \leq z$.

Lemma $max\_le\_compat : \forall\ x\ y\ z\ t{:}\ U,\ x \leq y \rightarrow z \leq t \rightarrow max\ x\ z \leq max\ y\ t$.
Hint Immediate max_le_compat.

Instance $max\_mon : monotonic2\ max$.
Save.

Definition $Max : U \xrightarrow{m} U \xrightarrow{m} U := mon2\ max$.

Lemma $max\_eq\_mult : \forall\ k\ x\ y,\ max\ (k{*}x)\ (k{*}y) == k \times max\ x\ y$.

Lemma $max\_eq\_plus\_cte\_right : \forall\ x\ y\ k,\ max\ (x{+}k)\ (y{+}k) == (max\ x\ y) + k$.

Hint Resolve $max\_eq\_mult\ max\_eq\_plus\_cte\_right$.

5.13 Definition and properties of min

Definition $min\ (x\ y : U) : U := [1\text{-}]\ ((y - x) + [1\text{-}]y)$.

Lemma $min\_eq\_right : \forall\ x\ y : U,\ x \leq y \rightarrow min\ x\ y == x$.

Lemma $min\_eq\_left : \forall\ x\ y : U,\ y \leq x \rightarrow min\ x\ y == y$.

Hint Resolve min_eq_right min_eq_left.

Lemma $min\_eq\_case : \forall\ x\ y : U,\ orc\ (min\ x\ y == x)\ (min\ x\ y == y)$.

Add $Morphism\ min$ with $signature\ Oeq \Longrightarrow Oeq \Longrightarrow Oeq$ as min_eq_compat.
Save.

Lemma $min\_le\_right : \forall\ x\ y : U,\ min\ x\ y <= x$.

Lemma $min\_le\_left : \forall\ x\ y : U,\ min\ x\ y \leq y$.

Hint Resolve $min\_le\_right\ min\_le\_left$.

Lemma $min\_le : \forall\ x\ y\ z : U,\ z \leq x \rightarrow z \leq y \rightarrow z \leq min\ x\ y$.

Lemma $Uinv\_min\_max : \forall\ x\ y,\ [1\text{-}](min\ x\ y) == \max\ ([1\text{-}]x)\ ([1\text{-}]y)$.

Lemma $Uinv\_max\_min : \forall\ x\ y,\ [1\text{-}](max\ x\ y) == \min\ ([1\text{-}]x)\ ([1\text{-}]y)$.

Lemma $min\_mult : \forall\ x\ y\ k,$
$min\ (k \times x)\ (k \times y) == k \times (min\ x\ y)$.
Hint Resolve min_mult.

Lemma $min\_plus : \forall\ x1\ x2\ y1\ y2,$
$(min\ x1\ x2) + (min\ y1\ y2) \leq min\ (x1{+}y1)\ (x2{+}y2)$.
Hint Resolve min_plus.

Lemma $min\_plus\_cte : \forall\ x\ y\ k,\ min\ (x + k)\ (y + k) == (min\ x\ y) + k$.
Hint Resolve min_plus_cte.

```
Lemma min_le_compat : ∀ x1 y1 x2 y2,
      x1<=y1 → x2 <=y2 → min x1 x2 ≤ min y1 y2.
Hint Immediate min_le_compat.

Instance min_mon2 : monotonic2 min.
Save.

Definition Min : U -m-> U -m-> U := mon2 min.

Lemma min_sym : ∀ x y, min x y == min y x.
Hint Resolve min_sym.

Lemma incr_decomp_aux : ∀ f g : nat -m-> U,
     ∀ n1 n2, (∀ m, ¬ ((n1<=m)%nat ∧ f n1 ≤ g m))
           → (∀ m, ~((n2<=m)%nat ∧ g n2 ≤ f m)) → (n1<=n2)%nat → False.

Lemma incr_decomp : ∀ f g: nat -m-> U,
     orc (∀ n, exc (fun m ⇒ (n<=m)%nat ∧ f n ≤ g m))
           (∀ n, exc (fun m ⇒ (n<=m)%nat ∧ g n ≤ f m)).
```

5.14 Other properties

```
Lemma Uplus_minus_simpl_right : ∀ x y, y ≤ [1-] x → (x + y) - y == x.
Hint Resolve Uplus_minus_simpl_right.

Lemma Uplus_minus_simpl_left : ∀ x y, y ≤ [1-] x → (x + y) - x == y.

Lemma Uminus_assoc_left : ∀ x y z, (x - y) - z == x - (y + z).

Hint Resolve Uminus_assoc_left.

Lemma Uminus_perm : ∀ x y z, (x - y) - z == (x - z) - y.
Hint Resolve Uminus_perm.

Lemma Uminus_le_perm_left : ∀ x y z, y ≤ x → x - y ≤ z → x ≤ z + y.

Lemma Uplus_le_perm_left : ∀ x y z, x ≤ y + z → x - y ≤ z.

Lemma Uminus_eq_perm_left : ∀ x y z, y ≤ x → x - y == z → x == z + y.

Lemma Uplus_eq_perm_left : ∀ x y z, y ≤ [1-] z → x == y + z → 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 : ∀ x y z, z ≤ y → x ≤ y - z → x + z ≤ y.

Lemma Uplus_le_perm_right : ∀ x y z, z ≤ [1-] x → x + z ≤ y → x ≤ y - z.
Hint Resolve Uminus_le_perm_right Uplus_le_perm_right.

Lemma Uminus_le_perm : ∀ x y z, z ≤ y → x ≤ [1-] z → x ≤ y - z → z ≤ y - x.
Hint Resolve Uminus_le_perm.

Lemma Uminus_eq_perm_right : ∀ x y z, z ≤ y → x == y - z → x + z == y.
Hint Resolve Uminus_eq_perm_right.

Lemma Uminus_plus_perm : ∀ x y z, y ≤ x → z ≤ [1-]x → (x - y) + z == (x + z) - y.

Lemma Uminus_zero_le : ∀ x y, x - y == 0 → x ≤ y.

Lemma Uminus_lt_non_zero : ∀ x y, x < y → ¬0 == y - x.
Hint Immediate Uminus_zero_le Uminus_lt_non_zero.

Lemma Ult_le_nth_minus : ∀ x y, x < y → exc (fun n ⇒ x ≤ y - [1/]1+n).

Lemma Ult_le_nth_plus : ∀ x y, x < y → exc (fun n : nat ⇒ x + [1/]1+n ≤ y).

Lemma Uminus_distr_left : ∀ x y z, (x - y) × z == (x × z) - (y × z).

Hint Resolve Uminus_distr_left.

Lemma Uminus_distr_right : ∀ x y z, x × (y - z) == (x × y) - (x × z).
```

Hint Resolve *Uminus_distr_right*.

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

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

Lemma *Udiv_minus* : $\forall\ x\ y\ z, \neg 0 == z \to x \leq z \to (x$ - $y)\ /\ z == x$/z - y/z.

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

Lemma *Uinv_mult_minus* : $\forall\ x\ y,$ ([1-]$x) \times y == y$ - $x \times y$.
Hint Resolve *Uinv_mult_minus*.

Lemma *Uminus_plus_perm_right* : $\forall\ x\ y\ z,\ y \leq x \to y \leq z \to (x$ - $y) +$ z $== x + (z$ - $y)$.
Hint Resolve *Uminus_plus_perm_right*.

- triangular inequality

Lemma *Uminus_triangular* : $\forall\ x\ y\ z,\ x$ - $y \leq (x$ - $z) + (z$ - $y)$.
Hint Resolve *Uminus_triangular*.

5.15 Definition and properties of generalized sums

Definition *sigma* : $(nat \to U) \to nat \xrightarrow{m} U$.
Defined.

Lemma *sigma_0* : $\forall\ (f$: nat $\to U),$ *sigma* $f\ O == 0$.

Lemma *sigma_S* : $\forall\ (f$:nat $\to U)\ (n$:nat), *sigma* $f\ (S\ n) = (f\ n) + (sigma\ f\ n)$.

Lemma *sigma_1* : $\forall\ (f : nat \to U),$ *sigma* $f\ (S\ 0) == f\ O$.

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

Hint Resolve *sigma_incr*.

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

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

Lemma *sigma_S_lift* : $\forall\ (f$:nat $\to U)\ (n$:nat),
sigma $f\ (S\ n) == (f\ O) +$ (*sigma* (fun $k \Rightarrow f\ (S\ k))\ n)$.

Lemma *sigma_plus_lift* : $\forall\ (f$:nat $\to$ U$)\ (n\ m$:nat),
sigma $f\ (n$+m)%nat $==$ *sigma* $f\ n +$ *sigma* (fun $k \Rightarrow f\ (n$+k)%nat$)\ m$.

Lemma *sigma_zero* : $\forall\ f\ n, (\forall\ k, (k<$n)%nat $\to f\ k == 0) \to$ *sigma* $f\ n == 0$.

Lemma *sigma_not_zero* : $\forall\ f\ n\ k, (k<$n)%nat $\to 0 < f\ k \to 0 <$ *sigma* $f\ n$.

Lemma *sigma_zero_elim* : $\forall\ f\ n,$ (*sigma* $f\ n) == 0 ->$forall $k, (k<$n)%nat $\to f\ k == 0$.

Hint Resolve *sigma_eq_compat sigma_le_compat sigma_zero*.

Lemma **sigma_le** : $\forall\ f\ n$ k, $(k<$n)%nat $\to f\ k \leq$ *sigma* $f\ n$.
Hint Resolve *sigma_le*.

Lemma *sigma_minus_decr* : $\forall\ f\ n, (\forall\ k, f\ (S\ k) \leq f\ k) \to$
sigma (fun $k \Rightarrow f$ k - $f\ (S\ k))\ n == f\ O$ - f n.

Lemma *sigma_minus_incr* : $\forall\ f\ n, (\forall\ k, f\ k \leq f\ (S\ k)) \to$
sigma (fun $k \Rightarrow f\ (S\ k)$ - $f\ k)\ n == f\ n$ - $f\ O$.

5.16 Definition and properties of generalized products

Definition $prod\ (alpha : nat \to U)\ (n{:}\mathrm{nat}) := compn\ Umult\ 1\ alpha\ \mathtt{n}$.

Lemma $prod\_0 : \forall\ (f : nat \to U),\ prod\ f\ 0 = 1$.

Lemma $prod\_S : \forall\ (f : \mathrm{nat} \to U)\ (n{:}\mathrm{nat}),\ prod\ f\ (S\ n) = (f\ n) \times (prod\ f\ n)$.

Lemma $prod\_1 : \forall\ (f : nat \to U),\ prod\ f\ (S\ 0) == f\ O$.

Lemma $prod\_S\_lift : \forall\ (f : \mathrm{nat} \to U)\ (n{:}\mathrm{nat}),$
$prod\ f\ (S\ n) == (f\ O) \times (prod\ (\mathtt{fun}\ k \Rightarrow f\ (S\ k))\ n)$.

Lemma $prod\_decr : \forall\ (\mathbf{f} : nat \to U)\ (n\ m{:}\mathrm{nat}),\ (n \le m)\%\mathrm{nat} \to prod\ f\ m \le prod\ f\ n$.

Hint Resolve $prod\_decr$.

Lemma $prod\_eq\_compat : \forall\ (f\ g{:}\ \mathsf{nat} \to U)\ (\mathtt{n}{:}\mathrm{nat}),$
$(\forall\ k,\ (k < n)\%\mathrm{nat} \to f\ k == g\ k) \to (prod\ f\ n) == (prod\ g\ n)$.

Lemma $prod\_le\_compat : \forall\ (f\ g{:}\ nat \to U)\ (n{:}\mathrm{nat}),$
$(\forall\ k,\ (k < n)\%\mathrm{nat} \to f\ k \le \mathtt{g}\ k) \to prod\ f\ n \le prod\ g\ n$.

Lemma $prod\_zero : \forall\ f\ n\ k,\ (k{<}\mathrm{n})\%\mathrm{nat} \to \mathbf{f}\ k == 0 \to prod\ f\ n == 0$.

Lemma $prod\_not\_zero : \forall\ f\ n,\ (\forall\ k,\ (k{<}\mathrm{n})\%\mathrm{nat} \to 0 < f\ k\ ) \to 0 < prod\ f\ n$.

Lemma $prod\_zero\_elim : \forall\ f\ n,\ prod\ f\ n == 0 \to exc\ (\mathtt{fun}\ \mathsf{k} \Rightarrow (k{<}\mathrm{n})\%\mathrm{nat} \wedge f\ k == 0)$.

Hint Resolve $prod\_eq\_compat\ prod\_le\_compat\ prod\_not\_zero$.

Lemma $prod\_le : \forall\ f\ n\ k,\ (k{<}\mathrm{n})\%\mathrm{nat} \to prod\ f\ n \le f\ k$.

Lemma $prod\_minus : \forall\ f\ n,\ prod\ f\ n$ - $prod\ f\ (S\ n) == ([1\text{-}]f\ n) \times prod\ f\ n$.

Definition $Prod : (nat \to U) \to nat \xrightarrow{m} - U$.
Defined.

Lemma $Prod\_simpl : \forall\ f\ n,\ Prod\ f\ n = prod\ f\ n$.
Hint Resolve $Prod\_simpl$.

5.17 Properties of *Unth*

Lemma $Unth\_zero : [1/]1{+}0 == 1$.

Notation "[1/2]" := $(Unth\ 1)$.

Lemma $Unth\_one : [1/2] == [1\text{-}]\ [1/2]$.

Hint Resolve $Unth\_zero\ Unth\_one$.

Lemma $Unth\_one\_plus : [1/2] + [1/2] == 1$.
Hint Resolve $Unth\_one\_plus$.

Lemma $Unth\_one\_refl : \forall\ t,\ [1/2] \times t + [1/2] \times t == t$.

Lemma $Unth\_not\_null : \forall\ n,\ \neg\ (0 == [1/]1{+}\mathrm{n})$.
Hint Resolve $Unth\_not\_null$.

Lemma $Unth\_lt\_zero : \forall\ n,\ 0 < [1/]1{+}\mathrm{n}$.
Hint Resolve $Unth\_lt\_zero$.

Lemma $Unth\_inv\_lt\_one : \forall\ n,\ [1\text{-}][1/]1{+}\mathrm{n}{<}1$.
Hint Resolve $Unth\_inv\_lt\_one$.

Lemma $Unth\_not\_one : \forall\ n,\ \neg\ (1 == [1\text{-}][1/]1{+}\mathrm{n})$.
Hint Resolve $Unth\_not\_one$.

Lemma $Unth\_prop\_sigma : \forall\ n,\ [1/]1{+}\mathrm{n} == [1\text{-}]\ (sigma\ (\mathtt{fun}\ k \Rightarrow [1/]1{+}\mathrm{n})\ n)$.
Hint Resolve $Unth\_prop\_sigma$.

Lemma $Unth\_sigma\_n : \forall\ n : nat,\ \neg\ (1 == sigma\ (\mathtt{fun}\ k \Rightarrow [1/]1{+}\mathrm{n})\ n)$.

Lemma $Unth\_sigma\_Sn : \forall\ n : nat,\ 1 == sigma\ (\mathtt{fun}\ k \Rightarrow [1/]1{+}\mathrm{n})\ (S\ n)$.

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.*

5.17.1 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.*

5.17.2 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, [1/2]^n == [1/2]^(S *n*) + [1/2]^(S *n*).

5.18 Diff function : $|x - y|$

```
Definition diff (x y:U) := (x - y) + (y - x).

Lemma diff_eq : ∀ x, diff x x == 0.
Hint Resolve diff_eq.

Lemma diff_sym : ∀ x y, diff x y == diff y x.
Hint Resolve diff_sym.

Lemma diff_zero : ∀ x, diff x 0 == x.
Hint Resolve diff_zero.

Add Morphism diff with signature Oeq ==>Oeq ==>Oeq as diff_eq_compat.
Qed.
Hint Immediate diff_eq_compat.

Lemma diff_plus_ok : ∀ x y, x - y ≤ [1-](y - x).
Hint Resolve diff_plus_ok.

Lemma diff_Uminus : ∀ x y, x ≤ y → diff x y == y - x.

Lemma diff_Uplus_le : ∀ x y, x ≤ diff x y + y.
Hint Resolve diff_Uplus_le.

Lemma diff_triangular : ∀ x y z, diff x y ≤ diff x z + diff y z.
Hint Resolve diff_triangular.
```

5.19 Density

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

Lemma Ule_nth_lim : ∀ x y, (∀ p, x ≤ y + [1/]1+p) → x ≤ y.
```

5.20 Properties of least upper bounds

```
Lemma lub_un : mlub (cte nat 1) == 1.
Hint Resolve lub_un.

Lemma UPlusk_eq : ∀ k, UPlus k == mon (Uplus k).

Lemma UMultk_eq : ∀ k, UMult k == mon (Umult k).

Lemma UPlus_continuous_right : ∀ k, continuous (UPlus k).
Hint Resolve UPlus_continuous_right.

Lemma UPlus_continuous_left : continuous UPlus.
Hint Resolve UPlus_continuous_left.

Lemma UMult_continuous_right : ∀ k, continuous (UMult k).
Hint Resolve UMult_continuous_right.

Lemma UMult_continuous_left : continuous UMult.
Hint Resolve UMult_continuous_left.

Lemma lub_eq_plus_cte_left : ∀ (f:nat -m-> U) (k:U), lub ((UPlus k) @ f) == k + lub f.
Hint Resolve lub_eq_plus_cte_left.

Lemma lub_eq_mult : ∀ (k:U) (f:nat -m-> U), lub ((UMult k) @ f) == k × lub f.
Hint Resolve lub_eq_mult.

Lemma lub_eq_plus_cte_right : ∀ (f : nat -m-> U) (k:U),
           lub ((mshift UPlus k) @ f) == lub f + k.
Hint Resolve lub_eq_plus_cte_right.

Lemma min_lub_le : ∀ f g : nat -m-> U,
          lub ((Min @2 f) g) ≤ min (lub f) (lub g).
```

Lemma *min_lub_le_incr_aux* : $\forall\ f\ g : nat \xrightarrow{m} U$,
$(\forall\ n,\ exc\ (\mathsf{fun}\ m \Rightarrow (n<=\mathrm{m})\%\mathrm{nat} \wedge f\ n \leq g\ m))$
$\rightarrow \mathsf{min}\ (lub\ f)\ (lub\ g) \leq lub\ ((Min\ @2\ f)\ g)$.

Lemma *min_lub_le_incr* : $\forall\ f\ g : nat \xrightarrow{m} U$,
$min\ (lub\ f)\ (lub\ g) \leq lub\ ((Min\ @2\ f)\ g)$.

Lemma *lub_eq_esp_right* :
$\forall\ (f : nat \xrightarrow{m} U)\ (\mathsf{k} : U),\ lub\ ((mshift\ \mathsf{UEsp\ k})\ @\ f) == lub\ f\ \&\ k$.
Hint Resolve *lub_eq_esp_right*.

Lemma *Udiv_continuous* : $\forall\ (k{:}\mathrm{U}),\ continuous\ (UDiv\ k)$.
Hint Resolve *Udiv_continuous*.

5.21 Greatest lower bounds

Definition *glb* $(f{:}\mathrm{nat} \xrightarrow{m} -\ U) := [1\text{-}](lub\ (UInv\ @\ f))$.

Lemma *glb_le*: $\forall\ (f : \mathsf{nat} \xrightarrow{m} -\ U)\ (n : nat),\ glb\ f \leq (f\ n)$.

Lemma *le_glb*: $\forall\ (f : nat \xrightarrow{m} -\ U)\ (x{:}\mathrm{U}),\ (\forall\ n : nat,\ x \leq f\ n) \rightarrow x \leq glb\ f$.
Hint Resolve *glb_le le_glb*.

Definition *Uopp* : $cpo\ (o{:=}\mathrm{Iord}\ U)\ U$.
Defined.

Lemma Uopp_lub_simpl
: $\forall\ h : nat \xrightarrow{m} -\ U,\ lub\ (cpo{:=}\mathrm{Uopp})\ h = glb\ h$.

Lemma *Uopp_mon_seq* : $\forall\ f{:}\mathrm{nat} \xrightarrow{m} -\ U$,
$\forall\ \mathsf{n}\ m{:}\mathrm{nat},\ (n \leq m)\%\mathrm{nat} \rightarrow f\ m \leq f\ n$.
Hint Resolve *Uopp_mon_seq*.

Infinite product $\Pi_{i=0}^{\infty} f\ i$ Definition *prod_inf* $(f : nat \rightarrow U) : U := glb\ (Prod\ f)$.

Properties of glb

Lemma *glb_le_compat*:
$\forall\ f\ g : nat \xrightarrow{m} -\ U,\ (\forall\ x, f\ x \leq g\ x) \rightarrow glb\ f \leq glb\ g$.
Hint Resolve *glb_le_compat*.

Lemma *glb_eq_compat*:
$\forall\ f\ g : nat \xrightarrow{m} -\ U, f == g \rightarrow glb\ f == glb\ g$.
Hint Resolve *glb_eq_compat*.

Lemma *glb_cte*: $\forall\ c : U,\ glb\ (mon\ (cte\ nat\ (\mathsf{o1}{:=}(\mathrm{Iord}\ U))\ c)) == c$.
Hint Resolve *glb_cte*.

Lemma *glb_eq_plus_cte_right*:
$\forall\ (f : nat \xrightarrow{m} -\ \mathsf{U})\ (k : U),\ glb\ (Imon\ (mshift\ UPlus\ k)\ @\ f) == glb\ f + k$.
Hint Resolve *glb_eq_plus_cte_right*.

Lemma *glb_eq_plus_cte_left*:
$\forall\ (f : nat \xrightarrow{m} -\ U)\ (k : U),\ glb\ (Imon\ (UPlus\ k)\ @\ f) == k + glb\ f$.
Hint Resolve *glb_eq_plus_cte_left*.

Lemma *glb_eq_mult*:
$\forall\ (k : U)\ (f : nat \xrightarrow{m} -\ U),\ glb\ (Imon\ (UMult\ k)\ @\ f) == k \times glb\ f$.

Lemma *Imon2_plus_continuous*
: $continuous2\ (c1{:=}\mathrm{Uopp})\ (\mathsf{c2}{:=}\mathrm{Uopp})\ (c3{:=}\mathrm{Uopp})\ (imon2\ Uplus)$.

Hint Resolve *Imon2_plus_continuous*.

Lemma *Uinv_continuous* : $continuous\ (c1{:=}\mathrm{Uopp})\ UInv$.

Definition *UInvopp* : $U \xrightarrow{m} -\ U$.

