ALEA: a library for reasoning on randomized algorithms in COQ Version 6

Christine Paulin-Mohring
with contributions by David Baelde and Pierre Courtieu
PROVAL Team
LRI, UMR 8623 Univ. Paris-Sud 11, CNRS & INRIA Saclay - Île-de-France

February 2, 2012

Contents

1 Misc.v: Preliminaries

```
Require Export Arith.
Require Import Coq.Classes.SetoidTactics.
Require Import Coq.Classes.SetoidClass.
Require Import Coq.Classes.Morphisms.

Open Local Scope signature_scope.

Lemma beq_nat_neq: ∀ x y : nat, x ≠ y → false = beq_nat x y.

Lemma if_beq_nat_nat_eq_dec : ∀ A (x y:nat) (a b:A),
  (if beq_nat x y then a else b) = if eq_nat_dec x y then a else b.

Definition ifte A (test:bool) (thn els:A) := if test then thn else els.

Add Parametric Morphism (A:Type) : (@ifte A)
  with signature (eq ==>eq ==> eq ==> eq) as ifte_morphism1.
```

Add *Parametric Morphism* (A:Type) x : (@*ifte* A x)
with *signature* ($eq \Rightarrow eq \Rightarrow eq$) as *ifte_morphism2*.

Add *Parametric Morphism* (A:Type) x y : (@*ifte* A x y)
with *signature* ($eq \Rightarrow eq$) as *ifte_morphism3*.

1.1 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:$A \to A \to A$) (x:A) (u:$nat \to A$) (n:nat) {struct n}: A :=
match n with $O \Rightarrow x$ | (S p) $\Rightarrow f$ (u p) (*compn* f x u p) end.

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

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

1.2 Reducing if constructs

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

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

1.3 Classical reasoning

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

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* $C \to (A \to C) \to (B \to C) \to C$.

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

Lemma *orc_right* : $\forall$ A B:Prop, $B \to$ *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, ($\neg A \to \neg B \to$ *False*) $\to$ *orc* A B.

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

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

Definition *exc* (A :Type)(P:$A \to$ Prop) :=
$\forall$ C:Prop, *class* $C \to (\forall x{:}A,\ P\ x \to C) \to C$.

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

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

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

Lemma *not_and_elim_left* : $\forall$ A B, $\neg (A \wedge B) \to A \to \neg B$.

Lemma *not_and_elim_right* : $\forall$ A B, $\neg (A \wedge B) \to B \to \neg A$.

Hint Resolve *class_orc class_and class_exc excluded_middle*.

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

1.4 Extensional equality

```
Definition feq A B (f g : A → B) := ∀ x, f x = g x.
```

```
Lemma feq_refl : ∀ A B (f:A→B), feq f f.
```

```
Lemma feq_sym : ∀ A B (f g : A → B), feq f g → feq g f.
```

```
Lemma feq_trans : ∀ A B (f g h: A → B), feq f g → feq g h → feq f h.
```

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

```
Add Parametric Relation (A B : Type) : (A → 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.
```

Computational version of elimination on CompSpec

```
Lemma CompSpec_rect : ∀ (A : Type) (eq lt : A → A → Prop) (x y : A)
       (P : comparison → Type),
       (eq x y → P Eq) →
       (lt x y → P Lt) →
       (lt y x → P Gt)
    → ∀ c : comparison, CompSpec eq lt x y c → P c.
```

Decidability `Require Omega.`

```
Lemma dec_sig_lt : ∀ P : nat → Prop, (∀ x, {P x}+{ ¬ P x})
  → ∀ n, {i | i < n ∧ P i}+{∀ i, i < n → ¬ P i}.
```

```
Lemma dec_exists_lt : ∀ P : nat → Prop, (∀ x, {P x}+{ ¬ P x})
  → ∀ n, {∃ i, i < n ∧ P i}+{~ ∃ i, i < n ∧ P i}.
```

```
Definition eq_nat2_dec : ∀ p q : nat×nat, { p=q }+{~ p=q }.
Defined.
```

2 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.
```

2.1 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.
Opaque OrderEquiv.

Class ord A :=
  { Oeq : relation A;
     Ole : relation A;
     order_rel :> Order Oeq Ole }.

Lemma OrdSetoid `(o:ord A) : Setoid A.

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.

Definition Oge {O} {o:ord O} := fun (x y:O) ⇒ y ≤ x.
Infix ">=" := Oge.

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.

Add Parametric Morphism `(o:ord A): (Ole (ord:=o))
with signature (Oeq (A:=A) ⇒ Oeq (A:=A) ⇒ iff) as Ole_eq_compat_iff.
```

```
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.
```

2.2 Definition and properties of $x < y$

```
Definition Olt `{o:ord A} (r1 r2:A) : Prop := (r1 ≤ r2) ∧ ¬ (r1 ≡ r2).

Infix "<" := Olt.

Lemma Olt_eq_compat `{o:ord A} :
∀ x1 x2 : A, x1 ≡ x2 → ∀ x3 x4 : A, x3 ≡ x4 → x1 < x3 → x2 < x4.

Add Parametric Morphism `{o:ord A} : (Olt (A:=A))
with signature Oeq ==>Oeq ==>iff as Olt_iff_morphism.
Save.

Lemma Olt_neq `{o:ord A} : ∀ x y:A, x < y → ¬ x ≡ y.

Lemma Olt_neq_rev `{o:ord A} : ∀ x y:A, x < y → ¬ y ≡ x.

Lemma Olt_le `{o:ord A} : ∀ x y, x < y → x ≤ y.

Lemma Olt_notle `{o:ord A} : ∀ x y, x < y → ¬ y ≤ x.

Lemma Olt_trans `{o:ord A} : ∀ x y z:A, x < y → y < z → x < z.

Lemma Ole_diff_lt `{o:ord A} : ∀ x y : A, x ≤ y → ¬ x ≡ y → x < y.

Hint Immediate @Olt_neq @Olt_neq_rev @Olt_le @Olt_notle.
Hint Resolve @Ole_diff_lt.

Lemma Olt_antirefl `{o:ord A} : ∀ x:A, ¬ x < x.

Lemma Ole_lt_trans `{o:ord A} : ∀ x y z:A, x ≤ y → y < z → x < z.

Lemma Olt_le_trans `{o:ord A} : ∀ x y z:A, x < y → y ≤ z → x < z.

Hint Resolve @Olt_antirefl.

Lemma Ole_not_lt `{o:ord A} : ∀ x y:A, x ≤ y → ¬ y < x.
Hint Resolve @Ole_not_lt.

Add Parametric Morphism `{o:ord A} : (Olt (A:=A))
    with signature Ole --> Ole ==>Basics.impl as Olt_le_compat.
Qed.
```

2.2.1 Dual order

- *Iord x y = y ≤ x*

```
Definition Iord : ∀ O {o:ord O}, ord O.
Defined.
Implicit Arguments Iord [[o]].
```

2.2.2 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 := ∀ 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.
```

2.3 Monotonicity

2.3.1 Definition and properties

```
Class monotonic `{o1:ord Oa} `{o2:ord Ob} (f : Oa → Ob) :=
       monotonic_def : ∀ x y, x ≤ y → f x ≤ f y.
Lemma monotonic_intro : ∀ `{o1:ord Oa} `{o2:ord Ob} (f : Oa → Ob),
  (∀ x y, x ≤ y → f x ≤ f y) → monotonic f.
Hint Resolve @monotonic_intro.

Add Parametric Morphism `{o1:ord Oa} `{o2:ord Ob} (f : Oa → Ob) {m:monotonic f} : f
with signature (Ole (A:=Oa) ⟹ Ole (A:=Ob))
as monotonic_morphism.
Save.
Class stable `{o1:ord Oa} `{o2:ord Ob} (f : Oa → Ob) :=
       stable_def : ∀ x y, x ≡ y → f x ≡ f y.
Hint Unfold stable.
Lemma stable_intro : ∀ `{o1:ord Oa} `{o2:ord Ob} (f : Oa → Ob),
  (∀ x y, x ≡ y → f x ≡ f y) → stable f.
Hint Resolve @stable_intro.
Add Parametric Morphism `{o1:ord Oa} `{o2:ord Ob} (f : Oa → Ob) {s:stable f} : f
with signature (Oeq (A:=Oa) ⟹ Oeq (A:=Ob))
as stable_morphism.
Save.
Typeclasses Opaque monotonic stable.
```

```
Instance monotonic_stable `{o1:ord Oa} `{o2:ord Ob} (f : Oa → Ob) {m:monotonic f}
          : stable f.
Save.
```

2.3.2 Type of monotonic functions

```
Record fmon `{o1:ord Oa} `{o2:ord Ob}:= mon
             {fmont :> Oa → Ob;
              fmonotonic: monotonic fmont}.

Implicit Arguments mon [[Oa] [o1] [Ob] [o2] [fmonotonic]].
Implicit Arguments fmon [[o1] [o2]].

Hint Resolve @fmonotonic.

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

Open Scope O_scope.

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

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

Hint Resolve @fstable.

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

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

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

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

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

Add Parametric Morphism `{o1:ord Oa} `{o2:ord Ob}
         : (fmont (Oa:=Oa) (Ob:=Ob))
         with signature Oeq ==>Oeq ==>Oeq as fmont_eq_morphism.
Qed.
```

2.3.3 Monotonicity and dual order

Lemma *Imonotonic* `{*o1*:*ord Oa*} `{*o2*:*ord Ob*} (*f*:*Oa* → *Ob*) {*m*:*monotonic f*}
  : *monotonic* (*o1*:=*Iord Oa*) (*o2*:=*Iord Ob*) *f*.

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

Definition *imon* `{*o1*:*ord Oa*} `{*o2*:*ord Ob*} (*f*:*Oa* → *Ob*) {*m*:*monotonic f*}
  : *Oa* −*m*→ *Ob* := *mon* (*o1*:=*Iord Oa*) (*o2*:=*Iord Ob*) *f*.

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

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

Lemma *Iord_app* {*A*} `{*o1*:*ord Oa*} (*x*: *A*) : ((*A* → *Oa*) −*m*→ *Oa*).

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

Definition *Imon* : ∀ `{*o1*:*ord Oa*} `{*o2*:*ord Ob*}, (*Oa* -*m*> *Ob*) → (*Oa* −*m*→ *Ob*).
Defined.

Lemma *Imon_simpl* : ∀ `{*o1*:*ord Oa*} `{*o2*:*ord Ob*} (*f*:*Oa* -*m*> *Ob*)(*x*:*Oa*),
  Imon f x = *f x*.

2.3.4 Monotonicity and equality

Lemma *mon_fun_eq_monotonic*
  : ∀ `{*o1*:*ord Oa*} `{*o2*:*ord Ob*} (*f*:*Oa* → *Ob*) (*g*:*Oa* -*m*> *Ob*),
  f ≡ *g* → *monotonic f*.

Definition *mon_fun_subst* `{*o1*:*ord Oa*} `{*o2*:*ord Ob*} (*f*:*Oa* → *Ob*) (*g*:*Oa* -*m*> *Ob*) (*H*:*f* ≡ *g*)
  : *Oa* -*m*> *Ob* := *mon f* (*fmonotonic*:= *mon_fun_eq_monotonic* _ _ *H*).

Lemma *mon_fun_eq*
  : ∀ `{*o1*:*ord Oa*} `{*o2*:*ord Ob*} (*f*:*Oa* → *Ob*) (*g*:*Oa* -*m*> *Ob*)
  (*H*:*f* ≡ *g*), *g* ≡ *mon_fun_subst f g H*.

2.3.5 Monotonic functions with 2 arguments

Class *monotonic2* `{*o1*: *ord Oa*} `{*o2*: *ord Ob*} `{*o3*:*ord Oc*} (*f*:*Oa* → *Ob* → *Oc*) :=
  monotonic2_intro : ∀ (*x y*:*Oa*) (*z t*:*Ob*), *x* ≤ *y* → *z* ≤ *t* → *f x z* ≤ *f y t*.

Instance *mon2_intro* `{*o1*: *ord Oa*} `{*o2*: *ord Ob*} `{*o3*:*ord Oc*} (*f*:*Oa* → *Ob* → *Oc*)
  {*m1*:*monotonic f*} {*m2*: ∀ *x*, *monotonic* (*f x*)} : *monotonic2 f* | 10.
Save.

Lemma *mon2_elim1* `{*o1*: *ord Oa*} `{*o2*: *ord Ob*} `{*o3*:*ord Oc*} (*f*:*Oa* → *Ob* → *Oc*)
  {*m*:*monotonic2 f*} : *monotonic f*.

Lemma *mon2_elim2* `{*o1*: *ord Oa*} `{*o2*: *ord Ob*} `{*o3*:*ord Oc*} (*f*:*Oa* → *Ob* → *Oc*)
  {*m*:*monotonic2 f*} : ∀ *x*, *monotonic* (*f x*).
Hint Immediate @*mon2_elim1* @*mon2_elim2*: *typeclass_instances*.

Definition *mon_comp* {*A*} `{*o1*: *ord Oa*} `{*o2*: *ord Ob*}
  (*f*:*A* → *Oa* → *Ob*) {*mf*:∀ *x*, *monotonic* (*f x*)} : *A* → *Oa* -*m*> *Ob*
  := fun *x* ⇒ *mon* (*f x*).

Instance *mon_fun_mon* `{*o1*: *ord Oa*} `{*o2*: *ord Ob*} `{*o3*:*ord Oc*} (*f*:*Oa* → *Ob* → *Oc*)
  {*m*:*monotonic2 f*} : *monotonic* (fun *x* ⇒ *mon* (*f x*)).
Save.

```
Class stable2 `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc} (f:Oa → Ob → Oc) :=
    stable2_intro : ∀ (x y:Oa) (z t:Ob), x≡y → z ≡ t → f x z ≡ f y t.

Instance monotonic2_stable2 `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc}
    (f:Oa → Ob → Oc) {m:monotonic2 f} : stable2 f.
Save.

Typeclasses Opaque monotonic2 stable2.

Definition mon2 `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc} (f:Oa → Ob → Oc)
     {mf:monotonic2 f} : Oa -m> Ob -m> Oc := mon (fun x ⇒ mon (f x)).

Lemma mon2_simpl : ∀ `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc} (f:Oa → Ob → Oc)
     {mf:monotonic2 f} x y, mon2 f x y = f x y.
Hint Resolve @mon2_simpl.

Lemma mon2_le_compat : ∀ `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc}
     (f g:Oa → Ob → Oc) {mf: monotonic2 f} {mg:monotonic2 g},
     f ≤ g → mon2 f ≤ mon2 g.

Definition fun2 `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc} (f:Oa → Ob -m> Oc)
     : Oa → Ob → Oc := fun x ⇒ f x.

Instance fmon2_mon `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc} (f:Oa → Ob -m> Oc) :
      ∀ x:Oa, monotonic (fun2 f x).
Save.

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

Instance fmonotonic2 `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} (f:Oa -m> Ob -m> Oc)
        : monotonic2 (fun2 f).
Save.
Hint Resolve @fmonotonic2.

Definition mfun2 `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} (f:Oa -m> Ob -m> Oc)
   : Oa-m> (Ob → Oc) := mon (fun2 f).

Lemma mfun2_simpl : ∀ `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} (f:Oa -m> Ob -m> Oc) x y,
     mfun2 f x y = f x y.

Instance mfun2_mon `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc}
        (f:Oa -m> Ob -m> Oc) x : monotonic (mfun2 f x).
Save.

Lemma mon2_fun2 : ∀ `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc}
     (f:Oa -m> Ob -m> Oc), mon2 (fun2 f) ≡ f.

Lemma fun2_mon2 : ∀ `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc}
      (f:Oa → Ob → Oc) {mf:monotonic2 f} , fun2 (mon2 f) ≡ f.
Hint Resolve @mon2_fun2 @fun2_mon2.

Instance fstable2 `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} (f:Oa -m> Ob -m> Oc)
              : stable2 (fun2 f).
Save.
Hint Resolve @fstable2.

Definition Imon2 : ∀ `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc},
     (Oa -m> Ob -m> Oc) → (Oa –m> Ob –m→ Oc).
Defined.

Lemma Imon2_simpl : ∀ `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc}
      (f:Oa -m> Ob -m> Oc) (x:Oa) (y: Ob),
      Imon2 f x y = f x y.
```

```
Lemma Imonotonic2 `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc}
      (f:Oa → Ob → Oc){mf : monotonic2 f}
      : monotonic2 (o1:=Iord Oa) (o2:=Iord Ob) (o3:=Iord Oc) f.
```

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

```
Definition imon2 `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc}
      (f:Oa → Ob → Oc){mf : monotonic2 f} : Oa −m> Ob −m→ Oc :=
      mon2 (o1:=Iord Oa) (o2:=Iord Ob) (o3:=Iord Oc) f.
```

```
Lemma imon2_simpl : ∀ `{o1: ord Oa} `{o2: ord Ob} `{o3:ord Oc}
      (f:Oa → Ob → Oc){mf : monotonic2 f} (x:Oa) (y:Ob),
      imon2 f x y = f x y.
```

2.4 Sequences

2.4.1 Usual order on natural numbers

```
Instance natO : ord nat :=
    { Oeq := fun n m : nat ⇒ n = m;
        Ole := fun n m : nat ⇒ (n ≤ m)%nat}.
Defined.
```

```
Lemma le_Ole : ∀ n m, ((n ≤ m)%nat)-> n ≤ m.
Hint Resolve le_Ole.
```

```
Lemma nat_monotonic : ∀ {O} {o:ord O}
                  (f:nat → O), (∀ n, f n ≤ f (S n)) → monotonic f.
Hint Resolve @nat_monotonic.
```

```
Definition fnatO_intro : ∀ {O} {o:ord O} (f:nat → O), (∀ n, f n ≤ f (S n)) → nat -m> O.
Defined.
```

```
Lemma fnatO_elim : ∀ {O} {o:ord O} (f:nat -m> O) (n:nat), f n ≤ f (S n).
Hint Resolve @fnatO_elim.
```

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

```
Definition seq_lift_left {O} (f:nat → O) n := fun k ⇒ f (n+k)%nat.
```

```
Instance mon_seq_lift_left
  : ∀ n {O} {o:ord O} (f:nat → O) {m:monotonic f}, monotonic (seq_lift_left f n).
Save.
```

```
Definition mseq_lift_left : ∀ {O} {o:ord O} (f:nat -m> O) (n:nat), nat -m> O.
Defined.
```

```
Lemma mseq_lift_left_simpl : ∀ {O} {o:ord O} (f:nat -m> O) (n k:nat),
    mseq_lift_left f n k = f (n+k)%nat.
```

```
Lemma mseq_lift_left_le_compat : ∀ {O} {o:ord O} (f g:nat -m> O) (n:nat),
               f ≤ g → mseq_lift_left f n ≤ 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 ==>eq ==>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 ==>eq ==>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 → O) n := fun k ⇒ f (k+n)%nat.

Instance mon_seq_lift_right
   : ∀ n {O} {o:ord O} (f:nat → O) {m:monotonic f}, monotonic (seq_lift_right f n).
Save.

Definition mseq_lift_right : ∀ {O} {o:ord O} (f:nat -m> O) (n:nat), nat -m> O.
Defined.

Lemma mseq_lift_right_simpl : ∀ {O} {o:ord O} (f:nat -m> O) (n k:nat),
     mseq_lift_right f n k = f (k+n)%nat.

Lemma mseq_lift_right_le_compat : ∀ {O} {o:ord O} (f g:nat -m> O) (n:nat),
                f ≤ g → mseq_lift_right f n ≤ 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.
```

2.4.2 Monotonicity and functions

- (shift f x) n = f n x

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

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

Infix "<o>" := shift (at level 30, no associativity) : O_scope.

Lemma shift_simpl : ∀ {A} `{o1:ord Oa} `{o2:ord Ob} (f:Oa -m> (A → Ob)) x y,
       (f <o> x) y = f y x.

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

Add Parametric Morphism {A} `{o1:ord Oa} `{o2:ord Ob}
     : (shift (A:=A) (Oa:=Oa) (Ob:=Ob)) with signature Oeq ⟹eq ⟹Oeq
as shift_eq_compat.
Save.

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

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

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

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

Add Parametric Morphism {A} `{o1:ord Oa} `{o2:ord Ob}
    : (ishift (A:=A) (Oa:=Oa) (Ob:=Ob)) with signature Oeq ⟹eq ⟹Oeq
as ishift_eq_compat.
Save.

Instance shift_fun_mon `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} (f:Oa -m> (Ob → Oc))
      {m:∀ x, monotonic (f x)} : monotonic (shift f).
Save.

Instance shift_mon2 `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} (f:Oa -m> Ob -m> Oc)
      : monotonic2 (fun x y ⇒ f y x).
Save.
Hint Resolve @shift_mon_fun @shift_fun_mon @shift_mon2.

Definition mshift `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} (f:Oa -m> Ob -m> Oc)
    : Ob -m> Oa -m> Oc := 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 Oa} (c:Oa) : A → Oa := fun x ⇒ c.

Instance mon_cte : ∀ `{o1:ord Oa} `{o2:ord Ob} (c:Ob), monotonic (cte Oa c).
Save.

Definition mseq_cte {O} {o:ord O} (c:O) : nat -m> O := mon (cte nat c).

Add Parametric Morphism `{o1:ord Oa} `{o2:ord Ob} : (@cte Oa Ob _)
  with signature Ole ⟹Ole as cte_le_compat.
Save.

Add Parametric Morphism `{o1:ord Oa} `{o2:ord Ob} : (@cte Oa Ob _)
  with signature Oeq ⟹Oeq as cte_eq_compat.
Save.

Instance mon_diag `{o1:ord Oa} `{o2:ord Ob}(f:Oa -m> (Oa -m> Ob))
      : monotonic (fun x ⇒ f x x).
Save.
Hint Resolve @mon_diag.

Definition diag `{o1:ord Oa} `{o2:ord Ob}(f:Oa -m> (Oa -m> Ob)) : Oa-m> Ob
      := mon (fun x ⇒ f x x).

Lemma fmon_diag_simpl : ∀ `{o1:ord Oa} `{o2:ord Ob} (f:Oa -m> (Oa -m> Ob)) (x:Oa),
                diag f x = f x x.

Lemma diag_le_compat : ∀ `{o1:ord Oa} `{o2:ord Ob} (f g:Oa -m> (Oa -m> Ob)),
                f ≤ g → diag f ≤ diag g.
```

```
Hint Resolve @diag_le_compat.

Add Parametric Morphism `{o1:ord Oa} `{o2:ord Ob} : (diag (Oa:=Oa) (Ob:=Ob))
   with signature Oeq ⟹Oeq as diag_eq_compat.
Save.

Lemma diag_shift : ∀ `{o1:ord Oa} `{o2:ord Ob} (f: Oa -m> Oa -m> Ob),
                        diag f ≡ diag (mshift f).

Hint Resolve @diag_shift.

Lemma mshift_simpl : ∀ `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc}
      (h:Oa -m> Ob -m> Oc) (x : Ob) (y:Oa), mshift h x y = h y x.

Lemma mshift_le_compat : ∀ `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc}
      (f g:Oa -m> Ob -m> Oc), f ≤ g → mshift f ≤ mshift g.
Hint Resolve @mshift_le_compat.

Add Parametric Morphism `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} : (@mshift Oa _ Ob _ Oc _)
     with signature Oeq ⟹Oeq as mshift_eq_compat.
Save.

Lemma mshift2_eq : ∀ `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} (h : Oa -m> Ob -m> Oc),
             mshift (mshift h) ≡ h.
```

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

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

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

Definition comp `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} (f:Ob -m> Oc) (g:Oa -m> Ob)
        : Oa -m> Oc := mon (fun x ⇒ f (g x)).

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

Lemma comp_simpl : ∀ `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc}
      (f:Ob -m> Oc) (g:Oa -m> Ob) (x:Oa), (f@g) x = f (g x).

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

Hint Immediate @comp_le_compat.

Add Parametric Morphism `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} : (@comp Oa _ Ob _ Oc _)
    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 Oa} `{o2:ord Ob} `{o3:ord Oc} `{o4:ord Od}
      (f:Ob → Oc → Od) (g:Oa → Ob) (h:Oa → Oc)
      {mf:monotonic2 f}{mg:monotonic g} {mh:monotonic h}
      : monotonic (fun x ⇒ f (g x) (h x)).
```

```
Save.

Instance mon_app2_mon `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} `{o4:ord Od}
        (f:Ob -m> Oc -m> Od) (g:Oa -m> Ob) (h:Oa -m> Oc)
        : monotonic (fun x ⇒ f (g x) (h x)).
Save.

Definition app2 `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} `{o4:ord Od}
          (f:Ob -m> Oc -m> Od) (g:Oa -m> Ob) (h:Oa -m> Oc) : Oa -m> Od
          := mon (fun x ⇒ f (g x) (h x)).

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

Add Parametric Morphism `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} `{o4:ord Od}:
          (@app2 Oa _ Ob _ Oc _ Od _)
     with signature Ole ++> Ole ++> Ole ++> Ole
     as app2_le_compat.
Save.

Hint Immediate @app2_le_compat.

Add Parametric Morphism `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} `{o4:ord Od}:
          (@app2 Oa _ Ob _ Oc _ Od _)
     with signature Oeq ==> Oeq ==> Oeq ==> Oeq
     as app2_eq_compat.
Save.

Hint Immediate @app2_eq_compat.

Lemma app2_simpl :
     ∀ `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} `{o4:ord Od}
              (f:Ob -m> Oc -m> Od) (g:Oa -m> Ob) (h:Oa -m> Oc) (x:Oa),
     (f@2 g) h x = f (g x) (h x).

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

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

Instance comp_monotonic2 : ∀ `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc},
              monotonic2 (@comp Oa _ Ob _ Oc _).
Save.
Hint Resolve @comp_monotonic2.

Definition fcomp `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} :
    (Ob -m> Oc) -m> (Oa -m> Ob) -m> (Oa -m> Oc) := mon2 (@comp Oa _ Ob _ Oc _).

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

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

Definition fcomp2 `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} `{o4:ord Od} :
          (Oc -m> Od) -m> (Oa -m> Ob -m> Oc) -m> (Oa -m> Ob -m> Od):=
          (fcomp Oa (Ob -m> Oc) (Ob -m> Od))@(fcomp Ob Oc Od).

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

Lemma fcomp2_simpl : ∀ `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc} `{o4:ord Od}
        (f:Oc -m> Od) (g:Oa -m> Ob -m> Oc) (x:Oa)(y:Ob), fcomp2 _ _ _ _ f g x y = f (g x y).

Lemma fmon_le_compat2 : ∀ `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc}
```

```
        (f: Oa -m> Ob -m> Oc) (x y:Oa) (z t:Ob), x≤y → z ≤t → f x z ≤ f y t.
Hint Resolve fmon_le_compat2.

Lemma fmon_cte_comp : ∀ `{o1:ord Oa} `{o2:ord Ob} `{o3:ord Oc}
        (c:Oc)(f:Oa -m> Ob), (mon (cte Ob c)) @ f ≡ mon (cte Oa c).
```

2.5 Abstract relational notion of lubs

```
Record islub O (o:ord O) I (f:I → O) (x:O) : Prop := mk_islub
      { le_islub : ∀ i, f i ≤ x;
        islub_le : ∀ y, (∀ i, f i ≤ y) → x ≤ y}.
Implicit Arguments islub [O o I].
Implicit Arguments le_islub [O o I f x].
Implicit Arguments islub_le [O o I f x].

Definition isglb O (o:ord O) I (f:I → O) (x:O) : Prop
      := islub (o:=Iord O) f x.
Implicit Arguments isglb [O o I].

Lemma le_isglb O (o:ord O) I (f:I → O) (x:O) :
          isglb f x → ∀ i, x ≤ f i.

Lemma isglb_le O (o:ord O) I (f:I → O) (x:O) :
          isglb f x → ∀ y, (∀ i, y ≤ f i) → y ≤ x.
Implicit Arguments le_isglb [O o I f x].
Implicit Arguments isglb_le [O o I f x].

Lemma mk_isglb O (o:ord O) I (f:I → O) (x:O) :
        (∀ i, x ≤ f i) → (∀ y, (∀ i, y ≤ f i) → y ≤ x)
        → isglb f x.

Lemma islub_eq_compat O (o:ord O) I (f g:I → O) (x y:O):
        f≡g → x ≡ y → islub f x → islub g y.

Lemma isglb_eq_compat O (o:ord O) I (f g:I → O) (x y:O):
        f≡g → x ≡ y → isglb f x → isglb g y.

Add Parametric Morphism {O} {o:ord O} I : (@islub _ o I)
with signature Oeq ==>Oeq ==>iff
as islub_morphism.
Save.

Add Parametric Morphism {O} {o:ord O} I : (@isglb _ o I)
with signature Oeq ==>Oeq ==>iff
as isglb_morphism.
Save.
```

2.6 Basic operators of omega-cpos

- Constant : 0
    - lub : limit of monotonic sequences

2.6.1 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 : ∀ (f : nat -m> D) (x:D), (∀ n, f n ≤ x) → lub f ≤ 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.
```

2.6.2 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.

Lemma lub_seq_eq : ∀ `{c:cpo D} (f:nat → D) (g: nat-m> D) (H:f ≡ g),
    lub g ≡ lub (mon_fun_subst f g H).
```

- (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 <o> 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.
```

2.6.3 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 = lub (h <o> x).

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

2.7 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:∀ 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:∀ 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 : ∀ `{c:cpo D} (h : nat -m> nat -m> D),
        lub (lub h) ≡ lub (lub (mshift h)).
Hint Resolve @double_lub_shift.
```

2.8 Continuity

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

```
Lemma lub_app2_le : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
        (F:D1 -m> D2 -m> D3) (f : nat -m> D1) (g: nat -m> D2),
        lub ((F @² f) g) ≤ F (lub f) (lub g).
Hint Resolve @lub_app2_le.
```

```
Class continuous `{c1:cpo D1} `{c2:cpo D2} (f:D1 -m> D2) :=
    cont_intro : ∀ (h : nat -m> D1), f (lub h) ≤ lub (f @ h).
```

```
Typeclasses Opaque continuous.
```

```
Lemma continuous_eq_compat : ∀ `{c1:cpo D1} `{c2:cpo D2}(f g:D1 -m> D2),
                f ≡ g → continuous f → continuous g.
```

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

```
Lemma lub_comp_eq :
    ∀ `{c1:cpo D1} `{c2:cpo D2} (f:D1 -m> D2) (h : nat -m> D1),
            continuous f → f (lub h) ≡ lub (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 Oa} `{o2:ord Ob} `{o3:ord Oc} `{o4: ord Od}
      (f:Oa -m> Oc -m> Od) (g:Ob -m> Oc)
        : Ob -m> (Oa -m> Od) := mon ((mshift f) @ g).
```

2.8.1 Continuity

```
Class continuous2 `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} (F:D1 -m> D2 -m> D3) :=
continuous2_intro : ∀ (f : nat -m> D1) (g :nat -m> D2),
                F (lub f) (lub g) ≤ lub ((F @² f) g).
```

```
Lemma continuous2_app : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
            (F : D1 -m> D2 -m> D3) {cF:continuous2 F} (k:D1), continuous (F k).
```

```
Typeclasses Opaque continuous2.
```

```
Lemma continuous2_eq_compat :
```

```
    ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} (f g : D1 -m> D2 -m> D3),
    f ≡ g → continuous2 f → continuous2 g.
Lemma continuous2_continuous : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
          (F : D1 -m> D2 -m> D3), continuous2 F → continuous F.
Hint Immediate @continuous2_continuous.
Lemma continuous2_left : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
          (F : D1 -m> D2 -m> D3) (h:nat -m> D1) (x:D2),
          continuous F → F (lub h) x ≤ lub (mshift (F @ h) x).
Lemma continuous2_right : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
          (F : D1 -m> D2 -m> D3) (x:D1)(h:nat -m> D2),
          continuous2 F → F x (lub h) ≤ lub (F x @ h).
Lemma continuous_continuous2 : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
      (F : D1 -m> D2 -m> D3) (cFr: ∀ k:D1, continuous (F k)) (cF: continuous F),
      continuous2 F.
Hint Resolve @continuous2_app @continuous2_continuous @continuous_continuous2.
Lemma lub_app2_eq : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
      (F : D1 -m> D2 -m> D3) {cFr:∀ k:D1, continuous (F k)} {cF : continuous F},
      ∀ (f:nat -m> D1) (g:nat -m> D2),
      F (lub f) (lub g) ≡ lub ((F@2 f) g).
Lemma lub_cont2_app2_eq : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
      (F : D1 -m> D2 -m> D3){cF : continuous2 F},
      ∀ (f:nat -m> D1) (g:nat -m> D2),
      F (lub f) (lub g) ≡ lub ((F@2 f) g).
Lemma mshift_continuous2 : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
          (F : D1 -m> D2 -m> D3), continuous2 F → continuous2 (mshift F).
Hint Resolve @mshift_continuous2.
Lemma monotonic_sym : ∀ `{o1:ord D1} `{o2:ord D2} (F : D1 → D1 → D2),
      (∀ x y, F x y ≡ F y x) → (∀ k:D1, monotonic (F k)) → monotonic F.
Hint Immediate @monotonic_sym.
Lemma monotonic2_sym : ∀ `{o1:ord D1} `{o2:ord D2} (F : D1 → D1 → D2),
      (∀ x y, F x y ≡ F y x) → (∀ k:D1, monotonic (F k)) → monotonic2 F.
Hint Immediate @monotonic2_sym.
Lemma continuous_sym : ∀ `{c1:cpo D1} `{c2:cpo D2} (F : D1 -m> D1 -m> D2),
      (∀ x y, F x y ≡ F y x) → (∀ k:D1, continuous (F k)) → continuous F.
Lemma continuous2_sym : ∀ `{c1:cpo D1} `{c2:cpo D2} (F : D1 -m>D1 -m>D2),
      (∀ x y, F x y ≡ F y x) → (∀ k, continuous (F k)) → continuous2 F.
Hint Resolve @continuous2_sym.
```

- continuity is preserved by composition

```
Lemma continuous_comp : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3}
   (f:D2 -m> D3)(g:D1 -m> D2), continuous f → continuous g → continuous (mon (f@g)).
Hint Resolve @continuous_comp.
Lemma continuous2_comp : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} `{c4:cpo D4}
  (f:D1 -m> D2)(g:D2 -m> D3 -m> D4),
  continuous f → continuous2 g → continuous2 (g @ f).
Hint Resolve @continuous2_comp.
Lemma continuous2_comp2 : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} `{c4:cpo D4}
    (f:D3 -m> D4)(g:D1 -m> D2 -m> D3),
    continuous f → continuous2 g → continuous2 (fcomp2 D1 D2 D3 D4 f g).
```

```
Hint Resolve @continuous2_comp2.

