INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Implementation of Bourbaki's Elements of Mathematics in Coq: Part Two Ordered Sets, Cardinals, Integers

José Grimm

N° 7150 — version 3

initial version December 2009 — revised version December 2010

Programs, Verification and Proofs

ISSN 0249-6399 ISRN INRIA/RR--7150--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

Implementation of Bourbaki's Elements of Mathematics in Coq: Part Two Ordered Sets, Cardinals, Integers

José Grimm*

Theme : Programs, Verification and Proofs

Équipe-Projet Apics

Rapport de recherche n° 7150 — version 3 — initial version December 2009 — revised version December 2010 — 438 pages

Abstract: We believe that it is possible to put the whole work of Bourbaki into a computer. One of the objectives of the Gaia project concerns homological algebra (theory as well as algorithms); in a first step we want to implement all nine chapters of the book Algebra. But this requires a theory of sets (with axiom of choice, etc.) more powerful than what is provided by Ensembles; we have chosen the work of Carlos Simpson as basis. This reports lists and comments all definitions and theorems of the Chapter "Ordered Sets, Cardinals, Integers".

Version 3 is based on the Coq ssreflect library. It implements some properties on ordinal numbers. The code (including some exercises) is available on the Web, under http://www-sop.inria.fr/apics/gaia.

Key-words: Gaia, Coq, Bourbaki, orders, cardinals, ordinals, integers

Work done in collaboration with Alban Quadrat

* Email: Jose.Grimm@sophia.inria.fr

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Implémentation de la théorie des ensembles de Bourbaki dans Coq partie 2
Ensembles Ordonnés, cardinaux, nombres entiers

Résumé : Nous pensons qu'il est possible de mettre dans un ordinateur l'ensemble de l'œuvre de Bourbaki. L'un des objectifs du projet Gaia concerne l'algèbre homologique (théeorie et algorithmes); dans une première étape nous voulons implémenter les neuf chapitres du livre Algèbre. Au préalable, il faut implémenter la théorie des ensembles. Nous utilisons l'Assistant de Preuve Coq; les choix fondamentaux et axiomes sont ceux proposés par Carlos Simpson. Ce rapport liste et commente toutes les définitions et théorèmes du Chapitre "Ensembles ordonnés, cardinaux, nombres entiers". Une petite partie des exercises a été résolue. Le code est disponible sur le site Web http://www-sop.inria.fr/apics/gaia.

Mots-clés : Gaia, Coq, Bourbaki, ordre, cardinaux, ordinaux, entiers

Chapter 1

Introduction

1.1 Objectives

Our objective (it will be called the *Bourbaki Project* in what follows) is to show that it is possible to implement the work of N. Bourbaki, "Éléments de Mathématiques"[4], into a computer, and we have chosen the Coq Proof Assistant, see [8, 2]. All references are given to the English version "Elements of Mathematics"[3], which is a translation of the French version (the only major difference is that Bourbaki uses an axiom for the ordered pair in the English version and a theorem in the French one). We start with the first book: theory of sets. It is divided into four chapters, the first one describes formal mathematics (logical connectors, quantifiers, axioms, theorems). Chapters II describes sets, unions, intersections, functions, products, equivalences; Chapter III defines orders, integers, cardinals, limits. The last chapter describes structures. The first part of this report[5] describes Chapter I and Chapter II, we consider here Chapter III.

1.2 Content of this document

This document describes the code found in the files *set5.v*, *set6.v*, *set7.v*, *set8.v*, *set9.v*, and *set10.v*, corresponding to sections 1 to 6 of Chapter III. The first section describes order relations and associated properties (like upper bounds, greatest elements, increasing functions, order isomorphisms). The second section studies well-ordered sets, and introduces the notion of transfinite induction. We show Zermelo's theorem (which is equivalent to the axiom of choice). Section 3 defines cardinals, addition, multiplication and order on cardinals (a cardinal is a representative of a class of equipotent sets; this class is not a set, and the axiom of choice is required). Section 4 defines natural integers as cardinals x such that $x \neq x+1$. It introduces induction on natural integers, so that a natural integer is any cardinal obtained by applying a "finite" number of times $x \mapsto x+1$ to the empty set. More formally, if E is a set containing zero and stable by $x \mapsto x+1$, it contains all natural integers. If E is any set with cardinal x, the set $\mathrm{E} \cup \{\mathrm{E}\}$ has cardinal $x+1$. As a consequence, one can define a mapping $n \mapsto \mathrm{E}_n$ that associates to each natural number n a set E_n of cardinal n (by transfinite induction, one can do this for any cardinal n). This set E_n is called an ordinal in [7] and pseudo-ordinal here (For Bourbaki, an ordinal is a representative of well-ordered sets). It allows one to define finite cardinals without the use of the axiom of choice. There is no set containing all cardinals (thus no set containing all ordinals) but given a cardinal (or ordinal) a, there is a set containing all cardinals (or ordinals) less than a, and it is well-ordered. Ev-

ery finite ordinal is a natural integer; infinite ordinals possess strange properties (addition and multiplication are non-commutative) studied in the Exercises. Section 5 studies some properties of integers (for instance division, expansion to base b) and computes the number of elements of various sets (for instance the number of subsets of p elements of a set of n elements, the number of permutations, etc). Section 6 studies infinite sets. If there exists an infinite set, then there exists a set **N** containg all natural integers. An axiom is required in Bourbaki; in Coq, there is an infinite set, namely *nat*, and it is canonically isomorphic to the set of natural integers. We use this isomorphism in section 5 (for instance, the factorial function and binomial coefficient are defined by induction on the Coq type *nat*, then shown to satisfy the Bourbaki definitions). There are few infinite cardinals (i.e., for any cardinal x there is a cardinal y such that $y > x$, for instance 2^x, but one could add an axiom saying that $y > x$ implies $y \geq 2^x$), so that one gets result like: the number of permutations of E is the number of mapping $\mathbf{N} \mapsto$ E, it is also the number of orderings on E (see Exercices).

Section 6 defines direct limits and inverse limits. It is not yet implemented. There are many exercises, only a few of them are solved.

The reader is invited to read the introduction of the first part. It explains some implementation details (for instance, what is a set? what formulation of the axiom of choice is used?).

1.3 Terminology

Chapter III is much less formal that Chapter II. Let's for instance quote Definition 9 [3, p. 146]: "Two elements of a preordered set E are said comparable if the relation "$x \leq y$ or $y \leq x$" is true. A set E is said to be totally ordered if is is ordered and if any two elements of E are comparable. The ordering in E is then said to be a total ordering and the corresponding order relation a total order relation."

One has to understand this as follows. A preordered set is a pair (E, G) which satisfies the preorder condition. The notation $x \leq y$ stands for $(x, y) \in$ G. An ordered set is a preordered set where additional conditions are required for G. The ordering is G, the corresponding order relation is "$(x, y) \in$ G". By definition, E is uniquely determined by G. Instead of saying that E is totally ordered, one can say: G is a total ordering if it is an ordering and for every x and y in the substrate of G one has "$(x, y) \in$ G or $(y, x) \in$ G". This non ambiguous, and will be our definition. The following sentence is ambiguous "The set **R** of real numbers is totally ordered", since the order is not specified. Note that a sentence like "$(\mathbf{R}, \leq)$ is totally ordered" is ambiguous since $\leq$ can denote any ordering. One must say "The set **R** of real numbers is totally ordered by the usual order on real numbers." We shall use the notation $x \leq_{\mathbf{R}} y$; an alternative would be $x \leq y \pmod{\mathbf{R}}$ as in [7].

Bourbaki says: "a well-ordered set is totally ordered". This is a short-hand for: for every (E, Γ), if Γ is a well-ordering on E, then Γ is a total ordering on E. It is impossible to say "for every equivalence relation R we have..." since relations cannot be quantified; there is only one theorem in E.II.6, the chapter on equivalence relations. It is of the form: a correspondence Γ between X and X is an equivalence if and only if... This might explain why Bourbaki defines an order as a correspondence, rather than a graph. As a consequence, there are few criteria (C59 to C63 define normal and transfinite induction).

1.4 Tactics

We give here the list of tactics that are defined in the files associated to this document (version 2). They have been adapted to the ssreflect version of the software.

This is now the tactic that exploits properties of order relations.

```
Ltac order_tac:=
  match goal with
    | H1: gle ?r ?x _ |- inc ?x (substrate ?r)
      => exact (inc_arg1_substrate H1)
    | H1: glt ?r ?x _ |- inc ?x (substrate ?r)
      => nin H1; order_tac
    | H1:gle ?r _ ?x |- inc ?x (substrate ?r)
      => exact (inc_arg2_substrate H1)
    | H1:glt ?r _ ?x |- inc ?x (substrate ?r)
      => nin H1; order_tac
    | H: order ?r, H1: inc ?u (substrate ?r) |- related ?r ?u ?u
      =>  wrr order_reflexivity
    | H: order ?r |- inc (J ?u ?u) ?r
       => change (related r u u);  wrr order_reflexivity
    | H: order ?r |- gle ?r ?u ?u
       =>  wrr order_reflexivity
    | H1: gle ?r ?x ?y, H2: gle ?r ?y ?x, H:order ?r |-
        ?x = ?y => exact (order_antisymmetry H H1 H2)
    | H:order ?r, H1:related ?r ?x ?y, H2: related ?r ?y ?x |- ?x = ?y
      => app (order_antisymmetry H H1 H2)
    | H:order ?r, H1:related ?r ?u ?v, H2: related ?r ?v ?w
      |- related ?r ?u ?w
      => app (order_transitivity H H1 H2)
    | H:order ?r, H1:gle ?r ?u ?v, H2: gle ?r ?v ?w
      |- gle ?r ?u ?w
      => app (order_transitivity H H1 H2)
    | H: order ?r, H1: inc (J ?u ?v) ?r, H2: inc (J ?v ?w) ?r |-
      inc (J ?u ?w) ?r
      => change (related r u w); app (order_transitivity H H1 H2)
    |  H1: gle ?r ?x ?y, H2: glt ?r ?y ?x, H: order ?r |- _
      => elim (not_le_gt H H1 H2)
    | H1:glt ?r ?x ?y, H2: glt ?r ?y ?x, H:order ?r |- _
      => destruct H1 as [H1 _] ; elim (not_le_gt H H1 H2)
    | H1:order ?r, H2:glt ?r ?x ?y, H3: gle ?r ?y ?z |- glt ?r ?x ?z.
       => exact (lt_leq_trans H1 H2 H3)
    | H1:order ?r, H2:gle ?r ?x ?y, H3: glt ?r ?y ?z |- glt ?r ?x ?z.
       => exact (leq_lt_trans H1 H2 H3)
    | H1:order ?r, H2:glt ?r ?x ?y, H3: glt ?r ?y ?z |- glt ?r ?x ?z.
       => exact (lt_lt_trans H1 H2 H3)
    | H1:order ?r, H2:gle ?r ?x ?y |- glt ?r ?x ?y
       => split; tv; red; ir
    | H:glt ?r ?x ?y |- gle ?r ?x ?y
       => nin H ; am
    | H1:(inc _ (Zo _ _)), H2 :(inc _ (Zo _ _)) |- _
      => nin (Z_all H1); nin (Z_all H2); clear H1; clear H2; ee
end.
```

This tactic is useful when we want to show $\{x \in A, P(x)\} = \{x \in B, Q(x)\}$. It splits the goal into two subgoals, with assuptions $x \in A$ and $P(x)$ or $x \in B$ and $Q(x)$. The second tactic is

useful to show $\{x \in A, P(x)\} \cap \{x \in B, Q(x)\} = \{x \in C, R(x)\}$. It splits the goal in two. In the first case, we must show $x \in C$ and $R(x)$ in a context where $x \in A$, $P(x)$, $x \in B$ and $Q(x)$. In the second case, we assume $x \in C$ and $R(x)$, and must show the other properties.

```
Ltac zztac:=
  set_extens; Ztac; ee; match goal with H: inc _ (Zo _ _) |- _ => clear H end.


Ltac zztac2:= uf_interval; set_extens ;
  [  match goal with H: inc _  (intersection2 _ _) |- _ =>
       nin (intersection2_both H)  end ;
     match goal with
       H1:(inc _ (Zo _ _)), H2 :(inc _ (Zo _ _)) |- _
      => nin (Z_all H1); nin (Z_all H2); clear H1; clear H2; ee end;Ztac
    |
      Ztac; match goal with H: inc _ (Zo _ _) |- _ => clear H end;
      app intersection2_inc; Ztac; uf glt;ee; try order_tac].
```

This unfolds all definitions of intervals.

```
Ltac uf_interval :=
  uf interval_cc;  uf interval_oo;  uf interval_co; uf interval_oc;
  uf interval_uu;  uf interval_uo;  uf interval_ou;
  uf interval_uc;  uf interval_cu.
```

This tactic uses transitivity and antisymmetry of $\leq_{\mathrm{Card}}$.

```
Ltac co_tac := match goal with
  | Ha:cardinal_le ?a ?b, Hb: cardinal_le ?b ?c |- cardinal_le ?a ?c
     =>  ap (cardinal_le_transitive Ha Hb)
  | Ha:cardinal_lt ?a ?b, Hb: cardinal_le ?b ?c |- cardinal_lt ?a ?c
     =>  ap (cardinal_lt_le_trans Ha Hb)
  | Ha:cardinal_le ?a ?b, Hb: cardinal_lt ?b ?c |- cardinal_lt ?a ?c
     =>  ap (cardinal_le_lt_trans Ha Hb)
  | Ha:cardinal_lt ?a ?b, Hb: cardinal_lt ?b ?c |- cardinal_lt ?a ?c
     =>  nin Ha; co_tac
  | Ha: cardinal_le ?a ?b, Hb: cardinal_lt ?b ?a |- _
    => elim (not_card_le_lt Ha Hb)
  | Ha:cardinal_le ?x ?y, Hb:  cardinal_le ?y ?x |- _
   => solve [ rw (cardinal_antisymmetry1 Ha Hb) ; fprops ]
end.
```

The first tactic uses the property that $\mathrm{Card}(x) = x$ if x is a cardinal.

```
Ltac eq_aux:= match goal with
    H: is_cardinal ?a |- cardinal ?b = ?a => wr (cardinal_le4 H); aw
  | H: is_cardinal ?a |- ?a = cardinal ?b => wr (cardinal_le4 H); aw
  | H: is_cardinal ?a |- cardinal ?a = ?b => wr (cardinal_le4 H); aw
end.
Ltac eqtrans u:= apply equipotent_transitive with u.
Ltac eqsym:= apply equipotent_symmetric.
```

Chapter 2

Order relations. Ordered sets

2.1 Definition of an order relation

In the last chapter of the first part of this document, we studied equivalence relations, that were reflexive, symmetric and transitive. If we replace symmetric by antisymmetric, we get an *order relation*. Remember that transitive means that if $x \sim y$ and $y \sim z$ then $x \sim z$, and symmetric means that if $x \sim y$ then $y \sim x$. If a relation is symmetric and transitive, then $x \sim y$ implies $x \sim x$ and $y \sim y$. The support of a relation isthe set of all x and y that are related by the relation. A relation is reflexive on a set E if $x \in$ E is equivalent to $x \sim x$.

We say that a relation (denoted here $\prec$) is antisymmetric if $x \prec y$ and $y \prec x$ imply $x = y$. We say that it is *reflexive* if $x \prec y$ implies $x \prec x$ and $y \prec y$ (this means that the relation is reflexive on the support). An order relation on E is an order relation whose support is E (or equivalently, that is reflexive on E). Bourbaki defines preorder relations later; giving the definition right now reduces a little bit the code. A *preorder* relation is reflexive and transitive. The opposite relation of $\prec$, denoted by $\succ$, is such that $x \prec y$ and $y \succ x$ are equivalent.

```
Definition antisymmetric_r (r:Set -> Set -> Prop) :=
  forall x y, r x y -> r y x -> x = y.
Definition is_antisymmetric (r:Set) :=
  is_graph r & forall x y, related r x y -> related r y x -> x = y.
Definition reflexive_rr (r:Set -> Set -> Prop) :=
  forall x y, r x y -> (r x x &  r y y).
Definition order_r(r:Set -> Set -> Prop) :=
  transitive_r r &  antisymmetric_r r & reflexive_rr r.
Definition order_re  (r:Set -> Set -> Prop) x :=
  order_r r & reflexive_r r x.
Definition preorder_r (r:Set -> Set -> Prop) :=
  transitive_r r & reflexive_rr r.
Definition opposite_relation (r:Set -> Set -> Prop) :=
  fun x y => r y x.
```

Equality and inclusion are order relations. The opposite of an order relation is an order relation.

```
Lemma equality_is_order: order_r(fun x y => x = y).
Lemma sub_is_order: order_r sub.
Lemma opposite_is_preorder_r: forall r,
  preorder_r r -> preorder_r (opposite_relation r).
```

```
Lemma opposite_is_order_r: forall r,
  order_r r -> order_r (opposite_relation r).
```

An *order* on a set E is a graph[1] G such that the relation $(x, y) \in G$ is an order relation between x and y with substrate E. A *preorder* is similarly defined. The opposite order relation corresponds to the inverse graph, which is thus called the opposite order.

```
Definition order (r: Set) :=
  is_reflexive r & is_transitive r & is_antisymmetric r.
Definition preorder (r: Set) :=
  is_reflexive r & is_transitive r.
Definition opposite_order := inverse_graph.
Lemma order_is_graph: forall r, order r -> is_graph r.
Lemma preorder_graph : forall r, preorder r -> is_graph r.
Lemma order_preorder : forall r, order r -> preorder r.
```

If we have an order relation on E, we can take its graph, and this gives an order (this is the same construction as for equivalence relations).

```
Lemma order_has_graph0: forall r x,
  order_re r x -> is_graph_of (graph_on r x) r.
Lemma order_has_graph: forall r x,
  order_re r x -> exists g, is_graph_of g r.
Lemma order_if_has_graph: forall r g,
  is_graph g ->  is_graph_of g r ->
  order_r r -> order_re r (domain g).
Lemma order_if_has_graph2: forall r g,
  is_graph g ->  is_graph_of g r ->
  order_r r -> order g.
Lemma order_has_graph2: forall r x,
  order_re r x -> exists g,
    g = graph_on r x &
    order g & (forall u v, r u v = related g u v).
```

The next two lemmas are more often used. They say that an order can be obtained from an order relation by taking its graph on a set.

```
Lemma preorder_from_rel: forall r x,
  preorder_r r -> preorder (graph_on r x).
Lemma order_from_rel: forall r x,
  order_r r -> order (graph_on r x).
```

The traditional notation for an order relation is $x \leq y$, or $y \geq x$, we shall use *gle* and *gge* in our code; if the elements are distinct, we shall use the notations $x < y$ and $y < x$. This last relation is not reflexive. It satisfies some transitivity properties. We give here some simple properties of orders.[2]

```
Definition gle (r x y:Set) := related r x y.
Definition gge (r x y:Set) := gle r y x.
Definition glt (r x y:Set) := gle r x y &  x <> y.
Definition ggt (r x y:Set) := gge r x y &  x <> y.
```

[1] This condition has been removed; it is a consequence of the other properties

[2] Definition of *gge* changed in V3: *related* has been replaced by *gle*.

```
Lemma order_reflexivity_pr: forall  r x u v,
  order_re r x -> r u v -> (inc u x & inc v x).
Lemma order_symmetricity_pr: forall r x u v,
  order_re r x -> (r u v & r v u) = (inc u x & inc v x & u = v).
Lemma order_reflexivity: forall r a,
  order r -> inc a (substrate r) = related r a a.
Lemma order_antisymmetry: forall r a b,
  order r -> related r a b -> related r b a -> a = b.
Lemma order_transitivity: forall r a b c,
  order r -> related r a b -> related r b c -> related r a c.
Lemma lt_leq_trans : forall r x y z,
  order r -> glt r x y -> gle r y z -> glt r x z.
Lemma leq_lt_trans : forall r x y z,
  order r -> gle r x y -> glt r y z -> glt r x z.
Lemma lt_lt_trans: forall r a b c, order r ->
  glt r a b -> glt r b c -> glt r a c.
Lemma not_le_gt: forall r x y, order r -> gle r x y -> glt r y x -> False.
```

By refexivity, the substrate of an order is the domain.

```
Lemma order_is_order: forall r,
  order r -> order_r (related r).
Lemma substrate_domain_order: forall f,
  order f -> substrate f = domain f.
Lemma substrate_opposite_order: forall r,
  order r -> substrate(opposite_order r) = substrate r.
```

Examples of orders: the opposite of an order is an order; the intersection of orders is an order; equaliy is an order (it is also an equivalence).

```
Lemma opposite_is_order: forall r,
  order r -> order (opposite_order r).
Lemma diagonal_order : forall x,  order (identity_g x).
Lemma inter_rel_order : forall z,
  nonempty z -> (forall r, inc r z -> order r) ->
  order (intersection z).
```

¶ Given a relation $\prec$ and a set X, we can consider the relation "$x \in X$ and $y \in X$ and $x \prec y$". It is an order under some conditions (see also *order_from_rel* above). Its support is X provided that $x \prec x$ for all $x \in X$.

```
Lemma order_from_re11: forall r x,
  transitive_r r ->
  (forall u v, inc u x -> inc v x -> r u v -> (r u u  & r v v)) ->
  (forall u v, inc u x -> inc v x -> r u v -> r v u -> u = v) ->
  (forall u, inc u x -> r u u) ->
  order (graph_on r x).
Lemma substrate_graph_on: forall (r:Set -> Set -> Prop) x,
  (forall u, inc u x -> r u u) ->
  substrate (graph_on r x) = x.
```

In many case, we get an order on a set E by taking for $x \prec y$ a relation of the form $f(x) \subset f(y)$. This is an ordering if f is injective; moreover the ordering will be total. In the definition that follows, we consider the case where f is the identity function, and E is any set, or is the powerset of a set A.

```
Definition inclusion_suborder (b:Set) :=
  graph_on sub b.
Definition inclusion_order (a:Set) := inclusion_suborder (powerset a).

Lemma subinclusion_is_order: forall a,
  order (inclusion_suborder a).
Lemma inclusion_is_order: forall a,
  order (inclusion_order a).
Lemma substrate_subinclusion_order:forall a,
  substrate (inclusion_suborder a) = a.
Lemma substrate_inclusion_order: forall a,
  substrate (inclusion_order a) = powerset a.
Lemma subinclusion_order_rw: forall a u v ,
  related (inclusion_suborder a) u v = (inc u a & inc v a & sub u v).
Lemma inclusion_order_rw: forall a u v ,
  related (inclusion_order a) u v = (sub u a & sub v a & sub u v).
```

¶ Second example. We consider the set $\Phi(E,F)$ of functions from a subset of E to F and the extension relation. If the two functions f and g have the same target, then g extends f if and only if its graph contains the graph of f. Thus, if x is in the source of f, both values $f(x)$ and $g(x)$ are defined and equal. This is an order on $\Phi(E,F)$. Note that Bourbaki considers "g extends f" as a relation between f and g. In our definition g comes before f.

```
Definition extension_order x y :=
  graph_on extends (set_of_sub_functions x y).

Lemma extends_in_prop:  forall x y,
  order_r extends &
  (forall u, inc u (set_of_sub_functions x y) -> extends u u).

Lemma extension_is_order: forall x y,
  order (extension_order x y).
Lemma substrate_extension_order: forall x y,
  substrate (extension_order x y) = (set_of_sub_functions x y).

Lemma extension_order_rw :  forall E F f g,
  related (extension_order E F) g f =
  (inc g (set_of_sub_functions E F) & inc f (set_of_sub_functions E F)
    & extends g f).
Lemma extension_order_pr1 :  forall E F f g,
  gle (extension_order E F) g f =
  (inc g (set_of_sub_functions E F) & inc f (set_of_sub_functions E F)
    & sub (graph f) (graph g)).
Lemma extension_order_pr2 :  forall E F f g,
  gle (opposite_order (extension_order E F)) g f =
  (inc g (set_of_sub_functions E F) & inc f (set_of_sub_functions E F)
    & sub (graph g) (graph f)).
Lemma extension_order_pr :  forall E F f g x,
  related (opposite_order (extension_order E F)) f g ->
  inc x (source f) ->  W x f =  W x g.
```

¶ Third example. Let E be a set, and W the subset of $\mathfrak{P}(\mathfrak{P}(E))$ formed of all partitions of E. Recall that a partition ϖ of E is a set of sets whose union is E, so that $\varpi \subset \mathfrak{P}(E)$. Additional conditions are: the empty set is not in ϖ, the elements of ϖ are mutually disjoint. We say that ϖ is *coarser* than ϖ' if for every $Y \in \varpi'$ there exists $X \in \varpi$ such that $Y \subset X$. We shall show that it

is an order. But we start with another property: assume that X is a partition of E; if $x \in X$ then x is a subset of E hence $x \in \mathfrak{P}(E)$. We can consider the canonical injection $X \to \mathfrak{P}(E)$; this is the function defined on X that maps x to itself; the graph of this function is a family of sets; this family is a partition (in the other sense of this word) of E.

```
Definition set_of_partition_set x :=
  Zo(powerset(powerset x)) (fun z => partition z x).

Definition partition_fun_of_set y x:=
  canonical_injection y (powerset x).

Lemma partition_set_in_double_powerset: forall y x,
  partition y x -> inc y (powerset (powerset x)).
Lemma set_of_partition_rw: forall x y,
  inc y (set_of_partition_set x) = partition y x.
Lemma pfs_function: forall y x,
  partition y x -> is_function (partition_fun_of_set y x).
Lemma pfs_W: forall y x a,
  partition y x -> inc a y ->  W a (partition_fun_of_set y x) = a.
Lemma pfs_partition: forall y x,
  partition y x ->  partition_fam (graph (partition_fun_of_set y x)) x.
```

We now define *coarser* and show that it is an order on W.

```
Definition coarser x := graph_on coarser_c (set_of_partition_set x).
Lemma related_coarser: forall x y y',
  related (coarser x) y y' =
  (partition y x &  partition y' x & coarser_c y y').
Lemma coarser_related_bis: forall x y y',
  related (coarser x) y y' =
  (partition y x &  partition y' x &
    forall a, inc a y' -> exists b, inc b y & sub a b).
Lemma coarser_substrate : forall x,
  substrate (coarser x) = set_of_partition_set x.
Lemma coarser_order : forall x, order (coarser x).
Lemma smallest_partition_is_smallest: forall x y,
  nonempty x ->
  partition y x -> related (coarser x) (smallest_partition x) y.
Lemma largest_partition_is_largest: forall x y,
  partition y x -> related (coarser x) y (largest_partition x).
```

Let ω be a partition; consider the set formed of all $A \times A$ with $A \in \omega$; let $\tilde{\omega}$ be the union of all these sets. We pretend that this set is the graph of the equivalence associated to the partition. The relation "ω coarser than ω'" is equivalent to $\tilde{\omega} \supset \tilde{\omega}'$. Bourbaki says that this shows that coarser is an order. The nontrivial point is antisymmetry: we must show that $\tilde{\omega} = \tilde{\omega}'$ implies $\omega = \omega'$. This is a consequence of the fact that the sets $A \times A$ are mutually disjoint. If a and b are in the same element of ω and in A and B for ω', the pair (a, b) is in $\tilde{\omega}$, hence in $\tilde{\omega}'$, hence in $A \times A$ and $B \times B$, so that the intersection of A and B is not empty, and A = B.

```
Definition partition_relation_set_aux y x :=
  Zo (powerset (coarse x)) (fun z => exists a, inc a y & z = coarse a).
Definition partition_relation_set y x :=
  partition_relation (partition_fun_of_set y x) x.

Lemma prs_is_equivalence: forall y x,
```

```
  partition y x -> is_equivalence (partition_relation_set y x).
Lemma partition_relation_set_pr1: forall y x a,
  partition y x ->
  inc a y -> inc (coarse a) (partition_relation_set_aux y x).
Lemma partition_relation_set_pr: forall y x,
  partition y x ->
  partition_relation_set y x =
  union (partition_relation_set_aux y x).   (* 16 *)
Lemma sub_partition_relation_set_coarse:  forall y x,
  partition y x -> sub (partition_relation_set y x) (coarse x).
Lemma nondisjoint :forall a b c, inc a b -> inc a c -> disjoint b c -> False.
Lemma partition_relation_set_order: forall x y y',
  partition_set y x -> partition_set y' x ->
  sub (partition_relation_set y' x)(partition_relation_set y x) =
  related (coarser x) y y'.  (* 28 *)
Lemma partition_relation_set_order_antisymmetric: forall x y y',
  partition y x -> partition y' x ->
  (partition_relation_set y' x = partition_relation_set y x) ->
   y = y'. (* 41 *)
```

Let G be a graph, Δ be the diagonal of the substrate of G. If $\Delta \subset G$ and $G \circ G \subset G$ then $G \circ G = G$. The two assumptions say that G is reflexive and transitive, this a preorder. Antisymmetry is $G \cap G^{-1} \subset \Delta$. Since Δ is symmetric (i.e., it is its inverse), reflexivity is also $\Delta \subset G^{-1}$, so that G is an order if and only $G \circ G = G$, and $G \cap G^{-1} = \Delta$ (we know that G is an equivalence if and only $G \circ G = G$ and $G = G^{-1}$). Proposition 1 of Bourbaki [3, p. 132] considers the case of a correspondence.

```
Lemma preorder_prop1: forall g,
  is_graph g ->
  sub (identity_g (substrate g)) g -> sub (compose_graph g g) g ->
  compose_graph g g = g.
Theorem order_pr: forall r,
  order r =
    (compose_graph r r = r &
    intersection2 r (opposite_order r) = identity_g (substrate r)).  (* 37 *)
```

2.2 Preorder relations

Consider the relation "R is coarser than R′" in the set of all coverings of E. This is a preorder relation, since it is reflexive and transitive; it is not antisymmetric, for if R is a covering, $R' = R \cup \{X\}$ with $X \subset E$, then R′ is a covering which is coarser then R. Now, R is coarser than R′ if there is a $Y \in R$ such that $X \subset Y$. It can happen that $X \notin R$; in such a case, R is coarser than R′, R′ is coarser than R, but $R \neq R'$. In the case of a partition, $X \in R'$, $Y \in R'$ and $X \subset Y$ implies $X = Y$.

Here is the definition and the basic properties.

```
Lemma preorder_is_preorder: forall r,
  preorder r -> preorder_r (related r).
Lemma opposite_is_preorder1: forall r,
  preorder r -> preorder (opposite_order r).
```

The relation "$x \prec y$ and $y \prec x$" is an equivalence if $\prec$ is a preorder. Denote it by $\sim$. Then $x \prec y$ is compatible with $x \sim x'$ and $y \sim y'$. This means that if these three relations hold, then

we have also $x' \prec y'$. If $\prec$ has a graph G, then $\sim$ has a graph, which is $G \cap G^{-1}$. If G is a graph, it is a preorder if and only if $\Delta \subset G$ and $G \circ G \subset G$. We know that it implies $G \circ G = G$.

```
Lemma equivalence_preorder: forall r,
  preorder_r r ->
  equivalence_r (fun x y => r x y & r y x).
Lemma compatible_equivalence_preorder: forall r,
  let s := (fun x y => r x y & r y x) in
  preorder_r r -> forall x y x' y', r x y -> s x x' -> s y y' -> r x' y'.
Definition equivalence_associated_o r:= intersection2 r (opposite_order r).

Lemma equivalence_preorder1: forall r,
  preorder r ->
  is_equivalence (equivalence_associated_o r).
Lemma substrate_equivalence_associated_o: forall r,
  preorder r ->
  substrate (equivalence_associated_o r) = substrate r.
Lemma compatible_equivalence_preorder1: forall r u v x y,
  preorder r -> related r x y ->
  related (equivalence_associated_o r) x u ->
  related (equivalence_associated_o r) y v->
  related r u v.
Lemma preorder_prop: forall g,
  is_graph g ->
  preorder g = (sub (identity_g (substrate g)) g & sub (compose_graph g g) g).
Lemma preorder_prop2: forall g,
  preorder g -> compose_graph g g = g.
Lemma preorder_reflexivity: forall r a,
  preorder r -> inc a (substrate r) = related r a a.
```

Let $\prec$ be a preorder on a set E, and $\sim$ the equivalence associated to it. Assume that R is the graph of $\prec$, and let π be the canonical projection $E/\sim$. Given two objects of the form $u = \pi(x)$ and $v = \pi(y)$ in the quotient, we can compare u and v via $x \prec y$, this is independent of the representatives x and y. This relation has a graph, namely $S \circ G \circ S^{-1}$, where S is the graph of π. This set is also $(\pi \times \pi)\langle G \rangle$, where $\pi \times \pi$ maps $E \times E$ into $(E/\sim) \times (E/\sim)$. This relation is an order on the quotient; it is called the order relation *associated* with $\prec$.

```
Definition order_associated (r:Set) :=
  let s := (graph(canon_proj (equivalence_associated_o r))) in
  compose_graph (compose_graph s r) (opposite_order s).

Lemma order_associated_graph: forall r,
  is_graph (order_associated r).
Lemma compose3_related: forall s r u v,
  related (compose_graph (compose_graph s r) (opposite_order s)) u v =
  exists x, exists y, related s x u  & related s y v & related r x y.
Lemma order_associated_related1: forall r u v,
  preorder r->
  related (order_associated r) u v =
  ( inc u (quotient (equivalence_associated_o r)) &
    inc v (quotient (equivalence_associated_o r)) &
    exists x, exists y, inc x u & inc y v & related r x y).   (* 28 *)
Lemma order_associated_related2: forall r u v,
  preorder r->
  related (order_associated r) u v =
```

```
  ( inc u (quotient (equivalence_associated_o r)) &
    inc v (quotient (equivalence_associated_o r)) &
    forall x y, inc x u -> inc y v -> related r x y).
Lemma eao_related: forall r a b,
 related (equivalence_associated_o r) a b -> related r a b.
Lemma order_associated_substrate: forall r,
  preorder r-> substrate (order_associated r) =
  (quotient (equivalence_associated_o r)).  (* 21 *)
Lemma order_associated_order: forall r,
  preorder r-> order (order_associated r). (* 32 *)
Lemma order_associated_pr: forall r,
  preorder r ->
  order_associated r = image_by_fun (
    ext_to_prod(canon_proj (equivalence_associated_o r))
    (canon_proj (equivalence_associated_o r))) r.   (* 35 *)
```

2.3 Notation and terminology

At this point in the document, Bourbaki denotes order relations by $x \leq y$ instead of $x \prec y$. He defines an order on E by

(RO_I) The relation "$x \leq y$ and $y \leq z$" implies $x \leq z$.
(RO_{II}) The relation "$x \leq y$ and $y \leq x$" implies $x = y$.
(RO_{III}) The relation $x \leq y$ implies "$x \leq x$ and $y \leq y$".
(RO_{IV}) The relation $x \leq x$ is equivalent to $x \in E$.

Note the absence of quantifiers; in Bourbaki, if x is a free variable, P$\{x\}$ is equivalent to $(\forall x)$(P$\{x\}$). Hence the definition is correct if x, y and z are three distinct letters that do not appear in E or $\leq$. The first part of C58 is: let $\leq$ be an order relation and let x, y be two distinct letters; the relation $x \leq y$ is equivalent to "$x < y$ or $x = y$". Here, Bourbaki explicitly says that x and y are distinct. The statement is true only if $x = x$ implies $x \leq x$, for instance if $\leq$ is an order on E and $x \in E$.

```
Definition order_axioms (r s:Set) :=
  (forall x y z, gle r x y ->  gle r y z -> gle r x z) &
  (forall x y, gle r x y ->  gle r y x -> x = y) &
  (forall x y, gle r x y ->  (inc x s & inc y s)) &
  (forall x, gle r x x = inc x s) &
  is_graph r.

Lemma axioms_of_order: forall r,
  order r = (order_axioms r (substrate r)).
Lemma lt_leq_trans :  forall r x y z,
  order r -> glt r x y -> gle r y z -> glt r x z.
Lemma leq_lt_trans :  forall r x y z,
  order r -> gle r x y -> glt r y z -> glt r x z.
Lemma lt_lt_trans: forall r a b c, order r ->
  glt r a b -> glt r b c -> glt r a c.
Lemma le_pr: forall r x y,
  inc x (substrate r) -> order r -> gle r x y = (glt r x y \/ x = y).
```

We say that f is a *morphism* or *isomorphism* for two orders denoted $\leq_E$ or $\leq_F$ on E and F if f is a function from E to F compatible with the orders (this means the obvious: if $x \leq_E y$ then $f(x) \leq_F f(y)$ and conversely). A morphism is required to be injective, and an isomorphism

is required to be bijective. Properties of morphisms will be studied later. A morphism is an isomorphism on its range.

```
Definition order_isomorphism f r r':=
  (order r) & (order r') &
  (bijective f) & (substrate r = source f) &  (substrate r' = target f) &
  (forall x y, inc x (source f) -> inc y (source f) ->
    gle r x y = gle r' (W x f) (W y f)).

Definition order_morphism f r r':=
  (order r) & (order r') &
  (injective f) & (substrate r = source f) &  (substrate r' = target f)  &
  (forall x y, inc x (source f) -> inc y (source f) ->
    gle r x y = gle r' (W x f) (W y f)).
```

2.4 Ordered subsets. Product of ordered sets

Let G be a graph, A a set. Define $G_A = G \cap (A \times A)$. This is a graph, whose substrate is a subset of A. If G is reflexive, with substrate E and $A \subset E$, then the substrate of G_A is A. As a consequence, if G is an order (or preorder) on E, then G_A is an order (or preorder) on A. It is called the order *induced* by G. By abuse of notations, if the order relation associated to G is denoted $x \leq y$ then the order relation associated to G_A also denoted $x \leq y$ instead of $x \leq_A y$. The first three lemmas here say that $x \leq_A y$ implies $x \leq y$, and $x <_A y$ implies $x < y$; moreover if $x \in A$ and $y \in A$, then $x \leq_A y$ is equivalent to $x \leq y$.

```
Definition induced_order (r a:Set):=
  intersection2 r (coarse a).

Lemma related_induced_order: forall r a x y, inc x a -> inc y a ->
  gle (induced_order r a) x y = gle r x y.
Lemma related_induced_order1: forall r a x y,
  gle (induced_order r a) x y -> gle r x y.
Lemma related_induced_order2: forall r a x y,
  glt (induced_order r a) x y -> glt r x y.

Lemma related_induced_order3: forall r a x y,
  gle (induced_order r a) x y -> (inc x a & inc y a).
Lemma related_induced_order4: forall r a x y,
  glt(induced_order r a) x y -> (inc x a & inc y a).

Lemma relation_induced_order: forall r a,
  is_graph r -> is_graph (induced_order r a).

Lemma substrate_induced_order1: forall r a,
  is_reflexive r ->
  sub a (substrate r) ->  substrate (induced_order r a) = a.
Lemma substrate_induced_order: forall r a,
  order r ->
  sub a (substrate r) ->  substrate (induced_order r a) = a.
Lemma reflexive_induced_order: forall r a,
  sub a (substrate r) ->
  is_reflexive r -> is_reflexive (induced_order r a).
Lemma transitive_induced_order: forall r a,
  sub a (substrate r) ->
```

```
  is_transitive r -> is_transitive (induced_order r a).
Lemma preorder_induced_order: forall r a,
  sub a (substrate r) ->
  preorder r -> preorder (induced_order r a).
Lemma order_induced_order: forall r a,
  sub a (substrate r) ->
  order r -> order (induced_order r a).
Lemma sub_graph_coarse_substrate: forall r,
  is_graph r -> sub r (coarse (substrate r)).
Lemma induced_order_substrate: forall r,
  order r -> (* OK if r is graph *)
  induced_order r (substrate r) = r.
Lemma opposite_induced: forall r x, order r ->
  induced_order (opposite_order r) x = opposite_order (induced_order r x).
```

Let $\Phi(E,F)$ be the set of all mappings of subsets of E into F (this is the union of all $\mathscr{F}(I;F)$, for $I \subset E$). Denote here by $\Psi(E,F)$ the set of functional graphs included in $E \times F$; one could show that this is the union of all F^I for $I \subset E$). The bijection between $\mathscr{F}(I;F)$ and F^I, that associates to each function its graph extens to $\Phi(E,F) \to \Psi(E,F)$ and is an order isomorphism where the source is endowed with the relation "g extends f" between f and g (this is the opposite order of *extension_order*) and the target with the inclusion order.

```
Definition set_of_fgraphs(x y :Set):=
  (Zo(powerset (product x y)) (fun z => functional_graph z)).

Definition graph_of_function (x y:Set):=
  BL graph (set_of_sub_functions x y)
  (set_of_fgraphs x y).

Lemma graph_of_function_sub: forall x y z,
  inc z (set_of_sub_functions x y) -> sub (graph z) (product x y).
Lemma set_of_graphs_pr: forall x y z,
  inc z (set_of_sub_functions x y) -> inc (graph z) (set_of_fgraphs x y).
Lemma of_function_axioms_graph: forall x y,
  transf_axioms(fun g => (graph g)) (set_of_sub_functions x y)
  (set_of_fgraphs x y).
Lemma graph_of_fonction_function: forall x y,
  is_function (graph_of_function x y).
Lemma graph_of_function_W: forall x y f,
  inc f (set_of_sub_functions x y) -> W f (graph_of_function x y) = graph f.
Lemma graph_of_function_bijective: forall x y,
  bijective (graph_of_function x y).
Lemma graph_of_function_isomorphism: forall x y,
  order_isomorphism (graph_of_function x y)
  (opposite_order (extension_order x y))
  (inclusion_suborder (set_of_fgraphs x y)).
```

¶ If E is a set, we consider the mapping $\varpi \mapsto \tilde{\varpi}$, that maps a partition to the graph of the associated equivalence. We know that "ϖ coarser than ϖ'" is equivalent to $\tilde{\varpi} \supset \tilde{\varpi}'$. This mapping is an isomorphism on its image (when the source is endowed with the opposite of the coarse relation, and the target with $\subset$). The source is the set of partitions, the target is some subset of $\mathfrak{P}(E \times E)$.

```
Definition graph_of_partition x :=
  BL(fun y => partition_relation_set y x)
```

```
  (set_of_partition_set x)(powerset (coarse x)).

Lemma gop_axioms: forall x,
  transf_axioms (fun y => partition_relation_set y x)
     (set_of_partition_set x) (powerset (coarse x)).
Lemma gop_W: forall x y,
  partition y x ->
  W y (graph_of_partition x) = partition_relation_set y x.
Lemma gop_morphism: forall x,
  order_morphism (graph_of_partition x) (coarser x)
  (opposite_order (inclusion_order (coarse x))).
```

¶ We consider the set of all preorders on E; more precisely the set of all graphs that are preorders, and order them by inclusion. A preorder s is finer than t if $s \subset t$; this is the same as to say that elements related by s are related by t.

```
Definition set_of_preorders x :=
  Zo (powerset (coarse x))(fun z => substrate z = x & preorder z).
Definition coarser_preorder (x:E) :=
  inclusion_suborder (set_of_preorders x).

Lemma set_of_preorders_rw: forall x z,
  inc z (set_of_preorders x) = (substrate z = x & preorder z).

Lemma coarser_preorder_order: forall x,
  order (coarser_preorder x).
Lemma coarser_preorder_substrate: forall x,
  substrate (coarser_preorder x) = set_of_preorders x.
Lemma coarser_preorder_related: forall x u v,
  related (coarser_preorder x) u v =
  (preorder u & preorder v & substrate u = x & substrate v = x & sub u v).
Lemma coarser_preorder_related1: forall x u v,
  related (coarser_preorder x) u v =
  (preorder u & preorder v & substrate u = x & substrate v = x &
  forall a b, inc a x -> inc b x -> related u a b -> related v a b).
```

¶ Consider now a family of orders G_ι with substrate E_ι. Denote the relation associated to G_ι by $\leq_\iota$. On the product $\prod E_\iota$ we can consider the relation: $\forall \iota$, $x_\iota \leq_\iota x'_\iota$ between $(x_\iota)_\iota$ and $(x'_\iota)_\iota$. This is an order relation; it will be called the *product* of the order relations and its graph will be called the *product* of the orders. The graph is $\prod G_\iota$ transported from $\prod(E_\iota \times E_\iota)$ to $\prod E_\iota \times \prod E_\iota$ via the canonical bijection.

```
Definition fam_of_substrates g :=
  L (domain g) (fun i => substrate (V i g)).
Definition prod_of_substrates g := productb (fam_of_substrates g).

Definition product_order_r (g:Set): Set -> Set -> Prop :=
  fun x x' => forall i, inc i (domain g) -> gle (V i g) (V i x) (V i x').
Definition product_order g :=
  graph_on (product_order_r g)(prod_of_substrates g).
Definition product_order_axioms g:=
  fgraph g & (forall i, inc i (domain g) -> order (V i g)).

Lemma prod_of_substrates_rw: forall g x,
  product_order_axioms g ->
```

```
  (inc x  (prod_of_substrates g) = (fgraph x & domain x = domain g &
    forall i, inc i (domain g) -> inc (V i x) (substrate (V i g)))).

Lemma product_order_order: forall g,
  product_order_axioms g -> order (product_order g).
Lemma product_order_related: forall g x x',
  product_order_axioms g ->
  gle (product_order g) x x' =
  (inc x (prod_of_substrates g) & inc x' (prod_of_substrates g) &
  forall i, inc i (domain g) -> gle (V i g) (V i x)(V i x')).

Lemma product_order_substrate: forall f g,
  product_order_axioms f g -> substrate(product_order f g) = productb f.
Lemma product_order_def: forall f g,  product_order_axioms f g ->
  image_by_fun (prod_of_products_canon f f)(product_order f g)
   = (productb g).  (* 48 *)
```

Since F^E is a product, an order on F gives an order on the set of graphs of functions. We can compare two functions f and g via $f(x) \leq g(x)$. This gives a natural order on $\mathscr{F}(E;F)$. The function that associates to a function $\mathscr{F}(E;F)$ its graph in F^E is an isomorphism.

```
Definition product2_order f g:=
  graph_on (fun x x' => gle f (P x) (P x') & gle g (Q x) (Q x'))
    (product (substrate f)(substrate g)).

Lemma product2_order_pr: forall f g x x',
  gle (product2_order f g) x x' =
  (inc x (product (substrate f)(substrate g)) &
    inc x' (product (substrate f)(substrate g)) &
    gle f (P x) (P x') & gle g (Q x) (Q x')).
Lemma product2_order_preorder_substrate: forall f g,
  preorder f -> preorder g -> substrate (product2_order f g) =
  product (substrate f) (substrate g).
Lemma product2_order_substrate: forall f g,
  order f -> order g -> substrate (product2_order f g) =
  product (substrate f) (substrate g).
Lemma product2_order_preorder: forall f g,
  preorder f -> preorder g -> preorder (product2_order f g).
Lemma product2_order_order: forall f g,
  order f -> order g -> order (product2_order f g).

Lemma substrate_preorder_product2_order: forall f g,
  preorder f -> preorder g -> substrate (product2_order f g) =
  product (substrate f) (substrate g).

Definition graph_order_r(x y g:Set): Set -> Set -> Prop :=
  fun z z' =>  forall i, inc i x-> related g (V i z)(V i z').
Definition graph_order x y g :=
  graph_on (graph_order_r x g) (set_of_gfunctions x y).

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 &
  forall 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 u v)
```

```
  (set_of_functions x y).
Definition cst_graph x y:= L x (fun _ => y).
```

Now some properties of graph orderings.

```
Lemma product_order_axioms_x:forall x y g, order g ->
  product_order_axioms (cst_graph x g).
Lemma cst_graph_pr: forall x y,
  productb (cst_graph x y) = set_of_gfunctions x y.

Lemma graph_order_r_pr: forall x g z z',
  graph_order_r x g z z' = product_order_r  (cst_graph x g) z z'.
Lemma graph_order_r_pr: forall x y g z z', substrate g = y ->
  graph_order_r x g z z' =
  product_order_r (cst_graph x g) z z'.
Lemma graph_order_pr1:forall x y g, substrate g = y ->
  (set_of_gfunctions x y) = prod_of_substrates (cst_graph x g).
Lemma graph_order_pr:forall x y g, order g -> substrate g = y ->
  graph_order x y g = product_order (cst_graph x g).
Lemma graph_order_order: forall x y g, order g -> substrate g = y ->
  order (graph_order x y g).
Lemma graph_order_substrate: forall x y g, order g -> substrate g = y ->
  substrate (graph_order x y g) = set_of_gfunctions x y.
```

Now some properties of function orderings.

```
Lemma function_order_reflexive: forall x y r u,
  order r -> substrate r = y ->
  inc u (set_of_functions x y) -> gle (function_order x y r) u u.
Lemma function_order_substrate: forall x y r,
  order r -> substrate r = y ->
  substrate (function_order x y r) =set_of_functions x y.
Lemma function_order_order: forall x y r,
  order r -> substrate r = y -> order (function_order x y r).
Lemma function_order_pr: forall x y r f f',
  order r -> substrate r = y ->
  gle (function_order x y r) f f' =
  (inc f (set_of_functions x y) &
    inc f' (set_of_functions x y) &
    forall i, inc i x ->  gle r (W i f) (W i f')).
Lemma function_order_isomorphism: forall x y r,
  order r -> substrate r = y ->
  order_isomorphism (BL graph (set_of_functions x y)(set_of_gfunctions x y))
  (function_order x y r)(graph_order x y r).
```

¶ We consider now the product of two order relations (it is similar to the product of a family of orders that has two elements).

```
Definition product2_order_r f g :=
  fun x x' =>
    inc x (product (substrate f)(substrate g)) &
    inc x' (product (substrate f)(substrate g)) &
    related f (P x) (P x') & related g (Q x) (Q x').

Definition product2_order f g:=
```

```
  graph_on (product2_order_r f g)(product (substrate f)(substrate g)).

Lemma preorder_r_product2_order: forall f g,
  preorder f -> preorder g -> preorder_r (product2_order_r f g).
Lemma order_r_product2_order: forall f g,
  order f -> order g -> order_r (product2_order_r f g).
Lemma preorder_product2_order: forall f g,
  preorder f -> preorder g -> preorder (product2_order f g).
Lemma order_product2_order: forall f g,
  order f -> order g -> order (product2_order f g).
Lemma product2_order_pr: forall f g x y,
  related (product2_order f g) x y = product2_order_r f g x y.
Lemma substrate_preorder_product2_order: forall f g,
  preorder f -> preorder g -> substrate (product2_order f g) =
  product (substrate f) (substrate g).
Lemma substrate_order_product2_order: forall f g,
  order f -> order g -> substrate (product2_order f g) =
  product (substrate f) (substrate g).
```

2.5 Increasing mappings

Consider two sets E and F orderd by $\leq_E$ and $\leq_F$ and a function $f : E \to F$. We say that a f is *increasing* if $x \leq_E y$ implies $f(x) \leq_F f(y)$ and *decreasing* if $x \leq_E y$ implies $f(x) \geq_F f(y)$. increasing when $\leq$ is a preorder. A function is *strictly increasing* or *strictly decreasing* if $x < y$ implies $f(x) < f(y)$ or $f(x) > f(y)$. (Bourbaki considers the case of preorders in the non-strict case). A function that is increasing or decreasing is *monotone*.[3]

```
Definition increasing_fun f r r':=
  is_function f & order r & order r' & substrate r = source f
  & substrate r' = target f  &
  forall x y,  gle r x y  -> gle r' (W x f) (W y f).
Definition decreasing_fun f r r':=
  is_function f & order r & order r' & substrate r = source f
  & substrate r' = target f  &
  forall x y,  gle r x y  -> gge r' (W x f) (W y f).
Definition monotone_fun f r r':=
  increasing_fun f r r' \/  decreasing_fun f r r'.

Definition strict_increasing_fun f r r':=
  is_function f & order r & order r' & substrate r = source f
  & substrate r' = target f  &
  forall x y,  glt r x y  -> glt r' (W x f) (W y f).
Definition strict_decreasing_fun f r r':=
  is_function f & order r & order r' & substrate r = source f
  & substrate r' = target f  &
  forall x y,  glt r x y  -> ggt r' (W x f) (W y f).
Definition strict_monotone_fun f r r':=
  strict_increasing_fun f r r' \/  strict_decreasing_fun f r r'.
```

Some consequences when we replace one order by its opposite.

```
Lemma increasing_fun_reva : forall f r r',
```

[3] The definition has been simplified in Version 3.

```
   increasing_fun f r r' -> decreasing_fun f r (opposite_order r').
Lemma increasing_fun_revb : forall f r r',
   increasing_fun f r r' -> decreasing_fun f (opposite_order r) r'.
Lemma decreasing_fun_reva : forall f r r',
   decreasing_fun f r r' -> increasing_fun f r (opposite_order r').
Lemma decreasing_fun_revb : forall f r r',
   decreasing_fun f r r' -> increasing_fun f (opposite_order r) r'.
Lemma monotone_fun_reva : forall f r r',
   monotone_fun f r r' -> monotone_fun f r (opposite_order r').
Lemma monotone_fun_revb : forall f r r',
   monotone_fun f r r' -> monotone_fun f (opposite_order r) r'.
```

Same for strictly monotone.

```
Lemma opposite_gle: forall r x y, order r ->
  gle (opposite_order r) x y = gle r y x.
Lemma opposite_gge: forall r x y, order r ->
  gge (opposite_order r) x y = gle r x y.

Lemma glt_inva: forall r x y,
  order r ->   glt r x y = ggt (opposite_order r) x y.
Lemma ggt_inva: forall r x y,
  order r ->   ggt r x y = glt (opposite_order r) x y.
Lemma ggt_invb: forall r x y,
  ggt r x y = glt r y x.
Lemma strict_increasing_fun_reva : forall f r r',
  strict_increasing_fun f r r' -> strict_decreasing_fun f r (opposite_order r').
Lemma strict_increasing_fun_revb : forall f r r',
  strict_increasing_fun f r r' -> strict_decreasing_fun f (opposite_order r) r'.
Lemma strict_decreasing_fun_reva : forall f r r',
  strict_decreasing_fun f r r' -> strict_increasing_fun f r (opposite_order r').
Lemma strict_decreasing_fun_revb : forall f r r',
  strict_decreasing_fun f r r' -> strict_increasing_fun f (opposite_order r) r'.
Lemma strict_monotone_fun_reva : forall f r r',
  strict_monotone_fun f r r' -> strict_monotone_fun f r (opposite_order r').
Lemma strict_monotone_fun_revb : forall f r r',
   strict_monotone_fun f r r' -> strict_monotone_fun f (opposite_order r) r'.
```

If f is constant, then f is increasing and decreasing. Conversely, the identity function is increasing and decreasing on a set ordered by equality. This function is not constant when the set has more than one element. Let E be set, f the function that maps $X \subset E$ to its complementary. This is a strictly decreasing function when the powerset is ordered by inclusion.

```
Lemma constant_fun_increasing: forall f r r',
  order r -> order r' -> substrate r = source f  ->
  substrate r' = target f -> is_constant_function f ->
  increasing_fun f r r'.
Lemma  constant_fun_decreasing: forall f r r',
  order r -> order r' -> substrate r = source f  ->
  substrate r' = target f -> is_constant_function f ->
  decreasing_fun f r r'.
Lemma identity_increasing_decreasing : forall x,
  let r := identity_g x in
    (increasing_fun (identity x) r r &decreasing_fun (identity x) r r).
Lemma complementary_decreasing: forall E,
  strict_decreasing_fun(BL (fun X => complement E X)(powerset E)(powerset E))
  (inclusion_order E)  (inclusion_order E).
```

Let U_x be the set of upper bounds of $\{x\}$ (the case of a non-singleton will be studied later). We have $x \leq y$ if and only if $U_y \subset U_x$. The function $x \mapsto U_x$ is strictly decreasing.

```
Definition set_of_majorants1 x r:=
  Zo (substrate r)(fun y => gle r x y).
Lemma set_of_majorants1_pr: forall x y r,
  order r -> inc x (substrate r) -> inc y (substrate r) ->
  (gle r x y = sub (set_of_majorants1 y r) (set_of_majorants1 x r)).
Lemma set_of_majorants1_decreasing: forall r, (* 19 *)
 order r ->
 strict_decreasing_fun(BL (fun x=> (set_of_majorants1 x r))
   (substrate r)(powerset (substrate r))) r  (inclusion_order (substrate r)).
```

If a function is injective, monotone implies strictly monotone. If a function is bijective, it is an isomorphism if and only if the function and its inverse are increasing. An isomorphism remains one if the ordering on the source and target are replaced by the opposite ones.

```
Lemma strict_increasing_from_injective: forall f r r',
  injective f -> increasing_fun f r r' -> strict_increasing_fun f r r'.
Lemma strict_decreasing_from_injective: forall f r r',
  injective f ->decreasing_fun f r r' -> strict_decreasing_fun f r r'.
Lemma strict_monotone_from_injective: forall f r r',
  injective f -> monotone_fun f r r' -> strict_monotone_fun f r r'.
Lemma order_isomorphism_increasing: forall f r r',
  order_isomorphism f r r' ->
  strict_increasing_fun f r r'.
Lemma order_morphism_increasing: forall f r r',
  order_morphism f r r' ->
  strict_increasing_fun f r r'.
Lemma order_isomorphism_pr: forall f r r',
  order r -> order r' ->
  bijective f -> substrate r = source f -> substrate r' = target f ->
  (order_isomorphism f r r' =
    (increasing_fun f r r' & increasing_fun (inverse_fun f) r' r)).
Lemma order_isomorphism_opposite: forall g r r',
  order_isomorphism g r r' ->
  order_isomorphism g (opposite_order r) (opposite_order r').
```

Assume that we have two ordered sets E and E′, decreasing functions u and v from E to E′ and E′ to E. Assume $u(v(x)) \geq x$ and $v(u(x')) \geq x'$ for all x and x'. Proposition 2 [3, p. 139] says $u \circ v \circ u = u$ and $v \circ u \circ v = v$.

```
Theorem decreasing_composition: forall u v r r',
  decreasing_fun u r r' -> decreasing_fun v r' r ->
  (forall x, inc x (substrate r) -> gle r (W (W x u) v) x) ->
  (forall x', inc x' (substrate r') -> gle r' (W (W x' v) u) x') ->
  (compose u (compose v u) = u & compose v (compose u v) = v).  (* 20 *)
```

2.6 Maximal and minimal elements

Bourbaki says: if E is a set with a preorder, then $a \in$ E is *minimal* (resp. *maximal*) in E if $x \leq a$ (resp. $x \geq a$) implies $x = a$. In the definition given here, we do not say that r is an order or a preorder.[4]

[4] Definition simplified in V3; we removed the condition that x is in the substrate since this is implied by $x \leq a$

```
Definition maximal_element r a:=
  inc a (substrate r) & forall x, gle r a x -> x = a.
Definition minimal_element r a:=
  inc a (substrate r) & forall x, gle r x a -> x = a.
```

Examples. In $\mathfrak{P}(E)$, the empty set is the least element for inclusion. If we remove it, minimal elements are singletons. On the set of partial functions ordered by extension, maximal elements are total functions (because non-total functions can be extended).

```
Lemma maximal_element_opp: forall r a,
  order r -> maximal_element r a -> minimal_element (opposite_order r) a.
Lemma minimal_element_opp: forall r a,
  order r -> minimal_element r a -> maximal_element (opposite_order r) a.
Lemma maximal_opposite: forall r x,
  order r -> maximal_element (opposite_order r) x = minimal_element r x.

Lemma minimal_inclusion: forall E y,
  let F:= complement (powerset E) (singleton emptyset) in
  inc y F -> (minimal_element (inclusion_suborder F) y = is_singleton y).

Lemma maximal_prolongation: forall E F x, (* 12 *)
  nonempty F -> inc x (set_of_sub_functions E F) ->
  maximal_element (opposite_order (extension_order E F)) x = (source x = E).
```

2.7 Greatest element and least element

We start with the definition of the *greatest* and *least* element of an order relation. Trivial properties follow. If $\leq$ is an order on E, then a is a greatest element for the order if $a \in E$ and if for all $x \in E$ we have $x \leq a$ (the condition $a \in E$ could be relaxed in E is not empty). A greatest element need not exist, but is unique. It is maximal. A set that has a greatest element has a unique maximal element. The greatest (resp. least) element is sometimes denoted max (resp. min). We give two properties: if E has a least element then $\min E$ is a least element, and if x is a least element of E then $\min E = x$. If $E' \subset E$ then $\min E = \min E'$ provided that the big set has a least element that belongs to the small set. If $\min E = \max E$, then E has a single element.

```
Definition greatest_element r a:=
  inc a (substrate r) & forall x, inc x (substrate r) -> gle r x a.
Definition least_element r a:=
  inc a (substrate r) & forall x, inc x (substrate r) -> gle r a x.
Definition the_least_element r:=
  choose (fun x=> least_element r x).
Definition the_greatest_element r:=
  choose (fun x=> greatest_element r x)

Lemma unique_greatest: forall a b r,
  order r -> greatest_element r a -> greatest_element r b  -> a = b.
Lemma unique_least: forall a b r,
  order r -> least_element r a -> least_element r b -> a = b.

Lemma the_least_element_pr: forall r,
  order r -> (exists u, least_element r u) ->
  least_element r (the_least_element r).
```

```
Lemma the_greatest_element_pr: forall r,
  order r -> (exists u, greatest_element r u) ->
  greatest_element r (the_greatest_element r).

Lemma the_least_element_pr2: forall r x, order r ->
  least_element r x ->   the_least_element r = x.
Lemma the_greatest_element_pr2: forall r x, order r ->
  greatest_element r x ->   the_greatest_element r = x.
Lemma greatest_induced: forall r s x, order r ->
  sub s (substrate r) -> inc x s ->
  greatest_element r x ->
  the_greatest_element r = the_greatest_element (induced_order r s).
Lemma least_induced: forall r s x, order r ->
  sub s (substrate r) -> inc x s ->
  least_element r x ->
  the_least_element r = the_least_element (induced_order r s).

Lemma least_not_greatest:  forall r x , order r ->
  least_element r x -> greatest_element r x ->
  is_singleton (substrate r).
```

More simple properties.

```
Lemma greatest_reverse: forall a r,
  order r -> greatest_element r a -> least_element (opposite_order r) a.
Lemma least_reverse: forall a r,
  order r -> least_element r a -> greatest_element (opposite_order r) a.
Lemma the_least_reverse: forall r, order r ->
  (exists a, greatest_element r a) ->
  the_least_element (opposite_order r) =  the_greatest_element r.
Lemma greatest_maximal: forall a r,
  order r -> greatest_element r a -> maximal_element r a.
Lemma least_minimal: forall a r,
  order r -> least_element r a -> minimal_element r a.
Lemma greatest_unique_maximal: forall a b r,
  greatest_element r a -> maximal_element r b -> a = b.
Lemma least_unique_minimal: forall a b r,
  least_element r a -> minimal_element r b -> a = b.
```

¶ Now some examples. If $\mathfrak{S}$ is a subset of $\mathfrak{P}(E)$, then the upper and lower bounds of $\mathfrak{S}$ are the intersection and union; we anticipate a bit: for the moment being, we just say that the least and greatest elements are the intersection and union, provided they are in $\mathfrak{S}$. Note that E need not be mentioned in the theorems.

```
Lemma least_is_intersection: forall s a,
 least_element (inclusion_suborder s) a ->
  nonempty s -> a = intersection s.
Lemma greatest_is_union: forall s a,
  greatest_element  (inclusion_suborder s) a -> a = union s.
Lemma intersection_is_least: forall s,
  inc (intersection s) s ->
  least_element (inclusion_suborder s) (intersection s) .
Lemma union_is_greatest: forall s,
  inc (union s) s -> greatest_element (inclusion_suborder s) (union s).
```

Some applications. On $\mathfrak{P}(E)$, the least element is $\emptyset$, the greatest is E. On the set of partial functions from E to F, there is no greatest element if F has at least two elements (constant

functions defined on the whole of E are maximal; if there is a greatest element, all constants are equal).

```
Lemma emptyset_is_least: forall E,
  least_element (inclusion_order E) emptyset.
Lemma wholeset_is_greatest: forall E,
  greatest_element (inclusion_order E) E.
Definition empty_function_tg F := BL (fun x => x) emptyset F.
Lemma empty_function_tg_function: forall F,
  is_function (empty_function_tg F).
Lemma least_prolongation: forall E F,
  least_element (opposite_order (extension_order E F))
  (empty_function_tg F).
Lemma greatest_prolongation: forall E F x,
  greatest_element (opposite_order (extension_order E F)) x ->
  nonempty E -> small_set F. (* 21 *)
```

Bourbaki notices that if E is a set, Δ the diagonal of E, then Δ is the least equivalence or preorder on E (for the inclusion order induced on $\mathfrak{P}(E \times E)$). This a consequence of the next lemma. Note: We have defined *finer_equivalence* and shown that if S and R are equivalences on a same set, then S is finer than R if and only if $S \subset R$. Thus, on the set of equivalences on E, the inclusion order is *finer_equivalence* and we have already shown that the least element is the diagonal and the greatest one is $E \times E$.

```
Lemma least_equivalence: forall r,
  is_reflexive r -> sub (diagonal (substrate r)) r.
```

Proposition 3 [3, p. 140] in Bourbaki says: Let E be an ordered set and let E' be the disjoint union of E and a set $\{a\}$ consisting of a single element. Then there exists a unique ordering on E' which induces the given ordering on E and for which a is the greatest element of E'.