```
Defined.
Lemma UInvopp_simpl : ∀ x, UInvopp x = [1-]x.
Lemma Uinvopp_continuous : continuous (c2:=Uopp) UInvopp.
Hint Resolve Uinv_continuous Uinvopp_continuous.
Instance Uminus_mon2 : monotonic2 (o2:=Iord U) Uminus.
Save.
Definition UMinus : U -m> U -m> U := mon2 Uminus.
Lemma UMinus_simpl : ∀ x y, UMinus x y = x - y.
Lemma Uminus_continuous2 : continuous2 (c2:=Uopp) UMinus.
Hint Resolve Uminus_continuous2.
Lemma glb_le_esp : ∀ f g :nat -m> U, (glb f) & (glb g) ≤ glb ((imon2 Uesp @2 f) g).
Hint Resolve glb_le_esp.
Lemma Uesp_min : ∀ a1 a2 b1 b2, min a1 b1 & min a2 b2 ≤ min (a1 & a2) (b1 & b2).
```

Defining lubs of arbitrary sequences

```
Fixpoint seq_max (f:nat → U) (n:nat) : U := match n with
                O ⇒ f O | S p ⇒ max (seq_max f p) (f (S p)) end.
Lemma seq_max_incr : ∀ f n, seq_max f n ≤ seq_max f (S n).
Hint Resolve seq_max_incr.
Lemma seq_max_le : ∀ f n, f n ≤ seq_max f n.
Hint Resolve seq_max_le.
Instance seq_max_mon : ∀ (f:nat → U), monotonic (seq_max f).
Save.
Definition sMax (f:nat → U) : nat -m> U := mon (seq_max f).
Lemma sMax_mult : ∀ k (f:nat->U), sMax (fun n ⇒ k × f n) == UMult k @ sMax f.
Lemma sMax_plus_cte_right : ∀ k (f:nat-> U),
    sMax (fun n ⇒ f n + k) == mshift UPlus k @ sMax f.
Definition Ulub (f:nat → U) := lub (sMax f).
Lemma le_Ulub : ∀ f n, f n ≤ Ulub f.
Lemma Ulub_le : ∀ f x, (∀ n, f n ≤ x) → Ulub f ≤ x.
Hint Resolve le_Ulub Ulub_le.
Lemma Ulub_le_compat : ∀ f g : nat->U, f ≤ g → Ulub f ≤ Ulub g.
Hint Resolve Ulub_le_compat.
Add Morphism Ulub with signature Oeq ==>Oeq as Ulub_eq_compat.
Save.
Hint Resolve Ulub_eq_compat.
Lemma Ulub_eq_mult : ∀ k (f:nat->U), Ulub (fun n ⇒ k × f n)== k × Ulub f.
Lemma Ulub_eq_plus_cte_right : ∀ (f:nat->U) k, Ulub (fun n ⇒ f n + k)== Ulub f + k.
Hint Resolve Ulub_eq_mult Ulub_eq_plus_cte_right.
Lemma Ulub_eq_esp_right :
  ∀ (f : nat → U) (k : U), Ulub (fun n ⇒ f n & k) == Ulub f & k.
Hint Resolve lub_eq_esp_right.
Lemma Ulub_le_plus : ∀ f g, Ulub (fun n ⇒ f n + g n) ≤ Ulub f + Ulub g.
Hint Resolve Ulub_le_plus.
Definition Uglb (f:nat → U) :U := [1-]Ulub (fun n ⇒ [1-](f n)).
Lemma Uglb_le: ∀ (f : nat → U) (n : nat), Uglb f ≤ f n.
```

Lemma *le_Uglb*: $\forall$ (f : *nat* $\to$ *U*) (x:U), ($\forall$ n : *nat*, $x \leq f\ n$)$\to$ $x \leq$ *Uglb f*.
Hint Resolve *Uglb_le le_Uglb*.

Lemma **Uglb_le_compat** : $\forall$ f g : *nat*->U, $f \leq g \to$ Uglb $f \leq$ *Uglb g*.
Hint Resolve *Uglb_le_compat*.

Add *Morphism Uglb* with *signature Oeq* $\Longrightarrow$*Oeq* as *Uglb_eq_compat*.
Save.
Hint Resolve *Uglb_eq_compat*.

Lemma *Uglb_eq_plus_cte_right*:
$\forall$ (f : *nat* $\to$ *U*) (k : *U*), *Uglb* (fun $n \Rightarrow f\ n + k$) == *Uglb f* + *k*.
Hint Resolve **Uglb_eq_plus_cte_right**.

Lemma *Uglb_eq_mult*:
$\forall$ (k : *U*) (f : *nat* $\to$ *U*), *Uglb* (fun $n \Rightarrow$ k $\times$ $f\ n$) == $k \times$ *Uglb* f.
Hint Resolve *Uglb_eq_mult Uglb_eq_plus_cte_right*.

Lemma *Uglb_le_plus* : $\forall$ f g, *Uglb f* + *Uglb g* $\leq$ *Uglb* (fun $n \Rightarrow f\ n + g\ n$).
Hint Resolve *Uglb_le_plus*.

Lemma Ulub_lub : $\forall$ f:nat $\xrightarrow{m}$ *U*, *Ulub f* == *lub f*.
Hint Resolve Ulub_lub.

Lemma *Uglb_glb* : $\forall$ f:nat $\xrightarrow{m}$ $-$ U, *Uglb f* == *glb f*.
Hint Resolve *Uglb_glb*.

Lemma *lub_le_plus* : $\forall$ (f g : *nat* $\xrightarrow{m}$ *U*), *lub* ((*UPlus* @2 f) g) $\leq$ *lub f* + *lub g*.
Hint Resolve *lub_le_plus*.

Lemma *glb_le_plus* : $\forall$ (f g:nat $\xrightarrow{m}$ $-$ *U*) , *glb f* + *glb g* $\leq$ *glb* ((*Imon2 UPlus* @2 f) g).
Hint Resolve *glb_le_plus*.

Lemma *lub_eq_plus* : $\forall$ f g : *nat* $\xrightarrow{m}$ *U*, *lub* ((*UPlus* @2 f) g) == *lub f* + *lub g*.
Hint Resolve *lub_eq_plus*.

Lemma *glb_mon* : $\forall$ f : *nat* $\xrightarrow{m}$ *U*, *Uglb f* == f *O*.

Lemma *lub_inv* : $\forall$ (f g : **nat** $\xrightarrow{m}$ *U*), ($\forall$ n, $f\ n \leq$ [1-] g n) $\to$ lub $f \leq$ [1-] (*lub g*).

Lemma *glb_lift_left* : $\forall$ (f:nat $\xrightarrow{m}$ $-$ *U*) n,
glb f == **glb** (*mon* (*seq_lift_left f n*)).
Hint Resolve *glb_lift_left*.

Lemma *Ulub_mon* : $\forall$ f : *nat* $\xrightarrow{m}$ $-$ *U*, *Ulub* f == f *O*.

Lemma *lub_glb_le* : $\forall$ (f:nat $\xrightarrow{m}$ *U*) (g:nat $\xrightarrow{m}$ $-$ *U*),
($\forall$ n, $f\ n \leq g\ n$) $\to$ *lub f* $\leq$ *glb g*.

Lemma *lub_lub_inv_le* : $\forall$ f g :nat $\xrightarrow{m}$ *U*,
($\forall$ n, $f\ n \leq$ [1-]$g\ n$) $\to$ *lub f* $\leq$ [1-] *lub g*.

Lemma *Uplus_opp_continuous_right* :
$\forall$ k, *continuous* (*c1*:=Uopp) (*c2*:=Uopp) (*Imon* (*UPlus k*)).

Lemma *Uplus_opp_continuous_left* :
continuous (*c1*:=Uopp) (*c2*:=fmon_cpo (*o*:=Iord *U*) (*c*:=Uopp))(Imon2 *UPlus*).

Hint Resolve *Uplus_opp_continuous_right Uplus_opp_continuous_left*.

Lemma *glb_eq_plus* : $\forall$ (f g : *nat* $\xrightarrow{m}$ $-$ *U*), *glb* ((*Imon2 UPlus* @2 f) g) == *glb f* + glb *g*.
Hint Resolve *glb_eq_plus*.

Instance *sigma_mon* :monotonic *sigma*.
Save.

BUG V8.2-1 Definition Sigma : (nat -> U) -m> nat-m> U := mon sigma. Anomaly: Non-functional construction. Please report.

Definition *Sigma* : (*nat* $\to$ *U*) $\xrightarrow{m}$ *nat*$\xrightarrow{m}$ *U*

$:= mon\ sigma$ (*fmonotonic*:=sigma_mon).

Lemma *Sigma_simpl* : $\forall f$, *Sigma* f = *sigma* f.

Lemma *sigma_continuous1* : *continuous Sigma*.

Lemma *sigma_lub1* : $\forall$ (f : $nat \xrightarrow{m} (nat \to U)$) n,
sigma (*lub* f) n == *lub* ((mshift Sigma n) @ f).

Definition *MF* (A:Type) : Type := $A \to U$.

Instance *MFcpo* (A:Type) : *cpo* (*MF* A).
Defined.

Definition *MFopp* (A:Type) : *cpo* (o:=Iord ($A \to U$)) (*MF* A).
Defined.

Lemma *MFopp_lub_eq* : $\forall$ (A:Type) (h:nat-m-> *MF* A),
lub (*cpo*:=MFopp A) h == fun $x \Rightarrow$ *glb* (*Iord_app* x @ h).

Lemma *fle_intro* : $\forall$ (A:Type) (f g : *MF* A), ($\forall$ x, f $x \leq$ g x) $\to$ $f \leq g$.
Hint Resolve *fle_intro*.

Lemma feq_intro : $\forall$ (A:Type) (f g : *MF* A), ($\forall$ x, f x == g x) $\to$ f == g.
Hint Resolve *feq_intro*.

Definition *fplus* (A:Type) (f g : *MF* A) : *MF* A :=
fun $x \Rightarrow f\ x + g\ x$.

Definition *fmult* (A:Type) (k:U) (f : *MF* A) : *MF* A :=
fun $x \Rightarrow k \times f\ x$.

Definition *finv* (A:Type) (f : *MF* A) : *MF* A :=
fun $x \Rightarrow$ [1-] $f\ x$.

Definition *fzero* (A:Type) : *MF* A :=
fun $x \Rightarrow 0$.

Definition *fdiv* (A:Type) (k:U) (f : *MF* A) : *MF* A :=
fun $x \Rightarrow (f\ x)\ /\ k$.

Definition *flub* (A:Type) (f : $nat \xrightarrow{m}$ *MF* A) : *MF* A := *lub* f.

Lemma *fplus_simpl* : $\forall$ (A:Type)(f g : *MF* A) (x : A),
fplus f g $x = f\ x + g\ x$.

Lemma *fmult_simpl* : $\forall$ (A:Type)(k:U) (f : *MF* A) (x : A),
fmult k f $x = k \times f\ x$.

Lemma *fdiv_simpl* : $\forall$ (A:Type)(k:U) (f : *MF* A) (x : A),
fdiv k f x $= f\ x\ /\ k$.

Implicit Arguments *fzero* [].

Lemma *fzero_simpl* : $\forall$ (A:Type)(x : A), *fzero* A $x = 0$.

Lemma *finv_simpl* : $\forall$ (*A*:Type)(f : *MF* A) (x : A), *finv* f x = [1-]f x.

Lemma *flub_simpl* : $\forall$ (A:Type)(f:nat $\xrightarrow{m}$ *MF* A) (x:A),
(*flub* f) x = *lub* ($f \diamond x$).

Hint Resolve *fplus_simpl fmult_simpl fzero_simpl finv_simpl flub_simpl*.

Definition *fone* (A:Type) : MF A := fun $x \Rightarrow 1$.
Implicit Arguments *fone* [].

Lemma *fone_simpl* : $\forall$ (A:Type) (x:A), *fone* A $x = 1$.

Definition *fcte* (A:Type) (k:U): *MF* A := fun $x \Rightarrow k$.
Implicit Arguments *fcte* [].

Lemma fcte_simpl : $\forall$ (A:Type) (k:U) (x:A), *fcte* A k $x = k$.

```
Definition fminus (A:Type) (f g :MF A) : MF A := fun x ⇒ f x - g x.
Lemma fminus_simpl : ∀ (A:Type) (f g: MF A) (x:A), fminus f g x = f x - g x.
Definition fesp (A:Type) (f g :MF A) : MF A := fun x ⇒ f x & g x.
Lemma fesp_simpl : ∀ (A:Type) (f g: MF A) (x:A), fesp f g x = f x & g x.
Definition fconj (A:Type)(f g:MF A) : MF A := fun x ⇒ f x × g x.
Lemma fconj_simpl : ∀ (A:Type) (f g: MF A) (x:A), fconj f g x = f x × g x.
Lemma MF_lub_simpl : ∀ (A:Type) (f : nat -m> MF A) (x:A),
                lub f x = lub (f <o>x).
Hint Resolve MF_lub_simpl.
```

5.21.1 Defining morphisms

```
Lemma fplus_eq_compat : ∀ A (f1 f2 g1 g2:MF A),
           f1==f2 → g1==g2 → fplus f1 g1 == fplus f2 g2.
Add Parametric Morphism (A:Type) : (@fplus A)
    with signature Oeq ⟹Oeq ⟹Oeq
    as fplus_feq_compat_morph.
Save.
Instance fplus_mon2 : ∀ A, monotonic2 (fplus (A:=A)).
Save.
Hint Resolve fplus_mon2.
Lemma fplus_le_compat : ∀ A (f1 f2 g1 g2:MF A),
           f1<=f2 → g1<=g2 → fplus f1 g1 ≤ fplus f2 g2.
Add Parametric Morphism A : (@fplus A) with signature Ole ++> Ole ++> Ole
    as fplus_fle_compat_morph.
Save.
Lemma finv_eq_compat : ∀ A (f g:MF A), f==g → finv f == finv g.
Add Parametric Morphism A : (@finv A) with signature Oeq ⟹Oeq
    as finv_feq_compat_morph.
Save.
Instance finv_mon : ∀ A, monotonic (o2:=Iord (MF A)) (finv (A:=A)).
Save.
Hint Resolve finv_mon.
Lemma finv_le_compat : ∀ A (f g:MF A), f ≤ g → finv g ≤ finv f.
Add Parametric Morphism A: (@finv A)
   with signature Ole -> Ole as finv_fle_compat_morph.
Save.
Lemma fmult_eq_compat : ∀ A k1 k2 (f1 f2:MF A),
           k1 == k2 → f1 == f2 → fmult k1 f1 == fmult k2 f2.
Add Parametric Morphism A : (@fmult A)
   with signature Oeq ⟹Oeq ⟹Oeq as fmult_feq_compat_morph.
Save.
Instance fmult_mon2 : ∀ A, monotonic2 (fmult (A:=A)).
Save.
Hint Resolve fmult_mon2.
Lemma fmult_le_compat : ∀ A k1 k2 (f1 f2:MF A),
           k1 ≤ k2 → f1 ≤ f2 → fmult k1 f1 ≤ fmult k2 f2.
Add Parametric Morphism A : (@fmult A)
```

```
    with signature Ole ++> Ole ++> Ole as fmult_fle_compat_morph.
Save.

Lemma fminus_eq_compat : ∀ A (f1 f2 g1 g2:MF A),
          f1 == f2 → g1 == g2 → fminus f1 g1 == fminus f2 g2.

Add Parametric Morphism A : (@fminus A)
    with signature Oeq ⟹Oeq ⟹Oeq as fminus_feq_compat_morph.
Save.

Instance fminus_mon2 : ∀ A, monotonic2 (o2:=Iord (MF A)) (fminus (A:=A)).
Save.
Hint Resolve fminus_mon2.

Lemma fminus_le_compat : ∀ A (f1 f2 g1 g2:MF A),
          f1 ≤ f2 → g2 ≤ g1 → fminus f1 g1 ≤ fminus f2 g2.

Add Parametric Morphism A : (@fminus A)
    with signature Ole ++> Ole -> Ole as fminus_fle_compat_morph.
Save.

Lemma fesp_eq_compat : ∀ A (f1 f2 g1 g2:MF A),
          f1==f2 → g1==g2 → fesp f1 g1 == fesp f2 g2.

Add Parametric Morphism A : (@fesp A) with signature Oeq ⟹Oeq ⟹Oeq as fesp_feq_compat_morph.
Save.

Instance fesp_mon2 : ∀ A, monotonic2 (fesp (A:=A)).
Save.
Hint Resolve fesp_mon2.

Lemma fesp_le_compat : ∀ A (f1 f2 g1 g2:MF A),
          f1<=f2 → g1<=g2 → fesp f1 g1 ≤ fesp f2 g2.

Add Parametric Morphism A : (@fesp A)
   with signature Ole ++> Ole ++> Ole as fesp_fle_compat_morph.
Save.

Lemma fconj_eq_compat : ∀ A (f1 f2 g1 g2:MF A),
          f1==f2 → g1==g2 → fconj f1 g1 == fconj f2 g2.

 Add Parametric Morphism A : (@fconj A)
   with signature Oeq ⟹Oeq ⟹Oeq
 as fconj_feq_compat_morph.
 Save.

Instance fconj_mon2 : ∀ A, monotonic2 (fconj (A:=A)).
Save.
Hint Resolve fconj_mon2.

Lemma fconj_le_compat : ∀ A (f1 f2 g1 g2:MF A),
          f1 ≤ f2 → g1 ≤ g2 → fconj f1 g1 ≤ fconj f2 g2.

Add Parametric Morphism A : (@fconj A) with signature Ole ++> Ole ++> Ole
as fconj_fle_compat_morph.
Save.

Hint Immediate fplus_le_compat fplus_eq_compat fesp_le_compat fesp_eq_compat
fmult_le_compat fmult_eq_compat fminus_le_compat fminus_eq_compat
fconj_eq_compat.

Hint Resolve finv_eq_compat.
```

5.21.2 Elementary properties

```
Lemma fle_fplus_left : ∀ (A:Type) (f g : MF A), f ≤ fplus f g.
```

Lemma *fle_fplus_right* : ∀ (*A*:Type) (*f g* : *MF A*), *g* ≤ *fplus f g*.

Lemma *fle_fmult* : ∀ (*A*:Type) (*k*:U)(f : *MF A*), *fmult k f* ≤ *f*.

Lemma *fle_zero* : ∀ (*A*:Type) (*f* : *MF A*), *fzero A* ≤ *f*.

Lemma *fle_one* : ∀ (*A*:Type) (*f* : *MF A*), *f* ≤ *fone A*.

Lemma *feq_finv_finv* : ∀ (*A*:Type) (*f* : *MF A*), *finv* (*finv f*) == *f*.

Lemma *fle_fesp_left* : ∀ (*A*:Type) (*f g* : *MF A*), *fesp f g* ≤ *f*.

Lemma *fle_fesp_right* : ∀ (*A*:Type) (*f g* : *MF A*), fesp *f g* ≤ g.

Lemma *fle_fconj_left* : ∀ (*A*:Type) (*f g* : MF *A*), fconj f *g* ≤ *f*.

Lemma fle_fconj_right : ∀ (*A*:Type) (*f g* : *MF A*), *fconj f g* ≤ *g*.

Lemma *fconj_decomp* : ∀ *A* (*f g* : *MF A*),
f == *fplus* (*fconj f g*) (*fconj f* (*finv g*)).
Hint Resolve fconj_decomp.

5.21.3 Compatibility of addition of two functions

Definition *fplusok* (*A*:Type) (*f g* : *MF A*) := *f* ≤ *finv g*.
Hint Unfold *fplusok*.

Lemma *fplusok_sym* : ∀ (*A*:Type) (*f g* : *MF A*) , *fplusok f g* → *fplusok g f*.
Hint Immediate *fplusok_sym*.

Lemma *fplusok_inv* : ∀ (*A*:Type) (*f* : *MF A*) , *fplusok f* (*finv f*).
Hint Resolve *fplusok_inv*.

Lemma *fplusok_le_compat* : ∀ (A:Type)(f1 *f2 g1 g2*:MF *A*),
fplusok f2 g2 → *f1* ≤ *f2* → *g1* ≤ *g2* → *fplusok f1 g1*.

Hint Resolve *fle_fplus_left fle_fplus_right fle_zero fle_one feq_finv_finv finv_le_compat fle_fmult fle_fesp_left fle_fesp_right fle_fconj_left fle_fconj_right*.

Lemma *fconj_fplusok* : ∀ (*A*:Type)(f *g h*:MF *A*),
fplusok g h → *fplusok* (*fconj f g*) (*fconj f h*).
Hint Resolve *fconj_fplusok*.

Definition *Fconj A* : MF *A* $\xrightarrow{m}$ *MF A* $\xrightarrow{m}$ *MF A* := mon2 (*fconj* (*A*:=A)).

Lemma *Fconj_simpl* : ∀ *A f g*, *Fconj A f g* = *fconj f g*.

Lemma *fconj_sym* : ∀ *A* (*f g* : *MF A*), *fconj f g* == *fconj g f*.

Lemma *lub_MF_simpl* : ∀ *A* (*h* : *nat* $\xrightarrow{m}$ *MF A*) (*x*:A), *lub h x* = *lub* (*h* ⋄ *x*).

Instance *fconj_continuous2 A* : *continuous2* (*Fconj A*).
Save.

Definition *Fmult A* : *U* $\xrightarrow{m}$ *MF A* $\xrightarrow{m}$ *MF A* := *mon2* (*fmult* (*A*:=A)).

Lemma *Fmult_simpl* : ∀ *A k f*, *Fmult A k f* = *fmult k f*.

Lemma *fmult_continuous2* : ∀ *A*, *continuous2* (*Fmult A*).

5.22 Fixpoints of functions of type $A \to [0,1]$

Section *FixDef*.
Variable *A* :Type.

Variable *F* : *MF A* $\xrightarrow{m}$ *MF A*.

Definition *mufix* : *MF A* := *fixp F*.

Definition *G* : MF *A* − $\xrightarrow{m}$ − *MF A* := *Imon F*.

Definition *nufix* : *MF A* := *fixp* (*c*:=MFopp *A*) *G*.

Lemma *mufix_inv* : $\forall\ f : MF\ A,\ F\ f \le f \to mufix \le f$.
Hint Resolve *mufix_inv*.

Lemma *nufix_inv* : $\forall\ f$:MF $A, f \le F\ f \to f \le nufix$.
Hint Resolve *nufix_inv*.

Lemma *mufix_le* : $mufix \le F\ mufix$.
Hint Resolve *mufix_le*.

Lemma *nufix_sup* : $F\ nufix \le nufix$.
Hint Resolve *nufix_sup*.

Lemma *mufix_eq* : *continuous* $F \to mufix == F\ mufix$.
Hint Resolve *mufix_eq*.

Lemma *nufix_eq* : *continuous* ($c1$:=MFopp A) ($c2$:=MFopp A) $G \to nufix == F\ nufix$.
Hint Resolve *nufix_eq*.

End *FixDef*.
Hint Resolve *mufix_le mufix_eq nufix_sup nufix_eq*.

Definition *Fcte* (A:Type) (f:MF A) : $MF\ A \xrightarrow{m} MF\ A := mon\ (cte\ (MF\ \mathbf{A})\ f)$.

Lemma *mufix_cte* : $\forall$ (A:Type) (f:MF A), $mufix\ (Fcte\ f) == f$.

Lemma *nufix_cte* : $\forall$ (A:Type) (f:MF A), $nufix\ (Fcte\ f) == f$.

Hint Resolve *mufix_cte nufix_cte*.

5.23 Properties of (pseudo-)barycenter of two points

Lemma *Uinv_bary* :
$\forall\ a\ b\ x\ y : U,\ a \le [1\text{-}]b \to$
$[1\text{-}]\ (a \times x + b \times y) == a \times [1\text{-}]\ x + b \times [1\text{-}]\ y + [1\text{-}]\ (a + b)$.
Hint Resolve *Uinv_bary*.

Lemma *Uinv_bary_le* :
$\forall\ a\ b\ x\ y : U,\ a \le [1\text{-}]b \to a \times [1\text{-}]\ x + b \times [1\text{-}]\ y \le [1\text{-}]\ (a \times x + b \times y)$.
Hint Resolve *Uinv_bary_le*.

Lemma *Uinv_bary_eq* : $\forall\ a\ b\ x\ y : U,\ a == [1\text{-}]b \to$
$[1\text{-}]\ (a \times x + b \times y) == a \times [1\text{-}]\ x + \mathtt{b} \times [1\text{-}]\ y$.
Hint Resolve *Uinv_bary_eq*.

Lemma *bary_refl_eq* : $\forall$ a $b\ x,\ a == [1\text{-}]b \to a \times x + b \times x == x$.
Hint Resolve *bary_refl_eq*.

Lemma *bary_le_left* : $\forall\ a\ b\ x$ y, $[1\text{-}]b \le a \to x \le y \to x \le a \times x + b \times y$.

Lemma *bary_le_right* : $\forall\ a\ b\ x\ y,\ a \le [1\text{-}]b \to x \le y \to a \times x + b \times y \le y$.

Hint Resolve *bary_le_left bary_le_right*.

Lemma *bary_up_eq* : $\forall\ a\ b\ x\ y : U,\ a == [1\text{-}]b \to x \le y \to a \times x + b \times \mathtt{y} == x + b \times (y - x)$.

Lemma *bary_up_le* : $\forall\ a\ b\ x\ y : U,\ a \le [1\text{-}]b \to a \times x + b \times y \le x + b \times (y - x)$.

Lemma *bary_Uminus_left* :
$\forall\ a\ b\ x\ y : U,\ a \le [1\text{-}]b \to (a \times x + b \times y) - x \le b \times (y - x)$.

Lemma *bary_Uminus_left_eq* :
$\forall\ a\ b\ x\ y : U,\ a == [1\text{-}]b \to x \le y \to (a \times x + \mathtt{b} \times y) - x == b \times (y - x)$.

Lemma *Uminus_bary_left*
: $\forall\ a\ b\ x\ y : U, [1\text{-}]a \le b \to x - (a \times x + b \times y) \le b \times (x - y)$.

Lemma *Uminus_bary_left_eq*
: $\forall\ a\ b\ x\ y : U,\ \mathtt{a} == [1\text{-}]b \to y \le x \to x - (a \times x + b \times y) == b \times (x - y)$.

Hint Resolve *bary_up_eq bary_up_le bary_Uminus_left* Uminus_bary_left *bary_Uminus_left_eq Uminus_bary_left_eq*.

Lemma *diff_bary_left_eq*
: $\forall\ a\ b\ x\ y : U,\ a == [1\text{-}]b \to diff\ x\ (a \times x + b \times y) == b \times diff\ x\ y.$
Hint Resolve *diff_bary_left_eq*.

Lemma *Uinv_half_bary* :
$\forall\ x\ y : U, [1\text{-}]\ ([1/2] \times x + [1/2] \times y) == [1/2] \times [1\text{-}]\ x + [1/2] \times [1\text{-}]\ y.$
Hint Resolve *Uinv_half_bary*.

5.24 Properties of generalized sums *sigma*

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

Definition *retract* $(f : nat \to U)\ (n : nat) := \forall\ k, (k < n)\%\text{nat} \to f\ k \le [1\text{-}]\ (sigma\ f\ k)$.

Lemma *retract0* : $\forall\ (f : nat \to U)$, *retract* $f\ 0$.

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

Lemma *retractS*: $\forall\ (f : nat \to U)\ (n :$ **nat**$)$, *retract* $f\ (S\ n) \to f\ n \le [1\text{-}]\ (sigma\ f\ n)$.

Hint Immediate *retract_pred retractS*.

Lemma *retractS_inv* :
$\forall\ (f : nat \to U)\ (n : nat)$, *retract* $f\ (S\ n) \to$ *sigma* $f\ n \le [1\text{-}]\ f\ n$.
Hint Immediate *retractS_inv*.

Lemma *retractS_intro*: $\forall\ (f : nat \to U)\ (n : nat)$,
retract $f\ n \to f\ n \le [1\text{-}]\ (sigma\ f\ n) \to$ *retract* $f\ (S\ n)$.

Hint Resolve *retract0 retractS_intro*.

Lemma *retract_lt* : $\forall\ (f : nat \to U)$ (n : *nat*), *sigma* $f\ n < 1 \to$ *retract* $f\ n$.

Lemma *retract_unif* :
$\forall\ (f : nat \to U)\ (n : nat)$,
$(\forall\ k, (k<=\text{n})\%\text{nat} \to f\ k \le [1/]1+\text{n}) \to$ *retract* $f\ (S\ n)$.

Hint Resolve *retract_unif*.

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

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

Hint Resolve *sigma_prod_maj*.

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

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

Hint Resolve *sigma_prod_maj sigma_prod_le sigma_prod_ge*.

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

5.25 Product by an integer

5.25.1 Definition of *Nmult n x* written *n */ x*

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.

5.25.2 Condition for n */ x to be exact : $n = 0$ or $x \leq \frac{1}{n}$

