Implémentation de Bourbaki en Coq
Théorie des ensembles

José Grimm

Projet Apics
Institut National de Recherche en Informatique et Automatique
Sophia Antipolis Méditerranée

24 Février 2010

Notations

Assemblies are words over letters and $\square$, τ, $\vee$, $\neg$, $=$, $\in$.
Two categories: terms (sets) and relations.
Many abreviations.

$\Longrightarrow$ $\quad \vee\neg$

$\wedge AB \quad \neg\vee\neg A\neg B.$

$\Longleftrightarrow AB \quad \wedge \Longrightarrow AB \Longrightarrow BA$

Infix notations with parentheses as in $(A) \Longleftrightarrow (B)$.

$(\boldsymbol{B}|\boldsymbol{x})\ \boldsymbol{A}$ substitution of $\boldsymbol{x}$ by $\boldsymbol{B}$ in $\boldsymbol{A}$.

$\boldsymbol{R}\{\boldsymbol{x}\}$ implicit substitution

$\tau_x(\boldsymbol{R})$ substitution of $\boldsymbol{x}$ by $\square$ linked to a τ in $\tau\boldsymbol{R}$.

$(\exists \boldsymbol{x})\boldsymbol{R}$ $(\tau_{\boldsymbol{x}}(\boldsymbol{R})|\boldsymbol{x})\boldsymbol{R}$.

$(\forall \boldsymbol{x})\boldsymbol{R}$ not $((\exists \boldsymbol{x})$ not $\boldsymbol{R})$

$x \subset y$ $(\forall z)((z \in x) \Longrightarrow (z \in y))$

$\mathrm{Coll}_{\boldsymbol{x}}\boldsymbol{R}$ $(\exists \boldsymbol{y})(\forall \boldsymbol{x})((\boldsymbol{x} \in \boldsymbol{y}) \Longleftrightarrow \boldsymbol{R})$

Bourbaki's Axioms

A1. $(\forall x)(\forall y)((x \subset y) \text{ and } (y \subset x)) \implies (x = y)$.
A2. $(\forall x)(\forall y)\ \mathrm{Coll}_z(z = x \text{ or } z = y)$.
A3.
$(\forall x)(\forall x')(\forall y)(\forall y')(((x, y) = (x', y')) \implies (x = x' \wedge y = y'))$
A4. $(\forall X)\ \mathrm{Coll}_Y(Y \subset X)$.
A5. There exists an infinite set.

In the 1970 edition, (x, y) is a shorthand for $\{\{x\}, \{x, y\}\}$ and A3 is a theorem.

Bourbaki's Schemes I

S1: If $\boldsymbol{A}$ is a relation in $\mathcal{T}$, the relation $(\boldsymbol{A}$ or $\boldsymbol{A}) \implies \boldsymbol{A}$ is an axiom of $\mathcal{T}$.

S2: If $\boldsymbol{A}$ and $\boldsymbol{B}$ are relations in $\mathcal{T}$, the relation $\boldsymbol{A} \implies (\boldsymbol{A}$ or $\boldsymbol{B})$ is an axiom of $\mathcal{T}$.

S3: If $\boldsymbol{A}$ and $\boldsymbol{B}$ are relations in $\mathcal{T}$, the relation $(\boldsymbol{A}$ or $\boldsymbol{B}) \implies (\boldsymbol{B}$ or $\boldsymbol{A})$ is an axiom of $\mathcal{T}$.

S4: If $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are relations in $\mathcal{T}$, the relation $(\boldsymbol{A} \implies \boldsymbol{B}) \implies ((\boldsymbol{C}$ or $\boldsymbol{A}) \implies (\boldsymbol{C}$ or $\boldsymbol{B}))$ is an axiom of $\mathcal{T}$.

S5: If $\boldsymbol{R}$ is a relation in $\mathcal{T}$, if $\boldsymbol{T}$ is a term in $\mathcal{T}$, and if $\boldsymbol{x}$ a letter, then the relation $(\boldsymbol{T}|\boldsymbol{x})\boldsymbol{R} \implies (\exists \boldsymbol{x})\boldsymbol{R}$ is an axiom.

Bourbaki's Schemes II

S6: Let $\boldsymbol{x}$ be a letter, let $\boldsymbol{T}$ and $\boldsymbol{U}$ be terms in $\mathcal{T}$, and let $\boldsymbol{R}\{\boldsymbol{x}\}$ a relation in $\mathcal{T}$; then the relation
$(\boldsymbol{T} = \boldsymbol{U}) \implies (\boldsymbol{R}\{\boldsymbol{T}\} \iff \boldsymbol{R}\{\boldsymbol{U}\})$ is an axiom.

S7: If $\boldsymbol{R}$ and $\boldsymbol{S}$ are relations in $\mathcal{T}$, and if $\boldsymbol{x}$ is a letter, then the relation $((\forall \boldsymbol{x})(\boldsymbol{R} \iff \boldsymbol{S})) \implies (\tau_{\boldsymbol{x}}(\boldsymbol{R}) = \tau_{\boldsymbol{x}}(\boldsymbol{S}))$ is an axiom.