There is no definition of "disjoint union", but section II.4.8 defines the sum of a family of sets as the union of a family of mutually disjoint sets. Thus, given E and $A = \{a\}$, there exists $\bar{E}$ and $\bar{A}$, two disjoint sets, two bijections $f : E \to \bar{E}$ and $g : A \to \bar{A}$, and E' is the union of $\bar{E}$ and $\bar{A}$. Because g is a bijection, there is an element a' such that $\{a'\} = \bar{A}$. The proposition is now: if G is the given order, then $(f \times f)\langle G \rangle$ is an order, its substrate is $f\langle E \rangle$, namely $\bar{E}$, and there is a unique order on $\bar{E} \cup \{a'\}$, extending $(f \times f)\langle G \rangle$ and with a' as greatest element. In our implementation we consider two separate lemmas.

```
Definition order_with_greatest r a :=
  union2 r (product (tack_on (substrate r) a) (singleton a)).

Lemma singleton_pr1: forall a x, nonempty x ->
  (forall z, inc z x -> z = a) -> x = singleton a.
Lemma order_with_greatest_pr: forall r a,  (* 51 *)
  let r':=order_with_greatest r a in
    order r ->  ~ (inc a (substrate r)) ->
    (order r' & substrate r' = tack_on (substrate r) a &
      r = intersection2 r' (coarse (substrate r)) & greatest_element r' a).
Lemma order_transportation: forall f r,
  let r':= image_by_fun (ext_to_prod f f) r in
  bijective f -> order r -> substrate r = source f ->
  (order r' & substrate r' = target f). (* 64 *)
Theorem adjoin_greatest: forall r a E,
  order r -> substrate r = E -> ~ (inc a E) ->
```

```
  exists_unique (fun r' => order r' & substrate r' = tack_on E a &
    r = intersection2 r' (coarse E) & greatest_element r' a).  (* 35 *)
```

If r is an order (denoted by $\leq$) on E (Bourbaki considers the case of pre-orders) we say that a part A of E is *cofinal* (or *coinitial*) if for all $x \in$ E there is a $y \in$ A such that $x \leq y$ (or $y \leq x$). Bourbaki says "To say that an ordered set E has a greatest element therefore means that E has a cofinal subset consisting of a single element". The lemmas given here talk about an object r and its substrate E. We do not assume that r is an order: we do not even assume that r is a graph (what happens is the following: if r' is the set of pairs (x, y) in r, then r' is the the graph of the relation associated to r, for which x and y are related if and only if $(x, y) \in r$).

```
Definition cofinal_set r a :=
  sub a (substrate r) &
  (forall x, inc x (substrate r) -> exists y, inc y a & gle r x y).
Definition coinitial_set r a :=
  sub a (substrate r) &
  (forall x, inc x (substrate r) -> exists y, inc y a & gle r y x).
Lemma exists_greatest_cofinal: forall r,
  (exists x, greatest_element r x) =
  (exists a, cofinal_set r a & is_singleton a).
Lemma exists_least_coinitial: forall r,
  (exists x, least_element r x) =
  (exists a, coinitial_set r a & is_singleton a).
```

2.8 Upper and lower bounds

Given an order r on a set E denoted by $\leq$ and a set X, an element x is said to be an *upper bound* for r and X if $y \in$ X implies $y \leq x$. A *lower bound* is an element x such that $y \in$ X implies $x \leq y$. If the set X is not empty, we deduce that $x \in$ E; in order to cover the case X $= \emptyset$, we add the condition $x \in$ E; the set of upper bounds of the empty set is thus E. Note that Bourbaki assumes that r is a preorder (but in most examples, and in the next section, it will be an order).

```
Definition upper_bound r X x :=
  inc x (substrate r) & forall y, inc y X -> gle r y x.
Definition lower_bound r X x :=
  inc x (substrate r) & forall y, inc y X -> gle r x y.
```

The first properties given here are trivial. If we have an order on E and if X is a subset of E, we can consider the order induced on X; this may have a least element m or a greatest element M. If these quantities exist, they are in X and are an upper or lower bound of X for the relation on E. Converse holds: if X has an upper bound in X, it is M.

```
Lemma opposite_upper_bound: forall x X r, order r ->
  upper_bound r X x = lower_bound (opposite_order r) X x.
Lemma opposite_lower_bound: forall x X r, order r ->
  lower_bound r X x = upper_bound  (opposite_order r) X x.
Lemma smaller_lower_bound: forall x y X r, preorder r ->
  lower_bound r X x -> gle r y x -> lower_bound r X y.
Lemma greater_upper_bound: forall x y X r, preorder r ->
  upper_bound r X x -> gle r x y -> upper_bound r X y.
Lemma sub_lower_bound: forall x X Y r,
```

```
  lower_bound r X x -> sub Y X-> lower_bound r Y x.
Lemma sub_upper_bound: forall x X Y r,
  upper_bound r X x -> sub Y X -> upper_bound r Y x.
Lemma least_element_pr: forall X r, order r -> sub X (substrate r) ->
  (exists a, least_element (induced_order r X) a) =
  (exists x, lower_bound r X x & inc x X).
Lemma greatest_element_pr: forall X r, order r -> sub X (substrate r) ->
  (exists a, greatest_element (induced_order r X) a) =
  (exists x, upper_bound r X x & inc x X).
```

We consider now *bounded* sets, that are sets that have a bound.

```
Definition bounded_above r X := exists x, upper_bound r X x.
Definition bounded_below r X := exists x, lower_bound r X x.
Definition bounded_both r X := bounded_above r X & bounded_below r X.

Lemma bounded_above_sub: forall X Y r,
  sub Y X ->bounded_above r X  -> bounded_above r Y.
Lemma bounded_below_sub: forall X Y r,
  sub Y X ->bounded_below r X  -> bounded_below r Y.
Lemma bounded_both_sub: forall X Y r,
  sub Y X ->bounded_both r X  -> bounded_both r Y.
Lemma singleton_bounded:forall x r,
  is_singleton x -> order r -> sub x (substrate r) -> bounded_both r x.
```

2.9 Least upper bound and greatest lower bound

The Bourbaki definition is: *let* E *be an ordered set and let* X *be a subset of* E. *An element of* E *is said to be* greatest lower bound *of* X *in* E *if it is the greatest element of the set of lower bounds of* X *in* E. Let W_X be the set of lower bounds of X; it is a subset of E, hence can be ordered with the order induced from E. This may have a greatest element x. This is in W_X hence is a lower bound of X and if z is another lower bound we have $z \leq x$. The converse is true, hence we have a characterization of the greatest lower bound that does not involve the set W_X. Similarly the *least upper bound* of X is an upper bound x such that if z is another upper bound, then $x \leq z$.

```
Definition greatest_lower_bound r X x :=
  greatest_element (induced_order r (Zo (substrate r)
    (fun z => lower_bound r X z))) x.
Definition least_upper_bound r X x :=
  least_element (induced_order r (Zo (substrate r)
    (fun z => upper_bound r X z))).

Lemma greatest_lower_bound_pr: forall r X x,
  order r -> sub X (substrate r) ->
  greatest_lower_bound r X x = (lower_bound r X x
    & forall z, lower_bound r X z -> gle r z x).
Lemma least_upper_bound_pr: forall r X x,
  order r -> sub X (substrate r) ->
  least_upper_bound r X x = (upper_bound r X x
    & forall z, upper_bound r X z -> gle r x z).
```

The greatest lower bound and least upper bound are also called supremum and infimum and denoted by $\sup_E X$ and $\inf_E X$. As usual, the order is $\leq$ and the substrate is E; in some cases

the set E is not mentioned. If X = $\{x, y\}$ we often write sup(x, y) and inf(x, y). The sup does not always exists, but is unique since there is a unique least element. We give a characterization of the sup and the inf when they exist. The case of two arguments is also provided.[5]

```
Lemma supremum_unique: forall x y X r, order r ->
  least_upper_bound r X x  -> least_upper_bound r X y  -> x = y.
Lemma infimum_unique: forall x y X r, order r ->
  greatest_lower_bound r X x  -> greatest_lower_bound r X y  -> x = y.

Definition supremum r X  := choose (fun x=> least_upper_bound r X x).
Definition infimum r X  := choose (fun x=> greatest_lower_bound r X x).
Definition has_supremum r X :=(exists x, least_upper_bound r X x).
Definition has_infimum r X :=(exists x, greatest_lower_bound r X x).
Definition sup r x y := supremum r (doubleton x y).
Definition inf r x y := infimum r (doubleton x y).

Lemma supremum_pr1: forall X r,
  has_supremum r X ->
  least_upper_bound r X (supremum r X).
Lemma infimum_pr1: forall X r,
  has_infimum r X ->
  greatest_lower_bound r X (infimum r X).
Lemma supremum_pr2: forall r X a, order  r ->
  least_upper_bound r X a -> a = supremum r X.
Lemma infimum_pr2: forall r X a, order  r ->
  greatest_lower_bound r X a -> a = infimum r X.
Lemma inc_supremum_substrate : forall X r,
  order r -> sub X (substrate r) -> has_supremum r X ->
  inc (supremum r X) (substrate r).
Lemma inc_infimum_substrate : forall X r,
  order r -> sub X (substrate r) -> has_infimum r X ->
  inc (infimum r X) (substrate r).
Lemma supremum_pr: forall X r,
  order r ->  sub X (substrate r) -> has_supremum r X ->
  (upper_bound r X (supremum r X) &
    forall z, upper_bound r X z -> gle r (supremum r X) z).
Lemma infimum_pr: forall X r,
  order r ->  sub X (substrate r) -> has_infimum r X ->
  (lower_bound r X (infimum r X) &
    forall z, lower_bound r X z -> gle r z (infimum r X)).
Lemma sup_pr: forall a b r,
  order r -> inc a (substrate r) -> inc b (substrate r)
  -> has_supremum r (doubleton a b) ->
  (gle r a (sup r a b) &  gle r b (sup r a b) &
    forall z, gle r a z ->  gle r b z -> gle r (sup r a b) z).
Lemma inf_pr: forall a b r,
  order r -> inc a (substrate r) -> inc b (substrate r)
  -> has_infimum r (doubleton a b) ->
  (gle r (inf r a b) a &  gle r (inf r a b) b&
    forall z, gle r z a ->  gle r z b -> gle r z (inf r a b)).
Lemma least_upper_bound_doubleton: forall r x y z,
  order r -> gle r x z -> gle r y z ->
  (forall t, gle r x t -> gle r y t -> gle r z t) ->
  least_upper_bound r (doubleton x y) z.
Lemma greatest_lower_bound_doubleton: forall r x y z,
```

[5] Some lemmas simplified in V3

```
  order r -> gle r z x  -> gle r z y ->
  (forall t, gle r t x -> gle r t y -> gle r t z) ->
  greatest_lower_bound r (doubleton x y) z.
```

We show here the following claim: if a subset X of E has a greatest element *a*, then *a* is the least upper bound of X in E.

```
Lemma greatest_is_sup: forall r X a,
  order r -> sub X (substrate r) ->
  greatest_element (induced_order r X) a -> least_upper_bound r X a.
Lemma least_is_inf: forall r X a,
  order r -> sub X (substrate r) ->
  least_element (induced_order r X) a -> greatest_lower_bound r X a.
```

The roles of inf and sup are exchanged if we replace the order by its opposite.

```
Lemma inf_sup_opp: forall r X a,
  order r -> sub X (substrate r) ->
  greatest_lower_bound r X a = least_upper_bound (opposite_order r) X a.
Lemma sup_inf_opp: forall r X a,
  order r -> sub X (substrate r) ->
  least_upper_bound r X a = greatest_lower_bound (opposite_order r) X a.
```

Examples. We study the sup and inf of the empty set.

```
Lemma set_of_lower_bounds_emptyset: forall r ,
  Zo (substrate r) (fun z => lower_bound r emptyset z) = substrate r.
Lemma set_of_upper_bounds_emptyset: forall r ,
  Zo (substrate r) (fun z => upper_bound r emptyset z) = substrate r.
Lemma least_upper_bound_emptyset: forall r x, order r ->
  least_upper_bound r emptyset x = least_element r x.
Lemma greatest_lower_bound_emptyset: forall r x, order r ->
  greatest_lower_bound r emptyset x = greatest_element r x.
```

If $\mathfrak{S}$ is a subset of $\mathfrak{P}$(E), then the upper and lower bounds of $\mathfrak{S}$ are the union and intersection, as claimed before. Is S is empty, the intersection is undefined, and the greatest lower bound is the greatest element, namely E. Assume $\mathfrak{S} \subset \mathfrak{F}$ and $\mathfrak{F} \subset \mathfrak{P}$(E); then the upper and lower bounds of $\mathfrak{S}$ in $\mathfrak{F}$ are the union and intersection, provided that these elements are in $\mathfrak{F}$.

```
Lemma intersection_is_inf: forall s E, sub s (powerset E) ->
  greatest_lower_bound (inclusion_order E) s
  (Yo (nonempty s) (intersection s) E).
Lemma union_is_sup: forall s E, sub s (powerset E) ->
  least_upper_bound (inclusion_order E) s (union s).

Lemma union_is_sup1: forall s F E, sub F (powerset E) ->
  sub s F -> inc (union s) F ->
  least_upper_bound (inclusion_suborder F) s (union s).

Lemma intersection_is_inf1: forall s F E, sub F (powerset E) ->
  sub s F -> inc (Yo (nonempty s) (intersection s) E) F ->
  greatest_lower_bound (inclusion_suborder F) s
  (Yo (nonempty s) (intersection s) E).
```

Third example. If u is in $\Phi(E,F)$, the set of partial functions from E to F, we denote its domain by $D(u)$. If Θ is subset of $\Phi(E,F)$, it has a least upper bound if and only if for all u and v in the family, for all $x \in D(u) \cap D(v)$ we have $u(x) = v(x)$. Denote this property by $P(u,v)$. We use an auxiliary result: if $u \leq w$, the condition $x \in D(u) \cap D(w)$ is equivalent to $x \in D(u)$, and $P(u,w)$ is true. Thus, if u and v are bounded by w, $P(u,v)$ is true. Conversely, we know that if P is true on Θ there is a f function defined on the union of the $D(u)$ such that $u \leq f$. This is the least upper bound.

```
Lemma sup_extension_order1: forall E F T f,
  sub T (set_of_sub_functions E F) ->
  least_upper_bound (opposite_order (extension_order E F)) T f ->
  forall u v x, inc u T -> inc v T ->  inc x (source u) ->  inc x (source v) ->
    W x u = W x v.
Lemma sup_extension_order2: forall E F T,
  sub T (set_of_sub_functions E F) ->
  (forall u v x, inc u T -> inc v T -> inc x (source u) -> inc x (source v) ->
    W x u = W x v) ->
  exists x, least_upper_bound (opposite_order (extension_order E F)) T x &
     (source x = unionf T source) &
     (range (graph x) = unionf T (fun u => (range (graph u))))&
     (graph x) = unionf T graph.   (* 29 *)
```

If f is a function with source A and if its target is an ordered set, the supremum of the image $f\langle A\rangle$ is denoted by $\sup_{x\in A} f(x)$. The infimum is denoted by $\inf_{x\in A} f(x)$. For typographical reasons, for in-text formulas, the notations $\sup_{x\in A} f(x)$, $\inf_{x\in A} f(x)$ are preferred. If f is the identity function, one can write $\sup_{x\in A} x$ or $\inf_{x\in A} x$. We give a characterization of the sup and inf of a function.

```
Definition is_sup_fun r f x := least_upper_bound r (image_of_fun f) x.
Definition is_inf_fun r f x := greatest_lower_bound r (image_of_fun f) x.

Lemma is_sup_fun_pr: forall r f x, order r -> substrate r = target f ->
  is_function f ->
 is_sup_fun r f x = (inc x (target f)
   & (forall a, inc a (source f) -> gle r (W a f) x)
   & forall z, inc z (target f) -> (forall a, inc a (source f)
     -> gle r (W a f) z) -> gle r x z).
Lemma is_inf_fun_pr: forall r f x, order r -> substrate r = target f ->
  is_function f ->
  is_inf_fun r f x = (inc x (target f)
   & (forall a, inc a (source f) -> gle r x (W a f))
   & forall z, inc z (target f) -> (forall a, inc a (source f)
     -> gle r z (W a f)) -> gle r z x).
```

In general, we consider a family rather than a function (i.e., a functional graph instead of a function).

```
Definition is_sup_graph r f x := least_upper_bound r (range f) x.
Definition is_inf_graph r f x := greatest_lower_bound r (range f) x.
Definition has_sup_graph r  f := has_supremum r (range f).
Definition has_inf_graph r  f := has_infimum r (range f).
Definition sup_graph r f := supremum r (range f).
Definition inf_graph r f := infimum r (range f).
```

Here are the characteristic properties.

```
Lemma is_sup_graph_pr1: forall r f,
  order r -> sub (range f) (substrate r) -> has_sup_graph r f ->
  least_upper_bound r (range f) (sup_graph r f).
Lemma is_inf_graph_pr1: forall r f,
  order r -> sub (range f) (substrate r) -> has_inf_graph r f ->
  greatest_lower_bound r (range f) (inf_graph r f).

Lemma is_sup_graph_pr: forall r f x, order r -> sub (range f) (substrate r) ->
  fgraph f ->
  is_sup_graph r f x = (inc x (substrate r)
   & (forall a, inc a (domain f) -> gle r (V a f) x)
   & forall z, inc z (substrate r) -> (forall a, inc a (domain f)
     -> gle r (V a f) z) -> gle r x z).
Lemma is_inf_graph_pr: forall r f x, order r -> sub (range f) (substrate r) ->
  fgraph f ->
  is_inf_graph r f x = (inc x (substrate r)
   & (forall a, inc a (domain f) -> gle r x (V a f))
   & forall z, inc z (substrate r) -> (forall a, inc a (domain f)
     -> gle r z (V a f)) -> gle r z x).
```

Assume that $A \subset E$ is a set that has an infimum and a supremum. If A is empty, we know that these elements are the least and greatest elements; otherwise, if $y \in A$ we have $\inf A \leq y \leq \sup A$, hence $\inf A \leq \sup A$. This is Proposition 4 [3, p. 142].

```
Theorem compare_inf_sup1: forall r A, order r -> sub A (substrate r) ->
  has_supremum r A -> has_infimum r A ->
  A = emptyset ->
  (greatest_element r (infimum r A) &  least_element r (supremum r A)).
Theorem compare_inf_sup2: forall r A, order r -> sub A (ssetubstrate r) ->
  has_supremum r A -> has_infimum r A ->
  nonempty A -> gle r (infimum r A) (supremum r A).
```

Proposition 5 [3, p. 142] says that sup is increasing and inf is decreasing (as a function from $\mathfrak{P}(E)$ into E, where $\mathfrak{P}$ is ordered by inclusion). Of course, these are only partial functions. As a corollary, consider a family $(x_\iota)_{\iota\in I}$ and $J \subset I$; we have $\sup_{\iota\in J} x_\iota \leq \sup_{\iota\in I} x_\iota$ if both quantities are defined. Note that the first term is the supremum of the restriction of the family to J.

```
Theorem sup_increasing: forall r A B, order r -> sub A (substrate r) ->
  sub B (substrate r) -> sub A B ->
  has_supremum r A -> has_supremum r B ->
  gle r (supremum r A) (supremum r B).
Theorem inf_decreasing: forall r A B, order r -> sub A (substrate r) ->
  sub B (substrate r) -> sub A B ->
  has_infimum r A -> has_infimum r B ->
  gge r (infimum r A) (infimum r B).
Lemma sup_increasing1: forall r f j,
  order r -> fgraph f -> sub (range f) (substrate r) -> sub j (domain f) ->
  has_sup_graph r f  -> has_sup_graph r (restr f j) ->
  gle r (sup_graph r (restr f j)) (sup_graph r f).
Lemma inf_decreasing1: forall r f j,
  order r -> fgraph f -> sub (range f) (substrate r) -> sub j (domain f) ->
  has_inf_graph r f  -> has_inf_graph r (restr f j) ->
  gge r (inf_graph r (restr f j)) (inf_graph r f).
```

Proposition 6 [3, p. 143] says that if f and g are two functions of type $\mathrm{F} \to \mathrm{E}$, if $f(x) \leq g(x)$ for all $x \in \mathrm{F}$ then $\sup f \leq \sup g$, provided that both quantities are defined. In fact, it is stated as: if for all $\iota \in \mathrm{I}$ we have $x_\iota \leq y_\iota$, then $\sup_{\iota\in\mathrm{I}} x_\iota \leq \sup_{\iota\in\mathrm{I}} y_\iota$, and $\inf_{\iota\in\mathrm{I}} x_\iota \leq \inf_{\iota\in\mathrm{I}} y_\iota$.

```
Lemma sup_increasing2: forall r f f',
  order r -> fgraph f ->  fgraph f' ->  domain f = domain f' ->
  sub (range f) (substrate r) -> sub (range f') (substrate r) ->
  has_sup_graph r f -> has_sup_graph r f' ->
  (forall x , inc x (domain f) -> gle r (V x f) (V x f')) ->
  gle r (sup_graph r f) (sup_graph r f').
Lemma inf_increasing2: forall r f f',
  order r -> fgraph f ->  fgraph f' ->  domain f = domain f' ->
  sub (range f) (substrate r) -> sub (range f') (substrate r) ->
  has_inf_graph r f  ->  has_inf_graph r f' ->
  (forall x , inc x (domain f) -> gle r (V x f) (V x f')) ->
  gle r (inf_graph r f) (inf_graph r f').
```

Proposition 7 [3, p. 143] is the following. Consider a family $(x_\iota)_{\iota\in\mathrm{I}}$, and let $(\mathrm{J}_\lambda)_{\lambda\in\mathrm{L}}$ be a covering of I. The family $(x_\iota)_{\iota\in\mathrm{J}_\lambda}$ is the restriction of (x_ι) to J_λ; we assume that it has a supremum $\sup_{\iota\in\mathrm{J}_\lambda} x_\iota$, and we consider the family $(\sup_{\iota\in\mathrm{J}_\lambda} x_\iota)_{\lambda\in\mathrm{L}}$. This family has a supremum if and only if $(x_\iota)_{\iota\in\mathrm{I}}$ has one, and the values are the same; the second equality in (1) is true under similar conditions.

$$\sup_{\iota\in\mathrm{I}} x_\iota = \sup_{\lambda\in\mathrm{L}} \Big(\sup_{\iota\in\mathrm{J}_\lambda} x_\iota\Big), \qquad \inf_{\iota\in\mathrm{I}} x_\iota = \inf_{\lambda\in\mathrm{L}} \Big(\inf_{\iota\in\mathrm{J}_\lambda} x_\iota\Big). \tag{1}$$

The first lemma here says that if x is a least upper bound for one family, it is also the least upper bound for the other one. Finally, since the supremum is a least upper bound, we get the result by uniqueness.

```
Lemma sup_distributive: forall r f c x, (* 48 *)
  order r -> fgraph f ->  sub (range f) (substrate r) ->
  covering c (domain f) ->
  (forall l, inc l (domain c) -> has_sup_graph r (restr f (V l c))) ->
  is_sup_graph r f x =
  is_sup_graph r (L (domain c) (fun l => sup_graph r (restr f (V l c)))) x.
Lemma inf_distributive: forall r f c x,  (* 48 *)
  order r -> fgraph f ->  sub (range f) (substrate r) ->
  covering c (domain f) ->
  (forall l, inc l (domain c) -> has_inf_graph r (restr f (V l c))) ->
  is_inf_graph r f x =
  is_inf_graph r (L (domain c) (fun l => inf_graph r (restr f (V l c)))) x.
Lemma sup_distributive1: forall r f c,
  order r -> fgraph f ->  sub (range f) (substrate r) ->
  covering c (domain f) ->
  (forall l, inc l (domain c) -> has_sup_graph r (restr f (V l c))) ->
  has_sup_graph r f =
  has_sup_graph r (L (domain c) (fun l => sup_graph r (restr f (V l c)))) .
Lemma inf_distributive1: forall r f c,
  order r -> fgraph f ->  sub (range f) (substrate r) ->
  covering c (domain f) ->
  (forall l, inc l (domain c) -> has_inf_graph r (restr f (V l c))) ->
  has_inf_graph r f =
  has_inf_graph r (L (domain c) (fun l => inf_graph r (restr f (V l c)))) .
```

```
Theorem sup_distributive2: forall r f c, (* 19 *)
  order r -> fgraph f ->  sub (range f) (substrate r) ->
  covering c (domain f) ->
  (forall l, inc l (domain c) -> has_sup_graph r (restr f (V l c))) ->
  ((has_sup_graph r f =
    has_sup_graph r (L (domain c) (fun l => sup_graph r (restr f (V l c))))) &
  (has_sup_graph r f -> sup_graph r f =
    sup_graph r (L (domain c) (fun l => sup_graph r (restr f (V l c)))))).

Theorem inf_distributive2: forall r f c,  (* 19 *)
  order r -> fgraph f ->  sub (range f) (substrate r) ->
  covering c (domain f) ->
  (forall l, inc l (domain c) -> has_inf_graph r (restr f (V l c))) ->
  ((has_inf_graph r f =
    has_inf_graph r (L (domain c) (fun l => inf_graph r (restr f (V l c))))) &
  (has_inf_graph r f -> inf_graph r f =
    inf_graph r (L (domain c) (fun l => inf_graph r (restr f (V l c)))))).
```

¶ Corollary. Let $(x_{\lambda\mu})_{(\lambda,\mu)\in L\times M}$ be a double family of elements of an ordered set E such that for each $\mu \in M$ the family $(x_{\lambda\mu})_{\lambda\in L}$ has a least upper bound in E. This family is the restriction of the double family to $L \times \{\mu\}$. For the double family to have a least upper bound in E it is necessary and sufficient that $(\sup_{\lambda\in L} x_{\lambda\mu})_{\mu\in M}$ has a least upper bound, and the bounds are the same. The second equality in (2) holds under similar conditions.

$$\sup_{(\lambda,\mu)\in L\times M} x_{\lambda\mu} = \sup_{\mu\in M}\big(\sup_{\lambda\in L} x_{\lambda\mu}\big), \qquad \inf_{(\lambda,\mu)\in L\times M} x_{\lambda\mu} = \inf_{\mu\in M}\big(\inf_{\lambda\in L} x_{\lambda\mu}\big). \tag{2}$$

```
Definition partial_fun f x m := restr f (product x (singleton m)).

Lemma sup_distributive3: forall r f x y,
  order r -> fgraph f ->  sub (range f) (substrate r) ->
  domain f = product x y ->
  (forall m, inc m y -> has_sup_graph r (partial_fun f x m)) ->
  ((has_sup_graph r f =
    has_sup_graph r (L y (fun m => sup_graph r  (partial_fun f x m)))) &
  (has_sup_graph r f -> sup_graph r f =
    sup_graph r (L y (fun m => sup_graph r (partial_fun f x m))))).
Lemma inf_distributive3: forall r f x y,
  order r -> fgraph f ->  sub (range f) (substrate r) ->
  domain f = product x y ->
  (forall m, inc m y -> has_inf_graph r (partial_fun f x m)) ->
  ((has_inf_graph r f =
    has_inf_graph r (L y (fun m => inf_graph r  (partial_fun f x m)))) &
  (has_inf_graph r f -> inf_graph r f =
    inf_graph r (L y (fun m => inf_graph r (partial_fun f x m))))).
```

Proposition 8 [3, p. 144] says that if we have a family of ordered sets E_ι, a subset A of $\prod E_\iota$, and if $A_\iota = \mathrm{pr}_\iota A$, then A has a least upper bound of the form $(x_\iota)_\iota$ if and only if each A_ι has one, and there is equality; a similar property holds for the greatest lower bound.

$$\sup A = (\sup A_\iota)_{\iota\in I} = \big(\sup_{x\in A} \mathrm{pr}_\iota x\big)_{\iota\in I}, \qquad \inf A = (\inf A_\iota)_{\iota\in I} = \big(\inf_{x\in A} \mathrm{pr}_\iota x\big)_{\iota\in I}.$$

```
Theorem sup_in_product: forall g A, (* 93 *)
  let f := fam_of_substrates g in
```

```
  let Ai:= fun i => (image_by_fun (pr_i f i) A) in
    let has_sup :=
      forall i, inc i (domain g) -> has_supremum (V i g) (Ai i) in
    product_order_axioms g -> sub A (productb f) ->
    ((has_sup = has_supremum (product_order g) A) &
      (has_sup -> supremum (product_order g) A =
        L (domain g) (fun i => supremum (V i g) (Ai i)))).
Theorem inf_in_product: forall g A,  (* 93 *)
  let f := fam_of_substrates g in
  let Ai:= fun i => (image_by_fun (pr_i f i) A) in
    let has_inf :=
      forall i, inc i (domain g) -> has_infimum (V i g) (Ai i) in
    product_order_axioms g -> sub A (productb f) ->
    ((has_inf = has_infimum (product_order g) A) &
      (has_inf -> infimum (product_order g) A =
        L (domain g) (fun i => infimum (V i g) (Ai i)))).
```

Proposition 9 [3, p. 144] assumes that E is an ordered set, F is a subset of E and A is a subset of F. It can happen that one of $\sup_E A$ and $\sup_F A$ exists, but not the other; they may be unequal. If the objects exist we have

$$\sup_E A \le \sup_F A, \qquad \inf_E \ge \inf_F A \qquad (F \subset E).$$

If $\sup_E A$ exists and is in F, it is $\sup_F A$.

```
Theorem sup_induced1: forall r A F, order r -> sub F (substrate r) -> sub A F ->
  has_supremum r A -> has_supremum (induced_order r F) A ->
  gle r (supremum r A) (supremum (induced_order r F) A).
Theorem inf_induced1: forall r A F, order r -> sub F (substrate r) -> sub A F ->
  has_infimum r A -> has_infimum (induced_order r F) A ->
  gge r (infimum r A) (infimum (induced_order r F) A).
Theorem sup_induced2: forall r A F, order r -> sub F (substrate r) -> sub A F ->
  has_supremum r A -> inc (supremum r A)  F ->
  (has_supremum (induced_order r F) A &
    supremum r A = supremum (induced_order r F) A).
Theorem inf_induced2: forall r A F, order r -> sub F (substrate r) -> sub A F ->
  has_infimum r A -> inc (infimum r A)  F ->
  (has_infimum (induced_order r F) A &
    infimum r A = infimum (induced_order r F) A).
```

2.10 Directed sets

An ordered set is said left or right *directed* if every doubleton is bounded (above or below).

```
Definition right_directed r :=
  order r & forall x, forall y, inc x (substrate r) -> inc y (substrate r) ->
    bounded_above r (doubleton x y).
Definition left_directed r :=
  order r & forall x, forall y, inc x (substrate r) -> inc y (substrate r) ->
    bounded_below r (doubleton x y).
```

We rewrite the definition as: for all x and y there is a z such that $x \le z$ and $y \le z$. A set that has a greatest element is right directed. A product of directed sets is directed. A cofinal set of a directed set is directed for the induced order.

```
Lemma right_directed_pr: forall r,
  right_directed r = (order r &
    forall x, forall y, inc x (substrate r) -> inc y (substrate r) -> exists z,
      inc z (substrate r) & gle r x z  & gle r y z).
Lemma left_directed_pr: forall r,
  left_directed r = (order r &
    forall x, forall y, inc x (substrate r) -> inc y (substrate r) -> exists z,
      inc z (substrate r) & gle r z x  & gle r z y).

Lemma greatest_right_directed : forall r, order r ->
  (exists a, greatest_element r a )  -> right_directed r .
Lemma least_left_directed : forall r, order r ->
  (exists a, least_element r a )  -> left_directed r.
Lemma opposite_right_directed : forall r, is_graph r ->
  right_directed r = left_directed(opposite_order r).
Lemma opposite_left_directed : forall r, is_graph r ->
  left_directed r = right_directed(opposite_order r).
Lemma product_right_directed: forall g, (* 20 *)
  product_order_axioms g ->
  (forall i, inc i (domain g) -> right_directed (V i g))
  -> right_directed (product_order g).
Lemma product_left_directed: forall g,  (* 20 *)
  product_order_axioms g ->
  (forall i, inc i (domain g) -> left_directed (V i g))
  -> left_directed (product_order g).
Lemma cofinal_right_directed: forall r A,
  right_directed r -> cofinal_set r A -> right_directed (induced_order r A).
Lemma coinitial_left_directed: forall r A,
  left_directed r -> coinitial_set r A -> left_directed (induced_order r A).
```

Proposition 10 [3, p. 145] says that in a right directed set, a maximal element is the greatest element.

```
Theorem right_directed_maximal: forall r x,
  right_directed r -> maximal_element r x -> greatest_element r x.
Theorem left_directed_mimimal: forall r x,
  left_directed r -> minimal_element r x -> least_element r x.
```

2.11 Lattices

A *lattice* is an ordered set on which each pair has a least upper bound and a greatest lower bound.

```
Definition lattice r :=  order  r &
  forall x, forall y, inc x (substrate r) -> inc y (substrate r) ->
    (has_supremum r (doubleton x y) & has_infimum r (doubleton x y)).

Lemma lattice_sup_pr: forall r a b,
  lattice r -> inc a (substrate r) -> inc b (substrate r) ->
  (gle r a (sup r a b) &  gle r b (sup r a b) &
    forall z, gle r a -> gle r b z -> gle r (sup r a b) z).
Lemma lattice_inf_pr: forall r a b,
  lattice r -> inc a (substrate r) -> inc b (substrate r) ->
  (gle r (inf r a b) a &  gle r (inf r a b) b&
    forall z, gle r z a ->  gle r z b -> gle r z (inf r a b)).
```

The powerset is a lattice. In fact each set has a supremum and an infimum.

```
Lemma inf_inclusion: forall A x y, sub x A -> sub y A  ->
  greatest_lower_bound (inclusion_order A) (doubleton x y) (intersection2 x y).
Lemma sup_inclusion: forall A x y, sub x A -> sub y A  ->
  least_upper_bound (inclusion_order A) (doubleton x y) (union2 x y).
Lemma powerset_lattice: forall A, lattice (inclusion_order A).
```

The product of lattices is a lattice. This is an easy consequence of Proposition 8, and the fact that $\mathrm{pr}_\iota A$ is a doubleton if A is a doubleton. A lattice is a directed set.

```
Lemma product_lattice: forall g,
  product_order_axioms g ->
  (forall i, inc i (domain g) -> lattice (V i g))
  -> lattice (product_order g).  (* 37 *)
Lemma lattice_directed: forall r,
  lattice r -> (right_directed r & left_directed r).
```

Other examples. The set of integers, with the order "x divides y" is a lattice. The set of subgroups of a group (ordered by inclusion) is a lattice. The set of topologies on a set is a lattice. The set of real functions on an interval is a lattice. The opposite of a lattice is a lattice.

```
Lemma lattice_inverse: forall r, lattice r -> lattice (opposite_order r).
```

2.12 Totally ordered sets

Two elements of a preordered set E are said comparable if the relation "$x \leq y$ or $y \leq x$" is true. A set E is said to be *totally ordered* if is is ordered and if any two elements of E are comparable.

```
Definition total_order r :=
  order r & forall x y, inc x (substrate r) -> inc y (substrate r) ->
    (gle r x y \/ gge r x y).
```

An order satisfies $G \circ G = G$ and $G \cap G^{-1} = \Delta_E$. It is total if moreover $G \cup G^{-1} = E \times E$. We have $x < y$ or $x > y$ or $x = y$. We have $x < y$ or $y \leq x$. A subset of a totally ordered set is totally ordered. A small set is totally ordered. The opposite of a totally ordered set is totally ordered.

```
Lemma total_order_pr: forall r,
  total_order r =
    (compose_graph r r = r &
    intersection2 r (opposite_order r) = identity_g (substrate r)&
    union2 r (opposite_order r) =  coarse (substrate r)).  (* 22 *)
Lemma total_order_pr1 : forall r x y,
  total_order r ->  inc x (substrate r) -> inc y (substrate r) ->
    (glt r x y \/ ggt r x y \/ x = y).
Lemma total_order_pr2 : forall r x y,
  total_order r ->  inc x (substrate r) -> inc y (substrate r) ->
    (glt r x y \/ gle r y x).

Lemma total_order_sub : forall r x,
```

```
  total_order r ->  sub x (substrate r) -> total_order (induced_order r x).
Lemma total_order_conterexample:
  ~ (total_order (inclusion_order two_points)).
Lemma total_order_opposite: forall r,
 total_order r -> total_order (opposite_order r).
```

If $x \leq y$, then $\sup(x, y) = y$ and $\inf(x, y) = x$, hence a totally ordered set is a lattice.

```
Lemma sup_comparable: forall r x y, gle r x y ->
  order r -> least_upper_bound r (doubleton x y) y.
Lemma inf_comparable: forall r x y, gle r x y ->
  order r -> greatest_lower_bound r (doubleton x y) x.
Lemma sup_comparable1: forall r x y, order r -> gle r x y -> sup r x y = y.
Lemma inf_comparable1: forall r x y, order r -> gle r x y -> inf r x y = x.
Lemma total_order_lattice: forall r,  total_order r -> lattice r.
Lemma total_order_directed: forall r,
  total_order r -> (right_directed r & left_directed r).
```

Proposition 11 [3, p. 147] says that if f is strictly monotone and the ordering on the source is total, then f is injective. If f is strictly increasing, it is a morphism (an isomorphism onto the image). We first show that if f is strictly increasing, then it is increasing.

```
Lemma increasing_fun_from_strict: forall f r r',
  strict_increasing_fun f r r' -> increasing_fun f r r'.
Lemma decreasing_fun_from_strict: forall f r r',
  strict_decreasing_fun f r r' -> decreasing_fun f r r'.
Theorem total_order_monotone_injective: forall f r r',
  total_order r -> strict_monotone_fun f r r' -> injective f.
Theorem total_order_increasing_morphism: forall f r r',
 total_order r -> strict_increasing_fun f r r' -> order_morphism f r r'.
```

Proposition 12 [3, p. 147] says that in a totally ordered set E, an element x is the least upper bound of a subset X if and only if it is an upper bound and, for all $y < x$, there is a $z \in X$ such that $y < z$ and $z \leq x$.

```
Theorem sup_in_total_order:forall r X x, total_order r -> sub X (substrate r)->
  least_upper_bound r X x = (upper_bound r X x &
    (forall y, glt r y x -> exists z, inc z X & glt r y z &  gle r z x)).
Theorem inf_in_total_order:forall r X x, total_order r -> sub X (substrate r)->
  greatest_lower_bound r X x = (lower_bound r X x &
    (forall y, glt r x y -> exists z, inc z X & glt r z y &  gle r x z)).
```

2.13 Intervals

There are many definitions of an *interval*. The set of all x such that $a \leq x \leq b$ is called the closed interval and denoted $[a, b]$; the set of all x such that $a < x < b$ is called the open interval and denoted $]a, b[$; intervals can be semi open. One can drop one of the conditions, for instance the set of all x such that $x < b$ is denoted by $]\leftarrow, b[$, this is an unbounded interval.

The letters o, c, u stand for open, close, and unbounded.

```
Definition interval_oo r a b := Zo(substrate r)(fun z => glt r a z & glt r z b).
Definition interval_oc r a b := Zo(substrate r)(fun z => glt r a z & gle r z b).
```

```
Definition interval_ou r a  := Zo (substrate r) (fun z  => glt r a z).
Definition interval_co r a b := Zo(substrate r)(fun z => gle r a z & glt r z b).
Definition interval_cc r a b := Zo(substrate r)(fun z => gle r a z & gle r z b).
Definition interval_cu r a  := Zo (substrate r) (fun z  => gle r a z).
Definition interval_uo r b := Zo (substrate r) (fun z  =>  glt r z b).
Definition interval_uc r b := Zo (substrate r) (fun z => gle r  z b).
Definition interval_uu r := Zo (substrate r) (fun z => True).


Definition is_closed_interval r x := exists a, exists b,
  inc a (substrate r) & inc b (substrate r) & gle r a b & x = interval_cc r a b.
Definition is_open_interval r x := exists a, exists b,
  inc a (substrate r) & inc b (substrate r) & gle r a b & x = interval_oo r a b.
Definition is_semi_open_interval r x := exists a, exists b,
  inc a (substrate r) & inc b (substrate r)  & gle r a b &
  (x = interval_oc r a b \/ x = interval_co r a b).
Definition is_bounded_interval r x := is_closed_interval r x \/
  is_open_interval r x \/ is_semi_open_interval r x.
Definition is_left_unbounded_interval r x :=
  exists b, inc b (substrate r) & (x = interval_uc r b \/ x = interval_uo r b).
Definition is_right_unbounded_interval r x :=
  exists a, inc a (substrate r) & (x = interval_cu r a \/ x = interval_ou r a).
Definition is_unbounded_interval r x :=
  is_left_unbounded_interval r x \/ is_right_unbounded_interval r x  \/
  x = interval_uu r.
Definition is_interval r x :=
  is_bounded_interval r x  \/ is_unbounded_interval r x.
```

A non-empty interval $[a,b]$ has a least and greatest elements, which are a and b respectively.

```
Lemma the_least_interval: forall r a b, order r ->
  gle r a b ->   the_least_element (induced_order r (interval_cc r a b)) = a.
Lemma the_greatest_interval: forall r a b, order r ->
  gle r a b -> the_greatest_element (induced_order r (interval_cc r a b)) = b.
```

A closed interval is never empty; however $[a,a[$, $]a,a]$ and $]a,a[$ are empty.

```
Lemma nonempty_closed_interval: forall r x,
  order r -> is_closed_interval r x -> nonempty x.
Lemma singleton_interval: forall r a,
  order r -> inc a (substrate r) -> is_singleton (interval_cc r a a).
Lemma empty_interval: forall r a,
  order r -> inc a (substrate r) ->
  (empty (interval_co r a a) &  empty (interval_oc r a a)&
    empty (interval_oo r a a)).
```

The only non trivial result here is Proposition 13 [3, p. 148] that says that, in a lattice, the intersection of two intervals is an interval. We start with a short proof.

Let's say that an interval is of type L if it is left unbounded, of type R if it is right unbounded (the interval U $=]\leftarrow,\rightarrow[$ is of both types, and $U \cap X = X$ for any interval X). Obviously, each interval is the intersection of two intervals of type L and R (if the interval is unbounded, consider intersection with U). The intersection of two intervals is thus of the form $(L_1 \cap R_1) \cap (L_2 \cap R_2) = (L_1 \cap L_2) \cap (R_1 \cap R_2)$. This is of the form $L_3 \cap R_3$.

```
Definition is_lu_interval r x :=
   x = interval_uu r \/ is_left_unbounded_interval r x.
Definition is_ru_interval r x :=
   x= interval_uu r \/ is_right_unbounded_interval r x.
Lemma intersection4: forall A B C D,
  intersection2 (intersection2 A B) (intersection2 C D)
  = intersection2 (intersection2 A C) (intersection2 B D).
Lemma intersection_i1: forall r x,
  is_interval r x -> intersection2 x (interval_uu r) = x. (* 18 *)
Lemma intersection_i2: forall r x,
  is_interval r x ->
  (exists u, exists v, is_lu_interval r u & is_ru_interval r v &
    intersection2 u v = x). (* 40 *)
Lemma intersection_i3: forall r x y, lattice r ->
  is_left_unbounded_interval r x -> is_left_unbounded_interval r y ->
  is_left_unbounded_interval r (intersection2 x y). (* 38 *)
Theorem intersection_interval: forall r x y,
  lattice r -> is_interval r x -> is_interval r y ->
  is_interval r (intersection2 x y). (* 49 *)
```

The total size of the proof of the theorem is 150 lines, including the tactics designed for this theorem. Originally, its size was 841 lines, reduced to 615 by using adequate tactics. Details can be foun in section 12.5.

Chapter 3

Well-ordered sets

This chapter defines the notion of a well-ordering, and the lexicographic ordering (we compare two sequences x_i and y_i according to the least index j such that $x_j \neq y_j$). We show Zermelo's theorem (there exists a well-ordering) and Zorn's lemma (every inductive ordered set has a maximal element). These theorems are equivalent to the axiom of choice, thus non-constructive. Ordinals (first, second, third, etc.) are introduced in Chapter 8, cardinals (one, two, three, etc.) are defined in the next chapter. To each finite ordinal corresponds a unique cardinal (see Chapter 5 for the definition of "finite"). The "etc." in the sentence above suggests that there is a unique way to add a new term to the sequence (for instance fourth and four). This is called "induction" in the case of finite cardinals; it is called "transfinite induction" in the case of well-ordered sets.

3.1 Segments of a well-ordered set

A relation $R\{\!\{x, y\}\!\}$ is said to be a *well-ordering relation* between x and y if R is an order relation between x and y and if for each non-empty set E on which $R\{\!\{x, y\}\!\}$ induces an order relation, E, ordered by this relation, has a least element. A set E ordered by an ordering Γ is said to be *well-ordered* if the relation $y \in \Gamma\langle x\rangle$ is a well ordering between x and y; Γ is then said to be a well-ordering on E.

These definitions are a bit complicated. We first rewrite "E, ordered by this relation, has a least element" as "the ordering induced by this relation on E has a least element", or simply "the graph of R on E has a least element". Assume that R is an order relation; then $R\{\!\{x, y\}\!\}$ induces an order relation on E provided that it is reflexive on E, thus, whenever $x \in$ E, we have $R\{\!\{x, x\}\!\}$. Let's consider the second definition. Recall that for Bourbaki, Γ is a correspondence and $\Gamma\langle x\rangle$ is (by abuse of notations) the image of the singleton $\{x\}$ under the correspondence. Let G be the graph of Γ. Then $y \in \Gamma\langle x\rangle$ is the same as $(x, y) \in$ G. It implies that x and y belong to E (conversely, if $x \in$ E, then $(x, x) \in$ G). Denote this relation by $R\{\!\{x, y\}\!\}$. The condition "A is a set on which $R\{\!\{x, y\}\!\}$ induces an order relation" is equivalent to A $\subset$ E. The condition "A, ordered by this relation, has a least element", written as "the graph of R on A has a least element" becomes "the order induced by G on A has a least element". Here are the two definitions

```
Definition worder_r r :=
  order_r r & forall x, (forall a, inc a x -> r a a) -> nonempty x ->
    exists y, least_element (graph_on r x) y.
```

```
Definition worder r :=
  order r & forall x, sub x (substrate x) -> nonempty x ->
    exists y, least_element (induced_order r x) y.

Lemma worder_is_order: forall r, worder r -> order r.
Lemma wordering_pr: forall r x, worder_r r ->
  (forall a, inc a x -> r a a)  ->
  (substrate (graph_on r x) = x & worder (graph_on r x)).
```

Bourbaki notes that a totally ordered set with two elements is well-ordered. In fact, it contains a and b such that $a < b$, so that the graph is the set with three elements (a,a), (a,b) and (b,b). All total orders on sets with two elements are isomorphic. We consider here the case where a and b are the elements of the doubleton *two_points*.

```
Definition canonical_doubleton_order :=
  union2 (doubleton (J TPa  TPa) (J TPb TPb)) (singleton (J TPa TPb)).
Lemma canonical_doubleton_order_pr: forall x y,
  related canonical_doubleton_order x y =
  ( (x= TPa & y = TPa) \/  (x= TPb & y = TPb) \/  (x= TPa & y = TPb)).

Lemma one_not_zero: (singleton emptyset) <> emptyset.
Lemma substrate_canonical_doubleton_order:
  substrate canonical_doubleton_order = two_points.
Lemma worder_canonical_doubleton_order:
  worder canonical_doubleton_order. (* 31 *)
```

By considering the least element of the doubleton $\{x, y\}$, one deduces that a well-ordering is a total ordering. Every subset that is bounded above has a least upper bound. A subset of a well-ordered set is well-ordered. Adjoining a greatest element to a well-ordered set gives a well-ordered set. A nonempty well-ordered set has a least element.

```
Lemma worder_total: forall r, worder r -> total_order r.
Lemma worder_hassup: forall r A, worder r -> sub A (substrate r) ->
  bounded_above r A -> has_supremum r A.
Lemma induced_trans: forall r x y,
  order r -> sub x y-> sub y (substrate r) ->
  induced_order r x = induced_order (induced_order r y) x.
Lemma worder_restriction: forall r A, worder r -> sub A (substrate r) ->
  worder (induced_order r A).
Lemma worder_adjoin_greatest: forall r a, worder r-> ~ (inc a (substrate r))
  -> worder (order_with_greatest r a). (* 22 *)
Lemma worder_least: forall r, worder r -> nonempty (substrate  r) ->
  exists y, least_element r y.
```

Some useful small lemmas. If $\leq$ is a relation on E, then $x < y$ means $x \leq y$ and $x \neq y$. Thus $x < x$ is false and $x < y$ implies that x and y are in E. If we have a total order on E, if $x \in$ E and $y \in$ E then $x < y$ or $y \leq x$; in the case of an order, one of these relations is false.

```
Lemma inc_lt1_substrate: forall r x y, glt r x y -> inc x (substrate r).
Lemma inc_lt2_substrate: forall r x y, glt r x y -> inc y (substrate r).
Lemma not_le_gt: forall r x y, order r -> gle r x y -> glt r y x -> False.
Lemma not_lt_self: forall  r x, glt r x x -> False.
```

Consider two strictly increasing functions f and g, with the same source and range. These functions are equal provided that the source is well-ordered (on **Z**, the mapping $x \mapsto$

$x+1$ is a strictly increasing bijection, different from the identity, so that the condition on the order is necessary). Assume $f \neq g$, and let x be the smallest element such that $f(x) \neq g(x)$. Since $f(x)$ is in the range of g, there is z such that $f(x) = g(z)$. If $z < x$ then $f(z) < f(x) = g(z)$, contradicting the definition of x. Since the source is totally ordered, we get $x < z$ (since $x \neq z$), hence $g(x) < g(z) = f(x)$. Since $g(x)$ is in the range of f, the same argument gives $f(x) < g(x)$, absurd.

```
Lemma strict_increasing_extens: forall f g r r',
  strict_increasing_fun f r r'-> strict_increasing_fun g r r' -> worder r ->
  range (graph f) = range (graph g) ->
  f = g. (* 42 *)
```

A *segment* S in an ordered set E is such that, if $x \in S$ and $y \in E$ and $y \leq x$, then $y \in S$. Note that if E is the substrate of $\leq$ then $y \leq x$ implies $y \in E$. An interval of the form $]\leftarrow, x[$ is a segment; it will be denoted by S_x, and is called the *segment with endpoint x*. The interval $]\leftarrow, x]$ will be called the closed segment.

```
Definition is_segment r s :=
  sub s (substrate r) & forall x y, inc x s -> gle r y x -> inc y s.
Definition segment r x := interval_uo r x.
Definition segment_c r x := interval_uc r x.
```

The 20-some following lemmas sometimes assume that we have an order $\leq$ on E, and that S is a segment. We assume $x' \in E$. They state that if $x \in S$ and $y < x$ then $y \in S$. If $y \in]\leftarrow, x[$ then $y < x$. We have $S \subset E$. We have $]\leftarrow, x[\subset E$, $]\leftarrow, x] \subset E$. We have $x \notin]\leftarrow, x[$ and $x' \in]\leftarrow, x']$. We have $y < x'$ if and only if $y \in]\leftarrow, x'[$ and $y \leq x'$ if and only if $y \in]\leftarrow, x']$. We have $]\leftarrow, x'] =]\leftarrow, x'[\cup \{x'\}$. We have $]\leftarrow, x[_F \subset F$, where the notation W_F means that we restrict $\leq$ to F. The empty set and E are segments. Intersections and unions of segments are segments. If S' is a segment for the order induced on S, then S' is a segment. Finally $]\leftarrow, x'[$ is a segment.

```
Lemma lt_in_segment: forall r s x y,
  is_segment r s ->  inc x s -> glt r y x -> inc y s.
Lemma inc_segment: forall r x y, inc y (segment r x) -> glt r y x.
Lemma not_in_segment : forall r x,  inc x (segment r x) -> False.
Lemma sub_segment: forall r x, sub (segment r x) (substrate r).
Lemma sub_segment1: forall r s,  is_segment r s -> sub s (substrate r).
Lemma sub_segment2: forall r x y,
  sub (segment (induced_order r x) y) x.
Lemma segment_inc: forall r x y,
  glt r y x -> inc y (segment r x).
Lemma segment_rw: forall r x y,
  inc y (segment r x) = glt r y x.
Lemma segmentc_rw: forall r x y, inc x (substrate r) ->
  inc y (segment_c r x) = gle r y x.
Lemma inc_bound_segmentc: forall r x, order r -> inc x (substrate r) ->
  inc x (segment_c r x).
Lemma sub_segmentc: forall r x, sub (segment_c r x) (substrate r).
Lemma segment_c_pr: forall r x, order r -> inc x (substrate r) ->
  tack_on (segment r x) x = segment_c r x.
Lemma empty_is_segment: forall r, is_segment r emptyset.
Lemma substrate_is_segment: forall r, order r -> is_segment r (substrate r).
Lemma intersection_is_segment: forall r s , order r -> nonempty s ->
```

```
  (forall x, inc x s -> is_segment r x) -> is_segment r (intersection s).
Lemma union_is_segment: forall r s , order r ->
  (forall x, inc x s -> is_segment r x) -> is_segment r (union s).
Lemma unionf_is_segment: forall r j s, order r ->
  (forall x, inc x j -> is_segment r (s x)) -> is_segment r (unionf j s).
Lemma subsegment_is_segment: forall r s s', order r ->
  is_segment r s -> is_segment (induced_order r s) s' -> is_segment r s'.
Lemma segment_is_segment: forall r x, order r ->
  inc x (substrate r) -> is_segment r (segment r x).
```

Proposition 1 [3, p. 149] is the converse of the previous lemma. In a well-ordered set, a segment is either the whole set or of the form S_x. If an ordered set has a least element a, then $S_x = [a, x[$; hence in a well-ordered set E, a segment S is either E, or else E is not empty, has a least element a and $S = [a, x[$.

```
Theorem well_ordered_segment: forall r s, worder r -> is_segment r s ->
  s = substrate r \/ (exists x,  inc x (substrate r) &  s = segment r x). (* 22 *)
Lemma segment_alt: forall r x a, order r -> least_element r a ->
  segment r x = interval_co r a x.
Lemma segment_alt1: forall r s, worder r -> is_segment r s ->
  s = substrate r \/ (exists x, exists a, s = interval_co  r a x).
```

Some useful lemmas. We consider a well-ordered set. If S and S′ are segments, then $S \subset S'$ or $S' \subset S$. If $S \subset S'$, if $x \in S$, the segments with endpoint x in S or S′ coincide. If $x \leq y$, then $S_x \subset S_y$ and $S_x \times S_x \subset S_y \times S_y$. If $z \leq y$ and $y \in S_x$ then $z \in S_x$. The set $]\leftarrow, x]$ is a segment. If S is a segment and $x \in S$, then S_x is the segment with endpoint x for the order induced on S. It is also the segment with endpoint x for the order induced on $]\leftarrow, y]$ or $]\leftarrow, y[$ if $x < y$.

```
Lemma segment_monotone: forall r x y, order r -> gle r x y ->
  sub (segment r x)  (segment r y).
Lemma segment_dichot_sub: forall r x y,
  worder r -> is_segment r x -> is_segment r y ->
 (sub x y \/ sub y x).
Lemma le_in_segment: forall r x y z, order r -> inc x (substrate r) ->
  inc y (segment r x) -> gle r z y -> inc z (segment r x).
Lemma coarse_segment_monotone: forall r x y, order r -> gle r x y ->
  sub (coarse (segment r x)) (coarse (segment r y)).
Lemma tack_on_segment: forall r x,
  order r -> inc x (substrate r) ->
  is_segment r (segment_c r x).
Lemma segment_induced_a: forall r s x,
  order r -> is_segment r s -> inc x s ->
  segment (induced_order r s) x = segment r x.
Lemma segment_induced: forall r x x0, order r ->  glt r x0 x ->
  segment (induced_order r (segment r x)) x0 = segment r x0.
Lemma segment_induced1: forall r x x0, order r ->  glt r x0 x ->
  segment (induced_order r (segment_c r x)) x0 = segment r x0.
```

In a totally ordered set E, the union of all segments is E minus its greatest elements (there is at most one such element). In fact the union is the set of all x for which there is a y such that $x < y$ (remember that $x \leq y$ implies $x \in E$ and $y \in E$).

```
Lemma union_segments: forall r, total_order r ->
  let E := substrate r in
```

```
    let A := union (fun_image E (fun x => (segment r x))) in
      ( (forall x, ~ (greatest_element r x)) -> A = E)
      & (forall x, greatest_element r x -> A = complement E (singleton x)).
```

The function $x \mapsto]\leftarrow, x[$ is stricly increasing , hence injective in a totally ordered set.

```
Lemma segment_monotone1: forall r x y, total_order r ->
  inc x (substrate r) ->  inc y (substrate r) ->
  sub (segment r x)(segment r y) -> gle r x y.
Lemma segment_injective: forall r x y, total_order r ->
  inc x (substrate r) ->  inc y (substrate r) -> segment r x = segment r y ->
  x = y.
Lemma segment_injective1: forall r x y, worder r ->
  inc x (substrate r) ->  inc y (substrate r) -> segment r x = segment r y ->
  x = y.
```

Assume that E is well-ordered. The mapping $x \mapsto S_x$ is a bijection on the set $E^* \setminus \{E\}$ where E^* is the set of all segments of E. This mapping is an order isomorphism. From this, we deduce that $E^* \setminus \{E\}$, hence E^*, is well-ordered. This is Proposition 2 in [3, p. 149].

```
Definition set_of_segments_strict r:=
  fun_image (substrate r) (fun x => (segment r x)).
Definition set_of_segments r:=
  tack_on (set_of_segments_strict r) (substrate r).
Definition  set_of_segments_iso r:=
  BL(segment r) (substrate r) (set_of_segments_strict r).

Lemma inc_set_of_segments: forall r x, worder r ->
  is_segment r x = inc x (set_of_segments r).
Lemma segmentc_insetof: forall r x, worder r -> inc x (substrate r) ->
  inc (segment_c r x) (set_of_segments r).
Lemma segment_insetof: forall r x, worder r -> inc x (substrate r) ->
  inc (segment r x) (set_of_segments r).
Lemma sub_set_of_segments: forall r x, worder r ->
  inc x (set_of_segments r) -> sub x (substrate r).
Lemma set_of_segments_axiom: forall r, worder r ->
  transf_axioms (segment r) (substrate r) (set_of_segments_strict r).
Lemma set_of_segments_iso_bijective: forall r, worder r ->
  bijective (set_of_segments_iso r).
Theorem set_of_segments_iso_isomorphism:forall r, worder r ->
  order_isomorphism (set_of_segments_iso r) r
  (inclusion_suborder (set_of_segments_strict r)).
Theorem set_of_segments_worder: forall r, worder r ->
  worder (inclusion_suborder (set_of_segments r)).  (* 43 *)
```

We state Lemma 1 [3, p. 150]. *Let* $(X_\alpha)_{\alpha\in A}$ *be a family of ordered sets, directed with respect to the relation* $\subset$*. Suppose that, for each pair of indices* (α,β) *such that* $X_\alpha \subset X_\beta$*, the ordering induced on* X_α *by that of* X_β *is identical with the given ordering on* X_α*. Under these conditions there exists a unique ordering on then set* $E = \bigcup_{\alpha\in A} X_\alpha$ *which induces the given ordering on each* X_α. In the lemma that follows, g is the family of orders (G_α) and the resulting order is the union of the family.

```
Definition common_extension_order g h:=
  order h & substrate h = unionf (domain g) (fun a =>  (substrate (V a g))) &
```

```
  (forall a, inc a (domain g) -> V a g = induced_order h (substrate (V a g))).
Definition common_extension_order_axiom g :=
  fgraph g &
  (forall x, inc x (domain g) -> order (V x g)) &
  (forall a b, inc a (domain g) -> inc b (domain g )-> exists c,
    inc c (domain g) & sub (substrate (V a g)) (substrate (V c g))
    & sub (substrate (V b g)) (substrate (V c g))) &
  (forall a b, inc a (domain g) -> inc b (domain g) ->
    sub (substrate (V a g)) (substrate (V b g)) ->
    V a g = induced_order (V b g) (substrate (V a g))).

Lemma order_merge1: forall g,  common_extension_order_axiom g ->
    common_extension_order g (unionb g).  (* 41 *)
Lemma order_merge2: forall g h1 h2,  common_extension_order_axiom g ->
  common_extension_order g h1 ->
  common_extension_order g h2 -> (h1 = h2).
```

We consider now Proposition 3 [3, p. 149]. It says *Let* $(X_\iota)_{\iota\in I}$ *be a family of well-ordered sets such that for each pair of indices* (ι,κ) *one of the sets* X_ι, X_κ *is a segment of the other. Then there exists a unique ordering on the set* $E = \bigcup_{\iota\in I} X_\iota$ *which induces the given ordering on each of the* X_ι. *Endowed with this ordering,* E *is a well-ordered set. Every segment of* X_ι *is a segment of* E*; for each* $x \in X_\iota$*, the segment with endpoint* x *in* X *is equal to the segment with endpoint* x *in* E*; and each segment of* E *is either* E *itself or a segment of one of the* X_ι. Note that if $X_\alpha = X_\beta$, then each set is a segment of the other; the orderings G_α and G_β may differ, in which case (since orders are total) there is a pair (x, y) such that $x < y$ for one order and $y < x$ for the other one, and no compatible order exists on the union. Thus an additional condition is needed (the same one as in the previous lemma). With this definition, the next lemmas become trivial.

```
Definition common_worder_axiom g:=
  fgraph g &
  (forall x, inc x (domain g) -> worder (V x g)) &
  (forall a b, inc a (domain g) -> inc b (domain g) ->
    is_segment (V a g) (substrate (V b g))
    \/ is_segment (V b g) (substrate (V a g))) &
  (forall a b, inc a (domain g) -> inc b (domain g) ->
    sub (substrate (V a g)) (substrate (V b g)) ->
    V a g = induced_order (V b g) (substrate (V a g))).

Lemma order_merge3: forall g,
  common_worder_axiom g -> common_extension_order_axiom g.
Lemma order_merge4: forall g,
  common_worder_axiom g -> common_extension_order g (unionb g).
Lemma order_merge5: forall g h1 h2,  common_worder_axiom g ->
  common_extension_order g h1 ->
  common_extension_order g h2 -> (h1 = h2).

Theorem worder_merge: forall g, common_worder_axiom g ->
 ( (common_extension_order g (unionb g)) &
   worder (unionb g) &
   (forall a x, inc a (domain g) -> is_segment (V a g) x
     -> is_segment (unionb g) x ) &
   (forall a x, inc a (domain g) ->  inc x (substrate (V a g)) ->
     segment (V a g) x = segment (unionb g) x) &
   (forall x, is_segment (unionb g) x ->
     x = substrate (unionb g) \/
```

```
      exists a,  inc a (domain g) & is_segment (V a g) x)). (* 64 *)
```

3.2 The principle of transfinite induction

The next result is Lemma 2 [3, p. 151]. It says that, given a well-ordered set E and a set $\mathfrak{S}$ of segments of E, all segments are in $\mathfrak{S}$ if $\mathfrak{S}$ is stable by union and by adjunction of a greatest element (i.e., if the segment S_x belongs to $\mathfrak{S}$ then $S_x \cup \{x\}$ belongs to $\mathfrak{S}$). As a consequence, the substrate of the order is in $\mathfrak{S}$.

```
Lemma transfinite_principle1: forall  r s,
  worder r ->  (forall x, inc x s -> is_segment r x) ->
  (forall s', sub s' s -> inc (union s') s) ->
  (forall x, inc x (substrate r) -> inc (segment r x) s
    -> inc (segment_c r x) s) ->
  (forall x, is_segment r x -> inc x s).  (* 48 *)
Lemma transfinite_principle2: forall  r s,
  worder r ->  (forall x, inc x s -> is_segment r x) ->
  (forall s', sub s' s -> inc (union s') s) ->
  (forall x, inc x (substrate r) -> inc (segment r x) s
    -> inc (segment_c r x) s) ->
  inc (substrate r) s.
```

Let H(x) be the assumption: "$x \in$ E, and $p(y)$ holds for all $y \in$ E such that $y < x$". Assume that $\leq$ is a well-ordering on E and for all x, H(x) implies $p(x)$; then p is true on E. This is C59 ("Principle of transfinite induction"), [3, p. 151]. Proof 1: the set of segments S such that p is trus en S contains E by the previous lemma. Proof 2: the set of elements not satisfying p has no least element.

```
Theorem transfinite_principle: forall r (p:Set -> Prop),
  worder r ->
  (forall x, inc x (substrate r) ->
       (forall y, inc y (substrate r) -> glt r y x -> p y)
     -> p x)
   ->  forall x, inc x (substrate r) -> p x.
```

We now define a mapping by transfinite induction. We consider a well-ordered set E, a function g, and an element $x \in$ E. Let $g^{(x)}$ denote the restriction of g to $]\leftarrow, x[$ as a surjective function.

```
Definition restriction_to_segment r x g :=
  restriction1 g (segment r x).
Definition restriction_to_segment_axiom r x g :=
  worder r & inc x (substrate r) & is_function g & sub (segment r x) (source g).

Lemma sub_image_target: forall g x, is_function g ->
  sub (image_by_fun g x) (target g).
Lemma rts_function: forall r x g,  restriction_to_segment_axiom r x g  ->
  is_function (restriction_to_segment r x g).
Lemma rts_W: forall r x g a,  restriction_to_segment_axiom r x g  ->
  glt r a x -> W a  (restriction_to_segment r x g) = W a g.
Lemma rts_surjective: forall r x g,  restriction_to_segment_axiom r x g  ->
  surjective(restriction_to_segment r x g).
```

```
Lemma rts_extensionality: forall r s x f g,
  worder r -> inc x (substrate r) -> worder s -> inc x (substrate s) ->
  segment r x = segment s x ->
  is_function f -> sub (segment r x) (source f) ->
  is_function g -> sub (segment s x) (source g) ->
  (forall a , inc a (segment r x) -> W a f = W a g) ->
  restriction_to_segment r x f = restriction_to_segment s x g.
```

We can now state Criterion C60 (Definition of a mapping by transfinite induction) [3, p. 151]: *Let u be a letter, $\mathrm{T}\{u\}$ 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 $x \in \mathrm{E}$ we have $f(x) = \mathrm{T}\{f^{(x)}\}$. Furthermore the set* U *and the mapping f are uniquely determined by these conditions.*

We shall denote by $f_{\mathrm{S}}^{(x)}$ the restriction of f to the segment x for the ordering S (more precisely, the ordering induced on S by the given ordering). Let's denote by $\mathscr{I}(\mathrm{S},\mathrm{T},f)$ the property of the criterion, namely, that f is a surjective function defined on S such that $f(x) = \mathrm{T}\{f_{\mathrm{S}}^{(x)}\}$ for every $x \in \mathrm{S}$.