Lemma continuous2_app2 : ∀ `{c1:cpo D1} `{c2:cpo D2} `{c3:cpo D3} `{c4:cpo D4}
    (F : D1 -m> D2 -m> D3) (f:D4 -m> D1)(g:D4 -m> D2), continuous2 F →
    continuous f → continuous g → continuous ((F @2 f) g).
Hint Resolve @continuous2_app2.
```

2.9 Cpo of continuous functions

```
Instance lub_continuous `{c1:cpo D1} `{c2:cpo D2}
     (f:nat -m> (D1 -m> D2)) {cf:∀ n, continuous (f n)}
     : continuous (lub f).
Save.

Record fcont `{c1:cpo D1} `{c2:cpo D2}: Type
     := cont {fcontm :> D1 -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) -m> (D1 -m> D2) :=
     mon (fcontm (D1:=D1) (D2:=D2)).

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

Definition fcont_lub `{c1:cpo D1} `{c2:cpo D2} : (nat -m> (D1 -c> D2)) → D1 -c> D2 :=
     fun f ⇒ cont (lub (D:=D1 -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:∀ k, continuous (f k)),
                monotonic (Ob:=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:∀ 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:∀ 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 Oa}, monotonic (fun x:Oa ⇒ x).
Save.

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

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

2.10 Fixpoints

```
Fixpoint iter_ {D} {o} `{c: @cpo D o} (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*).

Definition *Fixp_cont D* ‘{*c*:*cpo D*} : (*D* -*c*> *D*) -*m*> *D* := *Fixp D* @ (*Fcontm D D*).

Lemma *Fixp_cont_simpl* : ∀ ‘{*c*:*cpo D*} (*f*:*D* -*c*> *D*), *Fixp_cont D f* = *fixp* (*fcontm f*).

Instance *Fixp_cont_continuous* : ∀ *D* ‘{*c*:*cpo D*}, *continuous* (*Fixp_cont D*).
Save.

Definition *FIXP D* ‘{*c*:*cpo D*} : (*D* -*c*> *D*) -*c*> *D* := *cont* (*Fixp_cont D*).

Lemma *FIXP_simpl* : ∀ ‘{*c*:*cpo D*} (*f*:*D* -*c*> *D*), *FIXP D f* = *Fixp D* (*fcontm f*).

Lemma *FIXP_le_compat* : ∀ ‘{*c*:*cpo D*} (*f g* : *D* -*c*> *D*),
f ≤ *g* → *FIXP D f* ≤ *FIXP D g*.
Hint Resolve @*FIXP_le_compat*.

Lemma *FIXP_eq_compat* : ∀ ‘{*c*:*cpo D*} (*f g* : *D* -*c*> *D*),
f ≡ *g* → *FIXP D f* ≡ *FIXP D g*.
Hint Resolve @*FIXP_eq_compat*.

Lemma *FIXP_eq* : ∀ ‘{*c*:*cpo D*} (*f*:*D* -*c*> *D*), *FIXP D f* ≡ *f* (*FIXP D f*).
Hint Resolve @*FIXP_eq*.

Lemma *FIXP_inv* : ∀ ‘{*c*:*cpo D*} (*f*:*D* -*c*> *D*) (*g* : *D*), *f g* ≤ *g* → *FIXP D f* ≤ *g*.

2.10.1 Iteration of functional

Lemma *FIXP_comp_com* : ∀ ‘{*c*:*cpo D*} (*f g*:*D*-*c*>*D*),
g @_ *f* ≤ *f* @_ *g*→ *FIXP D g* ≤ *f* (*FIXP D g*).

Lemma *FIXP_comp* : ∀ ‘{*c*:*cpo D*} (*f g*:*D*-*c*>*D*),
g @_ *f* ≤ *f* @_ *g* → *f* (*FIXP D g*) ≤ *FIXP D g* → *FIXP D* (*f* @_ *g*) ≡ *FIXP D g*.

Fixpoint *fcont_compn* {*D*} {*o*} ‘{*c*:@*cpo D o*}(*f*:*D* -*c*> *D*) (*n*:*nat*) {struct *n*} : *D* -*c*> *D* :=
match *n* with *O* ⇒ *f* | *S p* ⇒ *fcont_compn f p* @_ *f* end.

Lemma *fcont_compn_Sn_simpl* :
∀ ‘{*c*:*cpo D*}(*f*:*D* -*c*> *D*) (*n*:*nat*), *fcont_compn f* (*S n*) = *fcont_compn f n* @_ *f*.

Lemma *fcont_compn_com* : ∀ ‘{*c*:*cpo D*}(*f*:*D*-*c*>*D*) (*n*:*nat*),
f @_ (*fcont_compn f n*) ≤ *fcont_compn f n* @_ *f*.

Lemma *FIXP_compn* :

$\forall$ ‘{c:cpo D} (f:D-c>D) (n:nat), FIXP D (fcont_compn f n) $\equiv$ FIXP D f.

Lemma *fixp_double* : $\forall$ ‘{c:cpo D} (f:D-c>D), FIXP D (f @_ f) $\equiv$ FIXP D f.

2.10.2 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).
Definition admissible2 ‘{c1:cpo D1}‘{c2:cpo D2}(R:D1 → D2 → Type) :=
    ∀ (f : nat -m> D1) (g:nat -m> D2), (∀ n, R (f n) (g n)) → R (lub f) (lub g).
Lemma fixp_ind_rel : ∀ ‘{c1:cpo D1}‘{c2:cpo D2}(F:D1 -m> D1) (G:D2-m> D2)
      (R:D1 → D2 → Type),
      admissible2 R → R 0 0 → (∀ x y, R x y → R (F x) (G y)) → R (fixp F) (fixp G).
Lemma lub_le_fixp : ∀ ‘{c1:cpo D1}‘{c2:cpo D2} (f:D1-m>D2) (F:D1 -m> D1)
                                                   (s:nat-m> D2),
        s O ≤ f 0 → (∀ x n, s n ≤ f x → s (S n) ≤ f (F x))
        → lub s ≤ f (fixp F).
Lemma fixp_le_lub : ∀ ‘{c1:cpo D1}‘{c2:cpo D2} (f:D1-m>D2) (F:D1 -m> D1)
                                                   (s:nat-m> D2) {fc:continuous f},
        f 0 ≤ s O → (∀ x n, f x ≤ s n → f (F x) ≤ s (S n)) → f (fixp F) ≤ lub s.

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.
```

2.10.3 Function for conditionnal choice defined as a morphism

```
Definition fif {A} (b:bool) : A → A → A := fun e1 e2 ⇒ if b then e1 else e2.
Instance fif_mon2 ‘{o:ord A} (b:bool) : monotonic2 (@fif _ b).
Save.
Definition Fif ‘{o:ord A} (b:bool) : A -m> A -m> A := mon2 (@fif _ b).
Lemma Fif_simpl : ∀ ‘{o:ord A} (b:bool) (x y:A), Fif b x y = fif b x y.
Lemma Fif_continuous_right ‘{c:cpo A} (b:bool) (e:A) : continuous (Fif b e).
Lemma Fif_continuous_left ‘{c:cpo A} (b:bool) : continuous (Fif (A:=A) b).
Hint Resolve @Fif_continuous_right @Fif_continuous_left.
Lemma fif_continuous_left ‘{c:cpo A} (b:bool) (f:nat-m> A):
    fif b (lub f) ≡ lub (Fif b@f).
Lemma fif_continuous_right ‘{c:cpo A} (b:bool) e (f:nat-m> A):
```

fif b e (lub f) ≡ lub (Fif b e@f).

Hint Resolve @*fif_continuous_right* @*fif_continuous_left*.

Instance *Fif_continuous2* ‘{*c*:*cpo A*} (*b*:*bool*) : *continuous2* (*Fif* (*A*:=*A*) *b*).
Save.

Lemma *fif_continuous2* ‘{*c*:*cpo A*} (*b*:*bool*) (*f g* : *nat-m*> *A*):
fif b (lub f) (lub g) ≡ lub ((Fif b@2 f) g).

Add *Parametric Morphism* ‘{*o*:*ord A*} (*b*:*bool*) : (@*fif A b*)
with *signature Ole* ⟹ *Ole* ⟹ *Ole*
as *fif_le_compat*.
Save.

Add *Parametric Morphism* ‘{*o*:*ord A*} (*b*:*bool*) : (@*fif A b*)
with *signature Oeq* ⟹ *Oeq* ⟹ *Oeq*
as *fif_eq_compat*.
Save.

3 Utheory.v: Specification of *U*, interval [0,1]

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

3.1 Basic operators of U

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

Module Type *Universe*.
Parameter *U* : Type.
Declare Instance ordU: *ord U*.
Declare Instance cpoU: *cpo U*.
Delimit Scope *U_scope* with *U*.

Parameter*s 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*.

3.2 Basic Properties

`Hypothesis` *Udiff_0_1* : $\tilde{}0 \equiv 1$.

`Hypothesis` *Uplus_sym* : $\forall\ x\ y{:}U,\ x + y \equiv y + x$.
`Hypothesis` *Uplus_assoc* : $\forall\ x\ y\ z{:}U,\ x + (y + z) \equiv x + y + z$.
`Hypothesis` *Uplus_zero_left* : $\forall\ x{:}U,\ 0 + x \equiv x$.

`Hypothesis` *Umult_sym* : $\forall\ x\ y{:}U,\ x \times y \equiv y \times x$.
`Hypothesis` *Umult_assoc* : $\forall\ x\ y\ z{:}U,\ x \times (y \times z) \equiv x \times y \times z$.
`Hypothesis` *Umult_one_left* : $\forall\ x{:}U,\ 1 \times x \equiv x$.

`Hypothesis` *Uinv_one* : [1-] $1 \equiv 0$.

`Hypothesis` *Umult_div* : $\forall\ x\ y,\ \neg\ 0 \equiv y \to x \le y \to y \times (x/y) \equiv x$.
`Hypothesis` *Udiv_le_one* : $\forall\ x\ y,\ \neg\ 0 \equiv y \to y \le x \to (x/y) \equiv 1$.
`Hypothesis` *Udiv_by_zero* : $\forall\ x\ y,\ 0 \equiv y \to (x/y) \equiv 0$.

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

`Hypothesis` *Uinv_plus_left* : $\forall\ x\ y,\ y \le$ [1-] $x \to$ [1-] $(x + y) + x \equiv$ [1-] y.

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

`Hypothesis` *Udistr_plus_right* : $\forall\ x\ y\ z,\ x \le$ [1-] $y \to (x + y) \times z \equiv x \times z + y \times z$.

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

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

- Totality of the order

`Hypothesis` *Ule_class* : $\forall\ x\ y\ :\ U$, *class* $(x \le y)$.
`Hypothesis` *Ule_total* : $\forall\ x\ y\ :\ U$, *orc* $(x \le y)\ (y \le x)$.
`Implicit Arguments` *Ule_total* [].

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

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

Declare Instance Umult_mon_right : $\forall$ *x*, *monotonic* (*Umult* *x*).
`Hypothesis` *Uinv_le_compat* : $\forall\ x\ y{:}U,\ x \le y \to$ [1-] $y \le$ [1-] x.

- Properties of simplification in case there is no overflow

`Hypothesis` *Uplus_le_simpl_right* : $\forall\ x\ y\ z,\ z \le$ [1-] $x \to x + z \le y + z \to x \le y$.
`Hypothesis` *Umult_le_simpl_left* : $\forall\ x\ y\ z{:}\ U,\ \neg\ 0 \equiv z \to z \times x \le z \times y \to x \le y$.

- Property of *Unth*: $1\ /\ n{+}1 \equiv 1 - n \times (1/n{+}1)$

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

- Archimedian property

`Hypothesis` *archimedian* : $\forall\ x,\ \tilde{}0 \equiv x \to$ *exc* (`fun` $n \Rightarrow$ [1/]1+$n \le x$).

- Stability properties of lubs with respect to + and ×

Hypothesis *Uplus_right_continuous* : ∀ *k*, *continuous* (*mon* (*Uplus* *k*)).
Hypothesis *Umult_right_continuous* : ∀ *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*.

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

Add *Rec LoadPath* "." as *ALEA*.

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

Open Local Scope *U_scope*.

Notation "[1/] n" := (*Unth* (*pred* *n*)) (at *level* 35, *right associativity*).

4.1 Direct consequences of axioms

Lemma *Uplus_le_compat_right* : ∀ *x* *y* *z*:*U*, $y \le z \to x + y \le x + z$.
Hint Resolve *Uplus_le_compat_right*.

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

Lemma *Uplus_le_compat_left* : ∀ *x* *y* *z*:*U*, $x \le y \to x + z \le y + z$.
Hint Resolve *Uplus_le_compat_left*.

Lemma *Uplus_le_compat* : ∀ *x* *y* *z* *t*, $x \le y \to z \le t \to x + z \le y + t$.
Hint Immediate *Uplus_le_compat*.

Lemma *Uplus_eq_compat_left* : ∀ *x* *y* *z*:*U*, $x \equiv y \to x + z \equiv y + z$.
Hint Resolve *Uplus_eq_compat_left*.

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

Hint Resolve *Uplus_eq_compat_left* *Uplus_eq_compat_right*.

Add *Morphism* *Uplus* with *signature* *Oeq* ⟹ *Oeq* ⟹ *Oeq* as *Uplus_eq_compat*.
Qed.
Hint Immediate *Uplus_eq_compat*.

Add *Morphism* *Uinv* with *signature* *Oeq* ⟹ *Oeq* as *Uinv_eq_compat*.
Qed.
Hint Resolve *Uinv_eq_compat*.

Lemma *Uplus_zero_right* : ∀ *x*:*U*, $x + 0 \equiv x$.
Hint Resolve *Uplus_zero_right*.

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

Lemma *Uinv_opp_right* : ∀ *x*, $x +$ [1-] $x \equiv 1$.

Hint Resolve *Uinv_opp_right*.

Lemma *Uinv_inv* : $\forall\ x :\ U$, [1-] [1-] $x \equiv x$.
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* $(x{\equiv}y)$.

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

Lemma *Ule_orc* : $\forall\ x\ y :\ U$, *orc* $(x{\leq}y)$ (˜ $x{\leq}y$).
Implicit Arguments *Ule_orc* [].

Lemma *Ueq_orc* : $\forall\ x\ y{:}U$, *orc* $(x{\equiv}y)$ (˜ $x{\equiv}y$).
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*.

4.2 Properties of $\equiv$ derived from properties of $\leq$

Definition *UPlus* : *U -m> U -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 *Uplus_lub_eq* : $\forall\ f\ g$: *nat -m> U*,
 lub f + *lub* $g \equiv$ *lub* ((*UPlus* @^{2} f) g).

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

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

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

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

Definition *UMult* : *U -m> U -m> U* := *mon2 Umult*.

Lemma *Umult_eq_compat_left* : $\forall\ x\ y\ z{:}U,\ x \equiv y \rightarrow (x \times z) \equiv (y \times z)$.
Hint Resolve *Umult_eq_compat_left*.

Lemma *Umult_eq_compat_right* : $\forall\ x\ y\ z{:}U,\ x \equiv y \rightarrow (z \times x) \equiv (z \times y)$.

Hint Resolve *Umult_eq_compat_left Umult_eq_compat_right*.

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

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

Lemma *Umult_lub_eq* : $\forall$ *f g* : *nat -m> U*,
    lub f $\times$ *lub g* $\equiv$ *lub* ((*UMult* @^{2} *f*) *g*).

Lemma *Umultk_lub_eq* : $\forall$ (*k*:*U*) (*f* : *nat -m> U*),
    k $\times$ *lub f* $\equiv$ *lub* (*UMult k* @ *f*).

4.3 *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* –*m*> *U* := *mon* (*o1*:=*Iord U*) *Uinv.*

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

Lemma *Ule_eq_compat* :
$\forall$ *x1 x2* : *U*, *x1* $\equiv$ *x2* $\rightarrow$ $\forall$ *x3 x4* : *U*, *x3* $\equiv$ *x4* $\rightarrow$ *x1* $\leq$ *x3* $\rightarrow$ *x2* $\leq$ *x4*.

4.4 Properties of $x < y$ on U

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

Lemma *Ult_notle_equiv* : $\forall$ *x y*:*U*, *x* < *y* $\leftrightarrow$ $\neg$ (*y* $\leq$ *x*).

Lemma *notUle_lt* : $\forall$ *x y*:*U*, $\neg$ (*y* $\leq$ *x*) $\rightarrow$ *x* < *y*.

Hint Immediate *notUle_lt.*

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

Hint Immediate *notUlt_le.*

4.4.1 Properties of $x \leq y$

Lemma *notUle_le* : $\forall$ *x y*:*U*, $\neg$ (*y* $\leq$ *x*) $\rightarrow$ *x* $\leq$ *y*.
Hint Immediate *notUle_le.*

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

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

Hint Immediate *Ule_zero_eq Uge_one_eq.*

4.4.2 Properties of $x < y$

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

Lemma *Ult_neq_one* : $\forall$ *x*, $\neg$ 1 $\equiv$ *x* $\rightarrow$ *x* < 1.

Hint Resolve *Ule_total Ult_neq_zero Ult_neq_one.*

Lemma *not_Ult_eq_zero* : $\forall$ *x*, $\neg$ 0 < *x* $\rightarrow$ 0 $\equiv$ *x*.

Lemma *not_Ult_eq_one* : $\forall$ *x*, $\neg$ *x* < 1 $\rightarrow$ 1 $\equiv$ *x*.

Hint Immediate *not_Ult_eq_zero not_Ult_eq_one.*

Lemma *Ule_lt_orc_eq* : $\forall$ *x y*, *x* $\leq$ *y* $\rightarrow$ *orc* (*x* < *y*) (*x* $\equiv$ *y*).
Hint Resolve *Ule_lt_orc_eq.*

Lemma *Udiff_lt_orc* : $\forall$ x y, $\neg$ $x \equiv y \rightarrow$ *orc* $(x < y)$ $(y < x)$.
Hint Resolve *Udiff_lt_orc*.

Lemma *Uplus_pos_elim* : $\forall$ x y,
$0 < x + y \rightarrow$ *orc* $(0 < x)$ $(0 < y)$.

4.5 Properties of + and ×

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

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

Hint Resolve *Uinv_eq_compat* *Udistr_plus_left* *Udistr_inv_left*.

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

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

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

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

Hint Resolve *Uplus_perm2* *Umult_perm2* *Uplus_perm3* *Umult_perm3*.

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

Hint Immediate *Uinv_simpl*.

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

4.6 More properties on + and × and *Uinv*

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

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

Lemma *Umult_one_left_eq* : $\forall$ x y:U, $x \equiv 1 \rightarrow x \times y \equiv y$.
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-] $y \rightarrow (x + z) \equiv (y + z) \rightarrow x \equiv y$.

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, $x \leq y \rightarrow x \leq$ [1-][1-]y.
Hint Resolve *Uinv_double_le_simpl_right*.

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

Hint Resolve $Uinv\_double\_le\_simpl\_left$.

Lemma $Uinv\_eq\_perm\_left$: $\forall\ x\ y{:}U,\ x \equiv [1\text{-}]\ y \rightarrow [1\text{-}]\ x \equiv y$.
Hint Immediate $Uinv\_eq\_perm\_left$.

Lemma $Uinv\_eq\_perm\_right$: $\forall\ x\ y{:}U,\ [1\text{-}]\ x \equiv y \rightarrow x \equiv [1\text{-}]\ y$.

Hint Immediate $Uinv\_eq\_perm\_right$.

Lemma $Uinv\_eq$: $\forall\ x\ y{:}U,\ x \equiv [1\text{-}]\ y \leftrightarrow [1\text{-}]\ x \equiv y$.
Hint Resolve $Uinv\_eq$.

Lemma $Uinv\_eq\_simpl$: $\forall\ x\ y{:}U,\ [1\text{-}]\ x \equiv [1\text{-}]\ y \rightarrow x \equiv y$.
Hint Immediate $Uinv\_eq\_simpl$.

Lemma $Uinv\_double\_eq\_simpl\_right$: $\forall\ x\ y,\ x{\equiv}y \rightarrow x \equiv [1\text{-}][1\text{-}]y$.
Hint Resolve $Uinv\_double\_eq\_simpl\_right$.

Lemma $Uinv\_double\_eq\_simpl\_left$: $\forall\ x\ y,\ x{\equiv}y \rightarrow [1\text{-}][1\text{-}]x \equiv y$.
Hint Resolve $Uinv\_double\_eq\_simpl\_left$.

Lemma $Uinv\_plus\_right$: $\forall\ x\ y,\ y \leq [1\text{-}]\ x \rightarrow [1\text{-}]\ (x + y) + y \equiv [1\text{-}]\ x$.
Hint Resolve $Uinv\_plus\_right$.

Lemma $Uplus\_eq\_simpl\_left$:
$\forall\ x\ y\ z{:}U,\ x \leq [1\text{-}]\ y \rightarrow x \leq [1\text{-}]\ z \rightarrow (x + y) \equiv (x + z) \rightarrow y \equiv z$.

Lemma $Uplus\_eq\_zero\_left$: $\forall\ x\ y{:}U,\ x \leq [1\text{-}]\ y \rightarrow (x + y) \equiv y \rightarrow x \equiv 0$.

Lemma $Uinv\_le\_trans$: $\forall\ x\ y\ z\ t,\ x \leq [1\text{-}]\ y \rightarrow z \leq x \rightarrow t \leq y \rightarrow z \leq [1\text{-}]\ t$.

Lemma $Uinv\_plus\_left\_le$: $\forall\ x\ y,\ [1\text{-}]y \leq [1\text{-}](x{+}y) + x$.

Lemma $Uinv\_plus\_right\_le$: $\forall\ x\ y,\ [1\text{-}]x \leq [1\text{-}](x{+}y) + y$.

Hint Resolve $Uinv\_plus\_left\_le$ $Uinv\_plus\_right\_le$.

4.7 Disequality

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

Lemma $Uinv\_neq\_compat$: $\forall\ x\ y,\ \neg\ x \equiv y \rightarrow \neg\ [1\text{-}]\ x \equiv [1\text{-}]\ y$.

Lemma $Uinv\_neq\_simpl$: $\forall\ x\ y,\ \neg\ [1\text{-}]\ x \equiv [1\text{-}]\ y \rightarrow \neg\ x \equiv y$.

Hint Resolve $Uinv\_neq\_compat$.
Hint Immediate $Uinv\_neq\_simpl$.

Lemma $Uinv\_neq\_left$: $\forall\ x\ y,\ \neg\ x \equiv [1\text{-}]\ y \rightarrow \neg\ [1\text{-}]\ x \equiv y$.

Lemma $Uinv\_neq\_right$: $\forall\ x\ y,\ \neg\ [1\text{-}]\ x \equiv y \rightarrow \neg x \equiv [1\text{-}]\ y$.

4.7.1 Properties of $<$

Lemma Ult_0_1 : $(0 < 1)$.

Hint Resolve Ult_0_1.

Lemma Ule_neq_zero : $\forall\ (x\ y{:}U),\ \neg\ 0 \equiv x \rightarrow x \leq y \rightarrow \neg\ 0 \equiv y$.

Lemma $Uplus\_neq\_zero\_left$: $\forall\ x\ y,\ \neg\ 0 \equiv x \rightarrow \neg\ 0 \equiv x{+}y$.

Lemma $Uplus\_neq\_zero\_right$: $\forall\ x\ y,\ \neg\ 0 \equiv y \rightarrow \neg\ 0 \equiv x{+}y$.

Lemma $Uplus\_le\_simpl\_left$: $\forall\ x\ y\ z\ :\ U,\ z \leq [1\text{-}]\ x \rightarrow z + x \leq z + y \rightarrow x \leq y$.

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

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

Hint Resolve $Uplus\_lt\_compat\_right$ $Uplus\_lt\_compat\_left$.

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

Hint Immediate *Uplus_lt_compat*.

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

Lemma *Ult_plus_right* : $\forall\ x\ y\ z : U,\ x < z \rightarrow x < y + z$.
Hint Immediate *Ult_plus_left Ult_plus_right*.

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

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

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

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

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

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

Lemma *Umult_zero_left_eq* : $\forall\ x\ y,\ x \equiv 0 \rightarrow x \times y \equiv 0$.

Lemma *Umult_zero_right_eq* : $\forall\ x\ y,\ y \equiv 0 \rightarrow x \times y \equiv 0$.

Lemma *Umult_zero_eq* : $\forall\ x\ y\ z,\ x \equiv 0 \rightarrow x \times y \equiv x \times z$.

4.7.2 Compatibility of operations with respect to order.

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

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

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

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

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

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

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

Hint Resolve *Umult_lt_compat_left Umult_lt_compat_right*.

Lemma *Umult_zero_simpl_right* : $\forall\ x\ y,\ 0 \equiv x \times y \rightarrow \neg\ 0 \equiv x \rightarrow 0 \equiv y$.

Lemma *Umult_zero_simpl_left* : $\forall\ x\ y,\ 0 \equiv x \times y \rightarrow \neg\ 0 \equiv y \rightarrow 0 \equiv x$.

Lemma *Umult_neq_zero* : $\forall\ x\ y,\ \neg\ 0 \equiv x \rightarrow \neg\ 0 \equiv y \rightarrow \neg\ 0 \equiv x{\times}y$.
Hint Resolve *Umult_neq_zero*.

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

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

4.7.3 More Properties

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

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

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

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

Hint Resolve *Uinv_mult_simpl*.

Lemma *Umult_inv_plus* : ∀ *x y*, *x* × [1-] *y* + *y* ≡ *x* + *y* × [1-] *x*.
Hint Resolve *Umult_inv_plus*.

Lemma *Umult_inv_plus_le* : ∀ *x y z*, *y* ≤ *z* → *x* × [1-] *y* + *y* ≤ *x* × [1-] *z* + *z*.
Hint Resolve *Umult_inv_plus_le*.

Lemma *Uplus_lt_Uinv* : ∀ *x y*, *x* + *y* < 1 → *x* ≤ [1-] *y*.

Lemma *Uinv_lt_perm_left*: ∀ *x y* : *U*, [1-] *x* < *y* → [1-] *y* < *x*.

Lemma *Uinv_lt_perm_right*: ∀ *x y* : *U*, *x* < [1-] *y* → *y* < [1-] *x*.

Lemma *Uinv_lt_compat* : ∀ *x y* : *U*, *x* < *y* → [1-] *y* < [1-] *x*.
Hint Resolve *Uinv_lt_compat*.

Lemma *Uinv_lt_simpl* : ∀ *x y* : *U*, [1-] *y* < [1-] *x* → *x* < *y*.
Hint Immediate *Uinv_lt_simpl*.

Lemma *Ult_inv_Uplus* : ∀ *x y*, *x* < [1-] *y* → *x* + *y* < 1.

Hint Immediate *Uplus_lt_Uinv Uinv_lt_perm_left Uinv_lt_perm_right Ult_inv_Uplus*.

Lemma *Uinv_lt_one* : ∀ *x*, 0 < *x* → [1-]*x* < 1.

Lemma *Uinv_lt_zero* : ∀ *x*, *x* < 1 → 0 < [1-]*x*.

Hint Resolve *Uinv_lt_one Uinv_lt_zero*.

Lemma *orc_inv_plus_one* : ∀ *x y*, *orc* (*x*<=[1-]*y*) (*x*+*y*==1).

Lemma *Umult_lt_right* : ∀ *p q*, *p* <1 → 0 < *q* → *p* × *q* < *q*.

Lemma *Umult_lt_left* : ∀ *p q*, 0 < *p* → *q* < 1 → *p* × *q* < *p*.

Hint Resolve *Umult_lt_right Umult_lt_left*.

4.8 Definition of *x* ^ *n*

Fixpoint *Uexp* (*x*:*U*) (*n*:*nat*) {struct *n*} : *U* :=
  match *n* with 0 ⇒ 1 | (*S p*) ⇒ *x* × *Uexp x p* end.

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

Lemma *Uexp_1* : ∀ *x*, *x*^1 ≡ *x*.

Lemma *Uexp_0* : ∀ *x*, *x*^0 ≡ 1.

Lemma *Uexp_zero* : ∀ *n*, (0<*n*)%*nat* → 0^*n* ≡ 0.

Lemma *Uexp_one* : ∀ *n*, 1^*n* ≡ 1.

Lemma *Uexp_le_compat_right* :
    ∀ *x n m*, (*n*≤*m*)%*nat* → *x*^*m* ≤ *x*^*n*.

Lemma *Uexp_le_compat_left* : ∀ *x y n*, *x* ≤ *y* → *x*^*n* ≤ *y*^*n*.
Hint Resolve *Uexp_le_compat_left Uexp_le_compat_right*.

Lemma *Uexp_le_compat* : ∀ *x y* (*n m*:*nat*),
      x ≤ *y* → *n* ≤ *m* → *x*^*m* ≤ *y*^*n*.

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

Definition *UExp* : *U* –*m*> (*nat* -*m*→ *U*) := *mon2 Uexp*.

Add *Morphism Uexp* with *signature Oeq* ⟹*eq* ⟹*Oeq* as *Uexp_eq_compat*.
Save.

Lemma *Uexp_inv_S* : ∀ *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*, $(0<p) \to (0<p\hat{\ }n)$.
Hint Resolve *Uexp_lt_zero*.

Lemma *Uexp_lt_one* : ∀ *p n*, $(0<n)\%nat \to p<1 \to (p\hat{\ }n<1)$.
Hint Resolve *Uexp_lt_one*.

Lemma *Uexp_lt_antimon*: ∀ *p n m*,
$(n<m)\%nat \to 0 < p \to p < 1 \to p\hat{\ }m < p\hat{\ }n$.
Hint Resolve *Uexp_lt_antimon*.

4.9 Properties of division

Lemma *Udiv_mult* : ∀ *x y*, $\neg\, 0 \equiv y \to x \leq y \to (x/y) \times y \equiv x$.
Hint Resolve *Udiv_mult*.

Lemma *Umult_div_le* : ∀ *x y*, $y \times (x\ /\ y) \leq x$.
Hint Resolve *Umult_div_le*.

Lemma *Udiv_mult_le* : ∀ *x y*, $(x/y) \times y \leq x$.
Hint Resolve *Udiv_mult_le*.

Lemma *Udiv_le_compat_left* : ∀ *x y z*, $x \leq y \to x/z \leq y/z$.
Hint Resolve *Udiv_le_compat_left*.

Lemma *Udiv_eq_compat_left* : ∀ *x y z*, $x \equiv y \to x/z \equiv y/z$.
Hint Resolve *Udiv_eq_compat_left*.

Lemma *Umult_div_le_left* : ∀ *x y z*, $\neg\, 0{=}{=}y \to x{\times}y \leq z \to x \leq z/y$.

Lemma *Udiv_le_compat_right* : ∀ *x y z*, $\neg\, 0{=}{=}y \to y \leq z \to x/z \leq x/y$.
Hint Resolve *Udiv_le_compat_right*.

Lemma *Udiv_eq_compat_right* : ∀ *x y z*, $y \equiv z \to x/z \equiv x/y$.
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*, $\neg\, 0 \equiv y \to x \times y \equiv z \to x \equiv z/y$.

Lemma *Umult_div_le_right* : ∀ *x y z*, $x \leq y \times z \to x/z \leq y$.

Lemma *Udiv_le* : ∀ *x y*, $\neg\, 0 \equiv y \to x \leq x/y$.

Lemma *Udiv_zero* : ∀ *x*, $0/x \equiv 0$.
Hint Resolve *Udiv_zero*.

Lemma *Udiv_zero_eq* : ∀ *x y*, $0 \equiv x \to x/y \equiv 0$.
Hint Resolve *Udiv_zero_eq*.

Lemma *Udiv_one* : ∀ *x*, $x/1 \equiv x$.
Hint Resolve *Udiv_one*.

Lemma *Udiv_refl* : ∀ *x*, $\neg\, 0 \equiv x \to x/x \equiv 1$.
Hint Resolve *Udiv_refl*.

Lemma *Umult_div_assoc* : ∀ *x y z*, $y \leq z \to (x \times y)\ /\ z \equiv x \times (y/z)$.

Lemma *Udiv_mult_assoc* : ∀ *x y z*, $x \leq y \times z \to x/(y \times z) \equiv (x/y)/z$.

Lemma *Udiv_inv* : ∀ *x y*, $\neg\, 0 \equiv y \to [1\text{-}](x/y) \leq ([1\text{-}]x)/y$.

Lemma *Uplus_div_inv* : ∀ *x y z*, $x{+}y \leq z \to x{<=}[1\text{-}]y \to x/z \leq [1\text{-}](y/z)$.
Hint Resolve *Uplus_div_inv*.

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

Lemma *Udiv_plus* : $\forall$ x y z, $(x+y)/z \equiv x/z + y/z$.
Hint Resolve *Udiv_plus*.

Lemma *Umult_div_simpl_r* : $\forall$ x y, $\neg$ $0 \equiv y \rightarrow (x \times y)$ / $y \equiv x$.
Hint Resolve *Umult_div_simpl_r*.

Lemma *Umult_div_simpl_l* : $\forall$ x y, $\neg$ $0 \equiv x \rightarrow (x \times y)$ / $x \equiv y$.
Hint Resolve *Umult_div_simpl_l*.

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

Definition *UDiv* (k:*U*) : *U* -m> *U* := *mon* (fun $x \Rightarrow (x/k)$).

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

4.10 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:*U*) := [1-] ([1-] x + [1-] y).

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

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

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

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

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

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

Lemma *Uesp_zero* : $\forall$ x y, $x \leq$ [1-] y $\rightarrow$ x & y $\equiv$ 0.

Hint Resolve *Uesp_sym Uesp_one_right Uesp_one_left Uesp_zero*.

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

Lemma *Uesp_zero_left* : $\forall$ x : *U*, 0 & x $\equiv$ 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, $x \leq y \rightarrow z \leq t \rightarrow x$ & $z \leq y$ & t .

Hint Immediate *Uesp_le_compat Uesp_eq_compat*.

Lemma *Uesp_assoc* : $\forall$ x y z, x & (y & z) $\equiv$ x & y & z.
Hint Resolve *Uesp_assoc*.

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

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

Hint Resolve *Uesp_zero_one_mult_left Uesp_zero_one_mult_right*.

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

Definition *UEsp* : *U* -m> *U* -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-] $y \leq x \rightarrow x \equiv x$ & y + [1-] y.

Hint Resolve *Uesp_plus_inv*.

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

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

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

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

Hint Immediate *Ult_esp_left* *Ult_esp_right*.

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

Lemma *Uesp_lt_compat_right* : $\forall$ x y z, [1-]$x \leq y \rightarrow y < z \rightarrow x$ & $y < x$ & z.
Hint Resolve *Uesp_lt_compat_left*.

4.11 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 $\equiv$ x.