S8: Let $\boldsymbol{R}$ be a relation, let $\boldsymbol{x}$ and $\boldsymbol{y}$ be distinct letters, and let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be letters distinct from $\boldsymbol{x}$ and $\boldsymbol{y}$ which do not appear in $\boldsymbol{R}$. Then the relation

$$(\forall \boldsymbol{y})(\exists \boldsymbol{X})(\forall \boldsymbol{x})(\boldsymbol{R} \implies (\boldsymbol{x} \in \boldsymbol{X})) \implies (\forall \boldsymbol{Y})\ \mathrm{Coll}_{\boldsymbol{x}}((\exists \boldsymbol{y})((\boldsymbol{y} \in \boldsymbol{Y}) \text{ and } \boldsymbol{R}))$$

is an axiom.

Proofs

An instantiated scheme is an axiom.
An axiom is a true statement.
A proof is a sequence of true statements, axioms or deducible from the previous ones.
Deduction rule is: if A and $A \implies B$ are true then B is true.
A true statement is a theorem (lemma, proposition, corollary).
A theorem with holes is a criterion.

Ex C60: *Let u be a letter, $T\{u\}$ a term in the theory $\mathcal{T}$ (in which E is a set well-ordered by a relation denoted $\leq$). There exists a set U and a mapping f of E onto U such that for all $x \in E$ we have $f(x) = T\{f^{(x)}\}$. Furthermore the set U and the mapping f are uniquely determined by these conditions.*

Example of theorem

```
Theorem transfinite_definition: ∀ r T,
  worder r →
  exists_unique (fun f ⇒
   surjective f &
   source f = substrate r &
   ∀ x, inc x (substrate r) →
        W x f = T (restriction1 f (segment r x))).
```

```
Lemma integer_induction: ∀ s a,
  exists_unique (fun f ⇒ source f = Bnat
  & surjective f
  & W card_zero f = a
  & ∀ n, inc n Bnat → W (succ n) f = s (W n f)).
```

$f(0) = a$ and $f(n+1) = s(f(n))$.

A Proof

```
Lemma cantor_bis: ~(∃ a, ∀ x, is_cardinal x -> inc x a).
Proof. red. ir. nin H. set (s:= union x). set (e:= cardinal s).
  assert  (cardinal_le  (card_pow card_two e) e).
  set (w:= (card_pow card_two e)). assert (is_cardinal w). uf w.
  uf card_pow. fprops. assert (sub w s). uf s. app union_sub.
  app H. cp (cardinal_le8 H1). red. ee. am. uf e. fprops.
  wr (cardinal_le4 H0).
  uf e. wr cardinal_le2. am. assert (is_cardinal e). uf e.
  fprops. cp (cantor H1). red in H2. ee.  elim H3.
  app cardinal_antisymmetry1.
Qed.
```

Simpson's Axioms

```
Parameter Ro : ∀ x : Set, x → Set.
Axiom R_inj : ∀ (x : Set) (a b : x),
  Ro a = Ro b → a = b.

Definition inc (x y : Set) :=
  ∃ a : y, Ro a = x.
Definition sub (a b : Set) :=
  ∀ x : Set, inc x a → inc x b.

Check (@Ro nat).
  Ro (x:=nat) : nat → Set
```

Notation $\mathcal{R}a$ for *Ro a*
Notation $a \in b$ for *inc a b*
Notation $a \subset b$ for *sub a b*

Extensionality

```
Axiom extensionality : ∀ a b : Set,
  sub a b → sub b a → a = b.
Axiom prod_extensionality :  ∀ (x : Type)
  (y : x → Type) (u v : ∀ a : x, y a),
    (∀ a : x, u a = v a) → u = v.

Lemma arrow_extensionality :
  ∀ (x y : Type) (u v : x → y),
    (∀ a : x, u a = v a) → u = v.
```

If $a, b : Set$ then $a = b \iff \forall x, x \in a \iff x \in b$

If $a, b : A \to B$ then $a = b \iff \forall x, a(x) = b(x)$

Emptyness

```
Inductive nonemptyT (t : Type) : Prop :=
  nonemptyT_intro : t → nonemptyT t.
Inductive nonempty (x : Set) : Prop :=
  nonempty_intro : ∀ y : Set,
       inc y x → nonempty x.

Definition empty (x : Set) :=
    ∀ y : Set, ~ inc y x.
Inductive emptyset : Set :=.
```

If $x : Set$, both definitions of nonempty are equivalent.
Notation $\emptyset$ for emptyset.

Axiom of Choice

```
Parameter chooseT : ∀ (t : Type) (p : t → Prop)
  (q:nonemptyT t),  t.

Axiom chooseT_pr : ∀ (t : Type) (p : t → Prop)
  (q : nonemptyT t),
    ex p → p (chooseT p q).
```

Example

```
Check (chooseT (fun n ⇒ lt n 0)(nonemptyT_intro 0)).
chooseT (fun n : nat ⇒ n < 0) (nonemptyT_intro 0)
     : nat
```

Image

```
Parameter IM : ∀ x : Set, (x → Set) → Set.

Axiom IM_exists :
        ∀ (x : Set) (f : x → Set) (y : Set),
        inc y (IM f) → ∃ a : x, f a = y.
Axiom IM_inc :
        ∀ (x : Set) (f : x → Set) (y : Set),
        (∃ a : x, f a = y) → inc y (IM f).
```

If E is a set and $f(x)$ is a set $\forall x \in E$, there exists a unique set F, $y \in F \iff \exists x \in E, y = f(x)$.
(See Scheme S8).