Assume that $\mathscr{I}(\mathrm{E},\mathrm{T},f)$ and $\mathscr{I}(\mathrm{E},\mathrm{T},f')$ are true. Assume $x \in \mathrm{E}$, f and f' agree on S_x. Then $f_{\mathrm{E}}^{(x)} = f_{\mathrm{E}}^{\prime(x)}$ (the functions have the same target, because they are surjective, and take the same value). By assumption, $\mathrm{T}\{f_{\mathrm{E}}^{(x)}\} = \mathrm{T}\{f_{\mathrm{E}}^{\prime(x)}\}$, so that f and f' agree on $\mathrm{S}_x \cup \{x\}$ (this *transfinite_unique1*). We consider the set of all segments $\mathfrak{S}$ on which f and f' agree. This set is clearly stable by union, and we have seen that, if it contains S_x, then it contains $\mathrm{S}_x \cup \{x\}$. Hence it contains E. Since f and f' agree on E, they are equal (same reasoning as above).

Assume now that $\mathscr{I}(\mathrm{S}',\mathrm{T},f')$ and $\mathscr{I}(\mathrm{S}'',\mathrm{T},f'')$ are true, and S' and S'' are segments such that $\mathrm{S}' \subset \mathrm{S}''$. Let f be the restriction of f'' on S'. This is a surjective function. We have $f_{\mathrm{S}'}^{(x)} = f_{\mathrm{S}'}^{\prime\prime(x)}$ (because the functions are surjective and take the same values on S'). We have $f_{\mathrm{S}'}^{\prime\prime(x)} = f_{\mathrm{S}''}^{\prime\prime(x)}$ because these are two restrictions to identical segments. We have $f(x) = f''(x)$. Assumption $\mathscr{I}(\mathrm{S}'',\mathrm{T},f'')$ gives thus $f(x) = \mathrm{T}\{f_{\mathrm{S}'}^{(x)}\}$, in other words, $\mathscr{I}(\mathrm{S}',\mathrm{T},f)$, hence $f = f'$ by uniqueness. This is *transfinite_aux1*.

Assume we have a set of segments $\mathfrak{S}$ and, for $\mathrm{S} \in \mathfrak{S}$, we have a function f_s such that $\mathscr{I}(s,\mathrm{T},f_s)$. We use our axiom of choice to select such a function f_s. Let t be the union of the targets of the f_s, and let g_s be the function that has the same source as f_s (namely s), same graph as f_s, but target t. If s and s' are two segments, we have $s \subset s'$ or $s' \subset s$; in the first case g_s and $g_{s'}$ agree on s (previous result). By symmetry g_s and $g_{s'}$ agree on s' in the other case. Using *extension_covering*, we get a function g that coincides with every g_s. The target of this function is t, so that the function is surjective. Assume $x \in \mathrm{S}$ and $\mathrm{S} \in \mathfrak{S}$. We have $g(x) = f_{\mathrm{S}}(x)$ because $f_{\mathrm{S}}(x) = g_{\mathrm{S}}(x)$. Thus the quantities $g_t^{(x)}$ and $f_{\mathrm{S}t}^{(x)}$ are well defined and equal. We have $f_{\mathrm{S}t}^{(x)} = f_{\mathrm{SS}}^{(x)}$ because these two functions are restriction of f_{S} to the segment defined by x, and whether this segment is defined by the ordering on t or S is irrelevant (we have $x \in \mathrm{S} \subset t$). We have $f_{\mathrm{S}}(x) = \mathrm{T}\{f_{\mathrm{SS}}^{(x)}\}$ by assumption. Thus $g(x) = \mathrm{T}\{g_t^{(x)}\}$. This means that $\mathscr{I}(t,\mathrm{T},g)$ is true. This is *transfinite_aux2*. We changed a bit the theorem; initially it was: if, for $\mathrm{S} \in \mathfrak{S}$, there exists a function f such that $\mathscr{I}(\mathrm{S},\mathrm{T},f)$, then there is a function g such that $\mathscr{I}(t,\mathrm{T},g)$. It is now: given functions f_s such that $\mathscr{I}(s,\mathrm{T},f_s)$, there exists a function g such that $\mathscr{I}(t,\mathrm{T},g)$ whose target is the union of the targets of the functions f_s (this target has to be the union, by uniqueness).

Assume now that $\mathscr{I}(\mathrm{S}_y,\mathrm{T},f)$. Let's extend f to f' via to $\mathrm{S}_y \cup \{y\}$ via the formula $f'(y) = \mathrm{T}\{f^{(y)}\}$. This function takes the same values as f on subsets of S_y hence $f'(x) = \mathrm{T}\{f_{\mathrm{S}_y}^{\prime(x)}\}$ for every $x \in \mathrm{S}_y$. The formula is also true for $y = x$, hence $\mathscr{I}(\mathrm{S}_y \cup \{y\},\mathrm{T},f')$. These facts allow us

to use the transfinite induction principle, to show the existence of a function defined everywhere. Assume that for some F we have $\mathrm{T}\{u\} \in \mathrm{F}$. If we change the definition of $\mathscr{I}(\mathrm{S},\mathrm{T},f)$ by adding the condition that the target of f is a subset of F, we can easily show that the target of the unique function defined by transfinite induction is a subset of F.

```
Definition transfinite_def r p f:=
  surjective f & source f = substrate r &
  forall x, inc x (substrate r) -> W x f = p (restriction_to_segment r x f).
Definition transfinite_defined r p:= choose (fun f => transfinite_def r p f).


Lemma transfinite_unique1: forall r p  f f' z, worder r ->
  inc z (substrate r) ->
  transfinite_def r p f -> transfinite_def r p f' ->
  (forall x : Set, inc x (segment r z) -> W x f = W x f') ->
  restriction_to_segment r z f = restriction_to_segment r z f'.
Lemma transfinite_unique: forall r p  f f', worder r ->
  transfinite_def r p f -> transfinite_def r p f' -> f = f'.  (* 37 *)
Lemma transfinite_pr: forall r x p, worder r -> transfinite_def r p x ->
  transfinite_defined r p = x.

Lemma transfinite_aux1: forall r p s' s'' f' f'',
  worder r -> is_segment r s' -> is_segment r s'' -> sub s' s'' ->
  transfinite_def (induced_order r s')  p f' ->
  transfinite_def (induced_order r s'') p f'' ->
  f' = restriction1 f'' s'.  (* 31 *)

(* old version
Lemma transfinite_aux2: forall r p s, worder r -> (* 70 *)
  (forall z, inc z s -> is_segment r z) ->
  (forall z, inc z s -> (exists f : correspondenceC,
                transfinite_def (induced_order r z) p f)) ->
  exists f : correspondenceC, transfinite_def (induced_order r (union s)) p f.
*)

Lemma transfinite_aux2: forall r p s tdf, worder r -> (* 80 *)
  (forall z, inc z s -> is_segment r z) ->
  (forall z, inc z s ->  transfinite_def (induced_order r z) p (tdf z)) ->
  exists f,
    (transfinite_def (induced_order r (union s)) p f &
      target f = unionf s (fun z => target (tdf z))).

Lemma transfinite_aux3: forall r p x g,
  worder r ->  inc x (substrate r)
  -> transfinite_def (induced_order r (segment r x)) p g
  -> transfinite_def (induced_order r (segment_c r x)) p
    (tack_on_f g x (p (restriction_to_segment r x g))). (* 44 *)

Theorem transfinite_definition: forall r p,
  worder r -> exists_unique (fun f => transfinite_def r p f). (* 27 *)

Lemma transfinite_defined_pr: forall r p, worder r ->
  transfinite_def r p  (transfinite_defined r p).

Theorem transfinite_definition_stable: forall r p F,
  worder r ->
```

```
  (forall f, is_function f -> is_segment r (source f) -> sub (target f) F ->
    inc (p f) F) ->
  sub (target (transfinite_defined r p)) F. (* 53 *)
```

3.3 Zermelo's theorem

We show here that every set is the substrate of a well-ordering. This requires quite a number of small results. Consider two well-orderings, Γ and Γ' on E and E$'$; denote by S_x and S'_x the segments with endpoints x in Γ and Γ' (these are subsets of E and E$'$). Let V be the set of all $x \in E \cap E'$ such that $S_x = S'_x$ and $(S_x \times S_x) \cap \Gamma = (S'_x \times S'_x) \cap \Gamma'$; the last condition says that Γ and Γ' induce the same ordering on S_x. This set is a segment for Γ and Γ', and both orderings coincide.

```
Definition common_ordering_set r r' :=
  Zo (intersection2 (substrate r) (substrate r'))
  (fun x => segment r x = segment r' x &
    induced_order r (segment r x) = induced_order r' (segment r' x)).

Lemma Zermelo_aux0: forall r r',
  common_ordering_set r r' = common_ordering_set r' r.
Lemma Zermelo_aux1: forall r r', worder r ->  worder r' ->
   is_segment r (common_ordering_set r r'). (* 33 *)
Lemma Zermelo_aux2: forall r r' v, worder r ->  worder r' ->
  v = common_ordering_set r r' ->  sub(induced_order r v)(induced_order r' v).
```

Let Q(Γ) denote the following property: Γ is a well-ordering of a subset of E, and if S_x denotes the segment of Γ with endpoint x, we have $S_x \in \mathfrak{S}$ and $p(S_x) = x$.

```
Definition Zermelo_axioms E p s r:=
  worder r &
  sub (substrate r) E &
  (forall x, inc x (substrate r) -> inc (segment r x) s) &
  (forall x, inc x (substrate r) -> p (segment r x)  = x).
```

Let $q(\Gamma, \Gamma')$ be the property that, if E and E$'$ are the substrates of Γ and Γ', then $E \subset E'$, E is a segment of Γ', and Γ is the ordering induced by Γ' on E. We pretend that if Q(Γ) and Q(Γ') are true, then either $q(\Gamma, \Gamma')$ or $q(\Gamma', \Gamma)$ is true. The previous lemmas show that this is true if V is either E or E$'$. Otherwise, we may consider x and x' the least element of E or E$'$ not in V, so that $V = S_x$ and $V = S_{x'}$ (for the orderings Γ and Γ'). By application of p we get $x = x'$. This implies $x \in V$, absurd.

```
Lemma Zermelo_aux3: forall E s p r r',
  let q := fun r r' => sub (substrate r) (substrate r')
    & r = induced_order r' (substrate r) & is_segment r' (substrate r) in
    Zermelo_axioms E p s r -> Zermelo_axioms E p s r' ->
    q r r' \/ q r' r.  (* 50 *)
```

Let Γ be a well-ordering and x an element not in the substrate. We can extend Γ as Γ' by adjoining x as greatest element. This is a well-ordering, a segment of this order is either the substrate of Γ', the substrate of Γ, or a segment S_x of Γ.

```
Lemma Zermelo_aux4: forall r a, worder r ->
  let owg := order_with_greatest r a in
    ~ (inc a (substrate r)) ->
    (worder owg &  segment owg a = (substrate r) &
      forall x, inc x (substrate owg) -> x = a \/
      segment owg x = segment r x).  (* 23 * )
```

Let E be a set, $\mathfrak{S}$ a part of $\mathfrak{P}(E)$ and p a function from $\mathfrak{S}$ into E such that $p(X) \notin X$. Consider $\mathfrak{M}$, the set of orderings Γ that satisfy property Q; by virtue of *Zermelo_aux3*, there is a well-ordering Γ (the union of the elements of $\mathfrak{M}$) that extends the orderings of $\mathfrak{M}$. It satisfies Q. Its substrate M is not in $\mathfrak{S}$. Proof: if $M \in \mathfrak{S}$, and $a = p(M)$, we know $a \notin M$, so that we can extend Γ to Γ' by adjoining a as greatest element. This new order satisfies Q (for all y, $S_y \in \mathfrak{S}$ and $p(S_y) = y$; this is true if y is in the support of Γ, otherwise $y = a$ and $S_a = M \in \mathfrak{S}$, $p(S_a) = p(M) = a$). This means that the new order is in $\mathfrak{M}$, its support is a subset of M, and $a \in M$, absurd.

```
Lemma Zermelo_aux:  forall E s p, sub s (powerset E) ->
  (forall x, inc x s -> inc (p x) E) & ~ (inc (p x) x))->
  exists r, Zermelo_axioms E p s r & (~  (inc (substrate r) s)). (* 68 *)
```

Let now $\mathfrak{S}$ be the set of all subsets of E but E itself. If p is the representative of the complement of x in E, the order defined by the lemma *Zermelo_aux* has its substrate in $\mathfrak{P}(E) - \mathfrak{S}$. Thus, it is a well-ordering on E. This is Theorem 1 [3, p. 153].

```
Theorem Zermelo: forall E, exists r, worder r & substrate r = E. (* 17 *)
```

3.4 Inductive sets

An ordered set is said to be *inductive* if every totally ordered subset of E has an upper bound in E. More precisely, let r be an order and E its substrate, then every subset X of E, for which the order induced by r is total, has an upper bound for r. The set $\Phi(A,B)$ of partial functions is inductive, another example is given page 174.

```
Definition inductive_set r :=
  forall X, sub X (substrate r) -> total_order (induced_order r X) ->
    exists x, upper_bound r X x.

Lemma inductive_graphs: forall a b,
  inductive_set (opposite_order (extension_order a b)). (* 35 *)
```

Consider an ordered set E; assume that each well-ordered subset of E is bounded above. Let $p(S)$ be an upper bound of S that is not in S. Let $\mathfrak{S}$ be the set of sets $S \subset E$ for which such a $p(S)$ exists. By *Zermelo_aux*, there is Γ that satisfies Q, i.e. Γ is a well-ordering of a subset M of E, and if S_x denotes the segment of Γ with endpoint x, we have $S_x \in \mathfrak{S}$ and $p(S_x) = x$. This last condition says that if $y < x$ for Γ, it is true for the ordering on E; hence Γ is the restriction to M of the ordering of E. By assumption, M is bounded, say by m. This element is maximal (this is Proposition 4 [3, p. 154]).

Theorem 2 [3, p. 154] says that every inductive ordered set has a maximal element. This is a trivial consequence of the previous result.

```
Theorem Zorn_aux: forall r, order  r ->
  (forall s, sub s (substrate r) -> worder (restriction_order r s) ->
    (bounded_above r s)) ->
  exists a, maximal_element r a.  (* 48 *)
Theorem Zorn_lemma: forall r, order  r -> inductive_set  r ->
  exists a, maximal_element r a.
```

Corollary. If E is inductive, $a \in E$, F is the set of all $x \geq a$, then F is inductive (if X is a totally ordered set in F, then $X \cup \{a\}$ is totally ordered; an upper bound m is in F since it satisfies $a \leq m$). Hence there is a maximal element m such that $a \leq m$. Second corollary: if $\mathfrak{F}$ is a subset of the powerset of E such that for every subset $\mathfrak{G}$ of $\mathfrak{F}$ which is totally ordered by inclusion, the union (resp. intersection) of the sets of $\mathfrak{G}$ belongs to $\mathfrak{F}$, then $\mathfrak{F}$ has a maximal or minimal element. The trick with the intersection is that the intersection is not defined if $\mathfrak{G}$ is empty. The set is nevertheless inductive, provided that $\mathfrak{F}$ is not empty (in the case of the union, it contains the empty set).

```
Lemma inductive_max_greater: forall r a, order r ->  inductive_set r ->
  inc a (substrate r) ->
  exists m, maximal_element r m & gle r a m. (* 32 *)
Lemma inductive_powerset: forall A F, sub A (powerset F) ->
  (forall S, (forall x y, inc x S -> inc y S -> sub x y \/ sub y x) ->
    sub S A ->inc (union S) A) ->
  inductive_set (inclusion_suborder A).
Lemma maximal_in_powerset: forall A F, sub A (powerset F) ->
  (forall So, (forall x y, inc x So -> inc y So -> sub x y \/ sub y x) ->
    sub So A -> inc (union So) A) ->
  exists a, maximal_element (inclusion_suborder A) a.
Lemma minimal_in_powerset: forall A F, sub A (powerset F) -> nonempty A ->
  (forall So, (forall x y, inc x So -> inc y So -> sub x y \/ sub y x) ->
    sub So A -> inc (intersection So) A) ->
  exists a, minimal_element (inclusion_suborder A) a.
```

3.5 Isomorphisms of well-ordered sets

Assume that E and F are two well-ordered sets. We show Theorem 3 [3, p. 155]: Let $I(u,v,f)$ be the property that f is an order isomorphism from u onto a segment w of v. We claim that there exists a unique f such that $I(E,F,f)$, or there exists a unique f such that $I(F,E,f)$. Note: The two cases are not excluded; in that case, f is bijection between E and F.

Recall that an order isomorphism is a bijection such that $x \leq y$ is equivalent to $f(x) \leq f(y)$. In the proof of the theorem we use the set $\mathfrak{F}$ of triples (A, B, C) associated to a function f, satisfying the following properties: The source A of f is a subset of E, the target B is the set F (this is also called a partial function from E to F). We assume that the source and the range of f are segments of E and E′; finally, we assume that f is an isomorphism onto its range. In other terms, $a \leq b$ is equivalent to $f(a) \leq f(b)$. The definition of *order_morphism* requires f to be injective. We start with a lemma that says that this condition is not needed.

```
Lemma order_morphism_pr1: forall f r r',
  order r -> order r' -> is_function f -> substrate r = source f ->
  substrate r' = target f ->
  (forall x y, inc x (source f) -> inc y (source f) ->
    gle r x y = gle r' (W x f) (W y f))
  -> order_morphism f r r'.
```

In order to show uniqueness we start with a lemma: if f is increasing and g is strictly increasing, if the image of f is a segment of F, then $f(x) \leq g(x)$ for all x. The proof is by contradiction. If a is the least element such that $g(a) < f(a)$, since the image of f is a segment there is a z such that $g(a) = f(z)$. Since f is increasing this gives $z < a$, hence $f(z) \leq g(z) < g(a)$, absurd.

```
Lemma increasing_function_segments: forall r r' f g,
  worder r -> worder r' ->
  increasing_fun f r r' -> strict_increasing_fun g r r'->
  is_segment r' (range (graph f)) ->
  forall x, inc x (source f) -> gle r' (W x f) (W x g).  (* 31 *)
Lemma isomorphism_worder_unique: forall r r' x y,
  worder r -> worder r' ->  is_segment r' (range (graph x)) ->
  is_segment r' (range (graph y)) ->
  order_morphism x r r' -> order_morphism y r r'
  ->  x = y.
Lemma induced_order_trans: forall a b c, sub c b ->
  induced_order (induced_order a b) c = induced_order a c.
```

Given a totally ordered subset X of $\mathfrak{F}$, we can apply lemma *sup_extension_order2*, that says that there exists a function f that extends all elements in X; we know that the source and range of f are the union of the sources and ranges of the elements of X, hence are segments. Given a and b in the source of f, there is a function g that is defined for both a and b (because X is totally ordered); since $a \leq b$ is equivalent to $g(a) \leq g(b)$ and $f(a) = g(a)$ and $f(b) = g(b)$ we deduce that $a \leq b$ is equivalent to $f(a) \leq f(b)$. As a consequence, f is increasing and hence is a morphism. Consider now a maximal element f. If the source of f is E, then I(E, F, f) is true. If the range of f is F, then f^{-1} is a bijection from F onto a subset of E, hence I(F, E, f^{-1}). Otherwise, if a is the least element of E not in the source of f and b the least element not in the range of b, we can extend f to a function g by saying $g(a) = b$. This function is in $\mathfrak{F}$. This contradicts the maximality of f.

```
Theorem isomorphism_worder: forall r r', (* 158 *)
 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).
```

Corollary 1. The only isomorphism from a well-ordered set into a segment of itself is the identity.

```
Lemma identity_isomorphism:  forall r, order r ->
  order_isomorphism (identity (substrate r)) r r.
Lemma identity_morphism:  forall r, order r ->
  order_morphism (identity (substrate r)) r r.
Lemma unique_isomorphism_onto_segment: forall r f, worder r ->
  is_segment r (range (graph f)) ->  order_morphism f r r ->
  f = identity (substrate r).
```

Corollary 2. If E and F are two well-ordered sets, f an isomorphism of E onto a segment of F, g an isomorphism of F onto a segment of E, then f and g are inverse bijections.

```
Lemma inverse_order_isomorphism: forall r r' f ,
   order_isomorphism f r r' -> order_isomorphism (inverse_fun f) r' r.
```

```
Lemma compose_order_isomorphism: forall r r' r'' f f',
 order_isomorphism f r r' -> order_isomorphism f' r' r''
  -> order_isomorphism (compose f' f) r r''.
Lemma compose_order_morphism: forall r r' r'' f f',
  composable f' f -> order_morphism f r r' -> order_morphism f' r' r''
  -> order_morphism (compose f' f) r r''.
Lemma bij_pair_isomorphism_onto_segment: forall r r' f f',
  worder r -> worder r' ->
  is_segment r' (range (graph f)) ->  order_morphism f r r' ->
  is_segment r (range (graph f')) ->  order_morphism f' r' r ->
  (order_isomorphism f r r' & order_isomorphism f' r' r &
    f = inverse_fun f'). (* 52 *)
```

Finally, we show that every subset of a well-ordered set is isomorphic to a segment of E.

```
Lemma isomorphic_subset_segment: forall r a,
  worder r -> sub a (substrate r) ->
  exists w, exists f, is_segment r w &
    order_isomorphism f (induced_order r a) (induced_order r w). (* 46 *)
```

3.6 Lexicographic products

Consider the follow definition. We assume that $(X_\iota)_{\iota\in I}$ is a family of sets, with domain I, g is a family of orderings (for each ι, g_ι is an ordering on f_ι) and r is a well-ordering on I. Denote the order relation associated to r by $\leq$, and the order relation associated to g_ι by $\leq_\iota$. The *lexicographic product* is the order associated to the relation: x and y are elements of the product $\prod X_\iota$, and either $x = y$, or, if ι is the smallest index (for the relation $\leq$) such that $x_\iota \neq y_\iota$ then $x_\iota \leq_\iota y_\iota$.

```
Definition lexicographic_order_r (r g:Set): Set -> Set -> Prop  :=
  fun x x' =>
    forall j, least_element (induced_order r (Zo (domain g)
      (fun i =>  V i x <> V i x'))) j -> glt (V j g) (V j x)(V j x').

Definition lexicographic_order_axioms r g:=
  worder r & substrate r = domain g & fgraph g &
  (forall i, inc i (domain g) -> order (V i g)).

Definition lexicographic_order r g :=
  graph_on (lexicographic_order_r r g)(prod_of_substrates g).
```

It is obvious that the lexicographic product is well-defined; it is a bit more longish to prove that it is an order on the product.

```
Lemma lexorder_substrate_aux: forall r g x,
  lexicographic_order_axioms r g ->
  lexicographic_order_r r g x x.
Lemma lexorder_substrate: forall r g,
  lexicographic_order_axioms r g ->
  substrate(lexicographic_order r g) = product_of_substrates g.
Lemma lexorder_order: forall r g,
  lexicographic_order_axioms r g -> order (lexicographic_order r g). (* 86 *)
Lemma lexorder_gle: forall r g x x',
```

```
  lexicographic_order_axioms r g ->
  related (lexicographic_order r g) x x' =
  (inc x (prod_of_substrates g)&  inc x' (prod_of_substrates g) &
    forall j, least_element (induced_order r (Zo (domain g)
      (fun i =>  V i x <> V i x'))) j -> glt (V j g) (V j x)(V j x')).
```

If all orders are total so is the lexicographic product.

```
Lemma total_lexicographic_order: forall r f g,
  lexicographic_order_axioms r f g ->
  (forall i, inc i (domain g) -> total_order (V i g)) ->
  total_order(lexicographic_order r f g). (* 23 *)
```

The section on ordinal numbers was moved in Version 3 to Chapter 8. All lemmas but the following two ones were also moved.

```
Lemma disjoint_union2_rw:  forall a b x y,  y <> x ->
  disjoint_union (variantL x y a b) =
  union2 (product a (singleton x)) (product b (singleton y)).
Lemma disjoint_union2_rw1:  forall a b,
  disjoint_union (variantLc a b) =
  union2 (product a (singleton TPa)) (product b (singleton TPb)).
```

Chapter 4

Equipotent Sets. Cardinals

Bourbaki denotes by Eq(X, Y) the property that there is a bijection between X and Y and denotes by Card(X) the set $\tau_Z(\mathrm{Eq}(X,Z))$. He calls this *the cardinal of* X, or *the power of* X. It is interesting to notice that no name is given to the notation $\mathfrak{a}^{\mathfrak{b}}$ when this means the cardinal of the set of mappings from one set into another (the operation is nevertheless called "exponentiation of cardinals"). The term "power" is used only in the phrase "power of the continuum", where it means the cardinal of the set of real numbers (to be defined elsewhere), or, equivalently, the cardinal of $\mathfrak{P}(\mathbf{N})$ (where **N** is the set of natural integers, defined in Chapter 6). For us, the term "power" will only be used to denote $\mathfrak{a}^{\mathfrak{b}}$.

Bourbaki does not define "a cardinal". The only possible interpretation of "$\mathfrak{r}$ is a cardinal" is "the object $\mathfrak{r}$ is of the form Card(E) for some set E". Using a specific font for cardinals suggests that a cardinal is some special object, and that we should perhaps introduce a type for these cardinals. If $A = \{\emptyset\}$ and $\mathfrak{a}$ denotes the cardinal of A, it is impossible to prove $\mathfrak{a} = A$ or $\mathfrak{a} \neq A$ (non-uniqueness of τ). This means that giving a different type to these objects invalidates no theorem of Bourbaki (with the exception of $0 = \emptyset$).

One can multiply cardinals. For instance $\mathfrak{a}.\mathfrak{a} = \mathrm{Card}(A \times A)$ is $\mathfrak{a}$ (if A is the set given above). This makes sense because $\mathrm{Card}(A \times B)$ is left unchanged when A is replaced by A' with $\mathrm{Card}(A) = \mathrm{Card}(A')$. But it becomes difficult to assert associativity or commutativity of the product: we need the notion of a family of cardinals, which is a function $f : I \to C$, where I is some index set and C is the type of the cardinals (we can also consider graphs of functions, which are subsets of $I \times C$; this is even more problematic, because C is too big to be a set, hence $I \times C$ cannot be identified to a set). If J is a subset of I, the cardinal product of the restriction of f to J is a cardinal, and this induces a function $\mathfrak{P}(I) \to C$. Such an object is neither a function in the Bourbaki sense (with a source, a target, a graph), neither in the Coq sense.

Bourbaki solves the problem by defining the cardinal product as the cardinal of the product. This means that cardinals are ordinary sets. This also means that we can consider the cardinal product of any two sets. The relation $\mathfrak{a}.\mathfrak{b} = \mathfrak{b}.\mathfrak{a}$, valid for two cardinals, is valid for any two sets.

Our theorems are easier to prove and use if we drop the requirements that the arguments are cardinals. The relation $\mathfrak{a}.1 = \mathfrak{a}$ is true only if $\mathfrak{a}$ is a cardinal. The relation $\mathfrak{a} \leq \mathfrak{a}\mathfrak{b}$ is true if $\mathfrak{b} \neq 0$; in order for $\leq$ to be antisymmetric we impose that its arguments are cardinals, so that we need $\mathfrak{a}$ to be a cardinal. Our proof uses $1 \leq \mathfrak{b}$ which is valid only if $\mathfrak{b}$ is a cardinal. Thus our theorem assumes that both arguments are cardinals. This is a case where assumptions are not minimal.

4.1 The cardinal of a set

We denote by Eq(X,Y) or *equipotent X Y* the property that there is a bijection between X and Y. We know that this relation is reflexive, symmetric and transitive (but is not an equivalence, because it has no graph). If a set E contains the n distinct elements $x_1, x_2 \ldots, x_n$, and if a set F contains the n distinct elements $y_1, y_2 \ldots, y_n$, then the set $G = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ is the graph of a bijection between E and F. We may use this method to prove equipotency of doubletons.

```
Lemma singletons_equipotent: forall x y,
  equipotent (singleton x) (singleton y).
Lemma doubleton_equipotent1: forall x y x' y',
  x <> x' -> y <> y' -> equipotent (doubleton x x')(doubleton y y').
```

Products of equipotent sets are equipotent. We first consider the case of *productf* since *ext_map_prod* is a bijection from $\prod E_\iota$ to $\prod F_\iota$ (See Part I, section 6.7). We also consider the case of the product of two sets (the bijection is *ext_to_prodC*).

```
Lemma equipotent_productf:  forall I p1 p2,
  (forall i, inc i I -> equipotent (p1 i) (p2 i)) ->
   equipotent (productf I p1) (productf I p2).
Lemma equipotent_productb:  forall x y, fgraph x -> fgraph y ->
  domain x = domain y ->
  (forall i, inc i (domain x) -> equipotent (V i x) (V i y)) ->
  equipotent (productb x) (productb y).
Lemma equipotent_product: forall a b a' b',
  equipotent a a' -> equipotent b b' ->
  equipotent (product a b) (product a' b').
```

If A, B and C are sets, then $A \times B$ is equipotent to $B \times A$, (bijection is *inv_graph_canon*) and $A \times (B \times C)$ is equipotent to $(A \times B) \times C$ (the bijection maps $(a,(b,c))$ to $((a,b),c)$). We have $(A \cup B) \times C = (A \times C) \cup (B \times C)$. Finally, if B is a singleton, A and $A \times B$ are equipotent (bijection is *bourbaki_ex5_17*).

```
Lemma equipotent_product_sym: forall a b,
  equipotent (product a b)(product b a).
Lemma product2associative : forall a b c,
  equipotent (product a (product b c)) (product (product a b) c).
Lemma distrib_inter_prod2: forall a b c,
  product (union2 a b) c = union2 (product a c) (product b c).
Lemma distrib_inter_prod3: forall a b c,
  product c (union2 a b) = union2 (product c a) (product c b).
Lemma equipotent_a_times_singl: forall a b,
  equipotent a  (product a (singleton b)).
Lemma equipotent_singl_times_a: forall a b,
  equipotent a (product (singleton b) a).
```

Two sets $A \times B$ and $A' \times B'$ are disjoint if B and B' are disjoint; this is the case when B and B' are distinct singletons.

```
Lemma disjoint_pr: forall a b,
  (forall u, inc u a -> inc u b -> False) -> disjoint a b.
Lemma disjoint_union2_pr0: forall a b x y,
```

```
  disjoint x y -> disjoint (product a x) (product b y).
Lemma disjoint_union2_pr1: forall x y,
  x <> y -> disjoint (singleton x) (singleton y).
Lemma disjoint_union2_pr: forall a b x y,
  x <> y -> disjoint (product a (singleton x)) (product b (singleton y)).
```

Two unions $\bigcup X_\iota$ and $\bigcup Y_\iota$ with the same index set are equipotent if the sets are equipotent and if each family is mutually disjoint. For if f_ι is a function from X_ι into Y_ι, we can find a function f such that $f(x) = f_\iota(x)$ whenever $x \in X_\iota$, provided that x is in a unique X_ι (i.e., the family is mutually disjoint). If the family $\bigcup Y_\iota$ is mutually disjoint, then $f(x) = f(y)$ implies that there is ι such that $x \in X_\iota$ and $y \in X_\iota$, hence $f_\iota(x) = f_\iota(y)$. Thus f is injective if each f_ι is injective. As a particular case, we get conditions for $A \cup B$ and $A' \cup B'$ to be equipotent.

Two disjoint unions of equipotent sets are equipotent. Remember that the disjoint union of a family of sets X_ι is the union of the sets $X'_\iota = X_\iota \times \{\iota\}$. Two lemmas mentioned above say that each X_ι is equipotent to X'_ι and the family is disjoint.

```
Lemma equipotent_disjoint_union: forall X Y,
  fgraph X -> fgraph Y ->  domain X = domain Y ->
  (forall i, inc i (domain X) -> equipotent (V i X) (V i Y)) ->
  mutually_disjoint X ->  mutually_disjoint Y ->
  equipotent (unionb X) (unionb Y).  (* 45 *)
Lemma equipotent_disjoint_union1: forall X Y,
  fgraph X -> fgraph Y ->  domain X = domain Y ->
  (forall i, inc i (domain X) -> equipotent (V i X) (V i Y)) ->
  equipotent (disjoint_union X) (disjoint_union  Y).
Lemma union2Lv:forall a b, union2 a b = unionb (variantLc a b).
Lemma disjointLv:forall a b, disjoint a b ->
   mutually_disjoint (variantLc a b).
Lemma equipotent_disjoint_union2: forall a b a' b',
  disjoint a b -> disjoint a' b' -> equipotent a a' -> equipotent b b' ->
  equipotent (union2 a b) (union2 a' b').
```

Given two sets A and B, we can consider a family f defined on a doubleton $\{x, y\}$ such that $f(x) = A$ and $f(y) = B$. An example is the canonical family *variantLc*, denoted here by F. We pretend that there is a bijection g such that $f = F \circ g$ (since f and F are graphs, we compose with the graph of g). The lemma asserts that F is a functional graph whose domain is the target of g, since these are the conditions of the associativity theorems of the sum and product.

```
Definition doubleton_fam f a b :=
  exists x, exists y, x<>y & fgraph f &  domain f = doubleton x y &
    V x f = a &   V y f = b.
Lemma two_terms_bij: forall a b f, doubleton_fam f a b ->
  let F := (variantLc a b) in
  exists g, (bijective g & target g = domain F & fgraph F &
    f = gcompose F (graph g)).    (* 40 *)
```

The representative of an equivalent class for equipotency (this is not a set) of the set X, is called its *cardinal*, and denoted Card(X). Bourbaki denotes by 0, 1 and 2, the cardinals of the empty set, a singleton, or a doubleton with two distinct elements.[1]

[1] Definition of 0 and 2 changed in V3

```
Definition cardinal x := choose (fun z => equipotent x z).
Definition card_zero :=  emptyset.
Definition card_one := cardinal (singleton emptyset).
Definition card_two := cardinal (two_points).
```

Proposition 1 [3, p. 158] states that X and Y are equipotent if and only if they have the same cardinal. Note that X is equipotent to Card(X) (since Eq is reflexive). Thus, if Card(X) = Card(Y), the sets X and Y are equipotent by transitivity. Assume X and Y equipotent. By symmetry and transitivity, Z is equipotent to X if and only if Z is equipotent to Y, thus Card(X) = Card(Y).

```
Lemma cardinal_pr: forall x, equipotent (cardinal x) x.
Lemma cardinal_pr0: forall x, equipotent x (cardinal x).
Theorem cardinal_equipotent: forall x y,
  (cardinal x = cardinal y) = (equipotent x y).
```

We study here some properties of the cardinal 0. The first lemma says that the only set equipotent to $\emptyset$ is $\emptyset$, so that Card($\emptyset$) = $\emptyset$. In Version 1, we define zero to be the LHS of this relation, now we use the RHS. For compatibility, we state the three equalities obtained from Card($\emptyset$) = $\emptyset$ = 0. Note that X is empty if and only if its cardinal is zero.

```
Lemma equipotent_to_emptyset:
  forall x, equipotent x emptyset -> x = emptyset.
Lemma cardinal_zero: cardinal emptyset = emptyset.
Lemma zero_is_emptyset: card_zero = emptyset.
Lemma cardinal_emptyset: cardinal emptyset = card_zero.
Lemma cardinal_nonemptyset: forall x,
  cardinal x = card_zero -> x = emptyset.
Lemma cardinal_nonemptyset1: forall x,
  nonempty x -> cardinal x <> card_zero.
```

Objects of the form Card(x) are called *cardinals*.

```
Definition is_cardinal x:= exists y, x = cardinal y.
Lemma cardinal_cardinal: forall x, is_cardinal (cardinal x).
Lemma cardinal_of_cardinal: forall x, is_cardinal x -> cardinal x = x.
Lemma cardinal_equipotent1: forall x y, is_cardinal x -> is_cardinal y ->
  equipotent x y -> x = y.

Lemma cardinal0: is_cardinal card_zero.
Lemma cardinal1: is_cardinal card_one.
Lemma cardinal2: is_cardinal card_two.
```

We study some properties of the cardinal 1.

```
Lemma cardinal_singleton: forall x, cardinal(singleton x) = card_one.
Lemma cardinal_one_is_singleton: is_singleton(card_one).
```

We study some properties of the cardinal 2.

```
Lemma cardinal_doubleton: forall x x',
  x <> x' ->  cardinal(doubleton  x x') = card_two.
Lemma set_of_card_two: forall x, cardinal x = card_two  ->
  exists u, exists v, u<>v & x = doubleton u v.
Lemma cardinal_two_is_doubleton: exists x, exists x',
  x <> x' &  card_two = doubleton  x x'.
```

These three cardinals are distinct.

```
Lemma card_one_not_zero: card_one <> card_zero.
Lemma card_two_not_zero: card_two <> card_zero.
Lemma card_one_not_two: card_one <> card_two.
```

4.2 Order relation between cardinals

We restate here that composition of injective functions is injective. Thus if f is a bijection $A \to B$ and $B \subset C$, its composition g with the canonical inclusion $B \to C$ is an injection $A \to C$. Conversely, given a function $g : A \to C$, its restriction to its image B is a surjective function, so that if g is injective, its restriction is bijective. The existence of g depends only on the cardinals of A and C, and this defines an ordering on cardinals.

```
Lemma inj_compose1: forall f f',
  injective f-> injective f' -> source f' = target f ->
  injective (compose f' f).
Definition restriction_to_image f :=
  restriction2  f (source f) (image_of_fun f).
Lemma restriction_to_image_axioms: forall f, is_function f ->
  restriction2_axioms f (source f) (image_of_fun f).
Lemma restriction_to_image_surjective: forall f, is_function f ->
  surjective (restriction_to_image f).
Lemma restriction_to_image_bijective: forall f, injective f ->
  bijective (restriction_to_image f).
```

We say that $\mathfrak{r} \leq_{\mathrm{Card}} \mathfrak{n}$ if $\mathfrak{r}$ and $\mathfrak{n}$ are cardinals, and $\mathfrak{r}$ is equipotent to a subset of $\mathfrak{n}$. The notation is often simplified to $\mathfrak{r} \leq \mathfrak{n}$. We also introduce the notation $\mathfrak{r} <_{\mathrm{Card}} \mathfrak{n}$. We do not introduce specific notations for $\geq_{\mathrm{Card}}$ or $>_{\mathrm{Card}}$.

```
Definition equipotent_to_subset x y:= exists z, sub z y & equipotent x z.
Definition cardinal_le x y :=
  is_cardinal x & is_cardinal y &equipotent_to_subset x y.
Definition cardinal_lt a b := cardinal_le a b & a <> b.
```

We list here some useful properties. Since the composition of two injections is an injection, and bijections are injections, we get some interesting results. A set A is equipotent to a subset of B if and only if there is an injection from A into B. There is an injection from A into B if and only if there is an injection from Card(A) into Card(B). Thus A is equipotent to a subset of B if and only if Card(A) is equipotent to a subset of Card(B). If $\mathfrak{a}$ is a cardinal, then $\mathrm{Card}(\mathfrak{a}) = \mathfrak{a}$ and $\mathfrak{a} \leq \mathfrak{a}$. If A is equipotent to a subset of B and B is equipotent to C, then A is equipotent to a subset of C. If $A \subset B$ then A is equipotent to a subset of B. If $f : A \to B$ is injective then $\mathrm{Card}(A) \leq \mathrm{Card}(B)$. Finally, if E is a set of cardinals, the support of the order induced by $\leq_{\mathrm{Card}}$ on E is E.

```
Lemma cardinal_le1: forall x y,
  equipotent_to_subset x y =
  (exists f, injective f & source f = x & target f = y).
Lemma cardinal_le2: forall x y,
  equipotent_to_subset x y =  equipotent_to_subset (cardinal x) (cardinal y).
Lemma cardinal_le3: forall x y,
  equipotent_to_subset x y = cardinal_le (cardinal x) (cardinal y).
```

```
Lemma cardinal_le9: forall f, injective f ->
  cardinal_le (cardinal (source f)) (cardinal (target f)).
Lemma cardinal_le_reflexive: forall x, is_cardinal x -> cardinal_le x x.
Lemma cardinal_le5: forall E, (forall x, inc x E -> is_cardinal x) ->
    substrate (graph_on cardinal_le E) = E.
Lemma cardinal_le7: forall a b c,
  equipotent b c -> equipotent_to_subset a b ->
  equipotent_to_subset a c.
Lemma cardinal_le8: forall a b, sub a b -> equipotent_to_subset a b.
Lemma sub_smaller: forall a b,
  sub a b -> cardinal_le (cardinal a) (cardinal b).
```

Theorem one [3, p. 159] says that the ordering between cardinals is a well-ordering. The idea is the following. Let E be a set of cardinals, and A its union. Consider a well-ordering on A. Let $\phi(x)$ be the smallest segment of A equipotent to x (if $x \in$ E, x is a subset of A hence isomorphic, hence equipotent, to a segment of A; hence ϕ is well-defined). The relation $a \leq_{\mathrm{Card}} b$ on E is equivalent to $\phi(a) \subset \phi(b)$ (if $a \leq_{\mathrm{Card}} b$ then a is isomorphic to a subset of $\phi(b)$, hence a is isomorphic to a segment u of $\phi(b)$; by definition $\phi(a) \subset u$, hence $\phi(a) \subset \phi(b)$; converse is easy). From this, one deduces that the relation $a \leq_{\mathrm{Card}} b$ is an order on E. This is a well-ordering, since the set of segments is well-ordered. Assume $a \leq_{\mathrm{Card}} b$ and $b \leq_{\mathrm{Card}} a$. If we consider the doubleton $\{a, b\}$, we have $a \leq b$ and $b \leq a$ for the induced order, hence $a = b$.

```
Lemma cardinal_le_transitive: forall a b c,
  cardinal_le a b -> cardinal_le b c -> cardinal_le a c.
Theorem wordering_cardinal_le: worder_r  cardinal_le.  (* 137 *)
```

Some consequences: if $\mathfrak{a}$, $\mathfrak{b}$, and $\mathfrak{c}$, are cardinals, then $\mathfrak{a} \leq_{\mathrm{Card}} \mathfrak{b}$ and $\mathfrak{b} \leq_{\mathrm{Card}} \mathfrak{c}$ implies $\mathfrak{a} \leq_{\mathrm{Card}} \mathfrak{c}$ (this is easy). Either $\mathfrak{a} \leq_{\mathrm{Card}} \mathfrak{b}$ or $\mathfrak{b} \leq_{\mathrm{Card}} \mathfrak{a}$; but if both relations hold, then $\mathfrak{a} = \mathfrak{b}$. This can be restated as: if there is an injection from A into B and an injection from B into A, then there is a bijection between the two sets A and B. This is sometimes called the Cantor-Bernstein Theorem. It can be shown without the axiom of choice.

```
Lemma wordering_cardinal_le_pr : forall x,
  (forall a, inc a x -> is_cardinal a) ->
  (substrate (graph_on cardinal_le x) = x &
  worder (graph_on cardinal_le x)).
Lemma cardinal_antisymmetry1: forall x y,
  cardinal_le x y -> cardinal_le y x -> x = y.
Lemma not_card_le_lt: forall a b, cardinal_le a b -> cardinal_lt b a -> False.
Lemma cardinal_antisymmetry2: forall a b,
  equipotent_to_subset a b -> equipotent_to_subset b a ->
  equipotent a b.
Lemma cardinal_lt_le_trans: forall a b c,
  cardinal_lt a b -> cardinal_le b c -> cardinal_lt a c.
Lemma cardinal_le_lt_trans: forall a b c,
  cardinal_le a b -> cardinal_lt b c -> cardinal_lt a c.
Lemma cardinal_le_total_order: forall a b,
  equipotent_to_subset a b \/ equipotent_to_subset b a.
Lemma cardinal_le_total_order1: forall a b,
  is_cardinal a -> is_cardinal b ->
  a = b \/ cardinal_lt a b \/ cardinal_lt b a.
Lemma cardinal_le_total_order2: forall a b,
  is_cardinal a -> is_cardinal b ->
  cardinal_le a b \/ cardinal_lt b a.
```

```
Lemma cardinal_le_total_order3: forall a b,
  is_cardinal a -> is_cardinal b ->
  cardinal_le a b \/ cardinal_le b a.
```

We have $0 \leq \mathfrak{a}$ for every cardinal $\mathfrak{a}$, and $1 \leq \mathfrak{a}$ if moreover $\mathfrak{a} \neq 0$. We have Card(E) ≥ 1 if and only if E is non-empty,

```
Lemma zero_smallest: forall x, is_cardinal x -> cardinal_le card_zero x.
Lemma zero_smallest1: forall x, cardinal_lt x card_zero -> False.
Lemma zero_smallest2: forall a, cardinal_le a card_zero -> a = card_zero.
Lemma one_small_cardinal: forall x, is_cardinal x -> x <> card_zero ->
  cardinal_le card_one x.
Lemma one_small_cardinal1: forall x, cardinal_lt card_zero x ->
   cardinal_le card_one x.

Lemma card_le_one_prop:forall E,
  cardinal_le card_one (cardinal E) -> nonempty E.

Lemma card_le_one_prop1:forall E,
  nonempty E-> cardinal_le card_one (cardinal E).
```

We have Card(E) ≥ 2 if and only if E has at least two elements.

```
Lemma card_le_two_prop: forall E,
  cardinal_le card_two (cardinal E) ->
  exists a, exists b, inc a E & inc b E & a <> b.
Lemma card_le_two_prop1:forall E x y,
  inc x E -> inc y E -> x <> y -> cardinal_le card_two (cardinal E).
```

For every cardinal $\mathfrak{a}$, the set of objects of the form Card(b) for $b \in \mathfrak{P}(\mathfrak{a})$ is the set of cardinals $\leq \mathfrak{a}$. Given a family of cardinals $(\mathfrak{a}_\iota)_{\iota\in I}$, we can find a cardinal $\mathfrak{b}$ greater than all $\mathfrak{a}_\iota$ (for instance the union) and consider the set E of cardinals $\leq \mathfrak{b}$. The family, being bounded in a well-ordered set, has a supremum $\mathfrak{c}$. If $\mathfrak{d}$ is another upper bound, either $\mathfrak{d} \geq \mathfrak{b}$ hence $\mathfrak{d} \geq \mathfrak{c}$, or $\mathfrak{d} \leq \mathfrak{b}$, hence is in E and $\mathfrak{d} \geq \mathfrak{c}$. It is called the supremum of the family. This is Proposition 2 in [3, p. 160]. We first show the same result for a set of cardinals.

```
Definition set_of_cardinals_le a:=
  fun_image(powerset a)(fun x => cardinal x).

Lemma set_of_cardinals_pr: forall a b,  is_cardinal a ->
  inc b (set_of_cardinals_le a) = (cardinal_le b a).

Lemma cardinal_supremum: forall x,
  (forall a, inc a x -> is_cardinal a) ->
  exists_unique (fun b => is_cardinal b &
    (forall a, inc a x -> cardinal_le a b) &
    (forall c, is_cardinal c ->  (forall a, inc a x -> cardinal_le a c) ->
      cardinal_le b c)).   (* 44 *)
Theorem cardinal_supremum1: forall x,
  fgraph x ->
  (forall a, inc a (domain x) -> is_cardinal (V a x)) ->
  exists_unique (fun b => is_cardinal b &
    (forall a, inc a (domain x) -> cardinal_le (V a x) b) &
    (forall c, is_cardinal c ->
      (forall a, inc a (domain x) -> cardinal_le (V a x) c) ->
      cardinal_le b c)).
```

Proposition 3 in [3, p. 160] says that Card(Y) ≤ Card(X) if there is a surjection of X onto Y. As a consequence, the range of a function is not bigger than the source.

```
Theorem surjective_cardinal_le: forall x y,
  (exists z, surjective z & source z = x & target z = y) ->
  cardinal_le (cardinal y) (cardinal x).
Lemma image_smaller_cardinal: forall f, is_function f ->
  cardinal_le (cardinal (image_of_fun f))(cardinal (source f)).
```

4.3 Operations on cardinals

Given a family of cardinals $(\mathfrak{a}_\iota)_{\iota\in I}$, the cardinal of the sum of these sets is called the *cardinal sum* and denoted by $\sum_{\iota\in I}\mathfrak{a}_\iota$; the cardinal of the product is called the *cardinal product* and denoted $\mathsf{P}_{\iota\in I}\mathfrak{a}_\iota$. The qualificative "cardinal" will be omitted if there is no risk of confusion. The associated operations are called *addition* and *multiplication*. The notation $\prod_{\iota\in I}\mathfrak{a}_\iota$ will later be used for both the normal product and the cardinal product. There is no need to introduce a specific notation for the cardinal sum (since there is no notation for the sum, aka the disjoint union).

```
Definition cardinal_sum x := cardinal (disjoint_union x).
Definition cardinal_prod x := cardinal (productb x).
```

Proposition 4 of [3, p. 160] says that the cardinal sum or cardinal product of the family $\mathrm{Card}(\mathrm{E}_\iota)$ is the cardinal of the sum or the product of the sets E_ι. In other terms, if $\mathfrak{a}_\iota = \mathrm{Card}(\mathrm{E}_\iota)$ then $\mathrm{Card}(\prod \mathrm{E}_\iota) = \mathsf{P}\mathfrak{a}_\iota$ and $\mathrm{Card}(\mathsf{S}\mathrm{E}_\iota) = \sum\mathfrak{a}_\iota$. One can notice that the cardinal of a union is at most the cardinal of the disjoint union.

```
Theorem cardinal_prod_pr: forall x, fgraph x ->
  cardinal (productb x) =
  cardinal_prod (L (domain x) (fun a => cardinal (V a x))).
Theorem cardinal_sum_pr: forall x, fgraph x ->
  cardinal (disjoint_union x) =
  cardinal_sum (L (domain x) (fun a => cardinal (V a x))).
Lemma cardinal_sum_pr3: forall X Y, fgraph X -> fgraph Y ->
  domain X = domain Y ->
  (forall i, inc i (domain X) -> cardinal (V i X) = cardinal (V i Y)) ->
  cardinal_sum X = cardinal_sum Y.
Lemma cardinal_sum_pr1: forall x, fgraph x ->
  cardinal_le (cardinal (unionb x))
  (cardinal_sum (L (domain x) (fun a => cardinal (V a x)))).
```

Proposition 5 [3, p. 161] says that if f is a bijection from K to I and if $\mathfrak{a}_\iota$ is a cardinal then ("commutativity" of the sum and product):

$$\sum_{\kappa\in K}\mathfrak{a}_{f(\kappa)} = \sum_{\iota\in I}\mathfrak{a}_\iota, \qquad \mathsf{P}_{\kappa\in K}\mathfrak{a}_{f(\kappa)} = \mathsf{P}_{\iota\in I}\mathfrak{a}_\iota. \tag{4}$$

If the family $(\mathrm{J}_\lambda)_{\lambda\in L}$ is a partition of I, then ("associativity" of the sum and product):

$$\sum_{\iota\in I}\mathfrak{a}_\iota = \sum_{\lambda\in L}\Big(\sum_{\iota\in \mathrm{J}_\lambda}\mathfrak{a}_\iota\Big), \qquad \mathsf{P}_{\iota\in I}\mathfrak{a}_\iota = \mathsf{P}_{\lambda\in L}\Big(\mathsf{P}_{\iota\in \mathrm{J}_\lambda}\mathfrak{a}_\iota\Big). \tag{5}$$

Let $((\mathfrak{a}_{\lambda,\iota})_{\iota\in J_\lambda})_{\lambda\in L}$ be a family of families of cardinals. Let $I = \prod J_\lambda$. Distributivity of product over sum is

$$\prod_{\lambda\in L}\Big(\sum_{\iota\in J_\lambda}\mathfrak{a}_{\lambda,\iota}\Big) = \sum_{f\in I}\Big(\prod_{\lambda\in L}\mathfrak{a}_{\lambda,f(\lambda)}\Big). \tag{6}$$

Note that we do not need $\mathfrak{a}_\iota$ be a cardinal in any of these theorems. The relations are trivial for the product, and in the case of the union, we have to check that the families are disjoint. These formulas are numbered (4), (5) and (6), in order to respect the original Bourbaki numbering.

```
Theorem cardinal_sum_commutative: forall X f,
  fgraph X ->  target f = domain X -> bijective f ->
  cardinal_sum X = cardinal_sum (gcompose X (graph f)). (* 30 *)
Theorem cardinal_prod_commutative: forall X f,
  fgraph X ->  target f = domain X ->  bijective f ->
  cardinal_prod X = cardinal_prod (gcompose X (graph f)).
Theorem cardinal_sum_assoc: forall f g,
  fgraph f  -> partition_fam g (domain f) ->
  cardinal_sum f  = cardinal_sum (L (domain g) (fun l =>
    cardinal_sum (restr f (V l g)))).   (* 45 *)
Theorem cardinal_prod_assoc: forall f g,
  fgraph f  -> partition_fam g (domain f) ->
  cardinal_prod f  = cardinal_prod (L (domain g) (fun l =>
    cardinal_prod (restr f (V l g)))).
Theorem cardinal_distrib_prod_sum: forall f,  (* 57 *)
  fgraph f ->
  (forall l, inc l (domain f) -> fgraph (V l f)) ->
  cardinal_prod (L (domain f) (fun l => cardinal_sum (V l f))) =
  cardinal_sum (L (productf (domain f) (fun l => (domain (V l f))))
    (fun g => (cardinal_prod (L (domain f) (fun l => V (V l g) (V l f)))))).
```

We have $\sum f_i + \sum g_i = \sum f_i + g_i$ where I belongs to some set I. The proof is a bit tricky. Let $K = I \times \alpha,\beta$, where α and β are two distinct elements. We define a function h on K by that associates f_i to (i,α) and g_i to (i,β). We have $\sum f_i = \sum_\alpha h_i$ where the index α denotes the restriction of h to the first part of K (this is the commutativity theorem, we use some auxiliary lemmas in order to ease the proof). We have $\sum f_i + \sum g_i = \sum h_k$ by associativity. If we consider K as the disjoint union where the first component is fixed, we can reapply the associativity theorem.

```
Lemma cardinal_commutativity_aux: forall X f I,
  (forall x, inc x I -> inc (f x) (domain X)) ->
  (forall x y, inc x I -> inc y I -> f x = f y -> x = y) ->
  (forall y, inc y (domain X)-> exists x, inc x I & f x = y) ->
  fgraph X ->
  let F := (BL f I (domain X)) in
    (target F = domain X & bijective F &
       gcompose X (graph F) = L I (fun z : Set => V (f z<) X)).

Lemma cardinal_sum_commutative2: forall X f I,
  (forall x, inc x I -> inc (f x) (domain X)) ->
  (forall x y, inc x I -> inc y I -> f x = f y -> x = y) ->
  (forall y, inc y (domain X)-> exists x, inc x I & f x = y) ->
  fgraph X ->
```

```
  cardinal_sum X = cardinal_sum (L I (fun z : Set => V (f z) X)).
Lemma cardinal_prod_commutative2: forall X f I,
  (forall x, inc x I -> inc (f x) (domain X)) ->
  (forall x y, inc x I -> inc y I -> f x = f y -> x = y) ->
  (forall y, inc y (domain X)-> exists x, inc x I & f x = y) ->
  fgraph X ->
  cardinal_prod X = cardinal_prod (L I (fun z : Set => V (f z) X)).

Lemma sum_of_sums: forall f g I,
  card_plus (cardinal_sum (L I f)) (cardinal_sum (L I g)) =
  cardinal_sum (L I (fun i => card_plus (f i) (g i))). (* 86 *)
```

Given two sets $\mathfrak{a}$ and $\mathfrak{b}$, we can consider a family F defined on a doubleton $\{x, y\}$ such that $F(x) = \mathfrak{a}$ and $F(y) = \mathfrak{b}$. By commutativity, the cardinal sum and cardinal product of the family depends only on $\mathfrak{a}$ and $\mathfrak{b}$. It is denoted by $\mathfrak{a} + \mathfrak{b}$ and $\mathfrak{a}.\mathfrak{b}$ respectively. Commutativity is obvious.

```
Definition card_plus a b :=
  cardinal_sum (variantLc a b).
Definition card_mult a b :=
  cardinal_prod (variantLc a b).

Lemma card_plus_is_cardinal: forall a b, is_cardinal (card_plus a b).
Lemma card_mult_is_cardinal: forall a b, is_cardinal (card_mult a b).

Lemma card_plus_pr: forall a b f,
  doubleton_fam f a b ->  card_plus a b = cardinal_sum f.
Lemma card_mult_pr: forall a b f,
  doubleton_fam f a b ->  card_mult a b = cardinal_prod f.

Lemma card_commutative_aux: forall a b,
  doubleton_fam (Lvariantc b a) a b.
Lemma card_plusC: forall a b,
  card_plus a b = card_plus b a.
Lemma card_mult_C: forall a b,
  card_mult a b = card_mult b a.
```

If α and β are the two elements of the canonical doubleton I, if f is any function, the sum (resp. product) of the graph of f on I is $f(\alpha) + f(\beta)$ and $f(\alpha) \cdot f(\beta)$ respectively.

```
Definition TPas := singleton TPa.
Definition TPbs := singleton TPb.

Lemma doubleton_fam_canon: forall f,
  doubleton_fam (L two_points f) (f TPa) (f TPb).

Lemma card_plus_pr0: forall f,
  cardinal_sum (L two_points f) = card_plus (f TPa) (f TPb).
Lemma card_mult_pr0: forall f,
  cardinal_prod (L two_points f) = card_mult (f TPa) (f TPb).

Lemma disjoint_union2_pr3: forall a b  x y, y <> x ->
  equipotent (card_plus a b)
    (union2 (product a (singleton x)) (product b (singleton y))).
Lemma disjoint_union2_pr4: forall a b,
```

```
  equipotent (card_plus a b) (union2 (product a TPas) (product b TPbs)).
Lemma cardinal_sum_pr2: forall a b a' b', equipotent a a' -> equipotent b b' ->
  card_plus a b = card_plus a' b'.
Lemma card_plus_pr1: forall a b a' b',
  disjoint a b -> equipotent a a' -> equipotent b b' ->
  cardinal (union2 a b) = card_plus a' b'.
Lemma card_mult_pr1:forall a b,
  card_mult a b = cardinal (product a b).

Lemma card_plus_pr2: forall a b a' b',
  cardinal a = cardinal a' -> cardinal b = cardinal b' ->
  card_plus a b = card_plus a' b'.
Lemma card_mult_pr2: forall a b a' b',
  cardinal a = cardinal a' -> cardinal b = cardinal b' ->
  card_mult a b = card_mult a' b'.
```

As a corollary, if $\mathfrak{a}$, $\mathfrak{b}$ and $\mathfrak{c}$ are cardinals we have

(1) $$\mathfrak{a}+\mathfrak{b}=\mathfrak{b}+\mathfrak{a} \qquad \text{and} \qquad \mathfrak{a}\mathfrak{b}=\mathfrak{b}\mathfrak{a},$$

(2) $$\mathfrak{a}+(\mathfrak{b}+\mathfrak{c})=(\mathfrak{a}+\mathfrak{b})+\mathfrak{c} \qquad \text{and} \qquad \mathfrak{a}(\mathfrak{b}\mathfrak{c})=(\mathfrak{a}\mathfrak{b})\mathfrak{c},$$

(3) $$\mathfrak{a}(\mathfrak{b}+\mathfrak{c})=\mathfrak{a}\mathfrak{b}+\mathfrak{a}\mathfrak{c}.$$

Note that the formulas are true even if $\mathfrak{a}$, $\mathfrak{b}$ and $\mathfrak{c}$ are not cardinals. Associativity of the product is a consequence of equipotency of $A \times (B \times C)$ and $(A \times B) \times C$. Associativity of the sum is a consequence of associativity of $\cup$, commutativity of $+$ and $\cup$, and the property that $\mathfrak{a}+(\mathfrak{b}+\mathfrak{c})$ is equipotent $\mathfrak{a}_i \cup (\mathfrak{b}_j \cup \mathfrak{c}_k)$, if the indices are distinct; the current proof is four times shorter than the original one. The same idea can be used for (3). The quantity $\mathfrak{a}(\mathfrak{b}+\mathfrak{c})$ is equipotent to $\mathfrak{a} \times (\mathfrak{b}_i \cup \mathfrak{c}_j)$ hence to $(\mathfrak{a} \times \mathfrak{b}_i) \cup (\mathfrak{a} \times \mathfrak{c}_j)$, and $(\mathfrak{a} \times \mathfrak{b})_i \cup (\mathfrak{a} \times \mathfrak{c})_j$, by associativity of the product. We show in fact $(\mathfrak{b}+\mathfrak{c})\mathfrak{a} = \mathfrak{b}\mathfrak{a}+\mathfrak{c}\mathfrak{a}$, because, basically we use equipotency of $(A \cup B) \times C$ and $(A \times C) \cup (B \times C)$. The same techniques as above show

(7) $$\mathfrak{a}\sum_{\iota\in I}\mathfrak{b}_\iota=\sum_{\iota\in I}\mathfrak{a}\mathfrak{b}_\iota.$$

```
Lemma card_multA: forall a b c,
  card_mult a (card_mult b c) = card_mult (card_mult a b) c.
Lemma card_plusA: forall a b c, (* 21 *)
  card_plus a (card_plus b c) = card_plus (card_plus a b) c.
Lemma equipotent_product1: forall a b c,
  equipotent a b -> equipotent (product a c) (product b c).
Lemma cardinal_distrib_prod_sum3 : forall a b c,
  card_mult a (card_plus b c) =
  card_plus (card_mult a b) (card_mult a c). (* 20 *)
Lemma cardinal_distrib_prod_sum2 : forall a b c,
  card_mult (card_plus b c) a =
  card_plus (card_mult b a) (card_mult c a).
Lemma distrib_prod2_sum: forall A f,
  product A (unionb f) = unionb (L (domain f) (fun x => product A (V x f))).
Lemma cardinal_distrib_prod2_sum: forall a f, is_cardinal a -> fgraph f ->
  card_mult a (cardinal_sum f) =
  cardinal_sum (L (domain f) (fun i => (card_mult a (V i f)))).
```

4.4 Properties of the cardinals 0 and 1

If a family is empty, the sum is zero and the product is one. If a family has a single element that is a cardinal, this element is the sum or the product.

```
Lemma trivial_cardinal_sum: forall f, domain f = emptyset ->
  cardinal_sum f = card_zero.
Lemma trivial_cardinal_prod: forall f, fgraph f -> domain f = emptyset ->
  cardinal_prod f = card_one.
Lemma trivial_cardinal_sum1: forall x f, domain f = singleton x ->
  is_cardinal (V x f) -> cardinal_sum f =  V x f.
Lemma trivial_card_plus: forall x a, is_cardinal a ->
  cardinal_sum (cst_graph (singleton x) a) = a.
Lemma trivial_cardinal_prod1: forall x f, fgraph f -> domain f = singleton x ->
  is_cardinal (V x f) -> cardinal_prod f =  V x f.
```

One can remove 0 in a sum and 1 in a product. This is Proposition 6 [3, p. 162]. The result is clear for the sum, because $0_i = \emptyset$ (where 0_i means $0 \times \{i\}$). In the case of a product, it is a trivial consequence of *bijective_prj*. If the family has two elements, this gives nice results. If a factor of a product is zero, so is the product itself.

```
Theorem zero_unit_sum: forall f j,
  fgraph f -> sub j (domain f) ->
  (forall i, inc i (complement (domain f) j) -> (V i f) = card_zero) ->
  cardinal_sum f = cardinal_sum (restr f j).
Theorem one_unit_prod: forall f j,
  fgraph f -> sub j (domain f) ->
  (forall i, inc i (complement (domain f) j) -> (V i f) = card_one) ->
  cardinal_prod f = cardinal_prod (restr f j).

Lemma zero_unit_sumr: forall a, is_cardinal a ->
  card_plus a card_zero = a.
Lemma zero_unit_suml: forall a, is_cardinal a ->
  card_plus card_zero a = a.
Lemma one_unit_prodr: forall a, is_cardinal a ->
  card_mult a card_one = a.
Lemma one_unit_prodl: forall a, is_cardinal a ->
  card_mult card_one a = a.
Lemma zero_prod_absorbing: forall a, <
  card_mult a card_zero = card_zero.
Lemma zero_product_absorbing:  forall f,
  fgraph f ->  (exists i, inc i (domain f) & cardinal (V i f) = card_zero)
  -> cardinal_prod f = card_zero.
```

Let $\mathfrak{a}$ and $\mathfrak{b}$ be two cardinals; consider a set I equipotent to $\mathfrak{b}$ and the two families $\mathfrak{a}_\iota = \mathfrak{a}$ and $\mathfrak{c}_\iota = 1$. Then

$$\mathfrak{a}\mathfrak{b} = \sum_{\iota\in I} \mathfrak{a}_\iota; \qquad \mathfrak{b} = \sum_{\iota\in I} \mathfrak{c}_\iota. \tag{8}$$

The first formula is obtained from the second after multiplication by $\mathfrak{a}$, and using distributivity.

```
Lemma sum_of_ones: forall b j, is_cardinal b -> equipotent b j ->
  cardinal_sum (L j (fun _ => card_one)) = b.
```

```
Lemma sum_of_ones1: forall b,
  equipotent (cardinal_sum (cst_graph b card_one))  b.
Lemma sum_of_same: forall a b j, is_cardinal a -> is_cardinal b ->
  equipotent b j ->
  cardinal_sum (cst_graph j a) = card_mult a b.
Lemma sum_of_same1: forall a b,
  cardinal_sum (cst_graph b a) = card_mult a b.
```

Proposition 7 [3, p. 162] says that a cardinal product is non-zero if and only if each factor is non-zero (because a product is non-empty if and only if no factor is empty). Proposition 8 [3, p. 162] asserts injectivity of the successor function, namely that if $\mathfrak{a}$ and $\mathfrak{b}$ are two cardinals such that $\mathfrak{a}+1=\mathfrak{b}+1$ then $\mathfrak{a}=\mathfrak{b}$. In effect, there exists X equipotent to $\mathfrak{a}$, Y equipotent to $\mathfrak{b}$, and $u\notin X$, $v\notin Y$ such that $X\cup\{u\}=Y\cup\{v\}$. If $u=v$, then $X=Y$; otherwise, if $Z=Y\cap Y$, we have $X=Z\cup\{v\}$ and $Y=Z\cup\{u\}$, so that if $\mathfrak{c}=\mathrm{Card}(Z)$ we have $\mathfrak{a}=\mathfrak{b}=\mathfrak{c}+1$.

```
Lemma disjoint_with_singleton: forall a b,
  ~ (inc b a) -> disjoint a (singleton b).
Theorem zero_cardinal_product: forall f,
  fgraph f ->  (forall i, inc i (domain f) -> V i f <> card_zero) =
  (cardinal_prod f <> card_zero).
Lemma zero_cardinal_product2: forall a b, a <> card_zero  -> b <> card_zero ->
  card_mult a b <> card_zero.
Theorem succ_injective: forall a b, is_cardinal a ->
  is_cardinal b -> card_plus a card_one = card_plus b card_one ->
  a = b.    (* 91 *)
```

4.5 Exponentiation of cardinals

If $\mathfrak{a}$ and $\mathfrak{b}$ are two cardinals, the cardinal of the set of functions from $\mathfrak{b}$ to $\mathfrak{a}$ is denoted $\mathfrak{a}^{\mathfrak{b}}$, by abuse of notations[2]. Proposition 9 [3, p. 163] says that we can replace $\mathfrak{a}$ and $\mathfrak{b}$ by equipotent sets.

```
Definition card_pow a b := cardinal (set_of_functions b a).
Lemma card_pow_pr: forall a b a' b',
  equipotent a a' -> equipotent b b' ->
  card_pow a b = card_pow a' b'.
Theorem card_pow_pr1: forall x y,
  cardinal (set_of_functions y x) = card_pow (cardinal x) (cardinal y).
```

Proposition 10 [3, p. 163] says that if $\mathfrak{a}$ and $\mathfrak{b}$ are two cardinals, I is a set with cardinal $\mathfrak{b}$ and $\mathfrak{a}_\iota$ is the constant family $\mathfrak{a}$, then $\mathfrak{a}^{\mathfrak{b}}=\prod_{\iota\in I}\mathfrak{a}_\iota$. This is a trivial consequence of the fact that the set of functions and the set of graphs of functions are equipotent. Note: we do not need $\mathfrak{a}$ and $\mathfrak{b}$ to be cardinals. Note also that $\mathrm{Card}(I)=\mathfrak{b}$ implies that $\mathfrak{b}$ is a cardinal, hence, we give also another version of the theorem.