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

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

Hint Resolve *Uminus_zero_right* *Uminus_one_left* *Uminus_le_zero*.

Lemma *Uminus_zero_left* : $\forall$ x, 0 - $x \equiv 0$.
Hint Resolve *Uminus_zero_left*.

Lemma *Uminus_one_right* : $\forall$ x, x - 1 $\equiv 0$.
Hint Resolve *Uminus_one_right*.

Lemma *Uminus_eq* : $\forall$ x, x - $x \equiv 0$.
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) + $y \equiv x$.

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

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) - $z \equiv$ (x - z) & y.

Lemma *Uesp_minus_distr_right* : ∀ x y z, $(x$ & $y)$ - $z \equiv x$ & $(y$ - $z)$.

Hint Resolve *Uesp_minus_distr_left* *Uesp_minus_distr_right*.

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

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

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

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

Hint Resolve *Uminus_esp_le_inv*.

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

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

4.12 Definition and properties of max

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

Lemma *max_eq_right* : ∀ x y : U, $y \leq x \rightarrow max\ x\ y \equiv x$.

Lemma *max_eq_left* : ∀ x y : U, $x \leq y \rightarrow max\ x\ y \equiv y$.

Hint Resolve *max_eq_right* *max_eq_left*.

Lemma *max_eq_case* : ∀ x y : U, *orc* $(max\ x\ y \equiv x)$ $(max\ x\ y \equiv y)$.

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

Lemma *max_le_right* : ∀ x y : U, $x \leq max\ x\ y$.

Lemma *max_le_left* : ∀ x y : U, $y \leq max\ x\ y$.

Hint Resolve *max_le_right* *max_le_left*.

Lemma *max_le* : ∀ x y z : U, $x \leq z \rightarrow y \leq z \rightarrow max\ x\ y \leq z$.

Lemma *max_le_compat* : ∀ 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*.

Lemma *max_idem* : ∀ x, $max\ x\ x \equiv x$.
Hint Resolve *max_idem*.

Lemma *max_sym_le* : ∀ x y, $max\ x\ y \leq max\ y\ x$.
Hint Resolve *max_sym_le*.

Lemma *max_sym* : ∀ x y, $max\ x\ y \equiv max\ y\ x$.
Hint Resolve *max_sym*.

Lemma *max_assoc* : ∀ x y z, $max\ x\ (max\ y\ z) \equiv max\ (max\ x\ y)\ z$.
Hint Resolve *max_assoc*.

Lemma *max_0* : ∀ x, $max\ 0\ x \equiv x$.
Hint Resolve *max_0*.

Instance *max_mon* : *monotonic2 max*.
Save.

Definition *Max* : U -m> U -m> U := *mon2 max*.

Lemma *max_eq_mult* : ∀ k x y, $max\ (k \times x)\ (k \times y) \equiv k \times max\ x\ y$.

Lemma *max_eq_plus_cte_right* : ∀ x y k, $max\ (x+k)\ (y+k) \equiv (max\ x\ y) + k$.

Hint Resolve *max_eq_mult* *max_eq_plus_cte_right*.

4.13 Definition and properties of min

Definition *min* (*x* *y* : *U*) : *U* := [1-] ((*y* - *x*) + [1-]*y*).

Lemma *min_eq_right* : ∀ *x* *y* : *U*, *x* ≤ *y* → *min* *x* *y* ≡ *x*.

Lemma *min_eq_left* : ∀ *x* *y* : *U*, *y* ≤ *x* → *min* *x* *y*≡ *y*.

Hint Resolve *min_eq_right* *min_eq_left*.

Lemma *min_eq_case* : ∀ *x* *y* : *U*, *orc* (*min* *x* *y* ≡ *x*) (*min* *x* *y* ≡ *y*).

Add *Morphism* *min* with *signature* *Oeq* ⟹*Oeq* ⟹*Oeq* as *min_eq_compat*.
Save.
Hint Immediate *min_eq_compat*.

Lemma *min_le_right* : ∀ *x* *y* : *U*, *min* *x* *y* ≤*x*.

Lemma *min_le_left* : ∀ *x* *y* : *U*, *min* *x* *y* ≤ *y*.

Hint Resolve *min_le_right* *min_le_left*.

Lemma *min_le* : ∀ *x* *y* *z* : *U*, *z* ≤ *x* → *z* ≤ *y* → *z* ≤ *min* *x* *y*.

Lemma *Uinv_min_max* : ∀ *x* *y*, [1-](*min* *x* *y*)==*max* ([1-]*x*) ([1-]*y*).

Lemma *Uinv_max_min* : ∀ *x* *y*, [1-](*max* *x* *y*)==*min* ([1-]*x*) ([1-]*y*).

Lemma *min_idem* : ∀ *x*, *min* *x* *x* ≡ *x*.

Lemma *min_mult* : ∀ *x* *y* *k*,
    min (*k* × *x*) (*k* × *y*) ≡ *k* × (*min* *x* *y*).
Hint Resolve *min_mult*.

Lemma *min_plus* : ∀ *x1* *x2* *y1* *y2*,
    (*min* *x1* *x2*) + (*min* *y1* *y2*) ≤ *min* (*x1*+*y1*) (*x2*+*y2*).
Hint Resolve *min_plus*.

Lemma *min_plus_cte* : ∀ *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*.

Lemma *min_sym_le* : ∀ *x* *y*, *min* *x* *y* ≤ *min* *y* *x*.
Hint Resolve *min_sym_le*.

Lemma *min_sym* : ∀ *x* *y*, *min* *x* *y* ≡ *min* *y* *x*.
Hint Resolve *min_sym*.

Lemma *min_assoc* : ∀ *x* *y* *z*, *min* *x* (*min* *y* *z*) ≡ *min* (*min* *x* *y*) *z*.
Hint Resolve *min_assoc*.

Lemma *min_0* : ∀ *x*, *min* 0 *x* ≡ 0.
Hint Resolve *min_0*.

Instance *min_mon2* : *monotonic2* *min*.
Save.

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

Lemma *Min_simpl* : ∀ *x* *y*, *Min* *x* *y* = *min* *x* *y*.

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*)).

4.14 Other properties

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

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

Lemma *Uminus_assoc_left* : $\forall\ x\ y\ z,\ (x\ \text{-}\ y)\ \text{-}\ z \equiv x\ \text{-}\ (y + z)$.

Hint Resolve *Uminus_assoc_left*.

Lemma *Uminus_perm* : $\forall\ x\ y\ z,\ (x\ \text{-}\ y)\ \text{-}\ z \equiv (x\ \text{-}\ z)\ \text{-}\ y$.
Hint Resolve *Uminus_perm*.

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

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

Lemma *Uminus_eq_perm_left* : $\forall\ x\ y\ z,\ y \leq x \rightarrow x\ \text{-}\ y \equiv z \rightarrow x \equiv z + y$.

Lemma *Uplus_eq_perm_left* : $\forall\ x\ y\ z,\ y \leq [1\text{-}]\ z \rightarrow x \equiv y + z \rightarrow x\ \text{-}\ y \equiv z$.

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

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

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

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

Lemma *Uminus_eq_perm_right* : $\forall\ x\ y\ z,\ z \leq y \rightarrow x \equiv y\ \text{-}\ z \rightarrow x + z \equiv y$.
Hint Resolve *Uminus_eq_perm_right*.

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

Lemma *Uminus_zero_le* : $\forall\ x\ y,\ x\ \text{-}\ y \equiv 0 \rightarrow x \leq y$.

Lemma *Uminus_lt_non_zero* : $\forall\ x\ y,\ x < y \rightarrow \neg\ 0 \equiv y\ \text{-}\ x$.
Hint Immediate *Uminus_zero_le Uminus_lt_non_zero*.

Lemma *Ult_le_nth_minus* : $\forall\ x\ y,\ x < y \rightarrow exc$ (fun $n \Rightarrow x \leq y\ \text{-}\ [1/]1{+}n$).

Lemma *Uinv_plus_minus_left* : $\forall\ x\ y,\ [1\text{-}](x + y) \equiv [1\text{-}]x\ \text{-}\ y$.

Lemma *Uinv_plus_minus_right* : $\forall\ x\ y,\ [1\text{-}](x + y) \equiv [1\text{-}]y\ \text{-}\ x$.

Hint Resolve *Uinv_plus_minus_left Uinv_plus_minus_right*.

Lemma *Ult_le_nth_plus* : $\forall\ x\ y,\ x < y \rightarrow exc$ (fun $n : nat \Rightarrow x + [1/]1{+}n \leq y$).

Lemma *Uminus_distr_left* : $\forall\ x\ y\ z,\ (x\ \text{-}\ y) \times z \equiv (x \times z)\ \text{-}\ (y \times z)$.

Hint Resolve *Uminus_distr_left*.

Lemma *Uminus_distr_right* : $\forall\ x\ y\ z,\ x \times (y\ \text{-}\ z) \equiv (x \times y)\ \text{-}\ (x \times z)$.

Hint Resolve *Uminus_distr_right*.

Lemma *Uminus_assoc_right* : $\forall\ x\ y\ z,\ y \leq x \rightarrow z \leq y \rightarrow x\ \text{-}\ (y\ \text{-}\ z) \equiv (x\ \text{-}\ y) + z$.

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

Lemma *Uplus_minus_assoc_le* : $\forall\ x\ y\ z,\ (x + y)\ \text{-}\ z \leq x + (y\ \text{-}\ z)$.
Hint Resolve *Uplus_minus_assoc_le*.

Lemma *Udiv_minus* : $\forall\ x\ y\ z,\ \tilde{}0 \equiv z \rightarrow x \leq z \rightarrow (x\ \text{-}\ y)\ /\ z \equiv x/z\ \text{-}\ y/z$.

Lemma *Umult_inv_minus* : $\forall\ x\ y,\ x \times [1\text{-}]y \equiv x\ \text{-}\ x \times y$.
Hint Resolve *Umult_inv_minus*.

Lemma *Uinv_mult_minus* : $\forall\ x\ y,\ ([1\text{-}]x) \times y \equiv y\ \text{-}\ x \times y$.

Hint Resolve $Uinv\_mult\_minus$.

Lemma $Uminus\_plus\_perm\_right : \forall\ x\ y\ z,\ y \leq x \to y \leq z \to (x\ \text{-}\ y) + z \equiv x + (z\ \text{-}\ y)$.
Hint Resolve $Uminus\_plus\_perm\_right$.

Lemma $Uminus\_plus\_simpl\_mid$:
$\forall\ x\ y\ z,\ z \leq x \to y \leq z \to x\ \text{-}\ y \equiv (x\ \text{-}\ z) + (z\ \text{-}\ y)$.
Hint Resolve $Uminus\_plus\_simpl\_mid$.

- triangular inequality

Lemma $Uminus\_triangular : \forall\ x\ y\ z,\ x\ \text{-}\ y \leq (x\ \text{-}\ z) + (z\ \text{-}\ y)$.
Hint Resolve $Uminus\_triangular$.

Lemma $Uesp\_plus\_right\_perm : \forall\ x\ y\ z$,
$x \leq [1\text{-}]\ y \to y \leq [1\text{-}]\ z \to x\ \&\ (y + z) \equiv (x + y)\ \&\ z$.
Hint Resolve $Uesp\_plus\_right\_perm$.

Lemma $Uplus\_esp\_assoc : \forall\ x\ y\ z$,
$x \leq [1\text{-}]y \to [1\text{-}]z \leq y \to x + (y\ \&\ z) \equiv (x + y)\ \&\ z$.
Hint Resolve $Uplus\_esp\_assoc$.

Lemma $Uesp\_plus\_left\_perm : \forall\ x\ y\ z$,
$[1\text{-}]x \leq y \to [1\text{-}]z \leq y \to x\ \&\ y \leq [1\text{-}]\ z \to (x\ \&\ y) + z \equiv x + (y\ \&\ z)$.
Hint Resolve $Uesp\_plus\_left\_perm$.

Lemma $Uesp\_plus\_left\_perm\_le : \forall\ x\ y\ z$,
$[1\text{-}]x \leq y \to [1\text{-}]z \leq y \to (x\ \&\ y) + z \leq x + (y\ \&\ z)$.
Hint Resolve $Uesp\_plus\_left\_perm\_le$.

Lemma $Uesp\_plus\_assoc : \forall\ x\ y\ z$,
$[1\text{-}]x \leq y \to y \leq [1\text{-}]z \to x\ \&\ (y + z) \equiv (x\ \&\ y) + z$.
Hint Resolve $Uesp\_plus\_assoc$.

Lemma $Uminus\_assoc\_right\_perm : \forall\ x\ y\ z$,
$x \leq [1\text{-}]\ z \to z \leq y \to x\ \text{-}\ (y\ \text{-}\ z) \equiv x + z\ \text{-}\ y$.
Hint Resolve $Uminus\_assoc\_right\_perm$.

Lemma $Uminus\_lt\_left : \forall\ x\ y,\ \neg\ 0 \equiv x \to \neg\ 0 \equiv y \to x\ \text{-}\ y < x$.
Hint Resolve $Uminus\_lt\_left$.

Lemma $Uesp\_mult\_le$:
$\forall\ x\ y\ z,\ [1\text{-}]x \leq y \to x \times z \leq [1\text{-}](y \times z)$
$\to (x\ \&\ y) \times z \equiv x \times z + y \times z\ \text{-}\ z$.
Hint Resolve $Uesp\_mult\_le$.

Lemma $Uesp\_mult\_ge$:
$\forall\ x\ y\ z,\ [1\text{-}]x \leq y \to [1\text{-}](x \times z) \leq y \times z$
$\to (x\ \&\ y) \times z \equiv (x \times z)\ \&\ (y \times z) + [1\text{-}]z$.
Hint Resolve $Uesp\_mult\_ge$.

4.15 Definition and properties of generalized sums

Definition $sigma : (nat \to U) \to nat\ \text{-}m\text{>}\ U$.
Defined.

Lemma $sigma\_0 : \forall\ (f : nat \to U),\ sigma\ f\ O \equiv 0$.

Lemma $sigma\_S : \forall\ (f : nat \to U)\ (n{:}nat),\ sigma\ f\ (S\ n) = (f\ n) + (sigma\ f\ n)$.

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

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* : ∀ (f g: nat → U) (n:nat),
(∀ k, (k < n)%nat → f k ≡ g k) → sigma f n ≡ sigma g n.

Lemma *sigma_le_compat* : ∀ (f g: nat → U) (n:nat),
(∀ k, (k < n)%nat → f k ≤ g k) → sigma f n ≤ sigma g n.

Lemma *sigma_S_lift* : ∀ (f :nat → U) (n:nat),
sigma f (S n) ≡ (f O) + (sigma (fun k ⇒ f (S k)) n).

Lemma *sigma_plus_lift* : ∀ (f :nat → U) (n m:nat),
sigma f (n+m)%nat ≡ sigma f n + sigma (fun k ⇒ f (n+k)%nat) m.

Lemma *sigma_zero* : ∀ f n,
(∀ k, (k<n)%nat → f k ≡ 0) → sigma f n ≡ 0.

Lemma *sigma_not_zero* : ∀ f n k, (k<n)%nat → 0 < f k → 0 < sigma f n.

Lemma *sigma_zero_elim* : ∀ f n,
(sigma f n) ≡ 0 → ∀ k, (k<n)%nat → f k ≡ 0.

Hint Resolve *sigma_eq_compat sigma_le_compat sigma_zero*.

Lemma *sigma_le* : ∀ f n k, (k<n)%nat → f k ≤ sigma f n.
Hint Resolve *sigma_le*.

Lemma *sigma_minus_decr* : ∀ f n, (∀ k, f (S k) ≤ f k) →
sigma (fun k ⇒ f k - f (S k)) n ≡ f O - f n.

Lemma *sigma_minus_incr* : ∀ f n, (∀ k, f k ≤ f (S k)) →
sigma (fun k ⇒ f (S k) - f k) n ≡ f n - f O.

4.16 Definition and properties of generalized products

Definition *prod* (alpha : nat → U) (n:nat) := compn Umult 1 alpha n.

Lemma *prod_0* : ∀ (f : nat → U), prod f 0 = 1.

Lemma *prod_S* : ∀ (f :nat → U) (n:nat), prod f (S n) = (f n) × (prod f n).

Lemma *prod_1* : ∀ (f : nat → U), prod f (S 0) ≡ f O.

Lemma *prod_S_lift* : ∀ (f :nat → U) (n:nat),
prod f (S n) ≡ (f O) × (prod (fun k ⇒ f (S k)) n).

Lemma *prod_decr* : ∀ (f : nat → U) (n m:nat), (n ≤ m)%nat → prod f m ≤ prod f n.

Hint Resolve *prod_decr*.

Lemma *prod_eq_compat* : ∀ (f g: nat → U) (n:nat),
(∀ k, (k < n)%nat → f k ≡ g k) → (prod f n) ≡ (prod g n).

Lemma *prod_le_compat* : ∀ (f g: nat → U) (n:nat),
(∀ k, (k < n)%nat → f k ≤ g k) → prod f n ≤ prod g n.

Lemma *prod_zero* : ∀ f n k, (k<n)%nat → f k ==0 → prod f n==0.

Lemma *prod_not_zero* : ∀ f n,
(∀ k, (k<n)%nat → 0 < f k) → 0 < prod f n.

Lemma *prod_zero_elim* : ∀ f n,
prod f n ≡ 0 → exc (fun k ⇒ (k<n)%nat ∧ f k ==0).

Hint Resolve *prod_eq_compat prod_le_compat prod_not_zero*.

Lemma *prod_le* : ∀ f n k, (k<n)%nat → prod f n ≤ f k.

Lemma *prod_minus* : ∀ f n, prod f n - prod f (S n) ≡ ([1-]f n) × prod f n.

Definition *Prod* : (nat → U) → nat -m→ U.
Defined.

Lemma *Prod_simpl* : ∀ f n, Prod f n = prod f n.
Hint Resolve *Prod_simpl*.

4.17 Properties of *Unth*

Lemma *Unth_eq_compat* : $\forall$ n m, $n = m \rightarrow [1/]1{+}n \equiv [1/]1{+}m$.
Hint Resolve *Unth_eq_compat*.

Lemma *Unth_zero* : $[1/]1{+}0 \equiv 1$.

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

Lemma *Unth_one* : $\frac{1}{2} \equiv [1\text{-}]\ \frac{1}{2}$.

Hint Resolve *Unth_zero* *Unth_one*.

Lemma *Unth_one_plus* : $\frac{1}{2} + \frac{1}{2} \equiv 1$.
Hint Resolve *Unth_one_plus*.

Lemma *Unth_one_refl* : $\forall$ t, $\frac{1}{2} \times t + \frac{1}{2} \times t \equiv t$.

Lemma *Unth_not_null* : $\forall$ n, $\neg$ $(0 \equiv [1/]1{+}n)$.
Hint Resolve *Unth_not_null*.

Lemma *Unth_lt_zero* : $\forall$ n, $0 < [1/]1{+}n$.
Hint Resolve *Unth_lt_zero*.

Lemma *Unth_inv_lt_one* : $\forall$ n, $[1\text{-}][1/]1{+}n{<}1$.
Hint Resolve *Unth_inv_lt_one*.

Lemma *Unth_not_one* : $\forall$ n, $\neg$ $(1 \equiv [1\text{-}][1/]1{+}n)$.
Hint Resolve *Unth_not_one*.

Lemma *Unth_prop_sigma* : $\forall$ n, $[1/]1{+}n \equiv [1\text{-}]$ (*sigma* (fun $k \Rightarrow [1/]1{+}n$) n).
Hint Resolve *Unth_prop_sigma*.

Lemma *Unth_sigma_n* : $\forall$ n : *nat*, $\neg$ $(1 \equiv$ *sigma* (fun $k \Rightarrow [1/]1{+}n$) n).

Lemma *Unth_sigma_Sn* : $\forall$ n : *nat*, $1 \equiv$ *sigma* (fun $k \Rightarrow [1/]1{+}n$) $(S\ n)$.

Hint Resolve *Unth_sigma_n* *Unth_sigma_Sn*.

Lemma *Unth_decr* : $\forall$ n m, $(n < m)\%nat \rightarrow [1/]1{+}m < [1/]1{+}n$.
Hint Resolve *Unth_decr*.

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

Lemma *Unth_le_compat* :
$\forall$ n m, $(n \leq m)\%nat \rightarrow [1/]1{+}m \leq [1/]1{+}n$.
Hint Resolve *Unth_le_compat*.

Lemma *Unth_le_equiv* :
$\forall$ n m, $[1/]1{+}n \leq [1/]1{+}m \leftrightarrow (m \leq n)\%nat$.

Lemma *Unth_eq_equiv* :
$\forall$ n m, $[1/]1{+}n \equiv [1/]1{+}m \leftrightarrow (m = n)\%nat$.

Lemma *Unth_le_half* : $\forall$ n, $[1/]1{+}(S\ n) \leq \frac{1}{2}$.
Hint Resolve *Unth_le_half*.

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

Definition *mean* $(x\ y{:}U) := \frac{1}{2} \times x + \frac{1}{2} \times y$.

Lemma *mean_eq* : $\forall$ $x{:}U$, *mean* x x $\equiv x$.

Lemma *mean_le_compat_right* : $\forall$ x y z, $y \leq z \rightarrow$ *mean* x y $\leq$ *mean* x z.

Lemma *mean_le_compat_left* : $\forall$ x y z, $x \leq y \rightarrow$ *mean* x z $\leq$ *mean* y z.

Hint Resolve *mean_eq* *mean_le_compat_left* *mean_le_compat_right*.

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

Lemma *mean_lt_compat_left* : $\forall$ x y z, $x < y \rightarrow$ *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* : $\forall\ x\ y,\ x \leq y \rightarrow mean\ x\ y \leq y$.

Lemma *mean_le_down* : $\forall\ x\ y,\ x \leq y \rightarrow x \leq mean\ x\ y$.

Lemma *mean_lt_up* : $\forall\ x\ y,\ x < y \rightarrow mean\ x\ y < y$.

Lemma *mean_lt_down* : $\forall\ x\ y,\ x < y \rightarrow x < mean\ x\ y$.

Hint Resolve *mean_le_up mean_le_down mean_lt_up mean_lt_down.*

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

Lemma *le_half_inv* : $\forall\ x,\ x \leq \frac{1}{2} \rightarrow x \leq$ [1-] x.

Hint Immediate *le_half_inv.*

Lemma *ge_half_inv* : $\forall\ x,\ \frac{1}{2} \leq x \rightarrow$ [1-] $x \leq x$.

Hint Immediate *ge_half_inv.*

Lemma *Uinv_le_half_left* : $\forall\ x,\ x \leq \frac{1}{2} \rightarrow \frac{1}{2} \leq$ [1-] x.

Lemma *Uinv_le_half_right* : $\forall\ x,\ \frac{1}{2} \leq x \rightarrow$ [1-] $x \leq \frac{1}{2}$.

Hint Resolve *Uinv_le_half_left Uinv_le_half_right.*

Lemma *half_twice* : $\forall\ x,\ x \leq \frac{1}{2} \rightarrow \frac{1}{2} \times (x + x) \equiv x$.

Lemma *half_twice_le* : $\forall\ x,\ \frac{1}{2} \times (x + x) \leq x$.

Lemma *Uinv_half* : $\forall\ x,\ \frac{1}{2} \times$ ([1-] x) $+ \frac{1}{2} \equiv$ [1-] ($\frac{1}{2} \times x$).

Lemma *Uinv_half_plus* : $\forall\ x$, [1-]$x + \frac{1}{2} \times x \equiv$ [1-] ($\frac{1}{2} \times x$).

Lemma *half_esp* :
$\forall\ x$, ([1/2] $\leq x$) $\rightarrow$ ([1/2]) $\times$ (x & x) $+ \frac{1}{2} \equiv x$.

Lemma *half_esp_le* : $\forall\ x,\ x \leq \frac{1}{2} \times$ (x & x) $+ \frac{1}{2}$.
Hint Resolve *half_esp_le.*

Lemma *half_le* : $\forall\ x\ y,\ y \leq$ [1-] $y \rightarrow x \leq y + y \rightarrow$ ([1/2]) $\times\ x \leq y$.

Lemma *half_Unth_le*: $\forall\ n,\ \frac{1}{2} \times$ ([1/]1+n) $\leq$ [1/]1+($S\ n$).
Hint Resolve *half_le half_Unth_le.*

Lemma *half_exp* : $\forall\ n$, [1/2]^$n \equiv$ [1/2]^($S\ n$) + [1/2]^($S\ n$).

4.18 Diff function : | x - y |

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

Lemma *diff_eq* : $\forall\ x$, *diff* $x\ x \equiv 0$.
Hint Resolve *diff_eq.*

Lemma *diff_sym* : $\forall\ x\ y$, *diff* $x\ y \equiv$ *diff* $y\ x$.
Hint Resolve *diff_sym.*

Lemma *diff_zero* : $\forall\ x$, *diff* $x\ 0 \equiv x$.
Hint Resolve *diff_zero.*

Add *Morphism diff* with *signature Oeq* $\Longrightarrow$ *Oeq* $\Longrightarrow$ *Oeq* as *diff_eq_compat.*
Qed.
Hint Immediate *diff_eq_compat.*

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

Lemma *diff_Uminus* : $\forall\ x\ y,\ x \leq y \rightarrow$ *diff* $x\ y \equiv y$ - x.

Lemma *diff_Uplus_le* : $\forall\ x\ y,\ x \leq$ *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*.

4.19 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*.

4.20 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* : ∀ *f g* : *nat -m> U*,
  (∀ *n*, *exc* (fun *m* ⇒ (*n*≤*m*)%*nat* ∧ *f n* ≤ *g m*))
  → *min* (*lub f*) (*lub g*) ≤ *lub* ((*Min* $@^2$ *f*) *g*).

Lemma *min_lub_le_incr* : ∀ *f g* : *nat -m> U*,
  min (*lub f*) (*lub g*) ≤ *lub* ((*Min* $@^2$ *f*) *g*).

Lemma *min_continuous2* : *continuous2 Min*.
Hint Resolve *min_continuous2*.

Lemma *lub_eq_esp_right* :
  ∀ (*f* : *nat -m> U*) (*k* : *U*), *lub* ((*mshift UEsp k*) @ *f*) ≡ *lub f* & *k*.
Hint Resolve *lub_eq_esp_right*.

Lemma *Udiv_continuous* : ∀ (*k*:*U*), *continuous* (*UDiv k*).
Hint Resolve *Udiv_continuous*.

4.21 Greatest lower bounds

Definition *glb* (*f*:*nat -m→ U*) := [1-](*lub* (*UInv* @ *f*)).

Lemma *glb_le*: ∀ (f : *nat -m→ U*) (n : *nat*), *glb f* ≤ (f n).

Lemma *le_glb*: ∀ (f : *nat -m→ U*) (x:U),
(∀ n : *nat*, x ≤ f n) → x ≤ *glb f*.
Hint Resolve *glb_le le_glb*.

Definition *Uopp* : *cpo* (*o*:=*Iord U*) *U*.
Defined.

Lemma *Uopp_lub_simpl*
: ∀ h : *nat -m→ U*, *lub* (*cpo*:=*Uopp*) h = *glb h*.

Lemma *Uopp_mon_seq* : ∀ f:*nat -m→ U*,
∀ n m:*nat*, (n ≤ m)%*nat* → f m ≤ f n.
Hint Resolve *Uopp_mon_seq*.

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

Properties of *glb*

Lemma *glb_le_compat*:
∀ f g : *nat -m→ U*, (∀ x, f x ≤ g x) → *glb f* ≤ *glb g*.
Hint Resolve *glb_le_compat*.

Lemma *glb_eq_compat*:
∀ f g : *nat -m→ U*, f ≡ g → *glb f* ≡ *glb g*.
Hint Resolve *glb_eq_compat*.

Lemma *glb_cte*: ∀ c : *U*, *glb* (*mon* (*cte nat* (*o1*:=(*Iord U*)) c)) ≡ c.
Hint Resolve *glb_cte*.

Lemma *glb_eq_plus_cte_right*:
∀ (f : *nat -m→ 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*:
∀ (f : *nat -m→ U*) (k : *U*), *glb* (*Imon* (*UPlus k*) @ f) ≡ k + *glb f*.
Hint Resolve *glb_eq_plus_cte_left*.

Lemma *glb_eq_mult*:
∀ (k : *U*) (f : *nat -m→ U*), *glb* (*Imon* (*UMult k*) @ f) ≡ k × *glb f*.

Lemma *Imon2_plus_continuous*
: *continuous2* (*c1*:=*Uopp*) (*c2*:=*Uopp*) (*c3*:=*Uopp*) (*imon2 Uplus*).

Hint Resolve *Imon2_plus_continuous*.

Lemma *Uinv_continuous* : *continuous* (*c1*:=*Uopp*) *UInv*.

Lemma *Uinv_lub_eq* : ∀ f : *nat -m→ U*, [1-](*lub* (*cpo*:=*Uopp*) f) ≡ *lub* (*UInv*@f).

Lemma *Uinvopp_mon* : *monotonic* (*o2*:= *Iord U*) *Uinv*.
Hint Resolve *Uinvopp_mon*.

Definition *UInvopp* : *U -m→ U*
:= *mon* (*o2*:= *Iord U*) *Uinv* (*fmonotonic*:=*Uinvopp_mon*).

Lemma *UInvopp_simpl* : ∀ x, *UInvopp* x = [1-]x.

Lemma *Uinvopp_continuous* : *continuous* (*c2*:=*Uopp*) *UInvopp*.

Lemma *Uinvopp_lub_eq*
: ∀ f : *nat -m> U*, [1-](*lub f*) ≡ *lub* (*cpo*:=*Uopp*) (*UInvopp*@f).

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: ∀ (f : nat → U) (x:U),
  (∀ n : nat, x ≤ f n) → x ≤ Uglb f.
Hint Resolve Uglb_le le_Uglb.

Lemma Uglb_le_compat : ∀ f g : nat → U, f ≤ g → Uglb f ≤ Uglb g.
Hint Resolve Uglb_le_compat.

Add Morphism Uglb with signature Oeq ⟹Oeq as Uglb_eq_compat.
Save.
Hint Resolve Uglb_eq_compat.

Lemma Uglb_eq_plus_cte_right:
```

∀ (f : nat → U) (k : U), Uglb (fun n ⇒ f n + k) ≡ Uglb f + k.
Hint Resolve Uglb_eq_plus_cte_right.

Lemma Uglb_eq_mult:
∀ (k : U) (f : nat → U), Uglb (fun n ⇒ k × f n) ≡ k × Uglb f.
Hint Resolve Uglb_eq_mult Uglb_eq_plus_cte_right.

Lemma Uglb_le_plus : ∀ f g, Uglb f + Uglb g ≤ Uglb (fun n ⇒ f n + g n).
Hint Resolve Uglb_le_plus.

Lemma Ulub_lub : ∀ f:nat -m> U, Ulub f ≡ lub f.
Hint Resolve Ulub_lub.

Lemma Uglb_glb : ∀ f:nat -m→ U, Uglb f ≡ glb f.
Hint Resolve Uglb_glb.

Lemma lub_le_plus : ∀ (f g : nat -m> U), lub ((UPlus @^{2} f) g) ≤ lub f + lub g.
Hint Resolve lub_le_plus.

Lemma glb_le_plus : ∀ (f g:nat -m→ U) , glb f + glb g ≤ glb ((Imon2 UPlus @^{2} f) g).
Hint Resolve glb_le_plus.

Lemma lub_eq_plus : ∀ f g : nat -m> U, lub ((UPlus @^{2} f) g) ≡ lub f + lub g.
Hint Resolve lub_eq_plus.

Lemma glb_mon : ∀ f : nat -m> U, Uglb f ≡ f O.

Lemma lub_inv : ∀ (f g : nat -m> U), (∀ n, f n ≤ [1-] g n) → lub f ≤ [1-] (lub g).

Lemma glb_lift_left : ∀ (f:nat -m→ U) n,
glb f ≡ glb (mon (seq_lift_left f n)).
Hint Resolve glb_lift_left.

Lemma Ulub_mon : ∀ f : nat -m→ U, Ulub f ≡ f O.

Lemma lub_glb_le : ∀ (f:nat -m> U) (g:nat -m→ U),
(∀ n, f n ≤ g n) → lub f ≤ glb g.

Lemma lub_lub_inv_le : ∀ f g :nat -m> U,
(∀ n, f n ≤ [1-]g n) → lub f ≤ [1-] lub g.

Lemma Uplus_opp_continuous_right :
∀ 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.

Instance Uplusopp_continuous2 : continuous2 (c1:=Uopp) (c2:=Uopp) (c3:=Uopp) (Imon2 UPlus).
Save.

Lemma Uplusopp_lub_eq : ∀ (f g : nat -m→ U),
lub (cpo:=Uopp) f + lub (cpo:=Uopp) g ≡ lub (cpo:=Uopp) ((Imon2 UPlus @^{2} f) g).

Lemma glb_eq_plus : ∀ (f g : nat -m→ U), glb ((Imon2 UPlus @^{2} f) g) ≡ glb f + glb g.
Hint Resolve glb_eq_plus.

Instance UEsp_continuous2 : continuous2 UEsp.
Save.

Lemma Uesp_lub_eq : ∀ f g : nat -m> U, lub f & lub g ≡ lub ((UEsp @^{2} f) g).

Instance sigma_mon :monotonic sigma.
Save.

Definition Sigma : (nat → U) -m> nat-m> U
:= mon sigma (fmonotonic:=sigma_mon).

Lemma Sigma_simpl : ∀ f, Sigma f = sigma f.

```
Lemma sigma_continuous1 : continuous Sigma.

Lemma sigma_lub1 : ∀ (f : nat -m> (nat → U)) n,
      sigma (lub f) n ≡ lub ((mshift Sigma n) @ f).

Definition MF (A:Type) : Type := A → U.

Definition MFcpo (A:Type) : cpo (MF A) := fcpo cpoU.

Definition MFopp (A:Type) : cpo (o:=Iord (A → U)) (MF A).
Defined.

Lemma MFopp_lub_eq : ∀ (A:Type) (h:nat-m→ MF A),
    lub (cpo:=MFopp A) h ≡ fun x ⇒ glb (Iord_app x @ h).

Lemma fle_intro : ∀ (A:Type) (f g : MF A), (∀ x, f x ≤ g x) → f ≤ g.
Hint Resolve fle_intro.

Lemma feq_intro : ∀ (A:Type) (f g : MF A), (∀ x, f x ≡ g x) → f ≡ g.
Hint Resolve feq_intro.

Definition fplus (A:Type) (f g : MF A) : MF A :=
              fun x ⇒ f x + g x.

Definition fmult (A:Type) (k:U) (f : MF A) : MF A :=
              fun x ⇒ k × f x.