Other axioms

```
Axiom excluded_middle : ∀ P : Prop,
  ~ ~ P → P.
Axiom proof_irrelevance : ∀ (P : Prop) (q p : P),
   p = q.
Axiom iff_eq : ∀ P Q : Prop,
  (P → Q) → (Q → P) → P = Q.

Lemma p_or_not_p : ∀ P : Prop, P ∨ ~ P.
Lemma equal_or_not: ∀  x y:Set, x= y  ∨ x<> y.
Lemma emptyset_dichot : ∀ x,
   (x = emptyset ∨ nonempty x).
```

Tactics

```
Ltac ir := intros.
Ltac tv := trivial.
Ltac am := assumption.
Ltac nin h := induction h.
Ltac rw u := rewrite u.
Ltac rww u := rw u; tv.
Ltac ap h := apply h.
Ltac app u := ap u; tv.
Ltac aw := autorewrite with aw;  tv.
Ltac fprops := auto with fprops.
Ltac set_extens:= app extensionality; unfold sub; ir.
Ltac Ztac :=
  match goal with
  | id1:(inc _ (Zo _ _)) |- _ => nin (Z_all id1)
  |  |- (inc _ (Zo _ _)) => ap Z_inc; au
  | _ => idtac
  end.
```

Inverse of Ro

```
Definition Bo (x y : Set) (hyp : inc x y) :=
  chooseT (fun a : y ⇒ ℛ a = x) (inc_nonempty hyp).

Lemma B_eq : ∀ x y (hyp : inc x y),
   ℛ (Bo hyp) = x.
Lemma B_back : ∀ (x:Set) (y:x)
   (hyp : inc (ℛ y) x), Bo hyp = y.

------
Check (B_back (R_inc (S 3))).
B_back (R_inc 4)
     : Bo (R_inc 4) = 4
```

Definition by cases

```
Variables (T:Type) (P:Prop) (a:P → T)(b: ~ P → T)
Definition by_cases:=
  chooseT (fun x : T ⇒  (∀ p : P, a p = x)
                      & (∀ q : ~ P, b q = x))
    (by_cases_nonempty a b).
```

Uses proof_irrelevance.

```
Lemma by_cases_if : ∀ (p : P),
    by_cases a b = a p.
Lemma by_cases_if_not : ∀ (q : ~ P),
    by_cases a b = b q.

Lemma by_cases_if : ∀ (p : P),
    (∀ p1 p2: P, a p1 = a p2) →
    by_cases a b = a p.
```

Axiom of choice for Sets

Is the equivalent of $\tau_x(P)$.

```
Definition choose (p:Set→ Prop) :=
  chooseT (fun x ⇒ (ex p → p x) & ~ ex p → x = ∅)
  (nonemptyT_intro ∅).
Definition rep (x : Set) :=
   choose (fun y : Set ⇒ inc y x).

Lemma choose_pr : ∀ p, ex p → p (choose p).
Lemma choose_not : ∀ p, ~(ex p) → choose p = ∅.
Lemma nonempty_rep: ∀ x, nonempty x → inc (rep x) x.
```

Fork

```
Definition Yt (A:Type): Prop → (A→ A→ A):=
  fun P x y ⇒
     by_cases (fun _ : P ⇒ x) (fun _ : ~ P ⇒ y).

Definition Yo : Prop → (Set→ Set→ Set) :=
  fun P x y ⇒
     by_cases (fun _ : P ⇒ x) (fun _ : ~ P ⇒ y).

Lemma Y_if_rw : ∀ (P:Prop) (hyp :P) x y,
    Yo P x y = x.
Lemma Y_if_not_rw : ∀ (P:Prop) (hyp : ~P) x y,
    Yo P x y = y.
```

Set of y in x such that P

```
Inductive Zorec (x : Set) (f : x → Prop) : Set :=
   Zorec_c : ∀ a: x, f a → Zorec f.

Definition Zo (x:Set) (p:EP) :=
  let f:= fun a : x ⇒ p (ℛ a) in
   IM (fun (z : Zorec f) ⇒ let (a, _):= z in ℛ a).

Lemma Z_all : ∀ x p y,
    inc y (Zo x p) → (inc y x & p y).
Lemma Z_inc : ∀ x : (p : EP) y,
    inc y x → p y → inc y (Zo x p).
```

Notation $\{y \in x,\ P(y)\}$

Singletons and doubletons

```
Inductive one_point : Set :=
    one_point_intro : one_point.
Inductive two_points : Set :=
  | two_points_a | two_points_b.

Definition singleton (x : Set) :=
  IM (fun p : one_point ⇒ x).
Definition doubleton (x y : Set) :=
  IM (fun t ⇒ two_points_rect (fun _ : two_points ⇒ Set)
                x y t).
Lemma singleton_rw: ∀ x y,
  inc y (singleton x) = (y = x).
Lemma doubleton_rw : ∀ x y z : Set,
  inc z (doubleton x y) = (z = x ∨ z = y).
```

Notation $\{x\}$ and $\{x, y\}$

Complement

```
Definition complement (a b : Set) :=
   Zo a (fun x : Set ⇒ ~ inc x b).

Lemma inc_complement :  ∀ a b x,
  inc x (complement a b) = (inc x a & ~ inc x b).

Lemma double_complement: ∀ a x,
  sub a x → complement x (complement x a) = a.
```

Notation $\complement_A B$ or $A - B$.