```
Definition Nmult_def (n: nat) (x : U) :=
  match n with O ⇒ True | S p ⇒ x ≤ [1/]1+p end.
Lemma Nmult_def_O : ∀ x, Nmult_def O x.
Hint Resolve Nmult_def_O.
Lemma Nmult_def_1 : ∀ x, Nmult_def (S O) x.
Hint Resolve Nmult_def_1.
Lemma Nmult_def_intro : ∀ n x , x ≤ [1/]1+n → Nmult_def (S n) x.
Hint Resolve Nmult_def_intro.
Lemma Nmult_def_Unth: ∀ n , Nmult_def (S n) ([1/]1+n).
Hint Resolve Nmult_def_Unth.
Lemma Nmult_def_pred : ∀ n x, Nmult_def (S n) x → Nmult_def n x.
Hint Immediate Nmult_def_pred.
Lemma Nmult_defS : ∀ n x, Nmult_def (S n) x → x ≤ [1/]1+n.
Hint Immediate Nmult_defS.
Lemma Nmult_def_class : ∀ n p, class (Nmult_def n p).
Hint Resolve Nmult_def_class.
Infix "*/" := Nmult (at level 60) : U_scope.
Add Morphism Nmult_def with signature eq ⟹ Oeq ⟹ iff as Nmult_def_eq_compat.
Save.
Lemma Nmult_def_zero : ∀ n, Nmult_def n 0.
Hint Resolve Nmult_def_zero.
```

5.25.3 Properties of n */ x

```
Lemma Nmult_0 : ∀ (x:U), O*/x = 0.
Lemma Nmult_1 : ∀ (x:U), (S O)*/x = x.
Lemma Nmult_zero : ∀ n, n */ 0 == 0.
Lemma Nmult_SS : ∀ (n:nat) (x:U), S (S n) */x = x + (S n */ x).
Lemma Nmult_2 : ∀ (x:U), 2*/x = x + x.
Lemma Nmult_S : ∀ (n:nat) (x:U), S n */ x == x + (n*/x).
Hint Resolve Nmult_1 Nmult_SS Nmult_2 Nmult_S.
Add Morphism Nmult with signature eq ⟹ Oeq ⟹ Oeq as Nmult_eq_compat.
Save.
Hint Immediate Nmult_eq_compat.
Lemma Nmult_eq_compat_left : ∀ (n:nat) (x y:U), x == y → n */ x == n */ y.
Lemma Nmult_eq_compat_right : ∀ (n m:nat) (x:U), (n = m)%nat → n */ x == m */ x.
Hint Resolve Nmult_eq_compat_right.
Lemma Nmult_le_compat_right : ∀ n x y, x ≤ y → n */ x ≤ n */ y.
Lemma Nmult_le_compat_left : ∀ n m x, (n ≤ m)%nat → n */ x ≤ m */ x.
Hint Resolve Nmult_eq_compat_right Nmult_le_compat_right Nmult_le_compat_left.
Lemma Nmult_le_compat : ∀ (n m:nat) x y, n ≤ m → x ≤ y → n */ x ≤ m */ y.
Hint Immediate Nmult_le_compat.
Instance Nmult_mon2 : monotonic2 Nmult.
Save.
Definition NMult : nat →m U →m U :=mon2 Nmult.
```

```
Lemma Nmult_sigma : ∀ (n:nat) (x:U), n */ x == sigma (fun k ⇒ x) n.
Hint Resolve Nmult_sigma.
Lemma Nmult_Unth_prop : ∀ n:nat, [1/]1+n == [1-] (n*/ ([1/]1+n)).
Hint Resolve Nmult_Unth_prop.
Lemma Nmult_n_Unth: ∀ n:nat, n */ [1/]1+n == [1-] ([1/]1+n).
Lemma Nmult_Sn_Unth: ∀ n:nat, S n */ [1/]1+n == 1.
Hint Resolve Nmult_n_Unth Nmult_Sn_Unth.
Lemma Nmult_ge_Sn_Unth: ∀ n k, (S n ≤ k)%nat → k */ [1/]1+n == 1.
Lemma Nmult_le_n_Unth: ∀ n k, (k ≤ n)%nat → k */ [1/]1+n ≤ [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_Umult_assoc_right plus_Nmult_distr Nmult_Uplus_distr
Nmult_mult_assoc 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 ≤ 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.
Lemma Nmult_Unth_factor :
  ∀ (n m1 m2:nat),
  (n × S m2= S m1)%nat → n */ [1/]1+m1 == [1/]1+m2.
Hint Resolve Nmult_Unth_factor.
Lemma Nmult_def_lt : ∀ n x, n */ x <1 → Nmult_def n x.
Hint Immediate Nmult_def_lt.
```

5.26 Conversion from booleans to U

```
Definition B2U :MF bool := fun (b:bool) ⇒ if b then 1 else 0.
Definition NB2U :MF bool := fun (b:bool) ⇒ if b then 0 else 1.
Lemma B2Uinv : NB2U == finv B2U.
Lemma NB2Uinv : B2U == finv NB2U.
Hint Resolve B2Uinv NB2Uinv.
```

5.27 Particular sequences

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

Definition *pmin* (*p*:U) (*n*:nat) := *p* - ([1/2]^n).

Add *Morphism pmin* with *signature Oeq* ⟹*eq* ⟹*Oeq* as *pmin_eq_compat*.
Save.

5.27.1 Properties of *pmin*

Lemma *pmin_esp_S* : ∀ *p n*, *pmin* (p & *p*) *n* == *pmin p* (*S n*) & *pmin p* (*S n*).

Lemma pmin_esp_le : ∀ *p n*, *pmin p* (*S n*) ≤ [1/2] × (*pmin* (*p* & *p*) *n*) + [1/2].

Lemma *pmin_plus_eq* : ∀ p *n*, *p* ≤ [1/2] → *pmin p* (*S n*) == [1/2] × (*pmin* (*p* + *p*) *n*).

Lemma *pmin_0* : ∀ *p*:U, *pmin p O* == 0.

Lemma *pmin_le* : ∀ (*p*:U) (*n*:nat), *p* - ([1/]1+n) ≤ pmin *p n*.

Hint Resolve *pmin_0 pmin_le*.

Lemma *pmin_le_compat* : ∀ *p* (*n m* : *nat*), *n* ≤ *m* → *pmin p n* ≤ *pmin p m*.
Hint Resolve *pmin_le_compat*.

Instance *pmin_mon* : ∀ *p*, monotonic (*pmin p*).
Save.

Definition *Pmin* (*p*:U) :nat $\xrightarrow{m}$ *U* := *mon* (*pmin p*).

Lemma *le_p_lim_pmin* : ∀ *p*, *p* ≤ *lub* (*Pmin p*).

Lemma *le_lim_pmin_p* : ∀ *p*, *lub* (*Pmin p*) ≤ *p*.
Hint Resolve *le_p_lim_pmin le_lim_pmin_p*.

Lemma *eq_lim_pmin_p* : ∀ *p*, *lub* (*Pmin p*) == *p*.

Hint Resolve *eq_lim_pmin_p*.

Particular case where p = 1

Definition *U1min* := *Pmin* 1.

Lemma *eq_lim_U1min* : *lub U1min* == 1.

Lemma *U1min_S* : ∀ *n*, *U1min* (*S n*) == [1/2]*(U1*min n*) + [1/2].

Lemma *U1min_0* : *U1min O* == 0.

Hint Resolve *eq_lim_U1min U1min_S U1min_0*.

Lemma *glb_half_exp* : *glb* (*UExp* [1/2]) == 0.
Hint Resolve *glb_half_exp*.

Lemma *Ule_lt_half_exp* : ∀ *x y*, (∀ *p*, *x* ≤ *y* + [1/2]^p) → *x* ≤ *y*.

Lemma *half_exp_le_half* : ∀ *p*, [1/2]^(S *p*) ≤ [1/2].
Hint Resolve *half_exp_le_half*.

Lemma *twice_half_exp* : ∀ *p*, [1/2]^(S *p*) + [1/2]^(S *p*) == [1/2]^p.
Hint Resolve *twice_half_exp*.

5.27.2 Dyadic numbers

Fixpoint *exp2* (*n*:nat) : *nat* :=
  match *n* with *O* ⇒ (1%nat) | *S p* ⇒ (2 × (*exp2 p*))%nat end.

Lemma *exp2_pos* : ∀ *n*, (*O* < *exp2 n*)%nat.
Hint Resolve exp2_pos.

Lemma *S_pred_exp2* : ∀ n, *S* (*pred* (*exp2 n*))=exp2 n.
Hint Resolve *S_pred_exp2*.

```
Notation "k /2^ p" := (k */ ([1/2])^p) (at level 35, no associativity).
Lemma Unth_eq : ∀ n p, n*/ p == [1-]p → p == [1/]1+n.
Lemma Unth_half : ∀ n, (O<n)%nat → [1/]1+(pred (n+n)) == [1/2] × [1/]1+pred n.
Lemma Unth_exp2 : ∀ p, [1/2]^p == [1/]1+pred (exp2 p).
Hint Resolve Unth_exp2.
Lemma Nmult_exp2 : ∀ p, (exp2 p)/2^p == 1.
Hint Resolve Nmult_exp2.
```

5.28 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 [(Uesp 0 ?x)] ⇒ setoid_rewrite (Uesp_zero_left x)
  | ⊢ context [(Uesp ?x 0)] ⇒ setoid_rewrite (Uesp_zero_right x)
  | ⊢ context [(Uesp 1 ?x)] ⇒ setoid_rewrite (Uesp_one_left x)
  | ⊢ context [(Uesp ?x 1)] ⇒ setoid_rewrite (Uesp_one_right x)
  | ⊢ context [(Uminus 0 ?x)] ⇒ setoid_rewrite (Uminus_le_zero 0 x);
                                              [idtac|apply (Upos x)]
  | ⊢ context [(Uminus ?x 0)] ⇒ setoid_rewrite (Uminus_zero_right x)
  | ⊢ context [(Uminus ?x 1)] ⇒ setoid_rewrite (Uminus_le_zero x 1);
                                              [idtac|apply (Unit x)]
  | ⊢ context [([1-] ([1-] ?x))] ⇒ setoid_rewrite (Uinv_inv x)
  | ⊢ context [([1-] 1)] ⇒ setoid_rewrite Uinv_one
  | ⊢ context [([1-] 0)] ⇒ setoid_rewrite Uinv_zero
  | ⊢ context [([1/]1+O)] ⇒ setoid_rewrite Unth_zero
  | ⊢ context [?x^O] ⇒ setoid_rewrite (Uexp_0 x)
  | ⊢ context [?x^(S O)] ⇒ setoid_rewrite (Uexp_1 x)
  | ⊢ context [0^(?n)] ⇒ setoid_rewrite Uexp_zero; [idtac|omega]
  | ⊢ context [U1^(?n)] ⇒ setoid_rewrite Uexp_one
  | ⊢ context [(Nmult 0 ?x)] ⇒ setoid_rewrite (Nmult_0 x)
  | ⊢ context [(Nmult 1 ?x)] ⇒ setoid_rewrite (Nmult_1 x)
  | ⊢ context [(Nmult ?n 0)] ⇒ setoid_rewrite (Nmult_zero n)
  | ⊢ context [(sigma ?f O)] ⇒ setoid_rewrite (sigma_0 f)
  | ⊢ context [(sigma ?f (S O))] ⇒ setoid_rewrite (sigma_1 f)
  | ⊢ (Ole (Uplus ?x ?y) (Uplus ?x ?z)) ⇒ apply Uplus_le_compat_right
  | ⊢ (Ole (Uplus ?x ?z) (Uplus ?y ?z)) ⇒ apply Uplus_le_compat_left
  | ⊢ (Ole (Uplus ?x ?z) (Uplus ?z ?y)) ⇒ setoid_rewrite (Uplus_sym z y);
                                                    apply Uplus_le_compat_left
  | ⊢ (Ole (Uplus ?x ?y) (Uplus ?z ?x)) ⇒ setoid_rewrite (Uplus_sym x y);
                                                    apply Uplus_le_compat_left
  | ⊢ (Ole (Uinv ?y) (Uinv ?x)) ⇒ apply Uinv_le_compat
  | ⊢ (Ole (Uminus ?x ?y) (Uplus ?x ?z)) ⇒ apply Uminus_le_compat_right
  | ⊢ (Ole (Uminus ?x ?z) (Uplus ?y ?z)) ⇒ apply Uminus_le_compat_left
  | ⊢ ((Uinv ?x) == (Uinv ?y)) ⇒ apply Uinv_eq_compat
  | ⊢ ((Uplus ?x ?y) == (Uplus ?x ?z)) ⇒ apply Uplus_eq_compat_right
  | ⊢ ((Uplus ?x ?z) == (Uplus ?y ?z)) ⇒ apply Uplus_eq_compat_left
```

```
| ⊢ ((Uplus ?x ?z) == (Uplus ?z ?y)) ⇒ setoid_rewrite (Uplus_sym z y);
                                                    apply Uplus_eq_compat_left
| ⊢ ((Uplus ?x ?y) == (Uplus ?z ?x)) ⇒ setoid_rewrite (Uplus_sym x y);
                                                    apply Uplus_eq_compat_left
| ⊢ ((Uminus ?x ?y) == (Uplus ?x ?z)) ⇒ apply Uminus_eq_compat;[apply Oeq_refl|idtac]
| ⊢ ((Uminus ?x ?z) == (Uplus ?y ?z)) ⇒ apply Uminus_eq_compat;[idtac|apply Oeq_refl]
| ⊢ (Ole (Umult ?x ?y) (Umult ?x ?z)) ⇒ apply Umult_le_compat_right
| ⊢ (Ole (Umult ?x ?z) (Umult ?y ?z)) ⇒ apply Umult_le_compat_left
| ⊢ (Ole (Umult ?x ?z) (Umult ?z ?y)) ⇒ setoid_rewrite (Umult_sym z y);
                                                    apply Umult_le_compat_left
| ⊢ (Ole (Umult ?x ?y) (Umult ?z ?x)) ⇒ setoid_rewrite (Umult_sym x y);
                                                    apply Umult_le_compat_left
| ⊢ ((Umult ?x ?y) == (Umult ?x ?z)) ⇒ apply Umult_eq_compat_right
| ⊢ ((Umult ?x ?z) == (Umult ?y ?z)) ⇒ apply Umult_eq_compat_left
| ⊢ ((Umult ?x ?z) == (Umult ?z ?y)) ⇒ setoid_rewrite (Umult_sym z y);
                                                    apply Umult_eq_compat_left
| ⊢ ((Umult ?x ?y) == (Umult ?z ?x)) ⇒ setoid_rewrite (Umult_sym x y);
                                                    apply Umult_eq_compat_left
end.
```

Properties of current values `Notation "[1/3]" := (Unth 2%nat).`
`Notation "[1/4]" := (Unth 3%nat).`
`Notation "[1/8]" := (Unth 7).`
`Notation "[3/4]" := (Uinv [1/4]).`

`Lemma half_square : [1/2]×[1/2]==[1/4].`

`Lemma half_cube : [1/2]×[1/2]×[1/2]==[1/8].`

`Lemma three_quarter_decomp : [3/4]==[1/2]+[1/4].`

Equality is not true, even for monotonic sequences fot instance n/m

`Lemma Ulub_Uglb_exch_le : ∀ f : nat → nat → U,`
`  Ulub (fun n ⇒ Uglb (fun m ⇒ f n m)) ≤ Uglb (fun m ⇒ Ulub (fun n ⇒ f n m)).`

5.29 Intervals

5.29.1 Definition

`Record IU : Type := mk_IU {low:U; up:U; proper:low ≤ up}.`

`Hint Resolve proper.`

the all set : [0,1] `Definition full := mk_IU 0 1 (Upos 1).`
singleton : [x] `Definition singl (x:U) := mk_IU x x (Ole_refl x).`
down segment : [0,x] `Definition inf (x:U) := mk_IU 0 x (Upos x).`
up segment : [x,1] `Definition sup (x:U) := mk_IU x 1 (Unit x).`

5.29.2 Relations

`Definition Iin (x:U) (I:IU) := low I ≤ x ∧ x ≤ up I.`

`Definition Iincl I J := low J ≤ low I ∧ up I ≤ up J.`

`Definition Ieq I J := low I == low J ∧ up I == up J.`
`Hint Unfold Iin Iincl Ieq.`

5.29.3 Properties

`Lemma Iin_low : ∀ I, Iin (low I) I.`

`Lemma Iin_up : ∀ I, Iin (up I) I.`

Hint Resolve *Iin_low Iin_up*.

Lemma *Iin_singl_elim* : $\forall\ x\ y,\ Iin\ x\ (singl\ y) \to x == y$.

Lemma *Iin_inf_elim* : $\forall\ x\ y,\ Iin\ x\ (inf\ y) \to x \leq y$.

Lemma *Iin_sup_elim* : $\forall\ x\ y,\ Iin\ x\ (sup\ y) \to y \leq x$.

Lemma *Iin_singl_intro* : $\forall\ x\ y,\ x == y \to Iin\ x\ (singl\ y)$.

Lemma *Iin_inf_intro* : $\forall\ x\ y,\ x \leq y \to Iin\ x\ (\mathsf{inf}\ y)$.

Lemma *Iin_sup_intro* : $\forall\ x\ y,\ y \leq x \to Iin\ x\ (sup\ y)$.

Hint Immediate *Iin_inf_elim Iin_sup_elim Iin_singl_elim*.
Hint Resolve *Iin_inf_intro Iin_sup_intro Iin_singl_intro*.

Lemma lin_class : $\forall\ I\ x,\ class\ (Iin\ x\ I)$.

Lemma *Iincl_class* : $\forall\ I\ J,\ class\ (Iincl\ I\ J)$.

Lemma *Ieq_class* : $\forall\ I\ J,\ class\ (Ieq\ I\ J)$.
Hint Resolve *Iin_class Iincl_class Ieq_class*.

Lemma *Iincl_in* : $\forall\ I\ J,\ Iincl\ I\ J \to \forall\ x,\ Iin\ x\ I \to Iin\ x\ J$.

Lemma *Iincl_low* : $\forall\ I\ J,\ Iincl\ I\ J \to low\ J \leq low\ I$.

Lemma *Iincl_up* : $\forall\ I\ J,\ Iincl\ I\ J \to up\ I \leq up\ J$.

Hint Immediate *Iincl_low Iincl_up*.

Lemma *Iincl_refl* : $\forall\ I,\ Iincl\ I\ I$.
Hint Resolve *Iincl_refl*.

Lemma *Iincl_trans* : $\forall\ I\ J\ K,\ Iincl\ I\ J \to Iincl\ J\ K \to Iincl\ I\ K$.

Instance *IUord* : $ord\ IU := \{Oeq := \mathtt{fun}\ I\ J \Rightarrow Ieq\ \mathsf{I}\ J;\ \mathit{Ole} := \mathtt{fun}\ I\ J \Rightarrow Iincl\ J\ I\}$.
Defined.

Lemma *low_le_compat* : $\forall\ I\ J{:}\mathrm{IU},\ I \leq \mathsf{J} \to low\ I \leq low\ J$.

Lemma *up_le_compat* : $\forall\ I\ J : IU,\ I \leq J \to up\ J \leq up\ I$.

Instance *low_mon* : $monotonic\ low$.
Save.

Definition *Low* : $\mathsf{IU} \xrightarrow{m} U := mon\ low$.

Instance *up_mon* : $monotonic\ (o2{:=}\mathrm{Iord}\ U)\ up$.
Save.

Definition *Up* : $IU \xrightarrow{m} -\ U := mon\ (o2{:=}\mathrm{Iord}\ U)\ up$.

Lemma leq_incl : $\forall\ I\ J,\ Ieq\ I\ J \to Iincl\ I\ J$.

Lemma *Ieq_incl_sym* : $\forall\ I\ J,\ Ieq\ I\ J \to Iincl\ J\ I$.
Hint Immediate *Ieq_incl Ieq_incl_sym*.

Lemma *lincl_eq_compat* : $\forall\ I\ J\ K\ L,$
$Ieq\ I\ J \to Iincl\ J\ K \to Ieq\ K\ L \to Iincl\ I\ L$.

Lemma *lincl_eq_trans* : $\forall\ I\ J\ \mathsf{K},$
$Iincl\ I\ J \to Ieq\ J\ K \to Iincl\ I\ K$.

Lemma *Ieq_incl_trans* : $\forall\ I\ J\ K,$
$Ieq\ I\ J \to Iincl\ J\ K \to Iincl\ I\ K$.

Lemma *Iincl_antisym* : $\forall\ I\ J,\ Iincl\ I\ J \to Iincl\ J\ I \to Ieq\ I\ J$.
Hint Immediate *Iincl_antisym*.

Lemma *Ieq_refl* : $\forall\ I,\ Ieq\ I\ I$.
Hint Resolve *Ieq_refl*.

Lemma *Ieq_sym* : $\forall\ I\ J,\ Ieq\ I\ J \to Ieq\ \mathsf{J}\ I$.

```
Hint Immediate Ieq_sym.
Lemma Ieq_trans : ∀ I J K, Ieq I J → Ieq J K → Ieq I K.
Lemma Isingl_eq : ∀ x y, Iincl (singl x) (singl y) → x==y.
Hint Immediate Isingl_eq.
Lemma Iincl_full : ∀ I, Iincl I full.
Hint Resolve Iincl_full.
```

5.29.4 Operations on intervals

```
Definition Iplus I J := mk_IU (low I + low J) (up I + up J)
                                                (Uplus_le_compat _ _ _ _ (proper I) (proper J)).
Lemma low_Iplus : ∀ I J, low (Iplus I J)=low I + low J.
Lemma up_Iplus : ∀ I J, up (Iplus I J)=up I + up J.
Lemma Iplus_in : ∀ I J x y, Iin x I → Iin y J → Iin (x+y) (Iplus I J).
Lemma lplus_in_elim :
∀ I J z, low I ≤ [1-]up J → Iin z (Iplus I J)
                  → exc (fun x ⇒ Iin x I ∧
                                                exc (fun y ⇒ Iin y J ∧ z==x+y)).
Definition Imult I J := mk_IU (low I × low J) (up I × up J)
                                                (Umult_le_compat _ _ _ _ (proper I) (proper J)).
Lemma low_Imult : ∀ I J, low (Imult I J) = low I × low J.
Lemma up_Imult : ∀ I J, up (Imult I J) = up I × up J.
Definition Imultk p I := mk_IU (p × low I) (p × up I) (Umult_le_compat_right p _ _ (proper I)).
Lemma low_Imultk : ∀ p I, low (Imultk p I) = p × low I.
Lemma up_Imultk : ∀ p I, up (Imultk p I) = p × up I.
Lemma Imult_in : ∀ I J x y, Iin x I → Iin y J → Iin (x*y) (Imult I J).
Lemma Imultk_in : ∀ p I x , Iin x I → Iin (p*x) (Imultk p I).
```

5.29.5 Limits of intervals

```
Definition Ilim : ∀ I: nat -m> IU, IU.
Defined.
Lemma low_lim : ∀ (I:nat -m> IU), low (Ilim I) = lub (Low @ I).
Lemma up_lim : ∀ (I:nat -m> IU), up (Ilim I) = glb (Up @ I).
Lemma lim_Iincl : ∀ (I:nat -m> IU) n, Iincl (Ilim I) (I n).
Hint Resolve lim_Iincl.
Lemma Iincl_lim : ∀ J (I:nat -m>IU), (∀ n, Iincl J (I n)) → Iincl J (Ilim I).
Lemma IIim_incl_stable : ∀ (I J:nat -m> IU), (∀ n, Iincl (I n) (J n)) → Iincl (Ilim I) (Ilim J).
Hint Resolve IIim_incl_stable.
Instance IUcpo : cpo IU := {D0:=full; lub:=Ilim}.
Defined.
```

5.30 Limits inf and sup

```
Definition fsup (f:nat->U) (n:nat) := Ulub (fun k ⇒ f (n+k)%nat).
Definition finf (f:nat->U) (n:nat) := Uglb (fun k ⇒ f (n+k)%nat).
```

Lemma *fsup_incr* : $\forall$ (f:nat->U) n, *fsup* f (S n) $\leq$ *fsup* f n.
Hint Resolve fsup_incr.