Definition finv (A:Type) (f : MF A) : MF A :=
              fun x ⇒ [1-] f x.

Definition fzero (A:Type) : MF A :=
              fun x ⇒ 0.

Definition fdiv (A:Type) (k:U) (f : MF A) : MF A :=
              fun x ⇒ (f x) / k.

Definition flub (A:Type) (f : nat -m> MF A) : MF A := lub f.

Lemma fplus_simpl : ∀ (A:Type)(f g : MF A) (x : A),
                          fplus f g x = f x + g x.

Lemma fplus_def : ∀ (A:Type)(f g : MF A),
                          fplus f g = fun x ⇒ f x + g x.

Lemma fmult_simpl : ∀ (A:Type)(k:U) (f : MF A) (x : A),
                          fmult k f x = k × f x.

Lemma fmult_def : ∀ (A:Type)(k:U) (f : MF A),
                          fmult k f = fun x ⇒ k × f x.

Lemma fdiv_simpl : ∀ (A:Type)(k:U) (f : MF A) (x : A),
                          fdiv k f x = f x / k.

Lemma fdiv_def : ∀ (A:Type)(k:U) (f : MF A),
                          fdiv k f = fun x ⇒ f x / k.

Implicit Arguments fzero [].

Lemma fzero_simpl : ∀ (A:Type)(x : A), fzero A x = 0.

Lemma fzero_def : ∀ (A:Type), fzero A = fun x:A ⇒ 0.

Lemma finv_simpl : ∀ (A:Type)(f : MF A) (x : A), finv f x = [1-]f x.

Lemma finv_def : ∀ (A:Type)(f : MF A), finv f = fun x ⇒ [1-](f x).

Lemma flub_simpl : ∀ (A:Type)(f:nat -m> MF A) (x:A),
                        (flub f) x = lub (f <o> x).

Lemma flub_def : ∀ (A:Type)(f:nat -m> MF A),
                        (flub f) = fun x ⇒ lub (f <o> x).
```

```
Hint Resolve fplus_simpl fmult_simpl fzero_simpl finv_simpl flub_simpl.

Definition fone (A:Type) : MF A := fun x ⇒ 1.
Implicit Arguments fone [].

Lemma fone_simpl : ∀ (A:Type) (x:A), fone A x = 1.

Lemma fone_def : ∀ (A:Type), fone A = fun (x:A) ⇒ 1.

Definition fcte (A:Type) (k:U): MF A := fun x ⇒ k.
Implicit Arguments fcte [].

Lemma fcte_simpl : ∀ (A:Type) (k:U) (x:A), fcte A k x = k.

Lemma fcte_def : ∀ (A:Type) (k:U), fcte A k = fun (x:A) ⇒ 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.

Lemma fminus_def : ∀ (A:Type) (f g: MF A), fminus f g = fun 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.

Lemma fesp_def : ∀ (A:Type) (f g: MF A) , fesp f g = fun 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 fconj_def : ∀ (A:Type) (f g: MF A), fconj f g = fun 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.

Lemma MF_lub_def : ∀ (A:Type) (f : nat -m> MF A),
             lub f = fun x ⇒ lub (f <o>x).
```

4.21.1 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*.

4.21.2 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*.

4.21.3 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* -*m*> *MF A* -*m*> *MF A* := *mon2* (*fconj* (*A*:=*A*)).

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

Lemma *fconj_sym* : $\forall$ *A* (*f g* : *MF A*), *fconj f g* $\equiv$ *fconj g f*.
Hint Resolve *fconj_sym*.

Lemma *Fconj_sym* : $\forall$ *A* (*f g* : *MF A*), *Fconj A f g* $\equiv$ *Fconj A g f*.
Hint Resolve *Fconj_sym*.

Lemma *lub_MF_simpl* : $\forall$ *A* (*h* : *nat -m> MF A*) (*x*:*A*), *lub h x* = *lub* (*h* <*o*> *x*).

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

Definition *Fmult A* : *U -m> MF A -m> MF A* := *mon2* (*fmult* (*A*:=*A*)).

Lemma *Fmult_simpl* : $\forall$ *A k f*, *Fmult A k f* = *fmult k f*.

Lemma *Fmult_simpl2* : $\forall$ *A k f x*, *Fmult A k f x* = *k* $\times$ (*f x*).

Lemma *fmult_continuous2* : $\forall$ *A*, *continuous2* (*Fmult A*).

Lemma *Umult_sym_cst*:
  $\forall$ *A* : Type,
  $\forall$ (*k* : *U*) (*f* : *MF A*), (fun *x* : *A* $\Rightarrow$ *f x* $\times$ *k*) $\equiv$ (fun *x* : *A* $\Rightarrow$ *k* $\times$ *f x*).

4.22 Fixpoints of functions of type $A \rightarrow U$

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

Variable *F* : *MF A -m> MF A*.

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

Definition *G* : *MF A* –*m*→ *MF A* := *Imon F*.

Definition *nufix* : *MF A* := *fixp* (*c*:=*MFopp A*) *G*.

Lemma *mufix_inv* : $\forall$ *f* : *MF A*, *F f* $\leq$ *f* $\rightarrow$ *mufix* $\leq$ *f*.
Hint Resolve *mufix_inv*.

Lemma *nufix_inv* : $\forall$ *f* :*MF A*, *f* $\leq$ *F f* $\rightarrow$ *f* $\leq$ *nufix*.
Hint Resolve *nufix_inv*.

Lemma *mufix_le* : *mufix* $\leq$ *F mufix*.
Hint Resolve *mufix_le*.

Lemma *nufix_sup* : *F nufix* $\leq$ *nufix*.
Hint Resolve *nufix_sup*.

Lemma *mufix_eq* : *continuous F* $\rightarrow$ *mufix* $\equiv$ *F mufix*.
Hint Resolve *mufix_eq*.

Lemma *nufix_eq* : *continuous* (*c1*:=*MFopp A*) (*c2*:=*MFopp A*) *G* $\rightarrow$ *nufix* $\equiv$ *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 -m> MF A* := *mon* (*cte* (*MF A*) *f*).

Lemma *mufix_cte* : $\forall$ (*A*:Type) (*f*:*MF A*), *mufix* (*Fcte f*) $\equiv$ *f*.

Lemma *nufix_cte* : $\forall$ (*A*:Type) (*f*:*MF A*), *nufix* (*Fcte f*) $\equiv$ *f*.

Hint Resolve *mufix_cte nufix_cte*.

4.23 Properties of (pseudo-)barycenter of two points

Lemma *Uinv_bary* :
  $\forall$ *a b x y* : *U*, *a* $\leq$ [1-]*b* $\rightarrow$
    [1-] (*a* $\times$ *x* + *b* $\times$ *y*) $\equiv$ *a* $\times$ [1-] *x* + *b* $\times$ [1-] *y* + [1-] (*a* + *b*).

Hint Resolve *Uinv_bary*.

Lemma *Uinv_bary_le* :
$\forall$ *a b x y* : *U*, $a \le$ [1-]$b \to a \times$ [1-] $x + b \times$ [1-] $y \le$ [1-] $(a \times x + b \times y)$.
Hint Resolve *Uinv_bary_le*.

Lemma *Uinv_bary_eq* : $\forall$ *a b x y* : *U*, $a \equiv$ [1-]$b \to$
[1-] $(a \times x + b \times y) \equiv a \times$ [1-] $x + b \times$ [1-] y.
Hint Resolve *Uinv_bary_eq*.

Lemma *bary_refl_eq* : $\forall$ *a b x*, $a \equiv$ [1-]$b \to a \times x + b \times x \equiv x$.
Hint Resolve *bary_refl_eq*.

Lemma *bary_refl_feq* : $\forall$ *A a b* (*f*:$A \to U$) ,
$a \equiv$ [1-]$b \to$ (fun $x \Rightarrow a \times f\ x + b \times f\ x) \equiv f$.
Hint Resolve *bary_refl_feq*.

Lemma *bary_le_left* : $\forall$ *a b x y*, [1-]$b \le a \to x \le y \to x \le a \times x + b \times y$.

Lemma *bary_le_right* : $\forall$ *a b x y*, $a \le$ [1-]$b \to x \le y \to a \times x + b \times y \le y$.

Hint Resolve *bary_le_left bary_le_right*.

Lemma *bary_up_eq* : $\forall$ *a b x y* : *U*, $a \equiv$ [1-]$b \to x \le y \to a \times x + b \times y \equiv x + b \times (y - x)$.

Lemma *bary_up_le* : $\forall$ *a b x y* : *U*, $a \le$ [1-]$b \to a \times x + b \times y \le x + b \times (y - x)$.

Lemma *bary_anti_mon* : $\forall$ *a b a' b' x y* : *U*,
$a \equiv$ [1-]$b \to a' \equiv$ [1-]$b' \to a \le a' \to x \le y \to a' \times x + b' \times y \le a \times x + b \times y$.
Hint Resolve *bary_anti_mon*.

Lemma *bary_Uminus_left* :
$\forall$ *a b x y* : *U*, $a \le$ [1-]$b \to (a \times x + b \times y) - x \le b \times (y - x)$.

Lemma *bary_Uminus_left_eq* :
$\forall$ *a b x y* : *U*, $a \equiv$ [1-]$b \to x \le y \to (a \times x + b \times y) - x \equiv b \times (y - x)$.

Lemma *Uminus_bary_left*
: $\forall$ *a b x y* : *U*, [1-]$a \le b \to x - (a \times x + b \times y) \le b \times (x - y)$.

Lemma *Uminus_bary_left_eq*
: $\forall$ *a b x y* : *U*, $a \equiv$ [1-]$b \to y \le x \to x - (a \times x + b \times y) \equiv b \times (x - y)$.

Hint Resolve *bary_up_eq bary_up_le bary_Uminus_left Uminus_bary_left bary_Uminus_left_eq Uminus_bary_left_eq*.

Lemma *bary_le_simpl_right*
: $\forall$ *a b x y* : *U*, $a \equiv$ [1-]$b \to \neg\ 0 \equiv a \to a \times x + b \times y \le y \to x \le y$.

Lemma *bary_le_simpl_left*
: $\forall$ *a b x y* : *U*, $a \equiv$ [1-]$b \to \neg\ 0 \equiv b \to x \le a \times x + b \times y \to x \le y$.

Lemma *diff_bary_left_eq*
: $\forall$ *a b x y* : *U*, $a \equiv$ [1-]$b \to$ *diff* $x\ (a \times x + b \times y) \equiv b \times$ *diff* $x\ y$.
Hint Resolve *diff_bary_left_eq*.

Lemma *Uinv_half_bary* :
$\forall$ *x y* : *U*, [1-] ([1/2] $\times x + \frac{1}{2} \times y) \equiv \frac{1}{2} \times$ [1-] $x + \frac{1}{2} \times$ [1-] y.
Hint Resolve *Uinv_half_bary*.

Lemma *Uinv_Umult* : $\forall$ *x y*, [1-]$x \times$ [1-]$y \equiv$ [1-]$(x - x \times y + y)$.
Hint Resolve *Uinv_Umult*.

4.24 Properties of generalized sums *sigma*

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

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

Lemma *retract_class* : $\forall$ *f n*, *class* (*retract f n*).
Hint Resolve *retract_class*.

Lemma *retract0* : $\forall$ (*f* : *nat* $\rightarrow$ *U*), *retract f* 0.

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

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

Hint Immediate *retract_pred retractS*.

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

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

Hint Resolve *retract0 retractS_intro*.

Lemma *retract_lt* : $\forall$ (*f* : *nat* $\rightarrow$ *U*) (*n* : *nat*), *sigma f n* $<$ 1 $\rightarrow$ *retract f n*.

Lemma *retract_unif* :
  $\forall$ (*f* : *nat* $\rightarrow$ *U*) (*n* : *nat*),
    ($\forall$ *k*, (*k*$\leq$*n*)%*nat* $\rightarrow$ *f k* $\leq$ [1/]1+*n*) $\rightarrow$ *retract f* (*S n*).

Hint Resolve *retract_unif*.

Lemma *retract_unif_Nnth* :
  $\forall$ (*f* : *nat* $\rightarrow$ *U*) (*n* : *nat*),
  ($\forall$ *k* : *nat*, (*k* $\leq$ *n*)%*nat* $\rightarrow$ *f k* $\leq$ [1/]*n*) $\rightarrow$ *retract f n*.
Hint Resolve *retract_unif_Nnth*.

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

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

Hint Resolve *sigma_prod_maj*.

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

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

Hint Resolve *sigma_prod_maj sigma_prod_le sigma_prod_ge*.

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

4.25 Product by an integer

4.25.1 Definition of *Nmult n x* written *n* */ *x*

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

4.25.2 Condition for *n* */ *x* to be exact : $n = 0$ or $x \leq 1/n$

Definition *Nmult_def* (*n*: *nat*) (*x* : *U*) :=
  match *n* with *O* $\Rightarrow$ *True* | *S p* $\Rightarrow$ *x* $\leq$ [1/]1+*p* end.

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

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

Hint Resolve *Nmult_def_1*.

Lemma *Nmult_def_intro* : ∀ *n x* , *x* ≤ [1/]1+*n* → *Nmult_def* (*S n*) *x*.
Hint Resolve *Nmult_def_intro*.

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

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

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

Lemma *Nmult_def_Nnth* : ∀ *n*, *Nmult_def n* ([1/]*n*).
Hint Resolve *Nmult_def_Nnth*.

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*.

4.25.3 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_0 Nmult_zero 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_n_Nnth : ∀ n : nat, (0 < n)%nat → n */ [1/]n ≡ 1.
Hint Resolve Nmult_n_Nnth.

Lemma Nnth_S : ∀ n, [1/](S n) ≡ [1/]1+n.

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_def_inv : ∀ n x, Nmult_def (S n) x → n */ x ≤ [1-] x.
Hint Resolve Nmult_def_inv.

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 Unth_eq : ∀ n p, n */ p ≡ [1-]p → p ≡ [1/]1+n.

Lemma mult_Nmult_Umult : ∀ n m x y,
  Nmult_def n x → Nmult_def m y → (n×m)%nat */ (x×y) ≡ (n*/x)*(m*/y).

Hint Resolve mult_Nmult_Umult.
```

Lemma *minus_Nmult_distr* : $\forall$ *n m x*,
Nmult_def n x $\rightarrow$ (*n* - *m*) */ *x*$\equiv$ (*n* */ *x*) - (*m* */ *x*).

Lemma *Nmult_Uminus_distr* : $\forall$ *n x y*,
Nmult_def n x $\rightarrow$ *n* */ (*x* - *y*) $\equiv$ (*n* */ *x*) - (*n* */ *y*).
Hint Resolve *minus_Nmult_distr Nmult_Uminus_distr*.

Lemma *Umult_Unth* : $\forall$ *n m*, [1/]1+*n* $\times$ [1/]1+*m* $\equiv$ [1/]1+(*n*+*m*+*n*$\times$*m*).
Hint Resolve *Umult_Unth*.

Lemma *Umult_Nnth* : $\forall$ *n m*,
(0 < *n*)%*nat* $\rightarrow$ (0 < *m*)%*nat* $\rightarrow$ [1/]*n* $\times$ [1/]*m* $\equiv$ [1/](*n*$\times$*m*)%*nat*.
Hint Resolve *Umult_Nnth*.

Lemma *Nnth_le_compat* : $\forall$ *n m*, (*n* $\leq$ *m*)%*nat* $\rightarrow$ [1/]*m* $\leq$ [1/]*n*.
Hint Resolve *Nnth_le_compat*.

Lemma *Nnth_le_equiv* : $\forall$ *n m*, (0 < *n*)%*nat* $\rightarrow$ (0 < *m*)%*nat* $\rightarrow$ ([1/]*n* $\leq$ [1/]*m* $\leftrightarrow$ *m* $\leq$ *n*).

Lemma *Nnth_eq_equiv* : $\forall$ *n m*, (0 < *n*)%*nat* $\rightarrow$ (0 < *m*)%*nat* $\rightarrow$ ([1/]*n* $\equiv$ [1/]*m* $\leftrightarrow$ *m* = *n*).

Lemma *half_Unth_eq* : $\forall$ *n*, $\frac{1}{2}$ $\times$ [1/]1+*n* $\equiv$ [1/]1+(2***n*+1).

Lemma *twice_half* : $\forall$ *p*, [1/]1+(2 $\times$ *p* + 1) + [1/]1+(2 $\times$ *p* + 1) $\equiv$ [1/]1+*p*.

Lemma *Nmult_def_lt* : $\forall$ *n x*, *n* */ *x* < 1 $\rightarrow$ *Nmult_def n x*.

Hint Immediate *Nmult_def_lt*.

4.26 Conversion from booleans to U

Definition *B2U* :*MF bool* := fun (*b*:*bool*) $\Rightarrow$ if *b* then 1 else 0.

Definition *NB2U* :*MF bool* := fun (*b*:*bool*) $\Rightarrow$ if *b* then 0 else 1.

Lemma *B2Uinv* : *NB2U* $\equiv$ *finv B2U*.

Lemma *NB2Uinv* : *B2U* $\equiv$ *finv NB2U*.

Hint Resolve *B2Uinv NB2Uinv*.

Lemma *Umult_B2U_andb* : $\forall$ *x y*, (*B2U x*) $\times$ (*B2U y*) $\equiv$ *B2U* (*andb x y*).

Lemma *Uplus_B2U_orb* : $\forall$ *x y*, (*B2U x*) + (*B2U y*) $\equiv$ *B2U* (*orb x y*).

4.27 Particular sequences

pmin p n = *p* - $\frac{1}{2}$ ^ *n*

Definition *pmin* (*p*:*U*) (*n*:*nat*) := *p* - ($\frac{1}{2}$ ^ *n*).

Add *Morphism pmin* with *signature Oeq* $\Longrightarrow$*eq* $\Longrightarrow$*Oeq* as *pmin_eq_compat*.
Save.

4.27.1 Properties of *pmin*

Lemma *pmin_esp_S* : $\forall$ *p n*, *pmin* (*p* & *p*) *n* $\equiv$ *pmin p* (*S n*) & *pmin p* (*S n*).

Lemma *pmin_esp_le* : $\forall$ *p n*, *pmin p* (*S n*) $\leq$ $\frac{1}{2}$ $\times$ (*pmin* (*p* & *p*) *n*) + $\frac{1}{2}$.

Lemma *pmin_plus_eq* : $\forall$ *p n*, *p* $\leq$ $\frac{1}{2}$ $\rightarrow$ *pmin p* (*S n*) $\equiv$ $\frac{1}{2}$ $\times$ (*pmin* (*p* + *p*) *n*).

Lemma *pmin_0* : $\forall$ *p*:*U*, *pmin p O* $\equiv$ 0.

Lemma *pmin_le* : $\forall$ (*p*:*U*) (*n*:*nat*), *p* - ([1/]1+*n*) $\leq$ *pmin p n*.

Hint Resolve *pmin_0 pmin_le*.

Lemma *pmin_le_compat* : $\forall$ *p* (*n m* : *nat*), *n* $\leq$ *m* $\rightarrow$ *pmin p n* $\leq$ *pmin p m*.
Hint Resolve *pmin_le_compat*.

`Instance` *pmin_mon* : ∀ *p*, *monotonic* (*pmin p*).
`Save`.

`Definition` *Pmin* (*p*:*U*) :*nat -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]*(*U1min n*) + $\frac{1}{2}$.

`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*) ≤ $\frac{1}{2}$.
`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*.

4.27.2 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_half* : ∀ *n*, (*O*<*n*)%*nat* → [1/]1+(*pred* (*n*+*n*)) ≡ $\frac{1}{2}$ × [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*.

`Section` *Sequence*.
`Variable` *k* : *U*.
`Hypothesis` *kless1* : *k* < 1.

`Lemma` *Ult_one_inv_zero* : ¬ 0 ≡ [1-]*k*.
`Hint Resolve` *Ult_one_inv_zero*.

`Lemma` *Umult_simpl_zero* : ∀ *x*, *x* ≤ *k* × *x* → *x* ≡ 0.

`Lemma` *Umult_simpl_one* : ∀ *x*, *k* × *x* + [1-]*k* ≤ *x* → *x* ≡ 1.

`Lemma` *bary_le_compat* : ∀ *k' x y*, *x* ≤ *y* → *k* ≤ *k'* → *k'* × *x* + [1-]*k'* × *y* ≤ *k* × *x* + [1-]*k* × *y*.

`Lemma` *bary_one_le_compat* : ∀ *k' x*, *k* ≤ *k'* → *k'* × *x* + [1-]*k'* ≤ *k* × *x* + [1-]*k*.

```
Lemma glb_exp_0 : glb (UExp k) ≡ 0.

Instance Uinvexp_mon : monotonic (fun n ⇒ [1-]k ^ n).
Save.

Lemma lub_inv_exp_1 : mlub (fun n ⇒ [1-]k ^ n) ≡ 1.

End Sequence.
Hint Resolve glb_exp_0 lub_inv_exp_1 bary_one_le_compat bary_le_compat.
```

4.28 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_zero_left x)
 | ⊢ context [(Uminus ?x 0)] ⇒ setoid_rewrite (Uminus_zero_right x)
 | ⊢ context [(Uminus ?x 1)] ⇒ setoid_rewrite (Uminus_one_right x)
 | ⊢ context [(Uminus ?x ?x)] ⇒ setoid_rewrite (Uminus_eq x)
 | ⊢ context [[1/2] + [1/2]] ⇒ setoid_rewrite Unth_one_plus
 | ⊢ context [([1/2] × ?x + 1/2 × ?x)] ⇒ setoid_rewrite (Unth_one_refl x)
 | ⊢ context [[1-][1/2]] ⇒ setoid_rewrite ← Unth_one
 | ⊢ context [([1-] ([1-] ?x))] ⇒ setoid_rewrite (Uinv_inv x)
 | ⊢ context [ ?x + ([1-] ?x)] ⇒ setoid_rewrite (Uinv_opp_right x)
 | ⊢ context [ ([1-]?x) + ?x ] ⇒ setoid_rewrite (Uinv_opp_left x)
 | ⊢ context [([1-] 1)] ⇒ setoid_rewrite Uinv_one
 | ⊢ context [([1-] 0)] ⇒ setoid_rewrite Uinv_zero
 | ⊢ context [([1/]1+O)] ⇒ setoid_rewrite Unth_zero
 | ⊢ context [(0/?x)] ⇒ setoid_rewrite (Udiv_zero x)
 | ⊢ context [(?x/1)] ⇒ setoid_rewrite (Udiv_one x)
 | ⊢ context [(?x/0)] ⇒ setoid_rewrite (Udiv_by_zero x); [idtac|reflexivity]
 | ⊢ 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
 | ⊢ context [(Nmult 1 ?x)] ⇒ setoid_rewrite Nmult_1
 | ⊢ context [(Nmult ?n 0)] ⇒ setoid_rewrite Nmult_zero
 | ⊢ context [(sigma ?f O)] ⇒ setoid_rewrite sigma_0
 | ⊢ context [(sigma ?f (S O))] ⇒ setoid_rewrite sigma_1
 | ⊢ (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) (Uminus ?x ?z)) ⇒ apply Uminus_le_compat_right
 | ⊢ (Ole (Uminus ?x ?z) (Uminus ?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.

Ltac Ucompute :=
 first [setoid_rewrite Uplus_zero_left |
         setoid_rewrite Uplus_zero_right |
         setoid_rewrite Uplus_one_left |
         setoid_rewrite Uplus_one_right |
         setoid_rewrite Umult_zero_left |
         setoid_rewrite Umult_zero_right |
         setoid_rewrite Umult_one_left |
         setoid_rewrite Umult_one_right |
         setoid_rewrite Uesp_zero_left |
         setoid_rewrite Uesp_zero_right |
         setoid_rewrite Uesp_one_left |
         setoid_rewrite Uesp_one_right |
         setoid_rewrite Uminus_zero_left |
         setoid_rewrite Uminus_zero_right |
         setoid_rewrite Uminus_one_right |
         setoid_rewrite Uinv_inv |
         setoid_rewrite Uinv_opp_right |
         setoid_rewrite Uinv_opp_left |
         setoid_rewrite Uinv_one |
         setoid_rewrite Uinv_zero |
         setoid_rewrite Unth_zero |
         setoid_rewrite Uexp_0 |
         setoid_rewrite Uexp_1 |
         (setoid_rewrite Uexp_zero; [idtac|omega]) |
         setoid_rewrite Uexp_one |
         setoid_rewrite Nmult_0 |
         setoid_rewrite Nmult_1 |
         setoid_rewrite Nmult_zero |
```

```
        setoid_rewrite sigma_0 |
        setoid_rewrite sigma_1
].
```

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].

`Hint Resolve` *half_square half_cube three_quarter_decomp*.

`Lemma` *half_dec_mult*
: $\forall\ p,\ p \le \frac{1}{2} \to ([1/2]+p) \times ([1/2]\text{-}p) \equiv \frac{1}{4}\ \text{-}\ (p \times p)$.

`Lemma` *half_Ult_Umult_Uinv* :
$\forall\ p,\ p < \frac{1}{2} \to p \times [1\text{-}]p < \frac{1}{4}$.
`Hint Resolve` *half_Ult_Umult_Uinv*.

`Lemma` *half_Ule_Umult_Uinv* :
$\forall\ p,\ p \le \frac{1}{2} \to p \times [1\text{-}]p \le \frac{1}{4}$.
`Hint Resolve` *half_Ule_Umult_Uinv*.

`Lemma` *Ult_Umult_Uinv* :
$\forall\ p,\ \neg\ p \equiv \frac{1}{2} \to p \times [1\text{-}]p < \frac{1}{4}$.

`Lemma` *Ule_Umult_Uinv* : $\forall\ p,\ p \times [1\text{-}]p \le \frac{1}{4}$.

Equality is not true, even for monotonic sequences fot instance n/m

`Lemma` *Ulub_Uglb_exch_le* : $\forall\ f : nat \to nat \to U$,
Ulub (`fun` $n \Rightarrow$ *Uglb* (`fun` $m \Rightarrow f\ n\ m$)) $\le$ *Uglb* (`fun` $m \Rightarrow$ *Ulub* (`fun` $n \Rightarrow f\ n\ m$)).

4.29 Intervals

4.29.1 Definition

`Record` *IU* : `Type` := *mk_IU* {*low*:*U*; *up*:*U*; *proper*:*low* $\le$ *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*).

4.29.2 Relations

`Definition` *Iin* (*x*:*U*) (*I*:*IU*) := *low* *I* $\le$ *x* $\wedge$ *x* $\le$ *up* *I*.

`Definition` *Iincl* *I* *J* := *low* *J* $\le$ *low* *I* $\wedge$ *up* *I* $\le$ *up* *J*.

`Definition` *Ieq* *I* *J* := *low* *I* $\equiv$ *low* *J* $\wedge$ *up* *I* $\equiv$ *up* *J*.
`Hint Unfold` *Iin Iincl Ieq*.

4.29.3 Properties

`Lemma` *Iin_low* : $\forall$ *I*, *Iin* (*low* *I*) *I*.

`Lemma` *Iin_up* : $\forall$ *I*, *Iin* (*up* *I*) *I*.

`Hint Resolve` *Iin_low Iin_up*.