Pairs

```
Definition pair_first (x y:Set):= singleton x.
Definition pair_second (x y:Set):=
   doubleton ∅ (singleton y).

Definition bpair (x y : Set) :=
  doubleton (pair_first x y) (pair_second x y).
Notation J := bpair.

Definition pr1 (u : Set) :=
  choose (fun x ⇒ ∃ y, u = J x y).
Definition pr2 (u : Set) :=
  choose (fun y ⇒ ∃ x, u = J x y).
Notation P := pr1.
Notation Q := pr2.
```

Notations $z = (x, y)$, $x = \mathrm{pr}_1 z$, $y = \mathrm{pr}_2 z$.

Properties of pairs

```
Lemma pair_is_pair : ∀ x y, is_pair (J x y).
Lemma pr1_pair : ∀ x y, P (J x y) = x.
Lemma pr2_pair : ∀ x y, Q (J x y) = y.
Lemma pair_extensionality : ∀ a b,
   is_pair a → is_pair b →
   P a = P b → Q a = Q b → a = b.
Lemma pr1_injective: ∀ a b c d,
   J a b = J c d → a = c.
Lemma pr2_injective: ∀ a b c d,
   J a b = J c d → b = d.
```

Powerset

```
Definition powerset (x : Set) :=
  IM (fun p : x → two_points ⇒
    Zo x (fun y ⇒ ∀ hyp : inc y x,
       p (Bo hyp) = two_points_a)).

Lemma powerset_inc_rw : ∀ x y,
   inc x (powerset y) = sub x y.
```

Notation $\mathfrak{P}(X)$. Is the set of all $f^{-1}(\{0\})$ for $f : X \to \{0, 1\}$.

Union

```
Record Uintegral (x : Set) : Set :=
  Union_param : x; Uelt : ℛ Union_param.

Definition union (x : Set) :=
  IM (fun i : Uintegral x ⇒ ℛ (Uelt i)).
Definition union2 x y := union (doubleton x y).
Definition tack_on x y := union2 x (singleton y).
Lemma union_inc : ∀ x y a,
  inc x y → inc y a → inc x (union a).
Lemma union_exists : ∀ x a,
  inc x (union a) → ∃ y,  inc x y & inc y a.
Lemma inc_union2_rw : ∀ a b x,
  inc x (union2 a b) = (inc x a ∨ inc x b).
```

Notations $\bigcup X$, $A \cup B$.

Intersection

```
Definition intersection (x : Set) :=
  Zo (rep x) (fun y ⇒ ∀ z, inc z x → inc y z).
Definition intersection2 x y :=
  intersection (doubleton x y).
Lemma intersection_inc :  ∀ x a, nonempty x →
   (∀ y, inc y x → inc a y)
      → inc a (intersection x).
Lemma intersection_forall : ∀ x a y,
  inc a (intersection x) → inc y x → inc a y.
Lemma intersection2_both: ∀ x y a,
  inc a (intersection2 x y) → (inc a x & inc a y).
Lemma intersection2_inc : ∀ x y a,
   inc a x → inc a y → inc a (intersection2 x y).
```

Notations $\bigcap X$, $A \cap B$.

Cartesian Product

```
Definition product (A B : Set) :=
  union (fun_image A (fun x ⇒
                 (fun_image B (fun y ⇒ J x y)))).

Lemma inc_product : ∀ x y z,
  inc x (product y z) =
     (is_pair x & inc (P x) y & inc (Q x) z).
```

Notation $A \times B$.

Functional Graphs

```
Definition is_graph r := ∀ y, inc y r → is_pair y.
Definition fgraph f :=
  is_graph f &
  (∀ x y, inc x f → inc y f → P x = P y → x = y).

Definition domain f := fun_image f P.
Definition range f := fun_image f Q.
Definition V x f := choose (fun y ⇒ inc (J x y) f).
Definition L (x : Set) (p : E) :=
  fun_image x (fun y ⇒ J y (p y)).

Lemma pr2_V : ∀ f x,
  fgraph f → inc x f → Q x = V (P x) f.
Lemma create_V_rewrite :  ∀ x p y,
   inc y x → V y (L x p) = p y.
```

Correspondences

```
Record correspondenceC :Type :=
  corresp{ source:Set; target:Set; graph :Set }.
Definition corr_axiom s t g:=
  is_graph g & sub (domain g) s & sub (range g) t.
Definition is_correspondence f :=
  corr_axiom (source f) (target f) (graph f).
Definition gacreate (a b:Set) (f:a → b) :=
  IM (fun y:a ⇒ J (ℛ y) (ℛ (f y))).
Definition acreate (a b:Set) (f:a → b) :=
  corresp a b (gacreate f).
```

This is not a set; but the associated triple is a set.
J (graph x) (J (source x) (target x)).