A consequence is that, if $\mathfrak{a}$ and $\mathfrak{b}$ are cardinals, $(\mathfrak{a}_\iota)_{\iota\in I}$ and $(\mathfrak{b}_\iota)_{\iota\in I}$ are families of cardinals we have

$$\mathfrak{a}^{\sum_{\iota\in I}\mathfrak{b}_\iota}=\prod_{\iota\in I}\mathfrak{a}^{\mathfrak{b}_\iota},\qquad\Big(\prod_{\iota\in I}\mathfrak{a}_\iota\Big)^{\mathfrak{b}}=\prod_{\iota\in I}\mathfrak{a}_\iota^{\mathfrak{b}}. \tag{9}$$

[2]The trouble seems to be that 4^2 and 2^4 denote the set of graphs of mappings from 2 to 4 or from 4 to 2; these sets are obviously distinct, but have the same number of elements; hence $4^2=2^4$ is true with these new notations, see page 77

The proof does not make use of the fact that the sets are cardinals, so we dropped the assumption. The proof of the first formula is as follows. Let $\mathfrak{a}_\iota = \mathfrak{a}$. We have $\mathfrak{a}^{\sum_{\iota\in I}\mathfrak{b}_\iota} = \prod_J \mathfrak{a}_\iota$, where J is any set whose cardinal is $\sum_{\iota\in I}\mathfrak{b}_\iota$; we chose the disjoint union of the sets $\mathfrak{b}_\iota$. We have a natural partition of J and we can apply the associativity of the product. For the second formula, let $\mathfrak{a}_{\iota\beta} = \mathfrak{a}_\iota$ for $\iota \in I$ and $\beta \in \mathfrak{b}$. This is a family defined on the product $I \times \mathfrak{b}$, for which we have have two natural partitions (all elements with the same ι, or all elements with the same β); we can apply associativity of the product twice. We also have

$$\mathfrak{a}^{\mathfrak{b}+\mathfrak{c}} = \mathfrak{a}^{\mathfrak{b}}\mathfrak{a}^{\mathfrak{c}}, \qquad (\mathfrak{a}\mathfrak{b})^{\mathfrak{c}} = \mathfrak{a}^{\mathfrak{c}}\mathfrak{b}^{\mathfrak{c}}, \qquad \mathfrak{a}^{\mathfrak{b}\mathfrak{c}} = (\mathfrak{a}^{\mathfrak{b}})^{\mathfrak{c}}. \tag{10}$$

```
Lemma card_pow_pr2: forall a b,
  cardinal_prod (cst_graph b a) = card_pow a b.
Theorem card_pow_pr3: forall a b j, cardinal j = b ->
  cardinal_prod (cst_graph j a) = card_pow a b.
Lemma power_of_sum: forall a f, fgraph f ->
  card_pow a (cardinal_sum f) =
  cardinal_prod (L (domain f) (fun i => card_pow a (V i f))).  (* 27 *)
Lemma power_of_prod: forall b f, fgraph f ->
  card_pow (cardinal_prod f) b =
  cardinal_prod (L (domain f) (fun i => card_pow (V i f) b)). (* 72 *)
Lemma power_of_sum2: forall a b c,
  card_pow a (card_plus b c) =
  card_mult (card_pow a b) (card_pow a c).
Lemma power_of_prod2: forall a b c,
  card_pow (card_mult a b) c =
  card_mult (card_pow a c) (card_pow b c).
Lemma power_of_prod3: forall a b c,
  card_pow a (card_mult b c)  =
  card_pow (card_pow a b) c.
```

Proposition 11 [3, p. 164] states that

$$\mathfrak{a}^0 = 1, \qquad \mathfrak{a}^1 = \mathfrak{a}, \qquad 1^{\mathfrak{a}} = 1, \qquad 0^{\mathfrak{b}} = 0 \quad (\mathfrak{b} \neq 0). \tag{11}$$

The Bourbaki proof is the following. We want to compute the number of functions from F to E is some cases. If F is empty, there is only the empty function; if F is a singleton then E^F are E equipotent (the bijection is *product1_canon*), if E has a single element, there is only one function, a constant; finally if the source is non-empty and the target is empty, there is no function. We use different properties. In the first two cases, we replace the power by a product whose index set has 0 or 1 element, and simplifiy the result. In the third case we rewrite 1 as a product whose index set is empty, and use distributivity (9b).

```
Lemma power_x_0: forall a, card_pow a card_zero =  card_one.
Lemma power_0_0: card_pow card_zero card_zero =  card_one.
Lemma power_x_1c: forall a, card_pow a card_one = cardinal a.
Lemma power_x_1: forall a, is_cardinal a -> card_pow a card_one = a.
Lemma power_1_x: forall a, card_pow  card_one  a = card_one.
Lemma power_0_x: forall a, a <> card_zero ->
  card_pow  card_zero  a = card_zero.
```

We have $1 + 1 = 2$, our proof uses formula (8.b), where I is the canonical doubleton. A consequence is $\mathfrak{a}^2 = \mathfrak{a}.\mathfrak{a}$.

```
Lemma card_two_pr: card_two = card_plus card_one card_one.
Lemma power_x_2: forall a, is_cardinal a ->
   card_pow a card_two = card_mult a a.
```

The final result in this section is Proposition 12 [3, p. 164], it states that the cardinal of the power set of X is 2^X (for each subset Y of X, we can consider the characteristic function, whose value is 0 on Y and 1 elsewhere).

```
Lemma card_powerset: forall X,
  cardinal (powerset X) = card_pow card_two X. (* 73 *)
```

4.6 Order relation and operations on cardinals

Proposition 13 [3, p. 164] states that $\mathfrak{a} \geq \mathfrak{b}$ if and only if there exists $\mathfrak{c}$ such that $\mathfrak{a} = \mathfrak{b} + \mathfrak{c}$ (for simplicity, we assume all three quantities to be cardinals, although $\mathfrak{c}$ could be any set, and both relations $\mathfrak{a} \geq \mathfrak{b}$ and $\mathfrak{a} = \mathfrak{b} + \mathfrak{c}$ imply that $\mathfrak{a}$ is a cardinal).

Proof. Assume B equipotent to a subset X of A and let C be the complement. Then $B + C$ is equipotent to $B_1 \cup C_2$ (where $B_1 = B \times \{1\}$). We can replace B by X, then omit the indices, so that $B + C$ is equipotent to A. Conversely, if A is equipotent $B_1 \cup C_2$, there is a bijection $B_1 \cup C_2 \to A$, and by restriction, an injection $B_1 \to A$, and by composition, an injection $B \to A$.

```
Theorem cardinal_le_when_complement: forall a b, (* 47 *)
  is_cardinal a -> is_cardinal b ->
  (cardinal_le b a) = (exists c, is_cardinal c & card_plus b c = a).
```

Proposition 14 [3, p. 165] says that if $\mathfrak{a}_\iota \geq \mathfrak{b}_\iota$ (for two families of cardinals) we have

$$\sum_{\iota \in I} \mathfrak{a}_\iota \geq \sum_{\iota \in I} \mathfrak{b}_\iota, \qquad \prod_{\iota \in I} \mathfrak{a}_\iota \geq \prod_{\iota \in I} \mathfrak{b}_\iota \tag{12}$$

The first formula is shown as follows. We have a bijection from $\mathfrak{b}_\iota$ into a subset E_ι of $\mathfrak{a}_\iota$, hence a bijection from $\mathfrak{b}_\iota \times \{\iota\}$ into a subset $E_\iota \times \{\iota\}$ of $\mathfrak{a}_\iota \times \{\iota\}$. This gives a bijection from the disjoint union $\bigcup \mathfrak{b}_\iota \times \{\iota\}$ into a subset $\bigcup E_\iota \times \{\iota\}$ of $\bigcup \mathfrak{a}_\iota \times \{\iota\}$. The proof of the second formula is similar: we get a bijection from $\prod \mathfrak{b}_\iota$ into a subset $\prod E_\iota$ of $\prod \mathfrak{a}_\iota$. As a corollary, we obtain a smaller result if we restrict the domain of the sum or the product; in the case of a product, we assume all factors distinct (proof: missing terms are replaced by zero, or one). The power is increasing with respect to both arguments.

```
Theorem sum_increasing: forall f g,
  fgraph f -> fgraph g -> domain f = domain g ->
  (forall x, inc x (domain f) -> cardinal_le (V x f) (V x g)) ->
  cardinal_le (cardinal_sum f) (cardinal_sum g).  (* 33 *)
Theorem product_increasing: forall f g,
  fgraph f -> fgraph g -> domain f = domain g ->
  (forall x, inc x (domain f) -> cardinal_le (V x f) (V x g)) ->
  cardinal_le (cardinal_prod f) (cardinal_prod g).

Lemma sum_increasing1: forall f j, fgraph f ->
  (forall x, inc x (domain f) -> is_cardinal (V x f)) ->
  sub j (domain f) -> cardinal_le (cardinal_sum (restr f j)) (cardinal_sum f).
Lemma product_increasing1: forall f j, fgraph f ->
  (forall x, inc x (domain f) -> is_cardinal (V x f)) ->
  (forall x, inc x (domain f) -> V x f <> card_zero) ->
  sub j (domain f) -> cardinal_le (cardinal_prod (restr f j)) (cardinal_prod f).
Lemma sum_increasing2: forall a b a' b',
  cardinal_le a a' -> cardinal_le b b' ->
```

```
  cardinal_le (card_plus a b)  (card_plus a' b').
Lemma product_increasing2: forall a b a' b',
  cardinal_le a a' -> cardinal_le b b' ->
  cardinal_le (card_mult a b)  (card_mult a' b').
Lemma sum_increasing3: forall a b, is_cardinal a ->is_cardinal b ->
  cardinal_le a (card_plus a b).
Lemma product_increasing3: forall a b, is_cardinal a ->is_cardinal b ->
  b <> card_zero ->
  cardinal_le a (card_mult a b).
Lemma power_increasing1 : forall a b a' b',
  a <> card_zero -> cardinal_le a a' -> cardinal_le b b' ->
  cardinal_le (card_pow a b)  (card_pow a' b').
```

To conclude this chapter, we prove Cantor's theorem (Theorem 2, [3, p. 165]) stating that $2^{\mathfrak{a}} > \mathfrak{a}$ for every cardinal $\mathfrak{a}$, so that there is no set containing all cardinals.

```
Theorem cantor: forall a, is_cardinal a ->
  cardinal_lt a (card_pow card_two a).
Lemma cantor_bis: ~ (exists a, forall x, is_cardinal x -> inc x a).
```

Chapter 5

Natural integers. Finite sets

Bourbaki makes a distinction between finite and infinite cardinals. Finite cardinals are identified with *natural integers*, which are entities satisfying some arithmetic properties (addition, multiplication, subtraction and division are studied in the next chapter) derived from an induction principle and a successor function. There is a set **N** containing all finite cardinals, so that we have statements of the form: if $n \in \mathbf{N}$ then $n \neq n+1$, instead of: if $\mathfrak{a}$ is an infinite cardinal then $\mathfrak{a} = \mathfrak{a}+1$. We end this chapter by studying the von Neumann ordinals; since the finite cardinals are the finite ordinals, this gives an explicit form for integers (for instance 2 is $\{\emptyset,\{\emptyset\}\}$). This form is however not adapted to computations. In the Bourbaki theory, a cardinal is a set: one could try to prove $1+1=2$ by showing that $x \in 1+1$ is equivalent to $x \in 2$. However; since 2 is constructed via the axiom of choice, we know that 2 is a set with two distinct elements, but $1 \in 2$ is unprovable. For this reason, natural integers are considerd as urelements (objects that may appear at the right-hand side of $\in$, but never the left-hand-side).

Integers are presented in [6] as follows. There is a symbol O and a symbol S, and two operations $a+b$ and $a \cdot b$ (sum and product), defined on integers, which are a finite (maybe empty) sequences of letters S followed by a single O. The five axioms are

Axiom 1 $\forall a, \mathrm{S}a \neq \mathrm{O}$.
Axiom 2 $\forall a, a+\mathrm{O} = a$.
Axiom 3 $\forall a \forall b, a+\mathrm{S}b = \mathrm{S}(a+b)$.
Axiom 4 $\forall a, a \cdot \mathrm{O} = \mathrm{O}$.
Axiom 5 $\forall a \forall b, a \cdot \mathrm{S}b = (a \cdot b)+a$.

The first axiom has an unusual form, since most axioms are of the form $a \implies b = c$. This axiom is built-in in Coq: an object of a type with n constructors is defined by a single constructor: an integer is either O or Sa, but nor both. This means that if c an integer, one and only one of axioms 2 and 3 apply to $a+c$.

The axiom implies injectivity of S. Note that Coq defines addition an multiplication by induction on the first argument.

In the system presented above, it is impossible to prove $\forall a, a = \mathrm{O}+a$, although the result is obvious for any a. Thus a new principle is needed. It says something like: "If all the strings in a pyramidal family are theorems, then so is the universally quantified string which summarizes them". (We get a pyramid if we center the statements $a = \mathrm{O}+a$, for consecutive values of a). The whole pyramid has an infinite number of statements, and proving it requires an infinite proof. Assume that each line can be shown from the previous one, using exactly the same argument. Then the proof has that form P and Q and Q and Q, etc. It is infinite, but not too much, hence is accepted. The induction principle is: "Suppose u is a variable,

and X{u} is a well-formed formula in which u occurs free. If both $\forall u : \langle X\{u\} \supset X\{S u/u\}\rangle$ and X{$0/u$} are theorems, then $\forall u : X\{u\}$ is also a theorem." This is built-in in Coq, under the form

```
nat_ind =
fun P : nat -> Prop => nat_rect P
     : forall P : nat -> Prop,
       P 0 -> (forall n : nat, P n -> P (S n)) -> forall n : nat, P n.
```

In this chapter we shall prove that the Bourbaki integers satisfy the induction principle, under the form

```
Lemma cardinal_c_induction: forall r:Set -> Prop,
  (r card_zero) -> (forall n, inc n  Bnat-> r n -> r (succ n))
  ->  (forall n, inc n Bnat -> r n).
```

and as a consequence, that all these definitions are essentially the same. The proof of the principle is as follows: the least element of the set (assumed non-empty) of elements not satisfying a property is either O or Sa. This is a consequence of the fact that **N** is well-ordered. Note that the property shown by induction (X, P, r, in the examples) is quantified in Coq, but neither in [6] nor in Bourbaki.

An important property of integers is the possibility of defining a function by induction. This is a Coq example

```
Fixpoint add (n m:nat) {struct n} : nat :=
   match n with
   | O => m
   | S p => S (add p m)
   end.
```

This definition says that the source is $\mathbb{N}\times\mathbb{N}$, induction is on n, and the result is of type $\mathbb{N}$. By induction, there is at most one function satisfying Axioms 2 and 3. In Bourbaki, one could define, for each m, a function $f_m : \mathbf{N} \to E_m$, which is the unique surjective function satisfying the two axioms, show that $E_m \subset \mathbf{N}$, extend the function $f'_m : \mathbf{N} \to \mathbf{N}$, then merge all these functions to get $f : \mathbf{N}\times\mathbf{N} \to \mathbf{N}$. This function cannot be defined without first showing the existence of the set of integers. It is however possible to prove by induction that addition satisfies the two axioms (if one replaces "$n \in \mathbf{N}$" by "n is a finite cardinal" in the induction principle). Obviously, definition by induction is a particular case of definition by transfinite induction. Exercise 2.17 defines sum and product, exercise 2.18 defines exponentiation of ordinals. The situation is simpler: since there is no set containing all ordinals, what has to be defined is not a function; it is called an "ordinal functional symbol", this is something that associates an ordinal to a pair of ordinals.

If we define '*succ x*' as $x+1$ for every cardinal x, we can define by induction the following function

```
Fixpoint nat_to_B (n:nat) :=
  match n with 0 => card_zero | S m => succ (nat_to_B m) end.
```

then define the set **N** as the image of this function. One can then show that **N** is the set of all finite integers. In a previous of this document, this isomorphism between $\mathbb{N}$ and **N** was used to convert theorems proved in the standard library of Coq into theorems about finite cardinals. This identification has been removed.

The function S on $\mathbb{N}$ has the following properties: it is injective; 0 is not in the range, and every non-zero element is in the range. Thus S is a bijection between $\mathbb{N}$ and $\mathbb{N}-\{0\}$. We use here two features of the axiom system of Carlos Simpson. The first is that there is a maping $\mathscr{R}$ such that $\mathscr{R}x$ is a set, for any natural number x. The second feature (General Scheme of Replacement) is that there is a set containg all $(\mathscr{R}x, \mathscr{R}\mathrm{S}x)$ for $x : \mathbb{N}$. This set is easily seen to be the graph of a bijection (in the Bourbaki sense) between $\mathbb{N}$ and $\mathbb{N}-\{0\}$. The function $\mathscr{R}$ and the General Scheme of Replacement have been used a lot in the first part of this report in order to prove existence of some sets (union, product, powerset, etc). We do not use it here except for a single purpose: to show that there exists an infinite set. This avoids introducing an axiom asserting the existence of an infinite set. Note also that applying the replacement scheme to the function *nat_to_B* yields a set containing zero and stable by *succ*; this set has to be infinite, by injectivity of *succ*. This is another way of proving existence of infinite sets, without using $\mathscr{R}$, but it presupposes that defining functions by induction in Coq is compabible with the Bourbaki theory.

5.1 Definition of integers

The quantity $\mathfrak{a}+1$ is called the *successor* of the cardinal $\mathfrak{a}$. A cardinal $\mathfrak{a}$ is called *finite* if it is unequal to is sucessor, it is called *infinite* otherwise. A set if called finite or infinite if its cardinal is finite or infinite respectively. A finite cardinal is also called an *integer*.

```
Definition succ x := card_plus x card_one.
Definition is_finite_c x :=  is_cardinal x & x <> succ x.
Definition is_finite_set x := is_finite_c (cardinal x).
Definition is_infinite_c a := is_cardinal a & ~ (is_finite_c a).
Definition infinite_set E := is_infinite_c (cardinal E).
```

If $\mathfrak{a}$ is finite, so is $\mathfrak{a}+1$, by injectivity of the successor function. The converse is obvious. This is Proposition 1 [3, p. 166].

```
Lemma integer_is_cardinal: forall x, is_finite_c x -> is_cardinal x.
Lemma succ_is_cardinal: forall a, is_cardinal (succ a).
Lemma cardinal_succ: forall a, cardinal (succ a) = succ a.
Lemma is_finite_succ1: forall x, is_finite_c x ->  is_finite_c (succ x).
Lemma cardinal_succ_pr0: forall a b, equipotent a b -> succ a = succ b.
Lemma succ_cardinal: forall z, succ (cardinal z) = succ z.

Theorem is_finite_succ: forall x,  is_cardinal x ->
  (is_finite_c x) = (is_finite_c (succ x)).
```

We show here that the successor of $\mathfrak{a}$ is the cardinal of the set obtained by adjoining some element. Let E be any set, $\mathrm{F} = \mathrm{E} - \{a\}$. The successor of the cardinal of F is then the cardinal of $\mathrm{F} \cup \{a\}$. This is E if $a \in \mathrm{E}$. We deduce that E is infinite if it is equipotent to F. Thus *nat* is an infinite set.

```
Lemma cardinal_succ_pr: forall a b, ~ (inc b a) ->
  cardinal (tack_on a b) = succ a.
Lemma cardinal_succ_pr1: forall a b,
  cardinal (tack_on (complement a (singleton b)) b) =
  succ (cardinal (complement a (singleton b))).
Lemma cardinal_succ_pr2: forall a b, inc b a ->
```

```
  cardinal a = succ (cardinal (complement a (singleton b))).
Lemma infinite_set_pr: forall a b, inc b a ->
  equipotent a  (complement a (singleton b)) ->
  infinite_set a.
Lemma nat_infinite_set: infinite_set nat.
```

5.2 Inequalities between integers

Proposition 2 [3, p. 166] says that if $\mathfrak{a}$ is a cardinal and n an integer, if $\mathfrak{a} \leq n$ then $\mathfrak{a}$ is an integer. If n is an integer and $n \neq 0$, then there is a unique integer m such that $n = m+1$. In this case $a < n$ is equivalent to $a \leq m$. If $\mathfrak{a}+\mathfrak{b}$ is an integer and $\mathfrak{a}$ is a cardinal, then $\mathfrak{a}$ is an integer.

```
Lemma cardinal_le_when_complement1: forall a b,
  cardinal_le b a -> (exists c, is_cardinal c & card_plus b c = a).
Lemma succ_injective: forall a b, is_cardinal a -> is_cardinal b ->
  succ a = succ b -> a = b.
Lemma is_less_than_succ: forall a, is_cardinal a ->
  cardinal_le a (succ a).
Lemma is_finite_in_sum: forall a b, is_cardinal b ->
  is_finite_c (card_plus a b) -> is_finite_c b.
Lemma is_finite_in_sum2: forall a b, is_cardinal a ->
  is_finite_c (card_plus a b) -> is_finite_c a.
Theorem le_int_is_int: forall a b, is_finite_c b ->
  cardinal_le a b -> is_finite_c a.
```

Since $\leq_{\mathrm{Card}}$ is a well-ordering, thus total, a finite cardinal is always smaller than an infinite cardinal. Fix some infinite cardinal $\mathfrak{b}$ (for instance the cardinal of $\mathbb{N}$), and consider the subset of $\{\mathfrak{a}, \mathfrak{a}$ is cardinal and $\mathfrak{a} \leq_{\mathrm{Card}} \mathfrak{b}\}$ formed of finite cardinals. It contains all finite cardinals. It is independent of $\mathfrak{b}$. It will be called the set of natural integers and denoted by *Bnat* or **N**.

```
Lemma finite_lt_infinite: forall a b,
  is_finite_c a -> is_infinite_c b -> cardinal_lt a b.
Lemma finite_le_infinite: forall a b,
  is_finite_c a -> is_infinite_c b -> cardinal_le a b.

Definition Bnat := Zo(set_of_cardinals_le (cardinal nat))
  (fun z => is_finite_c z).

Lemma inc_Bnat: forall a, inc a Bnat = is_finite_c a.
Opaque Bnat.
Lemma Bnat_is_cardinal: forall x, inc x Bnat -> is_cardinal x.
Lemma inc_Bnat_prop: forall a, inc a Bnat -> is_finite_c a.
```

We restate now some of the previous theorems.

```
Lemma Bnat_stable_succ: forall x, inc x Bnat -> inc (succ x) Bnat.
Lemma Bnat_stable_succ_bis: forall x,
  is_cardinal x -> inc (succ x) Bnat -> inc x Bnat.
Lemma Bnat_in_sum: forall a b, is_cardinal b ->
  inc (card_plus a b) Bnat ->inc b Bnat.
Lemma Bnat_in_sum2: forall a b, is_cardinal a ->
  inc (card_plus a b) Bnat ->inc a Bnat.
Lemma le_int_in_Bnat: forall a b, cardinal_le a b -> inc b Bnat ->  inc a Bnat.
```

The successor of zero is one, the successor of one is two. The successor of two is three and the successor of three is fiur by definitions. These quantites are integers.

```
Definition card_three := succ card_two.
Definition card_four := succ card_three.

Lemma succ_zero: succ card_zero  = card_one.
Lemma succ_one: succ card_one  = card_two.

Lemma inc0_Bnat: inc card_zero Bnat.
Lemma inc1_Bnat: inc card_one Bnat.
Lemma inc2_Bnat: inc card_two Bnat.
Lemma inc3_Bnat: inc card_three Bnat.
Lemma inc4_Bnat: inc card_four Bnat.
```

We show here that $2n = n + n$, hence $2 + 2 = 2.2 = 4$ and $2^4 = 4^2$.

```
Lemma two_plus_two: card_plus card_two card_two = card_four.
Lemma two_times_n: forall n, card_mult card_two n = card_plus n n.
Lemma two_times_two: card_mult card_two card_two = card_four.
Lemma power_2_4: card_pow card_two card_four = card_pow card_four card_two.
```

For each non-zero cardinal $\mathfrak{a}$, there is a unique cardinal $\mathfrak{b}$ such that $\mathfrak{a} = \mathfrak{b} + 1$. It is called the *predecessor* of $\mathfrak{a}$. Unqueness follows from injectivity of the successot function. An infinite cardinal is its own predecessor, this explains why we concentrate on finite cardinals. We shall denote it by $a - 1$ (subtraction will be defined later). As a consequence, whenever $\mathfrak{a}$ and $\mathfrak{b}$ are cardinals, the latter being non-zero, there exists $\mathfrak{c}$ such that $\mathfrak{a}\mathfrak{b} = \mathfrak{a}\mathfrak{c} + \mathfrak{a}$. If this expression is finite, so is $\mathfrak{a}$.

```
Definition predc n := choose (fun m => is_cardinal m & n = succ m).

Lemma predc_pr0: forall n, is_cardinal n -> n <> card_zero ->
  (is_cardinal (predc n) & n = succ (predc n)).
Lemma predc_pr: forall n, inc n Bnat -> n <> card_zero ->
  (inc (predc n) Bnat &  n = succ (predc n)).
Theorem exists_predc: forall n, inc n Bnat -> n <> card_zero ->
  exists_unique (fun m => inc m Bnat & n = succ m).
Lemma Bnat_in_product: forall a b, is_cardinal a -> is_cardinal b ->
  b <> card_zero -> inc (card_mult a b) Bnat -> inc a Bnat.
```

We have $a < b$ iff $a + 1 \leq b$ and similar relations.

```
Theorem lt_is_le_succ: forall a n, inc n Bnat ->
  cardinal_lt a (succ n) = cardinal_le a n.
Lemma is_lt_succ: forall n, inc n Bnat ->
  cardinal_lt n (succ n).
Lemma lt_is_le_succ1: forall a b, inc b Bnat ->
  cardinal_le a (succ b) -> a <> (succ b) -> cardinal_le a b.
Lemma lt_n_succ_le1: forall a b, is_cardinal a -> inc b Bnat ->
  cardinal_le a b = cardinal_le (succ a) (succ b).
Lemma succ_nonzero1:forall n, cardinal_le card_one (succ n).
Lemma succ_nonzero:forall n, succ n <> card_zero.
Lemma predc_pr1: forall n, is_cardinal n -> predc (succ n) = n.
Lemma predc_pr2: forall n, inc n Bnat -> predc (succ n) = n.
Lemma predc_pr3: forall n, inc n Bnat ->
```

```
  n = card_zero \/ exists m, inc m Bnat & n = succ m.
Lemma succ_positive: forall a, cardinal_lt card_zero (succ a).
Lemma zero_lt_one: cardinal_lt card_zero card_one.
Lemma zero_le_one: cardinal_lt card_zero card_one.
Lemma one_small_cardinal2: forall a, is_cardinal a ->
  cardinal_le card_one a = (a <> card_zero).
Lemma lt_n_succ_le0: forall a b, is_cardinal a -> is_finite_c b ->
  cardinal_le (succ a) b = cardinal_lt a b.
```

As a corollary, every subset of a finite set is finite. If $X \subset Y$, $X \neq Y$ and Y is finite, then Card(X) < Card(Y). Bourbaki says that the converse is true by definition. In fact, if Y is empty, it is finite, otherwise there is $x \in Y$, and if X is the complement of $\{x\}$ in Y, then Card(Y) = Card(X) + 1. The relation Card(X) < Card(Y) implies that Card(X) is finite, hence Card(X) + 1 is also finite.

```
Lemma sub_finite_set: forall x y, sub x y -> is_finite_set y ->
  is_finite_set x.
Lemma cardinal_succ_pr: forall a b, ~ (inc b a) ->
  cardinal (tack_on a b) = succ a.
Lemma cardinal_succ_pr1: forall a b,
  cardinal (tack_on (complement a (singleton b)) b) =
  succ (cardinal (complement a (singleton b))).
Lemma strict_sub_smaller: forall x y, sub x y -> is_finite_set y ->
  x<> y -> cardinal_lt (cardinal x) (cardinal y).
Lemma emptyset_finite: is_finite_set (emptyset).
Lemma strict_sub_smaller1: forall y,
  (forall x, sub x y -> x<> y -> cardinal_lt (cardinal x) (cardinal y)) ->
  is_finite_set y.
```

The image of a finite set by a function is finite. We give some variants of this property. Consider two sets E and F with the same cardinal, and a function $f : E \to F$. Assume E finite. If f is injective and not surjective, then $f(E)$ is a strict subset of F equipotent to F, thus cannot be finite. Hence we claim: if f injective, then f is bijective. Assume f surjective. It has a right inverse, which is injective, so that f is also bijective.

```
Lemma finite_image: forall f, is_function f -> is_finite_set (source f) ->
  is_finite_set (image_of_fun f).
Lemma finite_image_by: forall f A, is_function f ->  sub A (source f) ->
  is_finite_set A -> is_finite_set (image_by_fun f A).
Lemma finite_fun_image: forall a f, is_finite_set a ->
  is_finite_set  (fun_image a f).
Lemma finite_range: forall f, fgraph f -> is_finite_set(domain f) ->
  is_finite_set(range f).
Lemma finite_graph_domain: forall f, fgraph f ->
  is_finite_set f -> is_finite_set (domain f).
Lemma finite_graph_range: forall f, fgraph f ->
  is_finite_set f -> is_finite_set (range f).
Lemma finite_domain_graph: forall f, fgraph f ->
  is_finite_set (domain f) -> is_finite_set f.
Lemma bijective_if_same_finite_c_inj: forall f,
  cardinal (source f) = cardinal (target f) -> is_finite_set (source f) ->
  injective f -> bijective f.
Lemma sub_image_of_fun: forall f x, is_function f -> sub x (source f) ->
  sub (image_by_fun f x) (image_of_fun f).
Lemma bijective_if_same_finite_c_surj: forall f,
```

```
  cardinal (source f) = cardinal (target f) -> is_finite_set (source f) ->
  surjective f -> bijective f.
```

5.3 The set of natural integers

As explained as the start of the chapter, we introduced the set of integers as early as possible (Bourbaki introduced it very lately). This short section contain only one simple result, namely that **N** can be well-ordered by the "$x \in \mathbf{N}$ and $y \in \mathbf{N}$ and $x \leq_{\mathrm{Card}} y$" denoted $x \leq_{\mathbf{N}} y$. Thus every non-empty subset of **N** has a least element.

```
Definition Bnat_order := graph_on cardinal_le Bnat.
Definition Bnat_le x y := inc x Bnat & inc y Bnat & cardinal_le x y.
Definition Bnat_lt x y := Bnat_le x y & x<>y.

Lemma Bnat_order_substrate: substrate Bnat_order = Bnat.
Lemma Bnat_order_worder: worder Bnat_order.
Lemma Bnat_order_le: forall x y,
  gle Bnat_order x y = Bnat_le x y.

Lemma Bnat_wordered : forall X, sub X Bnat -> nonempty X ->
  inc card_zero X \/
  (exists a, inc a Bnat & inc (succ a) X & ~ (inc a X)).
```

We consider some properties of intervals (there will be more of them in the next chapter).

```
Lemma Bnat_interval_cc_pr: forall a b x, inc a Bnat -> inc b Bnat ->
  inc x (interval_cc Bnat_order a b) = (Bnat_le a x & Bnat_le x b).
Lemma Bnat_interval_co_pr: forall a b x, inc a Bnat -> inc b Bnat ->
  inc x (interval_co Bnat_order a b) = (Bnat_le a x & Bnat_lt x b).
Lemma Bnat_interval_cc_pr1: forall a b x, inc a Bnat -> inc b Bnat ->
  inc x (interval_cc Bnat_order a b) = (cardinal_le a x & cardinal_le x b).
Lemma Bnat_interval_co_pr1: forall a b x, inc a Bnat -> inc b Bnat ->
  inc x (interval_co Bnat_order a b) = (cardinal_le a x & cardinal_lt x b).
```

5.4 The principle of induction

Bourbaki states the principle of induction, the Criterion C61, in the following form: *Let* R$\{n\}$ *be a relation in a theory* $\mathcal{T}$ *(where* n *is not a constant of* $\mathcal{T}$*). Suppose that the relation*

$$\mathrm{R}\{0\} \text{ and } (\forall n)((n \text{ is an integer and } \mathrm{R}\{n\}) \implies \mathrm{R}\{n+1\})$$

is a theorem in $\mathcal{T}$*. Under these conditions, the relation*

$$(\forall n)((n \text{ is an integer }) \implies \mathrm{R}\{n\})$$

is a theorem in $\mathcal{T}$*.*

The proof is by contradiction. Assume the result false for some n, and consider the least element m of the set of all integers $\leq n$ that do not satisfy p, (it exists, since the set is non-empty and is well-ordered). Our proof is similar (with "the set of all integers $\leq n$ that do not satisfy p" replaced by "the set of integers that do not satisfy p").

We give four variants of the principle. Let S(n) be the relation: n is an integer and R is true for all integers $p < n$. If S implies R, then R is true for all integers.

If R(a) is true, and if R(n) together with $a \leq n$ (respectively $a \leq n < b$) implies R($n+1$), then for all n such that $a \leq n$ (respectively $a \leq n \leq b$), the relation R is true. In both cases n must be an integer (in the second case, we assume b integer, so that $n < b$ or $n \leq b$ implies that n is an integer). Bourbaki uses induction on $a \leq n < b \implies \mathrm{R}(n)$, but it is simpler to use $a \leq n \leq b \implies \mathrm{R}(n)$.

The last variant is: if $a \leq n < b$ and R($n+1$) implies R(n), if moreover R(b) is true, then $a \leq n \leq b$ implies R(n). The proof is by induction on P = ¬R. If R is false for some n with $a \leq n \leq b$ then P is true for some c with $a \leq c < b$. On the other hand, if $a \leq n < b$ then P(n) implies P($n+1$).

```
Lemma cardinal_c_induction: forall r:Set -> Prop,
  (r card_zero) -> (forall n, inc n  Bnat-> r n -> r (succ n))
  ->  (forall n, inc n Bnat -> r n).

Lemma cardinal_c_induction1: forall r:Set -> Prop,
  let s:= fun n => forall p, inc n Bnat -> inc p Bnat ->
    cardinal_lt p n -> r p in
  (forall n, inc n Bnat -> s n -> r n) ->
  (forall n, inc n Bnat -> r n).
Lemma cardinal_c_induction2:  forall (r:Set -> Prop) k,
  inc k Bnat -> r k ->
  (forall n, inc n Bnat -> cardinal_le k n -> r n -> r (succ n))
  ->  (forall n, inc  n Bnat  -> cardinal_le k n -> r n).
Lemma cardinal_c_induction3:  forall (r:Set -> Prop) a b,
  inc a  Bnat -> inc b Bnat ->  r a ->
  (forall n, cardinal_le a n -> cardinal_lt n b -> r n -> r (succ n))
  ->  (forall n, cardinal_le a n  -> cardinal_le n b -> r n).
Lemma cardinal_c_induction4:  forall (r:Set -> Prop) a b,
  inc a Bnat -> inc b Bnat ->  r b ->
  (forall n, cardinal_le a n -> cardinal_lt n b -> r (succ n) -> r n)
  ->  (forall n, cardinal_le a n  -> cardinal_le n b -> r n).
```

We rewrite our induction principle variants 3 and 4, replacing $a \leq n \leq b$ by $n \in [a, b]$.

```
Lemma cardinal_c_induction3_v:  forall (r:Set -> Prop) a b,
  inc a Bnat -> inc b Bnat ->  r a ->
  (forall n, inc n (interval_co Bnat_order a b) -> r n -> r (succ n))
  ->  (forall n, inc n (interval_cc Bnat_order a b) -> r n).
Lemma cardinal_c_induction4_v:  forall (r:Set -> Prop) a b,
  inc a Bnat -> inc b Bnat ->  r b ->
  (forall n, inc n (interval_co Bnat_order a b) -> r (succ n) -> r n)
  ->  (forall n, inc n (interval_cc Bnat_order a b) -> r n).
```

The empty set is finite, and if X is finite then X $\cup \{x\}$ is finite. We then show a partial converse, that will be useful for induction on finite sets. If X has cardinal zero, it is the empty set, and if X has cardinal $n+1$, it is of the form X$' \cup \{x\}$, where X$'$ has cardinal n.

```
Lemma tack_on_finite: forall X x,
  is_finite_set X -> is_finite_set(tack_on X x).
Lemma singleton_finite: forall x, is_finite_set(singleton x).
```

```
Lemma doubleton_finite: forall x y, is_finite_set(doubleton x y).
Lemma card_one_not_zero: card_one <> card_zero.
Lemma tack_if_succ_card: forall x n, is_cardinal n -> cardinal x = succ n ->
  exists u, exists v, x = tack_on u v & ~(inc v u) & cardinal u = n.
```

The induction principle on finite sets is now: If a property P is true for the empty set, if P(a) implies P($a \cup \{b\}$), then P is true for every finite set. In general P has the form: if A then B. Note: if $b \in a$, then $a \cup \{b\} = a$, and we have a version where we add the condition $b \notin a$.

```
Lemma finite_set_induction0: forall s:Set -> Prop,
  s emptyset -> (forall a b, s (a) ->  ~(inc b a) -> s (tack_on a b)) ->
  forall x, is_finite_set x -> s x.
Lemma finite_set_induction: forall s:Set -> Prop,
  s emptyset -> (forall a b, s (a) -> s (tack_on a b)) ->
  forall x, is_finite_set x -> s x.
Lemma finite_set_induction1: forall (A B:Set -> Prop) x,
  (A emptyset -> B emptyset)
  -> (forall a b, (A a -> B a) -> A(tack_on a b) -> B(tack_on a b))
  -> is_finite_set x -> A x -> B x.
```

In some cases P is false for the empty set. If P is true for all singletons, then P is true for every non-empty finite set.

```
Lemma finite_set_induction2: forall (A B:Set -> Prop) x,
  (forall a, A (singleton a) -> B (singleton a))
  -> (forall a b, (A a -> nonempty a -> B a) ->
    nonempty a -> A(tack_on a b) -> B(tack_on a b))
  -> is_finite_set x -> A x -> nonempty x -> B x.
```

The next lemmas correspond to Exercise 4.1. If we denote by $\mathfrak{F}(E)$ the set of finite subsets of E, and by $P(E,\mathfrak{G})$ the property: (i) $\emptyset \in \mathfrak{G}$; (ii) the relation $X \in \mathfrak{G}$ and $x \in E$ imply $X \cup \{x\} \in \mathfrak{G}$, then $P(E,\mathfrak{F})$ is true; by induction, $P(E,\mathfrak{G})$ implies $\mathfrak{F} \subset \mathfrak{G}$. As a consequence, the union of two finite sets is finite. The powerset of a finite set is finite (proof by induction: $\mathfrak{P}(X \cup \{x\})$ is the union of $\mathfrak{P}(X)$ and the image of $\mathfrak{P}(X)$ by the mapping $Y \mapsto (Y \cup \{x\})$; if $x \notin X$, the union is disjoint, and the cardinal is twice the cardinal of $\mathfrak{P}(X)$).

```
Definition set_of_finite_subsets E := Zo(powerset E)(fun X => is_finite_set X).
Definition set_of_finite_subsets_prop E F:=
  inc emptyset  F & forall x X, inc x E -> inc X F -> inc (tack_on X x) F.

Lemma set_of_finite_subsets_pr: forall E,
  set_of_finite_subsets_prop E (set_of_finite_subsets E) &
  (forall F, set_of_finite_subsets_prop E F -> sub (set_of_finite_subsets E) F)
Lemma set_of_finite_subsets_pr: forall E,
  set_of_finite_subsets_prop E (set_of_finite_subsets E) &
  (forall F, set_of_finite_subsets_prop E F -> sub (set_of_finite_subsets E) F).
Lemma finite_union2: forall x y, is_finite_set x -> is_finite_set y ->
  is_finite_set (union2 x y).
Lemma finite_powerset: forall x,
  is_finite_set x -> is_finite_set (powerset x).  (* 51 *)
```

If $s(x)$ is the successor of x, we have $a + s(b) = s(a + b)$ and $a \cdot s(b) = a \cdot b + a$, $a^{s(b)} = a^b.a$ We deduce by induction that **N** is stable by addition, multiplication and power.

```
Lemma plus_via_succ: forall a n,
  card_plus a (succ n) = succ (card_plus a n).
Lemma mult_via_plus: forall a b, is_cardinal a ->
  card_mult a (succ b) = card_plus (card_mult a b) a.
Lemma pow_succ: forall a b,
  card_pow a (succ b) = card_mult(card_pow a b) a.

Lemma Bnat_stable_plus: forall a b, inc a Bnat -> inc b Bnat ->
  inc (card_plus a b) Bnat.
Lemma Bnat_stable_mult: forall a b, inc a Bnat -> inc b Bnat ->
  inc (card_mult a b) Bnat.
Lemma Bnat_stable_pow: forall a b, inc a Bnat -> inc b Bnat ->
  inc (card_pow a b) Bnat.
```

5.5 Finite subsets of ordered sets

Let $\leq$ be an order relation on a set E that makes it a directed set, a lattice, or a totally ordered set; and let X be a finite non-empty subset of E. Then X has an upper bound, or has a least upper bound and a greatest lower bound, or has a least and greatest element respectively (Proposition 3, [3, p. 170]). We have to show that there is an x such that P(x,X). By assumption, this is true if X is a doubleton (therefore, if X is a singleton). If X = Y $\cup$ $\{b\}$ and P(a,X) we have to show the property for the doubleton $\{a,b\}$.

```
Lemma finite_set_induction3: forall (p:Set -> Set -> Prop) E X,
  (forall a b, inc a E -> inc b E -> exists y, p (doubleton a b) y) ->
  (forall a b x y, sub a E -> inc b E -> p a x -> p (doubleton x b) y->
    p (tack_on a b) y) ->
  (forall X x, sub X E -> nonempty X -> p X x -> inc x E) ->
  nonempty X -> is_finite_set X -> sub X E -> exists x, p X x.

Lemma finite_subset_directed_bounded: forall r X,
  right_directed r ->is_finite_set X -> sub X (substrate r) -> nonempty X
  -> bounded_above r X.
Lemma finite_subset_lattice_inf: forall r X,
  lattice r ->is_finite_set X -> sub X (substrate r) -> nonempty X
  -> exists x, greatest_lower_bound r X x.  (* 20 *)
Lemma finite_subset_lattice_sup: forall r X,
  lattice r ->is_finite_set X -> sub X (substrate r) -> nonempty X
  -> exists x, least_upper_bound r X x.  (* 20 *)
Lemma finite_subset_torder_greatest: forall r X,
  total_order r ->is_finite_set X -> sub X (substrate r) -> nonempty X
  -> exists x, greatest_element (induced_order r X) x. (* 25 *)
Lemma finite_subset_torder_least: forall r X,
  total_order r ->is_finite_set X -> sub X (substrate r) -> nonempty X
  -> exists x, least_element (induced_order r X) x.
```

Some consequences. A nonempty finite set (the word "nonempty" is missing in Bourbaki) has a maximal element, and if totally ordered, has a greatest element. A finite totally ordered set is well-ordered.

```
Lemma finite_set_torder_greatest: forall r,
  total_order r ->is_finite_set (substrate r) -> nonempty (substrate r)
  -> exists x, greatest_element r x.
```

```
Lemma finite_set_torder_worder: forall r,
  total_order r ->is_finite_set (substrate r) -> worder r.
Lemma finite_set_maximal: forall r,
  order r ->is_finite_set (substrate r) -> nonempty (substrate r) ->
  exists x, maximal_element r x.
```

5.6 Properties of finite character

If E is a set, a property P{X} (where X is a subset of E) is said to be of *finite character* if the set $\mathfrak{S}$ of all X satisfying P is of finite character; this means $X \in \mathfrak{S}$ if and only if every finite subset Y of X satisfies $Y \in \mathfrak{S}$. Example: the set of totally ordered subsets of an ordered set. Theorem 1 [3, p. 171] states: Every set $\mathfrak{S}$ of subsets of a set E which is of finite character has a maximal element (when ordered by inclusion). The word "nonempty" is missing: if $X \in \mathfrak{S}$, then $\emptyset$ is a finite subset of X; hence $\emptyset \in \mathfrak{S}$. But the empty set is of finite character.

```
Definition of_finite_character s:=
  forall x, (inc x s) = (forall y, (sub y x & is_finite_set y) -> inc y s).

Lemma of_finite_character_example: forall r, order r ->
  of_finite_character(Zo (powerset (substrate r)) (fun z =>
    total_order (induced_order r z))).
Lemma maximal_inclusion: forall s, of_finite_character s -> nonempty s ->
  exists x, maximal_element (inclusion_suborder s) x. (* 39 *)
```

5.7 Ordinals

Let $\mathrm{Is}(\Gamma, \Gamma')$ be the property that Γ and Γ' are ordered sets, and there is an order isomorphism between Γ and Γ'. This is an equivalence relation (without graph). The quantity $\tau_\Delta(\mathrm{Is}(\Gamma, \Delta))$ will be denoted by $\mathrm{Ord}(\Gamma)$ and is called the *order-type* of Γ (compare with Eq, Card, and cardinal, at the start of the previous Chapter). The ordinal sum and lexicographic product of orderings induce two operations (sum and product) on the family of order types, denoted by $\lambda + \mu$ and $\lambda\mu$. These operations are non-commutative: for instance $\lambda + 1$ and $1 + \lambda$ correspond to the orderings obtained by adjoining a greatest and a least element, respectively. The relation "there is an order isomorphism between Γ and a sub-ordereing of Γ'", denoted by $\Gamma \prec \Gamma'$ is a preorder.

Bourbaki defines an *ordinal* as the order type of a well-ordered set. The sum and product of ordinals are ordinals, and the relation $\lambda \prec \mu$ is a well-ordering, compatible with the two operations; we shall study ordinals in Chapter 8. In Exercise 20 section 2, Bourbaki considers some sets called "pseudo-ordinal", satisfying some properties; in particular these sets E are well-ordered by some natural order $o(\mathrm{E})$ and satisfy:

(OP) Any well-ordered set is uniquely isomorphic to some $o(\mathrm{E})$.

This set E (or its ordering $o(\mathrm{E})$) may be called the ordinal of the well-ordering. Its existence is obvious by transfinite induction; we shall also give von Neumann's proof, which does not use tranfinite induction, but the axiom of separation (see Part I of this report for details).

We say that a set X is *transitive* if $a \in b$ and $b \in X$ implies $a \in X$ (this is the same as: $b \in X \implies b \subset X$). We say that X is *irreflexive* if $X \notin X$. We say that $<$ is *asymmetric* if at least one of $x < y$, $y < x$ is false. We say that E is asymmetric if the relation "$x \in E$ and $y \in E$ and

$x \in y$" is asymmetric. A set whose elements are all irreflexive is called *decent* (all asymmetric sets are decent). A strict well-ordering is an asymmetric relation $a < b$ such that "$a < b$ or $a = b$" is a well-ordering.

Consider the following properties of a set E.

(1) $\emptyset \in \mathrm{E}$.

(2) E is transitive.

(3) The relation "$x \in \mathrm{E}$ and $y \in \mathrm{E}$ and ($x = y$ or $x \in y$)" is a well-ordering of E.

(4) The relation "$x \in \mathrm{E}$ and $y \in \mathrm{E}$ and $x \subset y$" is a well-ordering of E.

(5) The relation "$x \in \mathrm{E}$ and $y \in \mathrm{E}$ and $x \in y$" is a strict well-ordering of E.

(6) E is asymmetric for $\in$.

(7) $\forall x \in \mathrm{E}, x =]\leftarrow, x[$.

(8) Every transitive subset of E is E or an element of E.

(9) $\forall x \in \mathrm{E}, \forall y \in \mathrm{E} \implies$ exactly one of $x \in y, y \in x, x = y$.

(10) $\forall x \in \mathrm{E}, f(x) = f\langle]\leftarrow, x[\rangle$.

In this section, we shall study pseudo-ordinals, but call them "ordinals" for simplicity. Different variants can be found in the litterature. The 1956 Edition of Bourbaki considered (1), (2) and (3), the current edition uses (8); the most common definition is (2) and (5) (for instance [7, 1]. The von Neumann definition (see [9]) uses (4) and (7).

Let's start with a few comments. Condition (6) say that E is decent, condition (5) is equivalent to (3)(6), it implies (9). We shall see that the orderings defined by (3), (4), (5) are the same, they define the natural ordering $o(\mathrm{E})$ of relation (OP).

According to von Neumann, an ordinal is a well-ordered set satisfying (7), where $]\leftarrow, x[$ is the set of elements $y \in \mathrm{E}$ satisfying $y < x$. The relation $x =]\leftarrow, x[$ means: for any element x of E, the relation $y \in x$ is equivalent to $y \in \mathrm{E}$ and $y < x$. From $y \in \mathrm{E}$ we deduce that E is transitive. On the other hand, if x and y belong to E, $y \in x$ is equivalent to $y < x$, so that $a \leq b$ is equivalent to $a = b$ or $a \in b$. If $x \leq y$ we have $]\leftarrow, x[\subset]\leftarrow, y[$, thus $x \subset y$. Conversely, if $x \subset y$, since E is totally ordered, we have $x \leq y$ or $y \leq x$. In the second case we have $y \subset x$ thus $x = y$ by extensionality. Thus we have shown: $a \leq b$ if and only if $a \subset b$. As a consequence, the ordering of E satisfies (3) and (4). It satisfies (5) since $x \notin x$ is a consequence of $x \notin]\leftarrow, x[$. A *numeration* of an ordered set E is function f satisfying (10). This means that $f(x)$ is the set of all $f(y)$ for $y < x$. Relation (7) says that the identity function is a numeration. By transfinite induction, every well-ordered set has a numeration; relation (OP) follows from the fact that the image of a numeration is an ordinal.

In the 1956 version, Bourbaki defined an ordinal via (1), (2), and (3). Consider a set W such that $\mathrm{W} = \{\emptyset, \mathrm{W}\}$. According to the axiom of foundation, no such set exists; according to AFA (anti-foundation axiom) there is a unique set satisfying this conditition. These two axioms being independent of the Bourbaki theory, whether or not W exists is undecidable. This set is not decent, but it satisfies (1), (2) and (3), thus was an ordinal according to the 1956 Edition of Bourbaki. It is isomorphic to the set $\{\emptyset, \{\emptyset\}\}$, contradicting uniqueness of (OP). According to (1), the empty set is not an ordinal, contradicting existence. Note that the least element of a non-empty ordinal cannot have elements (by transitivity), thus must be empty.

```
Definition transitive_set X:= forall x, inc x X -> sub x X.
```

```
Definition decent_set x := forall y, inc y x -> ~ (inc y y).
Definition elt_suborder E :=  graph_on (fun a b => inc a b \/ a = b) E.
Definition asymmetric_set E :=
  forall  x y, inc x E -> inc y E -> inc x y -> inc y x -> False.
Definition Bordinal X:= forall Y, sub Y X -> transitive_set Y ->
  Y <> X -> inc Y X.
```

We start with the definition of [7]. There are four conditions K_1, K_2, K_3 and K_4. Conditions K_1 and K_4 say that E is decent and transitive. Condition K_2 says that $x \in y$ is a transitive relation on E. We show here that K_1 and K_2 say that the relation "$x \in y$ or $x = y$" is an ordering on E. Condition K_3 will imply that $\in$ is a well-ordering.

```
Definition Kordinal a :=
  ( (forall  x y, inc x a -> inc y a -> inc x y -> inc y x -> False)
   & (forall x y z, inc x a -> inc y a -> inc z a ->
       inc x y -> inc y z -> inc x z)
   & (forall z, sub z a -> nonempty z ->
       exists x, (inc x z & forall y, inc y z -> inc x y \/ x = y))
   & (forall x y, inc x a -> inc y x -> inc y a)).

Lemma trans_sym_order: forall x,
  (forall y,inc y x ->  transitive_set y) -> asymmetric_set x ->
  (order (elt_suborder x) & substrate (elt_suborder x) = x). (* 18 *)
```

Condition K_3 implies that $x \in y$ is a strict total ordering (at least one of $x \in y$, $y \in x$ or $x = y$ is true). It implies that $x \in y$ is equivalent to $x \subsetneq y$, and that $x \subset y$ is equivalent to "$x \in y$ or $x = y$". Thus $\subset$ is a well-ordering.

```
Lemma Kordinal_asymmetric: forall E, Kordinal E -> asymmetric_set E.
Lemma Kordinal_decent: forall E, Kordinal E -> decent_set E.
Lemma Kordinal_irreflexive: forall E , Kordinal E -> inc E E -> False.
Lemma Kordinal_transitive: forall E, Kordinal E -> transitive_set E.
Lemma Kordinal_trichotomy: forall E x y, Kordinal E ->
  inc x E -> inc y E -> (inc x y \/ inc y x \/ x = y).
Lemma Kordinal_inclusion: forall E, Kordinal E ->
  forall x y, inc x E -> inc y E -> (inc x y = strict_sub x y).
Lemma Kordinal_inclusion1: forall E, Kordinal E ->
  forall x y, inc x E -> inc y E -> (sub x y = (inc x y \/ x = y)).
Lemma Kordinal_inclusion2:  forall E,
  Kordinal E -> worder (inclusion_suborder E).
```

We now show that K_3 implies that $\in$ is a well-ordering, so that the definition of [7] is equivalent to (2), (3) and (6), in other terms: E is transitive, asymmetric and is a well-ordering.

```
Lemma trans_sym_order: forall x,
  (forall y,inc y x ->  transitive_set y) -> asymmetric_set x ->
  (order (elt_suborder x) & substrate (elt_suborder x) = x).
Lemma Kordinal_elt1:  forall E,
  Kordinal E -> worder (elt_suborder E).
Lemma Kordinal_pr:  forall E,
  Kordinal E =
  (transitive_set E & worder (elt_suborder E) & asymmetric_set E). (* 24 *)
```

If the set E is ordered by $\in$, the segment $]\leftarrow, x[$ is the set of all elements y in E such that $y \in x$. This is the intersection of x and E. If E is transitive, this is x. In particular, if E is a K-ordinal, we have $x =]\leftarrow, x[$. The same is true if we consider the ordering $\subset$.

We deduce that K-ordinals satisfy the von Neumann condition (7). The converse is true by the argument explained above (relation (7) says that E is transitive and that $x \in y$ is equivalent to $x \subsetneq y$. This implies that E is asymmetric and that the ordering induced by $\subset$ is the same as that of $\in$).

```
Lemma Kordinal_segment1: forall E x, decent_set E ->
  inc x E -> segment (elt_suborder E) x = intersection2 x E.
Lemma Kordinal_segment2: forall E x, decent_set E -> transitive_set E->
  inc x E -> segment (elt_suborder E) x = x.
Lemma Kordinal_segment3: forall E x, Kordinal E ->
  inc x E -> segment (elt_suborder E) x = x.
Lemma Kordinal_segment4: forall E x, Kordinal E ->
  inc x E -> segment (inclusion_suborder E) x = x.

Lemma Kordinal_pr2:  forall E,
  Kordinal E =
    (worder (inclusion_suborder E) &
    (forall x, inc x E -> segment (inclusion_suborder E) x = x)).
```

Every element of a K-ordinal is transitive since we can replace $\in$ by $\subsetneq$ which is transitive. Every transitive subset of E is a K-ordinal (this comes directly from the definition; it is also a consequence of the fact that a subset of a well-ordered set is well-ordered). Thus, every element of a K-ordinal is a K-ordinal. We now state: (9) is true whenever x and y are K-ordinals (there is no need to add the restrictions $x \in$ E and $y \in$ E). In fact, the intersection of two ordinals is a segment of each one. Since the sets are well-ordered, a segment is the whose set or of the form $]\leftarrow, x[$, and we know that this is x. Hence we get $x \cap y = x$ or $x \cap y \in x$, and similarly, $x \cap y = y$ or $x \cap y \in y$; the conclusion follows since $x \cap y \notin x \cap y$. It follows that, if E is a K-ordinal, X a transitive strict subset of E, then X is a K-ordinal, and X $\in$ E, so that E is an ordinal in the Bourbaki sense.

```
Lemma Kordinal_sub_trans: forall E x, Kordinal E -> inc x E ->
  transitive_set x.
Lemma Kordinal_sub_ordinal: forall E x, Kordinal E -> sub x E ->
  transitive_set x -> Kordinal x.
Lemma Kordinal_inc_ordinal: forall E x, Kordinal E -> inc x E ->
  Kordinal x.
Lemma Kordinal_trichotomy1 : forall x y,
  Kordinal x -> Kordinal y ->
  (inc x y \/ inc y x \/ x = y). (* 29 *)
```

Let's now prove that any B-ordinal is a K-ordinal. This requires a bunch of lemmas. We follow exercise 20. The first results are trivial: let T(x) denote $x \cup \{x\}$; we call it the successor—of x. If E is transtive, so is T(E). If E is formed of transitive sets, then the union and intersection of E is transitive. The same is true if "transitive" is replaced by "decent". We shall also prove that the empty set is an ordinal, and that T(E) is an ordinal whenever E is one.

```
Lemma transitive_union: forall x, (forall y, inc y x -> transitive_set y)
  -> transitive_set (union x).
Lemma transitive_intersection: forall x, (forall y, inc y x -> transitive_set y)
  -> nonempty x -> transitive_set (intersection x).
Lemma decent_union: forall x, (forall y, inc y x -> decent_set y)
  -> decent_set (union x).
Lemma decent_intersection: forall x,
  (forall y, inc y x -> decent_set y)
```

```
  -> nonempty x -> decent_set (intersection x).
Lemma transitive_tack_on: forall y,
  transitive_set y -> transitive_set (tack_on y y).
Lemma decent_tack_on: forall y,
  decent_set y -> decent_set (tack_on y y).
Lemma Bordinal_tack_on: forall x,
  Bordinal x ->  Bordinal (tack_on x x).
Lemma Bordinal_emptyset:  Bordinal emptyset.
```

Let p the property that a set is transitive and decent, let E be any set and Y the set of all subsets of E that satisfy p and $t = \bigcup Y$. We know that t and $T(t)$ satisfy p. Assume $t \in E$. Then $T(t) \subset E$ and $T(t) \in y$. Since $t \in T(t)$, we get $t \in \bigcup Y$, which is $t \in t$, contradicting the fact that $T(t)$ is decent. Assume now that E is a B-ordinal. Since t is a transitive subset of E, we have either $t \in E$ or $t = E$; since the first case is excluded, we deduce that E is decent and transitive.