```
Lemma Iin_singl_elim : ∀ x y, Iin x (singl y) → x ≡ y.
Lemma Iin_inf_elim : ∀ x y, Iin x (inf y) → x ≤ y.
Lemma Iin_sup_elim : ∀ x y, Iin x (sup y) → y ≤ x.
Lemma Iin_singl_intro : ∀ x y, x ≡ y → Iin x (singl y).
Lemma Iin_inf_intro : ∀ x y, x ≤ y → Iin x (inf y).
Lemma Iin_sup_intro : ∀ x y, y ≤ x → 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 Iin_class : ∀ I x, class (Iin x I).
Lemma Iincl_class : ∀ I J, class (Iincl I J).
Lemma Ieq_class : ∀ I J, class (Ieq I J).
Hint Resolve Iin_class Iincl_class Ieq_class.
Lemma Iincl_in : ∀ I J, Iincl I J → ∀ x, Iin x I → Iin x J.
Lemma Iincl_low : ∀ I J, Iincl I J → low J ≤ low I.
Lemma Iincl_up : ∀ I J, Iincl I J → up I ≤ up J.
Hint Immediate Iincl_low Iincl_up.
Lemma Iincl_refl : ∀ I, Iincl I I.
Hint Resolve Iincl_refl.
Lemma Iincl_trans : ∀ I J K, Iincl I J → Iincl J K → Iincl I K.
Instance IUord : ord IU := {Oeq := fun I J ⇒ Ieq I J; Ole := fun I J ⇒ Iincl J I}.
Defined.
Lemma low_le_compat : ∀ I J:IU, I ≤ J → low I ≤ low J.
Lemma up_le_compat : ∀ I J : IU, I ≤ J → up J ≤ up I.
Instance low_mon : monotonic low.
Save.
Definition Low : IU -m> U := mon low.
Instance up_mon : monotonic (o2:=Iord U) up.
Save.
Definition Up : IU -m→ U := mon (o2:=Iord U) up.
Lemma Ieq_incl : ∀ I J, Ieq I J → Iincl I J.
Lemma Ieq_incl_sym : ∀ I J, Ieq I J → Iincl J I.
Hint Immediate Ieq_incl Ieq_incl_sym.
Lemma lincl_eq_compat : ∀ I J K L,
    Ieq I J → Iincl J K → Ieq K L → Iincl I L.
Lemma lincl_eq_trans : ∀ I J K,
    Iincl I J → Ieq J K → Iincl I K.
Lemma Ieq_incl_trans : ∀ I J K,
    Ieq I J → Iincl J K → Iincl I K.
Lemma Iincl_antisym : ∀ I J, Iincl I J → Iincl J I → Ieq I J.
Hint Immediate Iincl_antisym.
Lemma Ieq_refl : ∀ I, Ieq I I.
Hint Resolve Ieq_refl.
Lemma Ieq_sym : ∀ I J, Ieq I J → Ieq J I.
Hint Immediate Ieq_sym.
Lemma Ieq_trans : ∀ I J K, Ieq I J → Ieq J K → Ieq I K.
```

Lemma *Isingl_eq* : $\forall$ *x y*, *Iincl* (*singl x*) (*singl y*) $\rightarrow$ *x*$\equiv$*y*.
Hint Immediate *Isingl_eq*.

Lemma *Iincl_full* : $\forall$ *I*, *Iincl I full*.
Hint Resolve *Iincl_full*.

4.29.4 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* : $\forall$ *I J*, *low* (*Iplus I J*)=*low I* + *low J*.

Lemma *up_Iplus* : $\forall$ *I J*, *up* (*Iplus I J*)=*up I* + *up J*.

Lemma *Iplus_in* : $\forall$ *I J x y*, *Iin x I* $\rightarrow$ *Iin y J* $\rightarrow$ *Iin* (*x*+*y*) (*Iplus I J*).

Lemma *lplus_in_elim* :
$\forall$ *I J z*, *low I* $\leq$ [1-]*up J* $\rightarrow$ *Iin z* (*Iplus I J*)
$\rightarrow$ *exc* (fun *x* $\Rightarrow$ *Iin x I* $\wedge$
exc (fun *y* $\Rightarrow$ *Iin y J* $\wedge$ *z*$\equiv$*x*+*y*)).

Definition *Imult I J* := *mk_IU* (*low I* $\times$ *low J*) (*up I* $\times$ *up J*)
(*Umult_le_compat* _ _ _ _ (*proper I*) (*proper J*)).

Lemma *low_Imult* : $\forall$ *I J*, *low* (*Imult I J*) = *low I* $\times$ *low J*.

Lemma *up_Imult* : $\forall$ *I J*, *up* (*Imult I J*) = *up I* $\times$ *up J*.

Definition *Imultk p I* := *mk_IU* (*p* $\times$ *low I*) (*p* $\times$ *up I*) (*Umult_le_compat_right p* _ _ (*proper I*)).

Lemma *low_Imultk* : $\forall$ *p I*, *low* (*Imultk p I*) = *p* $\times$ *low I*.

Lemma *up_Imultk* : $\forall$ *p I*, *up* (*Imultk p I*) = *p* $\times$ *up I*.

Lemma *Imult_in* : $\forall$ *I J x y*, *Iin x I* $\rightarrow$ *Iin y J* $\rightarrow$ *Iin* (*x*$\times$*y*) (*Imult I J*).

Lemma *Imultk_in* : $\forall$ *p I x* , *Iin x I* $\rightarrow$ *Iin* (*p*$\times$*x*) (*Imultk p I*).

4.29.5 Limits of intervals

Definition *Ilim* : $\forall$ *I*: *nat -m> IU*, *IU*.
Defined.

Lemma *low_lim* : $\forall$ (*I*:*nat -m> IU*), *low* (*Ilim I*) = *lub* (*Low* @ *I*).

Lemma *up_lim* : $\forall$ (*I*:*nat -m> IU*), *up* (*Ilim I*) = *glb* (*Up* @ *I*).

Lemma *lim_Iincl* : $\forall$ (*I*:*nat -m> IU*) *n*, *Iincl* (*Ilim I*) (*I n*).
Hint Resolve *lim_Iincl*.

Lemma *Iincl_lim* : $\forall$ *J* (*I*:*nat -m>IU*), ($\forall$ *n*, *Iincl J* (*I n*)) $\rightarrow$ *Iincl J* (*Ilim I*).

Lemma *IIim_incl_stable* : $\forall$ (*I J*:*nat -m> IU*), ($\forall$ *n*, *Iincl* (*I n*) (*J n*)) $\rightarrow$ *Iincl* (*Ilim I*) (*Ilim J*).
Hint Resolve *IIim_incl_stable*.

Instance *IUcpo* : *cpo IU* := {*D0*:=*full*; *lub*:=*Ilim*}.
Defined.

4.30 Limits inf and sup

Definition *fsup* (*f*:*nat* $\rightarrow$ *U*) (*n*:*nat*) := *Ulub* (fun *k* $\Rightarrow$ *f* (*n*+*k*)%*nat*).

Definition *finf* (*f*:*nat* $\rightarrow$ *U*) (*n*:*nat*) := *Uglb* (fun *k* $\Rightarrow$ *f* (*n*+*k*)%*nat*).

Lemma *fsup_incr* : $\forall$ (*f*:*nat* $\rightarrow$ *U*) *n*, *fsup f* (*S n*) $\leq$ *fsup f n*.
Hint Resolve *fsup_incr*.

Lemma *finf_incr* : $\forall$ (*f*:*nat* $\rightarrow$ *U*) *n*, *finf f n* $\leq$ *finf f* (*S n*).

Hint Resolve finf_incr.

Instance fsup_mon : ∀ f, monotonic (o2:=Iord U) (fsup f).
Save.

Instance finf_mon : ∀ f, monotonic (finf f).
Save.

Definition Fsup (f:nat → U) : nat -m→ U := mon (fsup f).
Definition Finf (f:nat → U) : nat -m> U := mon (finf f).

Lemma fn_fsup : ∀ f n, f n ≤ fsup f n.
Hint Resolve fn_fsup.

Lemma finf_fn : ∀ f n, finf f n ≤ f n.
Hint Resolve finf_fn.

Definition limsup f := glb (Fsup f).
Definition liminf f := lub (Finf f).

Lemma le_liminf_sup : ∀ f, liminf f ≤ limsup f.

Hint Resolve le_liminf_sup.

Definition has_lim f := limsup f ≤ liminf f.

Lemma eq_liminf_sup : ∀ f, has_lim f→ liminf f ≡ limsup f.

Definition cauchy f := ∀ (p:nat), exc (fun M:nat ⇒ ∀ n m,
(M ≤ n)%nat → (M ≤ m)%nat → f n ≤ f m + [1/2]^p).

Definition is_limit f (l:U) := ∀ (p:nat), exc (fun M:nat ⇒ ∀ n,
(M ≤ n)%nat → f n ≤ l + [1/2]^p ∧ l ≤ f n + [1/2]^p).

Lemma cauchy_lim : ∀ f, cauchy f → is_limit f (limsup f).

Lemma has_limit_cauchy : ∀ f l, is_limit f l → cauchy f.

Lemma limit_le_unique : ∀ f l1 l2, is_limit f l1 → is_limit f l2 → l1 ≤ l2.

Lemma limit_unique : ∀ f l1 l2, is_limit f l1 → is_limit f l2 → l1 ≡ l2.
Hint Resolve limit_unique.

Lemma has_limit_compute : ∀ f l, is_limit f l → is_limit f (limsup f).

Lemma limsup_eq_mult : ∀ k (f : nat → U),
limsup (fun n ⇒ k × f n) ≡ k × limsup f.

Lemma liminf_eq_mult : ∀ k (f : nat → U),
liminf (fun n ⇒ k × f n) ≡ k × liminf f.

Lemma limsup_eq_plus_cte_right : ∀ k (f : nat → U),
limsup (fun n ⇒ (f n) + k) ≡ limsup f + k.

Lemma liminf_eq_plus_cte_right : ∀ k (f : nat → U),
liminf (fun n ⇒ (f n) + k) ≡ liminf f + k.

Lemma limsup_le_plus : ∀ (f g: nat → U),
limsup (fun x ⇒ f x + g x) ≤ limsup f + limsup g.

Lemma liminf_le_plus : ∀ (f g: nat → U),
liminf f + liminf g ≤ liminf (fun x ⇒ f x + g x).

Hint Resolve liminf_le_plus limsup_le_plus.

Lemma limsup_le_compat : ∀ f g : nat → U, f ≤ g → limsup f ≤ limsup g.

Lemma liminf_le_compat : ∀ f g : nat → U, f ≤ g → liminf f ≤ liminf g.

Hint Resolve limsup_le_compat liminf_le_compat.

Lemma limsup_eq_compat : ∀ f g : nat → U, f ≡ g → limsup f ≡ limsup g.

Lemma liminf_eq_compat : ∀ f g : nat → U, f ≡ g → liminf f ≡ liminf g.

Hint Resolve *liminf_eq_compat limsup_eq_compat*.

Lemma *limsup_inv* : ∀ *f* : *nat* → *U*, *limsup* (fun *x* ⇒ [1-]*f x*) ≡ [1-] *liminf f*.

Lemma *liminf_inv* : ∀ *f* : *nat* → *U*, *liminf* (fun *x* ⇒ [1-]*f x*) ≡ [1-] *limsup f*.

Hint Resolve *limsup_inv liminf_inv*.

4.31 Limits of arbitrary sequences

Lemma *liminf_incr* : ∀ *f*:*nat -m> U*, *liminf f* ≡ *lub f*.

Lemma *limsup_incr* : ∀ *f*:*nat -m> U*, *limsup f* ≡ *lub f*.

Lemma *has_limit_incr* : ∀ *f*:*nat -m> U*, *has_lim f*.

Lemma *liminf_decr* : ∀ *f*:*nat -m→ U*, *liminf f* ≡ *glb f*.

Lemma *limsup_decr* : ∀ *f*:*nat -m→ U*, *limsup f* ≡ *glb f*.

Lemma *has_limit_decr* : ∀ *f*:*nat -m→ U*, *has_lim f*.

Lemma *has_limit_sum* : ∀ *f g*: *nat* → *U*, *has_lim f* → *has_lim g* → *has_lim* (fun *x* ⇒ *f x* + *g x*).

Lemma *has_limit_inv* : ∀ *f* : *nat* → *U*, *has_lim f* → *has_lim* (fun *x* ⇒ [1-]*f x*).

Lemma *has_limit_cte* : ∀ *c*, *has_lim* (fun *n* ⇒ *c*).

4.32 Definition and properties of series : infinite sums

Definition *serie* (*f* : *nat* → *U*) : *U* := *lub* (*sigma f*).

Lemma *serie_le_compat* : ∀ (*f g*: *nat* → *U*),
 (∀ *k*, *f k* ≤ *g k*) → *serie f* ≤ *serie g*.

Lemma *serie_eq_compat* : ∀ (*f g*: *nat* → *U*),
 (∀ *k*, *f k* ≡ *g k*) → *serie f* ≡ *serie g*.

Lemma *serie_sigma_lift* : ∀ (*f* :*nat* → *U*) (*n*:*nat*),
 serie f ≡ *sigma f n* + *serie* (fun *k* ⇒ *f* (*n* + *k*)%*nat*).

Lemma *serie_S_lift* : ∀ (*f* :*nat* → *U*),
 serie f ≡ *f O* + *serie* (fun *k* ⇒ *f* (*S k*)).

Lemma *serie_zero* : ∀ *f*, (∀ *k*, *f k* ==0) → *serie f* ==0.

Lemma *serie_not_zero* : ∀ *f k*, 0 < *f k* → 0 < *serie f*.

Lemma *serie_zero_elim* : ∀ *f*, *serie f* ≡ 0 → ∀ *k*, *f k* ==0.

Hint Resolve *serie_eq_compat serie_le_compat serie_zero*.

Lemma *serie_le* : ∀ *f k*, *f k* ≤ *serie f*.

Lemma *serie_minus_incr* : ∀ *f* :*nat -m> U*, *serie* (fun *k* ⇒ *f* (*S k*) - *f k*) ≡ *lub f* - *f O*.

Lemma *serie_minus_decr* : ∀ *f* : *nat -m→ U*,
 serie (fun *k* ⇒ *f k* - *f* (*S k*)) ≡ *f O* - *glb f*.

Lemma *serie_plus* : ∀ (*f g* : *nat* → *U*),
 serie (fun *k* ⇒ (*f k*) + (*g k*)) ≡ *serie f* + *serie g*.

Definition *wretract* (*f* : *nat* → *U*) := ∀ *k*, *f k* ≤ [1-] (*sigma f k*).

Lemma *retract_wretract* : ∀ *f*, (∀ *n*, *retract f n*) → *wretract f*.

Lemma *wretract_retract* : ∀ *f*, *wretract f* → ∀ *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.

Definition fun_cte n (a:U) : nat → U
      := fun p ⇒ if eq_nat_dec p n then a else 0.

Lemma fun_cte_eq : ∀ n a, fun_cte n a n = a.

Lemma fun_cte_zero : ∀ n a p, p ≠ n → fun_cte n a p = 0.

Lemma sigma_cte_eq : ∀ n a p, (n < p)%nat → sigma (fun_cte n a) p ≡ a.
Hint Resolve sigma_cte_eq.

Lemma serie_cte_eq : ∀ n a, serie (fun_cte n a) ≡ a.

Section PartialPermutationSerieLe.
Variables f g : nat → U.

Variable s : nat → nat → Prop.
Hypothesis s_dec : ∀ i j, {s i j}+{~s i j}.

Hypothesis s_inj : ∀ i j k : nat, s i k → s j k → i = j.
Hypothesis s_dom : ∀ i, ¬ f i ≡ 0 → ∃ j, s i j.

Hypothesis f_g_perm : ∀ i j, s i j → f i ≡ g j.

Lemma serie_perm_rel_le : serie f ≤ serie g.

End PartialPermutationSerieLe.

Section PartialPermutationSerieEq.
Variables f g : nat → U.

Variable s : nat → nat → Prop.
Hypothesis s_dec : ∀ i j, {s i j}+{~s i j}.
Hypothesis s_fun : ∀ i j k : nat, s i j → s i k → j = k.
Hypothesis s_inj : ∀ i j k : nat, s i k → s j k → i = j.
Hypothesis s_surj : ∀ j, ¬ g j ≡ 0 → ∃ i, s i j.
Hypothesis s_dom : ∀ i, ¬ f i ≡ 0 → ∃ j, s i j.
Hypothesis f_g_perm : ∀ i j, s i j → f i ≡ g j.

Lemma serie_perm_rel_eq : serie f ≡ serie g.

End PartialPermutationSerieEq.

Section PermutationSerie.
```

```
Variable s : nat → nat.
Hypothesis s_inj : ∀ i j : nat, s i = s j → i = j.
Hypothesis s_surj : ∀ j, ∃ i, s i = j.

Variable f : nat → U.

Lemma serie_perm_le : serie (fun i ⇒ f (s i)) ≤ serie f.

Lemma serie_perm_eq : serie f ≡ serie (fun i ⇒ f (s i)).

End PermutationSerie.
Hint Resolve serie_perm_eq serie_perm_le.

Section SerieProdRel.
Variable f : nat → U.
Variable g : nat → nat → U.
Variable s : nat → nat → nat → Prop.
Hypothesis s_dec : ∀ k n m, {s k n m}+{˜ s k n m}.
Hypothesis s_fun1 : ∀ k n1 m1 n2 m2, s k n1 m1 → s k n2 m2 → n1 = n2.
Hypothesis s_fun2 : ∀ k n1 m1 n2 m2, s k n1 m1 → s k n2 m2 → m1 = m2.
Hypothesis s_inj : ∀ k1 k2 n m, s k1 n m → s k2 n m → k1 = k2.
Hypothesis s_surj : ∀ n m, ¬ g n m ≡ 0 → ∃ k, s k n m.
Hypothesis f_g_perm : ∀ k n m, s k n m → f k ≡ g n m.

Section SPR.

Hypothesis s_dom : ∀ k, ¬ f k ≡ 0 → ∃ n, ∃ m, s k n m.

Lemma serie_le_rel_prod : serie f ≤ serie (fun n ⇒ serie (g n)).
End SPR.

Variable s_fst : nat → nat.
Hypothesis s_fst_ex : ∀ k, ∃ m, s k (s_fst k) m.

Lemma s_dom : ∀ k, ∃ n, ∃ m, s k n m.
Hint Resolve s_dom.

Lemma serie_rel_prod_le : serie (fun n ⇒ serie (g n)) ≤ serie f.

Lemma serie_rel_prod_eq : serie f ≡ serie (fun n ⇒ serie (g n)).

End SerieProdRel.

Section SerieProd.
Variable f : (nat × nat) → U.
Variable s : nat → nat × nat.
Variable s_inj : ∀ n m, s n = s m → n = m.
Variable s_surj : ∀ m, ∃ n, s n = m.

Lemma serie_enum_prod_eq : serie (fun k ⇒ f (s k)) ≡ serie (fun n ⇒ serie (fun m ⇒ f (n,m))).

End SerieProd.
Hint Resolve serie_enum_prod_eq.
```

5 Monads.v: Monads for randomized constructions

```
Require Export Uprop.
```

5.1 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 $star$: $\forall$ (A B:Type), M $A \rightarrow (A \rightarrow M$ $B) \rightarrow M$ B.
Defined.

Lemma $star\_simpl$: $\forall$ (A B:Type) (a:M A) (F:$A \rightarrow M$ B)(f:MF B),
$star$ a F f = a (fun $x \Rightarrow F$ x f).

5.2 Properties of monadic operators

Lemma $law1$: $\forall$ (A B:Type) (x:A) (F:$A \rightarrow M$ B) (f:MF B), $star$ ($unit$ x) F $f \equiv F$ x f.

Lemma $law2$:
$\forall$ (A:Type) (a:M A) (f:MF A), $star$ a (fun x:$A \Rightarrow unit$ x) $f \equiv a$ (fun x:$A \Rightarrow f$ x).

Lemma $law3$:
$\forall$ (A B C:Type) (a:M A) (F:$A \rightarrow M$ B) (G:$B \rightarrow M$ C)
(f:MF C), $star$ ($star$ a F) G $f \equiv star$ a (fun x:$A \Rightarrow star$ (F x) G) f.

5.3 Properties of distributions

5.3.1 Expected properties of measures

Definition $stable\_inv$ (A:Type) (m:M A) : Prop := $\forall$ f :MF A, m ($finv$ f) $\leq$ [1-] (m f).

Definition $stable\_plus$ (A:Type) (m:M A) : Prop :=
$\forall$ f g:MF A, $fplusok$ f $g \rightarrow m$ ($fplus$ f g) $\equiv$ (m f) + (m g).

Definition le_plus (A:Type) (m:M A) : Prop :=
$\forall$ f g:MF A, $fplusok$ f $g \rightarrow$ (m f) + (m g) $\leq m$ ($fplus$ f g).

Definition le_esp (A:Type) (m:M A) : Prop :=
$\forall$ f g: MF A, (m f) & (m g) $\leq m$ ($fesp$ f g).

Definition le_plus_cte (A:Type) (m:M A) : Prop :=
$\forall$ (f : MF A) (k:U), m ($fplus$ f ($fcte$ A k)) $\leq m$ f + k.

Definition $stable\_mult$ (A:Type) (m:M A) : Prop :=
$\forall$ (k:U) (f:MF A), m ($fmult$ k f) $\equiv k \times$ (m f).

5.3.2 Stability for equality

Lemma $stable\_minus\_distr$: $\forall$ (A:Type) (m:M A),
$stable\_plus$ $m \rightarrow stable\_inv$ $m \rightarrow$
$\forall$ (f g : MF A), $g \leq f \rightarrow m$ ($fminus$ f g) $\equiv m$ f - m g.

Hint Resolve $stable\_minus\_distr$.

Lemma inv_minus_distr : $\forall$ (A:Type) (m:M A),
$stable\_plus$ $m \rightarrow stable\_inv$ $m \rightarrow$
$\forall$ (f : MF A), m ($finv$ f) $\equiv m$ ($fone$ A) - m f.
Hint Resolve inv_minus_distr.

Lemma le_minus_distr : $\forall$ (A : Type)(m:M A),
$\forall$ (f g:$A \rightarrow U$), m ($fminus$ f g) $\leq m$ f.
Hint Resolve le_minus_distr.

Lemma le_plus_distr : $\forall$ (A : Type)(m:M A),
$stable\_plus$ $m \rightarrow stable\_inv$ $m \rightarrow \forall$ (f g:MF A), m ($fplus$ f g) $\leq m$ f + m g.
Hint Resolve le_plus_distr.

Lemma le_esp_distr : $\forall$ (A : Type) (m:M A),
$stable\_plus$ $m \rightarrow stable\_inv$ $m \rightarrow le\_esp$ m.

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

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

Lemma *unit_monotonic* : ∀ (*A*:Type) (*x*:*A*) (*f g* : *MF A*),
  f ≤ *g* → *unit x f* ≤ *unit x g*.

Lemma *star_monotonic* : ∀ (*A B*:Type) (*m*:*M A*) (*F*:*A* → *M B*) (*f g* : *MF B*),
  f ≤ *g* → *star m F f* ≤ *star m F g*.

Lemma *star_le_compat* : ∀ (*A B*:Type) (*m1 m2*:*M A*) (*F1 F2*:*A* → *M B*),
  m1 ≤ *m2* → *F1* ≤ *F2* → *star m1 F1* ≤ *star m2 F2*.
Hint Resolve *star_le_compat*.

5.3.3 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*).

5.3.4 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*).

5.3.5 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*).

5.3.6 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*).

6 Probas.v: The monad for distributions

Require Export *Monads*.

6.1 Definition of distribution

Distributions are monotonic measure functions such that

- μ (1-f) ≤ 1 - μ f
- $f \leq 1 - g \Rightarrow \mu\ (f+g) \equiv \mu\ f + \mu\ g$
- $\mu\ (k \times f) = k \times \mu\ (f)$

- μ (*lub f_n*) $\leq$ *lub* μ (*f_n*)

```
Record distr (A:Type) : Type :=
  {μ : M A;
   mu_stable_inv : stable_inv μ;
   mu_stable_plus : stable_plus μ;
   mu_stable_mult : stable_mult μ;
   mu_continuous : continuous μ}.
Hint Resolve mu_stable_plus mu_stable_inv mu_stable_mult mu_continuous.
```

6.2 Properties of measures

```
Lemma mu_monotonic : ∀ (A : Type)(m: distr A), monotonic (μ m).
Hint Resolve mu_monotonic.
Implicit Arguments mu_monotonic [A].

Lemma mu_stable_eq : ∀ (A : Type)(m: distr A), stable (μ m).
Hint Resolve mu_stable_eq.
Implicit Arguments mu_stable_eq [A].

Lemma mu_zero : ∀ (A : Type)(m: distr A), μ m (fzero A) ≡ 0.
Hint Resolve mu_zero.

Lemma mu_zero_eq : ∀ (A : Type)(m: distr A) f,
   (∀ x, f x ≡ 0) → μ m f ≡ 0.

Lemma mu_one_inv : ∀ (A : Type)(m:distr A),
   μ m (fone A) ≡ 1 → ∀ f, μ m (finv f) ≡ [1-] (μ m f).
Hint Resolve mu_one_inv.

Lemma mu_fplusok : ∀ (A : Type)(m:distr A) f g, fplusok f g →
               μ m f ≤ [1-] μ m g.
Hint Resolve mu_fplusok.

Lemma mu_le_minus : ∀ (A : Type)(m:distr A) (f g:MF A),
     μ m (fminus f g) ≤ μ m f.
Hint Resolve mu_le_minus.

Lemma mu_le_plus : ∀ (A : Type)(m:distr A) (f g:MF A),
     μ m (fplus f g) ≤ μ m f + μ m g.
Hint Resolve mu_le_plus.

Lemma mu_eq_plus : ∀ (A : Type)(m:distr A) (f g:MF A),
     fplusok f g → μ m (fplus f g) ≡ μ m f + μ m g.
Hint Resolve mu_eq_plus.

Lemma mu_plus_zero : ∀ (A : Type)(m:distr A) (f g:MF A),
   μ m f ≡ 0 → μ m g ≡ 0 → μ m (fplus f g) ≡ 0.
Hint Resolve mu_plus_zero.

Lemma mu_plus_pos : ∀ (A : Type)(m:distr A) (f g:MF A),
   0 < μ m (fplus f g) → orc (0 < μ m f) (0 < μ m g).

Lemma mu_fcte : ∀ (A : Type)(m:(distr A)) (c:U),
   μ m (fcte A c) ≡ c × μ m (fone A).
Hint Resolve mu_fcte.

Lemma mu_fcte_le : ∀ (A : Type)(m:distr A) (c:U), μ m (fcte A c) ≤ c.

Lemma mu_fcte_eq : ∀ (A : Type)(m:distr A) (c:U),
   μ m (fone A) ≡ 1 → μ m (fcte A c) ≡ c.

Hint Resolve mu_fcte_le mu_fcte_eq.
```

Lemma mu_cte : $\forall$ (A : Type)(m:($distr\ A$)) (c:U),
$\mu\ m$ (fun _ $\Rightarrow c$) $\equiv c \times \mu\ m$ ($fone\ A$).
Hint Resolve mu_cte.

Lemma mu_cte_le : $\forall$ (A : Type)(m:$distr\ A$) (c:U), $\mu\ m$ (fun _ $\Rightarrow c$) $\leq c$.

Lemma mu_cte_eq : $\forall$ (A : Type)(m:$distr\ A$) (c:U),
$\mu\ m$ ($fone\ A$) $\equiv 1 \rightarrow \mu\ m$ (fun _ $\Rightarrow c$) $\equiv c$.
Hint Resolve mu_cte_le mu_cte_eq.

Lemma $mu\_stable\_mult\_right$: $\forall$ (A : Type)(m:$distr\ A$) (c:U) (f : $MF\ A$),
$\mu\ m$ (fun $x \Rightarrow$ ($f\ x$) $\times c$) $\equiv$ ($\mu\ m\ f$) $\times c$.

Lemma mu_stable_minus : $\forall$ (A:Type) (m:$distr\ A$)($f\ g$: $MF\ A$),
$g \leq f \rightarrow \mu\ m$ (fun $x \Rightarrow f\ x$ - $g\ x$) $\equiv \mu\ m\ f$ - $\mu\ m\ g$.

Lemma mu_inv_minus :
$\forall$ (A:Type) (m:$distr\ A$)(f: $MF\ A$), $\mu\ m$ ($finv\ f$) $\equiv \mu\ m$ ($fone\ A$) - $\mu\ m\ f$.

Lemma $mu\_stable\_le\_minus$: $\forall$ (A:Type) (m:$distr\ A$)($f\ g$: $MF\ A$),
$\mu\ m\ f$ - $\mu\ m\ g \leq \mu\ m$ (fun $x \Rightarrow f\ x$ - $g\ x$).

Lemma $mu\_inv\_minus\_inv$: $\forall$ (A:Type) (m:$distr\ A$)(f: $MF\ A$),
$\mu\ m$ ($finv\ f$) + [1-]($\mu\ m$ ($fone\ A$)) $\equiv$ [1-]($\mu\ m\ f$).

Lemma $mu\_le\_esp\_inv$: $\forall$ (A:Type) (m:$distr\ A$)($f\ g$: $MF\ A$),
([1-]$\mu\ m$ ($finv\ f$)) & $\mu\ m\ g \leq \mu\ m$ ($fesp\ f\ g$).
Hint Resolve $mu\_le\_esp\_inv$.

Lemma $mu\_stable\_inv\_inv$: $\forall$ (A:Type) (m:$distr\ A$)(f : $MF\ A$),
$\mu\ m\ f \leq$ [1-] $\mu\ m$ ($finv\ f$).
Hint Resolve $mu\_stable\_inv\_inv$.

Lemma mu_stable_div : $\forall$ (A:Type) (m:$distr\ A$)(k:U)(f : $MF\ A$),
$\neg$ 0==$k \rightarrow f \leq fcte\ A\ k \rightarrow \mu\ m$ ($fdiv\ k\ f$) $\equiv \mu\ m\ f$ / k.

Lemma $mu\_stable\_div\_le$: $\forall$ (A:Type) (m:$distr\ A$)(k:U)(f : $MF\ A$),
$\neg$ 0==$k \rightarrow \mu\ m$ ($fdiv\ k\ f$) $\leq \mu\ m\ f$ / k.

Lemma mu_le_esp : $\forall$ (A:Type) (m:$distr\ A$)($f\ g$: $MF\ A$),
$\mu\ m\ f$ & $\mu\ m\ g \leq \mu\ m$ ($fesp\ f\ g$).
Hint Resolve mu_le_esp.

Lemma mu_esp_one : $\forall$ (A:Type)(m:$distr\ A$)($f\ g$:$MF\ A$),
$1 \leq \mu\ m\ f \rightarrow \mu\ m\ g \equiv \mu\ m$ ($fesp\ f\ g$).

Lemma mu_esp_zero : $\forall$ (A:Type)(m:$distr\ A$)($f\ g$:$MF\ A$),
$\mu\ m$ ($finv\ f$) $\leq 0 \rightarrow \mu\ m\ g \equiv \mu\ m$ ($fesp\ f\ g$).

Lemma mu_stable_mult2:
$\forall$ (A : Type) (d : $distr\ A$), $\forall$ (k : U)
(f : $MF\ A$), ($\mu\ d$) (fun $x \Rightarrow k \times f\ x$) $\equiv k \times$ ($\mu\ d$) f.

Lemma mu_stable_plus2:
$\forall$ (A : Type) (d : $distr\ A$) ($f\ g$: $MF\ A$),
$fplusok\ f\ g \rightarrow$ ($\mu\ d$) (fun $x \Rightarrow f\ x + g\ x$) $\equiv$ ($\mu\ d$) f + ($\mu\ d$) g.

Lemma mu_fzero_eq : $\forall\ A\ m$, @$\mu\ A\ m$ (fun $x \Rightarrow 0$) $\equiv 0$.

Instance $Odistr$ (A:Type) : ord ($distr\ A$) :=
{Ole := fun ($f\ g$: $distr\ A$) $\Rightarrow \mu\ f \leq \mu\ g$;
Oeq := fun ($f\ g$: $distr\ A$) $\Rightarrow \mu\ f \equiv \mu\ g$}.
Defined.

Probability of termination

Definition $pone\ A$ (m:$distr\ A$) := $\mu\ m$ ($fone\ A$).

Add $Parametric\ Morphism\ A$: ($pone$ (A:=A))

```
    with signature Oeq ==>Oeq as pone_eq_compat.
Save.
Hint Resolve pone_eq_compat.
```

6.3 Monadic operators for distributions

```
Definition Munit : ∀ A:Type, A → distr A.
Defined.

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

Lemma Munit_simpl : ∀ (A:Type) (q:A → U) x, μ (Munit x) q = q x.

Lemma Mlet_simpl : ∀ (A B:Type) (m:distr A) (M:A → distr B) (f:B → U),
     μ (Mlet m M) f = μ m (fun x ⇒ (μ (M x) f)).
```

6.4 Operations on distributions

```
Lemma Munit_eq_compat : ∀ A (x y : A), x = y → Munit x ≡ Munit y.

Lemma Mlet_le_compat : ∀ (A B : Type) (m1 m2:distr A) (M1 M2 : A → distr B),
  m1 ≤ m2 → M1 ≤ M2 → Mlet m1 M1 ≤ Mlet m2 M2.
Hint Resolve Mlet_le_compat.

Add Parametric Morphism (A B : Type) : (Mlet (A:=A) (B:=B))
  with signature Ole ==>Ole ==>Ole
  as Mlet_le_morphism.
Save.

Add Parametric Morphism (A B : Type) : (Mlet (A:=A) (B:=B))
  with signature Ole ==>(@pointwise_relation A (distr B) (@Ole _ _)) ==>Ole
  as Mlet_le_pointwise_morphism.
Save.

Instance Mlet_mon2 : ∀ (A B : Type), monotonic2 (@Mlet A B).
Save.

Definition MLet (A B : Type) : distr A -m> (A → distr B) -m> distr B
               := mon2 (@Mlet A B).

Lemma MLet_simpl0 : ∀ (A B:Type) (m:distr A) (M:A → distr B),
     MLet A B m M = Mlet m M.

Lemma MLet_simpl : ∀ (A B:Type) (m:distr A) (M:A → distr B)(f:B → U),
     μ (MLet A B m M) f = μ m (fun x ⇒ μ (M x) f).

Lemma Mlet_eq_compat : ∀ (A B : Type) (m1 m2:distr A) (M1 M2 : A → distr B),
  m1 ≡ m2 → M1≡M2 → 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 ==>Oeq ==>Oeq
  as Mlet_eq_morphism.
Save.

Add Parametric Morphism (A B : Type) : (Mlet (A:=A) (B:=B))
  with signature Oeq ==>(@pointwise_relation A (distr B) (@Oeq _ _)) ==>Oeq
  as Mlet_Oeq_pointwise_morphism.
Save.

Lemma mu_le_compat : ∀ (A:Type) (m1 m2:distr A),
  m1 ≤ m2 → ∀ f g : A → U, f ≤ g → μ m1 f ≤ μ m2 g.

Lemma mu_eq_compat : ∀ (A:Type) (m1 m2:distr A),
```

$m1 \equiv m2 \to \forall\, f\ g : A \to U, f \equiv g \to \mu\ m1\ f \equiv \mu\ m2\ g.$
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.

Add *Parametric Morphism* (A:Type) (a:*distr* A) : (@μ A a)
with *signature* (@*pointwise_relation* A U (@*eq* _) $\Longrightarrow$*Oeq*) as *mu_distr_eq_morphism*.
Save.

Add *Parametric Morphism* (A:Type) (a:*distr* A) : (@μ A a)
with *signature* (@*pointwise_relation* A U (@*Oeq* _ _) $\Longrightarrow$*Oeq*) as *mu_distr_Oeq_morphism*.
Save.

Add *Parametric Morphism* (A:Type) (a:*distr* A) : (@μ A a)
with *signature* (@*pointwise_relation* _ _ (@*Ole* _ _) $\Longrightarrow$*Ole*) as *mu_distr_le_morphism*.
Save.

Add *Parametric Morphism* (A B:Type) : (@*Mlet* A B)
with *signature* (*Ole* $\Longrightarrow$@*pointwise_relation* _ _ (@*Ole* _ _) $\Longrightarrow$*Ole*) as *mlet_distr_le_morphism*.
Save.

Add *Parametric Morphism* (A B:Type) : (@*Mlet* A B)
with *signature* (*Oeq* $\Longrightarrow$@*pointwise_relation* _ _ (@*Oeq* _ _) $\Longrightarrow$*Oeq*) as *mlet_distr_eq_morphism*.
Save.

6.5 Properties of monadic operators

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

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

Lemma *Mlet_assoc* : $\forall$ (A B C:Type) ($m1$: *distr* A) ($m2$:$A \to$ *distr* B) ($m3$:$B \to$ *distr* C),
Mlet (*Mlet* $m1$ $m2$) $m3 \equiv$ *Mlet* $m1$ (fun x:$A \Rightarrow$ *Mlet* ($m2$ x) $m3$).

Lemma *let_indep* : $\forall$ (A B:Type) ($m1$:*distr* A) ($m2$: *distr* B) (f:*MF* B),
$\mu\ m1$ (fun _ $\Rightarrow \mu\ m2\ f) \equiv$ *pone* $m1 \times (\mu\ m2\ f)$.

6.6 A specific distribution

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

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

6.7 Scaling a distribution

Definition *Mmult* A (k:*MF* A) (m:*M* A) : *M* A.
Defined.

Lemma *Mmult_simpl* : $\forall$ A (k:*MF* A) (m:*M* A) f, *Mmult* k m f = m (fun $x \Rightarrow k\ x \times f\ x$).

Lemma *Mmult_stable_inv* : $\forall$ A (k:*MF* A) (d:*distr* A), *stable_inv* (*Mmult* k (μ d)).

Lemma *Mmult_stable_plus* : $\forall$ A (k:*MF* A) (d:*distr* A), *stable_plus* (*Mmult* k (μ d)).

Lemma *Mmult_stable_mult* : $\forall$ A (k:*MF* A) (d:*distr* A), *stable_mult* (*Mmult* k (μ d)).

Lemma *Mmult_continuous* : $\forall$ *A* (*k*:*MF A*) (*d*:*distr A*), *continuous* (*Mmult k* (μ *d*)).

Definition *distr_mult A* (*k*:*MF A*) (*d*:*distr A*) : *distr A*.
Defined.

Lemma *distr_mult_assoc* : $\forall$ *A* (*k1 k2*:*MF A*) (*d*:*distr A*),
distr_mult k1 (*distr_mult k2 d*) $\equiv$ *distr_mult* (fun *x* $\Rightarrow$ *k1 x* $\times$ *k2 x*) *d*.

Add *Parametric Morphism* (*A B* : Type) : (*distr_mult* (*A*:=*A*))
with *signature Oeq* $\Longrightarrow$ *Oeq* $\Longrightarrow$ *Oeq*
as *distr_mult_eq_compat*.
Save.

Scaling with a constant functions

Definition *distr_scale A* (*k*:*U*) (*d*:*distr A*) : *distr A* := *distr_mult* (*fcte A k*) *d*.

Lemma *distr_scale_assoc* : $\forall$ *A* (*k1 k2*:*U*) (*d*:*distr A*),
distr_scale k1 (*distr_scale k2 d*) $\equiv$ *distr_scale* (*k1*$\times$*k2*) *d*.

Lemma *distr_scale_simpl* : $\forall$ *A* (*k*:*U*) (*d*:*distr A*)(*f*:*MF A*),
μ (*distr_scale k d*) *f* $\equiv$ *k* $\times$ μ *d f*.

Add *Parametric Morphism A* : (*distr_scale* (*A*:=*A*))
with *signature Oeq* $\Longrightarrow$ *Oeq* $\Longrightarrow$ *Oeq*
as *distr_scale_eq_compat*.
Save.
Hint Resolve *distr_scale_eq_compat*.

Lemma *distr_scale_one* : $\forall$ *A* (*d*:*distr A*), *distr_scale* 1 *d* $\equiv$ *d*.

Lemma *distr_scale_zero* : $\forall$ *A* (*d*:*distr A*), *distr_scale* 0 *d* $\equiv$ *distr_null A*.

Hint Resolve *distr_scale_simpl distr_scale_assoc distr_scale_one distr_scale_zero*.

Lemma *let_indep_distr* : $\forall$ (*A B*:Type) (*m1*:*distr A*) (*m2*: *distr B*),
Mlet m1 (fun _ $\Rightarrow$ *m2*) $\equiv$ *distr_scale* (*pone m1*) *m2*.

Definition *Mdiv A* (*k*:*U*) (*m*:*M A*) : *M A* := *UDiv k* @ *m*.

Lemma *Mdiv_simpl* : $\forall$ *A k* (*m*:*M A*) *f*, *Mdiv k m f* = *m f* / *k*.

Lemma *Mdiv_stable_inv* : $\forall$ *A* (*k*:*U*) (*d*:*distr A*)(*dk* : μ *d* (*fone A*) $\leq$ *k*),
stable_inv (*Mdiv k* (μ *d*)).

Lemma *Mdiv_stable_plus* : $\forall$ *A* (*k*:*U*)(*d*:*distr A*), *stable_plus* (*Mdiv k* (μ *d*)).

Lemma *Mdiv_stable_mult* : $\forall$ *A* (*k*:*U*)(*d*:*distr A*)(*dk* : μ *d* (*fone A*) $\leq$ *k*),
stable_mult (*Mdiv k* (μ *d*)).

Lemma *Mdiv_continuous* : $\forall$ *A* (*k*:*U*)(*d*:*distr A*), *continuous* (*Mdiv k* (μ *d*)).

Definition *distr_div A* (*k*:*U*) (*d*:*distr A*) (*dk* : μ *d* (*fone A*) $\leq$ *k*)
: *distr A*.
Defined.

Lemma *distr_div_simpl* : $\forall$ *A* (*k*:*U*) (*d*:*distr A*) (*dk* : μ *d* (*fone A*) $\leq$ *k*) *f*,
μ (*distr_div* _ _ *dk*) *f* = μ *d f* / *k*.

6.8 Conditional probabilities

Definition *mcond A* (*m*:*M A*) (*f*:*MF A*) : *M A*.
Defined.