Functions

```
Definition is_function f :=
  is_correspondence f & fgraph (graph f)
  & source f = domain (graph f).
Definition W x f := V x (graph f).
Lemma W_pr: ∀ f x y, is_function f →
  inc (J x y) (graph f) → y = W x f.
Lemma inc_W_target: ∀ f x, is_function f →
  inc x (source f) → inc (W x f) (target f).
Lemma function_acreate : ∀ (A B:Set) (f:A → B),
  is_function(acreate f).
Lemma W_acreate : ∀ (A B:Set) (f:A → B) (x:A),
  W (ℛ x) (acreate f) = ℛ (f x).
```

Functions and arrows

```
Definition bcreate1 f (H:is_function f) :=
  fun x:source f ⇒ Bo (inc_W_target H (R_inc x)).
Definition bcreate f A B
  (H:is_function f)(Ha:source f =A)(Hb:target f =B):=
  fun x:A ⇒ Bo (W_mapping Ha Hb H (R_inc x)).

Lemma bcreate_inv1: ∀ f (H:is_function f),
  acreate (bcreate1 H) = f.
Lemma bcreate_inv2: ∀ f A B
  (H:is_function f) (Ha:source f=A)(Hb:target f=B),
  acreate (bcreate H Ha Hb) = f.
Lemma bcreate_inv3: ∀ (A B:Set) (f:A→ B),
  (bcreate (function_acreate f) (source_acreate f)
    (target_acreate f)) = f.
```

Defining functions

```
Definition BL f a b := corresp a b (L a f).
Definition transf_axioms f a b :=
  ∀ c, inc c a → inc (f c) b.

Lemma af_function: ∀ f a b,
  transf_axioms f a b → is_function (BL f a b).
Lemma W_af_function: ∀ f a b c,
  transf_axioms f a b → inc c a →
  W c (BL f a b) = f c.
```

Composition

```
Definition compose r' r :=
  corresp (source r)(target r')
    (compose_graph (graph r')(graph r)).
Definition composable g f :=
  is_function g & is_function f & source g = target f

Theorem is_function_compose: ∀ g f,
  composable g f
  → is_function (compose g f).

Lemma W_compose: ∀ g f x,  composable g f →
  inc x (source f) →
  W x (compose g f) = W (W x f) g.
```

Bijections

```
Definition injective f:=
  is_function f &
  (∀ x y, inc x (source f) → inc y (source f) →
    W x f = W y f → x = y).

Definition surjective f :=
  is_function f & image_of_fun f = target f.

Definition bijective f :=
  injective f & surjective f.

Lemma bij_left_inverse: ∀ f,  bijective f →
 compose (inverse_fun f) f = identity_fun (source f).
Theorem exists_left_inv_from_inj_alt: ∀ f,
  injective f → nonempty (source f) →
  exists r, is_left_inverse r f.
```

Equipotency

```
Definition equipotent x y :=
  ∃ z, bijective z & source z = x & target z = y.

Lemma equipotent_reflexive: ∀ x, equipotent x x.
Lemma equipotent_symmetric: ∀ a b,
  equipotent a b → equipotent b a.
Lemma equipotent_transitive: ∀ a b c,
  equipotent a b → equipotent b c
 → equipotent a c.
```

Union

```
Record Uintegral (In :Set)(f :In→ Set) : Set :=
  {UI_z : In; UI_elt : f UI_z}.
Definition unionf (In:Set)(f : In→ Set) :=
   IM (fun i : Uintegral f ⇒ ℛ (UI_elt i)).
Definition unionf (x:Set)(f: Set→ Set) :=
   unionf (fun a:x ⇒ f (ℛ a)).
Definition unionb (g:Set) :=
   unionf (domain g)(fun a=> V a g).

Lemma unionb_rw: ∀ x f,
  inc x (unionb f) =
    ∃ y, inc y (domain f) & inc x (V y f).
```

$x \in \bigcup_{x \in I} X_i \iff \exists i \in I, x \in X_i$

Intersection

```
Definition intersectiont (In:Set)(f : In→ Set):=
  by_cases(fun H:nonemptyT In ⇒
      Zo (f (chooseT_any H))
         (fun y ⇒ ∀ z : In, inc y (f z)))
  (fun _:~ nonemptyT In ⇒ ∅).
Definition intersectionf (x:Set)(f: Set→ Set):=
  intersectiont(fun a:x ⇒ f (Ro a)).
Definition intersectionb (g:Set) :=
  intersectionf (domain g) (fun a ⇒ V a g).

Lemma intersectionb_inc : ∀ g x, nonempty g →
  (∀ i, inc i (domain g) → inc x (V i g))
  → inc x (intersectionb g).
Lemma intersectionb_forall : ∀ g x i,
  inc x (intersectionb g) → inc i (domain g)
  → inc x (V i g).
```