Lemma *finf_incr* : $\forall$ (f:nat->U) $\mathbf{n}$, *finf* f $\mathbf{n} \leq$ *finf* f (S n).

Hint Resolve *finf_incr*.

Instance *fsup_mon* : $\forall$ f, *monotonic* (*o2*:=Iord U) (*fsup* f).
Save.

Instance *finf_mon* : $\forall$ f, *monotonic* (*finf* f).
Save.

Definition *Fsup* (f:nat->U) : *nat* $\xrightarrow{m}$ $-$ U := *mon* (*fsup* $\mathbf{f}$).
Definition *Finf* (f:nat->U) : *nat* $\xrightarrow{m}$ U := *mon* (*finf* f).

Lemma *fn_fsup* : $\forall$ f n, f $n \leq$ *fsup* f n.
Hint Resolve *fn_fsup*.

Lemma *finf_fn* : $\forall$ f n, *finf* f $n \leq f$ n.
Hint Resolve *finf_fn*.

Definition *limsup* f := *glb* (*Fsup* f).
Definition *liminf* f := *lub* (*Finf* f).

Lemma *le_liminf_sup* : $\forall$ f, *liminf* $f \leq$ *limsup* f.

Hint Resolve *le_liminf_sup*.

Definition *has_lim* f := *limsup* $f \leq$ *liminf* f.

Lemma *eq_liminf_sup* : $\forall$ f, *has_lim* $f \rightarrow$ *liminf* f == *limsup* f.

Definition *cauchy* f := $\forall$ (p:nat), *exc* (fun M:nat $\Rightarrow$ $\forall$ n m,
($M \leq n$)%nat $\rightarrow$ ($M \leq m$)%nat $\rightarrow$ f $n \leq f$ m + [1/2]^p).

Definition *is_limit* f (l:U) := $\forall$ (p:nat), *exc* (fun M:nat $\Rightarrow$ $\forall$ n,
($M \leq n$)%nat $\rightarrow$ f $n \leq l$ + [1/2]^p $\wedge$ l $\leq f$ n + [1/2]^p).

Lemma *cauchy_lim* : $\forall$ f, *cauchy* f $\rightarrow$ *is_limit* f (*limsup* f).

Lemma *has_limit_cauchy* : $\forall$ f l, *is_limit* f l $\rightarrow$ *cauchy* f.

Lemma *limit_le_unique* : $\forall$ f *l1* *l2*, *is_limit* f *l1* $\rightarrow$ *is_limit* f *l2* $\rightarrow$ *l1* $\leq$ l2.

Lemma *limit_unique* : $\forall$ f *l1* *l2*, *is_limit* f l1 $\rightarrow$ is_limit f *l2* $\rightarrow$ *l1* == *l2*.
Hint Resolve *limit_unique*.

Lemma *has_limit_compute* : $\forall$ f l, *is_limit* f l $\rightarrow$ *is_limit* f (*limsup* f).

Lemma *limsup_eq_mult* : $\forall$ k (f : *nat* $\rightarrow$ U),
limsup (fun $n \Rightarrow k \times f$ n) == $k \times$ *limsup* f.

Lemma *liminf_eq_mult* : $\forall$ k (f : *nat* $\rightarrow$ U),
liminf (fun $n \Rightarrow k \times f$ n) == $k \times$ *liminf* f.

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

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

Lemma *limsup_le_plus* : $\forall$ (f g: *nat* $\rightarrow$ U),
limsup (fun $x \Rightarrow f$ x + g x) $\leq$ *limsup* f + *limsup* g.

Lemma *liminf_le_plus* : $\forall$ (f g: *nat* $\rightarrow$ U),
liminf f + *liminf* $g \leq$ *liminf* (fun $x \Rightarrow f$ x + g x).

Hint Resolve *liminf_le_plus* *limsup_le_plus*.

Lemma *limsup_le_compat* : $\forall$ f g : *nat* $\rightarrow$ U, $f \leq g \rightarrow$ *limsup* $f \leq$ *limsup* g.

Lemma *liminf_le_compat* : $\forall$ f g : *nat* $\rightarrow$ U, $f \leq g \rightarrow$ *liminf* $f \leq$ *liminf* g.

Hint Resolve *limsup_le_compat liminf_le_compat.*

Lemma *limsup_eq_compat* : $\forall\ f\ g : nat \to U, f == g \to limsup\ f == limsup\ g.$

Lemma *liminf_eq_compat* : $\forall\ f\ g : nat \to U, f == g \to liminf\ f == liminf\ g.$

Hint Resolve *liminf_eq_compat limsup_eq_compat.*

Lemma *limsup_inv* : $\forall\ f : nat \to U, limsup\ (\mathtt{fun}\ x \Rightarrow [1\text{-}]f\ x) == [1\text{-}]\ liminf\ f.$

Lemma *liminf_inv* : $\forall\ f : nat \to U, liminf\ (\mathtt{fun}\ x \Rightarrow [1\text{-}]f\ x) == [1\text{-}]\ limsup\ f.$

Hint Resolve *limsup_inv liminf_inv.*

5.31 Limits of arbitrary sequences

Lemma *liminf_incr* : $\forall\ f$:nat $\overset{m}{\to} U, liminf\ f == lub\ \mathsf{f}.$

Lemma *limsup_incr* : $\forall\ f$:nat $\overset{m}{\to} U, limsup\ f == lub\ f.$

Lemma *has_limit_incr* : $\forall\ f$:nat $\overset{m}{\to} U, \mathsf{has\_lim}\ f.$

Lemma *liminf_decr* : $\forall\ f$:nat $\overset{m}{\to} -\ U, liminf\ f == glb\ f.$

Lemma *limsup_decr* : $\forall\ f$:nat-m->U, $limsup\ f == glb\ f.$

Lemma *has_limit_decr* : $\forall\ f$:nat $\overset{m}{\to} -\ U, has\_lim\ f.$

Lemma *has_limit_sum* : $\forall\ f\ g$: $nat \to U, has\_lim\ f \to has\_lim\ g \to has\_lim\ (\mathtt{fun}\ x \Rightarrow f\ x + g\ x).$

Lemma *has_limit_inv* : $\forall\ f : nat \to U, has\_lim\ f \to has\_lim\ (\mathtt{fun}\ x \Rightarrow [1\text{-}]f\ x).$

Lemma *has_limit_cte* : $\forall\ c, has\_lim\ (\mathtt{fun}\ n \Rightarrow c).$

5.32 Definition and properties of series : infinite sums

Definition *serie* $(f : nat \to U) : U :=$ *lub* $(sigma\ f).$

Lemma *serie_le_compat* : $\forall\ (f\ g$: $nat \to U),$
$(\forall\ k, f\ \mathsf{k} \leq g\ k) \to serie\ f \leq serie\ g.$

Lemma *serie_eq_compat* : $\forall\ (f\ g$: $nat \to U),$
$(\forall\ k, f\ k == g\ k) \to serie\ f == serie\ g.$

Lemma *serie_sigma_lift* : $\forall\ (f$:nat $\to U)\ (n$:nat),
$serie\ f == sigma\ f\ n + serie\ (\mathtt{fun}\ k \Rightarrow f\ (n + k)\%\text{nat}).$

Lemma serie_S_lift : $\forall\ (\mathsf{f}$:nat $\to U),$
$serie\ f == f\ O + serie\ (\mathtt{fun}\ k \Rightarrow \mathsf{f}\ (S\ k)).$

Lemma *serie_zero* : $\forall\ f, (\forall\ k, f\ k == 0) \to serie\ f == 0.$

Lemma *serie_not_zero* : $\forall\ f\ k, 0 < f\ k \to 0 < serie\ f.$

Lemma *serie_zero_elim* : $\forall\ f, serie\ f == 0 \to \forall\ k, f\ \mathsf{k} == 0.$

Hint Resolve *serie_eq_compat serie_le_compat serie_zero.*

Lemma *serie_le* : $\forall\ f\ k, f\ k \leq serie\ f.$

Lemma *serie_minus_incr* : $\forall\ f$:nat $\overset{m}{\to} U, serie\ (\mathtt{fun}\ k \Rightarrow f\ (S\ k)$ - $f\ k) == \mathsf{lub}\ f$ - $f\ O.$

Lemma *serie_minus_decr* : $\forall\ f : nat \overset{m}{\to} -\ U,$
$serie\ (\mathtt{fun}\ k \Rightarrow f\ k$ - $\mathbf{f}\ (S\ k)) == f\ O$ - $glb\ f.$

Lemma *serie_plus* : $\forall\ (\mathsf{f}\ g : nat \to U),$
$serie\ (\mathtt{fun}\ k \Rightarrow (f\ k) + (g\ k)) == serie\ f + serie\ g.$

Definition *wretract* $(f : nat \to U) := \forall\ k, f\ k \leq [1\text{-}]\ (sigma\ f\ k).$

Lemma *retract_wretract* : $\forall\ f, (\forall\ n, retract\ f\ n) \to wretract\ \mathsf{f}.$

Lemma *wretract_retract* : $\forall\ f, wretract\ f \to \forall\ n, retract\ f\ n.$

Hint Resolve *wretract_retract.*

```
Lemma wretract_lt : ∀ (f : nat → U), (∀ (n : nat), sigma f n < 1) → wretract f.

Lemma retract_zero_wretract :
        ∀ f n, retract f n → (∀ k, (n ≤ k)%nat → f k == 0) → wretract f.

Lemma wretract_le : ∀ f g : nat->U, f ≤ g → wretract g → wretract f.

Lemma serie_mult :
  ∀ (f : nat → U) c, wretract f → serie (fun k ⇒ c × f k) == c × serie f.
Hint Resolve serie_mult.

Lemma serie_prod_maj : ∀ (f g : nat → U),
    serie (fun k ⇒ f k × g k) ≤ serie f.

Hint Resolve serie_prod_maj.

Lemma serie_prod_le : ∀ (f g : nat → U) (c:U), (∀ k, f k ≤ c)
    → wretract g → serie (fun k ⇒ f k × g k) ≤ c × serie g.

Lemma serie_prod_ge : ∀ (f g : nat → U) (c:U), (∀ k, c ≤ (f k))
    → wretract g → c × serie g ≤ serie (fun k ⇒ f k × g k).

Hint Resolve serie_prod_le serie_prod_ge.

Lemma serie_inv_le : ∀ (f g : nat → U), wretract f →
   serie (fun k ⇒ f k × [1-] (g k)) ≤ [1-] (serie (fun k ⇒ f k × g k)).

Definition Serie : (nat → U) -m-> U.
Defined.

Lemma Serie_simpl : ∀ f, Serie f = serie f.

Lemma serie_continuous : continuous Serie.
```

6 Monads.v: Monads for randomized constructions

```
Require Export Uprop.
```

6.1 Definition of monadic operators as the cpo of monotonic oerators

```
Definition M (A:Type) := MF A -m-> U.

Instance app_mon (A:Type) (x:A) : monotonic (fun (f:MF A) ⇒ f x).
Save.

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

Definition star : ∀ (A B:Type), M A → (A → M B) → M B.
Defined.

Lemma star_simpl : ∀ (A B:Type) (a:M A) (F:A → M B)(f:MF B),
        star a F f = a (fun x ⇒ F x f).
```

6.2 Properties of monadic operators

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

Lemma law2 :
 ∀ (A:Type) (a:M A) (f:MF A), 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:MF C), star (star a F) G f == star a (fun x:A ⇒ star (F x) G) f.
```

6.3 Properties of distributions

6.3.1 Expected properties of measures

Definition *stable_inv* (*A*:Type) (*m*:M *A*) : Prop := ∀ *f* :MF *A*, *m* (*finv f*) ≤ [1-] (*m* f).

Definition *stable_plus* (*A*:Type) (*m*:M *A*) : Prop :=
∀ *f g*:MF *A*, *fplusok f g* → *m* (*fplus f g*) == (*m f*) + (*m g*).

Definition *le_plus* (*A*:Type) (*m*:M *A*) : Prop :=
∀ *f g*:MF *A*, *fplusok f g* → (*m f*) + (*m g*) ≤ *m* (*fplus f g*).

Definition *le_esp* (*A*:Type) (*m*:M *A*) : Prop :=
∀ *f g*: *MF A*, (*m f*) & (*m g*) ≤ *m* (*fesp f g*).

Definition *le_plus_cte* (*A*:Type) (*m*:M *A*) : Prop :=
∀ (*f* : *MF A*) (*k*:U), *m* (*fplus f* (*fcte A k*)) ≤ *m f* + *k*.

Definition *stable_mult* (*A*:Type) (*m*:M *A*) : Prop :=
∀ (*k*:U) (*f*:MF *A*), *m* (*fmult k f*) == *k* × (m *f*).

6.3.2 Stability for equality

Lemma *stable_minus_distr* : ∀ (*A*:Type) (*m*:M *A*),
stable_plus m → *stable_inv m* →
∀ (*f g* : *MF A*), *g* ≤ *f* → *m* (*fminus f g*) == *m f* - *m g*.

Hint Resolve *stable_minus_distr*.

Lemma *inv_minus_distr* : ∀ (*A*:Type) (*m*:M *A*),
stable_plus m → *stable_inv m* →
∀ (*f* : *MF A*), *m* (*finv f*) == *m* (*fone* A) - *m f*.
Hint Resolve *inv_minus_distr*.

Lemma *le_minus_distr* : ∀ (*A* : Type)(*m*:M *A*),
∀ (f *g*:A → *U*), *m* (*fminus f g*) ≤ *m f*.
Hint Resolve *le_minus_distr*.

Lemma *le_plus_distr* : ∀ (*A* : Type)(*m*:M *A*),
stable_plus m → *stable_inv m* → ∀ (*f g*:MF *A*), *m* (*fplus f g*) ≤ *m f* + *m g*.
Hint Resolve *le_plus_distr*.

Lemma *le_esp_distr* : ∀ (*A* : Type) (*m*:M *A*),
stable_plus m → *stable_inv m* → *le_esp m*.

Lemma *unit_stable_eq* : ∀ (*A*:Type) (*x*:A), *stable* (*unit x*).

Lemma *star_stable_eq* : ∀ (*A B*:Type) (*m*:M *A*) (*F*:A → *M B*), stable (*star* m *F*).

6.3.3 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 → (∀ *a*:A, *stable_inv* (*F a*)) → *stable_inv* (*star m* F).

6.3.4 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 →
(∀ *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*).

Lemma *unit_le_plus* : ∀ (*A*:Type) (*x*:A), *le_plus* (*unit x*).

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

6.3.5 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 → (∀ *a*:A, *stable_mult* (*F a*)) → *stable_mult* (*star m F*).

6.3.6 Continuity

Lemma *unit_continuous* : ∀ (*A*:Type) (*x*:A), *continuous* (*unit x*).

Lemma *star_continuous* : ∀ (*A B* : Type) (*m* : *M A*)(*F*: *A* → *M B*),
continuous m → (∀ *x*, *continuous* (*F x*)) → *continuous* (*star m* **F**).

7 Probas.v: The monad for distributions

Require Export Monads.

7.1 Definition of distribution

Distributions are monotonic measure functions such that

- $\mu(1-f) \leq 1-\mu(f)$
- $f \leq 1-g \Rightarrow \mu(f+g) = \mu(f)+\mu(g)$
- $\mu(k \times f) = k \times \mu(f)$
- $\mu(lub f_n) <= lub \mu(f_n)$

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_continuous : *continuous mu*}.

Hint Resolve *mu_stable_plus mu_stable_inv mu_stable_mult mu_continuous*.

7.2 Properties of measures

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

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

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

Lemma *mu_zero_eq* : ∀ (*A* : Type)(*m*: *distr A*) *f*,
(∀ *x*, *f x* == 0) → *mu m f* == 0.

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

Lemma mu_fplusok : ∀ (A : Type)(m:distr A) f g, fplusok f g →
              mu m f ≤ [1-] mu m g.
Hint Resolve mu_fplusok.

Lemma mu_le_minus : ∀ (A : Type)(m:distr A) (f g:MF A),
      mu m (fminus f g) ≤ mu m f.
Hint Resolve mu_le_minus.

Lemma mu_le_plus : ∀ (A : Type)(m:distr A) (f g:MF A),
      mu m (fplus f g) ≤ mu m f + mu m g.
Hint Resolve mu_le_plus.

Lemma mu_cte : ∀ (A : Type)(m:(distr A)) (c:U),
    mu m (fcte A c) == c × mu m (fone A).
Hint Resolve mu_cte.

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

Lemma mu_cte_eq : ∀ (A : Type)(m:distr A) (c:U),
    mu m (fone A) == 1 → mu m (fcte A c) == c.

Hint Resolve mu_cte_le mu_cte_eq.

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

Lemma mu_stable_minus : ∀ (A:Type) (m:distr A)(f g : MF A),
 g ≤ f → mu m (fun x ⇒ f x - g x) == mu m f - mu m g.

Lemma mu_inv_minus :
∀ (A:Type) (m:distr A)(f: MF A), mu m (finv f) == mu m (fone A) - mu m f.

Lemma mu_stable_le_minus : ∀ (A:Type) (m:distr A)(f g : MF A),
 mu m f - mu m g ≤ mu m (fun x ⇒ f x - g x).

Lemma mu_inv_minus_inv : ∀ (A:Type) (m:distr A)(f: MF A),
      mu m (finv f) + [1-](mu m (fone A)) == [1-](mu m f).

Lemma mu_le_esp_inv : ∀ (A:Type) (m:distr A)(f g : MF A),
 ([1-]mu m (finv f)) & mu m g ≤ mu m (fesp f g).
Hint Resolve mu_le_esp_inv.

Lemma mu_stable_inv_inv : ∀ (A:Type) (m:distr A)(f : MF A),
              mu m f ≤ [1-] mu m (finv f).
Hint Resolve mu_stable_inv_inv.

Lemma mu_stable_div : ∀ (A:Type) (m:distr A)(k:U)(f : MF A),
           ¬0==k → f ≤ fcte A k → mu m (fdiv k f) == mu m f / k.

Lemma mu_stable_div_le : ∀ (A:Type) (m:distr A)(k:U)(f : MF A),
           ¬0==k → mu m (fdiv k f) ≤ mu m f / k.

Lemma mu_le_esp : ∀ (A:Type) (m:distr A)(f g : MF A),
 mu m f & mu m g ≤ mu m (fesp f g).
Hint Resolve mu_le_esp.

Lemma mu_esp_one : ∀ (A:Type)(m:distr A)(f g:MF A),
   1<=mu m f → mu m g == mu m (fesp f g).

Lemma mu_esp_zero : ∀ (A:Type)(m:distr A)(f g:MF A),
    mu m (finv f) ≤ 0 → mu m g == mu m (fesp f g).

Instance Odistr (A:Type) : ord (distr A) :=
    {Ole := fun (f g : distr A) ⇒ mu f ≤ mu g;
```

Oeq := fun $(f\ g : distr\ A) \Rightarrow mu\ f == mu\ g\}$.
Defined.

7.3 Monadic operators for distributions

Definition $Munit : \forall\ A$:Type, $A \to distr\ A$.
Defined.

Definition $Mlet : \forall\ A\ B$:Type, $distr\ A \to (A \to distr\ B) \to distr\ B$.
Defined.

Lemma $Munit\_simpl : \forall\ (A$:Type$)\ (q$:A $\to U)\ x,\ mu\ (Munit\ x)\ q = q\ x$.

Lemma $Mlet\_simpl : \forall\ (A\ B$:Type$)\ (m$:distr $A)\ (M$:A $\to distr\ B)\ (f$:B $\to U)$,
$mu\ (Mlet\ m\ M)\ f = mu\ m$ (fun $x \Rightarrow (mu\ (M\ x)\ f))$.

7.4 Operations on distributions

Lemma $Munit\_eq\_compat : \forall\ A\ (x\ y : A), x = y \to Munit\ x == Munit\ y$.

Lemma $Mlet\_le\_compat : \forall\ (A\ B$: Type$)\ (m1\ m2$:distr $A)\ (M1\ M2 : A \to distr\ B)$,
$m1 \leq m2 \to M1 \leq M2 \to Mlet\ m1\ M1 \leq Mlet\ m2\ M2$.
Hint Resolve $Mlet\_le\_compat$.

Add $Parametric\ Morphism\ (A\ B$: Type$) : (Mlet\ (A$:=A$)\ (B$:=B$))$
with $signature\ Ole \Longrightarrow Ole \Longrightarrow Ole$
as $Mlet\_le\_morphism$.
Save.

Instance $Mlet\_mon2 : \forall\ (A\ B$: Type$), monotonic2\ (@Mlet\ A\ B)$.
Save.

Definition $MLet\ (A\ B$: Type$) : distr\ A \xrightarrow{m} (A \to distr\ B) \xrightarrow{m} distr\ B$
$:= mon2\ (@Mlet\ A\ B)$.

Lemma $MLet\_simpl : \forall\ (A\ B$:Type$)\ (m$:distr $A)\ (M$:A $\to distr\ B)(f$:B $\to U)$,
$mu\ (MLet\ A\ B\ m\ M)\ f = mu\ m$ (fun $x \Rightarrow mu\ (M\ x)\ f)$.

Lemma $Mlet\_eq\_compat : \forall\ (A\ B$: Type$)\ (m1\ m2$:distr $A)\ (M1\ M2 : A \to distr\ B)$,
$m1 == m2 \to M1 ==$M2 $\to Mlet\ m1\ M1 == Mlet\ m2\ M2$.
Hint Resolve $Mlet\_eq\_compat$.

Add $Parametric\ Morphism\ (A\ B$: Type$) : (Mlet\ (A$:=A$)\ (B$:=B$))$
with $signature\ Oeq \Longrightarrow Oeq \Longrightarrow Oeq$
as $Mlet\_eq\_morphism$.
Save.

Lemma $mu\_le\_compat : \forall\ (A$:Type$)\ (m1\ m2$:distr $A)$,
$m1 \leq m2 \to \forall\ f\ g : A \to U, f \leq g \to mu\ m1\ f \leq mu\ m2\ g$.

Lemma $mu\_eq\_compat : \forall\ (A$:Type$)\ (m1\ m2$:distr $A)$,
$m1 == m2 \to \forall\ f\ g : A \to U, f == g \to mu\ m1\ f == mu\ m2\ g$.
Hint Immediate $mu\_le\_compat\ mu\_eq\_compat$.

Add $Parametric\ Morphism\ (A$: Type$) : (mu\ (A$:=A$))$
with $signature\ Ole \Longrightarrow Ole$
as $mu\_le\_morphism$.
Save.

Add $Parametric\ Morphism\ (A$: Type$) : (mu\ (A$:=A$))$
with $signature\ Oeq \Longrightarrow Oeq$
as $mu\_eq\_morphism$.
Save.

7.5 Properties of monadic operators

Lemma *Mlet_unit* : ∀ (*A B*:Type) (*x*:A) (*m*:A → *distr B*), *Mlet* (*Munit x*) *m* == *m x*.

Lemma *Mlet_ext* : ∀ (*A*:Type) (*m*:distr *A*), *Mlet m* (fun *x* ⇒ *Munit x*) == *m*.

Lemma *Mlet_assoc* : ∀ (*A B C*:Type) (*m1*: *distr A*) (*m2*:A → *distr B*) (*m3*:B → *distr C*),
Mlet (*Mlet m1 m2*) *m3* == *Mlet m1* (fun *x*:A ⇒ *Mlet* (*m2 x*) *m3*).

Lemma *let_indep* : ∀ (*A B*:Type) (*m1*:distr *A*) (*m2*: *distr B*) (*f*:MF *B*),
mu m1 (fun _ ⇒ *mu m2 f*) == *mu m1* (*fone A*) × (*mu m2 f*).

7.6 A specific distribution

Definition *distr_null* : ∀ *A* : Type, *distr A*.
Defined.

Lemma *le_distr_null* : ∀ (*A*:Type) (*m* : *distr A*), *distr_null A* ≤ *m*.
Hint Resolve *le_distr_null*.

7.7 Scaling a distribution

Definition *Mmult A* (*k*:MF *A*) (*m*:M *A*) : *M A*.
Defined.

Lemma *Mmult_simpl* : ∀ *A* (*k*:MF *A*) (*m*:M *A*) *f*, *Mmult k m f* = *m* (fun *x* ⇒ *k x* × *f x*).

Lemma *Mmult_stable_inv* : ∀ *A* (*k*:MF *A*) (*d*:distr *A*), *stable_inv* (*Mmult k* (*mu d*)).

Lemma *Mmult_stable_plus* : ∀ *A* (*k*:MF *A*) (*d*:distr *A*), *stable_plus* (*Mmult k* (*mu d*)).

Lemma *Mmult_stable_mult* : ∀ *A* (*k*:MF *A*) (*d*:distr *A*), *stable_mult* (*Mmult k* (*mu d*)).

Lemma *Mmult_continuous* : ∀ *A* (*k*:MF *A*) (*d*:distr *A*), *continuous* (*Mmult k* (*mu d*)).

Definition *distr_mult A* (*k*:MF *A*) (*d*:distr *A*) : *distr A*.
Defined.

Definition *Mdiv A* (*k*:U) (*m*:M *A*) : *M A* := *UDiv k* @ *m*.

Lemma *Mdiv_simpl* : ∀ *A k* (*m*:M *A*) *f*, *Mdiv k m f* = *m f* / *k*.