Lemma *mcond_simpl* : $\forall$ *A* (*m*:*M A*) (*f g*: *MF A*),
mcond m f g = *m* (*fconj f g*) / *m f*.

Lemma *mcond_stable_plus* : $\forall$ *A* (*m*:*distr A*) (*f*: *MF A*), *stable_plus* (*mcond* (μ *m*) *f*).

Lemma *mcond_stable_inv* : $\forall$ *A* (*m*:*distr A*) (*f*: *MF A*), *stable_inv* (*mcond* (μ *m*) *f*).

Lemma $mcond\_stable\_mult$: $\forall$ A (m:$distr$ A) (f: MF A), $stable\_mult$ ($mcond$ (μ m) f).

Lemma $mcond\_continuous$: $\forall$ A (m:$distr$ A) (f: MF A), $continuous$ ($mcond$ (μ 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 $Mcond\_total$: $\forall$ A (m:$distr$ A) (f:MF A),
  $\neg$ $0 \equiv \mu$ m f $\to$ μ ($Mcond$ m f) ($fone$ A) $\equiv 1$.

Lemma $Mcond\_simpl$: $\forall$ A (m:$distr$ A) (f g:MF A),
  μ ($Mcond$ m f) g = μ m ($fconj$ f g) / μ m f.
Hint Resolve $Mcond\_simpl$.

Lemma $Mcond\_zero\_stable$: $\forall$ A (m:$distr$ A) (f g:MF A),
  μ m $g \equiv 0 \to \mu$ ($Mcond$ m f) $g \equiv 0$.

Lemma $Mcond\_null$: $\forall$ A (m:$distr$ A) (f g:MF A),
  μ m $f \equiv 0 \to \mu$ ($Mcond$ m f) $g \equiv 0$.

Lemma $Mcond\_conj$: $\forall$ A (m:$distr$ A) (f g:MF A),
  μ m ($fconj$ f g) $\equiv \mu$ ($Mcond$ m f) $g \times \mu$ m f.

Lemma $Mcond\_decomp$:
  $\forall$ A (m:$distr$ A) (f g:MF A),
    μ m $g \equiv \mu$ ($Mcond$ m f) $g \times \mu$ m f + μ ($Mcond$ m ($finv$ f)) $g \times \mu$ m ($finv$ f).

Lemma $Mcond\_bayes$: $\forall$ A (m:$distr$ A) (f g:MF A),
  μ ($Mcond$ m f) $g \equiv$ (μ ($Mcond$ m g) $f \times \mu$ m g) / (μ m f).

Lemma $Mcond\_mult$: $\forall$ A (m:$distr$ A) (f g h:MF A),
  μ ($Mcond$ m h) ($fconj$ f g) $\equiv \mu$ ($Mcond$ m ($fconj$ g h)) $f \times \mu$ ($Mcond$ m h) g.

Lemma $Mcond\_conj\_simpl$: $\forall$ A (m:$distr$ A) (f g h:MF A),
  ($fconj$ f $f \equiv f$) $\to \mu$ ($Mcond$ m f) ($fconj$ f g) $\equiv \mu$ ($Mcond$ m f) g.

Hint Resolve $Mcond\_mult$ $Mcond\_conj\_simpl$.

6.9 Least upper bound of increasing sequences of distributions

Lemma M_lub_simpl : $\forall$ 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.

Lemma Mu_simpl : $\forall$ d f, Mu d f = μ d f.

Variable muf : nat -m> $distr$ A.

Definition mu_lub: $distr$ A.

Defined.

Lemma mu_lub_le : $\forall$ n:nat, muf $n \leq mu\_lub$.

Lemma mu_lub_sup : $\forall$ m: $distr$ A, ($\forall$ n:nat, muf $n \leq m$) $\to mu\_lub \leq m$.

End $Lubs$.
Hint Resolve mu_lub_le mu_lub_sup.

6.9.1 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),
        μ (lub h) f = lub (mshift (Mu A @ h) f).
Hint Resolve distr_lub_simpl.
```

6.10 Fixpoints

```
Definition Mfix (A B:Type) (F: (A → distr B) -m> (A → distr B))
     : A → distr B := fixp F.
```

```
Definition MFix (A B:Type) : ((A → distr B) -m> (A → distr B)) -m> (A → distr B)
               := Fixp (A → distr B).
```

```
Lemma Mfix_le : ∀ (A B:Type) (F: (A → distr B) -m> (A → distr B)) (x:A),
                Mfix F x ≤ F (Mfix F) x.
```

```
Lemma Mfix_eq : ∀ (A B:Type) (F: (A → distr B) -m> (A → distr B)),
continuous F → ∀ (x:A), Mfix F x ≡ F (Mfix F) x.
```

```
Hint Resolve Mfix_le Mfix_eq.
```

```
Lemma Mfix_le_compat : ∀ (A B:Type) (F G : (A → distr B)-m> (A → distr B)),
          F ≤ G → Mfix F ≤ Mfix G.
```

```
Definition Miter (A B:Type) := Ccpo.iter (D:=A → distr B).
```

```
Lemma Mfix_le_iter : ∀ (A B:Type) (F:(A → distr B) -m> (A → distr B)) (n:nat),
        Miter F n ≤ Mfix F.
```

6.11 Continuity

```
Section Continuity.
```

```
Variables A B:Type.
```

```
Instance Mlet_continuous_right
     : ∀ a:distr A, continuous (D1:= A → distr B) (D2:=distr B) (MLet A B a).
Save.
```

```
Lemma Mlet_continuous_left
     : continuous (D1:=distr A) (D2:=(A → distr B) -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 : ∀ (mun:nat -m> distr A) (Mn : nat -m> (A → distr B)),
                Mlet (lub mun) (lub Mn) ≤ lub ((MLet A B @² mun) Mn).
```

```
Lemma Mlet_lub_le_left : ∀ (mun:nat -m> distr A)
               (M : A → distr B),
                Mlet (lub mun) M ≤ lub (mshift (MLet A B @ mun) M).
```

```
Lemma Mlet_lub_le_right : ∀ (m:distr A)
               (Mun : nat -m> (A → distr B)),
                Mlet m (lub Mun) ≤ lub ((MLet A B m)@Mun).
```

```
Lemma Mlet_lub_fun_le_right : ∀ (m:distr A)
               (Mun : A → nat -m> distr B),
                Mlet m (fun x ⇒ lub (Mun x)) ≤ lub ((MLet A B m)@(ishift Mun)).
```

Lemma *Mfix_continuous* :
$\forall$ (*Fn* : *nat* -*m*> (*A* $\rightarrow$ *distr B*) -*m*> (*A* $\rightarrow$ *distr B*)),
($\forall$ *n*, *continuous* (*Fn n*)) $\rightarrow$
Mfix (*lub Fn*) $\leq$ *lub* (*MFix A B* @ *Fn*).

End *Continuity*.

6.12 Exact probability : probability of full space is 1

Class *Term A* (*m*:*distr A*) := *term_def* : μ *m* (*fone A*) $\equiv$ 1.
Hint Resolve @*term_def*.

Lemma *Mlet_indep_term* : $\forall$ *A B* (*d1*:*distr A*) (*d2*:*distr B*) {*T*:*Term d1*},
Mlet d1 (fun _ $\Rightarrow$ *d2*) $\equiv$ *d2*.
Hint Resolve *Mlet_indep_term*.

Lemma *mu_stable_inv_term* : $\forall$ *A* (*d*:*distr A*) {*T*:*Term d*} *f*, μ *d* (*finv f*) $\equiv$ [1-](μ *d f*).

Instance *Munit_term* : $\forall$ *A* (*a*:*A*), *Term* (*Munit a*).
Save.
Hint Resolve *Munit_term*.

Instance *Mlet_term* : $\forall$ *A B* (*d1*:*distr A*) (*d2*: *A* $\rightarrow$ *distr B*)
{*T1*:*Term d1*} {*T2*:$\forall$ *x*, *Term* (*d2 x*)}, *Term* (*Mlet d1 d2*).
Save.
Hint Resolve *Mlet_term*.

Lemma *fplusok_mu_term* : $\forall$ (*A B*:Type) (*d*:*distr B*) (*f f'*:*A* $\rightarrow$ *MF B*) {*T*:*Term d*},
($\forall$ *x*:*A*, *fplusok* (*f x*) (*f' x*)) $\rightarrow$
fplusok (fun *x* : *A* $\Rightarrow$ μ *d* (*f x*)) (fun *x* : *A* $\Rightarrow$ μ *d* (*f' x*)).

6.13 distribution for *flip*

The distribution associated to *flip* () is *f* -> $\frac{1}{2}$ (*f true*) + $\frac{1}{2}$ (*f false*)

Definition *flip* : *M bool* := *mon* (fun (*f* : *bool* $\rightarrow$ *U*) $\Rightarrow$ $\frac{1}{2}$ $\times$ (*f true*) + $\frac{1}{2}$ $\times$ (*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*.

Lemma *flip_true* : *flip B2U* $\equiv$ $\frac{1}{2}$.

Lemma *flip_false* : *flip NB2U* $\equiv$ $\frac{1}{2}$.

Hint Resolve *flip_true flip_false*.

Definition *Flip* : *distr bool*.
Defined.

Lemma *Flip_simpl* : $\forall$ *f*, μ *Flip f* = $\frac{1}{2}$ $\times$ (*f true*) + $\frac{1}{2}$ $\times$ (*f false*).

Instance *flip_term* : *Term Flip*.
Save.
Hint Resolve *flip_term*.

6.14 Uniform distribution beween 0 and n

Require *Arith*.

6.14.1 Definition of *fnth*

fnth n k is defined as [1/]1+*n*

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

6.14.2 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* [].

6.15 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* -*m*> *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*),
μ *d* (*sigma_fun f n*) ≤ *sigma* (fun *k* ⇒ μ *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*) →
μ *d* (*sigma_fun f n*) ≡ *sigma* (fun *k* ⇒ μ *d* (*f k*)) *n*.

Lemma *mu_serie_le* : ∀ *A* (*d*:*distr A*) (*f*:*nat* → *MF A*),
μ *d* (*serie_fun f*) ≤ *serie* (fun *k* ⇒ μ *d* (*f k*)).

Lemma *mu_serie_eq* : ∀ *A* (*d*:*distr A*) (*f*:*nat* → *MF A*),
(∀ *x*, *wretract* (fun *k* ⇒ *f k x*)) →
μ *d* (*serie_fun f*) ≡ *serie* (fun *k* ⇒ μ *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*).

6.16 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*),
μ (*Discrete d*) = *discrete* (*coeff d*) (*points d*).

Definition *is_discrete* (*A*:Type) (*m*: *distr A*) :=
∃ *d* : *discr A*, *m* ≡ *Discrete d*.

6.16.1 Distribution for *random n*

The distribution associated to *random n* is *f* −> *sigma* (*i*=0..*n*) [1]1+*n* (*f i*) we cannot factorize [1/]1+*n* 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*).

6.16.2 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*), *μ* (*Random n*) = *random n*.

Instance *Random_total* : ∀ *n* : *nat*, *Term* (*Random n*).
Save.
Hint Resolve *Random_total*.

Lemma *Random_inv* : ∀ *f* *n*, μ (*Random* *n*) (*finv* *f*) ≡ [1-] (μ (*Random* *n*) *f*).
Hint Resolve *Random_inv*.

6.17 Tacticals

```
Ltac mu_plus d :=
  match goal with
 | ⊢ context [fmont (μ 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 (μ d) (fun x ⇒ (Umult ?k (@?f x)))] ⇒
      rewrite (mu_stable_mult d k f)
  end.
```

7 SProbas.v: Definition of the monad for sub-distributions

Require Export *Probas*.

7.1 Definition of (sub)distribution

Subdistributions are measure functions μ such that

- μ (1-*f*) ≤ 1 - μ *f*
- *f* ≤ 1-*g* → μ *f* + μ *g* ≤ μ (*f*+*g*)
- μ *f* & μ *g* ≤ μ (*f* & *g*) - [μ (*f*+*k*) ≤ μ *f* + *k*] - [μ (*k* × *f*) = *k* × μ (*f*)] - [μ (*lub* *f_n*) ≤ *lub* μ (*f_n*)]

```
Record sdistr (A:Type) : Type :=
  {smu : M A;
   smu_stable_inv : stable_inv smu;
   smu_le_plus : le_plus smu;
   smu_le_esp : le_esp smu;
   smu_le_plus_cte : le_plus_cte smu;
   smu_stable_mult : stable_mult smu;
   smu_continuous : continuous smu}.
Hint Resolve smu_le_plus smu_stable_inv smu_le_esp smu_stable_mult
smu_continuous.
```

7.2 Properties of sub-measures

Lemma *smu_monotonic* : ∀ (*A* : Type)(*m*: *sdistr* *A*), *monotonic* (*smu* *m*).
Hint Resolve *smu_monotonic*.
Implicit Arguments *smu_monotonic* [*A*].

Lemma *smu_stable* : ∀ (*A* : Type)(*m*: *sdistr* *A*), *stable* (*smu* *m*).
Hint Resolve *smu_stable*.
Implicit Arguments *smu_stable* [*A*].

Lemma *smu_zero* : ∀ (*A* : Type)(*m*: *sdistr* *A*), *smu* *m* (*fzero* *A*) ≡ 0.
Hint Resolve *smu_zero*.

Lemma *smu_stable_mult_right* : ∀ (*A* : Type)(*m*:(*sdistr* *A*)) (*c*:*U*) (*f* : *A* → *U*),

$smu\ m$ (fun $x \Rightarrow (f\ x) \times c) \equiv (smu\ m\ f) \times c$.

Lemma *smu_le_minus_left* : $\forall$ (A : Type)(m:*sdistr* A) (f g:$A \rightarrow U$),
$smu\ m\ (fminus\ f\ g) \leq smu\ m\ f$.
Hint Resolve *smu_le_minus_left*.

Lemma *smu_le_minus* : $\forall$ (A:Type) (m:*sdistr* A)(f g: $A \rightarrow U$),
$g \leq f \rightarrow smu\ m\ (fminus\ f\ g) \leq smu\ m\ f$ - $smu\ m\ g$.
Hint Resolve *smu_le_minus*.

Lemma *smu_cte* : $\forall$ (A : Type)(m:(*sdistr* A)) (c:U),
$smu\ m\ (fcte\ A\ c) \equiv c \times smu\ m\ (fone\ A)$.
Hint Resolve *smu_cte*.

Lemma *smu_cte_le* : $\forall$ (A : Type)(m:(*sdistr* A)) (c:U),
$smu\ m\ (fcte\ A\ c) \leq c$.

Lemma *smu_cte_eq* : $\forall$ (A : Type)(m:(*sdistr* A)) (c:U),
$smu\ m\ (fone\ A) \equiv 1 \rightarrow smu\ m\ (fcte\ A\ c) \equiv c$.

Hint Resolve *smu_cte_le* *smu_cte_eq*.

Lemma *smu_le_minus_cte* : $\forall$ (A:Type) (m:*sdistr* A)(f: $A \rightarrow U$) (k:U),
$smu\ m\ f$ - $k \leq smu\ m\ (fminus\ f\ (fcte\ A\ k))$.

Lemma *smu_inv_le_minus* :
$\forall$ (A:Type) (m:*sdistr* A)(f: $A \rightarrow U$), $smu\ m\ (finv\ f) \leq smu\ m\ (fone\ A)$ - $smu\ m\ f$.

Lemma *smu_inv_minus_inv* : $\forall$ (A:Type) (m:*sdistr* A)(f: $A \rightarrow U$),
$smu\ m\ (finv\ f)$ + [1-]($smu\ m\ (fone\ A)$) $\leq$ [1-]($smu\ m\ f$).

Definition *stable_plus_sdistr* : $\forall$ A (m:M A),
stable_plus $m \rightarrow$ *stable_inv* $m \rightarrow$ *stable_mult* $m \rightarrow$ *continuous* $m \rightarrow$ *sdistr* A.
Defined.

Definition *distr_sdistr* : $\forall$ A, *distr* $A \rightarrow$ *sdistr* A.
Defined.

Definition *Sunit* A (x:A) : *sdistr* A := *distr_sdistr* (*Munit* x).

Lemma *Sunit_unit* : $\forall$ A (x:A), *smu* (*Sunit* x) = *unit* x.

Lemma *Sunit_simpl* : $\forall$ A (x:A) (f : MF A), *smu* (*Sunit* x) f = f x.

Definition *Slet* : $\forall$ A B:Type, (*sdistr* A) $\rightarrow$ ($A \rightarrow$ *sdistr* B) $\rightarrow$ *sdistr* B.
Defined.

Lemma *Slet_star* : $\forall$ (A B:Type) (m:*sdistr* A) (M : $A \rightarrow$ *sdistr* B),
smu (*Slet* m M) = *star* (*smu* m) (fun $x \Rightarrow$ *smu* (M x)).

Lemma *Slet_simpl* : $\forall$ A B (m:*sdistr* A) (M : $A \rightarrow$ *sdistr* B) (f:MF B),
smu (*Slet* m M) f = *smu* m (fun $x \Rightarrow$ *smu* (M x) f).

Non deterministic choice

Definition *Smin* (A:Type)($m1$ $m2$: *sdistr* A) : *sdistr* A.
Save.

7.3 Operations on sub-distributions

Instance *Osdistr* (A : Type) : *ord* (*sdistr* A) :=
{ *Ole* := fun f $g \Rightarrow$ *smu* $f \leq$ *smu* g;
Oeq := fun f $g \Rightarrow$ *smu* $f \equiv$ *smu* g}.
Defined.

Lemma *Sunit_compat* : $\forall$ A (x y : A), $x = y \rightarrow$ *Sunit* $x \equiv$ *Sunit* y.

Lemma *Slet_compat* : $\forall$ (A B : Type) ($m1$ $m2$:*sdistr* A) ($M1$ $M2$: $A\rightarrow$ *sdistr* B),

$m1 \equiv m2 \rightarrow M1 \equiv M2 \rightarrow Slet\ m1\ M1 \equiv Slet\ m2\ M2.$

Lemma $le\_sdistr\_gen : \forall\ (A{:}\mathtt{Type})\ (m1\ m2{:}sdistr\ A),$
$m1 \leq m2 \rightarrow \forall\ f\ g, f \leq g \rightarrow smu\ m1\ f \leq smu\ m2\ g.$

7.4 Properties of monadic operators

Lemma $Slet\_unit : \forall\ (A\ B{:}\mathtt{Type})\ (x{:}A)\ (m{:}A \rightarrow sdistr\ B),\ Slet\ (Sunit\ x)\ m \equiv m\ x.$

Lemma $M\_ext : \forall\ (A{:}\mathtt{Type})\ (m{:}sdistr\ A),\ Slet\ m\ (\mathtt{fun}\ x \Rightarrow Sunit\ x) \equiv m.$

Lemma $Mcomp : \forall\ (A\ B\ C{:}\mathtt{Type})\ (m1{:}(sdistr\ A))\ (m2{:}A \rightarrow sdistr\ B)\ (m3{:}B \rightarrow sdistr\ C),$
$Slet\ (Slet\ m1\ m2)\ m3 \equiv Slet\ m1\ (\mathtt{fun}\ x{:}A \Rightarrow (Slet\ (m2\ x)\ m3)).$

Lemma $Slet\_le\_compat : \forall\ (A\ B{:}\mathtt{Type})\ (m1\ m2{:}\ sdistr\ A)\ (f1\ f2\ :\ A \rightarrow sdistr\ B),$
$m1 \leq m2 \rightarrow f1 \leq f2 \rightarrow Slet\ m1\ f1 \leq Slet\ m2\ f2.$

7.5 A specific subdistribution

Definition $sdistr\_null : \forall\ A : \mathtt{Type},\ sdistr\ A.$
Defined.

Lemma $le\_sdistr\_null : \forall\ (A{:}\mathtt{Type})\ (m\ :\ sdistr\ A),\ sdistr\_null\ A \leq m.$
Hint Resolve *le_sdistr_null*.

7.6 Least upper bound of increasing sequences of sdistributions

Section *Lubs*.
Variable A : Type.

Definition $Smu : sdistr\ A\ \text{-}m\!>\ M\ A.$
Defined.

Lemma $Smu\_simpl : \forall\ d\ f,\ Smu\ d\ f = smu\ d\ f.$

Variable $smuf\ :\ nat\ \text{-}m\!>\ sdistr\ A.$

Definition $smu\_lub{:}\ sdistr\ A.$

Defined.

Lemma $smu\_lub\_simpl : smu\ smu\_lub = lub\ (Smu\ @\ smuf).$

Lemma $smu\_lub\_le : \forall\ n{:}nat,\ smuf\ n \leq smu\_lub.$

Lemma $smu\_lub\_sup : \forall\ m{:}sdistr\ A,\ (\forall\ n{:}nat,\ smuf\ n \leq m) \rightarrow smu\_lub \leq m.$

End *Lubs*.

7.7 Sub-distribution for *flip*

The distribution associated to *flip* () is $f \mapsto \frac{1}{2}f(true)+\frac{1}{2}f(false)$ Definition $Sflip : sdistr\ bool := distr\_sdistr\ Flip.$

Lemma $Sflip\_simpl : smu\ Sflip = flip.$

7.8 Uniform sub-distribution beween 0 and n

Require *Arith*.

7.8.1 Distribution for *Srandom n*

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

Definition $Srandom\ (n{:}nat){:}\ sdistr\ nat{:}= distr\_sdistr\ (Random\ n).$

Lemma $Srandom\_simpl : \forall\ n,\ smu\ (Srandom\ n) = random\ n.$

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 -m> distr A -m> distr A -m> distr A.
Defined.

Lemma MIf_simpl : ∀ A b d1 d2, MIf A b d1 d2 = Mif b d1 d2.

Instance if_mon : ∀ `{o:ord A} (b:bool), monotonic2 (fun (x y:A) ⇒ if b then x else y).
Save.

Definition If `{o:ord A} (b:bool) : A -m> A -m> A := mon2 (fun (x y:A) ⇒ if b then x else y).

Instance Mif_continuous2 : ∀ (A:Type) b, continuous2 (MIf A b).
Save.

Hint Resolve Mif_continuous2.

Instance Mif_cond_continuous : ∀ (A:Type), continuous (MIf A).
Save.

Hint Resolve Mif_cond_continuous.

Add Parametric Morphism (A:Type) : (Mif (A:=A))
  with signature Oeq ⟹ Oeq ⟹ Oeq ⟹ Oeq
as Mif_eq_compat.
Save.
Hint Immediate Mif_eq_compat.

Add Parametric Morphism (A:Type) : (Mif (A:=A))
  with signature Ole ⟹ Ole ⟹ Ole ⟹ Ole
as Mif_le_compat_morph.
Save.

Lemma Mif_lub_eq_left : ∀ (A:Type) b h (d: distr A),
    Mif b (lub h) d ≡ lub (MIf _ b @ h) d.

Lemma Mif_lub_eq_right : ∀ (A:Type) b h (d: distr A),
    Mif b d (lub h) ≡ lub (MIf _ b d @ h).

Lemma Mif_lub_eq2 : ∀ (A:Type) b (h1 h2 : nat -m> distr A),
    Mif b (lub h1) (lub h2) ≡ lub ((MIf _ b @² h1) h2).

Instance Mif_term : ∀ (A:Type) b (d1 d2:distr A)
     {Tb : Term b} {T1:Term d1} {T2:Term d2}, Term (Mif b d1 d2).
Save.
Hint Resolve Mif_term.
```

8.2 Probabilistic choice

The distribution associated to *pchoice p m1 m2* is *f* –> *p* (*m1 f*) + (1-*p*) (*m2 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,
        μ (Mchoice p m1 m2) f = p × μ m1 f + [1-]p × μ 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).

Instance Mchoice_term : ∀ A p (d1 d2:distr A) {T1:Term d1} {T2:Term d2},
       Term (Mchoice p d1 d2).
Save.
Hint Resolve Mchoice_term.
```

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),
       μ (im_distr f m) h = μ 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.
Instance im_distr_term : ∀ A B (f:A→B)(d:distr A){T:Term d},
                   Term (im_distr f d).
Save.
Hint Resolve im_distr_term.
```

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),
            μ (prod_distr d1 d2) (fun xy ⇒ f (fst xy) × g (snd xy)) ≡ μ d1 f × μ d2 g.
Lemma prod_distr_fst : ∀ (A B : Type) (d1: distr A) (d2:distr B) (f:A → U),
               μ (prod_distr d1 d2) (fun xy ⇒ f (fst xy)) ≡ pone d2 × μ d1 f.
Lemma prod_distr_snd : ∀ (A B : Type) (d1: distr A) (d2:distr B) (g:B → U),
               μ (prod_distr d1 d2) (fun xy ⇒ g (snd xy)) ≡ pone d1 × μ d2 g.
Lemma prod_distr_fst_eq : ∀ (A B : Type) (d1: distr A) (d2:distr B),
       pone d2 ≡ 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),
         pone d1 ≡ 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)) :=
        μ (prod_distr d1 d2) f ≡ μ (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 -m> MF (A×B)),
        (∀ n, prod_distr_com d1 d2 (f n)) → prod_distr_com d1 d2 (lub f).
Lemma prod_distr_com_sym : ∀ 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.
Lemma is_discrete_swap: ∀ A B C (d1:distr A) (d2:distr B) (f:A → B → distr C),
   is_discrete d1 →
   Mlet d1 (fun x ⇒ Mlet d2 (fun y ⇒ f x y)) ≡ Mlet d2 (fun y ⇒ Mlet d1 (fun x ⇒ f x y)).
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.
Add Parametric Morphism (A B : Type) : (fst_distr (A:=A) (B:=B))
   with signature Oeq ⟹Oeq as fst_distr_eq_compat.
Save.
Add Parametric Morphism (A B : Type) : (snd_distr (A:=A) (B:=B))
   with signature Oeq ⟹Oeq as snd_distr_eq_compat.
Save.
Lemma fst_prod_distr : ∀ A B (m1:distr A) (m2:distr B),
          fst_distr (prod_distr m1 m2) ≡ distr_scale (pone m2) m1.
Lemma snd_prod_distr : ∀ A B (m1:distr A) (m2:distr B),
          snd_distr (prod_distr m1 m2) ≡ distr_scale (pone m1) m2.
Lemma pone_prod : ∀ A B (m1:distr A) (m2:distr B),
          pone (prod_distr m1 m2) ≡ pone m1 × pone m2.
Instance prod_distr_term : ∀ A B (d1:distr A) (d2:distr B)
      {T1:Term d1} {T2:Term d2}, Term (prod_distr d1 d2).
Save.
Hint Resolve prod_distr_term.
Lemma fst_prod_distr_term : ∀ A B (d1:distr A) (d2:distr B) {T2:Term d2},
          fst_distr (prod_distr d1 d2) ≡ d1.
Lemma snd_prod_distr_term : ∀ A B (d1:distr A) (d2:distr B) {T1:Term d1},
          snd_distr (prod_distr d1 d2) ≡ d2.
Hint Resolve fst_prod_distr_term snd_prod_distr_term.
```

8.5 Independance of distribution

```
Definition prod_indep A B (m:distr (A×B)) :=
```

```
        distr_scale (pone m) m ≡ prod_distr (fst_distr m) (snd_distr m).
Lemma prod_distr_indep : ∀ A B (m1:distr A) (m2:distr B), prod_indep (prod_distr m1 m2).
Add Parametric Morphism A B : (prod_indep (A:=A) (B:=B))
  with signature Oeq ==>Basics.impl
  as prod_indep_eq_compat.
Save.
Hint Resolve prod_indep_eq_compat.
Lemma distr_indep_mult
  : ∀ A B (m:distr (A×B)), prod_indep m →
                      ∀ (f1 : MF A) (f2:MF B),
                      pone m × μ m (fun p ⇒ f1 (fst p) × f2 (snd p)) ≡
                      μ (fst_distr m) f1 × μ (snd_distr m) f2.
```

8.6 Range of a distribution

```
Definition range A (P:A → Prop) (d: distr A) :=
   ∀ f, (∀ x, P x → 0 ≡ f x) → 0 ≡ μ d f.
Lemma range_le : ∀ A (P: A → Prop) (d:distr A), range P d →
       ∀ f g, (∀ a, P a → f a ≤ g a) → μ d f ≤ μ d g.
Lemma range_eq : ∀ A (P: A → Prop) (d:distr A), range P d →
       ∀ f g, (∀ a, P a → f a ≡ g a) → μ d f ≡ μ d g.
Lemma im_range A B (f : A → B) :
    ∀ (d : distr A) (P : B → Prop),
    range (fun x ⇒ P (f x)) d → range P (im_distr f d).
Hint Resolve im_range.
Lemma range_impl A (P Q : A → Prop) :
  ∀ (d:distr A), (∀ x, P x → Q x)
    → range P d → range Q d.
Lemma im_range_map A B (f : A → B) :
    ∀ (d : distr A) (P : B → Prop) (Q:A → Prop),
    (∀ x, Q x → P (f x)) →
    range Q d → range P (im_distr f d).
Lemma im_range_prop A B (f : A → B) :
    ∀ (d : distr A) (P : B → Prop),
    (∀ x, P (f x)) → range P (im_distr f d).
Lemma range_le_compat : ∀ A (P : A → Prop) (d1 d2 : distr A),
    d1 ≤ d2 → range P d2 → range P d1.
Add Parametric Morphism A (P : A → Prop) : (range P)
    with signature Oeq ==>iff as range_distr_morph.
Save.
```

9 Prog.v: Axiomatic semantics

9.1 Definition of correctness judgements

- *ok p e q* is defined as $p \leq \mu\ e\ q$
- *up p e q* is defined as $\mu\ e\ q \leq p$

```
Definition ok (A:Type) (p:U) (e:distr A) (q:A → U) := p ≤ μ 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) := μ 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'.
```

```
Add Parametric Morphism (A:Type) : (@ok A)
  with signature Ole -> Oeq ==>Ole ==>Basics.impl
  as ok_le_morphism.
Save.
```

```
Add Parametric Morphism (A:Type) : (@ok A)
  with signature Oeq -> Oeq ==>Oeq ==>iff
  as ok_eq_morphism.
Save.
```

```
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 okfun_eq_compat :
∀ (A B:Type) (p p':A → U) (e e':A → distr B) (q q':A → B → U),
    p' ≡ p → q ≡ q' → e ≡ e' → okfun p e q → okfun p' e' q'.
```

```
Add Parametric Morphism (A B:Type) : (@okfun A B)
  with signature Ole -> Oeq ==>Ole ==>Basics.impl
  as okfun_le_morphism.
Save.
```

```
Add Parametric Morphism (A B:Type) : (@okfun A B)
  with signature Oeq -> Oeq ==>Oeq ==>iff
  as okfun_eq_morphism.
Save.
```

```
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 → μ e (finv f) ≤ [1-]p.
```

```
Lemma okup_le_compat : ∀ (A:Type) (p p':U) (e:distr A) (q q':A → U),
    p ≤ p' → q' ≤ q → okup p e q → okup p' e q'.
```

```
Lemma okup_eq_compat : ∀ (A:Type) (p p':U) (e e':distr A) (q q':A → U),
    p ≡ p' → q ≡ q' → e ≡ e' → okup p e q → okup p' e' q'.
```

```
Lemma upfun_le_compat : ∀ (A B:Type) (p p':A → U) (e:A → distr B)
    (q q':A → B → U),
    p ≤ p' → q'≤q → upfun p e q → upfun p' e q'.
```

```
Lemma okup_mult : ∀ (A:Type)(k p:U)(e:distr A)(f : A → U), okup p e f → okup (k×p) e (fmult k f).
```

9.3 Basic rules

9.3.1 Rules for application:

- $ok\ r\ a\ p$ and $\forall\ x$, $ok\ (p\ x)\ (f\ x)\ q$ implies $ok\ r\ (f\ a)\ q$
- $up\ r\ a\ p$ and $\forall\ x$, $up\ (p\ x)\ (f\ x)\ q$ implies $up\ r\ (f\ a)\ q$

Lemma $apply\_rule : \forall\ (A\ B{:}\texttt{Type})(a{:}(distr\ A))(f{:}A \to distr\ B)(r{:}U)(p{:}A \to U)(q{:}B \to U)$,
$ok\ r\ a\ p \to okfun\ p\ f\ (\texttt{fun}\ x \Rightarrow q) \to ok\ r\ (Mlet\ a\ f)\ q$.

Lemma $okup\_apply\_rule : \forall\ (A\ B{:}\texttt{Type})(a{:}distr\ A)(f{:}A \to distr\ B)(r{:}U)(p{:}A \to U)(q{:}B \to U)$,
$okup\ r\ a\ p \to upfun\ p\ f\ (\texttt{fun}\ x \Rightarrow q) \to okup\ r\ (Mlet\ a\ f)\ q$.

9.3.2 Rules for abstraction

Lemma $lambda\_rule : \forall\ (A\ B{:}\texttt{Type})(f{:}A \to distr\ B)(p{:}A \to U)(q{:}A \to B \to U)$,
$(\forall\ x{:}A,\ ok\ (p\ x)\ (f\ x)\ (q\ x)) \to okfun\ p\ f\ q$.

Lemma $okup\_lambda\_rule : \forall\ (A\ B{:}\texttt{Type})(f{:}A \to distr\ B)(p{:}A \to U)(q{:}A \to B \to U)$,
$(\forall\ x{:}A,\ okup\ (p\ x)\ (f\ x)\ (q\ x)) \to upfun\ p\ f\ q$.

9.3.3 Rules for conditional

- $ok\ p1\ e1\ q$ and $ok\ p2\ e2\ q$ implies $ok\ (p1 \times \mu\ b\ (\chi\ true) + p2 \times \mu\ b\ (\chi\ false)\ (\texttt{if}\ b\ \texttt{then}\ e1\ \texttt{else}\ e2)\ q$
- $up\ p1\ e1\ q$ and $up\ p2\ e2\ q$ implies $up\ (p1 \times \mu\ b\ (\chi\ true) + p2 \times \mu\ b\ (\chi\ false)\ (\texttt{if}\ b\ \texttt{then}\ e1\ \texttt{else}\ e2)\ q$

Lemma $combiok : \forall\ (A{:}\texttt{Type})\ p\ q\ (f1\ f2 : A \to U),\ p \le [1\text{-}]q \to fplusok\ (fmult\ p\ f1)\ (fmult\ q\ f2)$.
Hint Extern 1 $\Rightarrow$ apply *combiok*.

Lemma $fmult\_fplusok : \forall\ (A{:}\texttt{Type})\ p\ q\ (f1\ f2 : A \to U),\ fplusok\ f1\ f2 \to fplusok\ (fmult\ p\ f1)\ (fmult\ q\ f2)$.
Hint Resolve *fmult_fplusok*.

Lemma $ifok : \forall\ f1\ f2,\ fplusok\ (fmult\ f1\ B2U)\ (fmult\ f2\ NB2U)$.
Hint Resolve *ifok*.

Lemma $Mif\_eq : \forall\ (A{:}\texttt{Type})(b{:}(distr\ bool))(f1\ f2{:}distr\ A)(q{:}MF\ A)$,
$\mu\ (Mif\ b\ f1\ f2)\ q \equiv (\mu\ f1\ q) \times (\mu\ b\ B2U) + (\mu\ f2\ q) \times (\mu\ b\ NB2U)$.

Lemma $Mif\_eq2 : \forall\ (A : \texttt{Type})\ (b : distr\ bool)\ (f1\ f2 : distr\ A)\ (q : MF\ A)$,
$\mu\ (Mif\ b\ f1\ f2)\ q \equiv \mu\ b\ B2U \times \mu\ f1\ q + \mu\ b\ NB2U \times \mu\ f2\ q$.

Lemma *ifrule* :
$\forall\ (A{:}\texttt{Type})(b{:}(distr\ bool))(f1\ f2{:}distr\ A)(p1\ p2{:}U)(q{:}A \to U)$,
$ok\ p1\ f1\ q \to ok\ p2\ f2\ q$
$\to ok\ (p1 \times (\mu\ b\ B2U) + p2 \times (\mu\ b\ NB2U))\ (Mif\ b\ f1\ f2)\ q$.

Lemma *okup_ifrule* :
$\forall\ (A{:}\texttt{Type})(b{:}(distr\ bool))(f1\ f2{:}distr\ A)(p1\ p2{:}U)(q{:}A \to U)$,
$okup\ p1\ f1\ q \to okup\ p2\ f2\ q$
$\to okup\ (p1 \times (\mu\ b\ B2U) + p2 \times (\mu\ b\ NB2U))\ (Mif\ b\ f1\ f2)\ q$.

9.3.4 Rule for fixpoints

with $\phi\ x = F\ \phi\ x$, p an increasing sequence of functions starting from 0
$\forall\ f\ i,\ (\forall\ x,\ ok\ (p\ i\ x)\ f\ q \Rightarrow \forall\ x,\ ok\ p\ (i{+}1)\ x\ (F\ f\ x)\ q)$ implies $\forall\ x,\ ok\ (lub\ p\ x)\ (\phi\ x)\ q$ Section *Fixrule*.
Variables $A\ B$: Type.

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

Section *Ruleseq*.
Variable $q : A \to B \to U$.

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).
BUG: rewrite fails

Lemma *muF_stable* : stable muF.

Definition *mu_muF_commute_le* :=
∀ f x, f ≤ Mfix F → μ (F f x) (q x) ≤ muF (fun y ⇒ μ (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, μ (Mfix F x) (q x) ≤ mufix muF x.
Hint Resolve *mu_mufix_le*.

Lemma *muF_le* : ∀ f, muF f ≤ f
→ ∀ x, μ (Mfix F x) (q x) ≤ f x.