Consequences. Assume that x and y are two ordinals. The relation $x \in x$ contradicts the fact that x is decent. If $x \in y$ and $y \in x$, the $x \in x$ by transitivity, absurd. If $x \subset y$ then $x = y$ or $x \in y$ (this is true by definition if x is transitive and y and ordinal). Any element of y is a strict subset of y (it is a subset by transitivity, and we know $y \notin y$).

```
Lemma Bordinal_transitive_decent: forall x,
  Bordinal x -> (transitive_set x & decent_set x).
Lemma Bordinal_transitive: forall x,
  Bordinal x -> transitive_set x.
Lemma Bordinal_irreflexive: forall x, Bordinal x -> ~ (inc x x).
Lemma Bordinal_asymmetric1: forall x y,
  Bordinal x -> Bordinal y -> inc x y -> inc y x -> False.
Lemma Bordinal_sub1: forall x y,
  transitive_set x -> Bordinal y -> sub x y -> x= y \/ inc x y.
Lemma Bordinal_sub: forall x y,
  Bordinal x -> Bordinal y -> sub x y ->
  x= y \/ inc x y.
Lemma Bordinal_sub2: forall x y, Bordinal y ->
  inc x y -> strict_sub x y.
```

Consider a family X of ordinals, and their intersection Y. This is a transitive and decent set. The relation $Y \notin Y$ says that for some $A \in X$ we have $Y \notin A$. Since Y is a transitive subset of A we get Y = A. We restate this as $Y \in E$. In the case where X has two elements, x and y, this says that $x \cap y$ is either x or y, or equivalently that $x \subset y$ or $y \subset x$. It folllows one of $x \in y$, $y \in x$, $x = y$. A consequence is that the empty set (which is an ordinal) satisfies $\emptyset \in x$ or $\emptyset = x$.

```
Lemma intersection_of_ordinals: forall x, nonempty x ->
  (forall a, inc a x -> Bordinal a) -> inc (intersection x) x.
Lemma Bordinal_trichotomy : forall x y,
  Bordinal x -> Bordinal y ->
  (inc x y \/ inc y x \/ x = y).
Lemma empty_in_ordinal: forall x, Nordinal x ->
  x = emptyset \/ inc emptyset x.
```

We show here: a transitive set X whose elements are ordinals is an ordinal and the elements of an ordinal are ordinals. Claim 1: Let Y be a transitive strict subset of X, a an element of X not in Y, and $t \in Y$. Since a and t are in X, hence ordinals, we have $t = a$, $a \in t$, or $t \in a$. The first case is excluded, since it says $a \in Y$, the second case is excluded (by transitivity of Y, it also says $a \in Y$). Thus we have $t \in a$. This says $Y \subset a$. Since Y is transitive, we get $Y = a$

or Y ∈ a. This implies Y ∈ X (in the second case, we use transitivity of X). QED. Claim 2: Let X be an ordinal, and Z the union of all subsets Y of X such that Y is transitive and contains only ordinals. This set is transitive and contains only ordinals, thus is an ordinal, and we have one of Z = X, Z ∈ X or X ∈ Z. The first alternative is our result, the last implies x ∈ X, absurd. Assume then Z ∈ X. This is absurd, using the same argument as for proving that an ordinal is transitive.

Consequences: an ordinal is asymmetric, and ∈ is an ordering on E. It is a well-ordereing since the intersection of a family of ordinals is the least ordinal. Thus, the Bourbaki definition coincides with the two other definitions.

Assume that E is a set containing all ordinals. If $a \in b$ and $b \in$ E, then a is an ordinal, thus $a \in$ E. This shows that E is a transitive set, containing only ordinals, so is an ordinal and E ∈ E, absurd. Thus, there is no set containing all ordinals.

```
Lemma Bordinal_pr: forall x, transitive_set x ->
  (forall y, inc y x -> Bordinal y) ->
  Bordinal x.
Lemma elt_of_ordinal: forall x y, Bordinal x -> inc y x ->
  Bordinal y.
Lemma Bordinal_asymmetric: forall E, Bordinal E -> asymmetric_set E.
Lemma Bordinal_worder: forall x, Bordinal x ->
  worder (elt_suborder x).  (* 28 *)
Lemma Kordinal_pr3:  forall E,
  Kordinal E = Bordinal E.
Lemma non_collectivizing_ordinal:
  ~(exists x, forall a, Bordinal a -> inc a x).
```

Let's show that two isomorphic ordinals are equal. We consider two ordinals X and Y and a bijection f. Being an isomorphism, it satisfies $f(a) \subset f(b)$ if and only if $a \subset b$. Since elements of X and Y are ordinals, we can restate this as $f(a) \in f(b)$ if and only if $a \in b$. Assume $b \in$ X and $a \in f(b)$. By transitivity of Y, $a \in$ Y, hence $a = f(c)$ for some c; from $f(c) \in f(b)$ we deduce $c \in b$.

Let A be the set of elements a such that $f(a) \neq a$. Assume A non-empty, and let b be the least element. If $a \in b$, then $a \notin$ A, thus $f(a) = a$; on the other hand $f(b) \neq b$. If $a \in f(b)$, there is $c \in b$ such that $a = f(c) = c$, thus $a \in b$ and $b \subset f(b)$. This says $b = f(b)$ or $b \in f(b)$; the first alternative is excluded. The second says $f(f(b)) = f(b)$, thus $f(b) = b$ by injectivity. This is absurd, thus A is empty, f is the identity function, and X = Y.

We now prove the converse, namely that every well-ordered set is uniquely isomorphic to an ordinal. We shall give two proofs. We start with the short one. Consider a well-ordered set E, and let f be the function defined by transfinite induction via *target*. This means that, for every $x \in$ E, $f(x)$ is the target of the restriction of f to the segment $]\leftarrow, x[$; by definition of the restriction, this is the set of values $f(y)$ for $y \in]\leftarrow, x[$, thus the set of values $f(y)$ for $y < x$. This function satisfies (10).

Assume a and b in E. Then obviously $a \leq b$ implies $f(a) \subset f(b)$, and $a < b$ implies $f(a) \in f(b)$. Moreover $b \in f(a)$ implies $b \subset f(a)$. From $f(a) \in f(a)$ one deduces the existence of b such that $b < a$ and $f(b) \in f(b)$. As a consequence, there are no such object and f is injective; so that f is an order isomorphism (where the target of f is ordered by inclusion). If $x \in f(a)$, then $x = f(b)$ for some $b < a$, thus $f(b) \subset f(a)$ and $f(a)$ is transitive. If a is the smallest element such that $f(a)$ is not an ordinal, then all its elements are ordinals; contradiction. Since the image of f is transitive and contains only ordinals, it is an ordinal.

The first theorem can also be rewritten as: if X and Y are two ordinals isomorphic to the same ordered sets, then X = Y and the isomorphism are the same.

```
Lemma ordinal_isomorphism_unique: forall x y f,
  Bordinal x -> Bordinal y ->
  order_isomorphism f (inclusion_suborder x)(inclusion_suborder y) ->
  (x = y & f = identity x).

Lemma ordinal_isomorphism_unique2: forall x y z f g,
  Bordinal x -> Bordinal y ->
  order_isomorphism f (inclusion_suborder x) z ->
  order_isomorphism g (inclusion_suborder y) z ->
  (x = y & f = g).
Lemma ordinal_isomorphism_exists: forall r, worder r ->
  let f := transfinite_defined r target in
  Bordinal (target f) &
  order_isomorphism f r (inclusion_suborder (target f)).
```

The previous theorem says that for any well-ordered set E there exists an isomorphism f, whose target is some ordinal E′, that will be called the ordinal of E.

```
Definition ordinal r := target (transfinite_defined r target).
Lemma ordinal_p1: forall r, worder r -> Bordinal (ordinal r).
Lemma ordinal_p2: forall r, worder r -> exists f,
  order_isomorphism f r (inclusion_suborder (ordinal r)).
Lemma ordinal_p3: forall E, Bordinal E->
  ordinal (inclusion_suborder E) = E.
```

Denote the relation "x and y are ordinals and $x \subset y$" by $x \leq_{\mathrm{Ord}} y$. The associated relation $x <_{\mathrm{Ord}} y$ is the same as "x and y are ordinals and $x \in y$". This is a well-ordering, since the intersection I of a family of ordinals X is the least element, and we know $\mathrm{I} \in \mathrm{X}$. The union U is an ordinal, it is the least upper bound of the family (in the sense that $\mathrm{U} = \sup_{\mathrm{E}} \mathrm{X}$, where E is any set of ordinals containing U and the elements of X). The empty set is the least ordinal.

Consider a well-ordering $\leq$. Let's assume that the relation "$x < a$ and $a \leq y$" is collectivizing in a; If $x < y$, then the set $\{a, x < a \text{ and } a \leq y\}$ has a least element, that depends only on x, and will be called the successor of x. Such an element exists if $\leq$ is ordering with a graph, i.e., with a support E, provided that x is not the greatest element of E; it exists if $\leq$ is a relation without graph and if $a \leq y$ is collectivizing and x not maximal; for instance $\leq_{\mathrm{Card}}$ and $\leq_{\mathrm{Ord}}$ are such relations. We show here that the successor of an ordinal is $\mathrm{T}(x)$. The successor of a finite cardinal x is $x+1$; no explicit formula exists for infinite cardinals. For instance, let $\aleph_0$ denote the cardinal of $\mathbf{N}$ and $\aleph_1$ the cardinal of $\mathfrak{P}(\mathbf{N})$. The Cantor Theorem says $\aleph_0 < \aleph_1$, the "continuum hypothesis" is the assertion that $\aleph_1$ is the successor of $\aleph_0$. This is an unprovable statement.

According to Cantor, the limit x of a strictly increasing sequence of ordinals x_i is the least upper bound of the family $(x_i)_i$, and is called a *limit ordinal*. Set $\mathrm{X} = \{y, y <_{\mathrm{Ord}} x\}$. We have $x \leq \sup \mathrm{X}$, since the family x_i is a subset of X, and we also have $\sup \mathrm{X} \leq x$, so that $x = \sup \mathrm{X}$. Note that, if x is the successor of y, then $\sup \mathrm{X} = y$, so that a limit ordinal is not a successor. Note that $y <_{\mathrm{Ord}} x$ is the same as $y \in x$, thus $\mathrm{X} = x$ and $\sup \mathrm{X} = \bigcup x$. Thus we prove: either x is a successor or $x = \bigcup x$. Proof. For $y \in x$, the set $\mathrm{T}(x)$ is a subset of x (in other words, $y < x$ implies $\mathrm{T}(y) \leq x$). Any element z of x is in $\mathrm{T}(y)$ or satisfies $y \in z$ (z and y are ordinals, this is the trichotomy relation). This can be restated as: $x \subset \mathrm{T}(y)$ or $y \in \bigcup x$. Thus, either x is a successor, or $x \subset \bigcup x$ (hence equality). It is easy to show that one case excludes the other.

An alternative definiton is: a limit ordinal contains zero, and is stable by sucessor.

```
Definition ordinal_le x y :=
  Bordinal x & Bordinal y & sub x y.
Definition ordinal_lt x y :=
  Bordinal x & Bordinal y & inc x y.

Theorem wordering_ordinal_le: worder_r ordinal_le.
Lemma ordinal_least: forall x, Bordinal x -> ordinal_le emptyset x.
Lemma union_of_ordinals: forall x,
  (forall a, inc a x -> Bordinal a) -> Bordinal (union x).
Lemma union_of_ordinals_sup: forall x, (forall a, inc a x -> Bordinal a) ->
  let u:= union x in
    (Bordinal u
     & (forall a, inc a x -> ordinal_le a u)
     & (forall v, Bordinal v -> (forall a, inc a x -> ordinal_le a v)
        -> ordinal_le u v)).

Lemma ordinal_succ: forall x, Bordinal x ->
  let z := tack_on x x in
  ordinal_lt x z & (forall w, ordinal_lt x w -> ordinal_le z w).

Lemma ordinal_limit: forall x, Bordinal x ->
  let p := exists y, x = tack_on y y in
  let q := x = union x in
  (p \/ q) & (p-> q-> False).

Definition Bordinal_limit x:=
  Bordinal x & inc emptyset x & (forall y, inc y x -> inc (tack_on y y) x).

Lemma Bordinal_limit_pr: forall x, Bordinal x ->
  (Bordinal_limit x) = (nonempty x & x = union x).
```

We define the V-cardinal of X as the least B-ordinal equipotent to X. We can use the axiom of choice (in reality Zermelo's theorem) to show that X can be well-ordered, so that the set of ordinals equipotent to X is non-empty. Two sets have the same V-cardinal if and only if they are equipotent.

```
Definition Vcardinalp x y :=
  Bordinal y & equipotent x y &
  (forall z, Bordinal z ->  equipotent x z -> sub y z).
Definition Vcardinal x := choose (Vcardinalp x).

Lemma Vcardinal_unique: forall x y z,
  Vcardinalp x y -> Vcardinalp x z -> y = z.
Lemma Vcardinal_exists: forall x, exists y, Vcardinalp x y.
Lemma Vcardinal_pr: forall x,
  Bordinal (Vcardinal x) & equipotent x (Vcardinal x) &
  (forall z, Bordinal z ->  equipotent x z -> sub (Vcardinal x) z).
Lemma Vcardinal_pr1: forall x y,
   equipotent x y = (Vcardinal x = Vcardinal y).
```

Let's say that an ordinal x is *V-infinite* if it is equipotent to its successor. Assume that f is a bijection between $s(x)$ and $s(y)$. Since $s(x) = x \cup \{x\}$, there exists a such that $f(a) = y$. Let's modify by f by exchanging the values of x and a, so as to obtain $f'(x) = y$. The restriction

is thus a bijection $x \to y$, thus x and y are equipotent. In particular, if x is infinite and is the successor of y, then y is infinite.

We deduce that infinite V-cardinals are limit ordinals (the converse ifs false: all limits ordinals are of the form ωα, where α is some ordinal, ω is defined below, the product in a future chapter; if α is a finite non-zero ordinal, then ωα is equipotent to ω, thus is a cardinal only if α = 1).

A variant of the previous construction. Assume $f : x \to x \cup \{x\}$ is a bijection. Assume $x \subset y$. Extend f to y by $f(x) = y$ anf $f(t) = t$ for other values. This is a bijection, and y is V-infinite.

```
Definition Vinfinite u := equipotent u (tack_on u u).
Definition Vfinite u := Bordinal u & ~ (Vinfinite u).

Lemma Vsucc_injective: forall x y, Bordinal x -> Bordinal y ->
   equipotent (tack_on x x) (tack_on y y) ->
   equipotent x y. (* 53 *)

Lemma Vinfinite_pr:  forall x,
  Bordinal x -> Vinfinite (tack_on x x) -> Vinfinite x.
Lemma Vfinite_pr: forall x, Vfinite x -> Vfinite (tack_on x x).
Lemma Vinfinite_pr1:  forall x y z,  z= Vcardinal x ->  Vinfinite z ->
  z = tack_on y y  -> False.
Lemma Vinfinite_increasing: forall x y, Bordinal x -> Bordinal y ->
  inc x y -> Vinfinite x -> Vinfinite y. (* 73 *)
```

We shall assume the existence of an infinite set (in the Bourbaki sense). Bourbaki has an axiom for this, in Coq the set ℕ in infinite, Zermelo assumed the existence of a set x such that $\{a\}$ is in x whenever $a \in x$, and [7] assumes the existence of a set stable by the successor function. Let x be the V-cardinal of this set; it is infinite; and it is easy to show that x is equipotent to $x \cup \{x\}$. This says that x is V-infinite. The set of V-infinite subsets of x that are ordinals is not empty, so that the intersection ω is a V-infinite ordinal, it is the least one.

If α is an infinite ordinal then $\omega \subset \alpha$. Conversely, we have either $\alpha = \omega$ or $\omega \in \alpha$; in both cases, α is infinite. We can also restate this as α is a finite ordinal if and only it is a member of ω. By transitivity of ω, an element of a finite ordinal is finite. Thus, an ordinal is finite if and only if its successor is finite.

Note that ω is a limit ordinal: if it were the successor of y, then $y \in \omega$ implies y finite, its successor finite, contradicting the fact that ω is infinite. In fact, it is the least limit ordinal. Proof. Let f be the successor function on A, together with $f(A) = \emptyset$. If A is a limit ordinal, this is a function $T(A) \to A$. It is obviously injective. Assume the function non-surjective; let B be the least element not in the image. This cannot be empty, nor a successor (if $B = t \cup \{t\}$, then $t \in B \in A$), thus is a non-zero limit ordinal. Let now A be the least non-zero limit ordinal. We get $A \subset B$, contradicting $B \in A$. Thus, our function is a bijection, and A is infinite. If A is any other limit ordinal, it is greater than an infinite ordinal, thus is infinite.

We can restate this as follows. Let x be a finite ordinal. If non-zero, it is a successor (of a a finite ordinal).

```
Definition Bomega :=
  intersection (Zo (powerset (Vcardinal nat))
    (fun z=> Bordinal z & Vinfinite z)).
Lemma Bomega_pr:
   Bordinal Bomega & Vinfinite Bomega &
```

```
   (forall z, Bordinal z -> Vinfinite z -> sub Bomega z).

Definition Komega := Vcardinal nat.
Definition Bomega :=
  intersection (Zo (powerset Komega) (fun z=> Bordinal z & Vinfinite z)).

Lemma exists_infinite_ordinal: Bordinal Komega & Vinfinite Komega.
Lemma Bomega_pr:
   Bordinal Bomega & Vinfinite Bomega &
   (forall z, Bordinal z -> Vinfinite z -> sub Bomega z).

Lemma Bomega_ordinal: Bordinal Bomega.
Lemma Bomega_infinite: Vinfinite Bomega.
Lemma Bomega_pr1: forall x, Bordinal x ->
  (Vinfinite x = sub Bomega x).
Lemma Bomega_pr2: forall x, inc x Bomega = Vfinite x.
Lemma Vfinite_pr2: forall x, Vfinite x = Vfinite (tack_on x x).
Lemma Vfinite_pr3: forall x y, inc x y -> Vfinite y
   ->  Vfinite x.
Lemma Bomega_limit: Bomega = union Bomega.
Lemma Bomega_limit1: Bordinal_limit Bomega.
Lemma Bomega_limit2: forall x,  Bordinal_limit x -> sub Bomega x. (* 69 *)
Lemma Bomega_limit3: forall x,  Vfinite x->
  x = emptyset \/ (exists y, Vfinite y & x = tack_on y y).
```

The von Neumann Proof We have shown that for any well-ordered set E there exists an isomorphism f, defined by transfinite induction, whose target is some ordinal E$'$. Von Neumann calls a function satisfying (10) a *numeration* (we shall use here a functional graph, instead of a function). He shows that it is unique, that it exists, and defines an ordinal as the image of the graph.

```
Definition numeration r f:=
  fgraph f &  domain f = substrate r &
  forall x, inc x (domain f) ->  V x f = image_by_graph f (segment r x).
Definition  numerable r :=
   worder r & exists f, numeration r f.
Definition the_numeration r := transfinite_defined r target.
```

The proofs are rather straightforward (compare with the proof of existence of a function by transfinite induction). The key relation is that, if n(E) is the numeration of E, and F a segment of E then n(F) is the restriction of n(E) to F. If all segments of E can be numerated, we can extent these numerations to a numeration of E; otherwise, there is a least non-numerable segment; all its segments can be numerated, so that the segment can be numerated; absurd.

```
Lemma worder_numerable: forall r,
  worder r -> numeration r (graph (the_numeration r)).
Lemma numeration_unique: forall r f f',
  worder r -> numeration r f -> numeration r f' -> f = f'.
Lemma sub_numeration: forall  r f x,
  let r' := induced_order r (segment r x) in
  worder r ->  numeration r f -> inc x (substrate r) ->
  numeration r' (restr f (segment r x)).
Lemma segments_numerables: forall r,
  let r' := fun x => induced_order r (segment r x) in
```

```
  worder r -> (forall x, inc x (substrate r)-> numerable (r' x)) ->
  numerable r. (* 64 *)
Lemma worder_numerable_bis: forall r,
  worder r ->  numerable r. (* 22 *)
```

Chapter 6

Properties of integers

Bourbaki introduces the set of integers in the next chapter, while we have done so in the previous one. As a consequence, in all theorems, the phrase "x is an integer" will be replaced by $x \in \mathbf{N}$.

6.1 Operations on integers and finite sets

Consider a finite family x_i. This means that we have a functional graph G, with domain I, and an associated mapping $i \mapsto x_i$. Induction can be performed on the domain, the range or the graph: for instance, we can show by induction on the range of the family that the union of finite sets is finite, or that the supremum is finite. In these two cases, $\sup x_i$ or $\bigcup x_i$, the result depends only on the range (and one can consider the supremum of union of a set of sets, instead of a family). This is false in the case of a sum (we know that if $x_i = 1$, the sum is the cardinal of the index set, so that the range can be a singleton and the sum infinite).

In all these cases, we have a function F (union, supremum, sum) and a function g such that $\mathrm{F}(x_1, x_2, \ldots, x_n) = g(x_1, \mathrm{F}(x_2, \ldots, x_n))$. This formula allows us to prove properties by induction on the length n of the sequence (for $n \geq 2$). We may have $\mathrm{F}(x) = x$, in which case we can handle the case $n = 1$. In the case of the sum or union, the case $n = 0$ is trivial, in the case of the intersection or supremum, the function is not defined if the sequence is empty. Finite sequences (i.e., sequences where indices are 1, 2, 3, etc.) will be studied later. If g is not commutative, we can define F only in the case where the index set is ordered; if g is commutative (as in the examples above), any ordering can be used, and there is no need to convert the finite family into a finite sequence. We know that union, sum, supremum are associative. In some cases, we want to prove an associativity formula for a function F defined as above, for a non-commutative function g (this is basic algebra). We shall explain how this can be done, in section 6.4.

We start with some helper lemmas. The first two say that the sum or product of the restriction of a family $(x_i)_i$ to a singleton $\{j\}$ is x_j. The other two ones say that we have a partition of $\mathrm{X} = a \cup \{b\}$ formed of a and $\{b\}$ whenver $b \notin a$. For any X, if $b \in \mathrm{X}$, if $a = \mathrm{X} - \{b\}$ then $\mathrm{X} = a \cup \{b\}$.

```
Lemma trivial_cardinal_sum3: forall f a, fgraph f ->
  inc a (domain f) -> is_cardinal (V a f) ->
  cardinal_sum (restr f (singleton a))  = V a f.
Lemma trivial_cardinal_prod3: forall f a, fgraph f ->
  inc a (domain f) -> is_cardinal (V a f) ->
```

```
  cardinal_prod (restr f (singleton a))  = V a f.
Lemma partition_tack_on: forall a b, ~ inc b a ->
  partition_fam (variantLc a (singleton b)) (tack_on a b).
Lemma partition_complement: forall a b, inc b a ->
  partition_fam (variantLc (complement a (singleton b)) (singleton b)) a.
```

We show here

$$x_j + \sum_{i\in J} x_i = \sum_{i\in J\cup\{j\}} x_i \tag{1}$$

$$x_j \cdot \prod_{i\in J} x_i = \prod_{i\in J\cup\{j\}} x_i \tag{2}$$

whenever $J\cup\{i\}$ is a subset of (or is equal to) the domain of the family x_i and $j\notin J$. The proofs are similar; we have a partition of $J\cup\{j\}$, and use the associativity formula, that says that the sum on the right hand side is a sum over the canonical doubleton, thus a sum of two terms.

```
Lemma induction_sum0: forall f a b, fgraph f ->
  sub (tack_on a b) (domain f) -> (~ inc b a) ->
  is_cardinal (V b f) ->
  cardinal_sum (restr f (tack_on a b)) =
  card_plus(cardinal_sum (restr f a)) (V b f).
Lemma induction_prod0: forall f a b, fgraph f ->
  sub (tack_on a b) (domain f) -> (~ inc b a) ->
  is_cardinal (V b f) ->
  cardinal_prod (restr f (tack_on a b)) =
  card_mult (cardinal_prod (restr f a)) (V b f).
Lemma induction_sum1: forall f a b, fgraph f ->
  domain f = tack_on a b -> (~ inc b a) -> is_cardinal (V b f) ->
  cardinal_sum f =
  card_plus(cardinal_sum (restr f a)) (V b f).
Lemma induction_prod1: forall f a b, fgraph f ->
  domain f = tack_on a b -> (~ inc b a) -> is_cardinal (V b f) ->
  cardinal_prod f =
  card_mult (cardinal_prod (restr f a)) (V b f).
```

Given a family of cardinals $\mathfrak{a}_i$ with sum S and product P we have $\mathfrak{a}_i \le S$ and $\mathfrak{a}_i \le P$, whenever all factors are non-zero.

```
Lemma sum_increasing6: forall f j, fgraph f ->
  (forall x, inc x (domain f) -> is_cardinal (V x f)) ->
  inc j (domain f) -> cardinal_le (V j f) (cardinal_sum f).
Lemma prod_increasing6: forall f j, fgraph f ->
  (forall x, inc x (domain f) -> is_cardinal (V x f)) ->
  (forall x, inc x (domain f) -> V x f <> card_zero)  ->
  inc j (domain f) -> cardinal_le (V j f) (cardinal_prod f).
```

A *finite family of integers* is a functional graph $i \mapsto x_i$ where the index set I is finite and each x_i is finite. In the definition below, we use $x_i \in \mathbf{N}$.

```
Definition finite_int_fam f:=
  fgraph f  &
  (forall i, inc i (domain f) -> inc (V i f) Bnat) &
  is_finite_set (domain f).
```

Proposition 1 [3, p. 171] says that if $(a_i)_{i\in I}$ is a finite family of integers, then $\sum_{i\in I} a_i$ and $\prod_{i\in I} a_i$ are integers. As a consequence, if $J \subset I$ then $\sum_{i\in J} a_i$ and $\prod_{i\in J} a_i$ are integers. The proof is by induction on the finite set J via formulas (1) and (2).

```
Lemma finite_sum_finite_aux: forall f x, finite_int_fam f ->
  sub x (domain f) -> inc (cardinal_sum (restr f x)) Bnat.
Lemma finite_product_finite_aux: forall f x,  finite_int_fam f ->
  sub x (domain f) -> inc (cardinal_prod (restr f x)) Bnat.
Theorem finite_sum_finite: forall f,  finite_int_fam f ->
  inc (cardinal_sum f) Bnat.
Theorem finite_product_finite: forall f, finite_int_fam f ->
  inc (cardinal_prod f) Bnat.
```

We have obvious consequences. For instance, a finite union of finite sets is finite. As explained above, this is easy by induction. Bourbaki says that, if E is the union and S is the sum, then S is finite, and, since there is a surjection from S onto E, we have Card(E) ≤ S, so that card(E) is finite. However Card(E) ≤ S has already been stated (as corollary to Proposition 4). A finite product of finite sets is a finite set (since the cardinal of the product is the product of the cardinals). Since a^b is a product, it is finite if a and b are finite. Thus, the powerset of a finite set is finite (these results wer e proved in the previous chapter).

```
Lemma finite_union_finite: forall f, fgraph f ->
  (forall i, inc i (domain f) -> is_finite_set (V i f))
  -> is_finite_set (domain f) -> is_finite_set(unionb f).
Lemma finite_product_finite_set: forall f, fgraph f ->
  (forall i, inc i (domain f) -> is_finite_set (V i f))
  -> is_finite_set (domain f) -> is_finite_set(productb f).
```

6.2 Strict inequalities between integers

Proposition 2 [3, p. 173] says that $a < b$ if and only if there is c such that $0 < c$ and $b = c + a$ (a, b and c being integers).[1] Assume $a < b$. We know that there exists a cardinal c such that $b = c + a$; obviously c is a non-zero integer. Conversely, since $a < a + 1$, and $1 \le c$, we get $a + 1 \le a + c$ and we conclude by transitivity.

```
Lemma strict_pos_pr: forall a, inc a Bnat ->
  (card_zero <> a) = (cardinal_lt card_zero a).
Lemma strict_pos_pr1: forall a, inc a Bnat ->
  (a <> card_zero) = (cardinal_lt card_zero a).

Therorem cardinal_lt_pr: forall a b, inc a Bnat -> inc b Bnat ->
  (cardinal_lt a b = exists c, inc c Bnat & c <> card_zero &
    card_plus a c = b).


Lemma lt_n_succ_le: forall a b, inc a Bnat -> inc b Bnat ->
  cardinal_le (succ a) b= cardinal_lt a b.
Lemma succ_apb1 : forall a b, succ (card_plus a b) = card_plus (succ a) b.
Lemma succ_apb2 : forall a b, succ (card_plus a b) = card_plus a (succ b).
```

[1] In version 3, we replaced all tests $0 < c$ by $c \neq 0$

Proposition 3 [3, p. 173] says that $\sum a_i < \sum b_i$ and $\prod a_i < \prod b_i$ for two families of integers with the same index set, if $a_i \leq b_i$ for each i and $a_j < b_j$ for some j. In the case of a product, $b_i > 0$ is required. The case where the family has two elements is a consequence of the statements given above [we have $ab < a(b+c)$ if a and c are non-zero, by distributivity). In the general case, assume $a_j < b_j$, and consider the partition $J \cup \{j\}$ of I. We have $\sum_{i\in I} a_i = A + a_j$, where $A = \sum_{i\in J} a_i$. In the same way, $\sum_{i\in I} b_i = B + b_j$, and $A \leq B$. The proof in the case of the product is similar. Note that A is finite since it is the sum of a restriction.

```
Lemma finite_sum2_lt: forall a b a' b',
  inc a Bnat -> inc b Bnat -> inc a' Bnat -> inc b' Bnat->
  cardinal_le a a' -> cardinal_lt b b' ->
  cardinal_lt (card_plus a b) (card_plus a' b').
Lemma finite_sum3_lt: forall a a' b,
  inc a Bnat -> inc b Bnat -> inc a' Bnat -> cardinal_lt a a' ->
  cardinal_lt (card_plus a b) (card_plus a' b).

Lemma finite_prod2_lt: forall a b a' b',
    inc a Bnat -> inc b Bnat -> inc a' Bnat -> inc b' Bnat->
  cardinal_le a a' -> cardinal_lt b b' -> a' <> card_zero ->
  cardinal_lt (card_mult a b) (card_mult a' b').
Theorem finite_sum_lt: forall f g,  (* 14 *)
  finite_int_fam f -> finite_int_fam g ->  domain f = domain g ->
  (forall i, inc i (domain f) -> cardinal_le (V i f) (V i g)) ->
  (exists i, inc i (domain f) & cardinal_lt (V i f) (V i g)) ->
  cardinal_lt (cardinal_sum f) (cardinal_sum g).
Theorem finite_product_lt: forall f g,  (* 18 *)
  finite_int_fam f -> finite_int_fam g -> domain f = domain g ->
  (forall i, inc i (domain f) -> cardinal_le (V i f) (V i g)) ->
  (exists i, inc i (domain f) & cardinal_lt (V i f) (V i g)) ->
  (forall i, inc i (domain f) ->  card_zero <> (V i g)) ->
  cardinal_lt (cardinal_prod f) (cardinal_prod g).
```

Consequences: $a < a'$ implies $0 < a'$. If $a \neq 0$ and $1 < b$ then $a < ab$. If $a \neq 0$ and $b \geq 1$ then $a^b \geq a$ (this holds also for infinite cardinals). We have $a^b < a'^b$ if $a < a'$ and $b \neq 0$ (the first condition implies $a' > 0$). We have $a^b < a^{b'}$ if $a > 1$ and $b < b'$ (the case $a = 0$ is special, if $a = 1$, both terms are 1). Writing $b' = b + x$ we must show $a^b < a^b a^x$. If $b > 1$ we get $a < b^a$.

```
Lemma one_small_cardinal2: forall a, is_cardinal a ->
  cardinal_le card_one a = (a <> card_zero).
Lemma finite_lt_a_ab: forall a b, inc a Bnat -> inc b Bnat ->
  a <> card_zero -> cardinal_lt card_one b ->
  cardinal_lt a (card_mult a b).
Lemma cardinal_le_a_apowb: forall a b,
  is_cardinal a -> a <> card_zero -> cardinal_le card_one b ->
  cardinal_le a (card_pow a b).
Lemma non_zero_apowb: forall a b,
 is_cardinal a -> is_cardinal b ->
  a<>  card_zero -> (card_pow a b) <> card_zero.
Lemma finite_power_lt1: forall a a' b,
  inc a Bnat ->  inc a' Bnat -> inc b Bnat->
  cardinal_lt a a' -> b <> card_zero ->
  cardinal_lt (card_pow a b) (card_pow a' b).
Lemma finite_power_lt2: forall a b b',
  inc a Bnat ->  inc b Bnat -> inc b' Bnat->
  cardinal_lt b b' -> cardinal_lt card_one a ->
```

```
  cardinal_lt (card_pow a b) (card_pow a b').
Lemma lt_a_power_b_a: forall a b, inc a Bnat ->  inc b Bnat ->
  cardinal_lt card_one b -> cardinal_lt a (card_pow b a).
Lemma pow_succ: forall a b, inc a Bnat -> inc b Bnat ->
  card_pow a (succ b) = card_mult(card_pow a b) a.
```

If $a+b=a+b'$ or if $ab=ab'$ then $b=b'$ (all arguments are integers; $a\neq 0$ in the case of a product).

```
Lemma plus_simplifiable_left: forall a b b',
 inc a  Bnat -> inc b Bnat -> inc b' Bnat ->
  card_plus a b = card_plus a b' -> b = b'.
Lemma plus_simplifiable_right: forall a b b',
 inc a  Bnat -> inc b Bnat -> inc b' Bnat ->
  card_plus b a = card_plus b' a -> b = b'.
Lemma mult_simplifiable_left: forall a b b',
  inc a  Bnat -> inc b Bnat -> inc b' Bnat -> a <> card_zero ->
  card_mult a b = card_mult a b' -> b = b'.
Lemma mult_simplifiable_right: forall a b b',
  inc a  Bnat -> inc b Bnat -> inc b' Bnat -> a <> card_zero ->
  card_mult b a = card_mult b' a -> b = b'.
```

If a and b are integers and $a\leq b$ there is a unique integer c such that $b=a+c$, it is called the *difference*, and denoted by $b-a$. The operation is called *subtraction*. There is a Coq function, denoted by *sub*, defined for all integers, whose value is zero for $a>b$; we extend our function so that they share the same behavior.

```
Definition card_sub a b :=
   Yo (cardinal_le b a)
     (choose (fun c => inc c Bnat & card_plus b c = a)) card_zero.

Lemma card_sub_pr0: forall 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).
Lemma Bnat_stable_sub: forall a b, inc a Bnat -> inc b Bnat ->
  inc (card_sub a b) Bnat.
Lemma card_sub_wrong: forall a b, inc a Bnat -> inc b Bnat ->
  ~ (cardinal_le b a) -> card_sub a b = card_zero.
Lemma card_sub_pr: forall a b, inc a Bnat-> inc b Bnat ->
  cardinal_le b a -> card_plus b (card_sub a b) = a.
Lemma card_sub_rpr: forall a b, inc a Bnat-> inc b Bnat ->
  cardinal_le b a -> card_plus (card_sub a b) b = a.
```

We have $(a+b)-b=a$ for all integers. Uniqueness means that if $a+b=c$ then $a=c-b$ and $b=c-a$.

```
Lemma card_sub_pr1: forall a b, inc a Bnat-> inc b Bnat ->
  card_sub (card_plus a b)  b  = a.
Lemma card_sub_pr2: forall a b c, inc a Bnat-> inc b Bnat ->
  card_plus a b = c -> card_sub c b = a.
```

We have $(b-a)+(b'-a')=(b+b')-(a+a')$ if the first two differences are defined. We have the trivial formulas: $a-a=0$ and $a-0=a$. If $a>0$, then $a-1$ is the predecessor of a. We have $(a-b)-c=a-(b+c)$ if $a\geq b+c$. Taking $c=1$, and denoting by P and S the predecessor and successor, we have $\mathrm{P}(a-b)=a-\mathrm{S}b$. If $b\leq a$, then $a-b\leq a$; and if $b<a$, the $(a-b)-1<a$. If $b+1\leq a$ then $b<a$.

```
Lemma minus_n_nC: forall a, inc a Bnat -> card_sub a a = card_zero.
Lemma minus_n_0C: forall a, inc a Bnat -> card_sub a card_zero =a.

Lemma card_sub_pr4: forall a b a' b', inc a Bnat-> inc b Bnat ->
  inc a' Bnat-> inc b' Bnat ->
  cardinal_le a b->  cardinal_le a' b' ->
  card_sub (card_plus b b') (card_plus a a') =
  card_plus (card_sub b a)  (card_sub b' a') .

Lemma card_sub_associative: forall a b c,
  inc a Bnat -> inc b Bnat -> inc c Bnat ->
  cardinal_le (card_plus b c) a ->
  card_sub (card_sub a b) c = card_sub a (card_plus b c).
Lemma prec_pr1: forall a, inc a Bnat -> a <> card_zero
  -> predc a = card_sub a card_one.
Lemma card_sub_non_zero: forall a b, inc a Bnat -> inc b Bnat ->
  cardinal_le (succ b) a -> card_sub a b <> card_zero.
Lemma card_sub_non_zero1: forall a b, inc a Bnat ->
  cardinal_lt b a -> card_sub a b <> card_zero.
Lemma card_sub_associative1: forall a b, inc a Bnat -> inc b Bnat ->
  cardinal_le (succ b) a -> predc (card_sub a b) = card_sub a (succ b).
Lemma sub_le_symmetry: forall  a b, inc a Bnat -> inc b Bnat ->
  cardinal_le b a -> cardinal_le (card_sub a b) a).
Lemma sub_lt_symmetry: forall n p,
  inc p Bnat -> cardinal_lt n p ->
  cardinal_lt (prec (card_sub p n)) p.
Lemma double_sub: forall n p, inc n Bnat -> cardinal_le p n ->
  card_sub n (card_sub n p) = p.
```

We state here some properties of the ordering on **N**.

```
Lemma Bnat_le_reflexive: forall a, inc a Bnat -> Bnat_le a a.
Lemma Bnat_le_transitive: forall a b c, Bnat_le a b -> Bnat_le b c ->
  Bnat_le a c.
Lemma Bnat_le_antisymmetric: forall a b, Bnat_le a b -> Bnat_le b a -> a = b.
Lemma Bnat_total_order: forall a b, inc a Bnat -> inc b Bnat ->
  Bnat_le a b \/ Bnat_lt b a.
Lemma Bnat_zero_smallest: forall a, inc a Bnat -> Bnat_le card_zero a.
Lemma Bnat_zero_smallest1: forall a, Bnat_le a card_zero -> a = card_zero.
```

We show here that if $a+b \leq a+b'$, $a+b < a+b'$, $ab \leq ab'$ or $ab < ab'$ then $b \leq b'$ or $b < b'$ if inequality is strict in the assumption; in the case of a product, a must be non-zero. This is because $\leq$ is a total ordering, and the opposite relation yields a contradiction. We deduce that $(a+c)-(b+c) = a-b$, even when b is greatesr than a.

```
Lemma plus_le_simplifiable: forall a b c,
  inc a Bnat -> inc b Bnat -> inc c Bnat->
  cardinal_le (card_plus a b)  (card_plus a c) ->  cardinal_le b c.
Lemma plus_lt_simplifiable: forall a b c,
  inc a Bnat -> inc b Bnat -> inc c Bnat->
  cardinal_lt (card_plus a b)  (card_plus a c) ->  cardinal_lt b c.
Lemma mult_le_simplifiable: forall a b c,
  inc a Bnat -> inc b Bnat -> inc c Bnat-> a <> card_zero ->
  cardinal_le (card_mult a b)  (card_mult a c) ->  cardinal_le b c.
Lemma mult_lt_simplifiable: forall a b c,
  inc a Bnat -> inc b Bnat -> inc c Bnat-> a <> card_zero ->
```

```
  cardinal_lt (card_mult a b)  (card_mult a c) ->  cardinal_lt b c.

Lemma card_sub_pr5: forall a b c, inc a Bnat -> inc b Bnat -> inc c Bnat ->
   card_sub (card_plus a c) (card_plus b c) = card_sub a b.
Lemma card_sub_pr6: forall a b, inc a Bnat -> inc b Bnat ->
   card_sub (succ a) (succ b) = card_sub a b.
```

6.3 Intervals in sets of integers

Bourbaki says that, for every integer a, there is a set containing all integers $\leq a$; he denotes this as $[0,a]$. The set $[a,b]$ is used without definition, in a context where this denotes the set of all integers x such that $a \leq x \leq b$. He says that $x \mapsto x+a$ is a strictly increasing isomorphism of $[0,b]$ onto $[a,a+b]$. In fact, these sets contain only cardinals, hence are well-ordered by $\leq_{\mathrm{Card}}$. As a consequence, all isomorphisms are strictly increasing.

We use here a different approach. If $a \in \mathbf{N}$ and $b \in \mathbf{N}$, we can consider $[a,b]$ as an interval for the ordering $\leq_{\mathbf{N}}$ on $\mathbf{N}$ (ordered by $\leq_{\mathbf{N}}$). This set is the same as the previous one, and has the same ordering.

```
Definition interval_Bnat a b := interval_cc Bnat_order a b.
Definition interval_co_0a a:= interval_co Bnat_order card_zero a.
Definition interval_Bnato a b :=
  graph_on cardinal_le (interval_cc Bnat_order a b).
```

We give here some properties of intervals. We have $x \in [a,b]$ if and only if $a \leq_{\mathbf{N}} x$ and $x \leq_{\mathbf{N}} b$, where $x \leq_{\mathbf{N}} b$ is the same as $x \in \mathbf{N}$, and $b \in \mathbf{N}$ and $x \leq b$. Note that $x \leq b$ implies $x \in \mathbf{N}$. Thus we state: $x \in [0,a[$ if and only if $x < a$ and $x \in [0,a]$ if and only if $x \leq a$. These two intervals are subsets of $\mathbf{N}$. We have $[0,a+1[= [0,a]$.

```
Lemma sub_interval_Bnat: forall a b,  sub (interval_Bnat a b) Bnat.
Lemma sub_interval_co_0a_Bnat: forall a,  sub (interval_co_0a a) Bnat.
Lemma interval_Bnat_pr: forall a b x, inc a Bnat -> inc b Bnat ->
  inc x (interval_Bnat a b) = (cardinal_le a x & cardinal_le x b).
Lemma interval_Bnat_pr0: forall b x, inc b Bnat ->
  inc x (interval_Bnat card_zero b) =  cardinal_le x b.
Lemma interval_Bnat_pr1: forall b x, inc b Bnat ->
  inc x (interval_Bnat card_one b) =  (x <> card_zero & cardinal_le x b).
Lemma interval_Bnat_pr1b: forall b x, inc b Bnat ->
  inc x (interval_Bnat card_one b) =
   (cardinal_le card_one x & cardinal_le x b).
Lemma interval_co_0a_pr2: forall a x, inc a Bnat ->
  inc x (interval_co_0a a) = cardinal_lt x a.
Lemma interval_co_0a_pr3: forall a x, inc a Bnat ->
  inc x (interval_co_0a (succ a)) = cardinal_le x a.
Lemma interval_co_cc: forall p, inc p Bnat ->
  interval_Bnat card_zero p = interval_co_0a (succ p).
Lemma interval_cc_0a_increasing:  forall a b, inc b Bnat ->
  cardinal_le a b ->
  sub (interval_Bnat card_zero a) (interval_Bnat card_zero b).
Lemma interval_cc_0a_increasing1:  forall a, inc a Bnat ->
  sub (interval_Bnat card_zero a) (interval_Bnat card_zero (succ a)).
Lemma inc_a_interval_co_succ: forall a, inc a Bnat ->
  inc a (interval_co_0a (succ a)).
```

```
Lemma interval_co_0a_increasing:  forall a, inc a Bnat ->
  sub (interval_co_0a a) (interval_co_0a (succ a)).
Lemma interval_co_0a_increasing1:  forall a b, inc a Bnat -> inc b Bnat ->
  cardinal_le a b -> sub (interval_co_0a a) (interval_co_0a b).
Lemma interval_co_pr4: forall n, inc n Bnat ->
  ( (tack_on (interval_co_0a n) n = (interval_co_0a (succ n)))
    & ~(inc n (interval_co_0a n))).
Lemma interval_bn_pr5: forall n, inc n Bnat ->
  let si := (interval_Bnat card_one n) in
  ( (tack_on si card_zero = interval_Bnat card_zero n)
    & ~(inc card_zero si)).
Lemma cardinal_c_induction5:  forall (r:Set -> Prop) a,
  inc a Bnat -> r card_zero ->
  (forall n, cardinal_lt n a -> r n -> r (succ n))
  ->  (forall n, cardinal_le n a -> r n).
```

Note that $x \leq_{[a,b]} y$ is equivalent to $a \leq x \leq y \leq b$, where $\leq$ is the ordering on cardinals, since this relation implies that x and y are finite.

```
Lemma interval_Bnato_worder: forall a b, inc a Bnat -> inc b Bnat ->
  worder (interval_Bnato a b).
Lemma interval_Bnato_substrate: forall a b, inc a Bnat -> inc b Bnat ->
  substrate (interval_Bnato a b) = interval_Bnat a b.
Lemma interval_Bnato_related: forall a b x y, inc a Bnat -> inc b Bnat ->
  gle (interval_Bnato a b) x y = (inc x (interval_Bnat a b) &
    inc y (interval_Bnat a b) & cardinal_le x y).
Lemma interval_Bnato_related1: forall a b x y, inc a Bnat -> inc b Bnat ->
  gle (interval_Bnato a b) x y = (cardinal_le a x  &
  cardinal_le a y &  cardinal_le x b & cardinal_le y b& cardinal_le x y).
Lemma interval_Bnato_related2: forall a b x y, inc a Bnat -> inc b Bnat ->
  gle (interval_Bnato a b) x y = (cardinal_le a x
    & cardinal_le y b& cardinal_le x y).
```

We define here the ordered interval $[0, a[$. This is a well-ordered set (in fact, it is a segment of $\mathbf{N}$). We have $x \leq_{[0,a[} y$ if and only if $x \leq y$ and $y < a$, since these two relations imply $x \in [0, a[$ and $y \in [0, a[$.

```
Definition interval_Bnatco a :=
  graph_on cardinal_le (interval_co_0a a).

Lemma interval_Bnatco_worder: forall a, inc a Bnat ->
  worder (interval_Bnatco a).
Lemma interval_Bnatco_substrate: forall a, inc a Bnat ->
  substrate (interval_Bnatco a) = interval_co_0a a.
Lemma interval_Bnatco_related: forall a x y, inc a Bnat ->
  gle (interval_Bnatco a) x y = (cardinal_le x y  & cardinal_lt y a).
Lemma segment_Bnat_order: forall x, inc x Bnat ->
  segment Bnat_order x = interval_co Bnat_order card_zero x.

Lemma sub_increasing2: forall a b c, inc a Bnat -> inc b Bnat -> inc c Bnat->
  cardinal_le c (card_plus a b) -> cardinal_le (card_sub c b) a.
```

We consider now the function $z \mapsto z + b$, $(z \in [0, a], z + b \in [a, a + b])$, that has $z \mapsto z - b$ as inverse, and hence is a bijection. Proposition 4 [3, p. 174] says that these functions are order isomorphisms.

```
Definition rest_plus_interval a b :=
  BL(fun z => card_plus z b)(interval_Bnat card_zero a)
    (interval_Bnat b (card_plus a b)).

Definition rest_minus_interval a b :=
  BL(fun z => card_sub z b) (interval_Bnat b (card_plus a b))
  (interval_Bnat card_zero a)

Theorem restr_plus_interval_isomorphism: forall a b, inc a Bnat -> inc b Bnat->
  (bijective (rest_plus_interval a b)
  & bijective (rest_minus_interval a b)
  &(rest_minus_interval a b) = inverse_fun (rest_plus_interval a b)
  &
   order_isomorphism (rest_plus_interval a b)
    (interval_Bnato card_zero a)
    (interval_Bnato b (card_plus a b))). (* 58 *)
```

If $a \leq b$, then $[a, b+1] = [a, b] \cup \{b+1\}$ and the union is disjoint. Thus $[a, b+1]$ has one more element than $[a, b]$. By induction, it has $b+1$ elements when $a = 0$, and by application of the isomorphism shown above, it has $(b-a)+1$ elements. This is Proposition 5 [3, p. 174]. As a consequence, the set of integers is infinite (since $[0, n]$ is subset of $\mathbf{N}$ we have $n+1 \leq \mathrm{Card}(\mathbf{N})$, so that $\mathrm{Card}(\mathbf{N})$ cannot be of the form n). This argument is used at the start of Chapter 6: the axiom that asserts the existence of an infinite set is equivalent to the assertion that there exists a set containing all finite cardinals.

```
Lemma interval_zero_zero:
  interval_Bnat card_zero card_zero = singleton card_zero.
Lemma cardinal_interval0a: forall a, inc a Bnat ->
  cardinal (interval_Bnat card_zero a) = succ a.
Theorem cardinal_interval: forall a b, Bnat_le a b->
  cardinal (interval_Bnat a b) = succ (card_sub b a).
Lemma finite_set_interval_Bnat: forall a b, Bnat_le a b ->
  is_finite_set (interval_Bnat a b).
Lemma finite_set_interval_co: forall a, inc a Bnat ->
  is_finite_set (interval_co Bnat_order card_zero a).
Lemma Bnat_infinite: ~(is_finite_set Bnat).
```

Proposition 6 [3, p. 175] asserts that every finite totally ordered set is isomorphic to a unique interval $[1, n]$, where n is the number of elements. Bourbaki adds the condition $n \geq 1$, this is not needed. We start with a lemma that says that $[1, n]$ has n elements (note that this is also true for $n = 0$). Then we pretend that if E and F are two finite equipotent totally ordered sets, there is a unique order isomorphism between them. This is because the sets are well-ordered, so that there exists a unique order morphism from E onto a segment of F; whatever the image, the function is bijective hence is an order isomorphism. The general theorem says that there is another possibility, namely that there exists an isomorphism from F onto a segment of E; it is bijective, so that its inverse is an isomorphism from E onto F. Bourbaki uses Corollary 2 to Proposition 2 of § 2, no. 2 (it says: if X is a subset of a finite set E, and $X \neq E$, then $\mathrm{Card}(X) < \mathrm{Card}(E)$. Maybe Corollary 4 was intended, since it says that an injection is bijective).

```
Lemma cardinal_interval1a: forall a, inc a Bnat ->
  cardinal (interval_Bnat card_one a) = a.
Lemma isomorphism_worder_finite: forall r r',  (* 43 *)
```

```
  total_order r -> total_order r' ->
  is_finite_set (substrate r) -> equipotent (substrate r) (substrate r') ->
  exists_unique (fun f => order_isomorphism f r r').
Theorem finite_ordered_interval: forall r, total_order r ->
  is_finite_set (substrate r) ->
  exists_unique (fun f => order_isomorphism f r
    (interval_Bnato card_one (cardinal (substrate r)))).
```

We consider here properties of $[0, b-1]$. If we denote it by I_b, it is the interval $[0, b[$. Note that $[0, b[$ exists even when $b = 0$. We can rewrite Proposition 6 as: every finite totally ordered set is isomorphic to a unique interval $[0, n[$, where n is the number of elements.

```
Lemma cardinal_interval_co_0a: forall a, inc a Bnat -> a <> card_zero ->
  cardinal (interval_Bnat card_zero (predc a)) = a.
Lemma interval_co_0a_pr: forall a x, inc a Bnat -> a <> card_zero ->
  inc x (interval_Bnat card_zero (predc a)) = (inc x Bnat & cardinal_lt x a).
Lemma interval_co_0a_pr1:  forall a, inc a Bnat -> a <> card_zero ->
  interval_Bnat card_zero (predc a) =  interval_co_0a a.
Lemma cardinal_interval_co_0a1: forall a, inc a Bnat ->
  cardinal (interval_co_0a a) = a.
Lemma emptyset_interval_00: interval_co_0a card_zero = emptyset.

Theorem finite_ordered_interval1: forall r, total_order r ->
  is_finite_set (substrate r) ->
  exists_unique (fun f => order_isomorphism f r
    (interval_Bnatco  (cardinal (substrate r)))).
```

More properties of intervals.

```
Lemma partition_tack_on_intco: forall a, inc a Bnat ->
  partition_fam (variantLc (interval_co_0a a)
    (singleton a)) (interval_co_0a (succ a)).
Lemma interval_co_0a_restr: forall a f, inc a Bnat ->
  (L (interval_co Bnat_order card_zero a) f
    = (restr (L (interval_co Bnat_order card_zero (succ a)) f)
      (interval_co_0a a))).

Lemma partition_tack_on_int: forall n, inc n Bnat ->
  partition_fam (variantLc (interval_Bnat card_one n)
       (singleton card_zero))
    (interval_Bnat card_zero n).
Lemma interval_int_restr: forall a f, inc a Bnat ->
  L (interval_Bnat card_one a) f
    = restr (L (interval_Bnat card_zero a) f)
      (interval_Bnat card_one a).
```

We show here the following. Assume that $f(n)$ is a cardinal for all n. Let $F(n)$ be the cardinal sum of the family $i \mapsto f(i)$ on $[0, n-1]$. Then $F(n+1) = f(n) + F(n)$, and there is a similar relation for the product.

```
Lemma induction_on_sum: forall a f,  inc a Bnat ->
  (forall a, inc a Bnat -> is_cardinal (f a)) ->
  let iter := fun n=> cardinal_sum (L (interval_co_0a n)f)
    in card_plus (iter a) (f a) = (iter (succ a)).
Lemma induction_on_prod: forall a f,  inc a Bnat ->
```

```
  (forall a, inc a Bnat -> is_cardinal (f a)) ->
  let iter := fun n=> cardinal_prod (L (interval_co_0a n ) f)
    in card_mult (iter a) (f a) = (iter (succ a)).
```

We show here tht the sum of $f(i)$ for $0 \leq i \leq n$ is $f(0)$ plus the sum $f(i)$ for $1 \leq i \leq n$. We rewrite this is the sum of $f(i+1)$ for $0 \leq i \leq n-1$.

```
Lemma fct_sum_rec0: forall f n, inc n Bnat ->
  is_cardinal (f card_zero) ->
  cardinal_sum (L (interval_Bnat card_zero n) f)
  = card_plus (cardinal_sum (L (interval_Bnat card_one n) f)) (f card_zero).

Lemma fct_sum_rec1: forall f n, inc n Bnat ->
  is_cardinal (f card_zero) ->
  cardinal_sum (L (interval_Bnat card_zero (succ n)) f)
  = card_plus
      (cardinal_sum (L (interval_Bnat card_zero n) (fun i=> f (succ i))))
   (f card_zero). (* 23 *)

Lemma fct_sum_rev: forall f n, inc n Bnat ->
  let I := (interval_co_0a (succ n)) in
  cardinal_sum (L I f) = cardinal_sum (L I (fun i=> f (card_sub n i))).
```

6.4 Finite sequences

A *finite sequence* is a family $(x_i)_{i\in \mathrm{I}}$ whose index set is a finite subset of **N** (Bourbaki says: a finite set of integers). Let f be the unique isomorphism f of the interval $[1,n]$ onto I (with the natural ordering on I). Then $x_{f(k)}$ is defined for $k \in [1,n]$. It is called the *kth term of the sequence*. If $k = 1$ or $k = n$, it is called the first or last term.

6.5 Characteristic functions on sets

We define $\phi_{\mathrm{A}}(x)$ to be 1 if $x \in \mathrm{A}$ and 0 otherwise. This induces a function on every set B, the characteristic function of B.

```
Definition char_fun A B := BL (fun z=> Yo (inc z A) card_one card_zero)
  B (doubleton  card_one card_zero).

Lemma char_fun_axioms: forall A B,
  transf_axioms (fun z=> Yo (inc z A) card_one card_zero)
  B (doubleton  card_one card_zero).
Lemma char_fun_function: forall A B, is_function (char_fun A B).

Lemma char_fun_W:forall A B x,
  inc x B -> W x (char_fun A B) =  Yo (inc x A) card_one card_zero.
Lemma char_fun_W_cardinal:forall A B x,
  inc x B -> is_cardinal (W x (char_fun A B)).
Lemma char_fun_W_a:forall A B x,  sub A B -> inc x A ->
  W x (char_fun A B) = card_one.
Lemma char_fun_W_b:forall A B x,  sub A B -> inc x (complement B A) ->
  W x (char_fun A B) = card_zero.
```

Now some properties. We have $\phi_A = \phi_B$ if and only if A = B. The function ϕ_A (for $A \subset E$) is constant if and only if A = E or $A = \emptyset$. Proposition 7 [3, p. 176] lists additional properties.

$$\phi_{E-A}(x) = 1 - \phi_A(x)$$

$$\phi_{A\cap B}(x) = \phi_A(x)\phi_B(x)$$

$$\phi_{A\cap B}(x) + \phi_{A\cup B}(x) = \phi_A(x) + \phi_B(x)$$

```
Lemma char_fun_injective: forall A A' B, sub A B -> sub A' B ->
  (A=A') = (char_fun A B = char_fun A' B).

Lemma char_fun_W_aa:forall A x,  inc x A ->
  W x (char_fun A A) = card_one.
Lemma char_fun_W_bb:forall A x,  inc x A ->
  W x (char_fun emptyset A) = card_zero.
Lemma char_fun_constant:forall A B, sub A B ->
  (forall x y, inc x B -> inc y B -> W x (char_fun A B) = W y (char_fun A B))
  -> (A=B \/ A = emptyset).
Lemma char_fun_complement: forall A B x, sub A B -> inc x B ->
  W x (char_fun (complement B A) B)
  = card_sub card_one (W x (char_fun A B)).
Lemma char_fun_inter: forall A A' B x, sub A B -> sub A' B -> inc x B ->
  W x (char_fun (intersection2 A A') B)
  =  card_mult (W x (char_fun A B))(W x (char_fun A' B)).
Lemma char_fun_union: forall A A' B x, sub A B -> sub A' B -> inc x B ->
  card_plus (W x (char_fun (intersection2 A A') B))
  (W x (char_fun (union2 A A') B) )
  =  card_plus (W x (char_fun A B))(W x (char_fun A' B)).
```

6.6 Euclidean Division

Assume that $P\{x\}$ is a property of integers, satisfied by at least one element. Since the set of integers is well-ordered, there is a least such element, hence x such that $P(x)$ is false and $P(x+1)$ is true, unless $P(0)$ is true. We give two variants of this fact.

```
Lemma least_int_prop: forall prop:Set -> Prop,
  (forall x, prop x -> inc x Bnat) -> (exists x, prop x) ->
  prop card_zero \/ (exists x, inc x Bnat & prop(succ x) & ~ prop x).
Lemma least_int_prop1: forall prop:Set -> Prop,
  (forall x, prop x -> inc x Bnat) -> ~(prop card_zero) ->
  (exists x, prop x) -> (exists x, inc x Bnat &prop(succ x) & ~ prop x).
```

Theorem 1 [3, p. 176] says that, if $b > 0$, a and b are integers, there exist unique integers q (called *quotient*) and r (called *remainder*) such that $a = bq + r$ and $r < b$. We show that the conditions are equivalent to $bq \le a < b(q+1)$ and $r = a - bq$. Thus q is the least integer such that $a < b(q+1)$. This inequality is satisfied for $q = a$, this shows existence and uniqueness of q. For simplicity[2], in the case $b = 0$, we define the quotient to be zero and the remainder is a, so that $a = bq + r$ will still be true.

In order to define the quotient and remainder, we must use the axiom of choice. We start with the definition of the quotient and remainder for finite cardinals. If the remainder of the

[2] New in version 3

division of a by b is zero we say that b *divides* a. We use here[3] a strict definition: a and b are assumed to be integers.

```
Definition division_prop a b q r :=
  a = card_plus (card_mult b q) r & cardinal_lt r b.
Definition card_rem0 a b:=
  choose (fun r => inc r Bnat  & exists q, inc q Bnat & division_prop a b q r).
Definition card_quo0 a b:=
  choose (fun q => inc q Bnat & exists r, inc r Bnat & division_prop a b q r).
Definition card_rem a b := Yo (b = card_zero) a (card_rem0 a b).
Definition card_quo a b := Yo (b = card_zero) card_zero (card_quo0 a b).
Definition BNdivides b a :=
  inc a Bnat & inc b Bnat & card_rem a b = card_zero.
```

We prove existence and uniqueness of the division; this shows that our definitions make sense.

```
Lemma division_result_integer: forall a b q r, inc a Bnat-> inc b Bnat ->
  b <> card_zero -> division_prop a b q r -> is_cardinal q ->
  (inc q Bnat & inc r Bnat).

Lemma division_prop_alt: forall a b q r, inc a Bnat-> inc b Bnat ->
  inc q Bnat-> inc r Bnat -> b <> card_zero ->
  division_prop a b q r = (cardinal_le (card_mult b q) a
  & cardinal_lt a (card_mult b (succ q))
  & r = card_sub a  (card_mult b q)).

Lemma division_unique:forall a b q r q' r', inc a Bnat-> inc b Bnat ->
  inc q Bnat-> inc r Bnat -> inc q' Bnat-> inc r' Bnat -> b <> card_zero ->
  division_prop a b q r -> division_prop a b q' r' ->
  (q = q' & r =r').

Lemma division_exists:forall a b, inc a Bnat-> inc b Bnat ->
  b <> card_zero -> exists q, exists r,
    (inc q Bnat & inc r Bnat & division_prop a b q r).
Lemma Bnat_division: forall 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))).
```

We state some properties of division.

```
Lemma card_quo_zero: forall a, card_quo a card_zero = card_zero.
Lemma card_rem_zero: forall a, card_rem a card_zero = a.
Lemma Bnat_stable_quo:forall a b, inc a Bnat-> inc b Bnat ->
 inc (card_quo a b) Bnat.
Lemma Bnat_stable_rem:forall a b, inc a Bnat-> inc b Bnat ->
  inc (card_rem a b) Bnat.
Lemma Bnat_div_pr: forall a b, inc a Bnat-> inc b Bnat ->
  a = card_plus (card_mult b (card_quo a b)) (card_rem a b).
Lemma Bnat_rem_pr: forall a b, inc a Bnat-> inc b Bnat -> b <> card_zero ->
  cardinal_lt  (card_rem a b) b.
Lemma Bnat_quorem_pr: forall a b q r,
  inc a Bnat-> inc b Bnat -> inc q Bnat -> inc r Bnat ->
  division_prop a b q r -> (q = card_quo a b & r = card_rem a b).
```

[3] New in Version 3

```
Lemma Bnat_quorem_pr0: forall a b q,
  inc a Bnat-> inc b Bnat -> inc q Bnat -> b <> card_zero ->
   a = (card_mult b q) -> (q = card_quo a b & card_zero = card_rem a b).
```

Note. Bourbaki says: the number q introduced above is *the integral part of the quotient of a by b*, since in the set of rational numbers $\mathbf{Q}$, there exists c such that $a = bc$, and q is the integral part of c. He reserves the term *quotient* only to the case where the remainder is zero. Moreover he says "writing a/b or $\frac{a}{b}$ will imply that b divides a". This is an abuse of notations. (It is all right to say that $a < b$ implies that a and b are integers, because we can consider as a short-hand for "$a < b$ and $a \in \mathbf{N}$ and $b \in \mathbf{N}$", but, if $d(a,b)$ is the property that b divides a, and $q(a,b)$ is the quotient, then a/b cannot be a short-hand for "$q(a,b)$ and $d(a,b)$" because this expression makes no sense (it is neither a term nor a relation). Moreover, the convention is not always respected since Bourbaki proves

$$\sum_{i=1}^{n} i = \frac{1}{2} n(n+1).$$

Now some consequences when division is exact. Bourbaki says: every multiple a' of a multiple a of b is a multiple of b. One can restate this as: if b divides a, then b divides ac.

```
Lemma BNdivides_pr: forall a b, inc a Bnat -> inc b Bnat ->
  BNdivides b a -> a =  card_mult b  (card_quo a b).
Lemma BNdivides_pr1: forall a b, inc a Bnat -> inc b Bnat ->
  BNdivides b (card_mult b a).
Lemma BNdivides_pr2: forall a b q,
  inc a Bnat -> inc b Bnat -> inc q Bnat -> b <> card_zero ->
  a = card_mult b q -> q = card_quo a b.
Lemma BNdivides_one: forall a, inc a Bnat -> BNdivides card_one a.
Lemma BN_quo_one: forall a,  inc a Bnat -> card_quo a card_one = a.
Lemma BNdivides_pr3: forall a b q,
  BNdivides b a -> q = card_quo a b ->
  a =  card_mult b  q.
Lemma BNdivides_pr4: forall b q, inc b Bnat -> inc q Bnat -> b <> card_zero->
  card_quo (card_mult b q) b = q.
Lemma BNdivision_itself: forall a, inc a Bnat -> a <> card_zero ->
  (BNdivides a a & card_quo a a = card_one).
Lemma BNdivides_itself: forall a,inc a Bnat -> a <> card_zero ->
 BNdivides a a.
Lemma BNquo_itself: forall a, inc a Bnat -> a <> card_zero ->
  card_quo a a = card_one.
Lemma BNdivision_of_zero: forall a,  inc a Bnat ->
  (BNdivides a card_zero & card_quo card_zero a = card_zero).
Lemma BNdivides_trans: forall a b a',
  BNdivides a a'-> BNdivides b a -> BNdivides b a'.
Lemma BNdivides_trans1: forall a b a',
   BNdivides a a' -> BNdivides b a
  -> card_quo a' b = card_mult  (card_quo a' a) (card_quo a b).
Lemma BNdivides_trans2: forall a b c, inc c Bnat ->
  BNdivides b a-> BNdivides b (card_mult a c).
```

The first lemma says $(ac)/(bc) = a/b$ even when division is not exact. If b divides a and a', it divides the sum and the difference.

```
Lemma card_quo_simplify:  forall a b c,
```

```
  inc a Bnat -> inc b Bnat -> inc c Bnat -> b <> card_zero-> c <> card_zero->
  card_quo (card_mult a c) (card_mult b c) = card_quo a b.
Lemma divides_and_sum:  forall a a' b,  BNdivides b a ->  BNdivides b a'
  -> (BNdivides b  (card_plus a a') &
    card_quo  (card_plus a a')  b =  card_plus (card_quo a b)(card_quo a' b)).
Lemma distrib_prod2_sub: forall a b c, inc a Bnat -> inc b Bnat -> inc c Bnat
  -> cardinal_le c b ->
  card_mult a (card_sub b c) = card_sub (card_mult a b) (card_mult a c).
Lemma divides_and_difference:  forall a a' b,
  cardinal_le  a' a ->
  BNdivides b a ->  BNdivides b a'
  -> (BNdivides b  (card_sub a a') &
    cardinal_le (card_quo a' b) (card_quo a b) &
    card_quo  (card_sub a a')  b = card_sub (card_quo a b) (card_quo a' b)).
```

6.7 Expansion to base b

Proposition 8 [3, p. 177] is *Let b be an integer > 1. For each integer $k > 0$ let E_k be the lexicographic product of the family $(\mathrm{J}_h)_{0\le h\le k-1}$ of intervals all identical with $[0, b-1]$; For each $r = (r_0, r_1, \ldots, r_{k-1}) \in \mathrm{E}_k$, let $f_k(r) = \sum_{h=0}^{k-1} r_h b^{k-h-1}$; then the mapping f_k is an isomorphism of the ordered set E_k onto the interval $[0, b^k - 1]$*. Bourbaki notes that (if $a > 0$) there is a least integer k such that $a < b^k$ hence a unique sequence r_h such that

$$a = \sum_{h=0}^{k-1} r_h b^{k-h-1} \tag{1}$$

subject to the conditions $0 \le r_h \le b-1$ for $0 \le h \le k-1$ and $r_0 > 0$.