Lemma *Mdiv_stable_inv* : ∀ *A* (*k*:U) (*d*:distr *A*)(*dk* : *mu d* (*fone A*) ≤ *k*),
stable_inv (*Mdiv k* (*mu d*)).

Lemma *Mdiv_stable_plus* : ∀ *A* (*k*:U)(d:distr *A*), *stable_plus* (*Mdiv k* (*mu d*)).

Lemma *Mdiv_stable_mult* : ∀ *A* (*k*:U)(d:distr *A*)(*dk* : *mu d* (*fone A*) ≤ *k*),
stable_mult (*Mdiv k* (*mu d*)).

Lemma *Mdiv_continuous* : ∀ *A* (*k*:U)(d:distr *A*), *continuous* (*Mdiv k* (*mu d*)).

Definition *distr_div A* (*k*:U) (*d*:distr *A*) (*dk* : *mu d* (*fone A*) ≤ *k*)
: *distr A*.
Defined.

7.8 Conditional probabilities

Definition *mcond A* (*m*:M *A*) (*f*:MF *A*) : *M A*.
Defined.

Lemma *mcond_simpl* : ∀ *A* (*m*:M *A*) (*f g*: *MF A*),
mcond m f g = *m* (*fconj f g*) / *m f*.

Lemma *mcond_stable_plus* : ∀ *A* (*m*:distr *A*) (*f*: *MF A*), *stable_plus* (*mcond* (*mu m*) *f*).

Lemma *mcond_stable_inv* : ∀ *A* (*m*:distr *A*) (*f*: *MF A*), *stable_inv* (*mcond* (*mu m*) *f*).

Lemma *mcond_stable_mult* : ∀ *A* (*m*:distr *A*) (*f*: *MF A*), *stable_mult* (*mcond* (*mu m*) *f*).

```
Lemma mcond_continuous : ∀ A (m:distr A) (f: MF A), continuous (mcond (mu m) f).

Definition Mcond A (m:distr A) (f:MF A) : distr A :=
  Build_distr (mcond_stable_inv m f) (mcond_stable_plus m f)
              (mcond_stable_mult m f) (mcond_continuous m f).

Lemma Mcond_total : ∀ A (m:distr A) (f:MF A),
          ¬ 0 == mu m f → mu (Mcond m f) (fone A) == 1.

Lemma Mcond_simpl : ∀ A (m:distr A) (f g:MF A),
      mu (Mcond m f) g = mu m (fconj f g) / mu m f.
Hint Resolve Mcond_simpl.

Lemma Mcond_zero_stable : ∀ A (m:distr A) (f g:MF A),
      mu m g == 0 → mu (Mcond m f) g == 0.

Lemma Mcond_null : ∀ A (m:distr A) (f g:MF A),
      mu m f == 0 → mu (Mcond m f) g == 0.

Lemma Mcond_conj : ∀ A (m:distr A) (f g:MF A),
          mu m (fconj f g) == mu (Mcond m f) g × mu m f.

Lemma Mcond_decomp :
    ∀ A (m:distr A) (f g:MF A),
          mu m g == mu (Mcond m f) g × mu m f + mu (Mcond m (finv f)) g × mu m (finv f).

Lemma Mcond_bayes : ∀ A (m:distr A) (f g:MF A),
          mu (Mcond m f) g == (mu (Mcond m g) f × mu m g) / (mu m f).
```

7.9 Least upper bound of increasing sequences of distributions

```
Lemma M_lub_simpl : ∀ A (h: nat →m M A) (f:MF A),
     lub h f = lub (mshift h f).

Section Lubs.
Variable A : Type.

Definition Mu : distr A →m M A.
Defined.

Variable muf : nat →m distr A.

Definition mu_lub: distr A.


Defined.

Lemma mu_lub_le : ∀ n:nat, muf n ≤ mu_lub.

Lemma mu_lub_sup : ∀ m: distr A, (∀ n:nat, muf n ≤ m) → mu_lub ≤ m.

End Lubs.
Hint Resolve mu_lub_le mu_lub_sup.
```

7.9.1 distributions seen as a Ccpo

```
Instance cdistr : ∀ (A:Type), cpo (distr A) :=
  {D0 := distr_null A; lub:=mu_lub (A:=A)}.
Defined.

Lemma distr_lub_simpl : ∀ A (h : nat →m distr A) (f:MF A),
      mu (lub h) f = lub (mshift (Mu A @ h) f).
Hint Resolve distr_lub_simpl.
```

7.10 Fixpoints

`Definition` *Mfix* (*A B*:Type) (*F*: $(A \to distr\ B) \xrightarrow{m} (A \to distr\ B)$)
: $A \to distr\ B :=$ *fixp F*.

`Definition` *MFix* (*A B*:Type) : $((A \to distr\ B) \xrightarrow{m} (A \to distr\ B)) \xrightarrow{m} (A \to distr\ B)$
$:=$ *Fixp* $(A \to distr\ B)$.

`Lemma` *Mfix_le* : $\forall$ (*A B*:Type) (*F*: $(A \to distr\ B) \xrightarrow{m} (A \to distr\ B)$) (*x*:A),
Mfix F x $\leq$ *F* (*Mfix F*) *x*.

`Lemma` *Mfix_eq* : $\forall$ (*A B*:Type) (*F*: $(A \to distr\ B) \xrightarrow{m} (A \to distr\ B)$),
continuous *F* $\to \forall$ (*x*:A), *Mfix F x* $==$ *F* (*Mfix F*) *x*.

`Hint Resolve` *Mfix_le Mfix_eq*.

`Lemma` *Mfix_le_compat* : $\forall$ (*A B*:Type) (*F G* : $(A \to distr\ B) \xrightarrow{m} (A \to distr\ B)$),
F $\leq$ *G* $\to$ *Mfix F* $\leq$ *Mfix G*.

`Definition` *Miter* (*A B*:Type) := *Ccpo.iter* (*D*:=A $\to$ *distr B*).

`Lemma` *Mfix_le_iter* : $\forall$ (*A B*:Type) (*F*:(A $\to$ *distr B*) $\xrightarrow{m}$ (*A* $\to$ *distr B*)) (*n*:nat),
Miter F n $\leq$ *Mfix F*.

7.11 Continuity

`Section` *Continuity*.

`Variables` *A B*:Type.

`Instance` *Mlet_continuous_right*
: $\forall$ *a*:distr *A*, *continuous* (*D1*:= *A* $\to$ *distr B*) (*D2*:=distr *B*) (*MLet A B a*).
`Save`.

`Lemma` *Mlet_continuous_left*
: *continuous* (*D1*:=distr *A*) (*D2*:=(A $\to$ *distr B*) $\xrightarrow{m}$ *distr B*) (*MLet A B*).

`Hint Resolve` *Mlet_continuous_right Mlet_continuous_left*.

`Lemma` *Mlet_continuous2* : *continuous2* (*D1*:=distr *A*) (*D2*:= *A*->distr *B*) (*D3*:=distr *B*) (*MLet A B*).
`Hint Resolve` *Mlet_continuous2*.

`Lemma` *Mlet_lub_le* : $\forall$ (*mun*:nat $\xrightarrow{m}$ *distr A*) (*Mn* : *nat* $\xrightarrow{m}$ (*A* $\to$ *distr B*)),
Mlet (*lub mun*) (*lub Mn*) $\leq$ *lub* ((*MLet A B* @2 *mun*) *Mn*).

`Lemma` *Mfix_continuous* :
$\forall$ (*Fn* : *nat* $\xrightarrow{m}$ (*A* $\to$ *distr B*) $\xrightarrow{m}$ (*A* $\to$ *distr B*)),
($\forall$ *n*, *continuous* (*Fn n*)) $\to$
Mfix (*lub Fn*) $\leq$ *lub* (*MFix A B* @ *Fn*).

`End` *Continuity*.

7.12 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* := *mon* (`fun` (*f* : *bool* $\to$ *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_continuous* : *continuous flip*.

`Definition` *ctrue* : *MF bool* := `fun` (*b*:bool) $\Rightarrow$ `if` *b* `then` 1 `else` 0.
`Definition` *cfalse* : *MF bool* := `fun` (*b*:bool) $\Rightarrow$ `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*.
Defined.

Lemma *Flip_simpl* : ∀ *f*, *mu Flip f* = [1/2] × (*f true*) + [1/2] × (*f false*).

7.13 Uniform distribution beween 0 and n

Require *Arith*.

7.13.1 Definition of *fnth*

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

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

7.13.2 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* [].

7.14 distributions and general summations

Definition *sigma_fun A* (*f*:nat → *MF A*) (*n*:nat) : *MF A* := fun *x* ⇒ *sigma* (fun *k* ⇒ *f k x*) *n*.
Definition *serie_fun A* (*f*:nat → *MF A*) : *MF A* := fun *x* ⇒ *serie* (fun *k* ⇒ *f k x*).

Definition *Sigma_fun A* (*f*:nat → *MF A*) : *nat* $\stackrel{m}{\to}$ *MF A* :=
ishift (fun *x* ⇒ *Sigma* (fun *k* ⇒ *f k x*)).

Lemma *Sigma_fun_simpl* : ∀ *A* (*f*:nat → *MF A*) (*n*:nat),
Sigma_fun f n = *sigma_fun f n*.

Lemma *serie_fun_lub_sigma_fun* : ∀ *A* (*f*:nat → *MF A*),
serie_fun f == *lub* (*Sigma_fun f*).
Hint Resolve *serie_fun_lub_sigma_fun*.

Lemma *sigma_fun_0* : ∀ *A* (*f*:nat → *MF A*), *sigma_fun f* 0 == *fzero A*.

Lemma *sigma_fun_S* : ∀ *A* (*f*:nat->MF *A*) (*n*:nat),
sigma_fun f (*S n*) == *fplus* (*f n*) (*sigma_fun f n*).

Lemma *mu_sigma_le* : ∀ *A* (*d*:distr *A*) (*f*:nat → *MF A*) (*n*:nat),
mu d (*sigma_fun f n*) ≤ *sigma* (fun *k* ⇒ *mu d* (*f k*)) *n*.

Lemma *retract_fplusok* : ∀ *A* (*f*:nat → *MF A*) (*n*:nat),
(∀ *x*, *retract* (fun *k* ⇒ *f k x*) *n*) →
∀ *k*, (*k* < *n*)%nat → *fplusok* (*f k*) (*sigma_fun f k*).

```
Lemma mu_sigma_eq : ∀ A (d:distr A) (f:nat → MF A) (n:nat),
      (∀ x, retract (fun k ⇒ f k x) n) →
       mu d (sigma_fun f n) == sigma (fun k ⇒ mu d (f k)) n.

Lemma mu_serie_le : ∀ A (d:distr A) (f:nat → MF A),
       mu d (serie_fun f) ≤ serie (fun k ⇒ mu d (f k)).

Lemma mu_serie_eq : ∀ A (d:distr A) (f:nat → MF A),
      (∀ x, wretract (fun k ⇒ f k x)) →
       mu d (serie_fun f) == serie (fun k ⇒ mu d (f k)).

Lemma wretract_fplusok : ∀ A (f:nat → MF A),
      (∀ x, wretract (fun k ⇒ f k x)) →
      ∀ k, fplusok (f k) (sigma_fun f k).
```

7.15 Discrete distributions

```
Instance discrete_mon : ∀ A (c : nat → U) (p : nat → A),
     monotonic (fun f : A->U ⇒ serie (fun k ⇒ c k × f (p k))).
Save.

Definition discrete A (c : nat → U) (p : nat → A) : M A :=
       mon (fun f : A->U ⇒ serie (fun k ⇒ c k × f (p k))).

Lemma discrete_simpl : ∀ A (c : nat → U) (p : nat → A) f,
     discrete c p f = serie (fun k ⇒ c k × f (p k)).

Lemma discrete_stable_inv : ∀ A (c : nat → U) (p : nat → A),
    wretract c → stable_inv (discrete c p).

Lemma discrete_stable_plus : ∀ A (c : nat → U) (p : nat → A),
    stable_plus (discrete c p).

Lemma discrete_stable_mult : ∀ A (c : nat → U) (p : nat → A),
    wretract c → stable_mult (discrete c p).

Lemma discrete_continuous : ∀ A (c : nat → U) (p : nat → A),
    continuous (discrete c p).

Record discr (A:Type) : Type :=
     {coeff : nat → U; coeff_retr : wretract coeff; points : nat → A}.
Hint Resolve coeff_retr.

Definition Discrete : ∀ A, discr A → distr A.
Defined.

Lemma Discrete_simpl : ∀ A (d:discr A),
     mu (Discrete d) = discrete (coeff d) (points d).

Definition is_discrete (A:Type) (m: distr A) :=
      ∃ d : discr A, m == Discrete d.
```

7.15.1 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

```
Instance random_mon : ∀ n, monotonic (fun (f:MF nat) ⇒ sigma (fun k ⇒ Unth n × f k) (S n)).
Save.

Definition random (n:nat):M nat := mon (fun (f:MF nat) ⇒ sigma (fun k ⇒ Unth n × f k) (S n)).

Lemma random_simpl : ∀ n (f : MF nat),
           random n f = sigma (fun k ⇒ Unth n × f k) (S n).
```

7.15.2 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_continuous* : ∀ *n*, *continuous* (*random n*).

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

Lemma *Random_simpl* : ∀ (*n*:nat), *mu* (*Random n*) = *random n*.

Lemma *Random_total* : ∀ *n* : *nat*, *mu* (*Random n*) (*fone nat*) == 1.
Hint Resolve *Random_total*.

Lemma *Random_inv* : ∀ *f n*, *mu* (*Random n*) (*finv f*) == [1-] (*mu* (*Random n*) *f*).
Hint Resolve *Random_inv*.

7.16 Tacticals

```
Ltac mu_plus d :=
  match goal with
 | ⊢ context [fmont (mu d) (fun x ⇒ (Uplus (@?f x) (@?g x)))] ⇒
      rewrite (mu_stable_plus d (f:=f) (g:=g))
  end.
```

```
Ltac mu_mult d :=
  match goal with
 | ⊢ context [fmont (mu d) (fun x ⇒ (Umult ?k (@?f x)))] ⇒
      rewrite (mu_stable_mult d k f)
  end.
```

8 Prog.v: Composition of distributions

Require Export Probas.

8.1 Conditional

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

Lemma *Mif_le_compat* : ∀ (*A*:Type) (**b1** *b2*:distr *bool*) (*m1 m2 n1 n2*: *distr A*),
    b1 ≤ *b2* → *m1* ≤ *m2* → *n1* ≤ *n2* → *Mif b1 m1 n1* ≤ *Mif b2 m2 n2*.

Hint Resolve *Mif_le_compat*.

Instance *Mif_mon2* : ∀ (*A*:Type) *b*, *monotonic2* (*Mif* (*A*:=A) *b*).
Save.

Definition *MIf* : ∀ (*A*:Type), *distr bool* $\xrightarrow{m}$ *distr A* $\xrightarrow{m}$ *distr A* $\xrightarrow{m}$ *distr A*.
Defined.

8.2 Probabilistic choice

8.2.1 The distribution associated to *pchoice p m1 m2* is

$f \mapsto p(m_1\, f) + (1-p)(m_2\, f)$

Definition pchoice : ∀ *A*, *U* → *M A* → *M A* → *M A*.

```
Defined.

Lemma pchoice_simpl : ∀ A p (m1 m2:M A) f,
      pchoice p m1 m2 f = p × m1 f + [1-]p × m2 f.

Definition Mchoice (A:Type) (p:U) (m1 m2: distr A) : distr A.
Defined.

Lemma Mchoice_simpl : ∀ A p (m1 m2:distr A) f,
      mu (Mchoice p m1 m2) f = p × mu m1 f + [1-]p × mu m2 f.

Lemma Mchoice_le_compat : ∀ (A:Type) (p:U) (m1 m2 n1 n2: distr A),
        m1<=m2 → n1<=n2 → Mchoice p m1 n1 ≤ Mchoice p m2 n2.
Hint Resolve Mchoice_le_compat.

Add Parametric Morphism (A:Type) : (Mchoice (A:=A))
  with signature Oeq ⟹Oeq ⟹Oeq ⟹Oeq
as Mchoice_eq_compat.
Save.
Hint Immediate Mchoice_eq_compat.

Instance Mchoice_mon2 : ∀ (A:Type) (p:U), monotonic2 (Mchoice (A:=A) p).
Save.

Definition MChoice A (p:U) : distr A →m distr A →m distr A :=
                mon2 (Mchoice (A:=A) p).

Lemma MChoice_simpl : ∀ A (p:U) (m1 m2 : distr A),
MChoice A p m1 m2 = Mchoice p m1 m2.

Lemma Mchoice_sym_le : ∀ (A:Type) (p:U) (m1 m2: distr A),
            Mchoice p m1 m2 ≤ Mchoice ([1-]p) m2 m1.
Hint Resolve Mchoice_sym_le.

Lemma Mchoice_sym : ∀ (A:Type) (p:U) (m1 m2: distr A),
            Mchoice p m1 m2 == Mchoice ([1-]p) m2 m1.

Lemma Mchoice_continuous_right
    : ∀ (A:Type) (p:U) (m: distr A), continuous (D1:=distr A) (D2:=distr A) (MChoice A p m).
Hint Resolve Mchoice_continuous_right.

Lemma Mchoice_continuous_left : ∀ (A:Type) (p:U),
    continuous (D1:=distr A) (D2:=distr A →m distr A) (MChoice A p).

Lemma Mchoice_continuous :
∀ (A:Type) (p:U), continuous2 (D1:=distr A) (D2:=distr A) (D3:=distr A) (MChoice A p).
```

8.3 Image distribution

```
Definition im_distr (A B : Type) (f:A → B) (m:distr A) : distr B :=
    Mlet m (fun a ⇒ Munit (f a)).

Lemma im_distr_simpl : ∀ A B (f:A → B) (m:distr A)(h:B → U),
     mu (im_distr f m) h = mu m (fun a ⇒ h (f a)).

Add Parametric Morphism (A B : Type) : (im_distr (A:=A) (B:=B))
  with signature (feq (A:=A) (B:=B)) ⟹Oeq ⟹Oeq
  as im_distr_eq_compat.
Save.

Lemma im_distr_comp : ∀ A B C (f:A → B) (g:B → C) (m:distr A),
            im_distr g (im_distr f m) == im_distr (fun a ⇒ g (f a)) m.

Lemma im_distr_id : ∀ A (f:A → A) (m:distr A), (∀ x, f x = x) →
            im_distr f m == m.
```

8.4 Product distribution

```
Definition prod_distr (A B : Type)(d1:distr A)(d2:distr B) : distr (A*B) :=
                Mlet d1 (fun x ⇒ Mlet d2 (fun y ⇒ Munit (x,y))).

Add Parametric Morphism (A B : Type) : (prod_distr (A:=A) (B:=B))
with signature Ole ++> Ole ++> Ole
as prod_distr_le_compat.
Save.
Hint Resolve prod_distr_le_compat.

Add Parametric Morphism (A B : Type) : (prod_distr (A:=A) (B:=B))
with signature Oeq ==>Oeq ==>Oeq
as prod_distr_eq_compat.
Save.
Hint Immediate prod_distr_eq_compat.

Instance prod_distr_mon2 : ∀ (A B :Type), monotonic2 (prod_distr (A:=A) (B:=B)).
Save.

Definition Prod_distr (A B :Type): distr A -m-> distr B -m-> distr (A*B) :=
     mon2 (prod_distr (A:=A) (B:=B)).

Lemma Prod_distr_simpl : ∀ (A B:Type)(d1: distr A) (d2:distr B),
     Prod_distr A B d1 d2 = prod_distr d1 d2.

Lemma prod_distr_rect : ∀ (A B : Type) (d1: distr A) (d2:distr B) (f:A → U)(g:B → U),
          mu (prod_distr d1 d2) (fun xy ⇒ f (fst xy) × g (snd xy)) == mu d1 f × mu d2 g.

Lemma prod_distr_fst : ∀ (A B : Type) (d1: distr A) (d2:distr B) (f:A → U),
               mu (prod_distr d1 d2) (fun xy ⇒ f (fst xy)) == mu d2 (fone B) × mu d1 f.

Lemma prod_distr_snd : ∀ (A B : Type) (d1: distr A) (d2:distr B) (g:B → U),
               mu (prod_distr d1 d2) (fun xy ⇒ g (snd xy)) == mu d1 (fone A) × mu d2 g.

Lemma prod_distr_fst_eq : ∀ (A B : Type) (d1: distr A) (d2:distr B),
mu d2 (fone B) == 1 → im_distr (fst (A:=A) (B:=B)) (prod_distr d1 d2) == d1.

Lemma prod_distr_snd_eq : ∀ (A B : Type) (d1: distr A) (d2:distr B),
mu d1 (fone A) == 1 → im_distr (snd (A:=A) (B:=B)) (prod_distr d1 d2) == d2.

Definition swap A B (x:A*B) : B × A := (snd x,fst x).

Definition arg_swap A B (f : MF (A*B)) : MF (B*A) := fun z ⇒ f (swap z).

Definition Arg_swap A B : MF (A*B) -m-> MF (B*A).
Defined.

Lemma Arg_swap_simpl : ∀ A B f, Arg_swap A B f = arg_swap f.

Definition prod_distr_com A B (d1: distr A) (d2:distr B) (f : MF (A × B)) :=
      mu (prod_distr d1 d2) f == mu (prod_distr d2 d1) (arg_swap f).

Lemma prod_distr_com_eq_compat : ∀ A B (d1: distr A) (d2:distr B) (f g: MF (A × B)),
  f == g → prod_distr_com d1 d2 f → prod_distr_com d1 d2 g.

Lemma prod_distr_com_rect : ∀ (A B : Type) (d1: distr A) (d2:distr B) (f:A → U)(g:B → U),
          prod_distr_com d1 d2 (fun xy ⇒ f (fst xy) × g (snd xy)).

Lemma prod_distr_com_cte : ∀ (A B : Type) (d1: distr A) (d2:distr B) (c:U),
          prod_distr_com d1 d2 (fcte (A*B) c).

Lemma prod_distr_com_one : ∀ (A B : Type) (d1: distr A) (d2:distr B),
          prod_distr_com d1 d2 (fone (A*B)).

Lemma prod_distr_com_plus : ∀ (A B : Type) (d1: distr A) (d2:distr B) (f g:MF (A*B)),
          fplusok f g →
          prod_distr_com d1 d2 f → prod_distr_com d1 d2 g →
```

prod_distr_com d1 d2 (fplus f g).

Lemma *prod_distr_com_mult* : ∀ (*A B* : Type) (*d1*: *distr A*) (*d2*:distr *B*) (*k*:U)(f:MF (*A**B)),
prod_distr_com *d1 d2 f* → *prod_distr_com d1 d2* (*fmult k f*).

Lemma *prod_distr_com_inv* : ∀ (*A B* : Type) (*d1*: *distr A*) (*d2*:distr B) (*f*:MF (*A**B)),
prod_distr_com *d1 d2 f* → *prod_distr_com d1 d2* (*finv f*).

Lemma *prod_distr_com_lub* : ∀ (*A B* : Type) (*d1*: *distr A*) (*d2*:distr *B*) (*f*:nat $\stackrel{m}{\to}$ *MF* (*A**B)),
(∀ *n*, *prod_distr_com d1 d2* (*f n*)) → *prod_distr_com d1 d2* (*lub f*).

Lemma *prod_distr_com_refl* : ∀ *A B* (*d1*:distr *A*) (*d2*:distr *B*) (*f*:MF (*A**B)),
prod_distr_com *d1 d2 f* → *prod_distr_com d2 d1* (*arg_swap f*).

Lemma *discrete_commute* : ∀ *A B* (*d1*:distr *A*) (*d2*:distr *B*) (*f*:MF (*A**B)),
is_discrete d1 → *prod_distr_com d1 d2 f.*

Definition *fst_distr A B* (*m* : distr (*A**B)) : *distr A* := *im_distr* (fst (*B*:=B)) *m.*
Definition *snd_distr A B* (*m* : distr (*A**B)) : *distr B* := *im_distr* (*snd* (*B*:=B)) *m.*

8.5 Range of a distribution

Definition **range** *A* (*P*:A → Prop) (*d*: *distr A*) :=
∀ *f*, (∀ *x*, *P x* → 0 == *f* x) → 0 == *mu d f.*

Lemma *range_le* : ∀ *A* (*P*:A → Prop) (*d*:distr *A*), *range P d* →
∀ *f g*, (∀ *a*, *P a* → *f a* ≤ *g a*) → *mu d f* ≤ *mu d g.*

Lemma *range_eq* : ∀ *A* (*P*:A → Prop) (*d*:distr *A*), *range P d* →
∀ *f g*, (∀ *a*, *P a* → *f a* == *g a*) → *mu d f* == *mu d g.*

9 Prog.v: Axiomatic semantics

9.1 Definition of correctness judgements

$p \le [e](q)$ is defined as $p \le \mu(e)(q)$ ↑ *peq* is defined as $\mu(e)(q) \le p$

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

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

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

9.2 Stability properties

Lemma *ok_le_compat* : ∀ (*A*:Type) (*p p'*:U) (*e*:distr *A*) (*q q'*:A → *U*),
p' ≤ *p* → *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* → *q* == *q'* → *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'*:A → *B* → *U*),
p' ≤ *p* → q ≤ *q'* → *okfun p e q* → *okfun p' e q'.*

Lemma *ok_mult* : ∀ (*A*:Type)(k *p*:U)(e:distr *A*)(*f* : *A* → *U*),
ok p e f → *ok* (*k**p) *e* (*fmult k f*).