$x \in \bigcap_{x \in I} X_i \iff \forall i \in I, x \in X_i$

Associativity

$$\bigcup_{\iota\in I} X_\iota = \bigcup_{\lambda\in L}\,(\bigcup_{\iota\in J_\lambda} X_\iota), \qquad \bigcap_{\iota\in I} X_\iota = \bigcap_{\lambda\in L}\,(\bigcap_{\iota\in J_\lambda} X_\iota).$$

```
Theorem union_assoc: ∀ sf sg f g,
  sf = unionf sg g →
  unionf sf f = unionf sg (fun l ⇒ unionf (g l) f).
Theorem intersection_assoc: ∀ sf sg f g,
  nonempty sg →
  (∀ i, inc i sg → nonempty (g i)) →
  sf = unionf sg g →
  intersectionf sf f =
  intersectionf sg (fun l ⇒ intersectionf (g l) f).
```

Products

```
Definition productt (In:Set) (f:In→ Set):=
  Zo (powerset (product In (uniont f)))
  (fun z ⇒ fgraph z & domain z = In
   & ∀ i:In, inc (V (ℛ i) z) (f i)).

Definition productb g:=
   productt (fun x:domain g ⇒ V (Ro x) g).
Definition productf sf f:=  productb (L sf f).
Lemma productb_pr: ∀ f x, fgraph f →
 inc x (productb f) =
 (fgraph x &  domain x = domain f &
  ∀ i, inc i (domain x) → inc (V i x) (V i f)).
```

$x \in \prod_{i \in I} X_i \iff \forall i \in I, x_i \in X_i$

Distributivity 1

$$\bigcup_{\lambda\in L}\Big(\bigcap_{\iota\in J_\lambda} X_{\lambda,\iota}\Big) = \bigcap_{f\in\prod J_\lambda}\Big(\bigcup_{\lambda\in L} X_{\lambda,f(\lambda)}\Big),$$

```
Theorem distrib_union_inter: ∀ f,
  fgraph f → nonempty (domain f) →
  (∀ l, inc l (domain f) → fgraph (V l f)) →
  (∀ l, inc l (domain f) → nonempty(domain (V l f)))
  → unionf (domain f) (fun l => intersectionb (V l f)) =
  intersectionf
     (productf (domain f) (fun l⇒ (domain (V l f))))
     (fun g ⇒ (unionf (domain f)
                      (fun l ⇒ V (V l g) (V l f)))).
```

Distributivity 2

$$\prod_{\lambda\in L}(\bigcup_{\iota\in J_\lambda} X_{\lambda,\iota}) = \bigcup_{f\in\prod J_\lambda}(\prod_{\lambda\in L} X_{\lambda,f(\lambda)})$$

```
Theorem distrib_prod_union: ∀ f,
  fgraph f →
  (∀ l, inc l (domain f) → fgraph (V l f)) →
  productf (domain f) (fun l ⇒ unionb (V l f)) =
  unionf
    (productf (domain f) (fun l ⇒ (domain (V l f))))
    (fun g ⇒ (productf (domain f)
                       (fun l ⇒ V (V l g) (V l f)))).
```

Equivalence and order relations

```
Definition substrate r :=  union2 (domain r) (range r).
Definition is_reflexive r :=
  is_graph r & (∀ y, inc y (substrate r)→ related r y y).
Definition is_symmetric r :=
  is_graph r  & (∀ x y, related r x y → related r y x).
Definition is_antisymmetric (r:Set) := is_graph r
   & ∀ x y, related r x y → related r y x → x = y.
Definition is_transitive r := is_graph r &
  ∀ x y z, related r x y → related r y z
    → related r x z.
Definition is_equivalence r :=
  is_graph r & is_reflexive r & is_transitive r
     & is_symmetric r.
Definition order r :=
  is_graph r & is_reflexive r & is_transitive r
     & is_antisymmetric r.
```

Quotient set

```
Definition class (r x:Set) :=
    fun_image (Zo r (fun z ⇒ P z = x)) Q.
Definition quotient r :=
    fun_image (substrate r) (class r).

Lemma is_class_rw : ∀ r x, is_equivalence r →
  is_class r x = (nonempty x & sub x (substrate r) &
    (∀ y z, inc y x → (inc z x = related r y z))).
Lemma inc_quotient : ∀ r x, is_equivalence r →
  inc x (quotient r) = is_class r x.
Lemma partition_from_equivalence: ∀ r,
  is_equivalence r →
  partition(quotient r)(substrate r).
```

More properties

- Relations compatible with an equivalence relation
- Saturated subsets
- Mappings compatible with equivalence relations
- Inverse image of an equivalence relation;
- Induced equivalence relation
- Quotients of equivalence relations
- Product of two equivalence relations

Ordered Sets

```
Definition gle (r x y:Set) := related r x y.
Definition induced_order (r a:Set):=
  intersection2 r (coarse a).
Definition order_isomorphism f r r':=
  (order r) & (order r') &
  (bijective f) &
  (substrate r = source f) &
  (substrate r' = target f) &
  (∀, inc x (source f) → inc y (source f) →
    gle r x y = gle r' (W x f) (W y f)).
```