Discussion. We say that (1) is a BE-expansion, and it is a normalized expansion if either $k = 0$ (the sum is empty) or $r_0 > 0$. The quantity r_h is the digit of index k, and r_0 is the leading digit. We can restate the theorem as: every integer has an expansion to base b for some k, and a unique normalized expansion. Two numbers expressed in base b with k digits can be compared using only the value of the digits, starting with the leading digits. We can complete the theorem as follows: one can add or remove zero leading digits in the expansion, hence given two numbers with k and k' digits, one can add leading zeroes to the smallest sequence, then apply the theorem, or else remove leading zeroes in order to get normalized expansions, and then the number that has the smallest number of digits is the smallest number.

Note that the conditions $0 \le r_h$ and $0 \le h$ are redundant, since negative integers have not yet been introduced. The conditions can be restated as $r_h < b$ for $h < k$, and the interval $[0, b-1]$ is the interval $[0, b[$, this is the set of all integers $< b$. The condition $0 \le h \le k-1$ is equivalent to $0 \le k-h-1 \le k-1$, and the sum can be rewritten as $\sum_{h=0}^{k-1} r_{k-h-1} b^h$. If $s_k = r_{k-h-1}$, we get

$$a = \sum_{h=0}^{k-1} s_h b^h \tag{2}$$

subject to the conditions $0 \le s_h \le b-1$ for $0 \le h \le k-1$ and $s_{k-1} \ne 0$ is the normalization condition. We call this is LE-expansion.[4] Associated to $f_k(r)$ is the function $g_k(s)$.

[4] According to Wikipedia, little-endian storage means: increasing numeric significance with increasing memory addresses; big-endian is its opposite, most-significant byte first.

If $\psi(s)$ is the sequence $(s_1,\dots,s_k)$, then

$$g_{k+1} = s_0 + b.g_k(\psi(s)) \tag{3}$$

(Bourbaki has a similar formula with f and ϕ). This is a recursive definition; one can convert it into an iterative one: as long as there are digits, multiply by b and add the next digit. We start with s_{k-1} and terminate with s_0. This means that we consider digits from left to right, r_0 then r_1, then r_2, etc. This is called the Horner scheme for the formula (1), and is used by every computer program to read numbers. If s is represented as a list, then s_0 is the head of the list and $\psi(s)$ is its tail, and (3) is the natural way to associate a value to the list.

A consequence of (3) is that s_0 and $g_k(\psi(s))$ are the remainder and quotient of the Euclidean division of a by b, and this shows uniqueness by induction. Computers use this method for printing numbers, i.e., finding the sequence s_h given a; the number k is not known a priori. In practice, one has either fixed-size numbers, say $< 2^{32}$, case where an apriori bound can be found; or else $a = \sum_0^{\mathrm{K}-1} \mathrm{S}_h \mathrm{B}^h$, for some B, case where $k \le n\mathrm{K}$ for some n, which is the size of the expansion of B in base b. The digits are computed one after the other, stored in a buffer. After that, the number is normalized (useless zeroes are removed).

Assume $s_0 < b$, $s_0' < b$; then $s_0 + bg \le s_0 + bg'$ if and only if either $g = g'$ and $s_0 \le s_0'$ or $g < g'$. This condition is equivalent to $(g, s_0) \le (g', s_0')$, where (g, s_0) is in the substrate of some lexicographic product $\mathrm{F}_k \times \mathrm{J}$ of two sets. By induction, we can identify F_k with the set of sequences $(s_1,\dots,s_k)$. Because s_0 comes after g, this set is the lexicographic product *in reverse order* of the sets J_h. Thus, the ordering of the sequence r_h is the lexicographic product of the sets J_h.

Consider now the sequence $\Psi(s) = (s_0,\dots,s_{k-1})$, then

$$g_{k+1} = g_k(\Psi(s)) + s_k.b^k. \tag{4}$$

It happens that s_k and $g_k(\Psi(s))$ are the quotient and remainder of the Euclidean division of g_{k+1} by b^k. This is an alternate way to show uniqueness of the expansion (one could use it to print a number in a computer program, the drawback being that one has to compute all b^k in decreasing order). Note that $s_k b^k \le g_{k+1} < (s_k+1)b^k$. This shows that two numbers with distinct leading digits compare as their leading digits, and shows the theorem. We shall use this approach since Ψ is just the restriction.

We define an *expansion* to be a family of k terms, all less than b, where k and b are integers, $b \ge 2$. The domain of the family is the interval $[0,k[$, it is the set of integers i with $i < k$. The associated *value* is $\sum f_i b^i$. It is an integer. If we have an expansion of length $k+1$, the restriction to $[0,k[$ is an expansion. The values are the same, up to the quantity $f_k b^k$. Similarly, we can extend an expansion from size k to size $k+1$.

```
Lemma b_power_k_large: forall a b, inc a Bnat ->  inc b Bnat ->
  cardinal_lt card_one b -> a <> card_zero -> exists k,
    inc k Bnat & cardinal_le  (card_pow b k) a
    & cardinal_lt a (card_pow b (succ k)).

Definition is_expansion f b k :=
  inc b Bnat & inc k Bnat & cardinal_lt card_one b &
  fgraph f & domain f = interval_co_0a k &
  forall i, inc i (domain f) -> cardinal_lt (V i f) b.

Definition expansion_value f b :=
  cardinal_sum (L (domain f) (fun i=> card_mult (V i f) (card_pow b i))).
```

We assume locally that (f,k,b) and $(g,k'+1,b)$ are expansions.

```
Section Base_b_expansion.
Variables f b k: Set.
Variable Exp: is_expansion f b k.
Variable Expg: is_expansion g b (succ k').

Lemma is_expansion_prop0: forall i,
   (inc i (domain f)) = cardinal_lt i k.
Lemma is_expansion_prop1: forall  i,
  cardinal_lt i k -> inc (V i f) Bnat.
Lemma is_expansion_prop2:
  finite_int_fam  (L (domain f) (fun i=> card_mult (V i f) (card_pow b i))).
Lemma is_expansion_prop3:
  inc (expansion_value f b) Bnat.
Lemma is_expansion_prop4: is_cardinal k' -> inc k' Bnat.
Lemma is_expansion_prop6: is_cardinal k' -> inc (V k' g) Bnat.
Lemma is_expansion_prop7: is_cardinal k' ->
  (expansion_value g b) =
  card_plus (expansion_value (restr g (interval_co_0a k')) b)
     (card_mult (V k' g) (card_pow b k')).
End  Base_b_expansion.
```

Denote by $s(f)$ or by $s_k(f)$ the sum $\sum_{i<k} f_i$. We have $s_k(f) < b^k$, and $s_{k+1}(f) = s_k(f) + f_k b^k$. As a consequence the quotient and remainder of the division of $s_{k+1}(f)$ by b^k are f_k and $s_k(f)$. This shows uniqueness of the expansion, namely that $s_k(f) = s_k(g)$ implies $f_i = g_i$ for all $i < k$.

```
Lemma is_expansion_prop8: forall f b k x,
  let g:=  L (interval_co_0a (succ k)) (fun i=> Yo (i=k) x (V i f)) in
    is_expansion f b k ->
    inc x Bnat -> cardinal_lt x b ->
    (is_expansion g b (succ k) &
      expansion_value g  b = card_plus (expansion_value f b)
      (card_mult (card_pow b k) x)).  (* 18 *)

Lemma is_expansion_prop9: forall f b k,  is_expansion f b k ->
  cardinal_lt (expansion_value f b)  (card_pow b k). (* 35 *)
Lemma is_expansion_prop10: forall f b k,  is_cardinal k ->
  is_expansion f b (succ k) ->
  division_prop (expansion_value f b) (card_pow b k) (V k f)
  (expansion_value (restr f (interval_co_0a k)) b).
Lemma is_expansion_unique: forall f g b k,
  is_expansion f b k -> is_expansion g b k ->
  expansion_value f b = expansion_value g b -> f = g. (* 47 *)
Lemma is_expansion_prop11: forall f g b k, is_cardinal k ->
  is_expansion f b (succ k) -> is_expansion g b (succ k) ->
  cardinal_lt (V k f) (V k g) ->
  cardinal_lt (expansion_value f b)  (expansion_value g b).
```

Consider the two following properties. $P(f,g)$ says that there exists an index i in the range of g, not in the range of f such that g_i is not zero. It obviously implies $s(f) < s(g)$. Condition $Q(f,g)$ first says that there is an index n such that $f_i = 0$ and $g_i = 0$ for $i \geq n$, provided that these expressions are defined (and f_i and g_i are defined for $i < n$). Such an index exists if $P(f,g)$ and $P(g,f)$ are false. We have then $s(f) = s_n(f)$ and $s(g) = s_n(g)$. The Bourbaki claim

is then that $s(f)$ and $s(g)$ can be compared lexicographically (replacing i by $n-i$). Condition Q says moreover that there exists k such that $f_i = g_i$ for $k < i < n$, so that $s(f)$ and $s(g)$ compare the same as $s_k(f)$ and $s_k(g)$. The case $k = -1$ is special, since it says that $s(f) = s(g)$. Thus, assuming $s(f) \neq s(g)$, there is a least such k [in other terms: if $s(f) \neq s(g)$ there is a greatest index k such that $f_k \neq g_k$]. Now condition Q says $f_k < g_k$. We have shown above that this implies $s_k(f) < s_k(g)$. The theorem is now $s(f) < s(g)$ if and only one of P or Q is true. Notice that the five cases $P(f,g)$, $P(g,f)$, $Q(f,g)$, $Q(g,f)$, $R(f,g)$ are mutually exclusive (here R is the condition that there is n, such that $f_i = g_i$ for $i < n$, and for all indices $i \geq n$ for which f_i and g_i are defined, the value is zero; it implies $s(f) = s(g)$).

```
Lemma is_expansion_restr1: forall f b k l,
  is_expansion f b k -> cardinal_le l k ->
  is_expansion (restr f  (interval_co_0a l)) b l.
Lemma is_expansion_restr2: forall f b k l, (* 35 *)
  is_expansion f b k -> cardinal_le l k ->
  (forall i, cardinal_le l i -> cardinal_lt i k -> V i f = card_zero) ->
  expansion_value (restr f  (interval_co_0a l)) b = expansion_value f b.

Lemma is_expansion_prop12: forall f g b kf kg l n,
  cardinal_le n kf -> cardinal_le n kg ->  cardinal_lt l n ->
  (forall i, cardinal_le n i -> cardinal_lt i kf -> V i f = card_zero) ->
  (forall i, cardinal_le n i -> cardinal_lt i kg -> V i g = card_zero) ->
  (forall i, cardinal_lt l i -> cardinal_lt i n -> V i f = V i g) ->
  is_expansion f b kf -> is_expansion g b kg ->
  cardinal_lt (V l f) (V l g) ->
  cardinal_lt (expansion_value f b)  (expansion_value g b). (* 99 *)

Lemma is_expansion_prop13: forall f g b kf kg l,
  cardinal_le kf l -> cardinal_lt l kg ->
  is_expansion f b kf -> is_expansion g b kg ->
  V l g <> card_zero ->
  cardinal_lt (expansion_value f b)  (expansion_value g b). (* 26 *)

Lemma is_expansion_prop14: forall f g b kf kg,
  is_expansion f b kf -> is_expansion g b kg ->
  cardinal_lt (expansion_value f b)  (expansion_value g b) ->
  (exists l,
    cardinal_le kf l & cardinal_lt l kg &
    V l g <> card_zero)
 \/ (
  exists l, exists n,
  (cardinal_le n kf & cardinal_le n kg & cardinal_lt l n &
  (forall i, cardinal_le n i -> cardinal_lt i kf -> V i f = card_zero) &
  (forall i, cardinal_le n i -> cardinal_lt i kg -> V i g = card_zero) &
  (forall i, cardinal_lt l i -> cardinal_lt i n -> V i f = V i g) &
  cardinal_lt (V l f) (V l g))). (* 102 *)
```

If $a < b^k$ there is an extension of length k (proof by induction, the highest term is the quotient of the division by b^{k-1}). Since $a < b^a$, there is at least one extension.

```
Lemma is_expansion_exists1: forall a b k,  inc a Bnat -> inc b Bnat ->
  cardinal_lt card_one b ->  cardinal_lt a (card_pow b k) ->
  exists f,  (is_expansion f b k & expansion_value f b = a).
Lemma is_expansion_exists: forall a b,  inc a Bnat -> inc b Bnat ->
  cardinal_lt card_one b -> exists k, exists f,
```

```
    (is_expansion f b k & expansion_value f b = a).
```

6.8 Combinatorial analysis

The following definition and lemmas have been moved here from the next chapter. See explanations on page 134.

```
Definition induction_defined s a:= choose(fun f=>
  source f = Bnat &  surjective f & W card_zero f = a &
  forall n, inc n Bnat -> W (succ n) f = s (W n f)).

Lemma integer_induction0: forall h a, exists_unique
   (fun f =>
   source f = Bnat & surjective f &
    W card_zero f = a
    &  forall n, inc n Bnat -> W (succ n) f = h n (W n f)).
Lemma integer_induction: forall s a, exists_unique (fun f =>
  source f = Bnat &  surjective f & W card_zero f = a &
  forall n, inc n Bnat -> W (succ n) f = s (W n f)).
Lemma induction_defined_pr0: forall h a,
  let f :=  induction_defined0 h a in
     source f = Bnat &  surjective f & W card_zero f = a &
     forall n, inc n Bnat -> W (succ n) f = h n (W n f).
```

We use here the previous definition as follows: for any element a and any function $s(x,y)$ we define a term T (depending on s and a) such that $\mathrm{T}(0) = a$ and $\mathrm{T}(n+1) = s(n, \mathrm{T}(n))$.

```
Definition induction_term s a := fun n => W n (induction_defined s a).
Lemma induction_term0: forall s a,
  induction_term s a card_zero =a.
Lemma induction_terms: forall s a n,
  inc n Bnat ->
  induction_term s a (succ n) = s n (induction_term s a n).
```

The next result is Proposition 9 [3, p. 179], known in French as the shepherd's principle. If f is a function from a set with cardinal $\mathfrak{a}$ onto a set with cardinal $\mathfrak{b}$, and if all sets $f^{-1}\langle\{x\}\rangle$ have the same cardinal $\mathfrak{c}$, then $\mathfrak{a} = \mathfrak{b}\mathfrak{c}$. Bourbaki assumes f surjective; in fact if x is in the target but not in the range, then $\mathfrak{c} = 0$, and the source is empty.

```
Theorem shepherd_principle: forall f c, is_function f ->
  (forall x, inc x (target f) -> cardinal (inv_image_by_fun f (singleton x))=c)
  -> cardinal (source f) = card_mult (cardinal (target f)) c. (* 29 *)
```

6.8.1 Factorial

Bourbaki defines the *factorial* of n, denoted by $n!$, as $\prod_{i<n}(i+1)$. It satisfies $0! = 1$ and $(n+1)! = n!(n+1)$. There is a unique function satisfying this property.

```
Definition factorial n :=
  cardinal_prod (L (interval_co_0a n) succ).
```

```
Lemma factorial_succ: forall n, inc n Bnat ->
  factorial (succ n) = card_mult (factorial n) (succ n).
Lemma factorial0: factorial card_zero = card_one.
Lemma factorial1: factorial card_one = card_one.
Lemma factorial2: factorial card_two = card_two.
Lemma factorial_nonzero: forall n, inc n Bnat -> factorial n <> card_zero.
Lemma factorial_integer: forall n, inc n Bnat -> inc (factorial n) Bnat.

Lemma factorial_prop: forall f, f 0 = 1 ->
  (forall n, f (S n) =  (f n) * (S n)) ->
   forall x, f x = factorial x.
Lemma factorial_prop1: forall n, factorial n = fct_prod S n.
```

We show how the factorial function could have been defined by induction.

```
Lemma factorial_induction: forall n, inc n Bnat ->
 factorial n = induction_term  (fun a b=> card_mult b (succ a)) card_one n.
```

6.8.2 Number of injections

Proposition 10 [3, p. 170] says that the number of injections from a set with m elements to a set with n elements is $A_{nm} = n!/(n-m)!$. Note that, if such an injection exists, we have $m \leq n$; otherwise the number is zero.

We first prove that the quantity A_{nm} is well-defined (by induction on $n-m$). Thus $A_{nm}(n-m)! = n!$. From this we deduce $A_{nm}(n-m) = A_{n,m+1}$.

```
Definition number_of_injections b a :=
  card_quo (factorial a) (factorial (card_sub a b)).

Lemma quotient_of_factorials: forall a b,
  inc a Bnat -> inc b Bnat -> cardinal_le b a ->
  BNdivides (factorial b) (factorial a).
Lemma quotient_of_factorials1: forall a b,
  inc a Bnat -> inc b Bnat -> cardinal_le b a ->
  BNdivides (factorial (card_sub a b)) (factorial a).

Lemma number_of_injections_pr: forall a b,
 inc a Bnat -> inc b Bnat -> cardinal_le b a ->
  card_mult (number_of_injections b a) (factorial (card_sub a b))
  = factorial a.
Lemma number_of_injections_base: forall a, inc a Bnat ->
  number_of_injections card_zero a = card_one.
Lemma number_of_injections_rec: forall a b,
  inc a Bnat -> inc b Bnat -> cardinal_lt b a ->
  card_mult (number_of_injections b a)  (card_sub a b) =
  number_of_injections (succ b) a.
Lemma number_of_injections_int: forall a b,
 inc a Bnat -> inc b Bnat -> cardinal_le b a ->
 inc (number_of_injections b a) Bnat.
```

Consider an injective function f from A into B, which are sets with cardinals a and b. Let c be the cardinal of the complenent of the image, we have $a + c = b$. We deduce $b - a = c$ when the target is finite.

```
Lemma cardinal_complement_image1: forall f, injective f ->
 card_plus (cardinal (complement (target f) (image_of_fun f)))
           (cardinal (source f))
  = cardinal (target f).
Lemma cardinal_complement_image: forall f, injective f ->
  is_finite_set (target f) ->
  (cardinal_le (cardinal (source f)) (cardinal (target f)) &
    cardinal (complement (target f) (image_of_fun f)) =
    card_sub (cardinal (target f)) (cardinal (source f))).
```

The proof of the proposition is as follows. If $m = 0$, then $A_{nm} = 1$; and there is a unique function from the empty set into E, this function is injective. Consider now $A = A' \cup \{a\}$, where A' has m elements. Let G_1 and G_2 be the sets of injective functions from A and A' to F, and H_1 and H_2 their cardinals. By induction, using the recurrence formula for A_{nm} we must show $H_1 = H_2(n-m)$. Given $f \in G_1$, its restriction $R(f)$ to A' is obviously injective, hence is in G_2. Given f' in G_2, and $b \in F$, there is a unique f with $f' = R(f)$ and $f(a) = b$. This function is an injection if and only if b is not in the range of f'. In other terms, $f \mapsto f(a)$ is a bijection from $R^{-1}\langle\{f'\}\rangle$ onto the complementary of the range of f' that has $n-m$ elements. Hence $R^{-1}\langle\{f'\}\rangle$ has $n-m$ elements; the conclusion follows from the shepherd's principle.

```
Definition set_of_injections  E F :=
  Zo (set_of_functions E F)(fun z=> injective z).

Lemma number_of_injections_prop: forall E F,
  is_finite_set F ->
  cardinal_le (cardinal E) (cardinal F) ->
  cardinal (set_of_injections E F)  =
    number_of_injections (cardinal E) (cardinal F). (* 128 *)
```

An injection from E into itself is a bijection when E is finite. Since $A_{nn} = n!$, we deduce that $n!$ is the number of *permutations* of E.

```
Lemma number_of_permutations: forall E,  is_finite_set E ->
  cardinal (set_of_permutations E)  = (factorial (cardinal E)).
```

6.8.3 Number of coverings

Proposition 11 [3, p. 180] says "let E be a finite set with p elements, and let $(p_i)_{1\le i\le h}$ be a finite sequence of integers such that $\sum_{i=1}^{h} p_i = n$. Then the number of coverings $(X_i)_{1\le i\le h}$ of E by mutually disjoints sets X_i such that $\mathrm{card} X_i = p_i$ for $1 \le i \le h$ is equal to $n!/(\prod_{i=1}^{h} p_i!)$."

In this section, p will denote a finite integer family (a mapping $I \to \mathbf{N}$, where I is a finite set). We shall replace the condition $1 \le i \le p$ by $i \in I$ everywhere. To say that $(X_i)_{i\in I}$ is a covering of E by mutually disjoints sets is the same as *partition_fam X E*, by abuse of language we shall call this a "partition" of E. Notice that X_i is a subset of E, so that the family X is an element of the set of functional graphs $I \to \mathfrak{P}(E)$. If the domain of X is I and $\mathrm{card} X_i = p_i$ whenever $i \in I$, we say that X is a partittion with p elements and write $X \in C_{p}E$. The proposition gives a formula for the cardinal of $C_{p}E$.

Note that $\sum p_i$ is finite. Thus, if $C_{p}E$ is non-empty, then E is finite. We start with a lemma thay says that, if $\mathrm{Card}(E) = \sum p_i$ then $C_{p}E$ is non-empty. The proof given here is a rather long (over 500 lines).

```
Definition partition_with_pi_elements p E f :=
```

```
  domain f = domain p &
  (forall i, inc i (domain p) -> cardinal (V i f) = V i p) &
  partition_fam f E.
Definition set_of_partitions p E :=
  Zo(set_of_gfunctions (domain p) (powerset E))
    (fun z=> partition_with_pi_elements p E z).
Lemma  set_of_partitions_rw: forall p E f,
  inc f  (set_of_partitions p E)= partition_with_pi_elements p E f.
Lemma equipotent_restriction: forall f x,
  sub x (source f) -> bijective f ->
  equipotent x (image_by_fun f x).
Lemma fif_cardinal: forall i p,
  finite_int_fam p -> inc i (domain p) -> is_cardinal (V i p).
Lemma pip_prop0: forall p E f, partition_with_pi_elements p E f ->
  forall i, inc i (domain f) ->  sub (V i f) E.
Definition equipotent_ex E F :=
   choose (fun z=>  bijective z & source z =E  & target z = F).

Lemma equipotent_ex_pr: forall E F, let z :=  equipotent_ex E F in
  equipotent E F ->
   (bijective z & source z =E  & target z = F).

Lemma number_of_partitions1: forall p E,
  finite_int_fam p -> cardinal_sum p = cardinal E ->
  nonempty(set_of_partitions p E). (* 45 *)
```

Define Q(E) to be the set of permutations of E. Assume $g \in$ Q(E) and $f \in \mathrm{C}_{p\mathrm{E}}$. Denote by $\psi(f,g)$ the mapping $i \mapsto g\langle f_i\rangle$; it belongs to $\mathrm{C}_{p\mathrm{E}}$ (same argumentation as before). Thus we have a function $\phi : g \mapsto \psi(f,g)$, from Q to $\mathrm{C}_{p\mathrm{E}}$. We pretend this is surjective (lemma 4) (if X_i is a partition with p_i elements, then F_i is equipotent to X_i; this gives a bijection h_i from F_i to X_i, hence a function g_i that extends h_i, with target E, and a function g that coincides with g_i on F_i).

```
Definition set_of_partitions_aux f g:=
  L (domain f) (fun i => image_by_fun g (V i f)).

Lemma number_of_partitions3: forall p E f g,
  partition_with_pi_elements p E f -> inc g  (set_of_permutations E) ->
  inc (set_of_partitions_aux f g) (set_of_partitions p E).  (* 26 *)

Lemma number_of_partitions4: forall p E f,
  finite_int_fam p -> cardinal_sum p = cardinal E ->
  partition_with_pi_elements p E f ->
  surjective (BL (fun g =>  (set_of_partitions_aux f g))
    (set_of_permutations E) (set_of_partitions p E)).  (* 72 *)
```

This function ϕ is not injective: lemma 5 says $\phi(g) = \phi(h)$ if and only if $h^{-1} \circ g$ is a bijection that leaves each $f(i)$ invariant. Fix h; for each g and i, the restriction w_i of $h^{-1} \circ g$ to $f(i)$ can be considered as a function from $f(i)$ to itself; it is in fact a bijection, hence the family $(w_i)_i$ is an element of the product of the sets $\mathrm{Q}(f(i))$ (this is lemma 6).

Consider now the function Φ that associates to each permutation k of E the family of restrictions to $\mathrm{A}_i = f(i)$. If k leaves A_i invariant, the restriction of k to A_i is a bijection, so that, if we consider as target of Φ the product of the permutations of A_i, Φ is well-defined for $h^{-1} \circ g$, for each $g \in \phi^{-1}(\{h\})$. This is a bijection (lemma 7). The main result follows: the

target of Φ has cardinal $\prod p_i!$; the source of Φ is $\phi^{-1}\langle\{h\}\rangle$ if we take $k = h^{-1} \circ g$; it then suffices to apply the shepherd's principle.

```
Lemma number_of_partitions5: forall p E f g h,
  finite_int_fam p -> cardinal_sum p = cardinal E ->
  partition_with_pi_elements p E f ->
  inc h (set_of_permutations E) -> inc g (set_of_permutations E) ->
  (set_of_partitions_aux p E f g = set_of_partitions_aux p E f h) =
  (forall i, inc i (domain p) -> image_by_fun (compose (inverse_fun h)
    g) (W i f) = (W i f)). (* 38 *)

Lemma number_of_partitions6: forall p E f h,
  finite_int_fam p -> cardinal_sum p = cardinal E ->
  partition_with_pi_elements p E f ->
  inc h (set_of_permutations E) ->
  transf_axioms (fun g=> L (domain p)(fun i=>  (restriction2
    (compose (inverse_fun h) g)
    (V i f) (V i f))))
  (Zo  (set_of_permutations E)
    (fun g => (set_of_partitions_aux f g = set_of_partitions_aux f h)))
  (productb (L (domain p)(fun i=> (set_of_permutations (V i f))))).


Lemma number_of_partitions7: forall p E f h, (* 130 *)
  finite_int_fam p -> cardinal_sum p = cardinal E ->
  partition_with_pi_elements p E f ->
  inc h (set_of_permutations E) ->
  bijective(BL (fun g=> L (domain p)(fun i=> (restriction2
    (compose (inverse_fun h) g)
    (V i f) (V i f))))
  (Zo  (set_of_permutations E)
    (fun g => (set_of_partitions_aux f g = set_of_partitions_aux f h)))
  (productb (L (domain p)(fun i=> (set_of_permutations (V i f)))))).
```

We give here the long and the short version of the theorem.

```
Theorem number_of_partitions: forall p E,
  finite_int_fam p -> cardinal_sum p = cardinal E ->
  let num:= factorialC (cardinal E) in
    let den := cardinal_prod (L (domain p) (fun z => factorial (V z p)))
      in (num = card_mult (cardinal (set_of_partitions p E)) den &
        is_finite_c num & is_finite_c den & den <> card_zero &
        is_finite_set (set_of_partitions p E)).  (* 63 *)

Theorem number_of_partitions_bis: forall p E,
  finite_int_fam p -> cardinal_sum p = cardinal E ->
  cardinal (set_of_partitions p E) =
  card_quo (factorialC (cardinal E))
           (cardinal_prod (L (domain p) (fun z => factorialC (V z p)))).
```

We consider here the special case where the family has two elements m and p, so that $n = m + p$.

```
Lemma number_of_partitions_p2:
 forall E m p, inc m Bnat -> inc p Bnat -> cardinal E = (card_plus m p) ->
 let num := factorial (card_plus m p) in
```

```
let den := card_mult (factorial m) (factorial p) in
let x := cardinal (set_of_partitions (variantLc m p) E) in
  inc x Bnat & num = card_mult x den &
     inc num Bnat & inc den Bnat & den <> card_zero.
```

6.8.4 The binomial coefficient

Bourbaki defines the *binomial coefficient* $b_{np} = \binom{n}{p}$ as the number of subsets of p elements in a set of n elements, after showing that

$$b_{n,p} = \frac{n!}{p!(n-p)!} \text{ if } p \le n, \qquad b_{n,p} = 0 \text{ otherwise.} \tag{1}$$

He shows in Proposition 13 [3, p. 181]) that it satisfies the following relation.

$$\binom{n}{0} = 1, \quad \binom{0}{p+1} = 0, \qquad \binom{n+1}{p+1} = \binom{n}{p+1} + \binom{n}{p}. \tag{2}$$

Our initial implementation used induction on the type *nat* in order to define the function *binom* below via (2); we showed showed that it satisfies (1). The *ssreflect* library provides a function *bin*, recursively defined by *bin_rec*, with exactly the same properties. We show here the power of the library by giving the proof od the following relation

$$\sum_{i=0}^{n} \binom{n}{i} a^{n-i} b^i = (a+b)^n. \tag{3}$$

```
(*
 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.

Fixpoint bin_rec (m n : nat) {struct m} :=
  match m, n with
  | m'.+1, n'.+1 => bin_rec m' n + bin_rec m' n'
  | _, 0 => 1
  | 0, _.+1 => 0
  end.
Theorem pascal : forall a b n,
  (a + b) ^ n = \sum_(i < n.+1) (bin n i * (a ^ (n - i) * b ^ i)).
Proof.
move=> a b; elim=> [|n IHn]; first by rewrite big_ord_recl big_ord0.
rewrite big_ord_recr big_ord_recl /= expnS {}IHn muln_addl !big_distrr.
rewrite big_ord_recl big_ord_recr /= !bin0 !binn !subn0 !subnn !mul1n !muln1.
rewrite -!expnS addnA; congr (_ + _); rewrite -addnA -big_split; congr (_ + _).
apply: eq_bigr => i _ /=; rewrite 2!(mulnCA b) (mulnCA a) (mulnA a) -!expnS.
by rewrite -leq_subS ?ltn_ord // -muln_addl -binS.
Qed.
*)
```

We define here a function c_{np} by induction on n; the definition is a bit obscure since we define by induction an auxiliary term f_n, where f_n is a functional graph on **N**, then say $c_{np} = f_n(p)$.

```
Definition binom n m :=
  V m (induction_term
      (fun _ T: Set => L Bnat (fun z => Yo (z = card_zero) card_one
          (card_plus (V z T) (V (predc z) T))))
       (L Bnat (fun z => Yo  (z = card_zero) card_one card_zero))
       n).
Lemma binom00:  binom card_zero card_zero = card_one.
Lemma binom0Sm:  forall m, inc m Bnat ->
  binom card_zero (succ m) = card_zero.
Lemma binomSn0:  forall n, inc n Bnat->
  binom (succ n) card_zero = card_one.
Lemma binomSnSm:  forall n m, inc n Bnat-> inc m Bnat ->
  binom (succ n) (succ m) =
   card_plus (binom n (succ m)) (binom n m).
Lemma binom_integer: forall n m, inc n Bnat -> inc m Bnat ->
  inc (binom n m) Bnat.
```

Let $f(n,p)$ be the product of c_{np} by $p!(n-p)!$. Note that if $p > n$, then $n-p$ is zero by convention and $(n-p)!$ is one. We have then $(n-p)! = \text{if}(p < n, n-p, 1)\cdot(n-(p+1))!$. This gives us an induction property for f, from which we deduce $f(n,p) = \text{if}(p \le n, n!, 0)$. We restate this as: if $p > n$ the binomial coefficient is zero, else $f(n,p) = n!$. This formula shows that $p!(n-p)!$ divides $n!$.

```
Lemma binom_alt_pr: forall n m, inc n Bnat -> inc m Bnat ->
  card_mult (binom n m)
     (card_mult  (factorial m) (factorial (card_sub n m))) =
  Yo (cardinal_le m n) (factorial n) card_zero.  (* 108 *)
Lemma binom_bad: forall n m, inc n Bnat -> inc m Bnat ->
  cardinal_lt n m -> binom n m = card_zero.

Lemma binom_good: forall n m, inc n Bnat -> inc m Bnat ->
  cardinal_le m n ->
  card_mult (binom n m)
     (card_mult  (factorial m) (factorial (card_sub n m))) =
       (factorial n).
```

We then have

$$\binom{n}{p} = \binom{n}{n-p} = \frac{n!}{p!(n-p)!} \quad \text{when } p \le n. \tag{4}$$

```
Lemma binom_pr0: forall n p, inc n Bnat -> inc p Bnat ->
  cardinal_le p n ->
  let num := (factorial n) in
  let den:=  card_mult (factorial p) (factorial (card_sub n p)) in
  BNdivides den num & binom n p = card_quo num den.
Lemma binom_pr1: forall n p, inc n Bnat -> inc p Bnat ->
  cardinal_le p n ->
  binom n p =
   card_quo (factorial n) (card_mult (factorial p) (factorial (card_sub n p))).

Lemma binom_symmetric:forall n p, inc n Bnat -> inc p Bnat ->
  cardinal_le p n ->
  binom n p = binom n (card_sub n p).
```

We show here

$$\binom{n}{0} = 1, \qquad \binom{n}{1} = n, \qquad \binom{n+1}{2} = \frac{n(n+1)}{2}. \tag{5}$$

If $p \leq n$, the binomial coefficient is non-zero; if $p = n$ it is one, and if $p \leq n+1$ it is a stricly increasing function of n.

```
Lemma binom0: forall n, inc n Bnat -> binom n card_zero = card_one.
Lemma binom1: forall n, inc n Bnat -> binom n card_one = n.
Lemma binom2a: forall n, inc n Bnat ->
  card_mult card_two (binom (succ n) card_two) = card_mult n (succ n).
Lemma binom2: forall n, inc n Bnat ->
  binom (succ n) card_two = card_quo (card_mult n (succ n)) card_two.
Lemma binom_nn : forall n, inc n Bnat -> binom n n = card_one.
Lemma binom_pr3: forall n p,  inc n Bnat -> inc p Bnat ->
  cardinal_le p n ->
  binom n p <> card_zero.

Lemma finite_sum4_lt: forall a b, inc a Bnat -> inc b Bnat ->
   a <> card_zero -> cardinal_lt b (card_plus b a).
Lemma binom_monotone1: forall k n m,
  inc k Bnat -> inc n Bnat -> inc m Bnat ->
  k <> card_zero -> cardinal_le k (succ n) -> cardinal_lt n m ->
  cardinal_lt (binom n k) (binom m k).  (* 26 *)
Lemma binom_monotone2: forall k n m,
  inc k Bnat -> inc n Bnat -> inc m Bnat ->
  k <> card_zero -> cardinal_le k (succ n) ->  cardinal_le k (succ m) ->
  (cardinal_lt n m =  cardinal_lt (binom n k) (binom m k)).
```

The last lemma of the previous section says that that $c_{m+p,p}$ is the number of partitions of a set E with $n = m + p$ elements into two sets with m and p elements. For completeness, we show that $c_{n,p}$ is the number of partitions of E into two sets with $n - p$ and p elements (if $p > n$, this number is zero, there is no partition of E into two subsets with p and k elements, whatever k). Let Q_p be the set of all subsets A of E that have p elements. If $\mathrm{A} \in \mathrm{Q}_p$ then $\mathrm{E} - \mathrm{A} \in \mathrm{Q}_{n-p}$ and $\mathrm{A} \to \mathrm{E} - \mathrm{A}$ is a bijection. Moreover $(\mathrm{A}, \mathrm{E} - \mathrm{A})$ is a partition with $(p, n - p)$ elements. The cardinal of Q_p is b_{np}, the Bourbaki definition of the binomial coefficient. Thus $b_{np} = c_{np}$ whenever $p \leq n$. The relation also holds if $p > n$ since Q_p is empty and $c_{np} = 0$.

```
Lemma number_of_partitions_p3: forall E m p, inc m Bnat -> inc p Bnat ->
  cardinal E = card_plus m p ->
  cardinal (set_of_partitions (variantLc m p) E) =
  binom (card_plus m p) m.
Lemma number_of_partitions_p4: forall E n m, inc n Bnat -> inc m Bnat ->
  cardinal E = n ->
  cardinal (set_of_partitions (variantLc m (card_sub n m)) E) =
  binom n m.

Definition subsets_with_p_elements p E:=
  Zo (powerset E)(fun z=> cardinal z =p).
Lemma cardinal_complement: forall A E, sub A E ->
  card_plus (cardinal A) (cardinal (complement E A))  = cardinal E.
Lemma cardinal_complement1:forall n p E A, inc n Bnat -> inc p Bnat ->
  cardinal E = n -> cardinal A = p -> sub A E ->
  cardinal (complement E A) = card_sub n p.
```

```
Lemma subsets_with_p_elements_pr: forall n p E, inc n Bnat -> inc p Bnat ->
  cardinal E = n ->
  binom n p = cardinal (subsets_with_p_elements p E).   (* 52 *)
Lemma subsets_with_p_elements_pr0: forall n p, inc n Bnat -> inc p Bnat ->
  binom n p = cardinal (subsets_with_p_elements p n).
Lemma bijective_complement: forall n p E, inc n Bnat -> inc p Bnat ->
  cardinal_le p n -> cardinal E = n ->
  bijective(BL (fun z => complement E z)
    (subsets_with_p_elements p E)(subsets_with_p_elements (card_sub n p) E)).
```

The following formula is true whenever a_i are permutable elements of a ring

$$\sum_{\sum p_i = n} \frac{n!}{p_1! \cdots p_k!} a_1^{p_1} \ldots a_k^{p_k} = (a_1 + a_2 + \cdots + a_k)^n. \tag{6}$$

Here $\sum p_i$ means $p_1 + p_2 + \cdots + p_k$. If we denote the sum $p_1 + p_2 + \cdots + p_{k-1}$ by $\sum p_j$, factor out powers of a_k and use associativity, the previous expression becomes

$$\sum_m \left(\sum_{\sum p_j = m} \frac{m!}{p_1! \cdots p_{k-1}!} a_1^{p_1} \ldots a_{k-1}^{p_{k-1}} \right) \binom{n}{m} a_k^{n-m}$$

We can proceed by induction on k, starting with $k = 2$, for which we use induction on n. We show here an alternate method, valid only when the quantities a_i are integers. In the special case $a_i = 1$ the formula reduces to

$$\sum_{\sum p_i = n} \frac{n!}{p_1! \cdots p_k!} = k^n, \qquad \sum_p \binom{n}{p} = 2^n. \tag{7}$$

Let E be a finite set of cardinal n, I an index set, and $\mathrm{A}_{n\mathrm{I}}$ be the set of all mappings p with $\sum_{i \in \mathrm{I}} p(i) = n$. This relation implies that $p(i) \in [0, n]$. Thus we complete the defintion of $\mathrm{A}_{n\mathrm{I}}$ by requiring that the target of p is the interval $[0, n]$. We shall compute the cardinal of this set later on.

Let $\mathrm{B}_{n\mathrm{I}}$ be the set of graphs of elements of $\mathrm{A}_{n\mathrm{I}}$. We have that $p \in \mathrm{B}_{n\mathrm{I}}$ if and only if p is a functional graph defined on I and $\sum p_i = n$. There is a similar property for graphs with Thus we can rewrite (6) and (7) with $\sum p_i \leq n$. $p \in \mathrm{B}_{n\mathrm{I}}$ instead of $\sum p_i = n$.

```
Definition set_of_functions_sum_eq F n:=
  Zo (set_of_functions F (interval_Bnat card_zero n))
  (fun z=> (cardinal_sum (P z)) = n).
Definition set_of_functions_sum_eq E n:=
  Zo (set_of_functions E (interval_Bnat card_zero n))
  (fun z=> (cardinal_sum (P z)) = n).

Definition set_of_graph_sum_eq F n:=
  fun_image (set_of_functions_sum_eq F n) graph.
Definition set_of_graph_sum_eq F n:=
  fun_image (set_of_functions_sum_eq F n) graph.

Lemma setof_suml_aux: forall F n f, inc n Bnat ->
  inc f (set_of_graph_sum_le F n) =
  (fgraph f & domain f = F & cardinal_le (cardinal_sum f) n &
    (forall i, inc i (domain f) -> is_cardinal (V i f))).
```

```
Lemma setof_sume_aux: forall F n f, inc n Bnat ->
  inc f (set_of_graph_sum_eq F n) =
  (fgraph f & domain f = F & cardinal_sum f = n &
    (forall i, inc i (domain f) -> is_cardinal (V i f))).
```

Consider now a finite sequence of integers a_i (We shall see in the next chapter that if a_i is inifnite, both terms in (6) are equal to the greatest element element of the family, the case where the number of terms in infinite can also be handled, bu the result has no great interest). We let k be the cardinal of the index set I, and consider a set F whose cardinal is $\sum a_i$. We know that there is a partition F_i of F with $\mathrm{Card}(\mathrm{F}_i) = a_i$. If f is a function $\mathrm{E} \to \mathrm{F}$, we consider the set $f_i = f^{-1}\langle \mathrm{F}_i \rangle$ of all elements of E such that $f(x) \in \mathrm{F}_i$. Denote by $\phi(f)$ the mapping $i \to \mathrm{Card}\, f_i$, and by $\Phi(f)$ the mapping $i \to f_i$. We have $\phi(f) \in \mathrm{B}_{n\mathrm{I}}$ and $\Phi(f) \in \mathrm{C}_{\phi(f),\mathrm{E}}$, where $\mathrm{C}_{p\mathrm{E}}$ denotes the set of all partitions with p_i elements of E.

If P is in $\mathrm{C}_{p\mathrm{E}}$ we consider a family of bijections $h_i : [1, p_i] \to \mathrm{P}_i$. If we take $\mathrm{P} = \Phi(f)$, the function $f \circ h_i$ maps $[1, p_i]$ to F_i. Its restriction to F_i is a element of $\mathscr{F}([1, p_i], \mathrm{F}_i)$. Denote it by $\Psi(f)(i)$. Now $\Psi(f) \in \prod \mathscr{F}([1, p_i], \mathrm{F}_i)$. The mapping $f \to (\Phi(f), \Psi(f)$ is a bijection (for fixed $\phi(f)$), as can be easily shown (the proof is a bit technical, thus long).

The case $a_i = 1$ is a bit easier. Here each F_i has one element, and we can identity Φ and Ψ (given a partition E_i of E, if $\mathrm{F}_i = \{y_i\}$, f maps E_i into F_i if and only if $f(x) = y_i$ for $x \in \mathrm{E}_i$). This makes the proof much shorter.

```
Lemma sum_of_gen_binom: forall E F n, inc n Bnat -> cardinal E = n ->
  cardinal_sum (L (set_of_graph_sum_eq F n)
              (fun p => cardinal (set_of_partitions p E)))
  = card_pow (cardinal F) n.  (* 115 *)

Lemma sum_of_gen_binom0: forall E n a, (* 251 *)
  inc n Bnat ->  cardinal E = n -> finite_int_fam a ->
  card_pow (cardinal_sum a) n =
     cardinal_sum  (L (set_of_graph_sum_eq (domain a) n)
       (fun p =>
         card_mult (cardinal (set_of_partitions p E))
            (cardinal_prod (L (domain a) (fun i=> (card_pow (V i a) (V i p)))))))).
```

We consider now a special case where F is the canonical doubleton. Its cardinal is 2. To each $m \in [0, n]$ we can associate the function defined on F that maps the first element to m and the second to $n - m$. This is a bijection onto $\mathrm{B}_{n\mathrm{F}}$, and we can use it to perform a change of variables in the sum, and we can apply *number_of_partitions_p4*, and we get the binomial coefficient (Formula (7b). We give an alternate proof: with count the number of subset of E, according to the their cardinal p.

We prove the Pascal formula (3) by induction. If we replace n by $n+1$, isolate the term $p = 0$, write $p = k+1$ and use the binomial relation we get

$$\sum_{i=0}^{n} \binom{n}{k} a^{k+1} b^{n+1-k-1} + \sum_{i=0}^{n} \binom{n}{k+1} a^{k+1} b^{n+1-k-1} + b^{n+1}$$

. We re-introduce p in the second sum, factor out a and b and we get

$$a \left[\sum_{i=0}^{n} \binom{n}{k} a^k b^{n-k} \right] + \left[\sum_{i=0}^{n} \binom{n}{p} a^p b^{n-p} \right] b.$$

It suffices to apply the induction hypothesis. The proof is similar to that given above, just much longer.

```
Lemma sum_of_gen_binom2: forall n, inc n Bnat ->
  cardinal_sum (L  (interval_Bnat card_zero n) (binom n))
  = card_pow card_two n. (* 47 *)
Lemma sum_of_binomial: forall n, inc n Bnat ->
  cardinal_sum (L (interval_Bnat card_zero n) (binom n))
  = card_pow card_two n.  (* 43 *)

Lemma sum_of_binomal2: forall a b n,
  is_cardinal a -> is_cardinal b -> inc n Bnat ->
  cardinal_sum (L (interval_Bnat card_zero n)
         (fun p => card_mult (binom n p)
            (card_mult (card_pow a p) (card_pow b (card_sub n p)))))
  = card_pow (card_plus a b) n.   (* 105 *)
```

6.8.5 Number of increasing functions

Consider two finite totally ordered sets E and F. Denote by $\mathscr{S}(\mathrm{E},\mathrm{F})$ the set of strictly increasing mappings from E into F, and by $\mathscr{A}(\mathrm{E},\mathrm{F})$ the set of increasing mappings from E into F. We pretend that

$$\mathrm{Card}(\mathscr{S}(\mathrm{E},\mathrm{F})) = \binom{\mathrm{Card}(\mathrm{F})}{\mathrm{Card}(\mathrm{F})}.$$

$$\mathrm{Card}(\mathscr{A}(\mathrm{E},\mathrm{F})) = \binom{n+p-1}{p} \text{ if } \mathrm{Card}(\mathrm{E}) = p \text{ and } \mathrm{Card}(\mathrm{F}) = n.$$

The first relation is a consequence of the the fact that that the mapping $f \mapsto \mathrm{R}(f)$ is a bijection from $\mathscr{S}(\mathrm{E},\mathrm{F})$ onto the set of subsets with p elements of F, where $\mathrm{R}(f)$ denotes the range of f. It requires Theorem 3 (§ 2, no. 5) (uniqueness of isomorphisms between well-ordered sets). [note; if u and v are two strictlty increasing functions with the same source range, target, they mmust be equal. For otherwise there would exists a list element x such that $u(x) \neq v(x)$ since the source is finite and totally ordered. We have $u(x) = v(a)$ for some a. If $a < x$ we get $v(a) = u(a) = u(x)$, hence $a = x$ by injectivtity of u. The case $a = x$ is also excluded, so that $x < a$ and $v(x) < v(a)$, thus $v(x) < u(x)$. By symmetry $u(x) < v(x)$, absurd.

```
Definition set_of_incr_functions r r' :=
  (Zo (set_of_functions (substrate r) (substrate r'))
    (fun z => increasing_fun z r r')).
Definition set_of_strict_incr_functions r r' :=
  (Zo (set_of_functions (substrate r) (substrate r'))
    (fun z => strict_increasing_fun z r r')).

Lemma cardinal_set_of_increasing_functions1: forall r r' f,
  total_order r -> strict_increasing_fun f r r' ->
  (order_morphism f r r' & cardinal (substrate r)  = cardinal (image_of_fun f)).

Lemma cardinal_set_of_increasing_functions2: forall r r',
  total_order r -> total_order r' ->
  is_finite_set (substrate r)  ->  is_finite_set (substrate r')  ->
  bijective(BL (fun z  => range (graph z))
    (set_of_strict_incr_functions r r')
```

```
    (subsets_with_p_elements (cardinal (substrate r)) (substrate r'))). (* 91 *)

Lemma cardinal_set_of_increasing_functions: forall r r',
  total_order r -> total_order r' ->
  is_finite_set (substrate r)  ->  is_finite_set (substrate r')  ->
  cardinal (set_of_strict_incr_functions r r')
  = binom (cardinal (substrate r')) (cardinal (substrate r)).
```

We shall give some variants of this property. We start with the case where E is a segment of **N**, say an interval $[0,p]$. If f is a function with values in an ordered set F, if $f(x) \leq f(x+1)$ for $x < p$, then $f(x) \leq_f (y)$ whenever $x \leq y \leq p$. The same holds if $\leq$ is replaced by $<$; the proof is obvious by induction. In the case of a strictly increasing function, we have an injection, since the source is totally ordered.

```
Lemma increasing_prop0 : forall p f r, inc p Bnat -> order r ->
  (forall i, cardinal_le i p -> inc (f i) (substrate r)) ->
  (forall n,  cardinal_lt n p -> gle r (f n) (f (succ n))) ->
  (forall i j, cardinal_le i j -> cardinal_le j p ->
    gle r (f i) (f j)).   (* 21 *)
Lemma increasing_prop1 : forall p f, inc p Bnat ->
  (forall i, cardinal_le i p -> inc (f i) Bnat) ->
  (forall n,  cardinal_lt n p -> cardinal_le (f n) (f (succ n))) ->
  (forall i j, cardinal_le i j -> cardinal_le j p ->
    cardinal_le (f i) (f j)).
 Lemma strict_increasing_prop0 : forall p f r, inc p Bnat -> order r ->
  (forall n,  cardinal_lt n p -> glt r (f n) (f (succ n))) ->
  (forall i j, cardinal_lt i j -> cardinal_le j p ->
    glt r (f i) (f j)).  (* 29 *)

Lemma increasing_prop : forall p f r, inc p Bnat -> is_function f ->
  source f =  interval_co_0a (succ p) -> order r -> substrate r = target f ->
  (forall n,  cardinal_lt n p -> gle r (W n f) (W (succ n) f)) ->
  increasing_fun f (interval_Bnato card_zero p) r.

Lemma strict_increasing_prop : forall p f r, inc p Bnat -> is_function f ->
  source f =  interval_co_0a (succ p) -> order r -> substrate r = target f ->
  (forall n,  cardinal_lt n p -> glt r (W n f) (W (succ n) f)) ->
  (injective f &
    strict_increasing_fun f (interval_Bnato card_zero p) r).
```

Denote by I_p the interval $[0,p[$. Assume now that f is strictly increasing $I_p \to \mathbf{N}$. Then $x \leq f(x)$ on I_p by induction on x. The function $i \mapsto f(i) - i$ is decreasing. Assume f maps I_p into I_{n+p}. When $i < p$ we have $f(i) - i \leq f(p-1) - (p-1) < n+p-p+1$, hence $f(i) - i \in I_{n+1}$.

```
Lemma strict_increasing_prop1: forall f p,
  inc p Bnat ->  (forall i, cardinal_lt i p -> inc (f i) Bnat)
  -> (forall i j, cardinal_lt i j -> cardinal_lt j p ->
    cardinal_lt (f i) (f j)) ->
  (forall i,  cardinal_lt i p -> cardinal_le i (f i)).
Lemma strict_increasing_prop2: forall f p,
  inc p Bnat ->  (forall i, cardinal_lt i p -> inc (f i) Bnat)
  -> (forall i j, cardinal_lt i j -> cardinal_lt j p ->
    cardinal_lt (f i) (f j)) ->
  (forall i j, cardinal_le i j -> cardinal_lt j p ->
    cardinal_le (card_sub (f i) i) (card_sub (f j) j)). (* 41 *)
```

```
Lemma strict_increasing_prop3: forall f p n,
  inc p Bnat -> inc n Bnat -> (forall i, cardinal_lt i p -> inc (f i) Bnat)
  -> (forall i j, cardinal_lt i j -> cardinal_lt j p ->
    cardinal_lt (f i) (f j)) ->
  (forall i, cardinal_lt i p -> cardinal_lt (f i) (card_plus n p)) ->
  (forall i, cardinal_lt i p -> cardinal_le (card_sub (f i) i) n).
```

If f is a mapping, denote by $s(f)$ the mapping $i \mapsto f(i) - i$, and by $a(f) : i \mapsto f(i) + i$. We have shown that s is a mapping from $\mathscr{S}(\mathrm{I}_p, \mathrm{I}_{n+p})$ into $\mathscr{A}(\mathrm{I}_p, \mathrm{I}_{n+1})$. It is a bijection with inverse a. Thus we get

$$\mathrm{Card}(\mathscr{A}(\mathrm{I}_p, \mathrm{I}_{n+1})) = \mathrm{Card}(\mathscr{S}(\mathrm{I}_p, \mathrm{I}_{n+p})) = \binom{n+p}{p}.$$

```
Lemma cardinal_set_of_increasing_functions3: forall n p,
  let r:= interval_Bnatco (nat_to_B p) in
    let r':= interval_Bnato card_zero (nat_to_B n) in
      cardinal (Zo (set_of_functions (substrate r) (substrate r'))
        (fun z => increasing_fun  z r r')) =
      nat_to_B(binom (n+p) p). (172 *)
```

We compute the cardinal of $\mathscr{A}(\mathrm{E}, \mathrm{F})$ in the general case. If F is empty, there is a unique function $\mathrm{E} \to \mathrm{F}$ if E is empty, and none if E is non-empty. Assume that E has p elements anfd F has $n > 0$ elements. There is a unique order isomorphism f between between E and $[0, p[$, and g between F and $[0, n-1]$. Assume $h : \mathrm{E} \to \mathrm{F}$ increasing. Then $k = g \circ h \circ f^{-1} : [0, p[\to [0, n[$. Since $h = g^{-1} \circ k \circ f$, we have that h is increasing if and only if k is increasing, thanjks to the additional lemmas.This gives an isomorphism between $\mathscr{A}(\mathrm{E}, \mathrm{F})$ and $\mathscr{A}(\mathrm{I}_p, \mathrm{I}_n)$, and these two sets have the same number of elements.

```
Lemma increasing_compose: forall f g r r' r'',
  increasing_fun f r r' ->  increasing_fun g r' r'' ->
  (composable g f &
    (forall x, inc x (source f) -> W x (compose g f) = W (W x f) g) &
    increasing_fun (compose g f) r r'').

Lemma increasing_compose3: forall f g h r r' r'' r''',
  strict_increasing_fun f r r' ->  increasing_fun g r' r'' ->
  strict_increasing_fun h r'' r''' ->
  let res:= compose (compose h g) f in
    (inc res (set_of_functions (source f) (target h)) &
      (forall x, inc x (source f) -> W x res = W (W (W x f) g) h) &
      increasing_fun res r r''').

Lemma cardinal_set_of_increasing_functions4: forall r r',
  let n :=  (cardinal (substrate r')) in
    let p := (cardinal (substrate r)) in
      total_order r -> total_order r' ->
      is_finite_set (substrate r)  ->  is_finite_set (substrate r')  ->
      cardinal (set_of_incr_functions r r')
      = binom (card_sub (card_plus n p) card_one) p. (* 116 *)
```

We compute now the number a_n of pairs (i, j) such that $1 \le i \le j \le n$, and the number b_n of pairs satisfying $1 \le i < j \le n$. This is the number of increasing (resp. strictly increasing) mappings of a set with two elements into the interval $[1, n]$, which has n elements. Our

previous results show

$$a_n = \frac{n(n+1)}{2} = \binom{n+1}{2}, \qquad b_n = \frac{n(n-1)}{2} = \binom{n}{2}.$$

The Bourbaki proof of these relations (Proposition 14 [3, p. 181]) is different. He notices that $a_n = b_n + n$ since $i \leq j$ is equivalent to $i < j$ or $i \leq j$. A subset of $[1, n]$ is of cardinal two if and only if it is a doubleton $\{i, j\}$ with $i \neq j$, and we may assume $i < j$; hence b_n is the number of subsets of cardinal two of $[1, n]$. The link between a_n and b_n is given by the following trivial relation:

$$\binom{n+1}{2} = \frac{n(n+1)}{2} = \binom{n}{2} + n.$$

```
Lemma binom_2plus: forall n, inc n Bnat ->
  binom (succ n) card_two = card_quo (card_mult n (succ n)) card_two.
Lemma binom_2plus0: forall n, inc n Bnat ->
  binom (succ n) card_two = card_plus (binom n card_two) n.

Lemma cardinal_pairs_lt: forall n, inc n Bnat ->
  cardinal(Zo (product Bnat Bnat)
    (fun z=> cardinal_le card_one (P z) &
      cardinal_lt (P z) (Q z) & cardinal_le (Q z) n))=
    (binom n card_two).  (* 55 *)

Lemma cardinal_pairs_le: forall n, inc n Bnat ->
  cardinal(Zo (product Bnat Bnat)
    (fun z=> cardinal_le card_one (P z) &
      cardinal_le (P z) (Q z) & cardinal_le (Q z) n)) =
     (binom (succ n) card_two).  (* 26 *)
```

A corollary is the following formula

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \binom{n+1}{2}$$

This formula is obvious by induction on the type *nat*; the prof by induction on **N** is a bit longer. Let s_n be the sum and s'_n be the sum $\sum_{i \leq n}(n-i)$. By re-ordering indices, we have $s'_n = s_n$: moreover $s_n + s'_n = \sum_{i \leq n} n$, which is $n(n+1)$, and we get the result by division by two; finally, we follow Bourbaki, showing that s_n is the cardinal of the set E of all pairs (i, j) with $1 \leq i \leq j \leq n$, since E is the union of the sets $[1, j] \times \{j\}$, i.e., the disjoint union of the intervals $[1, j]$, which are of cardinal j.

```
Lemma fct_sum_const1: forall f n m,
  inc n Bnat ->  (forall i, cardinal_lt i n -> f i = m) ->
  cardinal_sum (L (interval_co_0a n) f) = card_mult n m.
Lemma sum_of_i: forall n, inc n Bnat ->
  cardinal_sum (L (interval_co_0a n) (fun i=>i)) =
   binom n card_two.
Lemma sum_of_i3: forall n, inc n Bnat ->
  cardinal_sum (L (interval_co_0a n) (fun i=>i)) =
   binom n card_two.  (* 31 *)
Lemma sum_of_i2: forall n,  inc n Bnat ->
  cardinal_sum (L (interval_Bnat card_one n) (fun i=>i)) =
  (binom (succ n) card_two).   (* 27 *)
```

6.8.6 Number of monomials

Consider a set E, a law of composition of E, and elements x, y, z, etc, of E. Consider a combination of these variables, where x appears 3 times, z appears twice and y appears once. If the law is associative and commutative, the combination is equal to $x\cdot(x\cdot(x\cdot(y\cdot(z\cdot z))))$. This is called a monomial, and denoted by x^3yz^2. The total number of terms (here six) is called the degree. Assume that we have a second law of composition $a+b$, and that the usual rules apply. This means that $(a+b)^n$ can be expanded as sum monomials: $(a+b)^n = \sum \gamma_{ij} a^i b^j$. It happens that $\gamma_{ij} = \binom{n}{i}$ if $i+j=n$ (this is the explanation of the term "binomial coefficient"). More generally, $(\sum x_i)^n = \sum_{\mathrm{I}\in \mathrm{S}_n} \Gamma_{\mathrm{I}} x^{\mathrm{I}}$, where I is a mapping $i \mapsto n_i$, x^{I} denotes the monomial $x_1^{n_1}x_2^{n_2}\cdots x_p^{n_p}$. The total degree of the monomial is $\sum n_i = n$. Taking $a=b=1$, gives $\sum_p \binom{n}{p} = 2^n$. Taking $a=-1$ and $b=1$ gives $\sum_p (-1)^p \binom{n}{p} = 0$ (The first result has already been proved, the second is the object of Exercice 5.2). We have also $\sum_{\mathrm{I}\in \mathrm{S}_n} \Gamma_{\mathrm{I}} = p^n$ (it can be shown, by induction of the number of variables, that Γ_{I} is the number of coverings of a set with n elements by subsets with n_i elements). The cardinal of the set S_n is the object of the next theorem. We compute it by induction on both n and p.

Let E be a set with h elements, $\bar{\mathrm{A}}_n$ and $\bar{\mathrm{B}}_n$ be the sets of functions u with $\sum_{i\in \mathrm{E}} u(i) \le n$ and $\sum_{i\in \mathrm{E}} u(i) = n$ respectively. Let A_{nh} and B_{nh} the cardinals of these sets. Proposition 15 [3, p. 182]) says

$$\mathrm{A}_{nh} = \binom{n+h}{h} \qquad \mathrm{B}_{nh} = \binom{n+h-1}{h-1}.$$

We have $\mathrm{A}_{nh} = \mathrm{B}_{nh} + \mathrm{A}_{n-1,h}$ since $\bar{\mathrm{A}}_n$ is the disjoint union of $\bar{\mathrm{B}}_n$ and $\bar{\mathrm{A}}_{n-1}$ If $x \notin \mathrm{E}$, every function u such that $\sum u(i) \le n$ can be uniquely extented to $\mathrm{E}\cup\{x\}$ in such a way as $\sum_{i\in \mathrm{E}\cup\{x\}} u(i) = n$. This gives $\mathrm{B}_{n,h+1} = \mathrm{A}_{nh}$. The formulas follows by induction (they are trivial for $h=0$ and $n=0$). One difficulty of the Bourbaki's proof is that he has not yet defined the set of integers; as a consequence, he adds the condition that the target of u is the interval $[0,n]$, so that $\bar{\mathrm{A}}_{n-1}$ is not a subset of $\bar{\mathrm{A}}_n$; it is nevertheless isomorphic to the complement of $\bar{\mathrm{B}}_n$ in $\bar{\mathrm{A}}_n$. There are two other solutions: we may consider functions with target **N**, or graphs of functions. Since we already introduced the set of graphs of functions such that $\sum u_i = n$, we use graphs. We first show that the sets have the same number of elements.

```
Lemma sof_sum_eq_equi: forall F n, inc n Bnat ->
  equipotent (set_of_functions_sum_eq F n) (set_of_graph_sum_eq F n).
Lemma sof_sum_le_equi: forall F n, inc n Bnat ->
  equipotent (set_of_functions_sum_le F n) (set_of_graph_sum_le F n).

Lemma set_of_functions_sum0: forall f,
  (forall a,  f 0 a = card_one) ->
  (forall a , f a 0 = card_one) ->
  (forall a b, f (S a) (S b) = card_plus  (f (S a) b) (f a (S b))) ->
  forall a b, f a b = nat_to_B(binom (a+b) a).

Lemma set_of_functions_sum1: forall E x n,
  inc n Bnat -> ~ (inc x E) ->
  equipotent (set_of_functions_sum_le E n)
  (set_of_functions_sum_eq (tack_on E x) n). (* 55 * )

Lemma set_of_functions_sum2: forall E n, inc n Bnat ->
 cardinal(set_of_graph_sum_le E (succ n))
 = card_plus (cardinal (set_of_graph_sum_eq E (succ n)))
     (cardinal (set_of_graph_sum_le E n)).  (* 21 *)
```

```
Lemma set_of_functions_sum3: forall E,
  cardinal (set_of_functions_sum_le E card_zero) = card_one.  (* 18 *)
Lemma set_of_functions_sum4: forall n, is_cardinal n->
    cardinal (set_of_functions_sum_le emptyset n) = card_one.

Lemma set_of_functions_sum_pr: forall n h,
  inc n Bnat -> inc h Bnat ->
  let intv:= fun h => (interval_co_0a h) in
    let sle:= fun n h => set_of_graph_sum_le (intv h) n in
      let seq := fun n h =>  set_of_graph_sum_eq (intv h) n in
        let A:= fun n h => cardinal (sle n h) in
          let B:=  fun n h => cardinal (seq n h) in
            (A n h =  B n (succ h) & A n h = (binom (card_plus n h) n)).
```

We give now a variant of the theorem[5]. We pretend that the number of functions y defined on $[0,p]$ with values in $[0,n]$ and such that $\sum y_i \leq n$ is $A_{n,p+1} = \binom{n+p+1}{p+1}$. This is the previous result for $h = p+1$ (if $h = 0$, there is a unique function defined on a set with h elements, the empty function, and the sum is zero). The quantity $A_{n,p+1}$ is the number of subsets of $[0,n+p]$ with $p+1$ elements. Consider the sequence x_i, defined by induction (see next chapter) via $x_0 = y_0$ and $x_{i+1} = y_{i+1} + x_i + 1$. All we have to do is prove that the mapping $y \mapsto \{x_0, x_1, \ldots, x_p\}$ is bijective (as a function with values in the subset of $\mathfrak{P}([0,n+p])$ formed of sets with $p+1$ elements). The idea is that the function x is strictly increasing, and uniquely defined by its range. Since $A_{n,p+1}$ is the number of stricly increasing functions $[0,p] \to [0,n+p]$; all we have to to is to prove that the mapping $y \mapsto x$ is a bijection (as a function into $\mathscr{S}([0,p],[0,n+p])$.