```
Hypothesis muF_F_le :
    ∀ f x, f ≤ Mfix F → muF (fun y ⇒ μ (f y) (q y)) x ≤ μ (F f x) (q x).

Lemma mufix_mu_le : ∀ x, mufix muF x ≤ μ (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 → μ (F f x) (q x) ≡ muF (fun y ⇒ μ (f y) (q y)) x)
   → ∀ x, mufix muF x ≡ μ (Mfix F x) (q x).
Hint Resolve mufix_mu.
End Fix_muF.

Section Fix_Term.

Definition pterm : MF A := fun (x:A) ⇒ μ (Mfix F x) (fone B).
Variable muFone : MF A -m> MF A.

Hypothesis F_muF_eq_one :
  ∀ f x, f ≤ Mfix F → μ (F f x) (fone B) ≡ muFone (fun y ⇒ μ (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 -m> 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 ≤ μ (Mfix F x) (q x).

Lemma muF_le_term_minus :
∀ f, f ≤ pterm → muFqinv (fminus pterm f) ≤ fminus pterm f →
    ∀ x, f x ≤ μ (Mfix F x) (q x).

Variable muFq : MF A -m> 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 → μ (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 -m> MF A.
Defined.

Lemma Loop_eq :
    ∀ f x, Loop f x = μ (stop x) (fun b ⇒ if b then a else μ (step x) f).

Definition loop := mufix Loop.

Lemma Mfixvar :
  (∀ (f:A → distr B),
```

okfun (fun $x \Rightarrow$ *Loop* (fun $y \Rightarrow \mu$ (f y) (q y)) x) (fun $x \Rightarrow F$ f x) q)
$\rightarrow$ *okfun loop* (*Mfix* F) q.

Definition *up_loop* : *MF* A := *nufix Loop*.

Lemma *Mfixvar_up* :
($\forall$ (f:$A \rightarrow$ *distr* B),
upfun (fun $x \Rightarrow$ *Loop* (fun $y \Rightarrow \mu$ (f y) (q y)) x) (fun $x \Rightarrow F$ f x) q)
$\rightarrow$ *upfun up_loop* (*Mfix* F) q.

End *LoopRule*.

End *Fixrule*.

9.4 Rules for intervals

Distributions operates on intervals

Definition *Imu* : $\forall$ A:Type, *distr* $A \rightarrow (A \rightarrow IU) \rightarrow IU$.
Defined.

Lemma *low_Imu* : $\forall$ (A:Type) (e:*distr* A) (F: $A \rightarrow IU$),
low (*Imu* e F) = μ e (fun $x \Rightarrow$ *low* (F x)).

Lemma *up_Imu* : $\forall$ (A:Type) (e:*distr* A) (F: $A \rightarrow IU$),
up (*Imu* e F) = μ e (fun $x \Rightarrow$ *up* (F x)).

Lemma *Imu_monotonic* : $\forall$ (A:Type) (e:*distr* A) (F G : $A \rightarrow IU$),
($\forall$ x, *Iincl* (F x) (G x)) $\rightarrow$ *Iincl* (*Imu* e F) (*Imu* e G).

Lemma *Imu_stable_eq* : $\forall$ (A:Type) (e:*distr* A) (F G : $A \rightarrow IU$),
($\forall$ x, *Ieq* (F x) (G x)) $\rightarrow$ *Ieq* (*Imu* e F) (*Imu* e G).
Hint Resolve *Imu_monotonic Imu_stable_eq*.

Lemma *Imu_singl* : $\forall$ (A:Type) (e:*distr* A) (f:$A \rightarrow U$),
Ieq (*Imu* e (fun $x \Rightarrow$ *singl* (f x))) (*singl* (μ e f)).

Lemma *Imu_inf* : $\forall$ (A:Type) (e:*distr* A) (f:$A \rightarrow U$),
Ieq (*Imu* e (fun $x \Rightarrow$ *inf* (f x))) (*inf* (μ e f)).

Lemma *Imu_sup* : $\forall$ (A:Type) (e:*distr* A) (f:$A \rightarrow U$),
Iincl (*Imu* e (fun $x \Rightarrow$ *sup* (f x))) (*sup* (μ e f)).

Lemma *Iin_mu_Imu* :
$\forall$ (A:Type) (e:*distr* A) (F:$A \rightarrow IU$) (f:$A \rightarrow U$),
($\forall$ x, *Iin* (f x) (F x)) $\rightarrow$ *Iin* (μ e f) (*Imu* e F).
Hint Resolve *Iin_mu_Imu*.

Definition *Iok* (A:Type) (I:IU) (e:*distr* A) (F:$A \rightarrow IU$) := *Iincl* (*Imu* e F) I.
Definition *Iokfun* (A B:Type)(I:$A \rightarrow IU$) (e:$A \rightarrow$ *distr* B) (F:$A \rightarrow B \rightarrow IU$)
:= $\forall$ x, *Iok* (I x) (e x) (F x).

Lemma *Iin_mu_Iok* :
$\forall$ (A:Type) (I:IU) (e:*distr* A) (F:$A \rightarrow IU$) (f:$A \rightarrow U$),
($\forall$ x, *Iin* (f x) (F x)) $\rightarrow$ *Iok* I e F $\rightarrow$ *Iin* (μ e f) I.

9.4.1 Stability

Lemma *Iok_le_compat* : $\forall$ (A:Type) (I J:IU) (e:*distr* A) (F G:$A \rightarrow IU$),
Iincl I J $\rightarrow$ ($\forall$ x, *Iincl* (G x) (F x)) $\rightarrow$ *Iok* I e F $\rightarrow$ *Iok* J e G.

Lemma *Iokfun_le_compat* : $\forall$ (A B:Type) (I J:$A \rightarrow IU$) (e:$A \rightarrow$ *distr* B) (F G:$A \rightarrow B \rightarrow IU$),
($\forall$ x, *Iincl* (I x) (J x)) $\rightarrow$ ($\forall$ x y, *Iincl* (G x y) (F x y)) $\rightarrow$ *Iokfun* I e F $\rightarrow$ *Iokfun* J e G.

9.4.2 Rule for values

Lemma *Iunit_eq* : $\forall$ (A: Type) (a:A) (F:$A \rightarrow IU$), *Ieq* (*Imu* (*Munit* a) F) (F a).

9.4.3 Rule for application

Lemma *Ilet_eq* : $\forall$ (A B : Type) (a:*distr* A) (f:$A \to$ *distr* B)(I:*IU*)(G:$B \to$ *IU*),
Ieq (*Imu* (*Mlet* a f) G) (*Imu* a (fun $x \Rightarrow$ *Imu* (f x) G)).
Hint Resolve *Ilet_eq*.

Lemma *Iapply_rule* : $\forall$ (A B : Type) (a:*distr* A) (f:$A \to$ *distr* B)(I:*IU*)(F:$A \to$ *IU*)(G:$B \to$ *IU*),
Iok I a $F \to$ *Iokfun* F f (fun $x \Rightarrow G$) $\to$ *Iok* I (*Mlet* a f) G.

9.4.4 Rule for abstraction

Lemma *Ilambda_rule* : $\forall$ (A B:Type)(f:$A \to$ *distr* B)(F:$A \to$ *IU*)(G:$A \to B \to$ *IU*),
($\forall$ x:A, *Iok* (F x) (f x) (G x)) $\to$ *Iokfun* F f G.

9.4.5 Rule for conditional

Lemma *Imu_Mif_eq* : $\forall$ (A:Type)(b:*distr bool*)(*f1 f2*:*distr* A)(F:$A \to$ *IU*),
Ieq (*Imu* (*Mif* b *f1 f2*) F) (*Iplus* (*Imultk* (μ b *B2U*) (*Imu f1* F)) (*Imultk* (μ b *NB2U*) (*Imu f2* F))).

Lemma *Iifrule* :
$\forall$ (A:Type)(b:(*distr bool*))(*f1 f2*:*distr* A)(*I1 I2*:*IU*)(F:$A \to$ *IU*),
Iok I1 f1 $F \to$ *Iok I2 f2* F
$\to$ *Iok* (*Iplus* (*Imultk* (μ b *B2U*) *I1*) (*Imultk* (μ b *NB2U*) *I2*)) (*Mif* b *f1 f2*) F.

9.4.6 Rule for fixpoints

with $\phi\ x = F\ \phi\ x$, p a decreasing sequence of intervals functions ($p\ (i{+}1)\ x$ is a bubset of ($p\ i\ x$) such that ($p\ 0\ x$) contains 0 for all x.
$\forall f\ i$, ($\forall x$, *iok* ($p\ i\ x$) f ($q\ x$)) $\Rightarrow \forall x$, *iok* ($p\ (i{+}1)\ x$) ($F\ f\ x$) ($q\ x$) implies $\forall x$, *iok* (*lub* $p\ x$) ($\phi\ x$) ($q\ x$)
Section *IFixrule*.
Variables A B : Type.

Variable F : ($A \to$ *distr* B) -*m*> ($A \to$ *distr* B).

Section *IRuleseq*.
Variable Q : $A \to B \to$ *IU*.

Variable I : $A \to$ *nat* -*m*> *IU*.

Lemma *Ifixrule* :
($\forall$ x:A, *Iin* 0 (I x O)) $\to$
($\forall$ (i:*nat*) (f:$A \to$ *distr* B),
(*Iokfun* (fun $x \Rightarrow I$ x i) f Q) $\to$ *Iokfun* (fun $x \Rightarrow I$ x (S i)) (fun $x \Rightarrow F$ f x) Q)
$\to$ *Iokfun* (fun $x \Rightarrow$ *Ilim* (I x)) (*Mfix* F) Q.
End *IRuleseq*.

Section *ITransformFix*.

Section *IFix_muF*.
Variable Q : $A \to B \to$ *IU*.
Variable *ImuF* : ($A \to$ *IU*) -*m*> ($A \to$ *IU*).

Lemma *ImuF_stable* : $\forall$ I J, $I \equiv J \to$ *ImuF* $I \equiv$ *ImuF* J.

Section *IF_muF_results*.
Hypothesis *Iincl_F_ImuF* :
$\forall$ f x, $f \leq$ *Mfix* $F \to$
Iincl (*Imu* (F f x) (Q x)) (*ImuF* (fun $y \Rightarrow$ *Imu* (f y) (Q y)) x).

Lemma *Iincl_fix_ifix* : $\forall$ x, *Iincl* (*Imu* (*Mfix* F x) (Q x)) (*fixp* (D:=$A \to$ *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_true* : μ *Flip B2U* ≡ $\frac{1}{2}$.

Lemma *Flip_false* : μ *Flip NB2U* ≡ $\frac{1}{2}$.

Lemma *ok_Flip* : ∀ *q* : *bool* → *U*, *ok* ([1/2] × *q true* + $\frac{1}{2}$ × *q false*) *Flip q*.

Lemma *okup_Flip* : ∀ *q* : *bool* → *U*, *okup* ([1/2] × *q true* + $\frac{1}{2}$ × *q false*) *Flip q*.

Hint Resolve *ok_Flip okup_Flip Flip_true Flip_false*.

Lemma *Flip_eq* : ∀ *q* : *bool* → *U*, μ *Flip q* ≡ $\frac{1}{2}$ × *q true* + $\frac{1}{2}$ × *q false*.
Hint Resolve *Flip_eq*.

Lemma *IFlip_eq* : ∀ *Q* : *bool* → *IU*, *Ieq* (*Imu Flip Q*) (*Iplus* (*Imultk* $\frac{1}{2}$ (*Q true*)) (*Imultk* $\frac{1}{2}$ (*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* (μ ?*d1*) ?*f*) (*fmont* (μ ?*d2*) ?*g*)) ⇒ apply (*mu_le_compat* (*m1*:=*d1*) (*m2*:=*d2*) (*Ole_refl d1*) (*f*:=*f*) (*g*:=*g*)); intro
 | ⊢ (*Oeq* (*fmont* (μ ?*d1*) ?*f*) (*fmont* (μ ?*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* (μ (*Mlet* ?*m* ?*M*)) ?*f*)] ⇒ rewrite (*Mlet_simpl m M f*)
 | ⊢ *context* [(*fmont* (μ (*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* (μ *Flip*) ?*f*)] ⇒ rewrite (*Flip_simpl f*)

```
 | ⊢ context [(fmont (μ (Discrete ?d)) ?f)] ⇒ rewrite (Discrete_simpl d);
                                               rewrite (discrete_simpl (coeff d) (points
d) f)
 | ⊢ context [(fmont (μ (Random ?n)) ?f)] ⇒ rewrite (Random_simpl n);
                                               rewrite (random_simpl n f)
 | ⊢ context [(fmont (μ (Mif ?b ?f ?g)) ?h)] ⇒ unfold Mif
 | ⊢ context [(fmont (μ (Mchoice ?p ?m1 ?m2)) ?f)] ⇒ rewrite (Mchoice_simpl p m1 m2 f)
 | ⊢ context [(fmont (μ (im_distr ?f ?m)) ?h)] ⇒ rewrite (im_distr_simpl f m h)
 | ⊢ context [(fmont (μ (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),
```

$incl\ P\ Q \rightarrow incl\ Q\ R \rightarrow incl\ P\ R.$

`Lemma` $eqset\_incl : \forall\ (P\ Q : \texttt{set}),\ eqset\ P\ Q \rightarrow incl\ P\ Q.$

`Lemma` $eqset\_incl\_sym : \forall\ (P\ Q : \texttt{set}),\ eqset\ P\ Q \rightarrow incl\ Q\ P.$

`Lemma` $eqset\_incl\_intro$:
$\forall\ (P\ Q : \texttt{set}),\ incl\ P\ Q \rightarrow incl\ Q\ P \rightarrow eqset\ P\ Q.$

`Hint Resolve` $incl\_refl\ incl\_trans\ eqset\_incl\_intro.$
`Hint Immediate` $eqset\_incl\ eqset\_incl\_sym.$

10.5 Properties of operations on sets

`Lemma` $incl\_empty : \forall\ P,\ incl\ empty\ P.$

`Lemma` $incl\_empty\_false : \forall\ P\ a,\ incl\ P\ empty \rightarrow \neg\ P\ a.$

`Lemma` $incl\_add\_empty : \forall\ (a{:}A)\ (P{:}\texttt{set}),\ \neg\ incl\ (add\ a\ P)\ empty.$

`Lemma` $eqset\_empty\_false : \forall\ P\ a,\ eqset\ P\ empty \rightarrow P\ a \rightarrow False.$

`Hint Immediate` $incl\_empty\_false\ eqset\_empty\_false\ incl\_add\_empty.$

`Lemma` $incl\_rem\_stable : \forall\ a\ P\ Q,\ incl\ P\ Q \rightarrow incl\ (rem\ a\ P)\ (rem\ a\ Q).$

`Lemma` $incl\_add\_stable : \forall\ a\ P\ Q,\ incl\ P\ Q \rightarrow incl\ (add\ a\ P)\ (add\ a\ Q).$

`Lemma` $incl\_rem\_add\_iff$:
$\forall\ a\ P\ Q,\ incl\ (rem\ a\ P)\ Q \leftrightarrow incl\ P\ (add\ a\ Q).$

`Lemma` $incl\_rem\_add$:
$\forall\ (a{:}A)\ (P\ Q{:}\texttt{set}),$
$(P\ a) \rightarrow incl\ Q\ (rem\ a\ P) \rightarrow incl\ (add\ a\ Q)\ P.$

`Lemma` $incl\_add\_rem$:
$\forall\ (a{:}A)\ (P\ Q{:}\texttt{set}),$
$\neg\ Q\ a \rightarrow incl\ (add\ a\ Q)\ P \rightarrow incl\ Q\ (rem\ a\ P)\ .$

`Hint Immediate` $incl\_rem\_add\ incl\_add\_rem.$

`Lemma` $eqset\_rem\_add$:
$\forall\ (a{:}A)\ (P\ Q{:}\texttt{set}),$
$(P\ a) \rightarrow eqset\ Q\ (rem\ a\ P) \rightarrow eqset\ (add\ a\ Q)\ P.$

`Lemma` $eqset\_add\_rem$:
$\forall\ (a{:}A)\ (P\ Q{:}\texttt{set}),$
$\neg\ Q\ a \rightarrow eqset\ (add\ a\ Q)\ P \rightarrow eqset\ Q\ (rem\ a\ P).$

`Hint Immediate` $eqset\_rem\_add\ eqset\_add\_rem.$

`Lemma` $add\_rem\_eq\_eqset$:
$\forall\ x\ (P{:}\texttt{set}),\ eqset\ (add\ x\ (rem\ x\ P))\ (add\ x\ P).$

`Lemma` $add\_rem\_diff\_eqset$:
$\forall\ x\ y\ (P{:}\texttt{set}),$
$x{\neq}y \rightarrow eqset\ (add\ x\ (rem\ y\ P))\ (rem\ y\ (add\ x\ P)).$

`Lemma` add_eqset_in :
$\forall\ x\ (P{:}\texttt{set}),\ P\ x \rightarrow eqset\ (add\ x\ P)\ P.$

`Hint Resolve` $add\_rem\_eq\_eqset\ add\_rem\_diff\_eqset\ add\_eqset\_in.$

`Lemma` $add\_rem\_eqset\_in$:
$\forall\ x\ (P{:}\texttt{set}),\ P\ x \rightarrow eqset\ (add\ x\ (rem\ x\ P))\ P.$

`Hint Resolve` $add\_rem\_eqset\_in.$

`Lemma` $rem\_add\_eq\_eqset$:
$\forall\ x\ (P{:}\texttt{set}),\ eqset\ (rem\ x\ (add\ x\ P))\ (rem\ x\ P).$

```
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 : ∀ P Q : set,
      eqset (inter P Q) (inter Q P).

Lemma inter_empty_left : ∀ P : set,
      eqset empty (inter P empty).

Lemma inter_empty_right : ∀ P : set,
      eqset empty (inter empty P).

Lemma inter_add_left_in : ∀ (a:A) (P Q: set),
      (P a) → eqset (add a (inter P Q)) (inter P (add a Q)).

Lemma inter_add_left_out : ∀ (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*$\rightarrow$set) : set := fun *z* $\Rightarrow$ $\exists$ *i*, *F i z*.

Lemma *gunion_intro* : $\forall$ *I* (*F*:*I*$\rightarrow$set) *i*, *incl* (*F i*) (*gunion F*).

Lemma *gunion_elim* : $\forall$ *I* (*F*:*I*$\rightarrow$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*))-> *incl* (*gunion F*) (*gunion G*).

10.7 Decidable sets

Definition *dec* (*P*:set) := $\forall$ *x*, {*P x*}+{ $\neg$ *P x*}.

Definition *dec2bool* (*P*:set) : *dec P* $\rightarrow$ *A* $\rightarrow$ *bool* :=
fun *p x* $\Rightarrow$ if *p x* then *true* else *false*.

Lemma *compl_dec* : $\forall$ *P*, *dec P* $\rightarrow$ *dec* (*compl P*).

Lemma *inter_dec* : $\forall$ *P Q*, *dec P* $\rightarrow$ *dec Q* $\rightarrow$ *dec* (*inter P Q*).

Lemma *union_dec* : $\forall$ *P Q*, *dec P* $\rightarrow$ *dec Q* $\rightarrow$ *dec* (*union P Q*).

Hint Resolve *compl_dec inter_dec union_dec*.

10.8 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*)-> *size f* $\leq$ *size g*.

Lemma *size_unique* :
$\forall$ (*P*:set)(*f*:*finite P*) (*Q*:set)(*g*:*finite Q*),
(*eqset P Q*)-> *size f* = *size g*.

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* : $\forall$ *P*:set, *finite full* $\rightarrow$ *dec P* $\rightarrow$ *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

```
Add Rec LoadPath "." as ALEA.

Require Export Prog.
Require Export Sets.
Require Export Arith.
Require Import Setoid.
```

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.

Lemma B2U_zero_one: zero_one B2U.

Lemma NB2U_zero_one: zero_one NB2U.

Lemma B2U_zero_one2: ∀ b:bool,
  orc ((if b then 1 else 0) ≡ 0) ((if b then 1 else 0) ≡ 1).

Lemma NB2U_zero_one2: ∀ b:bool,
  orc ((if b then 0 else 1) ≡ 0) ((if b then 0 else 1) ≡ 1).

Hint Immediate B2U_zero_one NB2U_zero_one B2U_zero_one2 NB2U_zero_one2.

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),
```

$cover\ P\ f \to P\ z \to f\ z \equiv 1.$

Lemma $cover\_eq\_zero : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f{:}MF\ A)\ (z{:}A),$
$cover\ P\ f \to \neg\ P\ z \to f\ z \equiv 0.$

Lemma $cover\_orc\_0\_1 : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f{:}MF\ A),$
$cover\ P\ f \to \forall\ x,\ orc\ (f\ x \equiv 0)\ (f\ x \equiv 1).$

Lemma $cover\_zero\_one : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f{:}MF\ A),$
$cover\ P\ f \to zero\_one\ f.$

Lemma $zero\_one\_cover : \forall\ (A{:}\mathtt{Type})(f{:}MF\ A),$
$zero\_one\ f \to cover\ (\mathtt{fun}\ x \Rightarrow 1 \leq f\ x)\ f.$

Lemma $cover\_esp\_mult\_left : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f{:}MF\ A)(p{:}U),$
$cover\ P\ f \to \forall\ x, f\ x\ \&\ p \equiv f\ x \times p.$

Lemma $cover\_esp\_mult\_right : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(p{:}U)(f{:}MF\ A),$
$cover\ P\ f \to \forall\ x, p\ \&\ f\ x \equiv p \times f\ x.$
Hint Immediate $cover\_esp\_mult\_left\ cover\_esp\_mult\_right.$

Lemma $cover\_elim : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f{:}MF\ A),$
$cover\ P\ f \to \forall\ x,\ orc\ (\tilde{}P\ x \wedge f\ x \equiv 0)\ (P\ x \wedge f\ x \equiv 1).$

Lemma $cover\_eq\_one\_elim\_class : \forall\ (A{:}\mathtt{Type})(P\ Q{:}\mathtt{set}\ A)(f{:}MF\ A),$
$cover\ P\ f \to \forall\ z, f\ z \equiv 1 \to class\ (Q\ z) \to incl\ P\ Q \to Q\ z.$

Lemma $cover\_eq\_one\_elim : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f{:}MF\ A),$
$cover\ P\ f \to \forall\ z, f\ z \equiv 1 \to \neg\ \neg\ P\ z.$

Lemma $cover\_eq\_zero\_elim : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f{:}MF\ A)\ (z{:}A),$
$cover\ P\ f \to f\ z \equiv 0 \to \neg\ P\ z.$

Lemma $cover\_unit : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f{:}MF\ A)(a{:}A),$
$cover\ P\ f \to P\ a \to 1 \leq \mu\ (Munit\ a)\ f.$

Lemma $cover\_let : \forall\ (A\ B{:}\mathtt{Type})(m1{:}\ distr\ A)(m2{:}\ A \to distr\ B)\ (P{:}\mathtt{set}\ A)(cP{:}MF\ A)(f{:}MF\ B)(p{:}U),$
$cover\ P\ cP \to (\forall\ x{:}A,\ P\ x \to p \leq \mu\ (m2\ x)\ f) \to (\mu\ m1\ cP) \times p \leq \mu\ (Mlet\ m1\ m2)\ f.$

Lemma $cover\_let\_one : \forall\ (A\ B{:}\mathtt{Type})(m1{:}\ distr\ A)(m2{:}\ A \to distr\ B)\ (P{:}\mathtt{set}\ A)(cP{:}MF\ A)(f{:}MF\ B)(p{:}U),$
$cover\ P\ cP \to 1 \leq \mu\ m1\ cP \to (\forall\ x{:}A,\ P\ x \to p \leq \mu\ (m2\ x)\ f) \to p \leq \mu\ (Mlet\ m1\ m2)\ f.$

Lemma $cover\_incl\_fle : \forall\ (A{:}\mathtt{Type})(P\ Q{:}\mathtt{set}\ A)(f\ g{:}MF\ A),$
$cover\ P\ f \to cover\ Q\ g \to incl\ P\ Q \to f \leq g.$

Lemma $cover\_same\_feq{:}\ \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f\ g{:}MF\ A),$
$cover\ P\ f \to cover\ P\ g \to f \equiv g.$

Lemma $cover\_incl\_le : \forall\ (A{:}\mathtt{Type})(P\ Q{:}\mathtt{set}\ A)(f\ g{:}MF\ A)\ x,$
$cover\ P\ f \to cover\ Q\ g \to incl\ P\ Q \to f\ x \leq g\ x.$

Lemma $cover\_same\_eq : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f\ g{:}MF\ A)\ x,$
$cover\ P\ f \to cover\ P\ g \to f\ x \equiv g\ x.$

Lemma $cover\_eqset\_stable : \forall\ (A{:}\mathtt{Type})(P\ Q{:}\mathtt{set}\ A)(EQ{:}eqset\ P\ Q)(f{:}MF\ A),$
$cover\ P\ f \to cover\ Q\ f.$

Lemma $cover\_eq\_stable : \forall\ (A{:}\mathtt{Type})(P{:}\mathtt{set}\ A)(f\ g{:}MF\ A),$
$cover\ P\ f \to f \equiv g \to cover\ P\ g.$

Lemma $cover\_eqset\_eq\_stable : \forall\ (A{:}\mathtt{Type})(P\ Q{:}\mathtt{set}\ A)(f\ g{:}MF\ A),$
$cover\ P\ f \to eqset\ P\ Q \to f \equiv g \to cover\ Q\ g.$

Add $Parametric\ Morphism\ (A{:}\mathtt{Type}) : (cover\ (A{:=}A))$
with $signature\ eqset\ (A{:=}A) \Longrightarrow Oeq \Longrightarrow iff$ as $cover\_eqset\_compat.$
Save.

Lemma $cover\_union : \forall\ (A{:}\mathtt{Type})(P\ Q{:}\mathtt{set}\ A)(f\ g : MF\ A),$
$cover\ P\ f \to cover\ Q\ g \to cover\ (union\ P\ Q)\ (fplus\ f\ g).$

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*).

Lemma *cover_full* : ∀ (*A*:Type), *cover* (*full A*) (*fone A*).

Lemma *cover_comp* : ∀ (*A B*:Type)(*h*:*A* → *B*)(*P*:set *B*)(*f*:*MF B*),
cover P f → *cover* (fun *a* ⇒ *P* (*h a*)) (fun *a* ⇒ *f* (*h a*)).

Covering and image This direction requires a covering function for the property Lemma *im_range_elim A B* (*f* : *A* → *B*) :
∀ (*d* : *distr A*) (*P* : *B* → Prop) (*cP* : *B* → *U*),
cover P cP → *range P* (*im_distr f d*) → *range* (fun *x* ⇒ *P* (*f x*)) *d*.
Hint Resolve *im_range*.

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.
Hint Resolve *carac_zero carac_one*.

Lemma *carac_compl* : ∀ (*A*:Type)(*P*:*A* → Prop)(*Pdec*:*dec P*),
carac (*compl_dec Pdec*) ≡ *finv* (*carac Pdec*).
Hint Resolve *carac_compl*.

Lemma *cover_dec* : ∀ (*A*:Type)(*P*:set *A*)(*Pdec* : *dec P*), *cover P* (*carac Pdec*).
Hint Resolve *cover_dec*.

Lemma *carac_zero_one* : ∀ (*A*:Type)(*P*:set *A*)(*Pdec* : *dec P*), *zero_one* (*carac Pdec*).
Hint Resolve *carac_zero_one*.

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 Distribution by restriction

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 $Mrestr\_simpl : \forall\ A\ cp\ (m{:}M\ A)\ f,\ Mrestr\ cp\ m\ f = cp \times (m\ f).$

Lemma $Mrestr0 : \forall\ A\ cP\ (m{:}M\ A),\ cP \leq 0 \rightarrow \forall\ f,\ Mrestr\ cP\ m\ f \equiv 0.$

Lemma $Mrestr1 : \forall\ A\ cP\ (m{:}M\ A),\ 1 \leq cP \rightarrow \forall\ f,\ Mrestr\ cP\ m\ f \equiv m\ f.$

Definition $distr\_restr : \forall\ A\ (P{:}\texttt{Prop})\ (cp{:}U)\ (m{:}M\ A),$
$((P \rightarrow 1 \leq cp) \wedge (\tilde{}\ P \rightarrow cp \leq 0)) \rightarrow (P \rightarrow stable\_inv\ m) \rightarrow$
$(P \rightarrow stable\_plus\ m) \rightarrow (P \rightarrow stable\_mult\ m) \rightarrow (P \rightarrow continuous\ m)$
$\rightarrow distr\ A.$
Defined.

Lemma $distr\_restr\_simpl : \forall\ A\ (P{:}\texttt{Prop})\ (cp{:}U)\ (m{:}M\ A)$
$(Hp{:}\ (P \rightarrow 1 \leq cp) \wedge (\tilde{}\ P \rightarrow cp \leq 0))\ (Hinv{:}P \rightarrow stable\_inv\ m)$
$(Hplus{:}P \rightarrow stable\_plus\ m)(Hmult{:}P \rightarrow stable\_mult\ m)(Hcont{:}P \rightarrow continuous\ m)\ f,$
$\mu\ (distr\_restr\ cp\ Hp\ Hinv\ Hplus\ Hmult\ Hcont)\ f = cp \times m\ f.$

Modular reasoning on programs

Lemma $range\_cover : \forall\ A\ (P{:}A \rightarrow \texttt{Prop})\ d\ cP,\ range\ P\ d \rightarrow cover\ P\ cP \rightarrow$
$\forall\ f,\ \mu\ d\ f \equiv \mu\ d\ (\texttt{fun}\ x \Rightarrow cP\ x \times f\ x).$

Lemma $mu\_cut : \forall\ (A{:}\texttt{Type})(m{:}distr\ A)(P{:}\texttt{set}\ A)(cP\ f\ g{:}MF\ A),$
$cover\ P\ cP \rightarrow (\forall\ x,\ P\ x \rightarrow f\ x \equiv g\ x) \rightarrow 1 \leq \mu\ m\ cP$
$\rightarrow \mu\ m\ f \equiv \mu\ m\ g.$

11.4 Uniform measure on finite sets

Section $SigmaFinite.$
Variable $A{:}\texttt{Type}.$
Variable $decA : \forall\ x\ y{:}A,\ \{\ x{=}y\ \}{+}\{\ \neg\ x{=}y\ \}.$

Section $RandomFinite.$

11.4.1 Distribution for *random_fin P* over $\{k{:}nat \mid k \leq n\}$

The distribution associated to $random\_fin\ P$ is f -> $sigma\ (a\ \texttt{in}\ P)\ [1/]1{+}n\ (f\ a)$ with $[n{+}1]$ *the size of* $[P]$ *we cannot factorize* $[\ [1/]1{+}n\ ]$ *because of possible overflow*

Fixpoint $sigma\_fin\ (f{:}A \rightarrow U\ )(P{:}\ A \rightarrow \texttt{Prop})(FP{:}finite\ P)\{\texttt{struct}\ FP\} : U :=$
match FP with
$\mid (fin\_eq\_empty\ eq) \Rightarrow 0$
$\mid (fin\_eq\_add\ x\ Q\ nQx\ FQ\ eq) \Rightarrow f\ x + sigma\_fin\ f\ FQ$
end.

Definition $retract\_fin\ (P{:}A \rightarrow \texttt{Prop})\ (f{:}A \rightarrow U) :=$
$\forall\ Q\ (FQ{:}\ finite\ Q),\ incl\ Q\ P \rightarrow \forall\ x,\ \neg\ (Q\ x) \rightarrow P\ x$
$\rightarrow f\ x \leq [1\text{-}](sigma\_fin\ f\ FQ).$

Lemma $retract\_fin\_inv$:
$\forall\ (P{:}\ A \rightarrow \texttt{Prop})\ (f{:}\ A \rightarrow U),$
$retract\_fin\ P\ f \rightarrow \forall\ Q\ (FQ{:}\ finite\ Q),\ incl\ Q\ P \rightarrow$
$\forall\ x,\ \neg\ (Q\ x) \rightarrow P\ x \rightarrow sigma\_fin\ f\ FQ \leq [1\text{-}]f\ x.$

Hint Immediate $retract\_fin\_inv.$

Lemma $retract\_fin\_incl : \forall\ P\ Q\ f,\ retract\_fin\ P\ f \rightarrow incl\ Q\ P \rightarrow retract\_fin\ Q\ f.$

Lemma $sigma\_fin\_monotonic : \forall\ (f\ g : A \rightarrow U)(P{:}\ A \rightarrow \texttt{Prop})(FP{:}\ finite\ P),$

```
       (∀ 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.