Comparings graphs & functions

```
Definition product_order_r (f g:Set) :=
  fun x x' =>
    inc x (productb f) & inc x' (productb f) &
    ∀ i, inc i (domain f) →
          gle (V i g) (V i x)(V i x').
Definition function_order_r x y r f g :=
  is_function f & is_function g
  & source f = x & target f = y
  & source g = x & target g = y &
  ∀ i, inc i x → gle r (W i f) (W i g).
Definition function_order x y r :=
  graph_on (fun u v => function_order_r x y r
          (sof_value x y u) (sof_value x y v))
  (set_of_functions x y).
```

More Properties

- Maximal and minimal elements
- Greatest element and least element
- Upper and lower bounds
- Least upper bound and greatest lower bound
- Directed sets
- Lattices
- Totally ordered sets
- Intervals

Distributivity

$$\sup_{(\lambda,\mu)\in L\times M} x_{\lambda\mu} = \sup_{\mu\in M}\left(\sup_{\lambda\in L} x_{\lambda\mu}\right)$$

```
Lemma sup_distributive3: ∀ r f x y,
  let PF := fun f x m ⇒ restr f (product x (singleton m)) in
  let temp := (L y (fun m ⇒ sup_graph r (PF f x m))) in
  order r → fgraph f → sub (range f) (substrate r) →
    domain f = product x y →
    (∀ m, inc m y → has_sup_graph r (PF f x m)) →
    ( (has_sup_graph r f = has_sup_graph r temp)
     &
      (has_sup_graph r f →
        sup_graph r f = sup_graph r temp)).
```

Zermelo

Worder : every nonempty set has a least element
Segment x $= \{y, y < x\}$

```
Lemma Zermelo_aux: ∀ E s p,
  sub s (powerset E)
  → (∀ x, inc x s → inc (p x) (complement E x))
  → ∃ r, worder r &
      sub (substrate r) E &
      (∀ x, inc x (substrate r) →
           inc (segment r x) s)
        & p (segment r x) = x).
      & ~ inc (substrate r) s.
```

If S is the set of all strict subsets of E, then substrate $r = E$.

Zorn's Lemma

```
Theorem Zermelo: ∀ E, ∃ r,
   worder r & substrate r = E.

Definition inductive_set r :=
  ∀ X, sub X (substrate r) →
      total_order (induced_order r X)
  → ∀ x, upper_bound r X x.

Theorem Zorn_lemma: ∀ r, order r
  → inductive_set r
  → ∃ a, maximal_element r a.
```

Let S be the set of subsets X of E such that there is a $p(X)$, an upper bound of X not in X.

Well-Order isomorphisms

```
Theorem isomorphism_worder: ∀ r r',
  worder r → worder r' →
  let iso:= (fun u v f ⇒
                is_segment v (range (graph f))
              & order_morphism f u v) in
  exists_unique (fun f ⇒ iso r r' f)
  ∨
  exists_unique (fun f ⇒ iso r' r f).
```

Cardinals

```
Definition cardinal x :=
   choose (fun z ⇒ equipotent x z).
Definition card_zero :=
   cardinal ∅.
Definition card_one :=
   cardinal (singleton ∅).
Definition card_two :=
   cardinal (doubleton (singleton ∅) ∅).
Definition card_three := succ card_two.
Definition card_four := succ card_three.
```

Orderings on cardinal

```
  Definition cardinal_le x y :=
  is_cardinal x & is_cardinal y &
  equipotent_to_subset x y.
Theorem wordering_cardinal_le:
   worder_r cardinal_le.
```

Cantor Bernstein : Assume $f : E \to F$ and $g : F \to E$ injective; there is a bijection $k : E \to F$.

Define $k(x) = f(x)$ if $x \in M$, and $k(x) = y$ where $x = g(y)$ and $y \in F - f(M)$.

Assumes $E - M = g(F - f(M))$.

Let M be the union of all X with $h(X) \subset X$.
$h(X) := E - g(F - f(X))$.

Operations on cardinals

```
D:= domain
Definition cardinal_sum x := cardinal (disjoint_union x).
Definition cardinal_prod x := cardinal (productb x).
Theorem cardinal_sum_commutative: ∀ X f,
  fgraph X →  target f = D X → bijective f →
  cardinal_sum X = cardinal_sum (gcompose X (graph f)).
Theorem cardinal_sum_assoc: ∀ f g,
  fgraph f → partition_fam g (D f) →
  cardinal_sum f  = cardinal_sum (L (D g) (fun l ⇒
    cardinal_sum (restr f (V l g)))).
Theorem cardinal_distrib_prod_sum: ∀ f,
  fgraph f →
  (∀ l, inc l (D f) → fgraph (V l f)) →
  cardinal_prod (L (D f)(fun l ⇒ cardinal_sum (V l f)))=
  cardinal_sum (L (productf (D f) (fun l ⇒ (D (V l f))))
    (fun g ⇒ (cardinal_prod (L (D f)
                  (fun l ⇒ V (V l g) (V l f)))))).
```