The inverse mapping of $y \mapsto x$ is defined by $y_{i+1} = x_{i+1} - (x_i + 1)$. It is easier to consider $y_{i+1} = z_{i+1} - z_i$, where $z_i = x_i - i$. It happens that z_i is the sum of the restriction of y to the interval $[0,i]$ (there is no need to define it by induction) and is obviously increasing. Since $A_{n,p+1}$ is the cardinal of $\mathscr{A}([0,p],[0,n])$, the set of increasing functions $[0,p] \to [0,n]$, all we have to do is show that $y \mapsto z$ is a bijection $C_{pn} \to C'_{pn}$ where C_{pn} is the set of functions $y : [0,p] \to [0,n]$ such that $\sum y_i \leq n$ and $C'_{pn} = \mathscr{A}([0,p],[0,n])$. We first show that $A_{n,p+1}$ is the cardinal of C'_{pn}.

```
Definition set_of_graph_sum_le_int p n :=
  set_of_graph_sum_le (interval_Bnat card_zero p) n.

Definition set_of_increasing_functions_int p n :=
  (Zo (set_of_functions (interval_Bnat card_zero p)(interval_Bnat card_zero n))
    (fun z => increasing_fun z
      (interval_Bnato card_zero p)
      (interval_Bnato card_zero n))).

Lemma card_set_of_increasing_functions_int : forall p n,
  inc p Bnat -> inc n Bnat ->
  cardinal (set_of_increasing_functions_int p n) =
  binom (succ (card_plus n p)) (succ p).
```

If $R(f,i)$ denoted the restriction of the function f to the interval $[0,i]$ we have $R(R(f,i),j) = R(f,j)$ if $j \leq i$. If $S(f,i)$ is the sum of the restrition of $R(f,i)$ we have $S(f,0) = f(0)$ and $S(f,i+1) = f(i+1) + S(f,i)$.

[5] Suggested by Jean-Baptiste Pomet

```
Lemma double_restrc: forall f n p, fgraph f -> inc p Bnat ->
  cardinal_lt n p ->
  domain f = interval_Bnat card_zero p ->
  restr (restr f (interval_Bnat card_zero (succ n)))
     (interval_Bnat card_zero n) =
     restr f (interval_Bnat card_zero n).
Lemma induction_on_sum3: forall f m,
  fgraph f -> inc m Bnat ->
  domain f = interval_Bnat card_zero m ->
  (forall a, inc a (domain f) -> is_cardinal (V a f)) ->
  (cardinal_sum (restr f (interval_Bnat card_zero card_zero))
    = (V card_zero f)
    & (forall n, cardinal_le n m ->
      card_plus (cardinal_sum (restr f (interval_co_0a n))) (V n f)
      = cardinal_sum (restr f (interval_co_0a (succ n))))).  (* 29 *)
```

Given a function y, we consider z such that $z_i = \mathrm{S}(y,i)$. We first show that z maps $[0,p]$ into $[0,n]$, and then that $z \in \mathrm{C}'pn$ if $y \in \mathrm{C}_{pn}$ (nore that $i \mapsto \mathrm{S}(y,i)$ is increasing). We then show that $y \mapsto z$ is injective and surjective. The key relation is $z_0 = y_0$ and $z_{i+1} = y_{i+1} + z_i$ (trivial consequence of *induction_on_sum3*); it says that y is uniquely defined from z. Moreover, given an increasing function z', if $y_0 = z'_0$ and $y_{i+1} = z'_{i+1} - z'_i$, the same formula is satisfied by z', hence $z = z'$, thus proving surjectivity.

```
Definition sum_to_increasing_fun y :=
  fun i => cardinal_sum (restr y (interval_Bnat card_zero i)).

Definition sum_to_increasing_fct y n p :=
  BL (sum_to_increasing_fun y)
  (interval_Bnat card_zero p) (interval_Bnat card_zero n).

Lemma sum_to_increasing1: forall y n p,
  inc n Bnat -> inc p Bnat ->
  inc y (set_of_graph_sum_le_int p n) ->
  transf_axioms (sum_to_increasing_fun y)
    (interval_Bnat card_zero p)
    (interval_Bnat card_zero n).

Lemma sum_to_increasing2: forall n p,
  inc n Bnat -> inc p Bnat ->
  transf_axioms (fun y=> (sum_to_increasing_fct y n p))
  (set_of_graph_sum_le_int p n)
  (set_of_increasing_functions_int p n).  (* 46 *)

Lemma sum_to_increasing4: forall n p,
  inc n Bnat -> inc p Bnat ->
  injective (BL (fun y=> (sum_to_increasing_fct y n p))
  (set_of_graph_sum_le_int p n)
  (set_of_increasing_functions_int p n)).  (* 41 *)

Lemma sum_to_increasing5: forall n p,
  inc n Bnat -> inc p Bnat ->
  surjective (BL (fun y=> (sum_to_increasing_fct y n p))
  (set_of_functions_sum_le_int p n)
  (set_of_increasing_functions_int p n)). (* 86 *)

Lemma sum_to_increasing6: forall n p,
```

```
inc p Bnat -> inc n Bnat ->
cardinal (set_of_graph_sum_le_int p n) =
binom (succ (card_plus n p)) (succ p).
```

Chapter 7

Infinite sets

7.1 The set of natural integers

Bourbaki defines an *infinite set* as a set that is not finite. If such a set exists and $\mathfrak{a}$ is its cardinal, then all integers n satisfy $n < \mathfrak{a}$, since otherwise we would have $\mathfrak{a} \leq n$, which implies $\mathfrak{a}$ finite. This means that the set **N** of cardinals n such that n is finite contains all integers. By extensionality, it does not depend on $\mathfrak{a}$. Bourbaki has an axiom that asserts the existence of an infinite set, and deduces (Theorem 1, [3, p. 184]) that the set **N** of integers exists. Its cardinal is denoted by $\aleph_0$.

This set is infinite; in a previous version of the software, we used the fact that **N** and $\mathbb{N}$ are isomorphic. We could show that **N** is infinite using the same argument as when proving that $\mathbb{N}$ is infinite, namely that the successor function is injective, but not surjective. We consider here the Bourbaki argument: for any integer n, the cardinal of the interval $[0, n]$ is $n+1$, thus $> n$ (it is not n since n is finite). SInce $[0, n] \subset$ Ĺ**N**, we card $n < n+1 \leq \aleph_0$, so that $\aleph_0$, the cardinal of **N** vcannot be n.

```
(* Lemma equipotent_nat_Bnat: equipotent Bnat nat. *)
Lemma infinite_Bnat: is_infinite_c (cardinal Bnat).
```

Bourbaki defines a *sequence* as a family whose index set I is a subset of **N**. It is called an infinite sequence if I is infinite. Remember that a *finite sequence* is a family where I is finite and contains only integers; this means that I is a finite subset of **N**.

Let's quote Bourbaki [3, p. 184] "Let P$\{n\}$ be a relation and let I denote the set of integers n such that P$\{n\}$ is true. I is then a subset of **N**. A sequence $(x_n)_{n\in \mathrm{I}}$ is then sometimes written $(x_n)_{\mathrm{P}\{n\}}$, and x_n is called then *nth term* in the sequence." Example. Assume that P$\{n\}$ is the relation $n \in \mathbf{Z}$ where **Z** denotes the set of rational integers as a subset of **N**. According to the quote, $(x_n)_{n\in\mathbf{N}}$ and $(x_n)_{n\in\mathbf{Z}}$ are the same sequences. This may be confusing since they are obviously different families. In section 6.4, Bourbaki assumes that P$\{n\}$ implies that n is an integer; this is missing here. Other example: we consider the property "n even and $n < 10$". This is a finite sequence and the 4th term is 6; in the French version, we can read "x_4 est le terme d'indice 4" and "x_6 est le 4-ème terme". In English this is translated as "x_4 is the 4th term" and "x_6 is the 4th term". This may be confusing.

According to Bourbaki, if the property is $n \geq k$, the sequence is written as $(x_n)_{k\leq n}$ or $(x_n)_{n\geq k}$ or even (x_n) if $k = 0$ or $k = 1$. This last notation is obviously ambiguous. The sum of such a family may be denoted a $\sum_{n=k}^{\infty} x_n$.

Two sequences $(x_n)_{n\in \mathrm{I}}$ and $(y_n)_{n\in \mathrm{I}}$ with the same index set are said to *differ only in the order of their terms* if there exists a permutation f of the index set I such that $x_{f(n)} = y_n$ for all $n \in \mathrm{I}$. This makes sense even if I is not a subset of the integers. By commutativity, two sequences that differ only in the order of their terms have same sum and product.

A *multiple sequence* is a family whose index set is a subset of a product $\mathbf{N}^p$ (p is a integer).

Let f be a bijection of **N** onto a set I. For each family $(x_i)_{i\in \mathrm{I}}$, the sequence $n \mapsto x_{f(n)}$ is said to be obtained by *arranging the family* $(x_i)_{i\in \mathrm{I}}$ *in the order defined by* f.

7.2 Definition of mappings by induction

If we instantiate Criterion C60 (see page 48) to the well-ordered set **N** we get another criterion[1]; it assert that, for any term T, there exists a unique surjective function f such that

(TIND) $$\forall n, n \in \mathbf{N} \implies f(n) = \mathrm{T}\{\!\{f^{(n)}\}\!\}$$

where $u^{(x)}$ denotes the restriction of u to the segment $]\leftarrow, x[$. Recall that $]\leftarrow, x[$ is the set of all elements y such that $y < x$; this is just the interval $[0, x[$.

In Bourbaki, the operator $\mathrm{T}\{\!\{u\}\!\}$ is *defined* for any set u. In the proof, it is *used* only when u is of the form $u_n = f^{(n)}$, this is the restriction of some unknown function to a segment of our well-ordered set. If we denote by $\mathrm{M}(u)$ the cardinal of the source of u, then $\mathrm{M}(u_n) = n$, and $\mathrm{M}(u_{n+1}) - 1 = n$. By definition of u_n, we have $u_{n+1}(y) = f(y)$ for $y \leq n$, thus $f(n) = u_{n+1}(n) = u_{n+1}(\mathrm{M}(u_{n+1}) - 1)$. Write this as $f(n) = \mathrm{R}\{\!\{u_{n+1}\}\!\}$.

In the case of general transfinite induction on a set E, we have to change the definition of M, since cardinals are unlikely to be elements of E; we let $\mathrm{M}'(u)$ be the greatest element of the source of u. If we denote by $n+1$ the successor of n (this is the least element of the complementary of $]\leftarrow, n]$, it exists unless n is the greatest element of E), then the source of u_{n+1} is $]\leftarrow, n+1[$, and has n as greatest element; thus $\mathrm{M}'(u_{n+1}) = n$ and we still have $f(n) = \mathrm{R}\{\!\{u_{n+1}\}\!\}$ for some R.

If our term T satisfies $\mathrm{T}\{\!\{u\}\!\} = s(\mathrm{R}\{\!\{u\}\!\})$ whenever u is of the form u_{n+1}, then $f(n+1) = s(f(n))$ for all n. The function f is well-defined if we specify the values $f(m)$ when m is not a successor. In the case of general induction, there can be many such values. In the case of integers, there is only one, and it suffices to specify the value $f(0)$.

Consider the following definitions

```
M := fun u => cardinal (source u)
T := fun u => Yo (M u = card_zero) a (s (W (prec (M u)) u))
T':= fun u => Yo (M u = card_zero) a (h (prec (M u)) (W (prec (M u)) u))
```

Remember that *Yo a b c* evaluates to *b* if *a* is true and to *c* otherwise. The previous discusssion shows that for all a and s, there exists a unique surjective function f defined on **N** such that

(IND) $$f(0) = a \quad \text{and} \quad f(n+1) = s(f(n)).$$

It is easy to show by induction on n, that if E is any set, $a \in \mathrm{E}$ and $s(\mathrm{E}) \subset \mathrm{E}$, then $f(n) \in \mathrm{E}$ for all n.

[1] This section has been completely rewritten in Version 2

If we use T′ instead of T, we see that there exists a unique surjective function f defined on **N** such that

(IND0) $$f(0) = a \quad \text{and} \quad f(n+1) = h(n, f(n)).$$

As previously, if $a \in \mathrm{E}$ and if $h(n,x) \in \mathrm{E}$ whenever $n \in \mathbf{N}$ and $x \in \mathrm{E}$, then $f(n) \in \mathrm{E}$ for all n. It is trivial to deduce (IND) from (IND0) (consider the function $h(n,x) = s(x)$).

Bourbaki first shows (IND), then deduces a variant of (IND0) as follows. He considers a function $h : \mathbf{N} \times \mathrm{E} \to \mathrm{E}$, where E is some fixed set. Denote by F the set $\mathbf{N} \times \mathrm{E}$. Consider the function $\psi : \mathrm{F} \to \mathrm{F}$ defined by $y \mapsto (\mathrm{pr}_1 y + 1, h(y))$ and the surjective function g with values in F defined by induction as $g(0) = (0, a)$ and $g(n+1) = \psi(g(n))$. Define $a_n = \mathrm{pr}_1(g(n))$ and $b_n = \mathrm{pr}_2(g(n))$. From $g(n) \in \mathrm{F}$, one deduces $a_n \in \mathbf{N}$, $b_n \in \mathrm{E}$ and $g(n) = (a_n, b_n)$, hence the two recurrence relations $a_{n+1} = a_n + 1$ and $b_{n+1} = h(a_n, b_n)$. Obviously $a_n = n$, thus $b_{n+1} = h(n, b_n)$, and the function associated to b_n satisfies (IND0).

Consider the following example. Given a sequence of cardinals (x_i), we consider $h(n,y) = x_{n+1} + y$ or $h(n,y) = x_{n+1}^y$. Using (IND0), there exist two sequences satisfying $y_0 = x_0$ and $y_{n+1} = x_{n+1} + y_n$ or $z_0 = x_0$ and $z_{n+1} = x_{n+1}^{z_n}$. Let's try to apply the Bourbaki method. There is a set E stable by cardinal sum that contains the range E_0 of the sequence (x_i). If E_0 is a subset of **N**, just take **N**, otherwise let $\mathfrak{a}$ be the supremum of E_0 (see page 63). This is an infinite cardinal, and we shall see in the next section that the set of all cardinals $\leq \mathfrak{a}$ is stable by cardinal sum. As a consequence, we can use the Bourbaki construction and $y_n \in \mathrm{E}$. Doing the same for z_n is tricky. For any set E we define $p(\mathrm{E})$ to be the set of all x^y for $x \in \mathrm{E}$ and $y \in \mathrm{E}$, and for any cardinal $\mathfrak{a}$, we define $\mathrm{E}_\mathfrak{a}$ to be the set of all cardinals $\leq \mathfrak{a}$ and $s(\mathfrak{a})$ to be the supremum of $p(\mathrm{E}_\mathfrak{a})$. Let f be the function defined by induction via s (where $f(0)$ is any cardinal such that $\mathrm{E}_{f(0)}$ contains the range of the sequence x_i). Finally, let E be the union of the sets $\mathrm{E}_{f(i)}$. Since f is increasing, if x and y are two elements of E, there is an i such that $x \in \mathrm{E}_{f(i)}$ and $y \in \mathrm{E}_{f(i)}$ hence $x^y \in \mathrm{E}_{f(i+1)} \subset \mathrm{E}$. This is a very large set (we could reduce a bit its size by defining $p(\mathrm{E})$ to be the set of all x^y for $x \in \mathrm{E}_0$ and $y \in \mathrm{E}$).

In the first version of the Software we had

```
M := fun u => supremum Bnat_order (source u)
T:= fun u => Yo (u = empty_function) a (s (W (M u) u))
```

We recognise here the quantity M′; it is defined whenever u has nonempty source, which is equivalent to say that u is not the empty function.

The Bourbaki definition is much more complicated. He starts with

$$\mathrm{D}(u) = \mathscr{E}_x(x \in \mathbf{N} \text{ and } (\exists y)((x,y) \in \mathrm{pr}_1(\mathrm{pr}_1(u)))).$$

He says that, if u is a function defined on a subset of **N**, then $\mathrm{D}(u)$ is the domain. In fact, $\mathrm{D}(u)$ is by definition the intersection of the domain of the graph of u and **N**. If u is a function, $\mathrm{D}(u)$ is the intersection of its source and **N**. As mentioned above, in the proof we need only to consider the case where the source is a subset of **N**, and $\mathrm{D}(u)$ could be replaced by the source of u. Bourbaki defines $\mathrm{M}(u)$ as being the least upper bound of $\mathrm{D}(u)$, and in a footnote says that one could change the definition of the least upper bound, in order to give a meaning to this term even when $\mathrm{D}(u)$ is unbounded. Another idea would be to consider $\mathrm{Card}(\mathrm{D}(u)) - 1$; a possible definition of $x-1$ could be x when $x = 0$ or is an infinite cardinal, the usual definition otherwise. Let ϕ be the empty function, and consider the relation

$$\mathrm{R}\{y, u\} : \quad (u = \phi \text{ and } y = a) \text{ or } (u \neq \phi \text{ and } y = \mathrm{S}\{u(\mathrm{M}(u))\}).$$

Let T$\{u\}$ be the term $\tau_y(R\{y,u\})$. If $u=\phi$ then T$\{u\}$ is a, otherwise it is S$\{u(M(u))\}$; our equivalent of the if-then-else construct is *Yo*. The argument u of the term T will always be (this is obvious by induction) the restriction of f to the interval $[0,n-1[$. If $n=0$, the restriction is ϕ hence $f(0)=a$, and if $n=m+1$, the restriction has source $[0,m]$, whose supremum is m, so that $f(m+1)=S\{f(m)\}$.

We give here six definitions[2]. In the the first cases, the function f is assumed to be surjective, and in the other cases, the target will be a given set E. The function will satisfy one of (IND) or (IND0) or

(IND1') $\qquad f(0)=a \quad \text{and} \quad f(n+1)=g(n,f(n)) \quad \text{if} \quad n<m.$

We call this partial induction. In order to get uniqueness, we either have to restrict the source of f to the interval $[0,m]$ or specify a value $f(n)$ for $n>m$. We consider

(IND1) $\qquad f(0)=a \quad \text{and} \quad f(n+1)=g(n,f(n)) \quad \text{if} \quad n<m \quad \text{and} \quad f(n)=a \quad \text{otherwise.}$

```
Definition induction_defined s a:= choose(fun f=>
  source f = Bnat &  surjective f & W card_zero f = a &
  forall n, inc n Bnat -> W (succ n) f = s (W n f)).

(*
Definition induction_defined0 h a := choose(fun f=>
  source f = Bnat & surjective f & W card_zero f = a &
  forall n,  inc n Bnat -> W (succ n) f = h n (W n f)).
*)
Definition induction_defined1 h a p := choose(fun f=>
  source f = Bnat & surjective f & W card_zero f = a &
  (forall n,  cardinal_lt n p -> W (succ n) f = h n (W n f)) &
  (forall n, inc n Bnat -> ~ (cardinal_le n p) -> W n f = a)).

Definition induction_defined_set s a E:= choose(fun f=>
  is_function f &source f = Bnat & target f = E & W card_zero f = a &
  forall n, inc n Bnat -> W (succ n) f = s (W n f)).

Definition induction_defined0_set h a E:= choose(fun f=>
  is_function f & source f = Bnat & target f = E & W card_zero f = a &
  forall n, inc n Bnat -> W (succ n) f = h n (W n f)).

Definition induction_defined1_set h a p E := choose(fun f=>
  is_function f &source f = Bnat & target f = E & W card_zero f = a &
  (forall n,  cardinal_lt n p -> W (succ n) f = h n (W n f)) &
  (forall n, inc n Bnat -> ~ (cardinal_le n p) -> W n f = a)).
```

We state some theorems that say that such a function exists and is unique, and we use the axiom of choice in order to show that the chosen function satisfies the property.

```
(*
  Lemma integer_induction0: forall h a,
    exists_unique (fun f=> source f = Bnat & surjective f &
      W card_zero f = a
      &  forall n, inc n Bnat -> W (succ n) f = h n (W n f)).
  Lemma integer_induction: forall s a, exists_unique (fun f =>
    source f = Bnat &  surjective f & W card_zero f = a &
```

[2] One defintion, and two properties have been given in the previous chapter; they are commented out.

```
    forall n, inc n Bnat -> W (succ n) f = s (W n f)).
  Lemma induction_defined_pr0: forall h a,
    let f :=  induction_defined0 h a in
       source f = Bnat &  surjective f & W card_zero f = a &
       forall n, inc n Bnat -> W (succ n) f = h n (W n f).
*)
Lemma integer_induction1: forall h a p, inc p Bnat ->
  exists_unique (fun f=>  source f = Bnat & surjective f &
    W card_zero f = a &
     (forall n,  cardinal_lt n p -> W (succ n) f = h n (W n f))&
     (forall n, inc n Bnat -> ~ (cardinal_le n p) -> W n f = a)).

Lemma induction_defined_pr: forall s a,
  let f :=  induction_defined s a in
     source f = Bnat &  surjective f & W card_zero f = a &
     forall n, inc n Bnat -> W (succ n) f = s (W n f).
Lemma induction_defined_pr1: forall h a p,
  let f :=  induction_defined1 h a p in
    inc p Bnat ->
    ( source f = Bnat & surjective f &
      W card_zero f = a &
      (forall n,  cardinal_lt n p -> W (succ n) f = h n (W n f))&
      (forall n, inc n Bnat -> ~ (cardinal_le n p) -> W n f = a)).
```

We now show that the target of the function defined by induction is a subset of E under some conditions. It follows that there is a variant where the target is E. We shall not prove uniqueness, it is obvious.

```
Lemma integer_induction_stable: forall E g  a,
  inc a E -> (forall x, inc x E -> inc (g x) E) ->
  sub (target (induction_defined g a)) E.
Lemma integer_induction_stable0: forall E h a,
  inc a E -> (forall n x, inc x E -> inc n Bnat -> inc (h n x) E) ->
  sub (target (induction_defined0 h a)) E.
Lemma integer_induction_stable1: forall E h a p,
  inc p Bnat ->
  inc a E -> (forall n x, inc x E -> cardinal_lt n p -> inc (h n x) E) ->
  sub (target (induction_defined1 h a p)) E.

Lemma induction_defined_pr_set: forall E g a,
  let f := induction_defined_set g a E in
    inc a E -> (forall x, inc x E -> inc (g x) E) ->
     (is_function f & source f = Bnat & target f = E & W card_zero f = a &
     forall n, inc n Bnat -> W (succ n) f = g (W n f)).
Lemma induction_defined_pr_set0: forall E h a,
  let f := induction_defined0_set h a E in
    inc a E -> (forall n x, inc x E -> inc n Bnat -> inc (h n x) E) ->
     (is_function f & source f = Bnat & target f = E & W card_zero f = a &
     forall n, inc n Bnat -> W (succ n) f = h n (W n f)).
Lemma induction_defined_pr_set1: forall E h a p,
  let f := induction_defined1_set h a p E in
    inc p Bnat ->
    inc a E -> (forall n x, inc x E -> cardinal_lt n p -> inc (h n x) E) ->
    (is_function f & source f = Bnat & target f = E & W card_zero f = a &
      (forall n,  cardinal_lt n p -> W (succ n) f = h n (W n f))&
      (forall n, inc n Bnat -> ~ (cardinal_le n p) -> W n f = a)).
```

7.3 Properties of infinite cardinals

Bourbaki claims (Lemma 1, [3, p. 186]) that every infinite set E has a subset F equipotent to **N**; the argument being that there exists a well-ordering on E. If E is isomorphic to a segment of **N**, then E is equipotent to **N** since all other segments are of the form $]\leftarrow, x[$, hence $[0, x[$, thus are finite. Otherwise, **N** is isomorphic to a segment of E, hence equipotent to a subset of E.

```
Lemma equipotent_range: forall f, injective f ->
  equipotent (source f)  (range (graph f)).
Lemma morphism_range: forall f a b,
  order_morphism f a b -> equipotent (substrate a) (range (graph f)).
Lemma morphism_range1: forall f a b,
  order_morphism f a b -> cardinal (substrate a) = cardinal (range (graph f)).
Lemma infinite_greater_countable: forall E,
  infinite_set E -> exists F, sub F E & cardinal F = cardinal Bnat.
```

Bourbaki claims that $\mathbf{N}\times\mathbf{N}$ is equipotent to **N**: the relation $\mathrm{Card}(\mathbf{N}) \le \mathrm{Card}(\mathbf{N}\times\mathbf{N})$ is a consequence of $\{0\}\times\mathbf{N} \subset \mathbf{N}\times\mathbf{N}$; moreover there is an injection from $\mathbf{N}\times\mathbf{N}$ into **N**. He uses expansion to base 2. Assume $x = \sum x_i 2^i$ and $y = \sum y_i 2^i$, then $\sum(2x_i + y_i)4^i$ is an injective function of x and y. We use here a different function: consider

$$f(n,m) = n + \binom{n+m+1}{2} = n + g(n+m).$$

The function g is the binomial coefficient with indices $a+1$ and 2, it is also $g(a) = a(a+1)/2$; it satisfies $g(a+1) = g(a) + a + 1$, hence $g(n+m) \le f(n,m) < g(n+m+1)$. This relation shows that $n+m$ is uniquely defined by $f(n,m)$, from which injectivity of f follows. Consider x and the least a such that $x < g(a)$. Then $x = f(n,m)$, where n and m are the unique integers satisfying $n+m+1 = a$ and $x = n + g(a-1)$. This shows that f is bijective.

```
Lemma equipotent_N2_N: equipotent (product Bnat Bnat) Bnat. (* 87 *)
```

We show here Theorem 2 ([3, p. 186]): for every infinite cardinal $\mathfrak{a}$, we have $\mathfrak{a} = \mathfrak{a}^2$. The proof is by induction (in reality, Zorn's lemma, since there is no induction for infinite cardinals).

We consider an infinite set E, and study bijections of the form $\psi : A \to A\times A$, where $A \subset E$. This is the same as studying the set $\mathfrak{M}$ of functions defined on a subset A of E that are injective, whose target is $E\times E$, and whose range is $A\times A$. Bourbaki says "It is immediately seen that $\mathfrak{M}$ is inductive". (The example in section 3.4 is the set of functions without these conditions). The proof is similar to the one that states that there exists an isomorphism between well-ordered sets. The proof is tedious but straightforward (50 lines of proof).

Since E is infinite there exists a subset D equipotent to **N**, hence (previous theorem) a bijection between D and $D\times D$. This gives us a element ψ_0 of $\mathfrak{M}$. Let $\mathfrak{M}_0$ be the set of elements of $\mathfrak{M}$ that extend ψ_0. This set is inductive as well. We consider a maximal element with source F and cardinal $\mathfrak{b}$. We have $\mathrm{Card}(F) \ge \mathrm{Card}(D)$, so that $\mathfrak{b}$ is infinite. We have $\mathfrak{b} = \mathfrak{b}^2$. If $\mathfrak{c} \le \mathfrak{b}$ then $\mathfrak{b} \le \mathfrak{c} + \mathfrak{b} \le 2\mathfrak{b} \le \mathfrak{b}^2 = \mathfrak{b}$, hence $\mathfrak{c} + \mathfrak{b} = \mathfrak{b}$. In particular $\mathfrak{b} = 2\mathfrak{b} = 3\mathfrak{b}$.

Our theorem is true if $\mathfrak{b} = \mathrm{Card}(E)$. Assume otherwise $\mathfrak{b} < \mathfrak{a}$. The cardinal $\mathfrak{c}$ of the complementary of F in E is $\ge \mathfrak{b}$ (a consequence of the theorem will be $\mathfrak{c} = \mathfrak{a}$). Hence, there exists a subset Y of E disjoint from F with cardinal $\mathfrak{b}$. Let $Z = Y \cup F$. The relation $\mathfrak{b} = 3\mathfrak{b}^2$ implies that Y is equipotent to $(Z\times Z) - (F\times F)$ hence Z is equipotent to $Z\times Z$. This contradicts maximality. .

```
Theorem equipotent_inf2_inf: forall a, is_infinite_c a ->
  card_pow a (card_two) = a. (* 227 *)
```

By induction, $\mathfrak{a}^n = \mathfrak{a}$ if $\mathfrak{a}$ is an infinite cardinal and $n \geq 1$ is an integer. As a consequence, if $(\mathfrak{a}_i)_{i\in \mathrm{I}}$ is a finite family of non-zero cardinals, if the largest one is an infinite cardinal $\mathfrak{a}$, then the product is $\mathfrak{a}$. If $\mathfrak{a}_i \leq \mathfrak{a}$ then $\sum \mathfrak{a}_i \leq \mathfrak{a}$ and we have equality if one of the cardinals is $\mathfrak{a}$. The cardinals can be zero, and the index set can be infinite, provided that Card(I) $\leq a$. Finally, if $\mathfrak{a}$ and $\mathfrak{b}$ are two non-zero cardinals, one of them being infinite, then the sum and the product is the greatest of them.

Note. Bourbaki writes "sup$(\mathfrak{a}, \mathfrak{b})$" instead of "the greatest of them"; our function *sup* takes three arguments, one of them being the order; in this case, there is no order (because there is no set containing all cardinals). We know that the supremum of any family of cardinals exists, so that the supremum of two cardinals is well-defined, but we have no notation for this operation. Note also that using a lemma of the form: if $\mathfrak{a}$ is infinite or $\mathfrak{b}$ is infinite, then $\mathfrak{a}+\mathfrak{b} = \sup(\mathfrak{a}, \mathfrak{b})$ is uneasy, since this means that $\mathfrak{a}+\mathfrak{b}$ is at least $\mathfrak{a}$, at least $\mathfrak{b}$, and at most any $\mathfrak{c}$ that is at least $\mathfrak{a}$ and at least $\mathfrak{b}$. It is much easier to say: if $\mathfrak{a}$ is infinite and $\mathfrak{b} \leq \mathfrak{a}$ then $\mathfrak{a}+\mathfrak{b} = \mathfrak{a}$, and use commutativity of addition when needed. Note that $\mathfrak{b}$ infinite implies $\mathfrak{a}$ infinite, and $\mathfrak{b}$ can be zero (since $\mathfrak{a}$ is infinite, it is clearly non-zero). In the case of a product, we need the condition $\mathfrak{b} \neq 0$.

```
Lemma square_of_infinite: forall a, is_infinite_c a ->
  card_mult a a = a.
Lemma power_of_infinite: forall a n, is_infinite_c a -> inc n Bnat ->
  n <> card_zero -> card_pow a n = a.
Lemma power_of_infinite1: forall a n, is_infinite_c a -> inc n Bnat ->
  cardinal_le (card_pow a n)  a.
Lemma finite_family_product: forall a f, fgraph f ->
  is_finite_set (domain f) -> is_infinite_c a ->
  (forall i, inc i (domain f) -> cardinal_le (V i f) a) ->
  (forall i, inc i (domain f) -> V i f <> card_zero) ->
  (exists j,  inc j (domain f) & (V j f) = a) ->
  cardinal_prod f = a.
Lemma product2_infinite: forall a b, cardinal_le b a ->
  is_infinite_c a -> b <> card_zero -> card_mult a b = a.
Lemma product2_infinite1:  forall a b, cardinal_le b a ->
  is_infinite_c a -> cardinal_le (card_mult a b) a.
Lemma product2_infinite2: forall a b, is_finite_c b ->
  is_infinite_c a -> cardinal_le (card_mult a b) a.
Lemma notbig_family_sum: forall a f, fgraph f ->
  is_infinite_c a -> cardinal_le (cardinal (domain f)) a ->
  (forall i, inc i (domain f) -> cardinal_le (V i f) a) ->
  cardinal_le (cardinal_sum f) a.
Lemma notbig_family_sum1: forall a f, fgraph f ->
  is_infinite_c a -> cardinal_le (cardinal (domain f)) a ->
  (forall i, inc i (domain f) -> cardinal_le (V i f) a) ->
  (exists j,  inc j (domain f) & (V j f) = a) ->
  (cardinal_sum f) = a.
Lemma sum2_infinite1: forall a, is_infinite_c a -> card_plus a a = a.
Lemma sum2_infinite: forall a b, cardinal_le b a ->
  is_infinite_c a -> card_plus a b = a.
```

7.4 Countable sets

A countable set is one that is equipotent to a subset of **N**. Proposition 2 [3, p. 188] says that an infinite countable set is equipotent to **N**. We rewrite this as: a countable set is finite or equipotent to **N**.

Proposition 1 [3, p. 188] says that a subset of a countable set is countable; the product of a finite family of countable sets is countable; the union of a countable family of countable sets is countable.

Proposition 3 [3, p. 189] says that an infinite set E has a partition $(X_\iota)_{\iota\in I}$ where X_ι is infinite countable and I is equipotent to E. Proposition 4 [3, p. 189] says that if f is a function from E onto F, such that F is infinite and $\overset{-1}{f}\langle\{x\}\rangle$ is countable for any $x \in$ F, then F is equipotent to E.

```
Definition is_countable_set E:= equipotent_to_subset E Bnat.

Lemma cardinal_comp_singl_inf: forall E x,
  infinite_set E -> cardinal E = cardinal (complement E (singleton x)).
Lemma countable_prop: forall E,
  is_countable_set E  =  cardinal_le (cardinal E) (cardinal Bnat).
Lemma infinite_greater_countable1: forall E,
  infinite_set E ->  cardinal_le (cardinal Bnat) (cardinal E).
Lemma countable_finite_or_N: forall E, is_countable_set E ->
  is_finite_c (cardinal E) \/ cardinal E = cardinal Bnat.
Lemma countable_finite_or_N_b: forall E, is_countable_set E ->
  is_finite_set E \/ equipotent E Bnat.
Lemma countable_finite_or_N_c: forall E, is_countable_set E ->
  infinite_set E -> equipotent E Bnat.
Theorem countable_subset: forall E F, sub E F -> is_countable_set F  ->
  is_countable_set E.
Theorem countable_product: forall f,  fgraph f ->
  is_finite_set (domain f) ->
  (forall i, inc i (domain f) -> is_countable_set (V i f)) ->
  is_countable_set (productb f).
Theorem countable_union: forall f,  fgraph f ->
  is_countable_set (domain f) ->
  (forall i, inc i (domain f) -> is_countable_set (V i f)) ->
  is_countable_set (unionb f).
Lemma card_bnat_not_zero: cardinal Bnat <> card_zero.
Theorem infinite_partition: forall E, infinite_set E ->
  exists f, partition_fam f E & equipotent (domain f) E &
    (forall i, inc i (domain f) -> (infinite_set (V i f) &
      is_countable_set (V i f))). (* 41 *)
Theorem countable_inv_image: forall f, surjective f ->
  (forall y, inc y (target f) ->
    is_countable_set (inv_image_by_fun f (singleton y))) ->
  infinite_set (target f) ->
  equipotent (source f) (target f). (* 44 *)
```

Proposition 5 [3, p. 189] says that the set $\mathfrak{F}$ of finite subsets of an infinite set E is equipotent to E. The proof of Bourbaki is not clear. He defines $\mathfrak{F}_n$ as the set of all subsets with n elements of E and claims $\mathrm{Card}(\mathfrak{F}_n) \leq \mathrm{Card}(E)$. Thus, the cardinal of the union of these sets is at most $\sum_{n\in\mathbf{N}} \mathrm{Card}(E) = \mathrm{Card}(E)$. Thus $\mathrm{Card}(\mathfrak{F}) \leq \mathrm{Card}(E)$; equality holds because the set of singletons is equipotent to E and is a subset of $\mathfrak{F}$.

The Bourbaki claim is: for every X ∈ $\mathfrak{F}_n$ there is a bijection from $[1,n]$ onto X, so that the cardinal of $\mathfrak{F}_n$ is at most the cardinal of the set of functions from $[1,n]$ into X which is Card(E^n) = Card(E). Our proof is as follows.

For every X ∈ $\mathfrak{F}_n$ there is a bijection $[1,n] \to X$, hence an injective function from $[1,n]$ into E with range X, but it is not unique. Let K be the set of injections from $[1,n]$ into E. Let f be the function that associates to each element of K its range. The target of this function is clearly $\mathfrak{F}_n$. Let Q be the set of permutations of $[1,n]$, and c it cardinal. This is a non-zero integer. We pretend that the cardinal of $f^{-1}\langle\{x\}\rangle$ is c. We take an element g in this set (it exists, by the remark above). For every permutation h of $[1,n]$, we consider $g \circ h$. This operation is a bijection from Q onto $f^{-1}\langle\{x\}\rangle$. Surjectivity of this operation uses the fact that for any k there exists g such that $k = g \circ h$, if the ranges are the same (since h is injective) and this function is surjective. It is bijective since it is an endomorphism of a finite set. We can now apply the shepherd's principle. The product of the cardinal a of $\mathfrak{F}_n$ and c is the cardinal b of the set of injections, that is smaller than the cardinal d of the set of functions from $[1,n]$ into E. If $n = 0$, we clearly have $a \leq$ Card(E); otherwise $d =$ Card(E). Hence $ac = b \leq$ Card(E). If a is finite, we have $a \leq$ Card(E); but if a is infinite, we have $a = ac$ (since c is non-zero finite). This implies $a \leq$ Card(E).

As a corollary, the set of finite sequences with value into E is equipotent to E; in fact, this set is the union of the sets of functions from I into E (that has the same cardinal as E^I) for all finite subsets I of **N**. Since E^I and I are equipotent and since the set of finite subsets of **N** is countable, the result is immediate.

```
Theorem infinite_finite_subsets: forall E, infinite_set E ->
  equipotent (Zo (powerset E) (fun z => is_finite_set z)) E. (* 177 *)

Lemma infinite_finite_sequence: forall E, infinite_set E ->
 equipotent (Zo(set_of_sub_functions Bnat E) (fun z=> is_finite_set (source z)))
 E. (* 48 *)
```

A set is said to have *the power of the continuum* if it is equipotent to $\mathfrak{P}(\mathbf{N})$. In this case, its cardinal is $2^{\aleph_0}$, and the set is not countable.

7.5 Stationary sequences

A sequence $(x_n)_{n\in\mathbf{N}}$ is *stationary* if there exists an integer m such that $x_n = x_m$ for $n \geq m$. We define here the notion of increasing and decreasing sequences. It is the graph of an increasing function where the source is **N** with its natural order. Note that a decreasing sequence is increasing for the opposite order.

```
Definition stationary_sequence f :=
  fgraph f & domain f = Bnat &
  exists m,  inc m Bnat & forall n, inc n Bnat -> cardinal_le m n ->
    V n f = V m f.
Definition increasing_sequence f r:=
  fgraph f & domain f = Bnat & sub (range f) (substrate r) &
  forall n m, inc n Bnat  -> inc m Bnat -> cardinal_le n m ->
    gle r (V n f) (V m f).

Definition decreasing_sequence f r:=
  fgraph f & domain f = Bnat & sub (range f) (substrate r) &
  forall n m, inc n Bnat  -> inc m Bnat -> cardinal_le n m ->
```

```
    gle r  (V m f) (V n f).
```

Proposition 6 [3, p. 190] says that, if E is an ordered set, each non-empty set has a maximal element if and only if each increasing sequence is stationary. We start with a lemma: a function f such that $f(n) \leq f(n+1)$ is increasing (by induction on m, we have $f(n) \leq f(n+m)$). By definition *increasing_fun f r r'* says that the target of f is the substrate of r; in our case, it is merely a subset, so that the definition will not be used: we show that the graph of f is an increasing sequence.

The Proposition is shown as follows. Given an increasing sequence, its range is non-empty. It it has a maximal element x_n and $m \geq n$ then $x_n \leq x_m$ implies $x_m = x_n$. Conversely, assume that we have a set A that has no maximal element. For each x, the subset T_x of elements of A greater than x is non-empty. This means that the product $\prod \mathrm{T}_x$ is non-empty, hence there is a function $f : \mathrm{A} \to \mathrm{A}$ such that $f(x) > x$ and a sequence $x_{n+1} = f(x_n)$. This sequence is strictly increasing, absurd.

As a consequence a totally ordered set E is well-ordered if and only if each decreasing sequence is stationary (to show that it is well-ordered, we consider the opposite order; thus every non-empty set has a minimal element, this element is the least element, since all subsets of E are directed). Moreover, an increasing sequence in a finite ordered set has a maximal element.

```
Lemma increasing_prop: forall f r, order r ->
  is_function f -> source f = Bnat -> sub (target f) (substrate r) ->
  (forall n, inc n Bnat -> gle r (W n f) (W (succ n) f)) ->
  increasing_sequence (graph f) r.
Lemma decreasing_prop: forall f r, order r ->
  is_function f -> source f = Bnat -> sub (target f) (substrate r) ->
  (forall n, inc n Bnat -> gge r (W n f) (W (succ n) f)) ->
  decreasing_sequence (graph f) r.
Theorem  increasing_stationary: forall r, order r ->
  (forall X, sub X (substrate r) -> nonempty X ->
     exists a,  maximal_element (induced_order r X) a) =
  (forall f, increasing_sequence f r -> stationary_sequence f). (* 47 *)
Theorem  decreasing_stationary: forall r, total_order r ->
  (worder r) =
  (forall f, decreasing_sequence f r -> stationary_sequence f). (* 32 *)
Theorem  finite_increasing_stationary: forall r, order r ->
  is_finite_set (substrate r) ->
  (forall f, increasing_sequence f r -> stationary_sequence f).
```

Proposition 7 [3, p. 190] says that if E is noetherian (every non-empty set has maximal element) and if F is a subset of E such that for $a \in \mathrm{E}$, if $\forall x, x > a \implies x \in \mathrm{F}$ then $a \in \mathrm{F}$; then $\mathrm{F} = \mathrm{E}$.

```
Theorem noetherian_induction: forall r F,  order r ->
  (forall X, sub X (substrate r) -> nonempty X ->
    exists a,  maximal_element (induced_order r X) a) ->
  sub F (substrate r) ->
  (forall a, inc a (substrate r) -> (forall x, glt r a x -> inc x F)
    -> inc  a F)
  -> F = substrate r.
```

Chapter 8

Ordinal numbers

What follows is not part of the main text of Bourbaki, but may come from exercises. Ordinal sums are defined in Exercise 1.3, and ordinal numbers in Exercise 2.13.

8.1 Order sums and products

In Exercise 2.13, Bourbaki says "The order-type of the ordinal sum of the family of sets [...] is called the *ordinal sum* of the order-types [...] and is denoted by $\sum_{\iota\in I}\lambda_\iota$. The order-type of the lexicographic product of the family of sets [...] is called the *ordinal product* of the order-types [...] and is denoted by $\mathsf{P}_{\iota\in I}\lambda_\iota$.".

The first definition makes the term "ordinal sum" ambiguous. For this reason, we shall define two quantities the order-sum and the ordinal sum. In the same fashion, we shall define the order-product and the ordinal product. The "order-product" is nothing else than the lexicographic order defined in section 3.6 but has no relation with the product defined in section 2.4. Both definitions make the notation Σ and P ambiguous, but this is inevitable. In what follows, we shall introduce more confusion by using Π instead of P. Thus Π may denote: the product of a family of sets, the cardinal product, the order-product, and the ordinal product. In the same way, σ may denote: the disjoint union of a family of sets, the cardinal sum, the order-sum, and the ordinal sum. We shall overload the notations $a+\mathrm{B}$ and $a\cdot b$ in the obvious way.

Consider a family of sets $(\mathrm{X}_\iota)_{\iota\in\mathrm{I}}$. The product $\prod_{\iota\in\mathrm{I}}\mathrm{X}_\iota$ is the set of all families $x=(x_\iota)_{\iota\in\mathrm{I}}$, where $x_\iota\in\mathrm{X}_\iota$ for any $\iota\in\mathrm{I}$, while the sum $\sum_{\iota\in\mathrm{I}}\mathrm{X}_\iota$ is the disjoint union, containing all x which are pairs (a,b) with $b\in\mathrm{I}$ and $a\in\mathrm{E}_b$.

Assume that we have an ordering $\leq$ on I and an ordering $\leq_\iota$ on each X_ι. To these orderings $\leq$ and $\leq_\iota$ we shall associate two orderings, the order-product (defined in an earlier chapter) whose substrate is $\prod_{\iota\in\mathrm{I}}\mathrm{X}_\iota$, and the order-sum, whose substrate is $\sum_{\iota\in\mathrm{I}}\mathrm{X}_\iota$. These orderings will be, by abuse of notations, denoted like the substrate, but depend only on $\leq$ and $\leq_\iota$.

The product is defined so that $x\leq y$ if either $x=y$ or for the least λ such that $x_\lambda\neq y_\lambda$ we have $x_\lambda<y_\lambda$. The index set I is assumed well-ordered, so that such a λ always exists.

We restate $x\leq y$ in the product as "either $x=y$, or there is an index j such that $x_j<y_j$, and whenever $i<j$ we have $x_i=y_i$".

```
Definition order_product := lexicographic_order.
Definition order_product_a := lexicographic_order_axioms.
```

```
Lemma order_product_order: forall r g,
  order_product_a r g -> order (order_product r g).
Lemma order_product_substrate: forall r g,
  order_product_a r g ->
  substrate(order_product r g) = prod_of_substrates g.

Lemma prod_of_substrates_pr: forall i z g,
  inc i (domain g) -> inc z (prod_of_substrates g) ->
  inc (V i z) (substrate (V i g)).
Lemma order_product_gle: forall r g x x', order_product_a r g ->
  gle (order_product r g) x x' =
  (inc x (prod_of_substrates g) & inc x' (prod_of_substrates g) &
    (x= x' \/ exists j, inc j (substrate r)  &
      glt (V j g) (V j x) (V j x') &
      forall i, glt r i j -> V i x = V i x')).
```

The ordering of the sum is defined by $x \leq y$ when Consider the relation "either $\mathrm{pr}_2 x < \mathrm{pr}_2 y$, or $\mathrm{pr}_2 x = \mathrm{pr}_2 y = \lambda$ and then $\mathrm{pr}_1 x \leq_\lambda \mathrm{pr}_1 y$". This makes sense, since $\mathrm{pr}_2 x$ and $\mathrm{pr}_2 y$ are in the substrate of $\leq$, while, if $\mathrm{pr}_2 x = \mathrm{pr}_2 y = \lambda$, then $\mathrm{pr}_1 x$ and $\mathrm{pr}_1 y$ are in the substrate of $\leq_\lambda$. In Exercise 1.3, Bourbaki assumes that the sets X_ι are non-empty, but this makes $x+0$ undefined, so that we shall ignore this condition.

```
Definition order_sum_a r g:=
  order r & substrate r = domain g & fgraph g &
  (forall i, inc i (domain g) -> order (V i g)).

Definition order_sum_axioms1 r g:=
  order_sum_a r g &
  (forall i, inc i (domain g) -> nonempty (substrate (V i g))).

Definition order_sum_r r g x x' :=
    (glt r (Q x) (Q x') \/ (Q x = Q x' & gle (V (Q x) g) (P x) (P x'))).
Definition order_sum r g :=
  graph_on (order_sum_r r g)(disjoint_union (fam_of_substrates g)).
```

We consider some lemmas that explain when $x \in \sum_{\iota\in I} \mathrm{X}_\iota$ given the family $(\leq_\iota)$.

```
Definition sum_of_substrates g := disjoint_union (fam_of_substrates g).
Lemma du_index_pr: forall f x, inc x (disjoint_union f) ->
  (inc (Q x) (domain f) &  inc (P x) (V (Q x) f) & is_pair x).
Lemma du_index_pr1: forall g x, inc x (sum_of_substrates g) ->
  (inc (Q x) (domain g) &  inc (P x) (substrate (V (Q x) g)) & is_pair x).
Lemma inc_disjoint_union: forall f x y,
  inc y (domain f) -> inc x (V y f) ->
  inc (J x y) (disjoint_union f).
Lemma inc_disjoint_union1: forall g x y,
  inc y (domain g) -> inc x (substrate (V y g)) ->
  inc (J x y) (sum_of_substrates g).
```

We show here that this definition induces an order on the disjoint union.

```
Lemma order_sum_order: forall r g,
  order_sum_a r g -> order (order_sum r  g).
Lemma order_sum_gle: forall r g x x', order_sum_a r g ->
```

```
  gle (order_sum r g) x x' =
  (inc x (sum_of_substrates g) & inc x' (sum_of_substrates g) &
    order_sum_r r g x x').
Lemma order_sum_gle1: forall r g x x', order_sum_a r g ->
  gle (order_sum r g) x x' ->
  (glt r (Q x) (Q x') \/ (Q x = Q x' & gle (V (Q x) g) (P x) (P x'))).
Lemma order_sum_gle2: forall r g a b a' b', order_sum_a r g ->
  gle (order_sum r g) (J a b) (J a' b')  ->
  (glt r b b' \/ (b = b' & gle (V b g) a a')).
Lemma order_sum_gle_id: forall r g x x', order_sum_a r g ->
  gle (order_sum r g) x x' ->  gle r (Q x) (Q x').

Lemma order_sum_substrate: forall r g, order_sum_a r g ->
  substrate (order_sum r g) = (sum_of_substrates g).
```

We consider now the case of the sum and product of two sets. This operation is non-commutative, and we shall use our canonical doubleton as ordering. Recall that we have two distinguished elements, *TPa* and *TPb*, let's call then α and β. The canonical doubleton is the set $\{\alpha,\beta\}$, ordered by $\alpha < \beta$. This is a well-ordering. Note the ordering of the product: it is so that $x + x = x \cdot 2$.

```
Definition order_prod2 r r' :=
  lexicographic_order canonical_doubleton_order (variantLc r' r).

Definition order_sum2 r r' :=
  order_sum canonical_doubleton_order (variantLc r r').

Lemma order_sum2_axioms: forall r r', order r -> order r' ->
  order_sum_a canonical_doubleton_order (variantLc r r').
Lemma order_prod2_axioms: forall r r', order r -> order r' ->
  order_product_a canonical_doubleton_order (variantLc r' r).
Lemma order_sum2_order: forall r r', order r -> order r' ->
  order (order_sum2 r r').
Lemma order_prod2_order: forall r r',  order r -> order r' ->
  order (order_prod2 r r').
```

The disjoint union of the family $\alpha \mapsto E_1$ and $\beta \mapsto E_2$ is $E_1 \times \{\alpha\} \cup E_2 \times \{\beta\}$. In the special case where α and β are as above, we call this the canonical disjoint union. This is the substrate of the ordinal sum $E_1 + E_2$. The substrate of the lexicographic product $E_1 \cdot E_2$ is known as *product2*; it is the set of functional graphs x such that $x_\alpha \in E_2$ and $x_\beta \in E_1$, it is set-isomorphic to $E_2 \times E_1$ (note that this is set-isomorphic to $E_1 \times E_2$, by these set-isomorphisms are not used, we shall consider only order-preserving isomorphisms).

```
Definition canonical_du2 a b :=
  disjoint_union (variantLc a b).

Lemma canonical_du2_rw: forall a b,
  canonical_du2 a b = union2 (product a (singleton TPa))
  (product b (singleton TPb)).
Lemma canonical_du2_pr: forall a b x,
  inc x (canonical_du2 a b) = (is_pair x &
  ((inc (P x) a & Q x = TPa) \/ (inc (P x) b & Q x = TPb))).
Lemma canonical2_substrate: forall r r',
  fam_of_substrates (variantLc r r') = Lvariantc (substrate r) (substrate r').
```

```
Lemma order_sum2_substrate: forall  r r',  order r -> order r' ->
  substrate (order_sum2 r r') = canonical_du2 (substrate r) (substrate r').
Lemma order_prod2_substrate: forall r r',  order r -> order r' ->
  substrate (order_prod2 r r') = product2 (substrate r') (substrate r).
```

The ordering on $E_1 + E_2$ is defined by $x \leq_s y$ if and only if either $\mathrm{pr}_2 x = \mathrm{pr}_2 y = \alpha$ and $\mathrm{pr}_1 x \leq_r \mathrm{pr}_1 y$, or $\mathrm{pr}_2 x = \mathrm{pr}_2 y = \beta$ and $\mathrm{pr}_1 x \leq_{r'} \mathrm{pr}_1 y$, or $\mathrm{pr}_2 x <_c \mathrm{pr}_2 y$, where $\leq_r$ is the ordering on E_1, $\leq_{r'}$ the ordering on E_2, $<_c$ is the ordering on the canonical doubleton. We have: $\mathrm{pr}_2 x <_c \mathrm{pr}_2 y$ is the same as $\mathrm{pr}_2 x = \alpha$ and $\mathrm{pr}_2 y \neq \alpha$, while $\mathrm{pr}_2 x = \mathrm{pr}_2 y = \beta$ is the same as $\mathrm{pr}_2 x \neq \alpha$ and $\mathrm{pr}_2 y \neq \alpha$.

In the case of a product $E \cdot I$, we have $x < y$ if either $x_\alpha < y_\alpha$ in I or if $x_\alpha = y_\alpha$, and $x_\beta < y_\beta$ in E.

```
Lemma order_sum2_gle: forall r r' x x',  order r -> order r' ->
  gle (order_sum2 r r') x x' =
  (inc x (canonical_du2 (substrate r) (substrate r')) &
    inc x' (canonical_du2 (substrate r) (substrate r')) &
    ((Q x = TPa & Q x' = TPa & gle r (P x) (P x'))
      \/ (Q x <> TPa & Q x' <> TPa & gle r' (P x) (P x'))
      \/ (Q x = TPa & Q x' <> TPa))).
Lemma order_sum2_gle_spec: forall r r' x x',  order r -> order r' ->
  inc x (substrate r) -> inc x' (substrate r') ->
  glt (order_sum2 r r') (J x TPa) (J x' TPb).
Lemma order_prod2_gle: forall r r' a b,
  order r -> order r' ->
  gle (order_prod2 r r') a b =
  (inc a  (product2 (substrate r') (substrate r))
    & inc b  (product2 (substrate r') (substrate r))
    & ( (V TPa a = V TPa b & gle r (V TPb a) (V TPb b))
      \/ (glt r' (V TPa a)(V TPa b)))).  (* 34 *)
a order_prod2_gle2: forall r r' a b,
  order r -> order r' ->
  gle (order_prod2 r r') a b =
  (inc a (substrate (order_prod2 r r')) &
    inc b (substrate (order_prod2 r r')) &
     ( (V TPa a = V TPa b & gle r (V TPb a) (V TPb b))
      \/ (glt r' (V TPa a)(V TPa b)))).
```

If I and each E_i are well-ordered, so is the ordinal sum. If moreover I is finite, then the product is also well-ordered (for the converse, see Exercises 2.9 and 2.11). If I and each E_i are totally ordered, so is the ordinal sum; the converse is true (provided that E_ι is non-empty). The ordinal sum or product of two well-ordered sets are well-ordered.

```
Lemma order_sum_worder : forall r g,
  order_sum_a r  g -> worder r ->
  (forall i, inc i (domain g) -> worder (V i g))
  ->  worder (order_sum r g).  (* 25 *)
Lemma order_product_worder: forall r g,
  order_product_a r g ->
  (forall i, inc i (domain g) -> worder (V i g)) ->
  is_finite_set (substrate r) ->
  worder (lexicographic_order r  g). (* 138 *)

Lemma order_prod2_worder: forall r r',
  worder r -> worder r' ->  worder (order_prod2 r r').
```

```
Lemma order_sum_totalorder: forall r g,
  order_sum_axioms1 r g ->
  total_order (order_sum r g) = (total_order r &
    forall i, inc i (domain g) -> total_order (V i g)).
Lemma order_sum2_totalorder: forall r r',
  total_order r -> total_order r' -> total_order (order_sum2 r r').
Lemma order_sum2_worder: forall r r',
  worder r -> worder r' -> worder (order_sum2 r r').
```

8.2 Order types

In Exercise 2.13, Bourbaki defines Is(Γ,Γ') as the relation "Γ is an ordering (on E) and Γ' is an ordering (on E$'$), and there exists an isomorphism of E, ordered by Γ, onto E$'$, ordered by Γ'". The definition given here is simpler.

```
Definition order_isomorphic r r':=
  exists f, order_isomorphism f r r'.
```

If f is an order isomorphism, then $a < b$ is the same as $f(a) < f(b)$. Note the subtlety: if $a < b$, then a and b are in the source of f. Converse is false: if $f(a) < f(b)$, then $f(a)$ is in the target of f, but this does not imply that a is in the source.

```
Lemma order_isomorphism_pr1: forall r r' f a b,
  order_isomorphism f r r' -> inc a (source f) -> inc b (source f) ->
  (glt r a b = glt r' (W a f) (W b f)).
Lemma order_isomorphism_pr: forall r r' f a b,
  order_isomorphism f r r' -> inc a (substrate r) -> inc b (substrate r) ->
  (glt r a b = glt r' (W a f) (W b f)).
Lemma order_isomorphism_pr2: forall r r' f a b,
  order_isomorphism f r r' -> glt r a b ->
  glt r' (W a f) (W b f).
Lemma order_morphism_pr1: forall r r' f a b,
  order_morphism f r r' -> inc a (source f) -> inc b (source f) ->
  (glt r a b = glt r' (W a f) (W b f)).
Lemma order_morphism_pr: forall r r' f a b,
  order_morphism f r r' -> inc a (substrate r) -> inc b (substrate r) ->
  (glt r a b = glt r' (W a f) (W b f)).
Lemma order_morphism_pr2: forall r r' f a b,
  order_morphism f r r' -> glt r a b ->
  glt r' (W a f) (W b f).
```

We know that a strict increasing function A → B is an order morphism if the source is totally ordered. We restate "strict increasing" as "injective and increasing". In particular, "bijective and increasing" implies isomorphism. This will be useful, as we shall consider mostly well-order sets.

```
Lemma total_order_morphism: forall f r r',
  total_order r -> order r' ->
  injective f ->  substrate r = source f -> substrate r' = target f ->
  (forall x y, inc x (source f) -> inc y (source f) ->
    gle r x y -> gle r' (W x f) (W y f)) ->
  order_morphism f r r'.
```

```
Lemma total_order_isomorphism: forall f r r',
  total_order r -> order r' ->
  bijective f ->  substrate r = source f -> substrate r' = target f ->
  (forall x y, inc x (source f) -> inc y (source f) ->
    gle r x y -> gle r' (W x f) (W y f)) ->
  order_isomorphism f r r'.
```

Exercise 2.13(a) says that "Is(Γ,Γ') is an equivalence relation on every set whose elements are orderings". We prefer the three following lemmas

```
Lemma order_isomorphic_reflexive: forall r,
  order r ->  order_isomorphic r r.
Lemma order_isomorphic_symmetric: forall r r',
  order_isomorphic r r' -> order_isomorphic r' r.
Lemma order_isomorphic_transitive: forall r r' r'',
  order_isomorphic r r' -> order_isomorphic r' r'' ->
  order_isomorphic r r''.
```

Exercise 1.10 says that $E\cdot I$ is isomorphic to the sum $\sum_{\iota\in I} E_\iota$ where each E_ι is equal to E. The isomorphism is simply $x\mapsto(x_\beta,x_\alpha)$.

```
Lemma order_prod_pr: forall r r',
  order r -> order r' ->
  order_isomorphic (order_prod2 r r')
    (order_sum r' (cst_graph (substrate r') r)).  (* 35 *)
```

The order-type of Γ, denoted Ord(Γ), is the generic order isomorphic to Γ. An order-type is Ord(Γ) for some Γ. It is called an *ordinal* whenever Γ is well-ordered.

```
Definition order_type x := choose (fun z => order_isomorphic x z).
Definition is_order_type x := exists y, order y & x = order_type y.
Definition is_ordinal x := exists y, worder y & x = order_type y.
```

We give here some trivial lemmas.

```
Lemma ordinal_pr1: forall r, order r ->  order_isomorphic r (order_type r).
Lemma ordinal_pr2: forall r, order r ->  order (order_type r).
Lemma ordinal_pr3: forall r, is_order_type r -> order r.
Lemma ordinal_pr4: forall r r',  order r -> order r' ->
  (order_type r = order_type r') = order_isomorphic r r'.
```

A well-ordered set is mapped by isomorphism to a well-ordered sets; hence an ordinal is a well-ordered set.

```
Lemma worder_invariance: forall r r',
  order_isomorphic r r' -> worder r -> worder r'.
Lemma ordinal_pr5: forall r, is_ordinal r -> worder r.
Lemma ordinal_pr51: forall r, is_ordinal r -> order r.
Lemma ordinal_pr52: forall r, is_ordinal r -> total_order r.

Lemma ordinal_pr6 : forall r, order r -> is_order_type (order_type r).
Lemma ordinal_pr7 : forall r, worder r -> is_ordinal (order_type r).
Lemma ordinal_pr8 : forall y, is_order_type y -> order_type y = y.
Lemma ordinal_pr9 : forall x y, order x -> is_order_type y ->
```

```
  order_isomorphic x y -> order_type x = y.
Lemma ordinal_pr10 : forall x, is_ordinal x -> is_order_type x.
Lemma ordinal_pr11: forall x y, is_ordinal x -> is_ordinal y ->
  (order_isomorphic x y) = (x = y).
```

We write $E \prec E'$ if E and E' are ordered sets and there is an isomorphism from E onto a subset of E'. This can be rephrased as; there is an order morphism of E to E'. This relation is reflexive and transitive. Note that if $E \prec E'$ and $E' \prec E$ then $\mathrm{Card}(E) \leq \mathrm{Card}(E')$ and $\mathrm{Card}(E') \leq \mathrm{Card}(E)$ so that the sets are equipotent. If the sets are well-ordered, they are isomorphic. In the definition that follows, we shall assume that both sets are order-types. We show that if there an isomorphism from E onto a subset of E', where E and E' are ordered sets, then $\mathrm{Ord}(E) \prec \mathrm{Ord}(E')$.

```
Definition order_type_le r r' :=
  is_order_type r & is_order_type r' &
  exists f, exists x,
    sub x (substrate r') & order_isomorphism f r (induced_order r' x).

Lemma order_type_pr0: forall f a b x, order a -> order b ->
  sub x (substrate b) ->
  let F := (BL (fun z => W z f) (substrate a) (substrate b)) in
    order_isomorphism f a (induced_order b x)
    -> (order_morphism F a b  & range (graph F) = x).
Lemma order_type_le_pr2: forall r r',
   order r -> order r' ->
   (exists f, order_morphism f r r')
  ->  (exists f, exists x,  sub x (substrate r') &
       order_isomorphism f r (induced_order r' x)).
Lemma order_type_le_reflexive:
   forall c, is_order_type c ->  order_type_le c c.
Lemma order_type_le_preorder: preorder_r order_type_le.  (* 45 *)
Lemma order_type_le_pr: forall r r',
   order r -> order r' ->
   (exists f, exists x,  sub x (substrate r') &
       order_isomorphism f r (induced_order r' x))
  ->  order_type_le (order_type r) (order_type r').   (* 50 *)
Lemma order_type_le_pr1: forall r r',
  order_type_le r r' ->  exists f, order_morphism f r r'.
Lemma order_type_le_pr3: forall r r',
   is_order_type r -> is_order_type r' ->
   (exists f, order_morphism f r r') ->
   order_type_le r r'.
Lemma order_type_le_pr4: forall r r',
   order r -> order r' ->
   (exists f, order_morphism f r r')
  ->  order_type_le (order_type r) (order_type r').
```

We write $\lambda \leq_{\mathrm{Ord}} \mu$, (in short, $\lambda \leq \mu$) if λ and μ are ordinals and $\lambda \prec \mu$. This means that there is an order isomorphism from λ onto a subset of μ. Let F be this subset; it is isomorphic to a segment G of μ (here F and G both denote a subset of the substrate of μ and the ordering induced by μ on these sets). Thus $\lambda \leq_{\mathrm{Ord}} \mu$ if and only if there is as order morphism $\lambda \to \mu$ whose range is a segment G. An immediate consequence is that $\leq_{\mathrm{Ord}}$ is an ordering. If G is the substrate, the morphism is an isomorphism and $\lambda = \mu$. Otherwise $G =]\leftarrow, x[$ for some x. In this case $\lambda \neq \mu$, since a segment $]\leftarrow, x[$ cannot be isomorphic to the substrate. Thus

we have $\lambda <_{\mathrm{Ord}} \mu$, if and only if there is as order morphism $\lambda \to \mu$ whose range is a segment $]\leftarrow, x[$. Moreover, if $\lambda <_{\mathrm{Ord}} \mathrm{Ord}(\mu)$, where μ is an ordering, then there is as order morphism $\lambda \to \mu$ whose range is a segment $]\leftarrow, x[$ then $\lambda <_{\mathrm{Ord}} \mu$, since μ and its order-type are order isomorphic.

```
Definition  ordinal_le r r' :=
  is_ordinal r & is_ordinal r' &  & order_type_le r r'.
Definition  ordinal_lt r r' := ordinal_le r r' & r <> r'.

Lemma ordinal_le_pr: forall r r',
   ordinal_le r r' = (
    is_ordinal r & is_ordinal r' &
     exists f, exists x,
    is_segment r' x & order_isomorphism f r (induced_order r' x)).
Lemma ordinal_le_pr1: forall r r',
   ordinal_le r r' = (
    is_ordinal r & is_ordinal r' &
     exists f, is_segment r' (range (graph f)) & order_morphism f r r').

Lemma ordinal_lt_pr1: forall r r',
   ordinal_lt r r' = (
    is_ordinal r & is_ordinal r' &
     exists f,  exists x,
        inc x (substrate r') &
        (range (graph f)) = segment r' x & order_morphism f r r').
Lemma ordinal_lt_pr2: forall a b,
  order b -> ordinal_lt a (order_type b) ->
     exists f,  exists x,
        inc x (substrate b) &
        (range (graph f)) = segment b x & order_morphism f a b.  (* 27 *)

Lemma  ordinal_lt_pr3: forall a x, is_ordinal a -> inc x (substrate a) ->
  ordinal_lt (order_type (induced_order a (segment a x))) a.  (* 28 *)

Lemma ordinal_le_order_r: order_r ordinal_le. (* 21 *)
Lemma ordinal_le_reflexive: forall x, is_ordinal x ->
  ordinal_le x x.
Lemma ordinal_le_transitive: forall x y z,
  ordinal_le x y -> ordinal_le y z -> ordinal_le x z.
Lemma ordinal_le_antisymmetric: forall x y,
  ordinal_le x y -> ordinal_le y x ->  x = y.
Lemma ordinal_le_antisymmetry1: forall a b,
  ordinal_le a b -> ordinal_lt b a -> False.
Lemma ordinal_lt_le_trans: forall a b c,
  ordinal_lt a b -> ordinal_le b c -> ordinal_lt a c.
Lemma ordinal_le_lt_trans: forall a b c,
  ordinal_le a b -> ordinal_lt b c -> ordinal_lt a c.
```

We pretend that $\mathrm{Ord}(\mathrm{E}_j) \leq \mathrm{Ord}(\sum \mathrm{E}_i)$ (since E_j is clearly isomorphic to a subset of the sum). Consider now a family of ordinals λ_i, put a well-ordering on the index set and denote by F the quantity $\mathrm{Ord}(\sum_{\mathrm{I}} \lambda_i)$. We have now $\lambda_i \leq \mathrm{F}$, so that that exists a segment E_i of F which is isomorphic to λ_i. Since the set of segments is well-ordered, there is an index j so that E_j is the least segment; this means that $\mathrm{E}_j \subset \mathrm{E}_i$ for all i. From this, we deduce $\lambda_j \leq \lambda_i$. We have thus proved that $\leq$ is a well-ordering. In particular, this is a total ordering.

```
Lemma ordinal_le_sum: forall r g j, order_sum_a r g -> worder r
```

```
  -> (forall i, inc i (domain g) -> worder (V i g)) -> inc j (domain g)
  -> ordinal_le (order_type (V j g)) (order_type (order_sum r g)). (* 27 *)

Theorem wordering_ordinal_le: worder_r ordinal_le.  (* 108 *)
Lemma ordinal_le_total_order: forall a b,
  is_ordinal a -> is_ordinal b ->
  ordinal_le a b \/ ordinal_le b a.
Lemma ordinal_le_total_order1: forall a b,
  is_ordinal a -> is_ordinal b ->
  ordinal_le a b \/ ordinal_lt b a.
```

We consider here some examples: ω will be the order type of $\mathbb{N}$, the set of natural integer, and $o(n)$ the order type of the set of integers $< n$ (with the natural orderings). We shall denote by 0, 1 and 2 the order types of some set with 0, 1 and 2 elements respectively.

```
Definition ord_zero := order_type (emptyset).
Definition ord_one := order_type (singleton (J emptyset emptyset)).
Definition ord_two := order_type canonical_doubleton_order.
Definition ord_Bnat n := order_type (interval_Bnatco b).
Definition ord_omega := order_type Bnat_order.
```

The emptyset is an ordered set. Its ordinal is zero. Since any set isomorphic to the emptyset is empty, the ordinal zero must be the empty set (for the same reason, the cardinal zero is the empty set).

```
Lemma emptyset_order: order emptyset.
Lemma emptyset_substrate : substrate emptyset = emptyset.
Lemma is_ordinal_0: is_ordinal ord_zero.
Lemma ordinal0_pr1: forall x, order x -> substrate x = emptyset ->
  order_type x = ord_zero.
Lemma ordinal0_pr: ord_zero = emptyset.
Lemma empty_substrate_zero: forall x, order x -> substrate x = emptyset ->
   x = ord_zero.
```

There is a unique ordering on a singleton. Thus all singletons are order isomorphic. They are well-ordered. Thus the ordinal one is the ordering of some set $\{x\}$ and has the form $\{(x,x)\}$. There are two orderings on a doubleton $\{x,y\}$, characterised by $x < y$ or $y < x$; one ordering is the opposite of the other, so that there is essentially a unique ordinal on a doubleton. For simplicity, we define the ordinal two as the order type of the canonical doubleton. A more conveninet definition would be $2 = 1 + 1$, but we have not defined the ordinal sum yet.

```
Lemma singleton_worder: forall x,
  let E := singleton (J x x) in
   substrate E = singleton x & worder E.
Lemma singleton_order_isomorphic: forall x y,
  order_isomorphic (singleton (J x x)) (singleton (J y y)).
Lemma singleton_order_isomorphic1: forall x,
  order x -> is_singleton (substrate x) ->
  exists v, x = singleton (J v v).
Lemma singleton_order_isomorphic2: forall x y,
  order x -> order y ->
  is_singleton (substrate x) -> is_singleton (substrate y)
  ->  order_isomorphic x y.
```

```
Lemma is_ordinal_1: is_ordinal ord_one.
Lemma ordinal_1_indep: forall x, order x -> is_singleton (substrate x) ->
  ord_one = order_type x.
Lemma ordinal_1_value: is_singleton (substrate ord_one).
Lemma ordinal_1_value_bis: exists x, ord_one = singleton (J x x).
Lemma is_ordinal_2: is_ordinal ord_two.
```

For any set x, x and its ordinal have the same cardinal, since an order isomorphism is a set-isomorphism. In particular, if $n \in \mathbf{N}$, if $o(n)$ is the ordinal of the interval $[0, n[$, then the cardinal of $o(n)$ is n. Thus, if μ is a finite ordinal, we have $o(\mathrm{Card}(\mu)$. The mapping $n \mapsto o(n)$ is injective. The ordinals 0, 1 and 2 are $o(0)$, $o(1)$ and $o(2)$.