Lemma *ok_inv* : ∀ (*A*:Type)(p:U)(e:distr *A*)(f : *A* → *U*),
ok p *e f* → *mu e* (*finv f*) ≤ [1-]*p.*

Lemma *okup_le_compat* : ∀ (*A*:Type) (*p p'*:U) (*e*:distr *A*) (*q q'*:A → *U*),

$p \leq p' \to q' \leq q \to okup\ p\ e\ q \to okup\ p'\ e\ q'$.

Lemma *okup_eq_compat* : $\forall$ (A:Type) ($p\ p'$:U) ($e\ e'$:distr A) ($q\ q'$:A $\to U$),
$p == p' \to q == q' \to e == e' \to okup\ p\ e\ q \to okup\ p'\ e'\ q'$.

Lemma *upfun_le_compat* : $\forall$ ($A\ B$:Type) ($p\ p'$:A $\to U$) (e:A $\to$ *distr* B)
($q\ q'$:A $\to B \to U$),
$p \leq p' \to q'$<=q $\to$ *upfun* $p\ e\ q \to$ *upfun* $p'\ e\ q'$.

Lemma *okup_mult* : $\forall$ (A:Type)(k p:U)(e:distr A)(f : $A \to U$), *okup* $p\ e\ f \to okup\ (k$*p) e (*fmult* $k\ f$).

9.3 Basic rules

9.3.1 Rules for application

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

Lemma *apply_rule* : $\forall$ ($A\ B$:Type)(a:(distr A))(f:A $\to$ *distr* B)(r:U)(p:A $\to U$)(q:B $\to U$),
ok $r\ a\ p \to$ okfun p f (fun $x \Rightarrow q$) $\to$ *ok* r (*Mlet* $a\ f$) q.

Lemma *okup_apply_rule* : $\forall$ (A **B**:Type)(a:distr A)(f:A $\to$ *distr* B)(r:U)(p:A $\to U$)(q:B $\to U$),
okup $r\ a\ p \to$ upfun p f (fun $x \Rightarrow q$) $\to$ *okup* r (*Mlet* $a\ f$) q.

9.3.2 Rules for abstraction

Lemma *lambda_rule* : $\forall$ ($A\ B$:Type)(f:A $\to$ *distr* B)(p:A $\to U$)(q:A $\to B \to U$),
($\forall\ x$:A, *ok* $(p\ x)\ (f\ x)\ (q\ x)$) $\to$ *okfun* $p\ f\ q$.

Lemma *okup_lambda_rule* : $\forall$ ($A\ B$:Type)(f:A $\to$ *distr* B)(p:A $\to U$)(q:A $\to B \to U$),
($\forall\ x$:A, *okup* $(p\ x)\ (f\ x)\ (q\ x)$) $\to$ *upfun* $p\ f\ q$.

9.3.3 Rule for conditional

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

Lemma *combiok* : $\forall$ (*A*:Type) $p\ q$ ($f1\ f2$: $A \to U$), $p \leq$ [1-]$q \to$ *fplusok* (*fmult* $p\ f1$) (*fmult* $q\ f2$).
Hint Resolve *combiok*.

Lemma *fmult_fplusok* : $\forall$ (*A*:Type) $p\ q$ ($f1\ f2$: A $\to U$), fplusok $f1\ f2 \to$ *fplusok* (*fmult* $p\ f1$) (*fmult* $q\ f2$).
Hint Resolve *fmult_fplusok*.

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

Lemma *Mif_eq* : $\forall$ (A:Type)(b:(distr *bool*))($f1$ f2:distr A)(q:MF A),
mu (*Mif* $b\ f1\ f2$) q == (*mu* $f1\ q$) $\times$ (*mu* b *ctrue*) + (*mu* $f2\ q$) $\times$ (*mu* b *cfalse*).

Lemma *ifrule* :
$\forall$ (A:Type)(b:(distr *bool*))($f1\ f2$:distr A)($p1\ p2$:U)(q:A $\to U$),
ok $p1\ f1$ q $\to$ *ok* $p2\ f2\ q$
$\to$ *ok* (p1 $\times$ (*mu* b *ctrue*) + $p2 \times$ (*mu* b *cfalse*)) (*Mif* $b\ f1\ f2$) q.

Lemma *okup_ifrule* :
$\forall$ (A:Type)(b:(distr *bool*))($f1\ f2$:distr A)($p1\ p2$:U)(q:A $\to U$),
okup $p1\ f1\ q \to$ *okup* $p2\ f2\ q$
$\to$ *okup* (p1 $\times$ (*mu* b *ctrue*) + $p2 \times$ (*mu* b *cfalse*)) (*Mif* $b\ f1\ f2$) q.

9.3.4 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) -m> (A → distr B).
Section Ruleseq.
Variable q : A → B → U.
Lemma fixrule_Ulub : ∀ (p : A → nat → U),
  (∀ x:A, p x O == 0)→
  (∀ (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 ⇒ Ulub (p x)) (Mfix F) q.
Lemma fixrule : ∀ (p : A → nat -m> U),
  (∀ x:A, p x O == 0)→
  (∀ (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) q.
Lemma fixrule_up_Ulub : ∀ (p : A → nat → U),
  (∀ (i:nat) (f:A → distr B),
    (upfun (fun x ⇒ p x i) f q) → upfun (fun x ⇒ p x (S i)) (fun x ⇒ F f x) q)
  → upfun (fun x ⇒ Ulub (p x)) (Mfix F) q.
Lemma fixrule_up_lub : ∀ (p : A → nat -m> U),
  (∀ (i:nat) (f:A → distr B),
    (upfun (fun x ⇒ p x i) f q) → upfun (fun x ⇒ p x (S i)) (fun x ⇒ F f x) q)
  → upfun (fun x ⇒ lub (p x)) (Mfix F) q.
Lemma okup_fixrule_glb :
  ∀ p : A → nat -m> - U,
  (∀ (i:nat) (f:A → distr B),
    (upfun (fun x ⇒ p x i) f q) → upfun (fun x ⇒ p x (S i)) (fun x ⇒ F f x) q)
  → upfun (fun x ⇒ glb (p x)) (Mfix F) q.
End Ruleseq.
Lemma okup_fixrule_inv : ∀ (p : A → U) (q : A → B → U),
  (∀ (f:A → distr B), upfun p f q → upfun p (fun x ⇒ F f x) q)
        → upfun p (Mfix F) q.
```

9.3.5 Rules using commutation properties

```
Section TransformFix.
Section Fix_muF.
Variable q : A → B → U.
Variable muF : MF A -m> MF A.
Definition admissible (P:(A → distr B) → Prop) := P 0 ∧ ∀ f, P f → P (F f).
Lemma admissible_true : admissible (fun f ⇒ True).
Lemma admissible_le_fix :
  continuous (D1:=A → distr B) (D2:=A → distr B) F → admissible (fun f ⇒ f ≤ Mfix F).
Lemma muF_stable : stable muF.
Definition mu_muF_commute_le :=
  ∀ f x, f ≤ Mfix F → mu (F f x) (q x) ≤ muF (fun y ⇒ mu (f y) (q y)) x.
```

```
Hint Unfold mu_muF_commute_le.

Section F_muF_results.
Hypothesis F_muF_le : mu_muF_commute_le.

Lemma mu_mufix_le : ∀ x, mu (Mfix F x) (q x) ≤ mufix muF x.
Hint Resolve mu_mufix_le.

Lemma muF_le : ∀ f, muF f ≤ f
      → ∀ x, mu (Mfix F x) (q x) ≤ f x.

Hypothesis muF_F_le :
    ∀ f x, f ≤ Mfix F → muF (fun y ⇒ mu (f y) (q y)) x ≤ mu (F f x) (q x).

Lemma mufix_mu_le : ∀ x, mufix muF x ≤ mu (Mfix F x) (q x).

End F_muF_results.
Hint Resolve mu_mufix_le mufix_mu_le.

Lemma mufix_mu :
   (∀ f x, f ≤ Mfix F → mu (F f x) (q x) == muF (fun y ⇒ mu (f y) (q y)) x)
   → ∀ x, mufix muF x == mu (Mfix F x) (q x).
Hint Resolve mufix_mu.
End Fix_muF.

Section Fix_Term.

Definition pterm : MF A := fun (x:A) ⇒ mu (Mfix F x) (fone B).
Variable muFone : MF A ⟶ MF A.

Hypothesis F_muF_eq_one :
  ∀ f x, f ≤ Mfix F → mu (F f x) (fone B) == muFone (fun y ⇒ mu (f y) (fone B)) x.

Hypothesis muF_cont : continuous muFone.

Lemma muF_pterm : pterm == muFone pterm.
Hint Resolve muF_pterm.
End Fix_Term.

Section Fix_muF_Term.

Variable q : A → B → U.
Definition qinv x y := [1-]q x y.

Variable muFqinv : MF A ⟶ MF A.

Hypothesis F_muF_le_inv : mu_muF_commute_le qinv muFqinv.

Lemma muF_le_term : ∀ f, muFqinv (finv f) ≤ finv f →
    ∀ x, f x & pterm x ≤ mu (Mfix F x) (q x).

Lemma muF_le_term_minus :
∀ f, f ≤ pterm → muFqinv (fminus pterm f) ≤ fminus pterm f →
    ∀ x, f x ≤ mu (Mfix F x) (q x).

Variable muFq : MF A ⟶ MF A.

Hypothesis F_muF_le : mu_muF_commute_le q muFq.

Lemma muF_eq : ∀ f, muFq f ≤ f → muFqinv (finv f) ≤ finv f →
    ∀ x, pterm x == 1 → mu (Mfix F x) (q x) == f x.

End Fix_muF_Term.
End TransformFix.

Section LoopRule.
Variable q : A → B → U.
Variable stop : A → distr bool.
Variable step : A → distr A.
Variable a : U.
```

```
Definition Loop : MF A ⟶ MF A.
Defined.

Lemma Loop_eq :
    ∀ f x, Loop f x = mu (stop x) (fun b ⇒ if b then a else mu (step x) f).

Definition loop := mufix Loop.

Lemma Mfixvar :
  (∀ (f:A → distr B),
      okfun (fun x ⇒ Loop (fun y ⇒ mu (f y) (q y)) x) (fun x ⇒ F f x) q
 → okfun loop (Mfix F) q.

Definition up_loop : MF A := nufix Loop.

Lemma Mfixvar_up :
  (∀ (f:A → distr B),
      upfun (fun x ⇒ Loop (fun y ⇒ mu (f y) (q y)) x) (fun x ⇒ F f x) q
 → upfun up_loop (Mfix F) q.

End LoopRule.

End Fixrule.
```

9.4 Rules for intervals

Distributions operates on intervals

```
Definition Imu : ∀ A:Type, distr A → (A → IU) → IU.
Defined.

Lemma low_Imu : ∀ (A:Type) (e:distr A) (F: A → IU),
                low (Imu e F) = mu e (fun x ⇒ low (F x)).

Lemma up_Imu : ∀ (A:Type) (e:distr A) (F: A → IU),
               up (Imu e F) = mu e (fun x ⇒ up (F x)).

Lemma Imu_monotonic : ∀ (A:Type) (e:distr A) (F G : A → IU),
              (∀ x, Iincl (F x) (G x)) → Iincl (Imu e F) (Imu e G).

Lemma Imu_stable_eq : ∀ (A:Type) (e:distr A) (F G : A → IU),
              (∀ x, Ieq (F x) (G x)) → Ieq (Imu e F) (Imu e G).
Hint Resolve Imu_monotonic Imu_stable_eq.

Lemma Imu_singl : ∀ (A:Type) (e:distr A) (f:A → U),
            Ieq (Imu e (fun x ⇒ singl (f x))) (singl (mu e f)).

Lemma Imu_inf : ∀ (A:Type) (e:distr A) (f:A → U),
            Ieq (Imu e (fun x ⇒ inf (f x))) (inf (mu e f)).

Lemma Imu_sup : ∀ (A:Type) (e:distr A) (f:A → U),
            Iincl (Imu e (fun x ⇒ sup (f x))) (sup (mu e f)).

Lemma Iin_mu_Imu :
   ∀ (A:Type) (e:distr A) (F:A → IU) (f:A → U),
            (∀ x, Iin (f x) (F x)) → Iin (mu e f) (Imu e F).
Hint Resolve Iin_mu_Imu.

Definition Iok (A:Type) (I:IU) (e:distr A) (F:A → IU) := Iincl (Imu e F) I.
Definition Iokfun (A B:Type)(I:A → IU) (e:A → distr B) (F:A → B → IU)
                  := ∀ x, Iok (I x) (e x) (F x).

Lemma Iin_mu_Iok :
   ∀ (A:Type) (I:IU) (e:distr A) (F:A → IU) (f:A → U),
            (∀ x, Iin (f x) (F x)) → Iok I e F → Iin (mu e f) I.
```

9.4.1 Stability

Lemma *Iok_le_compat* : ∀ (*A*:Type) (*I J*:IU) (*e*:distr *A*) (*F G*:A → *IU*),
Iincl I J → (∀ *x*, *Iincl* (*G* x) (*F x*)) → *Iok I e F* → *Iok J e G*.

Lemma *Iokfun_le_compat* : ∀ (*A B*:Type) (*I J*:A → *IU*) (*e*:A → *distr B*) (*F G*:A → *B* → *IU*),
(∀ *x*, *Iincl* (*I* x) (*J x*)) → (∀ *x y*, lincl (*G x y*) (*F x y*)) → *Iokfun I e F* → *Iokfun J e G*.

9.4.2 Rule for values

Lemma *Iunit_eq* : ∀ (*A*: Type) (*a*:A) (*F*:A → *IU*), *Ieq* (*Imu* (*Munit a*) *F*) (*F a*).

9.4.3 Rule for application

Lemma *Ilet_eq* : ∀ (*A B* : Type) (*a*:distr *A*) (*f*:A → *distr B*)(I:IU)(G:B → *IU*),
Ieq (*Imu* (*Mlet a f*) *G*) (*Imu a* (fun *x* ⇒ *Imu* (*f x*) *G*)).
Hint Resolve *Ilet_eq*.

Lemma *Iapply_rule* : ∀ (*A B* : Type) (*a*:distr *A*) (*f*:A → *distr B*)(*I*:IU)(F:A → *IU*)(*G*:B → *IU*),
Iok I a F → *Iokfun F f* (fun *x* ⇒ *G*) → *Iok I* (*Mlet a f*) *G*.

9.4.4 Rule for abstraction

Lemma *Ilambda_rule* : ∀ (*A B*:Type)(f:A → *distr B*)(*F*:A → *IU*)(*G*:A → *B* → *IU*),
(∀ *x*:A, *Iok* (*F x*) (*f x*) (*G x*)) → *Iokfun F f G*.

9.4.5 Rule for conditional

Lemma *Imu_Mif_eq* : ∀ (*A*:Type)(b:distr *bool*)(*f1* f2:distr *A*)(*F*:A → *IU*),
Ieq (*Imu* (*Mif b f1 f2*) *F*) (*Iplus* (*Imultk* (mu *b ctrue*) (lmu *f1 F*)) (*Imultk* (*mu b cfalse*) (*Imu f2 F*))).

Lemma *Iifrule* :
∀ (*A*:Type)(b:(distr *bool*))(*f1 f2*:distr *A*)(*I1 I2*:IU)(F:A → *IU*),
Iok I1 f1 F → *Iok I2 f2 F*
→ lok (*Iplus* (lmultk (mu *b* ctrue) *I1*) (*Imultk* (*mu b cfalse*) *I2*)) (*Mif b f1 f2*) *F*.

9.4.6 Rule for fixpoints

with $\phi(x) = F(\phi)(x)$, p_i an decreasing sequence of intervals functions ($p_{i+1}(x) \subseteq p_i(x)$) such that $p_0(x)$ contains 0 for all x.

$$\frac{\forall f\ i, (\forall x, p_i(x) \leq [f](q\ x)) \Rightarrow \forall x, p_{i+1}(x) \leq [F(f)(x)](q\ x)}{\forall x, \bigcap_i p_i\ x \leq [\phi(x)](q\ x)}$$ Section *IFixrule*.

Variables *A B* : Type.

Variable $F : (A \to distr\ B) \xrightarrow{m} (A \to distr\ B)$.

Section *IRuleseq*.
Variable *Q* : *A* → *B* → *IU*.

Variable $I : A \to nat \xrightarrow{m} IU$.

Lemma *Ifixrule* :
(∀ *x*:A, *Iin* 0 (l *x O*)) →
(∀ (*i*:nat) (*f*:A → *distr B*),
(*Iokfun* (fun *x* ⇒ *I x i*) *f Q*) → *Iokfun* (fun *x* ⇒ *I x* (*S i*)) (fun *x* ⇒ *F f x*) *Q*)
→ *Iokfun* (fun *x* ⇒ *Ilim* (*I x*)) (*Mfix F*) *Q*.
End *IRuleseq*.

Section *ITransformFix*.

Section *IFix_muF*.
Variable *Q* : *A* → *B* → *IU*.

```
Variable ImuF : (A → IU) -m> (A → IU).
Lemma ImuF_stable : ∀ I J, I==J → ImuF I == ImuF J.
Section IF_muF_results.
Hypothesis Iincl_F_ImuF :
    ∀ f x, f ≤ Mfix F →
                          Iincl (Imu (F f x) (Q x)) (ImuF (fun y ⇒ Imu (f y) (Q y)) x).
Lemma Iincl_fix_ifix : ∀ x, Iincl (Imu (Mfix F x) (Q x)) (fixp (D:=A → IU) ImuF x).
Hint Resolve Iincl_fix_ifix.
End IF_muF_results.
End IFix_muF.
End ITransformFix.
End IFixrule.
```

9.5 Rules for *Flip*

```
Lemma Flip_ctrue : mu Flip ctrue == [1/2].
Lemma Flip_cfalse : mu Flip cfalse == [1/2].
Lemma ok_Flip : ∀ q : bool → U, ok ([1/2] × q true + [1/2] × q false) Flip q.
Lemma okup_Flip : ∀ q : bool → U, okup ([1/2] × q true + [1/2] × q false) Flip q.
Hint Resolve ok_Flip okup_Flip Flip_ctrue Flip_cfalse.
Lemma Flip_eq : ∀ q : bool → U, mu Flip q == [1/2] × q true + [1/2] × q false.
Hint Resolve Flip_eq.
Lemma IFlip_eq : ∀ Q : bool → IU, leq (Imu Flip Q) (Iplus (Imultk [1/2] (Q true)) (Imultk [1/2] (Q false))).
Hint Resolve IFlip_eq.
```

9.6 Rules for total (well-founded) fixpoints

```
Section Wellfounded.
Variables A B : Type.
Variable R : A → A → Prop.
Hypothesis Rwf : well_founded R.
Variable F : ∀ x, (∀ y, R y x → distr B) → distr B.
Definition WfFix : A → distr B := Fix Rwf (fun _ ⇒ distr B) F.
Hypothesis Fext : ∀ x f g, (∀ y (p:R y x), f y p == g y p) → F f == F g.
Lemma Acc_iter_distr :
   ∀ x, ∀ r s : Acc R x, Acc_iter (fun _⇒ distr B) F r == Acc_iter (fun _⇒ distr B) F s.
Lemma WfFix_eq : ∀ x, WfFix x == F (fun (y:A) (p:R y x) ⇒ WfFix y).
Variable P : distr B → Prop.
Hypothesis Pext : ∀ m1 m2, m1 == m2 → P m1 → P m2.
Lemma WfFix_ind :
      (∀ x f, (∀ y (p:R y x), P (f y p)) → P (F f))
  → ∀ x, P (WfFix x).
End Wellfounded.
Ltac distrsimpl := match goal with
 | ⊢ (Ole (fmont (mu ?d1) ?f) (fmont (mu ?d2) ?g)) ⇒ apply (mu_le_compat (m1:=d1) (m2:=d2) (Ole_refl
d1) (f:=f) (g:=g)); intro
 | ⊢ (Oeq (fmont (mu ?d1) ?f) (fmont (mu ?d2) ?g)) ⇒ apply (mu_eq_compat (m1:=d1) (m2:=d2) (Oeq_refl
d1) (f:=f) (g:=g)); intro
```

```
 | ⊢ (Oeq (Munit ?x) (Munit ?y)) ⇒ apply (Munit_eq_compat x y)
 | ⊢ (Oeq (Mlet ?x1 ?f) (Mlet ?x2 ?g))
              ⇒ apply (Mlet_eq_compat (m1:=x1) (m2:=x2) (M1:=f) (M2:=g) (Oeq_refl x1)); intro
 | ⊢ (Ole (Mlet ?x1 ?f) (Mlet ?x2 ?g))
              ⇒ apply (Mlet_le_compat (m1:=x1) (m2:=x2) (M1:=f) (M2:=g) (Ole_refl x1)); intro
 | ⊢ context [(fmont (mu (Mlet ?m ?M)) ?f)] ⇒ rewrite (Mlet_simpl m M f)
 | ⊢ context [(fmont (mu (Munit ?x)) ?f)] ⇒ rewrite (Munit_simpl f x)
 | ⊢ context [(Mlet (Mlet ?m ?M) ?f)] ⇒ rewrite (Mlet_assoc m M f)
 | ⊢ context [(Mlet (Munit ?x) ?f)] ⇒ rewrite (Mlet_unit x f)
 | ⊢ context [(fmont (mu Flip) ?f)] ⇒ rewrite (Flip_simpl f)
 | ⊢ context [(fmont (mu (Discrete ?d)) ?f)] ⇒ rewrite (Discrete_simpl d);
                                                    rewrite (discrete_simpl (coeff d) (points d)
f)
 | ⊢ context [(fmont (mu (Random ?n)) ?f)] ⇒ rewrite (Random_simpl n);
                                                    rewrite (random_simpl n f)
 | ⊢ context [(fmont (mu (Mif ?b ?f ?g)) ?h)] ⇒ unfold Mif
 | ⊢ context [(fmont (mu (Mchoice ?p ?m1 ?m2)) ?f)] ⇒ rewrite (Mchoice_simpl p m1 m2 f)
 | ⊢ context [(fmont (mu (im_distr ?f ?m)) ?h)] ⇒ rewrite (im_distr_simpl f m h)
 | ⊢ context [(fmont (mu (prod_distr ?m1 ?m2)) ?h)] ⇒ unfold prod_distr
 | ⊢ context [((mon ?f (fmonotonic:=?mf)) ?x)] ⇒ rewrite (mon_simpl f (mf:=mf) x)
end.

Require Export Setoid.
Require Omega.
```

10 Sets.v: Definition of sets as predicates over a type A

```
Section sets.
Variable A : Type.
Variable decA : ∀ x y :A, {x=y}+{x<>y}.

Definition set := A->Prop.
Definition full : set := fun (x:A) ⇒ True.
Definition empty : set := fun (x:A) ⇒ False.
Definition add (a:A) (P:set) : set := fun (x:A) ⇒ x=a ∨ (P x).
Definition singl (a:A) :set := fun (x:A) ⇒ x=a.
Definition union (P Q:set) :set := fun (x:A) ⇒ (P x) ∨ (Q x).
Definition compl (P:set) :set := fun (x:A) ⇒ ~P x.
Definition inter (P Q:set) :set := fun (x:A) ⇒ (P x) ∧ (Q x).
Definition rem (a:A) (P:set) :set := fun (x:A) ⇒ x<>a ∧ (P x).
```

10.1 Equivalence

```
Definition eqset (P Q:set) := ∀ (x:A), P x ↔ Q x.

Implicit Arguments full [].
Implicit Arguments empty [].

Lemma eqset_refl : ∀ P:set, eqset P P.

Lemma eqset_sym : ∀ P Q:set, eqset P Q → eqset Q P.

Lemma eqset_trans : ∀ P Q R:set,
   eqset P Q → eqset Q R → eqset P R.

Hint Resolve eqset_refl.
Hint Immediate eqset_sym.
```

10.2 Setoid structure

```
Lemma set_setoid : Setoid_Theory set eqset.

Add Setoid set eqset set_setoid as Set_setoid.

Add Morphism add : eqset_add.
Save.

Add Morphism rem : eqset_rem.
Save.
Hint Resolve eqset_add eqset_rem.

Add Morphism union : eqset_union.
Save.
Hint Immediate eqset_union.

Lemma eqset_union_left :
  ∀ P1 Q P2,
    eqset P1 P2 → eqset (union P1 Q) (union P2 Q).

Lemma eqset_union_right :
  ∀ P Q1 Q2 ,
    eqset Q1 Q2 → eqset (union P Q1) (union P Q2).

Hint Resolve eqset_union_left eqset_union_right.

Add Morphism inter : eqset_inter.
Save.
Hint Immediate eqset_inter.

Add Morphism compl : eqset_compl.
Save.
Hint Resolve eqset_compl.

Lemma eqset_add_empty : ∀ (a:A) (P:set), ~eqset (add a P) empty.
```

10.3 Finite sets given as an enumeration of elements

```
Inductive finite (P: set) : Type :=
    fin_eq_empty : eqset P empty → finite P
  | fin_eq_add : ∀ (x:A)(Q:set),
                 ¬ Q x→ finite Q → eqset P (add x Q) → finite P.
Hint Constructors finite.