Operations and Order

```
Theorem sum_increasing: ∀ f g,
  fgraph f → fgraph g → domain f = domain g →
  (∀ x, inc x (D f)→ cardinal_le (V x f) (V x g))
  → cardinal_le (cardinal_sum f) (cardinal_sum g).

Definition card_plus a b :=
  cardinal_sum (Lvariantc a b).

Lemma sum_increasing2: ∀ a b a' b',
  cardinal_le a a' → cardinal_le b b' →
  cardinal_le (card_plus a b)  (card_plus a' b').
Lemma sum_increasing3: ∀ a b,
  is_cardinal a → is_cardinal b →
  cardinal_le a (card_plus a b).
Theorem cantor: ∀ a, is_cardinal a →
  cardinal_lt a (card_pow card_two a).
```

Integers

```
Definition succ x := card_plus x card_one.
Definition is_finite_c x :=
   is_cardinal x & x <> succ x.
Theorem is_finite_succ: ∀ x,  is_cardinal x →
   (is_finite_c x) = (is_finite_c (succ x)).
Theorem le_int_is_int: ∀ a b, is_finite_c b →
   cardinal_le a b → is_finite_c a.

Definition prec n :=
   choose (fun m ⇒ is_finite_c m & n = succ m).
Lemma prec_pr: ∀ n, is_finite_c n
  → n <> card_zero →
  (is_finite_c (prec n) & n = succ (prec n)).
```

nat

```
Fixpoint nat_to_B (n:nat) :=
  match n with 0 ⇒ card_zero
             | S m ⇒ succ (nat_to_B m) end.
Lemma nat_B_inj: ∀ a b,
  nat_to_B a = nat_to_B b → a = b.
Lemma nat_to_B_surjective: ∀ n, inc n Bnat →
  ∃ m, nat_to_B m = n.

Lemma nat_B_S: ∀ n,
  nat_to_B (S n) = succ (nat_to_B n).
Lemma nat_B_plus: ∀ a b,
  nat_to_B (a+b) =
  card_plus (nat_to_B a) (nat_to_B b).
```

Subtraction & Division

```
Lemma card_sub_pr0: ∀ a b, inc a Bnat → inc b Bnat →
  cardinal_le b a →
  (inc (card_sub a b) Bnat  &
   card_plus b (card_sub a b) = a).

Definition division_prop a b q r :=
  a = card_plus (card_mult b q) r & cardinal_lt r b.

Lemma Bnat_division: ∀ a b, inc a Bnat → inc b Bnat →
  b <> card_zero →
  ( inc (card_rem a b) Bnat&
    (inc (card_quo a b) Bnat)&
    (division_prop a b (card_quo a b)(card_rem a b))).
```

The Binomial coefficient

```
Fixpoint binom (n p:nat) struct n : nat :=
  match n, p with
    | 0, 0 => 1
    | 0, S m => 0
    | S q, 0 => 1
    | S q, S m =>  (binom q (S m)) + (binom q m)
end.

Lemma binom_pr1: ∀ n p,
  p<=n → binom n p =
       Nquo (factorial n)
            ( (factorial p) * (factorial (n -p))).
```

Properties

```
Theorem binomial7: ∀ n p E,
  cardinal E = nat_to_B n →
  cardinal (subsets_with_p_elements (nat_to_B p) E)
  = nat_to_B(binom n p).

Lemma sum_to_increasing6: ∀ n p,
  cardinal (set_of_functions_sum_le_int
                (nat_to_B p) (nat_to_B n))
  = nat_to_B(binom (n+p+1) (p+1)).
```

$\text{Card}\ \{X, X \subset E,\ \text{Card}\ X = p\} = \binom{n}{p}$

$\text{Card}\ \{(x_i)_{0 \leq i \leq p}, \sum x_i \leq n\} = \binom{n+p+1}{p+1}$

Infinite Sets

```
Lemma product2_infinite: ∀ a b, cardinal_le b a →
  is_infinite_c a → b <> card_zero
  → card_mult a b = a.
Lemma sum2_infinite: ∀ a b, cardinal_le b a →
  is_infinite_c a  → card_plus a b = a.
Theorem countable_union: ∀ f,  fgraph f →
  is_countable_set (domain f) →
  (∀ i, inc i (domain f)→ is_countable_set (V i f))
  → is_countable_set (unionb f).
```

$A + B = A \cdot B = \max(A, B)$ if non-zero, one cardinal infinite.

The end

```
Lemma cantor_bis: ~ (exists a, forall x, is_cardinal x -> inc x a
Proof. red. ir. nin H. set (s:= union x). set (e:= cardinal s).
  assert  (cardinal_le  (card_pow card_two e) e).
  set (w:= (card_pow card_two e)). assert (is_cardinal w). uf w.
  uf card_pow. fprops. assert (sub w s). uf s. app union_sub.
  app H. cp (cardinal_le8 H1). red. ee. am. uf e. fprops.
  wr (cardinal_le4 H0).
  uf e. wr cardinal_le2. am. assert (is_cardinal e). uf e.
  fprops. cp (cantor H1). red in H2. ee.  elim H3.
  app cardinal_antisymmetry1.
Qed.
```