Instance sigma_fin_mon : ∀ (P: A → Prop)(FP:finite P),
        monotonic (fun (f:MF A) ⇒ sigma_fin f FP).
Save.

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 → sigma_fin f FP ≡ f a + sigma_fin f (finite_rem decA a FP).

Lemma sigma_fin_incl : ∀ f P (FP: finite P) Q (FQ: finite Q),
      incl P Q → sigma_fin f FP ≤ sigma_fin f FQ.

Lemma sigma_fin_unique : ∀ f P Q (FP: finite P) (FQ: finite Q),
      eqset P Q → sigma_fin f FP ≡ sigma_fin f FQ.

Lemma sigma_fin_cte : ∀ c P (FP:finite P),
       sigma_fin (fun _ ⇒ c) FP ≡ (size FP) */ c.

Definition Sigma_fin P (FP:finite P) := mon (fun (f:MF A) ⇒ sigma_fin f FP).

Lemma Sigma_fin_simpl : ∀ P (FP:finite P) f, Sigma_fin FP f = sigma_fin f FP.

Lemma sigma_fin_continuous : ∀ P (FP:finite P),
            continuous (Sigma_fin FP).
```

11.4.2 Definition and Properties of *random_fin*

```
Variable P : A → Prop.
Variable FP : finite P.
Let s:= (size FP - 1)%nat.

Lemma pred_size_le : (size FP ≤S s)%nat.
```

Hint Resolve *pred_size_le*.

Lemma *pred_size_eq* : *notempty* $P \to$ *size FP* = *S s*.

Instance *fmult_mon* : ∀ *A k*, *monotonic* (*fmult* (*A*:=*A*) *k*).
Save.

Definition *random_fin* : *M A* := *Sigma_fin FP* @ (*Fmult A* ([1/]1+*s*)).

Lemma *random_fin_simpl* : ∀ (*f*:*MF A*),
    random_fin f = *sigma_fin* (fun *x* ⇒ ([1/]1+*s*) × *f x*) *FP*.

Lemma *fnth_retract_fin*:
        ∀ *n*, (*size FP* ≤ *S n*)%*nat* → *retract_fin P* (fun _ ⇒ [1/]1+*n*).

Lemma *random_fin_stable_inv* : *stable_inv random_fin*.

Lemma *random_fin_stable_plus* : *stable_plus random_fin*.

Lemma *random_fin_stable_mult* : *stable_mult random_fin*.

Lemma *random_fin_monotonic* : *monotonic random_fin*.

Lemma *random_fin_continuous* : *continuous random_fin*.

Definition *Random_fin* : *distr A*.
Defined.

Lemma *Random_fin_simpl* : μ *Random_fin* = *random_fin*.

Lemma *random_fin_total* : *notempty P* → μ *Random_fin* (*fone A*) ≡ 1.
End *RandomFinite*.

Lemma *random_fin_cover* :
    ∀ *P Q* (*FP*:*finite P*) (*decQ*:*dec Q*),
        μ (*Random_fin FP*) (*carac decQ*) ≡ *size* (*finite_inter decQ FP*) */ [1/]1+(*size FP*-1)%*nat*.

Lemma *random_fin_P* : ∀ *P* (*FP*:*finite P*) (*decP*:*dec P*),
        notempty P → μ (*Random_fin FP*) (*carac decP*) ≡ 1.

End *SigmaFinite*.

11.5 Properties of the Random distribution

Definition *dec_le* (*n*:*nat*) : *dec* (fun *x* ⇒ (*x* ≤ *n*)%*nat*).
Defined.

Definition *dec_lt* (*n*:*nat*) : *dec* (fun *x* ⇒ (*x* < *n*)%*nat*).
Defined.

Definition *dec_gt* : ∀ *x*, *dec* (*lt x*).
Defined.

Definition *dec_ge* : ∀ *x*, *dec* (*le x*).
Defined.

Definition *carac_eq n* := *carac* (*eq_nat_dec n*).
Definition *carac_le n* := *carac* (*dec_le n*).
Definition *carac_lt n* := *carac* (*dec_lt n*).
Definition *carac_gt n* := *carac* (*dec_gt n*).
Definition *carac_ge n* := *carac* (*dec_ge n*).

Definition *is_eq* (*n*:*nat*) : *cover* (fun *x* ⇒ *n* = *x*) (*carac_eq n*) := *cover_dec* (*eq_nat_dec n*).
Definition *is_le* (*n*:*nat*) : *cover* (fun *x* ⇒ (*x* ≤*n*)%*nat*) (*carac_le n*) := *cover_dec* (*dec_le n*).
Definition *is_lt* (*n*:*nat*) : *cover* (fun *x* ⇒ (*x* < *n*)%*nat*) (*carac_lt n*) := *cover_dec* (*dec_lt n*).
Definition *is_gt* (*n*:*nat*) : *cover* (fun *x* ⇒ (*n* < *x*)%*nat*) (*carac_gt n*):= *cover_dec* (*dec_gt n*).
Definition *is_ge* (*n*:*nat*) : *cover* (fun *x* ⇒ (*n* ≤ *x*)%*nat*) (*carac_ge n*) := *cover_dec* (*dec_ge n*).

Lemma *carac_gt_S* :

```
  ∀ x y, carac_gt (S y) (S x) ≡ carac_gt y x.
```

```
Lemma carac_lt_S : ∀ x y, carac_lt (S x) (S y) ≡ carac_lt x y.
```

```
Lemma carac_le_S : ∀ x y, carac_le (S x) (S y) ≡ carac_le x y.
```

```
Lemma carac_ge_S : ∀ x y, carac_ge (S x) (S y) ≡ carac_ge x y.
```

```
Lemma carac_eq_S : ∀ x y, carac_eq (S x) (S y) ≡ carac_eq x y.
```

```
Lemma carac_lt_0 : ∀ y, carac_lt 0 y ≡ 0.
```

```
Lemma carac_lt_zero : carac_lt 0 ≡ fzero _.
```

lifting "if then else".

```
Lemma carac_if_compat : ∀ A (P:set A) (Pdec : dec P) (t:bool) u v,
  (carac Pdec (if t then u else v))
  ≡
  (if t
    then (carac Pdec u)
    else (carac Pdec v)).
```

```
Lemma carac_lt_if_compat : ∀ x (t:bool) u v,
  (carac_lt x (if t then u else v))
  ≡
  (if t
    then (carac_lt x u)
    else (carac_lt x v)).
```

```
Hint Resolve carac_le_S carac_eq_S carac_lt_S carac_ge_S carac_gt_S carac_lt_0 carac_lt_zero.
```

```
Instance carac_ge_mon (n:nat) : monotonic (carac_ge n).
Save.
```

```
Definition Carac_ge (n:nat) : nat -m> U := mon (carac_ge n).
```

```
Lemma dec_inter : ∀ A (P Q : set A), dec P → dec Q → dec (inter P Q).
```

```
Lemma dec_union : ∀ A (P Q : set A), dec P → dec Q → dec (union P Q).
```

```
Lemma carac_conj : ∀ A (P Q : set A) (dP:dec P) (dQ:dec Q),
      carac (dec_inter dP dQ) ≡ fconj (carac dP) (carac dQ).
```

```
Lemma carac_plus : ∀ A (P Q : set A) (dP:dec P) (dQ:dec Q),
      carac (dec_union dP dQ) ≡ fplus (carac dP) (carac dQ).
```

Count the number of elements between 0 and n-1 which satisfy P

```
Fixpoint nb_elts (P:nat → Prop)(Pdec : dec P)(n:nat) {struct n} : nat :=
match n with
   0 ⇒ 0%nat
| S n ⇒ if Pdec n then (S (nb_elts Pdec n)) else (nb_elts Pdec n)
end.
```

```
Lemma nb_elts_true : ∀ (P:nat → Prop)(Pdec : dec P)(n:nat),
        (∀ k, (k < n)%nat → P k) → nb_elts Pdec n =n.
Hint Resolve nb_elts_true.
```

```
Lemma nb_elts_false : ∀ P,∀ Pdec:dec P,∀ n,
  (∀ x, (x<n)%nat → ¬ P x) → nb_elts Pdec n = 0%nat.
```

- 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),
     μ (Random n) (carac Pdec) ≡ (nb_elts Pdec (S n)) */ [1/]1+n.
```

```
Lemma nb_elts_lt_le : ∀ k n, (k ≤ n)%nat → nb_elts (dec_lt k) n = k.
```

```
Lemma nb_elts_lt_ge : ∀ k n, (n ≤ k)%nat → nb_elts (dec_lt k) n = n.
```

Lemma $nb\_elts\_eq\_nat\_ge$:$\forall$ n k,
$(n \leq k)\%nat \rightarrow nb\_elts\ (eq\_nat\_dec\ k)\ n = 0\%nat$.

Lemma beq_nat_neq: $\forall$ x y : nat, $x \neq y \rightarrow false = beq\_nat\ x\ y$.

Lemma nb_elt_eq :$\forall$ n k,
$(k < n)\%nat \rightarrow nb\_elts\ (eq\_nat\_dec\ k)\ n = 1\%nat$.

Hint Resolve $nb\_elts\_lt\_ge$ $nb\_elts\_lt\_le$ $nb\_elts\_eq\_nat\_ge$ nb_elt_eq.

Lemma $Random\_lt$: $\forall$ n k, $\mu\ (Random\ n)\ (carac\_lt\ k) \equiv k$ */ [1/]1+n.

Hint Resolve $Random\_lt$.

Lemma $Random\_le$: $\forall$ n k, $\mu\ (Random\ n)\ (carac\_le\ k) \equiv (S\ k)$ */ [1/]1+n.

Hint Resolve $Random\_le$.

Lemma $Random\_eq$: $\forall$ n k, $(k \leq n)\%nat \rightarrow \mu\ (Random\ n)\ (carac\_eq\ k) \equiv 1$ */ [1/]1+n.

Hint Resolve $Random\_eq$.

11.6 Properties of distributions and set

Section $PickElemts$.
Variable A : Type.
Variable P : $A \rightarrow$ Prop.
Variable cP : $A \rightarrow U$.
Hypothesis $coverP$: $cover\ P\ cP$.
Variable ceq : $A \rightarrow A \rightarrow U$.
Hypothesis $covereq$: $\forall$ x, $cover\ (eq\ x)\ (ceq\ x)$.

Variable d : $distr\ A$.

Variable k : U.

Hypothesis $deqP$: $\forall$ x, $P\ x \rightarrow k \leq \mu\ d\ (ceq\ x)$.

Lemma d_coverP : $\forall$ x, $P\ x \rightarrow k \leq \mu\ d\ cP$.

Lemma d_coverP_exists : $(\exists\ x,\ P\ x) \rightarrow k \leq \mu\ d\ cP$.

Lemma $d\_coverP\_not\_empty$: $\neg\ (\forall\ x,\ \neg\ P\ x) \rightarrow k \leq \mu\ d\ cP$.

End $PickElemts$.

12 IsDiscrete.v: distributions over discrete domains

Contributed by David Baelde. This has been adapted from Certicrypt : Santiago Zanella and Benjmain Grégoire.

12.1 Definition of discrete domains and decidable equalities

Class $Discrete\_domain$ (A:Type) :=
{ $points$: $nat \rightarrow A$;
$points\_surj$: $\forall$ x, $\exists$ n, $points\ n = x$ }.

Class $DecidEq$ (A:Type) :=
{ eq_dec : $\forall$ x y : A, { x=y }+{ $x \neq y$ } }.

12.2 Useful functions on discrete domains

Section $Discrete$.

Variable A : Type.

Hypothesis *A_discrete* : *Discrete_domain A*.
Hypothesis *A_decidable* : *DecidEq A*.

Definition *uequiv* : $A \to MF\ A :=$ fun $a \Rightarrow$ *carac* (*eq_dec a*).

Lemma *cover_uequiv* : ∀ *a*, *cover* (*eq a*) (*uequiv a*).

not_first_repr k decide if *points k* is not the first point in is class, in that case *points k* is not the representant of the class

Definition *not_first_repr k* := *sigma* (fun *i* ⇒ *uequiv* (*points k*) (*points i*)) *k*.

Lemma *cover_not_first_repr* :
cover (fun *k* ⇒ *exc* (fun *k0* ⇒ (*k0* < *k*)%*nat* ∧ (*points k*) = (*points k0*))) *not_first_repr*.

in_classes a decides if *a* is in relation with one element of *points* Definition *in_classes a* := *serie* (fun *k* ⇒ *uequiv a* (*points k*)).

Definition *In_classes a* := *exc* (fun *k* ⇒ *a* = (*points k*)).

Lemma *cover_in_classes* : *cover In_classes in_classes*.

in_class a k decides if *a* is in relation with *points k* and *points k* is the representant of it class Definition *in_class a k* := [1-] (*not_first_repr k*) × *uequiv* (*points k*) *a*.

Definition *In_class a k* :=
(*points k*) = *a* ∧
(∀ *k0*, (*k0* < *k*)%*nat* → ¬ (*points k* = *points k0*)).

Lemma *cover_in_class* : ∀ *a*, *cover* (*In_class a*) (*in_class a*).

Lemma *in_class_wretract* : ∀ *x*, *wretract* (*in_class x*).

Lemma *in_classes_refl* : ∀ *k*, *in_classes* (*points k*) ≡ 1.

Lemma *cover_serie_in_class* : *cover* (fun *a* ⇒ *exc* (*In_class a*)) (fun *a* ⇒ *serie* (*in_class a*)).

Lemma *in_classes_in_class* : ∀ *a*, *in_classes a* ≡ *serie* (*in_class a*).

12.3 Any distribtion on a discrete domain is discrete

Variable *d* : *distr A*.

Lemma *range_in_classes* : *range In_classes d*.

Definition *coeff k* := ([1-] (*not_first_repr k*)) × μ *d* (*uequiv* (*points k*)).

Lemma *mu_discrete* : μ *d* ≡ *discrete coeff points*.

Lemma *coeff_retract* : *wretract coeff*.

Theorem *domain_is_discrete* : *is_discrete d*.

End *Discrete*.

Implicit Arguments *domain_is_discrete* [[*A*] [*A_discrete*] [*A_decidable*]].

12.4 Instances for common discrete and decidable domains

Instance *nat_discrete* : *Discrete_domain nat*.

Instance *nat_decid_eq* : *DecidEq nat* := *Build_DecidEq eq_nat_dec*.

Definition *bool_points* := *beq_nat* 0.
Instance *bool_discrete* : *Discrete_domain bool*.

Require Import *Bool*.

Instance *bool_decid_eq* : *DecidEq bool* := *Build_DecidEq bool_dec*.

12.5 Building a bijection between *nat* and *nat* × *nat*

```
Require Import Even.
Require Import Div2.

Lemma bij_n_nxn_aux : ∀ k,
 (0 < k)%nat → sigT (fun (i:nat) ⇒ {j : nat | k = (exp2 i × (2 × j + 1))%nat}).

Definition bij_n_nxn k :=
 match @bij_n_nxn_aux (S k) (lt_O_Sn k) with
 | existT i (∃ j _) ⇒ (i, j)
 end.

Lemma mult_eq_reg_l : ∀ n m p,
 (0 < p → p × n = p × m → n = m)%nat.

Lemma even_exp2 : ∀ n, even (exp2 (S n)).

Lemma odd_2p1 : ∀ n, odd (2 × n + 1).

Lemma bij_surj : ∀ i j, ∃ k,
 bij_n_nxn k = (i, j).
```

12.6 The product of two discrete domains is discrete

```
Instance prod_discrete : ∀ A B,
   Discrete_domain A → Discrete_domain B → Discrete_domain (A×B).
```

13 BinCoeff.v: Binomial coefficients

Contributed by David Baelde, 2011

```
Require Import Arith.
Require Import Omega.
```

13.1 Definition 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.
```

13.2 Properties of binomial coefficients

```
Lemma comb_0_n : ∀ n, comb 0 n = 1%nat.

Lemma comb_not_le : ∀ n k, (S n ≤ k)%nat → 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).

Notation H := (fun n k ⇒ comb (S k) (S n) × (S k) = comb k (S n) × (S n - k)).
Notation V := (fun n k ⇒ comb k (S n) × (S n - k) = comb k n × (S n)).

Lemma comb_relations : ∀ n k, H n k ∧ V n k.
```

Lemma *comb_incr_n* : ∀ *n k*, *comb k* (*S n*) × (*S n* - *k*) = *comb k n* × (*S n*).

Lemma *comb_incr_k* : ∀ *n k*, *comb* (*S k*) (*S n*) × (*S k*) = *comb k* (*S n*) × (*S n* - *k*).

Lemma *comb_fact* : ∀ *n k*, *k*≤*n* → *comb k n* × *fact k* × *fact* (*n*-*k*) = *fact n*.

Lemma *comb_le_0_lt* : ∀ *k n*, *k* ≤ *n* → 0 < *comb k n*.

Lemma *mult_simpl_right* : ∀ *m n p*, 0 < *p* → *m* × *p* = *n* × *p* → *m* = *n*.

Corollary *comb_symmetric* : ∀ *k n*, *k*≤*n* → *comb k n* = *comb* (*n*-*k*) *n*.

Lemma *mult_lt_compat_l* : ∀ *n m p* : *nat*, *n* < *m* → 0 < *p* → *p* × *n* < *p* × *m*.

Lemma *comb_monotonic_k* : ∀ *k n k'*, 0<*n* → *k*≤*k'* → 2**k'*≤*n* → *comb k n* ≤ *comb k' n*.

Lemma *comb_monotonic_n* : ∀ *k n n'*, *k*≤*n* → *n*≤*n'* → *comb k n* ≤ *comb k n'*.

Lemma *comb_monotonic* :
  ∀ *k n k' n'*, 0<*n* → *k*≤*n* → *k*≤*k'* → 2**k'*≤*n'* → *n*≤*n'* → *comb k n* ≤ *comb k' n'*.

Lemma *comb_max_half* : ∀ *k n*, *comb k n* ≤ *comb* (*Div2.div2 n*) *n*.

14 Bernoulli.v: Simulating Bernoulli and Binomial distributions

Require Export *Cover*.
Require Export *Misc*.
Require Export *BinCoeff*.

14.1 Program for computing a Bernoulli distribution

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

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

Hypothesis *dec_demi* : ∀ *x* : *U*, {*x* < [1/2]}+{[1/2] ≤ *x*}.

Instance *Fbern_mon* : *monotonic*
    (fun (*f*:*U* → *distr bool*) *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*)).
Save.

Definition *Fbern* : (*U* → *distr bool*) -*m*> (*U* → *distr bool*)
    := *mon* (fun *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* → *distr bool* := *Mfix Fbern*.

14.2 *fc p n k* is defined as ($C(k,n)$ p^k (1-p)^(n-k))

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* $\Longrightarrow$ *eq* $\Longrightarrow$ *eq* $\Longrightarrow$ *Oeq*
as *fc_eq_compat*.
Save.

Hint Resolve *fc_eq_compat*.

14.2.1 Sum of *fc* objects

Lemma *sigma_fc0* : $\forall$ *n k*, *sigma* (*fc* 0 *n*) (*S k*) $\equiv$ 1.

Intermediate results for inductive proof of [1-]*p*^*n* $\equiv$ *sigma* (*fc p n*) *n*

Lemma *fc_retract* :
    $\forall$ *p n*, [1-]*p*^*n* $\equiv$ *sigma* (*fc p n*) *n* $\rightarrow$ *retract* (*fc p n*) (*S n*).
Hint Resolve *fc_retract*.

Lemma *fc_Nmult_def* :
    $\forall$ *p n k*, ([1-]*p*^*n* $\equiv$ *sigma* (*fc p n*) *n*) $\rightarrow$
        Nmult_def (*comb k n*) (*p*^*k* $\times$ ([1-]*p*) ^(*n*-*k*)).
Hint Resolve *fc_Nmult_def*.

Lemma *fcp_S* :
    $\forall$ *p n k*, ([1-]*p*^*n* $\equiv$ *sigma* (*fc p n*) *n*)
        $\rightarrow$ *fc p* (*S n*) (*S k*) $\equiv$ *p* $\times$ (*fc p n k*) + ([1-]*p*) $\times$ (*fc p n* (*S k*)).

Lemma *sigma_fc_1*
  : $\forall$ *p n*, [1-]*p*^*n* $\equiv$ *sigma* (*fc p n*) *n* $\rightarrow$ 1 $\equiv$ *sigma* (*fc p n*) (*S n*).
Hint Resolve *sigma_fc_1*.

Main result : [1-](*p*^*n*) $\equiv$ *sigma* (*k*=0..(*n*-1)) *C*(*k*,*n*) *p*^*k* (1-*p*)^(*n*-*k*)

Lemma *Uinv_exp* : $\forall$ *p n*, [1-](*p*^*n*) $\equiv$ *sigma* (*fc p n*) *n*.

Hint Resolve *Uinv_exp*.

Lemma *Nmult_comb*
  : $\forall$ *p n k*, *Nmult_def* (*comb k n*) (*p* ^ *k* $\times$ ([1-] *p*) ^ (*n* - *k*)).
Hint Resolve *Nmult_comb*.

14.3 Program for computing a binomial distribution

Recursive definition of binomial distribution using bernoulli (*binomial p n*) gives *k* with probability *C*(*k*,*n*) *p*^*k* (1-*p*)^(*n*-*k*)

```
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.
```

14.4 Properties of the Bernoulli program

```
Lemma Fbern_simpl : ∀ 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).
```

14.4.1 Proofs using fixpoint rules

```
Instance Mubern_mon : ∀ (q: bool → U),
  monotonic
   (fun bern (p:U) ⇒ if dec_demi p then [1/2]*(q false)+[1/2]*(bern (p+p))
                                   else [1/2]*(bern (p&p)) + [1/2]*(q true)).
Save.
```

```
Definition Mubern (q: bool → U) : MF U -m> MF U
  := mon (fun bern (p:U) ⇒ if dec_demi p then [1/2]*(q false)+[1/2]*(bern (p+p))
                                         else [1/2]*(bern (p&p)) + [1/2]*(q true)).
```

```
Lemma Mubern_simpl : ∀ (q: bool → 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).
```

Mubern commutes with the measure of Fbern

```
Lemma Mubern_eq : ∀ (q: bool → U) (f:U → distr bool) (p:U),
              μ (Fbern f p) q ≡ Mubern q (fun y ⇒ μ (f y) q) p.
Hint Resolve Mubern_eq.
```

```
Lemma Bern_eq :
     ∀ q : bool → U, ∀ p, μ (bernoulli p) q ≡ mufix (Mubern q) p.
Hint Resolve Bern_eq.
```

```
Lemma Bern_commute : ∀ q : bool → U,
    mu_muF_commute_le Fbern (fun (x:U) ⇒ q) (Mubern q).
Hint Resolve Bern_commute.
```

bernoulli terminates with probability 1

```
Lemma Bern_term : ∀ p, μ (bernoulli p) (fone bool) ≡ 1.
Hint Resolve Bern_term.
```

14.4.2 p is an invariant of Mubern qtrue

```
Lemma MuBern_true : ∀ p, Mubern B2U (fun q ⇒ q) p ≡ p.
Hint Resolve MuBern_true.
```

```
Lemma MuBern_false : ∀ p, Mubern (finv B2U) (finv (fun q ⇒ q)) p ≡ [1-]p.
Hint Resolve MuBern_false.
```

```
Lemma Mubern_inv : ∀ (q: bool → U) (f:U → U) (p:U),
       Mubern (finv q) (finv f) p ≡ [1-] Mubern q f p.
```

prob(*bernoulli* = *true*) = *p*

```
Lemma Bern_true : ∀ p, μ (bernoulli p) B2U ≡ p.
```

prob(*bernoulli* = *false*) = 1-*p*

```
Lemma Bern_false : ∀ p, μ (bernoulli p) NB2U ≡ [1-]p.
```

14.4.3 Direct proofs using lubs

Invariant *pmin p* with *pmin p n* = *p* - $\frac{1}{2}$ ^ *n*

Property : ∀ *p*, *ok p* (*bernoulli p*) χ (.=*true*)

```
Definition qtrue (p:U) := B2U.
Definition qfalse (p:U) := NB2U.
```

```
Lemma bernoulli_true : okfun (fun p ⇒ p) bernoulli qtrue.
```

Property : ∀ *p*, *ok* (1-*p*) (*bernoulli p*) (χ (.=*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*, μ (*bernoulli p*) (*qtrue p*) ≡ *p*.

Lemma *bernoulli_eq_false* : ∀ *p*, μ (*bernoulli p*) (*qfalse p*)== [1-]*p*.

Lemma *bernoulli_eq* : ∀ *p f*,
  μ (*bernoulli p*) *f* ≡ *p* × *f true* + ([1-]*p*) × *f false*.

Lemma *bernoulli_total* : ∀ *p* , μ (*bernoulli p*) (*fone bool*)==1.

14.5 Properties of Binomial distribution

prob(*binomial p n* = *k*) = *C*(*k*,*n*) *p* ^ *k* (1-*p*)^(*n*-*k*)

Lemma *binomial_eq_k* :
  ∀ *p n k*, μ (*binomial p n*) (*carac_eq k*) ≡ *fc p n k*.

  prob(*binomial p n* ≤ *n*) = 1

Lemma *binomial_le_n* :
  ∀ *p n*, 1 ≤ μ (*binomial p n*) (*carac_le n*).

  prob(*binomial p* (*S n*) ≤ *S k*) = *p prob*(*binomial p n* ≤ *k*) + (1-*p*) *prob*(*binomial p n* ≤ *S k*)

Lemma *binomial_le_S* : ∀ *p n k*,
  μ (*binomial p* (*S n*)) (*carac_le* (*S k*)) ≡
  p × (μ (*binomial p n*) (*carac_le k*)) + ([1-]*p*) × (μ (*binomial p n*) (*carac_le* (*S k*))).

  prob(*binomial p* (*S n*) < *S k*) = *p prob*(*binomial p n* < *k*) + (1-*p*) *prob*(*binomial p n* < *S k*)

Lemma *binomial_lt_S* : ∀ *p n k*,
  μ (*binomial p* (*S n*)) (*carac_lt* (*S k*)) ≡
  p × (μ (*binomial p n*) (*carac_lt k*)) + ([1-]*p*) × (μ (*binomial p n*) (*carac_lt* (*S k*))).

15 DistrTactic.v: tactics for reasoning on distributions.

Contributed by Pierre Courtieu CNAM

The tactics to use are

- *simplmu* for one step simplification,
- *rsimplmu* for repeated simplifications.
- These two tactics can be cloned and extended using *simplmu_arg*.

```
Hint Extern 2 ⇒ Usimpl.
Ltac simpl_mu_rewrite tacsubgoals := first [
progress setoid_rewrite Umult_sym_cst|rewrite Umult_sym_cst|
progress setoid_rewrite Mif_eq2|rewrite Mif_eq2|
progress setoid_rewrite Bern_true|rewrite Bern_true|
progress setoid_rewrite Bern_false|rewrite Bern_false|
progress setoid_rewrite Mlet_simpl|rewrite Mlet_simpl|
progress setoid_rewrite Munit_simpl|rewrite Munit_simpl|

progress setoid_rewrite bary_refl_feq;[|complete auto]|rewrite bary_refl_feq;[|complete auto]|

progress setoid_rewrite Uinv_inv|rewrite Uinv_inv|
progress setoid_rewrite bernoulli_eq|rewrite bernoulli_eq|
progress setoid_rewrite binomial_lt_S|rewrite binomial_lt_S|
```

```
progress setoid_rewrite carac_lt_S|rewrite carac_lt_S|


progress setoid_rewrite mu_stable_mult2|rewrite mu_stable_mult2|
progress setoid_rewrite mon_simpl|rewrite mon_simpl|

progress setoid_rewrite im_distr_simpl|rewrite im_distr_simpl|
progress setoid_rewrite Mchoice_simpl|rewrite Mchoice_simpl|
progress setoid_rewrite Random_total|rewrite Random_total|
progress setoid_rewrite discrete_simpl|rewrite discrete_simpl|
progress setoid_rewrite Discrete_simpl|rewrite Discrete_simpl|
progress setoid_rewrite Flip_simpl|rewrite Flip_simpl|

progress setoid_rewrite (@mu_fzero_eq _ _) | rewrite (@mu_fzero_eq _ _) |
progress setoid_rewrite mu_fzero_eq |rewrite mu_fzero_eq |
progress setoid_rewrite Mlet_unit|rewrite Mlet_unit|
progress setoid_rewrite Mlet_assoc|rewrite Mlet_assoc|

progress setoid_rewrite mu_stable_plus2;[|complete tacsubgoals ] | rewrite mu_stable_plus2;[|complete tacsubgoals ]|

progress setoid_rewrite carac_lt_if_compat | rewrite carac_lt_if_compat
].
```

Try simplification of Oeq and Ole at top level.

```
Ltac simplmu_aux :=
  match goal with
     | ⊢ (Ole (fmont (μ ?d1) ?f) (fmont (μ ?d2) ?g)) ⇒ apply (mu_le_compat (m1:=d1) (m2:=d2) (Ole_refl d1) (f:=f) (g:=g)); intro
     | ⊢ (Oeq (fmont (μ ?d1) ?f) (fmont (μ ?d2) ?g)) ⇒ apply (mu_eq_compat (m1:=d1) (m2:=d2) (Oeq_refl d1) (f:=f) (g:=g)); unfold Oeq;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
  end.

Ltac simplmu_arg tacsidecond :=
   Usimpl || simplmu_aux || simpl_mu_rewrite ltac:tacsidecond.
Ltac simplmu := simplmu_arg idtac.
Ltac rsimplmu := (repeat progress (simplmu;simpl)).
```

16 IterFlip.v: An example of probabilistic termination

```
Require Export Prog.
```

16.1 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 $Fiter : (Z \to distr\ Z)$ -m> $(Z \to distr\ Z)$
:= mon (fun $f\ (x{:}Z) \Rightarrow Mif\ Flip\ (f\ (Zsucc\ x))\ (Munit\ x)$).

Lemma $Fiter\_simpl : \forall\ f\ x,\ Fiter\ f\ x = Mif\ Flip\ (f\ (Zsucc\ x))\ (Munit\ x)$.

Definition $iterflip : Z \to distr\ Z := Mfix\ Fiter$.

16.2 Main result

Probability for *iter* to terminate is 1

16.2.1 Auxiliary function *p*

Definition $p\_n = 1 - \frac{1}{2}\ \hat{}\ n$

Fixpoint $p\_\ (n : nat) : U$:= match n with $O \Rightarrow 0 \mid (S\ n) \Rightarrow \frac{1}{2} \times p\_\ n + \frac{1}{2}$ end.

Lemma $p\_incr : \forall\ n,\ p\_\ n \le p\_\ (S\ n)$.

Hint Resolve p_incr.

Definition $p : nat$ -m> $U := fnatO\_intro\ p\_\ p\_incr$.

Lemma $pS\_simpl : \forall\ n,\ p\ (S\ n) = \frac{1}{2} \times p\ n + \frac{1}{2}$.

Lemma $p\_eq : \forall\ n{:}nat,\ p\ n \equiv [1\text{-}]([1/2]\hat{}n)$.
Hint Resolve p_eq.

Lemma $p\_le : \forall\ n{:}nat,\ [1\text{-}]([1/]1{+}n) \le p\ n$.

Hint Resolve p_le.

Lemma $lim\_p\_one : 1 \le lub\ p$.

Hint Resolve lim_p_one.

16.2.2 Proof of probabilistic termination

Definition $q1\ (z1\ z2{:}Z) := 1$.

Lemma $iterflip\_term : okfun$ (fun $k \Rightarrow 1$) $iterflip\ q1$.

17 Choice.v: An example of probabilistic choice

Require Export *Prog*.

17.1 Definition of a probabilistic choice

We interpret the probabilistic program *p* which executes two probabilistic programs *p1* and *p2* 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 $y \Rightarrow Munit\ (choice\ x\ y)$)).

17.2 Main result

We estimate the probability for *p* to satisfy *Q* given estimations for both *p1* and *p2*.

17.2.1 Assumptions

We need extra properties on *p1*, *p2* and *choice*.

- *p1* and *p2* 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*$\neg x \neg y$) will also satisfy *Q*

```
Hypothesis p1_terminates : (μ p1 (fone A))==1.
Hypothesis p2_terminates : (μ p2 (fone A))==1.
```

```
Variable Q : MF A.
Hypothesis choiceok : ∀ x y, Q x + Q y ≤ Q (choice x y).
```

17.2.2 Proof of estimation:

ok k1 p1 Q and *ok k2 p2 Q* implies *ok* (*k1*(1-*k2*)+*k2*) *p Q*

```
Lemma choicerule : ∀ k1 k2,
  k1 ≤ μ p1 Q → k2 ≤ μ p2 Q → (k1 × ([1-] k2) + k2) ≤ μ p Q.
```

```
End CHOICE.
```

18 RandomList.v : pick uniformely an element in a list

Contributed by David Baelde, 2011

```
Fixpoint choose A (l : list A) : distr A :=
  match l with
    | nil ⇒ distr_null A
    | cons hd tl ⇒ Mchoice ([1/](length l)) (Munit hd) (choose tl)
  end.
```

```
Lemma choose_uniform : ∀ A (d : A) (l : list A) f,
  μ (choose l) f ≡ sigma (fun i ⇒ ([1/](length l)) × f (nth i l d)) (length l).
```

```
Lemma In_nth : ∀ A (x:A) l, In x l → ∃ i, (i < length l)%nat ∧ nth i l x = x.
```

```
Lemma choose_le_Nnth :
  ∀ A (l:list A) x f alpha,
    In x l →
    alpha ≤ f x →
    [1/](length l) × alpha ≤ μ (choose l) f.
```

18.1 List containing elements from 0 to n

```
Fixpoint lrange n := match n with
  | O ⇒ cons O nil
  | S m ⇒ cons (S m) (lrange m)
end.
```

```
Lemma range_len : ∀ n, length (lrange n) = S n.
```

```
Lemma leq_in_range : ∀ n x, (x≤n)%nat → In x (lrange n).
```