Lemma fin_empty : (finite empty).

Lemma fin_add : ∀ (x:A)(P:set),
                ¬ P x → finite P → finite (add x P).

Lemma fin_eqset: ∀ (P Q : set), (eqset P Q)->(finite P)->(finite Q).

Hint Resolve fin_empty fin_add.
```

10.3.1 Emptyness is decidable for finite sets

```
Definition isempty (P:set) := eqset P empty.
Definition notempty (P:set) := not (eqset P empty).

Lemma isempty_dec : ∀ P, finite P → {isempty P}+{notempty P}.
```

10.3.2 Size of a finite set

```
Fixpoint size (P:set) (f:finite P) {struct f}: nat :=
    match f with fin_eq_empty _ ⇒ 0%nat
```

```
                | fin_eq_add _ Q _ f' _ ⇒ S (size f')
  end.
Lemma size_eqset : ∀ P Q (f:finite P) (e:eqset P Q),
    (size (fin_eqset e f)) = (size f).
```

10.4 Inclusion

```
Definition incl (P Q:set) := ∀ x, P x → Q x.
Lemma incl_refl : ∀ (P:set), incl P P.
Lemma incl_trans : ∀ (P Q R:set),
incl P Q → incl Q R → incl P R.
Lemma eqset_incl : ∀ (P Q : set), eqset P Q → incl P Q.
Lemma eqset_incl_sym : ∀ (P Q : set), eqset P Q → incl Q P.
Lemma eqset_incl_intro :
∀ (P Q : set), incl P Q → incl Q P → eqset P Q.
Hint Resolve incl_refl incl_trans eqset_incl_intro.
Hint Immediate eqset_incl eqset_incl_sym.
```

10.5 Properties of operations on sets

```
Lemma incl_empty : ∀ P, incl empty P.
Lemma incl_empty_false : ∀ P a, incl P empty → ¬ P a.
Lemma incl_add_empty : ∀ (a:A) (P:set), ¬ incl (add a P) empty.
Lemma eqset_empty_false : ∀ P a, eqset P empty → P a → False.
Hint Immediate incl_empty_false eqset_empty_false incl_add_empty.
Lemma incl_rem_stable : ∀ a P Q, incl P Q → incl (rem a P) (rem a Q).
Lemma incl_add_stable : ∀ a P Q, incl P Q → incl (add a P) (add a Q).
Lemma incl_rem_add_iff :
  ∀ a P Q, incl (rem a P) Q ↔ incl P (add a Q).
Lemma incl_rem_add:
  ∀ (a:A) (P Q:set),
     (P a) → incl Q (rem a P) → incl (add a Q) P.
Lemma incl_add_rem :
  ∀ (a:A) (P Q:set),
     ¬ Q a → incl (add a Q) P → incl Q (rem a P) .
Hint Immediate incl_rem_add incl_add_rem.
Lemma eqset_rem_add :
 ∀ (a:A) (P Q:set),
     (P a) → eqset Q (rem a P) → eqset (add a Q) P.
Lemma eqset_add_rem :
 ∀ (a:A) (P Q:set),
     ¬ Q a → eqset (add a Q) P → eqset Q (rem a P).
Hint Immediate eqset_rem_add eqset_add_rem.
Lemma add_rem_eq_eqset :
  ∀ x (P:set), eqset (add x (rem x P)) (add x P).
Lemma add_rem_diff_eqset :
  ∀ x y (P:set),
```

x<>y → eqset (add x (rem y P)) (rem y (add x P)).

Lemma add_eqset_in :
∀ x (P:set), P x → eqset (add x P) P.

Hint Resolve add_rem_eq_eqset add_rem_diff_eqset add_eqset_in.

Lemma add_rem_eqset_in :
∀ x (P:set), P x → eqset (add x (rem x P)) P.

Hint Resolve add_rem_eqset_in.

Lemma rem_add_eq_eqset :
∀ x (P:set), eqset (rem x (add x P)) (rem x P).

Lemma rem_add_diff_eqset :
∀ x y (P:set),
x<>y → eqset (rem x (add y P)) (add y (rem x P)).

Lemma rem_eqset_notin :
∀ x (P:set), ~P x → eqset (rem x P) P.

Hint Resolve rem_add_eq_eqset rem_add_diff_eqset rem_eqset_notin.

Lemma rem_add_eqset_notin :
∀ x (P:set), ~P x → eqset (rem x (add x P)) P.

Hint Resolve rem_add_eqset_notin.

Lemma rem_not_in : ∀ x (P:set), ¬ rem x P x.

Lemma add_in : ∀ x (P:set), add x P x.

Lemma add_in_eq : ∀ x y P, x=y → add x P y.

Lemma add_intro : ∀ x (P:set) y, P y → add x P y.

Lemma add_incl : ∀ x (P:set), incl P (add x P).

Lemma add_incl_intro : ∀ x (P Q:set), (Q x) → (incl P Q) → (incl (add x P) Q).

Lemma rem_incl : ∀ x (P:set), incl (rem x P) P.

Hint Resolve rem_not_in add_in rem_incl add_incl.

Lemma union_sym : ∀ P Q : set,
eqset (union P Q) (union Q P).

Lemma union_empty_left : ∀ P : set,
eqset P (union P empty).

Lemma union_empty_right : ∀ P : set,
eqset P (union empty P).

Lemma union_add_left : ∀ (a:A) (P Q: set),
eqset (add a (union P Q)) (union P (add a Q)).

Lemma union_add_right : ∀ (a:A) (P Q: set),
eqset (add a (union P Q)) (union (add a P) Q).

Hint Resolve union_sym union_empty_left union_empty_right
union_add_left union_add_right.

Lemma union_incl_left : ∀ P Q, incl P (union P Q).

Lemma union_incl_right : ∀ P Q, incl Q (union P Q).

Lemma union_incl_intro : ∀ P Q R, incl P R → incl Q R → incl (union P Q) R.

Hint Resolve union_incl_left union_incl_right union_incl_intro.

Lemma incl_union_stable : ∀ *P1 P2 Q1 Q2*,
incl *P1 P2* → incl *Q1 Q2* → incl (union *P1 Q1*) (union *P2 Q2*).
Hint Immediate incl_union_stable.

Lemma inter_sym : $\forall$ P Q : set,
eqset (inter P Q) (inter Q P).

Lemma inter_empty_left : $\forall$ P : set,
eqset empty (inter P empty).

Lemma inter_empty_right : $\forall$ P : set,
eqset empty (inter empty P).

Lemma inter_add_left_in : $\forall$ (a:A) (P Q: set),
(P a) $\rightarrow$ eqset (add a (inter P Q)) (inter P (add a Q)).

Lemma inter_add_left_out : $\forall$ (a:A) (P Q: set),
$\neg$ P a $\rightarrow$ eqset (inter P Q) (inter P (add a Q)).

Lemma inter_add_right_in : $\forall$ (a:A) (P Q: set),
Q a $\rightarrow$ eqset (add a (inter P Q)) (inter (add a P) Q).

Lemma inter_add_right_out : $\forall$ (a:A) (P Q: set),
$\neg$ Q a $\rightarrow$ eqset (inter P Q) (inter (add a P) Q).

Hint Resolve inter_sym inter_empty_left inter_empty_right
inter_add_left_in inter_add_left_out inter_add_right_in inter_add_right_out.

10.6 Generalized union

Definition gunion (I:Type)(F:I->set) : set := fun z $\Rightarrow$ $\exists$ i, F i z.

Lemma gunion_intro : $\forall$ I (F:I->set) i, incl (F i) (gunion F).

Lemma gunion_elim : $\forall$ I (F:I->set) (P:set), ($\forall$ i, incl (F i) P) $\rightarrow$ incl (gunion F) P.

Lemma gunion_monotonic : $\forall$ I (F G : I $\rightarrow$ set),
($\forall$ i, incl (F i) (G i))$\rightarrow$ incl (gunion F) (gunion G).

10.7 Removing an element from a finite set

Lemma finite_rem : $\forall$ (P:set) (a:A),
finite P $\rightarrow$ **finite** (rem a P).

Lemma size_finite_rem:
$\forall$ (P:set) (a:A) (f:finite P),
(P a) $\rightarrow$ size f = S (size (finite_rem a f)).

Require Import Arith.

Lemma size_incl :
$\forall$ (P:set)(f:finite P) (Q:set)(g:finite Q),
(incl P Q)$\rightarrow$ size f $\leq$ size g.

Lemma size_unique :
$\forall$ (P:set)(f:finite P) (Q:set)(g:finite Q),
(eqset P Q)$\rightarrow$ size f = size g.

10.8 Decidable sets

Definition dec (P:set) := $\forall$ x, {P x}+{~ P x}.

Lemma finite_incl : $\forall$ P:set,
finite P $\rightarrow$ $\forall$ Q:set, dec Q $\rightarrow$ incl Q P $\rightarrow$ **finite** Q.

Lemma finite_dec : $\forall$ P:set, **finite** P $\rightarrow$ dec P.

Lemma fin_add_in : $\forall$ (a:A) (P:set), **finite** P $\rightarrow$ **finite** (add a P).

Lemma finite_union :
$\forall$ P Q, **finite** P $\rightarrow$ **finite** Q $\rightarrow$ **finite** (union P Q).

```
Lemma finite_full_dec : ∀ P:set, finite full → dec P → finite P.
Require Import Lt.
```

10.8.1 Filter operation

```
Lemma finite_inter : ∀ P Q, dec P → finite Q → finite (inter P Q).
Lemma size_inter_empty : ∀ P Q (decP:dec P) (e:eqset Q empty),
  size (finite_inter decP (fin_eq_empty e))=O.
Lemma size_inter_add_in :
    ∀ P Q R (decP:dec P)(x:A)(nq:~Q x)(FQ:finite Q)(e:eqset R (add x Q)),
     P x ->size (finite_inter decP (fin_eq_add nq FQ e))=S (size (finite_inter decP FQ)).
Lemma size_inter_add_notin :
    ∀ P Q R (decP:dec P)(x:A)(nq:~Q x)(FQ:finite Q)(e:eqset R (add x Q)),
  ¬ P x → size (finite_inter decP (fin_eq_add nq FQ e))=size (finite_inter decP FQ).
Lemma size_inter_incl : ∀ P Q (decP:dec P)(FP:finite P)(FQ:finite Q),
   (incl P Q) → size (finite_inter decP FQ)=size FP.
```

10.8.2 Selecting elements in a finite set

```
Fixpoint nth_finite (P:set) (k:nat) (PF : finite P) {struct PF}: (k < size PF) → A :=
  match PF as F return (k < size F) → A with
      fin_eq_empty H ⇒ (fun (e : k<0) ⇒ match lt_n_O k e with end)
    | fin_eq_add x Q nqx fq eqq ⇒
        match k as k0 return k0<S (size fq)->A with
            O ⇒ fun e ⇒ x
          | (S k1) ⇒ fun (e:S k1<S (size fq)) ⇒ nth_finite fq (lt_S_n k1 (size fq) e)
        end
  end.
```

A set with size > 1 contains at least 2 different elements

```
Lemma select_non_empty : ∀ (P:set), finite P → notempty P → sigT P.
Lemma select_diff : ∀ (P:set) (FP:finite P),
    (1 < size FP)%nat → sigT (fun x ⇒ sigT (fun y ⇒ P x ∧ P y ∧ x<>y)).
End sets.
Hint Resolve eqset_refl.
Hint Resolve eqset_add eqset_rem.
Hint Immediate eqset_sym finite_dec finite_full_dec eqset_incl eqset_incl_sym eqset_incl_intro.
Hint Resolve incl_refl.
Hint Immediate incl_union_stable.
Hint Resolve union_incl_left union_incl_right union_incl_intro incl_empty rem_incl
incl_rem_stable incl_add_stable.
Hint Constructors finite.
Hint Resolve add_in add_in_eq add_intro add_incl add_incl_intro union_sym union_empty_left union_empty_right
union_add_left union_add_right finite_union eqset_union_left
eqset_union_right.
Implicit Arguments full [].
Implicit Arguments empty [].
Add Parametric Relation (A:Type) : (set A) (eqset (A:=A))
      reflexivity proved by (eqset_refl (A:=A))
      symmetry proved by (eqset_sym (A:=A))
      transitivity proved by (eqset_trans (A:=A))
```

```
as eqset_rel.
Add Parametric Relation (A:Type) : (set A) (incl (A:=A))
      reflexivity proved by (incl_refl (A:=A))
      transitivity proved by (incl_trans (A:=A))
as incl_rel.
```

11 Cover.v: Characteristic functions

```
Require Export Prog.
Require Export Sets.
Require Export Arith.
```

Properties of zero_one functions

```
Definition zero_one (A:Type)(f:MF A) := ∀ x, orc (f x == 0) (f x == 1).
Hint Unfold zero_one.

Lemma zero_one_not_one :
∀ (A:Type)(f:MF A) x, zero_one f → ¬ 1 ≤ f x → f x == 0.

Lemma zero_one_not_zero :
∀ (A:Type)(f:MF A) x, zero_one f → ¬ f x ≤ 0 → f x == 1.

Hint Resolve zero_one_not_one zero_one_not_zero.

Definition fesp_zero_one : ∀ (A:Type)(f g:MF A),
      zero_one f → zero_one g → zero_one (fesp f g).
Save.

Lemma fesp_conj_zero_one : ∀ (A:Type)(f g:MF A),
      zero_one f → fesp f g == fconj f g.

Lemma fconj_zero_one : ∀ (A:Type)(f g:MF A),
      zero_one f → zero_one g → zero_one (fconj f g).

Lemma fplus_zero_one : ∀ (A:Type)(f g:MF A),
      zero_one f → zero_one g → zero_one (fplus f g).

Lemma finv_zero_one : ∀ (A:Type)(f :MF A),
      zero_one f → zero_one (finv f).

Lemma fesp_zero_one_mult_left : ∀ (A:Type)(f:MF A)(p:U),
      zero_one f → ∀ x, f x & p == f x × p.

Lemma fesp_zero_one_mult_right : ∀ (A:Type)(p:U)(f:MF A),
     zero_one f → ∀ x, p & f x == p × f x.
Hint Resolve fesp_zero_one_mult_left fesp_zero_one_mult_right.
```

11.1 Covering functions

```
Definition cover (A:Type)(P:set A)(f:MF A) :=
      ∀ x, (P x → 1 ≤ f x) ∧ (¬ P x → f x ≤ 0).

Lemma cover_eq_one : ∀ (A:Type)(P:set A)(f:MF A) (z:A),
       cover P f → P z → f z == 1.

Lemma cover_eq_zero : ∀ (A:Type)(P:set A)(f:MF A) (z:A),
       cover P f → ¬ P z → f z == 0.

Lemma cover_orc_0_1 : ∀ (A:Type)(P:set A)(f:MF A),
   cover P f → ∀ x, orc (f x == 0) (f x == 1).

Lemma cover_zero_one : ∀ (A:Type)(P:set A)(f:MF A),
   cover P f → zero_one f.
```

```
Lemma zero_one_cover : ∀ (A:Type)(f:MF A),
   zero_one f → cover (fun x ⇒ 1 ≤ f x) f.

Lemma cover_esp_mult_left : ∀ (A:Type)(P:set A)(f:MF A)(p:U),
      cover P f → ∀ x, f x & p == f x × p.

Lemma cover_esp_mult_right : ∀ (A:Type)(P:set A)(p:U)(f:MF A),
     cover P f → ∀ x, p & f x == p × f x.
Hint Immediate cover_esp_mult_left cover_esp_mult_right.

Lemma cover_elim : ∀ (A:Type)(P:set A)(f:MF A),
cover P f → ∀ x, orc (~P x ∧ f x == 0) (P x ∧ f x == 1).

Lemma cover_eq_one_elim_class : ∀ (A:Type)(P Q:set A)(f:MF A),
       cover P f → ∀ z, f z == 1 → class (Q z) → incl P Q → Q z.

Lemma cover_eq_one_elim : ∀ (A:Type)(P:set A)(f:MF A),
       cover P f → ∀ z, f z == 1 → ¬ ¬ P z.

Lemma cover_eq_zero_elim : ∀ (A:Type)(P:set A)(f:MF A) (z:A),
       cover P f → f z == 0 → ¬ P z.

Lemma cover_unit : ∀ (A:Type)(P:set A)(f:MF A)(a:A),
            cover P f → P a → 1 ≤ mu (Munit a) f.

Lemma cover_let : ∀ (A B:Type)(m1: distr A)(m2: A->distr B) (P:set A)(cP:MF A)(f:MF B)(p:U),
     cover P cP → 1 ≤ mu m1 cP → (∀ x:A, P x → p ≤ mu (m2 x) f) → p ≤ mu (Mlet m1 m2) f.

Lemma cover_incl_fle : ∀ (A:Type)(P Q:set A)(f g:MF A),
       cover P f → cover Q g → incl P Q → f ≤ g.

Lemma cover_incl_eq: ∀ (A:Type)(P:set A)(f g:MF A),
       cover P f → cover P g → f == g.

Lemma cover_eqset_stable : ∀ (A:Type)(P Q:set A)(EQ:eqset P Q)(f:MF A),
       cover P f → cover Q f.

Lemma cover_eq_stable : ∀ (A:Type)(P:set A)(f g:MF A),
       cover P f → f == g → cover P g.

Lemma cover_eqset_eq_stable : ∀ (A:Type)(P Q:set A)(f g:MF A),
       cover P f → eqset P Q → f == g → cover Q g.

Add Parametric Morphism (A:Type) : (cover (A:=A))
with signature eqset (A:=A) ==>Oeq ==>iff as cover_eqset_compat.
Save.

Lemma cover_union : ∀ (A:Type)(P Q:set A)(f g : MF A),
     cover P f → cover Q g → cover (union P Q) (fplus f g).

Lemma cover_inter_esp : ∀ (A:Type)(P Q:set A)(f g : MF A),
       cover P f → cover Q g → cover (inter P Q) (fesp f g).

Lemma cover_inter_mult : ∀ (A:Type)(P Q:set A)(f g : MF A),
       cover P f → cover Q g → cover (inter P Q) (fun x ⇒ f x × g x).

Lemma cover_compl : ∀ (A:Type)(P:set A)(f:MF A),
       cover P f → cover (compl P) (finv f).

Lemma cover_empty : ∀ (A:Type), cover (empty A) (fzero A).
```

11.2 Caracteristic functions for decidable predicates

```
Definition carac (A:Type)(P:set A)(Pdec : dec P) : MF A
     := fun z ⇒ if Pdec z then 1 else 0.

Lemma carac_incl: ∀ (A:Type)(P Q:A → Prop)(Pdec: dec P)(Qdec: dec Q),
                                  incl P Q → carac Pdec ≤ carac Qdec.
```

Lemma *carac_monotonic* : ∀ (*A B*:Type)(P:A → Prop)(*Q*:B->Prop)(Pdec: *dec P*)(*Qdec*: *dec Q*) *x y*,
(*P x* → *Q y*) → *carac Pdec x* ≤ *carac Qdec y*.
Hint Resolve *carac_monotonic*.

Lemma *carac_eq_compat* : ∀ (*A B*:Type)(P:A → Prop)(*Q*:B->Prop)(Pdec: *dec P*)(*Qdec*: *dec Q*) *x y*,
(*P x* ↔ *Q y*) → *carac Pdec x* == *carac Qdec y*.
Hint Resolve *carac_eq_compat*.

Lemma *carac_one* : ∀ (*A*:Type)(P:A → Prop)(*Pdec*:dec *P*)(*z*:A),
P z → *carac Pdec z* == 1.

Lemma *carac_zero* : ∀ (*A*:Type)(P:A → Prop)(*Pdec*:dec *P*)(*z*:A),
¬ *P z* → *carac Pdec z* == 0.

Lemma *cover_dec* : ∀ (*A*:Type)(P:set *A*)(*Pdec* : *dec P*), *cover P* (*carac Pdec*).

Lemma *cover_mult_fun* : ∀ (*A*:Type)(P:set *A*)(*cP* : *MF A*)(*f g*:A->U),
(∀ *x*, *P x* → *f x* == *g x*) → *cover P cP* → ∀ *x*, *cP x* × *f x* == *cP x* × *g x*.

Lemma *cover_esp_fun* : ∀ (*A*:Type)(P:set *A*)(*cP* : *MF A*)(*f g*:A->U),
(∀ *x*, *P x* → *f x* == *g x*) → *cover P cP* → ∀ *x*, *cP x* & *f x* == *cP x* & *g x*.

Lemma *cover_esp_fun_le* : ∀ (*A*:Type)(P:set *A*)(*cP* : *MF A*)(*f g*:A->U),
(∀ *x*, *P x* → *f x* ≤ *g x*) → *cover P cP* → ∀ *x*, *cP x* & *f x* ≤ *cP x* & *g x*.
Hint Resolve *cover_esp_fun_le*.

Lemma *cover_ok* : ∀ (*A*:Type)(P *Q*:set *A*)(*f g* : *MF A*),
(∀ *x*, *P x* → ¬ *Q* x) → *cover P f* → *cover Q g* → *fplusok f g*.
Hint Resolve *cover_ok*.

11.3 Assuming m is a distribution under assumption P and cP is 0 or 1, builds

a distribution which is m if cP is 1 and 0 otherwise

Definition Mrestr A (*cp*:U) (*m*:M *A*) : *M A* := *UMult cp* @ *m*.

Lemma *Mrestr_simpl* : ∀ *A cp* (*m*:M *A*) *f*, *Mrestr cp m f* = *cp* × (*m f*).

Lemma *Mrestr0* : ∀ *A cP* (*m*:M *A*), *cP* ≤ 0 → ∀ *f*, *Mrestr cP m f* == 0.

Lemma *Mrestr1* : ∀ *A cP* (*m*:M *A*), 1 ≤ *cP* → ∀ *f*, *Mrestr cP m f* == *m f*.

Definition *distr_restr* : ∀ *A* (*P*:Prop) (*cp*:U) (*m*:M *A*),
((P → 1 ≤ *cp*) ∧ (¬ *P* → cp ≤ 0)) → (*P* → *stable_inv m*) →
(*P* → *stable_plus m*) → (*P* → *stable_mult m*) → (*P* → *continuous m*)
→ *distr A*.
Defined.

Lemma *distr_restr_simpl* : ∀ *A* (*P*:Prop) (*cp*:U) (*m*:M *A*)
(*Hp*: (P → 1 ≤ *cp*) ∧ (¬ *P* → *cp* ≤ 0)) (*Hinv*:P → *stable_inv m*)
(*Hplus*:P → *stable_plus m*)(*Hmult*:P → *stable_mult m*)(*Hcont*:P → *continuous m*) *f*,
mu (*distr_restr cp Hp Hinv Hplus Hmult Hcont*) *f* = *cp* × *m f*.

11.4 Modular reasoning on programs

Lemma *range_cover* : ∀ *A* (P:A → Prop) d *cP*, *range P d* → *cover P* cP →
∀ *f*, *mu d f* == *mu d* (fun *x* ⇒ *cP x* × *f x*).

Lemma *mu_cut* : ∀ (*A*:Type)(m:distr A)(*P*:set A)(*cP f g*:MF *A*),
cover P cP → (∀ *x*, *P x* → *f x* == *g x*) → 1<=mu *m cP* → *mu m f* == *mu* m *g*.

11.5 Uniform measure on finite sets

Section *SigmaFinite*.

```
Variable A:Type.
Variable decA : ∀ x y:A, {x=y}+{~x=y}.
Section RandomFinite.
```

11.5.1 Distribution for *random_fin P* over $\{k : nat | k \leq n\}$

The distribution associated to *random_fin P* is $f \mapsto \Sigma_{a \in P} \frac{f(a)}{n+1}$ with $n+1$ the size of P we cannot factorize $\frac{1}{n+1}$ because of possible overflow

```
Fixpoint sigma_fin (f:A->U)(P:A->Prop)(FP:finite P){struct FP}:U :=
match FP with
  | (fin_eq_empty eq) ⇒ 0
  | (fin_eq_add x Q nQx FQ eq) ⇒ (f x) + sigma_fin f FQ
end.

Definition retract_fin (P:A->Prop) (f:A->U) :=
     ∀ Q (FQ: finite Q), incl Q P → ∀ x, ~(Q x) → (P x) → f x ≤ [1-](sigma_fin f FQ).

Lemma retract_fin_inv :
     ∀ (P:A->Prop) (f:A->U),
               retract_fin P f → ∀ Q (FQ: finite Q), incl Q P → ∀ x, ~(Q x) → (P x) → sigma_fin f FQ
≤[1-]f x.

Hint Immediate retract_fin_inv.

Lemma retract_fin_incl : ∀ P Q f, retract_fin P f → incl Q P → retract_fin Q f.

Lemma sigma_fin_monotonic : ∀ (f g : A → U)(P:A->Prop)(FP:finite P),
        (∀ x, P x → (f x)<=(g x))→ sigma_fin f FP ≤ sigma_fin g FP.

Lemma sigma_fin_eq_compat :
∀ (f g : A → U)(P:A->Prop)(FP:finite P),
        (∀ x, P x → (f x)==(g x))→ sigma_fin f FP == sigma_fin g FP.

Lemma retract_fin_le : ∀ (P:A->Prop) (f g:A->U),
         (∀ x, P x → f x ≤ g x) → retract_fin P g → retract_fin P f.

Lemma sigma_fin_mult : ∀ (f : A → U) c (P:A->Prop)(FP:finite P),
        retract_fin P f → sigma_fin (fun k ⇒ c × f k) FP == c × sigma_fin f FP.

Lemma sigma_fin_plus : ∀ (f g: A → U) (P:A->Prop)(FP:finite P),
        sigma_fin (fun k ⇒ f k + g k) FP == sigma_fin f FP + sigma_fin g FP.

Lemma sigma_fin_prod_maj :
∀ (f g : A → U)(P:A->Prop)(FP:finite P),
        sigma_fin (fun k ⇒ f k × g k) FP ≤ sigma_fin f FP.

Lemma sigma_fin_prod_le :
∀ (f g : A → U) (c:U) , (∀ k, f k ≤ c) → ∀ (P:A->Prop)(FP:finite P),
    retract_fin P g → sigma_fin (fun k ⇒ f k × g k) FP ≤ c × sigma_fin g FP.

Lemma sigma_fin_prod_ge :
∀ (f g : A → U) (c:U) , (∀ k, c ≤ f k) → ∀ (P:A->Prop)(FP:finite P),
    retract_fin P g → c × sigma_fin g FP ≤ sigma_fin (fun k ⇒ f k × g k) FP.
Hint Resolve sigma_fin_prod_maj sigma_fin_prod_ge sigma_fin_prod_le.

Lemma sigma_fin_inv : ∀ (f g : A → U)(P:A->Prop)(FP:finite P),
        retract_fin P f →
        [1-] sigma_fin (fun k ⇒ f k × g k) FP ==
        sigma_fin (fun k ⇒ f k × [1-] g k) FP + [1-] sigma_fin f FP.

Lemma sigma_fin_eqset : ∀ f P Q (FP:finite P) (e:eqset P Q),
     (sigma_fin f (fin_eqset e FP)) = (sigma_fin f FP).

Lemma sigma_fin_rem : ∀ f P (FP:finite P) a,
```

$P\ a \to sigma\_fin\ f\ FP == \mathsf{f}\ a + sigma\_fin\ f\ (finite\_rem\ decA\ a\ FP)$.

Lemma *sigma_fin_incl* : $\forall\ f\ P$ (**FP**:finite P) Q (FQ:finite Q),
$(incl\ P\ Q) \to sigma\_fin\ f\ FP \leq sigma\_fin\ f\ FQ$.

Lemma *sigma_fin_unique* : $\forall\ f\ P\ Q$ (FP:finite P) (FQ:finite Q), $(eqset\ P\ Q) \to sigma\_fin\ f\ FP == sigma\_fin\ f\ FQ$.

Lemma *sigma_fin_cte* : $\forall\ c\ P$ (FP:finite P),
$sigma\_fin$ (fun _ $\Rightarrow$ c) $FP == (size\ FP)$ */ c.

End RandomFinite.

End *SigmaFinite*.

11.6 Properties of the Random distribution

Definition *le_dec* (n:nat) : *dec* (fun $x \Rightarrow (x \leq n)$%nat).
Defined.

Definition *lt_dec* (n:nat) : *dec* (fun $x \Rightarrow (x < n)$%nat).
Defined.

Definition gt_dec : $\forall\ x$, *dec* (*lt* x).
Defined.

Definition ge_dec : $\forall\ x$, *dec* (*le* x).
Defined.

Definition *carac_eq* n := *carac* (*eq_nat_dec* n).
Definition *carac_le* n := *carac* (*le_dec* n).
Definition *carac_lt* n := *carac* (*lt_dec* n).
Definition *carac_gt* n := *carac* (gt_dec n).
Definition *carac_ge* n := *carac* (*ge_dec* n).

Definition *is_eq* (n:nat) : *cover* (fun $x \Rightarrow n = x$) (*carac_eq* n) := *cover_dec* (*eq_nat_dec* n).
Definition *is_le* (n:nat) : *cover* (fun $x \Rightarrow (x <=$n)%nat) (carac_le n) := *cover_dec* (*le_dec* n).
Definition *is_lt* (n:nat) : *cover* (fun $x \Rightarrow (x < n)$%nat) (*carac_lt* n) := *cover_dec* (*lt_dec* n).
Definition *is_gt* (n:nat) : *cover* (fun $x \Rightarrow (n < x)$%nat) (*carac_gt* n):= *cover_dec* (*gt_dec* n).
Definition *is_ge* (n:nat) : *cover* (fun $x \Rightarrow (n \leq x)$%nat) (*carac_ge* n) := *cover_dec* (*ge_dec* n).

Lemma *carac_gt_S* :
$\forall\ x\ y$, *carac_gt* ($S\ y$) ($S\ x$) == *carac_gt* $y\ x$.

Lemma *carac_lt_S* : $\forall\ x\ y$, *carac_lt* ($S\ x$) ($S\ y$) == *carac_lt* $x\ y$.

Lemma *carac_le_S* : $\forall\ x\ y$, *carac_le* (S x) ($S\ y$) == *carac_le* $x\ y$.

Lemma *carac_ge_S* : $\forall\ x\ y$, *carac_ge* ($S\ x$) ($S\ y$) == *carac_ge* $x\ y$.

Lemma *carac_eq_S* : $\forall\ x\ y$, carac_eq ($S\ x$) ($S\ y$) == carac_eq $x\ y$.

Hint Resolve *carac_le_S carac_eq_S carac_lt_S carac_ge_S carac_gt_S*.

count the number of elements between 0 and n-1 which satisfy P Fixpoint *nb_elts* (P:nat $\to$ Prop)($Pdec$: *dec* P)(n:nat) {struct n} : *nat* :=
match n with
$0 \Rightarrow 0$%nat
| $S\ n \Rightarrow$ if *Pdec* n then (S (*nb_elts Pdec* n)) else (*nb_elts Pdec* n)
end.

Lemma *nb_elts_true* : $\forall$ (P:nat $\to$ Prop)($Pdec$: *dec* P)(n:nat),
$(\forall\ k, (k < n)$%nat $\to P\ k) \to$ *nb_elts Pdec* n =n.
Hint Resolve nb_elts_true.

- the probability for a random number between 0 and n to satisfy P is equal to the number of elements below n which satisfy P divided by n+1

Lemma *Random_carac* : ∀ (*P*:nat → Prop)(*Pdec* : *dec P*)(*n*:nat),
mu (*Random n*) (*carac Pdec*) == (*nb_elts Pdec* (*S n*)) */ [1/]1+n.

Lemma *nb_elts_lt_le* : ∀ *k n*, (*k* ≤ *n*)%nat → *nb_elts* (*lt_dec k*) *n* = *k*.

Lemma *nb_elts_lt_ge* : ∀ *k n*, (*n* ≤ *k*)%nat → *nb_elts* (*lt_dec k*) *n* = *n*.
Hint Resolve *nb_elts_lt_ge nb_elts_lt_le*.

Lemma *Random_lt* : ∀ *n k*, *mu* (*Random n*) (*carac_lt k*) == *k* */ [1/]1+n.

Hint Resolve *Random_lt*.

Lemma *Random_le* : ∀ *n k*, *mu* (*Random n*) (*carac_le k*) == (*S k*) */ [1/]1+n.

Hint Resolve *Random_le*.

12 IterFlip.v: An example of probabilistic termination

Require Export Prog.

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*.

Instance *Fiter_mon* :
monotonic (fun (*f*:Z → *distr Z*) (*x*:Z) ⇒ *Mif Flip* (*f* (*Zsucc x*)) (*Munit x*)).
Save.

Definition *Fiter* : (*Z* → *distr Z*) $\stackrel{m}{\to}$ (*Z* → *distr Z*)
:= *mon* (fun *f* (*x*:Z) ⇒ **Mif** *Flip* (*f* (*Zsucc x*)) (*Munit x*)).

Lemma *Fiter_simpl* : ∀ *f x*, *Fiter* f *x* = *Mif Flip* (*f* (*Zsucc x*)) (*Munit x*).

Definition *iterflip* : *Z* → *distr Z* := *Mfix* Fiter.

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* ⇒ 0 | (*S n*) ⇒ [1/2] × *p_ n* + [1/2] end.

Lemma *p_incr* : ∀ *n*, *p_ n* ≤ *p_* (*S n*).

Hint Resolve *p_incr*.

Definition *p* : *nat* $\stackrel{m}{\to}$ U := *fnatO_intro p_* p_incr.

Lemma *pS_simpl* : ∀ *n*, *p* (*S n*) = [1/2] × *p n* + [1/2].

Lemma *p_eq* : ∀ *n*:nat, *p n* == [1-]([1/2]^n).
Hint Resolve *p_eq*.

Lemma *p_le* : ∀ *n*:nat, [1-]([1/]1+n) ≤ *p n*.

Hint Resolve *p_le*.

Lemma *lim_p_one* : 1 ≤ *lub p*.

Hint Resolve *lim_p_one*.

12.2.2 Proof of probabilistic termination

Definition *q1* (*z1* *z2*:Z) := 1.

Lemma *iterflip_term* : **okfun** (fun $k \Rightarrow 1$) *iterflip* *q1*.

13 Choice.v: An example of probabilistic choice

Require Export Prog.

13.1 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 $x \Rightarrow$ *Mlet* *p2* (fun $y \Rightarrow$ Munit (*choice* x y))).

13.2 Main result

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

13.2.1 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* (*fone* A))==1.
Hypothesis *p2_terminates* : (*mu* *p2* (*fone* A))==1.

Variable Q : *MF* A.
Hypothesis *choiceok* : $\forall$ x y, Q x + Q $y \leq Q$ (*choice* x y).

13.2.2 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* : $\forall$ *k1* *k2*,
k1 $\leq$ *mu* *p1* $Q \to$ *k2* $\leq$ *mu* *p2* $Q \to$ (*k1* $\times$ ([1-] *k2*) + *k2*) $\leq$ *mu* *p* Q.

End *CHOICE*.

14 Bernoulli.v: Simulating Bernoulli and Binomial distributions

Require Export Cover.
Require Export *Misc*.

14.1 Program for computing a Bernoulli distribution

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

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

Hypothesis *dec_demi* : $\forall\ x :\ U$, $\{x < [1/2]\}+\{[1/2] \leq x\}$.

Instance *Fbern_mon* : *monotonic*
    (fun (f:U $\to$ *distr bool*) $p \Rightarrow$ *Mif Flip*
        (if *dec_demi* p then *Munit false* else f (p & p))
        (if *dec_demi* p then f (p + p) else *Munit true*)).
Save.

Definition *Fbern* : (*U* $\to$ *distr bool*) $\xrightarrow{m}$ (*U* $\to$ *distr bool*)
    := *mon* (fun f $p \Rightarrow$ *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_simpl* : $\forall\ f\ p$,
Fbern f p = *Mif Flip*
        (if *dec_demi* p then *Munit false* else f (p & p))
        (if *dec_demi* p then f (p + p) else *Munit true*).

Definition *bernoulli* : *U* $\to$ *distr bool* := *Mfix Fbern*.

14.2 Properties of the Bernoulli program

14.2.1 Proofs using fixpoint rules

Definition *Mubern* (q: *bool* $\to$ *U*) : MF *U* $\xrightarrow{m}$ *MF U*.
Defined.

Lemma *Mubern_simpl* : $\forall$ (q: *bool* $\to$ *U*) f p,
  Mubern **q** f p = if *dec_demi* p then [1/2]*(q *false*)+[1/2]*(f (p+p))
                          else [1/2]*(f (p&p)) + [1/2]*(q *true*).

Lemma *Mubern_eq* : $\forall$ (q: *bool* $\to$ *U*) (f:U $\to$ *distr* **bool**) (p:U),
            mu (*Fbern* f p) q == *Mubern* q (fun $y \Rightarrow$ *mu* (f y) q) p.
Hint Resolve *Mubern_eq*.

Lemma *Bern_eq* :
    $\forall\ q$: bool $\to$ *U*, $\forall\ p$, *mu* (**bernoulli** p) q == *mufix* (Mubern q) p.
Hint Resolve *Bern_eq*.

Lemma *Bern_commute* : $\forall\ q$: *bool* $\to$ *U*,
   mu_muF_commute_le Fbern (fun (x:U)=>q) (*Mubern* q).
Hint Resolve *Bern_commute*.

Lemma *Bern_term* : $\forall\ p$, *mu* (*bernoulli* p) (*fone bool*) == 1.
Hint Resolve *Bern_term*.

14.2.2 p is an invariant of Mubern qtrue

Lemma *MuBern_true* : $\forall\ p$, *Mubern B2U* (fun $q \Rightarrow q$) p == p.
Hint Resolve *MuBern_true*.

Lemma *MuBern_false* : $\forall\ p$, Mubern (*finv B2U*) (*finv* (fun $q \Rightarrow q$)) p == [1-]p.
Hint Resolve *MuBern_false*.

Lemma *Bern_true* : ∀ *p*, *mu* (*bernoulli p*) *B2U* == *p*.

Lemma *Bern_false* : ∀ *p*, *mu* (*bernoulli p*) *NB2U* == [1-]p.

Lemma *Mubern_inv* : ∀ (*q*: *bool* → *U*) (*f*:U → *U*) (*p*:U),
Mubern (*finv q*) (*finv f*) p == [1-] *Mubern* q *f p*.

14.2.3 Proofs using lubs

Invariant *pmin p pmin*$(p)(n) = p - \frac{1}{2^n}$

Property : $\forall p, p \leq$ [bernoulli p](**result** = true)

Definition *qtrue* (*p*:U) := *B2U*.
Definition *qfalse* (*p*:U) := *NB2U*.

Lemma *bernoulli_true* : *okfun* (fun *p* ⇒ *p*) *bernoulli qtrue*.

Property : $\forall p, 1 - p \leq$ [bernoulli p](**result** = false)

Lemma *bernoulli_false* : *okfun* (fun *p* ⇒ [1-] *p*) *bernoulli qfalse*.

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

Lemma *qtrue_qfalse_inv* : ∀ (*b*:bool) (*x*:U), *qtrue x b* == [1-] (*qfalse x b*).

Lemma *bernoulli_eq_true* : ∀ *p*, *mu* (*bernoulli p*) (*qtrue p*) == *p*.

Lemma *bernoulli_eq_false* : ∀ *p*, *mu* (*bernoulli p*) (*qfalse* p)== [1-]*p*.

Lemma *bernoulli_eq* : ∀ *p* f, *mu* (*bernoulli p*) *f* == *p* × *f true* + ([1-]*p*) × *f false*.

Lemma *bernoulli_total* : ∀ p , *mu* (*bernoulli p*) (*fone bool*)==1.

14.3 Binomial distribution

(binomial p n) gives k with probability $C_k^n p^k (1-p)^{n-k}$

14.3.1 Definition and properties of binomial coefficients

```
Fixpoint comb (k n:nat) {struct n} : nat :=
          match n with O ⇒ match k with O ⇒ (1%nat) | (S l) ⇒ O end
                 | (S m) ⇒ match k with O ⇒ (1%nat)
                                        | (S l) ⇒ ((comb l m) + (comb k m))%nat end
          end.
```

Lemma *comb_0_n* : ∀ *n*, *comb* 0 *n* = 1%nat.

Lemma *comb_not_le* : ∀ *n k*, *le* (*S n*) *k* → *comb k n*=0%nat.

Lemma *comb_Sn_n* : ∀ *n*, *comb* (*S n*) *n* =0%nat.

Lemma *comb_n_n* : ∀ *n*, *comb n n* = (1%nat).

Lemma *comb_1_Sn* : ∀ *n*, *comb* 1 (*S n*) = (*S n*).

Lemma *comb_inv* : ∀ *n k*, (*k*<=n)%nat → *comb k n* = *comb* (*n*-k) *n*.

Lemma *comb_n_Sn* : ∀ *n*, *comb n* (*S n*) = (*S n*).

Definition *fc* (*p*:U)(n *k*:nat) := (*comb k n*) */ (*p*^k × ([1-]p)^(n-k)).

Lemma *fcp_0* : ∀ *p n*, *fc p* n *O* == ([1-]*p*)^n.

Lemma *fcp_n* : ∀ *p n*, *fc p* n *n* == *p*^n.

Lemma *fcp_not_le* : ∀ *p n k*, (*S n* ≤ *k*)%nat → *fc p n k* == 0.

Lemma *fc0* : ∀ *n k*, *fc* 0 *n* (*S k*) == 0.
Hint Resolve *fc0*.

Add *Morphism fc* with *signature Oeq* ⟹ *eq* ⟹ *eq* ⟹ *Oeq*

```
as fc_eq_compat.
Save.

Hint Resolve fc_eq_compat.

Lemma sigma_fc0 : ∀ n k, sigma (fc 0 n) (S k) ==1.

Lemma retract_class : ∀ f n, class (retract f n).
Hint Resolve retract_class.

Lemma fc_retract :
    ∀ p n, ([1-]p^n == sigma (fc p n) n) → retract (fc p n) (S n).
Hint Resolve fc_retract.

Lemma fc_Nmult_def :
    ∀ p n k, ([1-]p^n == sigma (fc p n) n) → Nmult_def (comb k n) (p^k × ([1-]p) ^(n-k)).
Hint Resolve fc_Nmult_def.

Lemma fcp_S :
    ∀ p n k, ([1-]p^n == sigma (fc p n) n) → fc p (S n) (S k) == p × (fc p n k) + ([1-]p) × (fc p n (S k)).

Lemma sigma_fc_1 : ∀ p n, ([1-]p^n == sigma (fc p n) n) →1==sigma (fc p n) (S n).
Hint Resolve sigma_fc_1.

Lemma Uinv_exp : ∀ p n, [1-](p^n)==sigma (fc p n) n.

Hint Resolve Uinv_exp.

Lemma Nmult_comb : ∀ p n k, Nmult_def (comb k n) (p ^ k × ([1-] p) ^ (n - k)).
Hint Resolve Nmult_comb.
```

14.3.2 Definition of binomial distribution

```
Fixpoint binomial (p:U)(n:nat) {struct n}: distr nat :=
    match n with O ⇒ (Munit O)
                 | S m ⇒ Mlet (binomial p m)
                              (fun x ⇒ Mif (bernoulli p) (Munit (S x)) (Munit x))
    end.
```

14.3.3 Properties of binomial distribution

```
Lemma binomial_eq_k :
   ∀ p n k, mu (binomial p n) (carac_eq k) == fc p n k.

Lemma binomial_le_n :
   ∀ p n, 1 ≤ mu (binomial p n) (carac_le n).

Lemma binomial_le_S : ∀ p n k,
   mu (binomial p (S n)) (carac_le (S k)) ==
   p × (mu (binomial p n) (carac_le k)) + ([1-]p) × (mu (binomial p n) (carac_le (S k))).

Lemma binomial_lt_S : ∀ p n k,
   mu (binomial p (S n)) (carac_lt (S k)) ==
   p × (mu (binomial p n) (carac_lt k)) + ([1-]p) × (mu (binomial p n) (carac_lt (S k))).

Require Export Cover.
Require Arith.
```

15 Ycart.v: An exemple of partial termination

```
Section YcartExample.
```

15.1 Program giving an example of partiality

Given a function K : nat ->nat, N:nat

```
let rec ycart p x = if random p < K x then x else ycart (x+1)
```

The probability of termination is $1 - \prod_{k=x}^{\infty}(1 - K(k)/p + 1)$ `Variable` K : $nat \to nat$.
`Variable` N : nat.

`Instance` $FYcart\_mon$: $monotonic$
(`fun` (f:nat $\to$ $distr\ nat$) $n \Rightarrow$
$Mlet\ (Random\ N)$ (`fun` $x \Rightarrow$ `if` lt_dec (K n) x `then` $Munit\ n$ `else` $f\ (S\ n)$)).
`Save`.

`Definition` $FYcart$: $(nat \to distr\ nat) \xrightarrow{m} (nat \to distr$ nat)
:= **mon** (`fun` (f:nat $\to$ $distr\ nat$) $n \Rightarrow$
$Mlet\ (Random\ N)$ (`fun` $x \Rightarrow$ `if` lt_dec (K n) x `then` $Munit\ n$ `else` $f\ (S\ n)$)).

`Lemma` $FYcart\_simpl$: $\forall\ f\ n$,
$FYcart$ f $n = Mlet\ (Random\ N)$ (`fun` $x \Rightarrow$ `if` lt_dec (K n) x `then` $Munit\ n$ `else` $f\ (S\ n)$).

`Definition` $Ycart$: $nat \to distr$ nat := $Mfix\ FYcart$.

`Definition` $F\ x := (K\ x)$ */ [1/]1+N.

15.2 Properties of Ycart

`Lemma` $FYcart\_val$: $\forall$ (q: $MF\ nat$) f x,
$mu\ (FYcart\ f\ x)\ q == $ F $x \times q\ x +$ [1-]$(F$ x$) \times mu\ (f\ (S\ x))\ q$.

`Definition` $P\ (x : nat)$: $nat \xrightarrow{m} - \ U := Prod$ (`fun` $i \Rightarrow$ [1-]$F\ (x$+i)).

`Definition` **p** (x:nat) : $nat \xrightarrow{m} U := sigma$ (`fun` $k \Rightarrow F\ (x$+k) $\times\ P\ x\ k$).

`Lemma` P_prod : $\forall\ x\ k$, $F\ (x$+k) $\times\ P\ x\ k == P\ x\ k$ - $P\ x\ (S\ k)$.
`Hint Resolve` P_prod.

`Lemma` p_diff : $\forall\ x\ n$, (p x n : U) == [1-] $P\ x\ n$.
`Hint Resolve` p_diff.

`Lemma` p_lub : $\forall\ x$, $lub\ (p\ x) ==$ [1-] $prod\_inf$ (`fun` $i \Rightarrow$ [1-]$F\ (x$+i)).
`Hint Resolve` p_lub.

`Lemma` $p\_equation$: $\forall\ x\ n$, $p\ x\ (S$ n$) == F\ x +$ [1-]$(F\ x) \times p\ (S\ x)\ n$.
`Hint Resolve` $p\_equation$.

`Lemma` $Ycart\_term1$: $\forall\ x$, $mu\ (Ycart\ x)\ (fone\ nat) ==$ [1-] $prod\_inf$ (`fun` $i \Rightarrow$ [1-]$F\ (x$+i)).

A shorter proof using mu (Ycart x) (f_one nat) = mu h. muYcart h x

`Instance` $Ycart\_one\_mon$
: $monotonic$ (`fun` (p:nat $\to$ U) (y:nat) $\Rightarrow F\ y +$ [1-]$(F\ y) \times p\ (S\ y)$).
`Save`.

`Definition` $Ycart\_one$: $(nat \to U) \xrightarrow{m} (nat \to U)$:=
mon (`fun` (p:nat->U) (y:nat) $\Rightarrow F\ y +$ [1-]$(F\ y) \times p\ (S$ y)).

`Lemma` $Ycart\_one\_simpl$: $\forall\ p\ y$, $Ycart\_one$ p $y = F\ y +$ [1-]$(F\ y) \times p\ (S\ y)$.

`Lemma` $Ycart\_term2$: $\forall\ x$, $mu\ (Ycart\ x)\ (fone\ nat) ==$ [1-] $prod\_inf$ (`fun` $i \Rightarrow$ [1-]$F\ (x$+i)).

`Instance` $FYcart\_lt\_mon$: $monotonic$ (`fun` (p: $nat \to U$) (y:nat) $\Rightarrow$ [1-]$(F\ y) \times p\ (S\ y)$).
`Save`.

`Definition` $FYcart\_lt$: $(nat \to U) \xrightarrow{m} (nat \to U)$:=
mon (`fun` (p:nat->U) (y:nat) $\Rightarrow$ [1-]$(F\ y) \times p\ (S$ y)).

`Lemma` $FYcart\_lt\_simpl$: $\forall$ (p:nat $\to$ U) (y:nat), $FYcart\_lt\ p\ y =$ [1-]$(F\ y) \times p\ (S\ y)$.

`Lemma` $Ycart\_ltx$: $\forall\ x$, $mu\ (Ycart\ x)\ (carac\_lt\ x) \leq 0$.

```
Lemma Ycart_eqx : ∀ x, mu (Ycart x) (carac (eq_nat_dec x)) == F x.
End YcartExample.
```

References

[1] Philippe Audebaud and Christine Paulin-Mohring. Proofs of randomized algorithms in Coq. In Tarmo Uustalu, editor, *Mathematics of Program Construction, MPC'2006*, volume 4014 of *LNCS*, Kuressaare, Estonia, 2006. Springer-Verlag.

[2] Philippe Audebaud and Christine Paulin-Mohring. Proofs of randomized algorithms in Coq. *Science of Computer Programming*, 74(8):568–589, 2009.

[3] Dexter Kozen. Semantics of probabilistic programs. *J. of Computer and System Science*, 1981.

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

[5] Matthieu Sozeau and Nicolas Oury. First-Class Type Classes. In Sofiène Tahar, Otmame Ait-Mohamed, and César Muñoz, editors, *21th International Conference on Theorem Proving in Higher Order Logics*, LNCS. Springer-Verlag, August 2008.

[6] The Coq Development Team. *The Coq Proof Assistant Reference Manual – Version V8.1*, July 2006. .