Note: one can define the cardinal of a set E as the least ordinal μ equipotent to E. The axiom of choice asserts the existence of a well-ordering on E, hence at least one ordinal equipotent to E. The previous discussion says then that μ is finite, its cardinal is μ. One can show that if μ is countable its cardinal is ω (this is equivalent to say that if μ is an infinite ordinal, then $\omega \leq \mu$).

```
Lemma segments_iso1: forall a A B, worder a -> sub A B ->
  is_segment a A -> is_segment a B ->
  order_isomorphic (induced_order a B) (induced_order a A) -> A = B. (* 31 *)

Lemma is_ordinal_omega: is_ordinal ord_omega.
Lemma is_ordinal_ord_Bnat: forall n, inc n Bnat -> is_ordinal (ord_Bnat n).
Lemma cardinal_ord_Bnat: forall n, inc n Bnat ->
  cardinal (substrate (ord_Bnat n)) = n.
Lemma ord_Bnat_injective: forall n m, inc n Bnat -> inc m Bnat ->
  (ord_Bnat n = ord_Bnat m) -> n = m.
Lemma finite_ordinal: forall x, is_ordinal x -> is_finite_set (substrate x)
  ->  x = ord_Bnat (cardinal (substrate x)).
Lemma ord_zero_card: ord_Bnat card_zero = ord_zero.
Lemma ord_one_card: ord_Bnat card_one = ord_one.
Lemma ord_two_card_pr1: forall x,
  is_ordinal x -> card_two = cardinal (substrate x) ->
  x = ord_Bnat card_two.
Lemma ord_two_card: ord_Bnat card_two = ord_two.
```

8.3 Operations on order types

The order-type of the order-sum of a family of order-types will be called the *ordinal sum* and denoted by $\sum_{i\in \mathrm{I}} \lambda_i$ (abuse of notations here). The order-type of the order-product of a family will be called the *ordinal product.* If arguments are order-types so is the result of the operation; if arguments are ordinals so is the result (provided the index set is well-ordered for the sum, finite for the product).

```
Definition ord_sum r g := order_type (order_sum r g).
Definition ord_prod r g := order_type (order_product r g).
Definition ord_sum2 a b := order_type (order_sum2 a b).
Definition ord_prod2 a b := order_type (order_prod2 a b).

Lemma ord_sum_type: forall r g,
  order r -> substrate r = domain g -> fgraph g ->
  (forall i, inc i (domain g) -> order (V i g))
```

```
  -> is_order_type (ord_sum r g).
Lemma ord_sum_ordinal: forall r g,
  worder r -> substrate r = domain g -> fgraph g ->
  (forall i, inc i (domain g) -> is_ordinal (V i g))
  -> is_ordinal (ord_sum r g).
Lemma ord_prod_type: forall r g,
  worder r -> substrate r = domain g -> fgraph g ->
  (forall i, inc i (domain g) -> order (V i g))
  -> is_order_type (ord_prod r g).
Lemma ord_prod_ordinal: forall r g,
  worder r -> substrate r = domain g -> fgraph g ->
  (forall i, inc i (domain g) -> is_ordinal (V i g))
  -> is_finite_set (substrate r)
  -> is_ordinal (ord_prod r g).
```

We state some properties of the the sum or product of two order types.

```
Lemma ord_sum2_pr:  forall a b,
  ord_sum2 a b = ord_sum canonical_doubleton_order (variantLc a b).
Lemma ord_prod2_pr:  forall a b,
  ord_prod2 a b = ord_prod canonical_doubleton_order (variantLc b a).

Lemma ord_sum2_type: forall a b, order a -> order b ->
  is_order_type (ord_sum2 a b).
Lemma ord_sum2_ordinal: forall a b, worder a -> worder b ->
  is_ordinal (ord_sum2 a b).
Lemma ord_prod2_type: forall a b, order a -> order b ->
  is_order_type (ord_prod2 a b).
Lemma ord_prod2_ordinal: forall a b, worder a -> worder b ->
  is_ordinal (ord_prod2 a b).
```

Bourbaki claims that

$$\mathrm{Ord}(\sum_{\iota\in I} E_\iota) = \sum_{\iota\in I} \mathrm{Ord}(E_\iota).$$

There is an abuse of notations here. Let f denote the ordinal sum of ordered sets (*ordinal_sum* in Coq) and f' the ordinal sum of order-types (*ord_sum* in Coq). The claim is $\mathrm{Ord}(f(I,E_\iota)) = f'(I,\mathrm{Ord}(E_\iota))$, it is equivalent to $f'(I,E_\iota) = f'(I,\mathrm{Ord}(E_\iota))$ (lemma *ord_sum_invariant3*). This is the same as saying that $f(I,E_\iota)$ is isomorphic to $f(I,\mathrm{Ord}(E_\iota))$. We show in fact that $f(I,E_\iota)$ is isomorphic to $f(K,F_\kappa)$, whenever E_ι and $F_{g(\iota)}$ are isomorphic and where g is an order isomorphism between I and K. The same is true for the product. This is rather straightforward, but proofs are long (especially for the product).

```
Lemma ord_sum_invariant1: forall r r' f g g',
  order r -> substrate r = domain g -> fgraph g ->
  order r' -> substrate r' = domain g' -> fgraph g' ->
  order_isomorphism f r r' ->
  (forall i, inc i (substrate r) -> order_isomorphic (V i g)  (V (W i f) g'))
  ->  ord_sum r g = ord_sum r' g'.  (* 75 *)

Lemma ord_prod_invariant1: forall r r' f g g',
  worder r -> substrate r = domain g -> fgraph g ->
  order r' -> substrate r' = domain g' -> fgraph g' ->
  order_isomorphism f r r' ->
  (forall i, inc i (substrate r) -> order_isomorphic (V i g)  (V (W i f) g'))
```

```
  ->  ord_prod r g = ord_prod r' g'.  (* 120 *)

Lemma ord_sum_invariant2: forall r g g',
  order r -> substrate r = domain g -> fgraph g ->
  substrate r = domain g' -> fgraph g' ->
  (forall i, inc i (substrate r) -> order_isomorphic (V i g)  (V i g'))
  ->  ord_sum r g = ord_sum r g'.

Lemma ord_prod_invariant2: forall r g g',
  worder r -> substrate r = domain g -> fgraph g ->
  substrate r = domain g' -> fgraph g' ->
  (forall i, inc i (substrate r) -> order_isomorphic (V i g)  (V i g'))
  ->  ord_prod r g = ord_prod r g'.

Lemma ord_sum_invariant3: forall r g,
  order r -> substrate r = domain g -> fgraph g ->
  (forall i, inc i (substrate r) -> order (V i g)) ->
  ord_sum r g = ord_sum r (L (substrate r) (fun i => order_type (V i g))).
Lemma ord_prod_invariant3: forall r g,
  worder r -> substrate r = domain g -> fgraph g ->
  (forall i, inc i (substrate r) -> order (V i g)) ->
  ord_prod r g = ord_prod r (L (substrate r) (fun i => order_type (V i g))).

Lemma ord_sum_invariant4: forall r1 r2 r3 r4,
  order_isomorphic r1 r3 ->  order_isomorphic r2 r4 ->
  ord_sum2 r1 r2  =  ord_sum2 r3 r4.
Lemma ord_prod_invariant4: forall r1 r2 r3 r4,
  order_isomorphic r1 r3 ->  order_isomorphic r2 r4 ->
  ord_prod2 r1 r2  =  ord_prod2 r3 r4.

Lemma ord_sum_invariant5: forall a b c, order a -> order b -> order c ->
  order_isomorphic (order_sum2 a b) c ->
  ord_sum2 (order_type a) (order_type b) = order_type c.
Lemma ord_prod_invariant5: forall a b c, order a -> order b -> order c ->
  order_isomorphic (order_prod2 a b) c ->
  ord_prod2 (order_type a) (order_type b) = order_type c.
```

We have $\sum_{\iota\in I}\mu_\iota = \mu\lambda$ whenever $\mu_\iota = \mu$ for all indices $\iota \in I$ and I is isomorphic to λ (In Exercise 2.13, Bourbaki adds the condition I well-ordered, but this is not necessary). We first show the result when $I = \lambda$.

```
Lemma order_prod_pr1: forall a b,
  is_order_type a -> is_order_type b ->
  ord_prod2 a b = (ord_sum b (cst_graph (substrate b) a)).
Lemma order_prod_pr2: forall a b c,
  is_order_type a -> is_order_type b -> order_isomorphic b c ->
  ord_prod2 a b = (ord_sum c (cst_graph (substrate c) a)).
Lemma order_prod_pr3: forall a b,
  is_ordinal a -> is_ordinal b ->
  ord_prod2 a b = (ord_sum b (cst_graph (substrate b) a)).
Lemma order_prod_pr4: forall a b c,
  is_ordinal a -> is_ordinal b -> order_isomorphic b c ->
  ord_prod2 a b = (ord_sum c (cst_graph (substrate c) a)).
```

We show here that an ordinal sum remains unchanged if zero terms are removed. In a similar fashion, one can remove ones in a product. In fact, consider $\prod \mathrm{E}_i$, assume E_j is one

for $j \in \mathrm{I}-\mathrm{J}$; this means that $\mathrm{E}_j = \{w\}$ for some w; we know that the restriction to J is a bijection from $\prod_{\mathrm{I}} \mathrm{E}_i$ to $\prod_{\mathrm{J}} \mathrm{E}_i$. It is clearly an order isomorphism. We deduce

$$0 + x = x + 0 = x; \qquad 1 \cdot x = x \cdot 1.$$

```
Lemma zero_unit_sum_ord1: forall r g j,
  order_sum_a r g -> sub j (domain g) ->
  (forall i, inc i (complement (domain g) j) -> V i g = ord_zero) ->
  ord_sum r g = ord_sum (induced_order r j) (restr g j).   (* 41 *)
Lemma one_unit_prod_ord1: forall r g j,
  order_product_a r g -> sub j (domain g) ->
  (forall i, inc i (complement (domain g) j) -> V i g = ord_one) ->
  ord_prod r g = ord_prod (induced_order r j) (restr g j). (* 104 *)

Lemma ord_stable_plus:  forall a b, is_ordinal a -> is_ordinal b ->
  is_ordinal (ord_sum2 a b).
Lemma ord_stable_mult:  forall a b, is_ordinal a -> is_ordinal b ->
  is_ordinal (ord_prod2 a b).

Lemma unit_helper: forall x y j,
  let g :=  (variantLc x y) in
  is_order_type x -> is_order_type y -> sub j (domain g) ->
     (order_sum_a canonical_doubleton_order g
     & order_product_a canonical_doubleton_order g
     & order (induced_order canonical_doubleton_order j)
  & substrate (induced_order canonical_doubleton_order j) = j
  &substrate (induced_order canonical_doubleton_order j) = domain (restr g j)
  & fgraph (restr g j)
  & (forall i : Set, inc i (domain (restr g j)) -> order (V i (restr g j)))).

Lemma ord_0_plus_unit_r: forall x, is_order_type x ->
  ord_sum2 x ord_zero = x.
Lemma ord_0_plus_unit_l: forall x, is_order_type x ->
  ord_sum2 ord_zero x = x.
Lemma ord_1_mult_unit_r: forall x, is_order_type x ->
  ord_prod2 x ord_one  = x.
Lemma ord_1_mult_unit_l: forall x, is_order_type x ->
  ord_prod2  ord_one x = x.
```

We want to show the associativity of the ordinal sum and product.

$$\sum_{\iota\in\mathrm{I}} \mathrm{E}_\iota = \sum_{\lambda\in\mathrm{L}} \Big(\sum_{\iota\in\mathrm{J}_\lambda} \mathrm{E}_\iota \Big), \qquad \mathrm{I} = \sum_{\lambda\in\mathrm{L}} \mathrm{J}_\lambda.$$

$$\prod_{\iota\in\mathrm{I}} \mathrm{E}_\iota = \prod_{\lambda\in\mathrm{L}} \Big(\prod_{\iota\in\mathrm{J}_\lambda} \mathrm{E}_\iota \Big), \qquad \mathrm{I} = \sum_{\lambda\in\mathrm{L}} \mathrm{J}_\lambda.$$

There is an abuse of notations here. Since I is an ordinal sum, it is an ordering, and $\iota \in \mathrm{I}$ has to be interpreted as ι belongs to the support. On the other hand J_λ is a set, but not a subset of the support (the support is the union of the $\mathrm{J}_\lambda \times \{\lambda\}$).

In the first two theorems, we indicate the function that realises the isomorphisms. These functions are clearly bijective, it is a bit harder to show that they are order preserving (but the proofs are straightforward).

```
Lemma order_sum_assoc_iso:  forall r g r' g',
```

```
  order_sum_a r g ->  order_sum_a r' g' ->
  r = order_sum r' g' ->
  let order_sum_assoc_aux :=
    fun l =>
    order_sum (V l g') (L (substrate (V l g')) (fun i => V (J i l) g)) in
  let  order_sum_assoc :=
    order_sum r' (L (domain g')  order_sum_assoc_aux)
  in order_isomorphism (BL (fun x=> J (J (P x) (P (Q x))) (Q (Q x)))
      (sum_of_substrates g) (substrate (order_sum_assoc)))
  (order_sum r g) (order_sum_assoc).    (* 103 *)

Lemma ordinal_prod_assoc_iso:  forall r g r' g',
  order_product_a r g ->  order_sum_a r' g' ->
  r = order_sum r' g' ->
  worder r' ->
  (forall i, inc i (domain g') -> worder( V i g')) ->
  let order_sum_assoc_aux :=
    fun l =>
    order_product (V l g') (L (substrate (V l g')) (fun i => V (J i l) g)) in
  let  ordinal_prod_assoc :=
    order_product r' (L (domain g') order_sum_assoc_aux)
  in order_isomorphism (BL
     (fun z  => L (domain g') (fun l  =>
       L (substrate (V l g')) (fun j => V (J j l) z)))
      (prod_of_substrates g) (substrate (ordinal_prod_assoc)))
  (order_product r g) (ordinal_prod_assoc). (* 128 *)

Lemma order_sum_assoc1:  forall r g r' g',
  order_sum_a r g ->  order_sum_a r' g' ->
  r = order_sum r' g' ->
  let order_sum_assoc_aux :=
    fun l =>
    ord_sum (V l g') (L (substrate (V l g')) (fun i => V (J i l) g)) in
  ord_sum r g= ord_sum r' (L (domain g') (order_sum_assoc_aux)).

Lemma ordinal_prod_assoc1:  forall r g r' g',
  order_product_a r g ->  order_sum_a r' g' ->
  r = order_sum r' g' ->
  worder r' ->
  (forall i, inc i (domain g') -> worder( V i g')) ->
  let order_sum_assoc_aux :=
    fun l =>
    order_product (V l g') (L (substrate (V l g')) (fun i => V (J i l) g)) in
  ord_prod r g = ord_prod  r' (L (domain g') order_sum_assoc_aux).
```

We may consider a set with three elements, well-order it, and use this to define $a+b+c$ or $a\cdot b\cdot c$, the sum or product of three terms. We could then partition our set, taking apart the least or greatest element, and apply twice the associativity theorem, then get

$$a+(b+c)=(a+b)+c \text{ and } a\cdot(b\cdot c)=(a\cdot b)\cdot c.$$

This is rather difficult; we prefer a direct proof; there is no difficulty here, except that the proofs are rather long, especially for the sum. We have similarly

$$a\cdot(b+c)=a\cdot b+a\cdot c.$$

Direct proof is easy. Note that there is a natural bijection between $(b+c)\cdot a$ and $b\cdot a+c\cdot a$ but it is not always order preserving.

¶**TO DO**. We shall see later that addition is not commutative by proving that $\omega+1\neq 1+\omega$, multiplication is not commutative by proving $\omega\cdot 2\neq 2\cdot\omega$. Finally, we cannot change the ordering of the factors in the distributivity formula since $(1+1)\cdot\omega\neq 1\cdot\omega+1\cdot\omega$.

```
Lemma order_sum_assoc2: forall a b c,
  order a -> order b -> order c ->
  ord_sum2 a (ord_sum2 b c) = ord_sum2 (ord_sum2 a b) c.  (* 240 *)

Lemma ordinal_prod_assoc2: forall a b c,
  order a -> order b -> order c ->
  ord_prod2 a (ord_prod2 b c) = ord_prod2 (ord_prod2 a b) c. (* 107 *)
Lemma order_sum_distributive: forall x y z,      (* 155 *)
  order x -> order y -> order z ->
  ord_prod2 z (ord_sum2 x y) = ord_sum2 (ord_prod2 z x) (ord_prod2 z y).

Lemma order_sum_assoc3: forall a b c,
  is_ordinal a -> is_ordinal b -> is_ordinal c ->
  ord_sum2 a (ord_sum2 b c) = ord_sum2 (ord_sum2 a b) c.
Lemma ordinal_prod_assoc3: forall a b c,
  is_ordinal a -> is_ordinal b -> is_ordinal c ->
  ord_prod2 a (ord_prod2 b c) = ord_prod2 (ord_prod2 a b) c.
Lemma order_sum_distributive3: forall a b c,
  is_ordinal a -> is_ordinal b -> is_ordinal c ->
  ord_prod2 c (ord_sum2 a b) = ord_sum2 (ord_prod2 c a) (ord_prod2 c b).
```

We show here

$$\sum_{\emptyset} \mathrm{E}_i = 0, \quad \prod_{\emptyset} \mathrm{E}_i = 1, \quad \sum_{i\in\{a\}} \mathrm{E}_i = \mathrm{E}_a, \quad \prod_{i\in\{a\}} \mathrm{E}_i = \mathrm{E}_a.$$

In the last two formulas, it is assumed that each E_i is an ordered set, and that E_a is an order-type (without this condition, we should use $\mathrm{Ord}(\mathrm{E}_a)$ instead).

```
Lemma ord_sum_emptyset: forall r x,
  order r -> substrate r = domain x -> fgraph x ->
  substrate r = emptyset ->
  ord_sum r x = ord_zero.
Lemma ord_prod_emptyset: forall r x,
  worder r -> substrate r = domain x -> fgraph x ->
  substrate r = emptyset ->
  ord_prod r x = ord_one.

Lemma ord_sum_singleton: forall r x e,
  order r -> substrate r = domain x -> fgraph x ->
  (forall i, inc i (domain x) -> order (V i x)) ->
  substrate r = singleton e -> is_order_type (V e x) ->
  ord_sum r x = V e x. (* 29 *)
Lemma ord_prod_singleton: forall r x e,
  worder r -> substrate r = domain x -> fgraph x ->
  (forall i, inc i (domain x) -> order (V i x)) ->
  substrate r = singleton e ->  is_order_type (V e x) ->
  ord_prod r x = V e x.  (* 40 *)
```

If zero is a factor of a product, then the product is zero. Conversely, if the product is zero then at least one factor is zero (we state the theorem in the case of two ordinals, but it is true for any orderings). Zero is the least ordinal. A non-zero ordinal satisfies $1 \leq x$.

```
Lemma ord_zero_absorbing: forall r g,
   order_product_a r g ->
    (exists i, inc i (domain g) & V i g = ord_zero)
  -> ord_prod r g = ord_zero.
Lemma ord_0_prod_l: forall x, is_order_type x ->
  ord_prod2 ord_zero x = ord_zero.
Lemma ord_0_prod_r: forall x, is_order_type x ->
  ord_prod2 x ord_zero = ord_zero.
Lemma ord0_prod_r: forall x, is_ordinal x ->
  ord_prod2 x ord_zero = ord_zero.
Lemma ord0_prod_l: forall x, is_ordinal x ->
  ord_prod2 ord_zero x = ord_zero.
Lemma ord_prod2_nz: forall a b, is_ordinal a -> is_ordinal b ->
   a <> ord_zero -> b <> ord_zero -> ord_prod2 a b <> ord_zero.

Lemma zero_least_ordinal: forall x, is_order_type x -> order_type_le ord_zero x.
Lemma zero_least_ordinal1: forall x, is_ordinal x -> ordinal_le ord_zero x.
Lemma zero_least_ordinal2: forall x, is_order_type x -> x <> ord_zero ->
   order_type_le ord_one x.
Lemma zero_least_ordinal3: forall x, is_ordinal x -> x <> ord_zero ->
   ordinal_le ord_one x. (* 42 *)
```

We show that $\sum a_i \leq \sum b_i$ whenever $a_i \leq b_i$. This is true for order types as well as ordinals. We then deduce that the sum over J is less than the sum over I when $J \subset I$.

```
Lemma ord_sum_increasing1: forall r f g ,
  order_sum_a r f -> order_sum_a r g ->
  (forall x, inc x (domain f) -> order_type_le (V x f) (V x g)) ->
  order_type_le (ord_sum r f) (ord_sum r g).  (* 64 *)
Lemma ord_sum_increasing2: forall r f g, worder r ->
  fgraph f -> fgraph g -> substrate r = domain f -> substrate r = domain g ->
  (forall x, inc x (domain f) -> ordinal_le (V x f) (V x g)) ->
  ordinal_le (ord_sum r f) (ord_sum r g).

Lemma ord_sum_increasing3: forall r f j,
  order_sum_a r f ->
  sub j (domain f) ->
  (forall j, inc j (domain f) -> is_order_type (V j f)) ->
  order_type_le (ord_sum (induced_order r j) (restr f j)) (ord_sum r f). (* 29 *)
Lemma ord_sum_increasing4: forall r f j,  worder r ->
  fgraph f -> substrate r = domain f ->
  sub j (domain f) ->
  (forall j, inc j (domain f) -> is_ordinal (V j f)) ->
  ordinal_le (ord_sum (induced_order r j) (restr f j)) (ord_sum r f).
```

We show here the same for the product. Note that the product over J is less than the sum over I when $J \subset I$, provided that factors in $I - J$ are non-zero.

```
Lemma ord_prod_increasing1: forall r f g ,
  order_product_a r f -> order_product_a r g ->
  (forall x, inc x (domain f) -> order_type_le (V x f) (V x g)) ->
```

```
  order_type_le (ord_prod r f) (ord_prod r g).  (* 67 *)
Lemma ord_prod_increasing2: forall r f g, worder r ->
  fgraph f -> fgraph g -> substrate r = domain f -> substrate r = domain g ->
  (forall x, inc x (domain f) -> ordinal_le (V x f) (V x g)) ->
  is_finite_set (substrate r) ->
  ordinal_le (ord_prod r f) (ord_prod r g).
Lemma ord_prod_increasing3: forall r f j,   (* 30 *)
  order_product_a r f ->
  sub j (domain f) ->
  (forall j, inc j (domain f) -> is_order_type (V j f)) ->
  (forall x, inc x  (complement (domain f) j) -> V x f <> ord_zero) ->
  order_type_le (ord_prod (induced_order r j) (restr f j)) (ord_prod r f).
Lemma ord_prod_increasing4: forall r f j,  worder r ->
  fgraph f -> substrate r = domain f ->
  sub j (domain f) ->
  (forall j, inc j (domain f) -> is_ordinal (V j f)) ->
  is_finite_set (substrate r) ->
  (forall x, inc x (complement (domain f) j)  -> V x f <> ord_zero) ->
  ordinal_le (ord_prod (induced_order r j) (restr f j)) (ord_prod r f).
```

We deduce the following relations. We prove it when arguments are ordinals, but the result holds in the case of order-types. In the case of a product, we may assume sometimes that arguments are non-zero.

$$a \le b \text{ and } a' \le b' \implies a + a' \le b + b', \qquad a \le a + b, \qquad b \le a + b.$$

$$a \le b \text{ and } a' \le b' \implies a \cdot a' \le b \cdot b', \qquad a \le a \cdot b, \qquad b \le a \cdot b.$$

```
Lemma ord_sum2_increasing1: forall a b c d,
  ordinal_le a b -> ordinal_le c d ->
  ordinal_le (ord_sum2 a c) (ord_sum2 b d).
Lemma ord_sum2_increasing2_aux: forall a b,
  is_ordinal a -> is_ordinal b ->
  ( ordinal_le a (ord_sum2 a b) & ordinal_le b (ord_sum2 a b)).
Lemma ord_sum2_increasing2: forall a b,
  is_ordinal a -> is_ordinal b ->
  ordinal_le a (ord_sum2 a b).
Lemma ord_sum2_increasing3: forall a b,
  is_ordinal a -> is_ordinal b ->
  ordinal_le b (ord_sum2 a b).

Lemma ord_prod2_increasing1: forall a b c d,
  ordinal_le a b -> ordinal_le c d ->
  ordinal_le (ord_prod2 a c) (ord_prod2 b d).
Lemma ord_prod2_increasing2: forall a b,
  is_ordinal a -> is_ordinal b -> b <> ord_zero ->
  ordinal_le a (ord_prod2 a b).   (* 31 *)
Lemma ord_prod2_increasing3: forall a b,
  is_ordinal a -> is_ordinal b -> a <> ord_zero ->
  ordinal_le b (ord_prod2 a b).   (* 31 *)

Lemma ordinal_lt_succ: forall a b, is_ordinal a -> is_ordinal b ->
  ordinal_lt a (ord_sum2 b ord_one) = ordinal_le a b.  (* 50 *)
Lemma ordinal_succ_lt: forall a b, is_ordinal a -> is_ordinal b ->
  ordinal_lt a b = ordinal_le (ord_sum2 a ord_one) b.
Lemma ordinal_succ_inj: forall a b, is_ordinal a -> is_ordinal b ->
   (ord_sum2 a ord_one = ord_sum2 b ord_one) -> a = b.
```

We pretend here that the cardinal of the ordinal sum $a+b$ is the cardinal sum of the cardinals of a and b. If $o(n)$ is the ordinal of the interval $[0,n[$, we get $o(n)+o(m)=o(n+m)$. In particular $1+1=2$ (considered as a sum of ordinals). We show that $n+\omega=n\cdot\omega=\omega$, whenever n is a finite ordinal (and non-zero in the case of a product). This gives an example where $x+a=y+a$ or $x\cdot a=y\cdot a$ does not imply $x=y$. We have $(1+1)\cdot\omega=2\cdot\omega=\omega$. On the other have $1\cdot\omega+1\cdot\omega=\omega+\omega$. We shall see ina minute that this is not ω, thsi gives a counter-example to distributivity. In the case, of a sum, if $x\in o(n)$ we define $f(x)=x$, otherwise $f(x)=x+n$. In the case of the product, if $x\in o(n)\cdot\omega$, then x is a pair (a,b) with $b<n$. We define $f(x)=an+b$. This gives the desired isomorphism (the inverse of f is defined Euclidean division).

```
Lemma cardinal_ord_sum: forall a b, is_ordinal a -> is_ordinal b ->
  cardinal (substrate (ord_sum2 a b)) =
  card_plus (cardinal (substrate a)) (cardinal (substrate b)).
Lemma cardinal_ord_sum2: forall a b,
  inc a Bnat -> inc b Bnat ->
  ord_sum2 (ord_Bnat a) (ord_Bnat b) = ord_Bnat (card_plus a b).
Lemma ord_11_2: ord_two = ord_sum2 ord_one ord_one.

Lemma ord_plus_int_omega: forall n, inc n Bnat ->
  ord_sum2 (ord_Bnat n) ord_omega = ord_omega.  (* 87 *)
Lemma ord_mult_int_omega: forall n, inc n Bnat ->
  n <> card_zero ->
  ord_prod2 (ord_Bnat n) ord_omega = ord_omega. (* 106 *)
Lemma ordinal_a_ne_ab: forall a b, is_ordinal a -> is_ordinal b ->
   a = ord_sum2 a b -> b = ord_zero.  (* 49 *)
```

We end with a lemma that says $a+b=a$ implies $b=0$ (note that a is naturally isomorphic to a segment of $a+b$, so that we have a segment of $a+b$ isomorphic to the whole set).

```
Lemma ordinal_a_ne_ab: forall a b, is_ordinal a -> is_ordinal b ->
   a = ord_sum2 a b -> b = ord_zero.  (* 49 *)
```

8.4 Strict ordering

Given an ordinal a, its successor will be $a+1$. It is distinct from a; hence $a<a+1$. In particular we have $0<1<2$.

```
Lemma ord_not01: ord_zero <> ord_one.
Lemma succ_ordinal: forall a, is_ordinal a -> is_ordinal (ord_sum2 a ord_one)
Lemma ord_succ_lt: forall a, is_ordinal a -> ordinal_lt a (ord_sum2 a ord_one)
Lemma ord_lt_12: ordinal_lt ord_one ord_two.
Lemma ord_lt_01: ordinal_lt ord_zero ord_one.
Lemma ord_lt_02: ordinal_lt ord_zero ord_two.
```

We show here

$$\alpha<\beta \iff \alpha+1\leq\beta \text{ and } \alpha<\beta+1 \iff \alpha\leq\beta.$$

Assume first $\alpha<\beta+1$. This means that α is isomorphic to a segment S_x of $\beta+1$, which is an ordering on a disjoint union $B_1\cup C_2$. since C is a singleton, the relation $y<x$ implies $y\in B$, hence α is isomorphic to a subset of β.

Assume $\alpha\leq\beta$. We know $\beta<\beta+1$ thus $\alpha<\beta+1$. Assume $\alpha+1\leq\beta$. Since $\alpha<\alpha+1$ we get $\alpha<\beta$. Conversely, since the ordering of ordinals is total, if $\alpha<\beta$ we have $\alpha+1\leq\beta$ since

the other alternative $\beta < \alpha + 1$ yields $\beta \leq \alpha$, absurd. We deduce injectivity of the successor function.

```
Lemma ordinal_lt_succ: forall a b, is_ordinal a -> is_ordinal b ->
  ordinal_lt a (ord_sum2 b ord_one) = ordinal_le a b.  (* 50 *)
Lemma ordinal_succ_lt: forall a b, is_ordinal a -> is_ordinal b ->
  ordinal_lt a b = ordinal_le (ord_sum2 a ord_one) b.
Lemma ordinal_succ_inj: forall a b, is_ordinal a -> is_ordinal b ->
   (ord_sum2 a ord_one = ord_sum2 b ord_one) -> a = b.
```

We consider here subtraction. The hypothetical quantity $a-b$ should satisfy $(a-b)+b = a$ by associativity. If we take $b = 1$, this means that a is a successor, and this is is not always the case. Moreover, take $a = b = \omega$, then any integer can be used for $a - b$, thus there is no uniqueness. For this reason, we shall consider $(-b) + a$, it satisfies $(-a) + (a + b) = b$, and is defined whenever $b \leq a$. The notation $(-b) + a$ is strange and we shall not use it.

Assume $a \leq b$, where a and b are ordinals. Let f be an isomorphim of a onto a segment of b, and C the complementary of the image. This is a well-ordered set, and $a+$C is isomorphic to b. We have then $a+$Ord(C) $= b$. If $c =$ Ord(C), we get $a + c = b$. Bourbaki notices that $c \leq b$ (this is trivial). He also says that if $a + c = b$, then $a \leq b$, which is equally trivial.

Assume now $a+x = a+y$. If $x \leq y$ we have $y = x+z$, for some z, hence $a+x = (a+x)+z$ by associativity, thus $z = 0$. From this we deduce: if $a+x = a+y$ then $x = y$. It implies uniqueness of subtraction.

Exercice 2.15(a) says that if $a \leq b$, then $c+a \leq c+b$, $a+c \leq b+c$, $c \cdot a \leq c \cdot b$ and $a \cdot c \leq b \cdot c$. We have already shown these claims. Assume now $a < b$ then $c + a < c + b$ and $c \cdot a < c \cdot b$ when $c \neq 0$. The first result is a consequece of $c + a = c + b \implies a = b$. The second result is equivalent to $c \cdot a = c \cdot b \implies a = b$ for $c \neq 0$. We have now solved Exercise 2.15, parts (a), (d) and (e).

```
Definition ord_sub a b := choose (fun c => is_ordinal c & a = ord_sum2 b c).

Lemma ord_sub_exists: forall a b, ordinal_le a b ->
   exists c, is_ordinal c & b = ord_sum2 a c. (* 73 *)

Lemma ord_sub_pr: forall a b, ordinal_le a b ->
  (is_ordinal (ord_sub b a) & b = ord_sum2 a (ord_sub b a)).
Lemma ord_plus_simp_left: forall a b c,
  is_ordinal a -> is_ordinal b -> is_ordinal c ->
  ord_sum2 c a = ord_sum2 c b -> a = b.

Lemma ord_sub_pr1: forall a b,
   is_ordinal a -> is_ordinal b ->
   ord_sub (ord_sum2 a b) a = b.
Lemma ord_sub_smaller: forall a b,
  ordinal_le a b -> ordinal_le (ord_sub b a) b.

Lemma ord_plus_compat_lt: forall a b c,
  ordinal_lt a b -> is_ordinal c ->
  ordinal_lt (ord_sum2 c a)(ord_sum2 c b).
Lemma ord_mult_compat_lt: forall a b c,
  ordinal_lt a b -> is_ordinal c -> c <> ord_zero ->
  ordinal_lt (ord_prod2 c a)(ord_prod2 c b).
Lemma ord_mult_simp_left: forall a b c,
```

```
  is_ordinal a -> is_ordinal b -> is_ordinal c -> c <> ord_zero ->
  ord_prod2 c a = ord_prod2 c b -> a = b.
```

In Exercice 14(c) Bourbaki say that if α is an ordinal, then the relation "ξ is an ordinal and $\xi \leq_{\mathrm{Ord}} \alpha$" is collectivizing in ξ. This means that there is a set O'_α whose elements are all sets ξ such that $\xi \leq_{\mathrm{Ord}} \alpha$ (note that this implies that ξ is an ordinal). There is also a set O_α whose elements are all sets ξ such that $\xi <_{\mathrm{Ord}} \alpha$. This set is well-ordered by $\leq_{\mathrm{Ord}}$ and $\mathrm{Ord}(\mathrm{O}_\alpha) = \alpha$. Notice that the first set is the set of all order-types of segments S of α, and the second set is obtained by considering the strict segments, which are of the form $]\leftarrow, x[$. The mapping $x \mapsto \mathrm{Ord}(]\leftarrow, x[)$ is the desired isomorphism. The important property here is that two distinct segments of a well-ordered sets are never isomorphic.

```
Definition set_of_ordinal_le a:=
   fun_image (set_of_segments a)  (fun z => order_type (induced_order a z)).
Definition set_of_ordinal_lt a:=
  fun_image (substrate a) (fun z => order_type (induced_order a (segment a z))).

Lemma segments_iso2: forall a A B, worder a ->
   inc A  (set_of_segments a) -> inc B  (set_of_segments a)  ->
   order_isomorphic (induced_order a A)(induced_order a B) -> A = B.

Lemma set_ord_le_prop: forall a, is_ordinal a ->
  (forall x, inc x (set_of_ordinal_le a) = ordinal_le x a).
Lemma set_ord_lt_prop: forall a, is_ordinal a ->
  (forall x, inc x (set_of_ordinal_lt a) = ordinal_lt x a). (* 26 *)
Lemma set_ord_lt_prop2: forall a, is_ordinal a ->
  order_isomorphism (BL (fun z => order_type (induced_order a (segment a z)))
       (substrate a) (set_of_ordinal_lt a))
    a (graph_on ordinal_le (set_of_ordinal_lt a)).  (* 62 *)
Lemma set_ord_lt_prop3: forall a, is_ordinal a ->
  order_type (graph_on ordinal_le (set_of_ordinal_lt a)) = a.
```

Exercice 14(d) says that, for every family of ordinals $(\xi_i)_{i\in \mathrm{I}}$, there exists a unique ordinal α such that "λ is an ordinal and $\xi_i \leq \lambda$ for all $i \in \mathrm{I}$" is equivalent to $\alpha \leq \lambda$. It is called the least upper bound or supremum (by abuse of language, since $\leq$ is an ordering without graph). Let $\mathrm{Q}(\lambda)$ be the property that $\xi_i \leq \lambda$ for all $i \in \mathrm{I}$, and let $\mathrm{P}(\lambda)$ be the property "λ is an ordinal and $\mathrm{Q}(\lambda)$". Whatever Q, there is at most one ordinal α such that $\alpha \leq \lambda$ is equivalent to $\mathrm{P}(\lambda)$ (by antisymmetry of $\leq$). Assume $\mathrm{P}(\lambda)$ true for some λ. Then the supremum is the least such λ. In what follows, we consider a set of ordinals rather that a family; then supremum of the family is the supremum of the range. It exists, since the ordinal sum of the family is an upper bound.

```
Definition ord_sup_pr E x:=
   is_ordinal x &
   forall y, ordinal_le x y =
    (is_ordinal y & forall i, inc i E -> ordinal_le i y).
Definition ord_sup E := choose (fun x => ord_sup_pr E x).
Definition ord_supf f := ord_sup (range f).

Lemma ord_sup_unique: forall E x y,
  ord_sup_pr E x  -> ord_sup_pr E y -> x = y.
Lemma ord_sup_pr1: forall E y, is_ordinal y ->
  (forall i, inc i E -> ordinal_le i y) ->
  exists x, (ord_sup_pr E x & ordinal_le x y).
```

```
Lemma ord_sup_pr2: forall E, (forall i, inc i E -> is_ordinal i) ->
  ord_sup_pr E (ord_sup E).

Lemma ord_sup_pr3: forall E,
  (forall i, inc i E -> is_ordinal i) ->
  forall i, inc i E -> ordinal_le i (ord_sup E).
Lemma ord_sup_pr4: forall E y,
  (forall i, inc i E -> is_ordinal i) ->
  is_ordinal y -> (forall i, inc i E -> ordinal_le i y) ->
  ordinal_le (ord_sup E) y.
Lemma ord_sup_pr5: forall f, fgraph f ->
  (forall i, inc i (domain f) -> is_ordinal (V i f)) ->
  ( (forall i, inc i (domain f) -> ordinal_le (V i f) (ord_supf f))
  & (forall y, is_ordinal y ->
    (forall i, inc i (domain f) -> ordinal_le (V i f) y) ->
    ordinal_le  (ord_supf f) y)).
Lemma ord_sup_pr6: forall E,
  (forall i, inc i E -> is_ordinal i) ->
  greatest_element  (graph_on ordinal_le (tack_on E (ord_sup E)))
    (ord_sup E).
```

Bourbaki consider now the supremum of O_α. Call it β. If $\beta+1<\alpha$ then $\beta+1$ is in O_α thus $\leq\beta$, absurd. Thus $\alpha\leq\beta+1$. We have have equality, thus $\alpha=\beta+1$; otherwise we know that this is is the same as $\alpha\leq\beta$, but $\beta\leq\alpha$ (since α is an upper bound of O_α), thus $\alpha=\beta$.

If $\alpha\leq\beta+1$, we say that β is the predecessor of α; such an ordinal is unique. In the other case, we say that α is a limit ordinal. Note that these cases are mutually disjoint: if $\beta+1=\alpha=\sup(O_\alpha)$, then β is an upper bound of O_α, thus $\beta+1\leq\beta$, absurd.

```
Lemma ord_sup_pr7: forall a, is_ordinal a ->
  let b := ord_sup (set_of_ordinal_lt a) in
    b = a \/ a = ord_sum2 b ord_one.
```

We consider now Exercise 15. We have already shown a big part of it. Part (b) asserts that no set contains all ordinals; for if F is a set containing some ordinals, α its supremum, then α is stricly smaller than its successor, so that the successor cannot be in F.

Point (c) says Let α, β, μ be three ordinals. Show that each of the relations $\mu+\alpha<\mu+\beta$, $\alpha+\mu<\beta+\mu$ implies $\alpha<\beta$; and that each of the relations $\mu\alpha<\mu\beta$, $\alpha\mu<\beta\mu$ implies $\alpha<\beta$ provided that $\mu>0$. Proof; if the conclusion were false, we would have $\beta\leq\alpha$. We can add or multiply μ, use transitivity, and we get $x<x$ for some x. Note that if $\mu=0$, all products are zero, hence the "provided that $\mu>0$" is useless.

```
Lemma ordinal_not_collectivizing: forall E,
   (forall x, is_ordinal x -> inc x E) -> False.

Lemma ord_plus_compat_lt1: forall a b c,
  is_ordinal a -> is_ordinal b ->is_ordinal c ->
  ordinal_lt (ord_sum2 c a) (ord_sum2 c b)
  -> ordinal_lt a b.
Lemma ord_plus_compat_lt2: forall a b c,
  is_ordinal a -> is_ordinal b ->is_ordinal c ->
  ordinal_lt (ord_sum2 a c) (ord_sum2 b c)
  -> ordinal_lt a b.
Lemma ord_prod_compat_lt1: forall a b c,
```

```
  is_ordinal a -> is_ordinal b ->is_ordinal c ->
  ordinal_lt (ord_prod2 c a) (ord_prod2 c b)
  -> ordinal_lt a b.
Lemma ord_prod_compat_lt2: forall a b c,
  is_ordinal a -> is_ordinal b ->is_ordinal c ->
  ordinal_lt (ord_prod2 a c) (ord_prod2 b c)
  -> ordinal_lt a b.
```

Part (f) says: let a, b and b and c be three ordinals such that $a < b \cdot c$. There exists two ordinals q and r such that $a = b \cdot q + r$, $r < b$ and $q < c$, and these ordinals are uniquely defined by these relations. Note that $a < b \cdot c$ implies $b \neq 0$. Conversely, if b is not zero, we have $a < b \cdot (a+1)$, so this operation is really the division of $a$by b.

Consider the set of all Q such that $a < b \cdot \mathrm{Q}$, it is non-empty, since it contains c. Assume that its least element is a successor $q+1$. We have $q < c$ and $bq \leq a$, thus $a = b \cdot q + r$ for some r. Assume $b \leq r$ so that $r = b + r'$. Thus we have $a = b \cdot q + b + r' = b \cdot \mathrm{Q} + r'$. This contradicts $a < b\mathrm{Q}$. This is the same argument as that used to define division of finite cardinals, and holds in the case of finite ordinals. However, in general, we do not know that Q is not a limit ordinal. Hence we proceed as follows.

By assumption a is isomorphic to a strict segment S_x of $b \cdot c$. Consider y such that $y_\alpha = x_\alpha$ and y_β is the least element of b. This element is in the product and $y \leq x$. The ordinal a is the ordinal sum of the ordinal of the segment S_y and the ordinal of the interval $[y, x[$. If t is in $b \cdot c$, then $t_\alpha \in c$ and $t_\beta \in b$. Moreover $t < y$ is equivalent to $t_\alpha < x_\alpha$, while $y \leq t < x$ is equivalent to $t_\alpha = x_\alpha$ and $t_\beta < x_\beta$. This means that the ordinal of S_y is the ordinal product of b and $\mathrm{Ord}(]\leftarrow, x_\alpha[)$, while the ordinal of $[y, x[$ is $\mathrm{Ord}(]\leftarrow, x_\beta[)$ (the segments beeing respectively in c and b).

```
Definition ord_div_pr a b c q r :=
  is_ordinal q & is_ordinal r &
   a = ord_sum2 (ord_prod2 b q) r
  & ordinal_lt r b & ordinal_lt q c.
Lemma ord_div_nonzero_b: forall a b c,
  is_ordinal a -> is_ordinal b ->is_ordinal c ->
  ordinal_lt a (ord_prod2 b c)  -> b <> ord_zero.
Lemma ord_div_nonzero_b_bis: forall a b,
  is_ordinal a -> is_ordinal b -> b <> ord_zero ->
  exists c, is_ordinal c & ordinal_lt a (ord_prod2 b c).
Lemma ord_division_unique: forall a b c q r q' r',
  is_ordinal a -> is_ordinal b -> is_ordinal c ->
  ord_div_pr a b c q r -> ord_div_pr a b c q' r' ->
  (q = q' & r = r').   (* 27 *)

Lemma ord_division_exists: forall a b c,
  is_ordinal a -> is_ordinal b -> is_ordinal c ->
  ordinal_lt a (ord_prod2 b c) ->
  exists q, exists r, ord_div_pr a b c q r. (* 190 *)
```

8.5 Indecomposable ordinals

We start some properties of ω. We say that a is a limit ordinal if it is neither zero nor a successor. Note that the order-sum of any set and a singletion has a greatest element; thus a successor has a greatest element. Thus, a limit ordinal is a non-zero ordinal without greatest

element. In particular, it has to be infinite, and the least limit ordinal is ω. We start with: if x is finite and y infinite, then $x < \omega \leq y$.

```
Definition limit_ordinal x :=
   is_ordinal x & x <> ord_zero &
   (forall y, is_ordinal y -> ord_sum2 y ord_one <> x).

Lemma omega_infinite0: cardinal (substrate ord_omega) = cardinal Bnat.
Lemma omega_infinite1: forall x, is_ordinal x ->
  is_finite_set (substrate x) -> ordinal_lt x ord_omega.
Lemma omega_infinite2: forall x, is_ordinal x ->
  infinite_set (substrate x) -> ordinal_le ord_omega x.
Lemma omega_infinite3: forall x, limit_ordinal x ->
  ordinal_le ord_omega x.
Lemma successor_has_greatest: forall a, order a ->
  exists y, greatest_element (ord_sum2 a ord_one) y.
Lemma omega_infinite4: limit_ordinal ord_omega.
```

We know that $a + b = c$ implies $a \leq c$ and $b \leq c$. In general we have $a < c$ and $b < c$ (in the case of finite cardinals, if $a \neq 0$, we get $b < c$). An ordinal is called *indecomposable* if this never happens; in other terms, c is indecomposable if c is non-zero, and never the sum of two ordinals a and b such that $a < c$ and $b < c$.

We prove here the following claims: (a) A non-zero ordinal x is indecomposable if and only if $y < x$ implies $y + x = x$; (b) if $y > 0$, and $x \neq 1$, then x indecomposable if and only if yx is indecomposable and (c) if x is indecomposable and $y < x$ then y divides x and the quotient is indecomposable.

Result (a) is follows from existence of division. Assume now $z < yx$, x indecomposable, $x > 1$. By division we have $z = yq + r$, with $r < y$ and $q < x$. Thus $z + yx \leq yq + r + yx \leq yq + y + yx = y(q + 1 + x)$. If $q + 1 < x$, this is yx. We deduce $z + yx = yx$. Otherwise, we have $q + 1 = x$; this is absurd since $1 < x$. This shows (b). Finally, assume x is indecomposable and $y < x$. Statement (b) is true if $x = yx$. Otherwise we have $x < yx$, thus $x = yq + r$ with $q < x$ and $r < y$. Since x is indecomposable, one of the two terms y and r must be x. Since $r < y < x$ we get $x = yq$. If $q > 1$, point (b) implies that q is indecomosable. The result is aslo true if $q = 1$.

If x is any ordinal, the set of indecomposable ordinals $\leq x$ has a greatest element. Consider the set of all $(-y) + x$ where y is indecomposable ane $\leq x$. This is a non-empty set (consider $y = 1$) and has a least element. This is obvious if x is indecomposable.

```
Definition ord_indecomposable z :=
   z <> ord_zero &
  (forall x y, ordinal_lt x z -> ordinal_lt y z -> ord_sum2 x y <> z).

Lemma indecomposable_pr: forall x a b,
  is_ordinal x -> is_ordinal a ->is_ordinal b ->
  ord_indecomposable x -> ord_sum2 a b = x -> (a = x \/ b = x).
Lemma indecomp_example: forall x, is_ordinal x -> x <> ord_zero ->
  ~ (ord_indecomposable (ord_sum2 x ord_one)).
Lemma indecomp_one: ord_indecomposable ord_one.
Lemma indecomposable_prop: forall x, is_ordinal x -> x <> ord_zero ->
  ord_indecomposable x = (forall y, ordinal_lt y x -> ord_sum2 y x = x).
Lemma indecomposable_prod: forall x y, is_ordinal x -> x <> ord_zero ->
  x <> ord_one -> is_ordinal y -> y <> ord_zero ->
```

```
ord_indecomposable x = ord_indecomposable (ord_prod2 y x).  (* 39 *)
```

We shall see later on that $2\omega = \omega$. We know that $\omega 1 = \omega$. We shall see that $\omega + \omega \neq \omega$.

to do. $1+1=2$, $a \prec a+b$, $b \prec a+b$, $\mathbf{N}^* = \mathbf{N}^* + 1$ $a < a+1$ if a ordinal. $\omega + 1 \neq \omega$, $1+\omega = \omega$, $\omega.2 \neq 2.\omega$

The relation $(a+b).c = (a.c)+(b.c)$ may be false. For instance, Exercise 6.11 considers the case $c = \omega$; whenever n is an integer, we have $n.\omega = \omega$. We get a counter-example by taking $a = 1$, since $\omega + \omega \neq \omega$. The relation $a+b = b+a$ may be false (take $a = 1$ and $b = \omega$). The relation $a.b = b.a$ may be false (take $a = 2$ and $b = \omega$).

TODO show $1+\omega = 0+\omega$

$2\omega = \omega$ et $\omega 2 not = \omega$.

Chapter 9

The size of one

When I was young, I was intrigued by the following quote of Bourbaki:

Bien entendu il ne faut pas confondre le *terme* mathématique *désigné* (chap. I, § 1, n° 1) par le symbole « 1 » et le mot « *un* » du langage ordinaire. Le terme désigné par « 1 » est égal, en vertu de la définition donnée ci-dessus, au terme désigné par le symbole

$$
\begin{aligned}
(*) \qquad & \tau_Z((\exists u)(\exists \mathrm{U})(u = (\mathrm{U}, \{\emptyset\}, \mathrm{Z}) \text{ and } \mathrm{U} \subset \{\emptyset\} \times \mathrm{Z} \\
& \text{and } (\forall x)((x \in \{\emptyset\}) \implies (\exists y)((x, y) \in \mathrm{U})) \\
& \text{and } (\forall x)(\forall y)(\forall y')(((x, y) \in \mathrm{U} \text{ and } (x, y') \in \mathrm{U}) \implies (y = y')) \\
& \text{and } (\forall y)((y \in \mathrm{Z}) \implies (\exists x)((x, y) \in \mathrm{U})))).
\end{aligned}
$$

Une estimation grossière montre que le terme ainsi *désigné* est un assemblage de plusieurs dizaines de milliers de signes (chacun de ces signes étant l'un des signes τ, $\Box$, $\vee$, $\neg$, $=$, $\in$, $\supset$).

English translation is ([3, p. 158]): The mathematical *term denoted* (Chapter 1, § 1, no. 1) by the symbol "1" is of course not to be confused with the *word* "one" in ordinary language. The term denoted by "1" is equal, by virtue of the definition above, to the term denoted by the symbol (*). As a rough estimate, the term so *denoted* is an assembly of several tens of thousands of signs (each of which is one of τ, $\Box$, $\vee$, $\neg$, $=$, $\in$, $\supset$).

The same expression appears in the English version and in the French one (except that "and" is replaced by "et" in French). By definition, 1 is $\mathrm{Card}(\{\emptyset\}) = \tau_Z(\mathrm{Eq}(\{\emptyset\}, \mathrm{Z}))$.

If 1 is a big object, then how big is 2? or the set N of integers, or the set R of real numbers? If $\mathrm{P}(\mathrm{X}, n)$ is: $\mathrm{X} = (x, y, z)$ and $x^n + y^n = z^n$ and $x \neq 0$ and $y \neq 0$ and $z \neq 0$, what is the size of the formula $(\forall n)(n \in \mathrm{N} \implies (\exists \mathrm{X})(\mathrm{X} \in \mathrm{R}^3 \text{ and } \mathrm{P}(\mathrm{X}, n)))$? If this is an assembly with millions of signs, how big will be a proof of it? If billions of signs are needed, how can one check the proof? I am convinced that it is a bad idea to try to reduce the object under consideration to its basic components (a kind of normal form) and apply low-level theorems to it; Fermat's theorem cannot be proved by exhibiting an assembly that is a proof in the sense of Bourbaki. On the other hand, the estimation of Bourbaki shows that it should be possible to print the whole assembly denoting 1 on an A0-size poster. Alas, the estimation is wrong.

First comments. For simplicity, we shall write Y instead of $\{\emptyset\}$. The formula is hence

$$
\begin{aligned}
(**)\qquad \tau_Z((\exists u)(\exists U)(\quad & u = (U,Y,Z) \\
& \text{and } U \subset Y \times Z \\
& \text{and } (\forall x)((x \in Y) \implies (\exists y)((x,y) \in U)) \\
& \text{and } (\forall x)(\forall y)(\forall y')(((x,y) \in U \text{ and } (x,y') \in U) \implies (y = y')) \\
& \text{and } (\forall y)((y \in Z) \implies (\exists x)((x,y) \in U)) \\
& \text{and } (\forall x)(\forall x')(\forall y)(((x,y) \in U \text{ and } (x',y) \in U) \implies (x = x')))).
\end{aligned}
$$

This is of the form $\tau_Z(W_Z)$ where W_Z is $(\exists u)(\exists U)\,W$, and where W is of the form P1 and P2 and P3 and P4 and P5 and P6. Properties P1 and P2 say that u is a correspondence with source Y, target Z and graph U, properties P3 and P4 say that u is a function, property P5 that u is injective, and P6 that u is injective. In other words, W says that U is the graph of a bijection with source Y and target Z. Our formula is thus $\tau_Z(\mathrm{Eq}(Y,Z))$, as it should be. Note that P6 is missing in (*). In fact, if $(x,y) \in U$, then $x \in Y$, hence $x = \emptyset$, so that P6 is a consequence of other relations. Hence, the term designed by (*) is *equal* but not *identical* to 1. In what follows, we shall discuss the size of (**).

The case of Coq. The expression W has the form

```
exists u : Set,
   exists U : Set,
     u = J (J U emptyset) Z &
     (forall x : Set, inc x U -> inc x (record Y (fun _ : Set => Z))) &
     (forall x : Set, inc x Y -> exists y : Set, inc (J x y) U) &
     (forall x y y' : Set, inc (J x y) U -> inc (J x y') U -> y = y') &
     (forall y : Set, inc y Z -> exists x : Set, inc (J x y) U) &
     (forall x x' y : Set, inc (J x y) U -> inc (J x' y) U -> x = x')
```

Expanding everything but J gives:

```
exists u : Set,
  exists U : Set,
    u = J (J U emptyset) Z &
    (forall x : Set,
     (exists a : U, Ro a = x) ->
     exists  a : record (IM (fun _ : one_point => emptyset))
              (fun _ : Set => Z), Ro a = x) &
    (forall x : Set,
     (exists a : IM (fun _ : one_point => emptyset), Ro a = x) ->
     exists y : Set, exists a : U, Ro a = J x y) &
    (forall x y y' : Set,
     (exists a : U, Ro a = J x y) -> (exists a : U, Ro a = J x y') -> y = y') &
    (forall y : Set,(exists a : Z, Ro a = y) ->
     exists x : Set, exists a : U, Ro a = J x y) &
    (forall x x' y : Set, (exists a : U, Ro a = J x y) ->
     (exists a : U, Ro a = J x' y) -> x = x')
```

Because of our implementation of τ in Coq, the full expansion of 1 is three times as big as this quantity. This gives a total of 500 tokens (we count *forall, exists, Set,* etc, as a single token). In [4], the symbol that looks like $\supset$ has been withdrawn, and a pair is no more a primitive. As explained in the first part of this document, we do not use the same implementation as

Bourbaki. In particular, we use a primitive object *two_points* which is a set with two distinct elements. The expansion of the last line of the previous code is then:

```
(exists a : U,
  Ro a = IM (fun t : two_points =>
        match t with
        | two_points_a => IM (fun _ : one_point => x')
        | two_points_b =>
           IM (fun t0 : two_points =>
              match t0 with
              | two_points_a => emptyset
              | two_points_b => IM (fun _ : one_point => y)
               end)
        end)) -> x = x')
```

If we estimate this as 50 tokens, this gives a total size of one that is less than 2000 tokens. For Bourbaki, a correspondence is an object that satisfies P1 and P2 (namely $u = (U, Y, Z)$ and $U \subset Y \times Z$), and for us, a correspondence is a data structure, that has three fields (source, target and graph) and some properties; the quantity U should be replaced by *graph u*, whose normal form is *let (_, _, graph) := u in graph*. Denoting by u the triple formed of these three objects, by U, Y and Y the components of the triples, then U is a graph, whose domain is a subset of Y and whose range is a subset of Z. This is equivalent to $U \subset Y \times Z$. This is also equivalent to say that the graph of the correspondence is a subset of the product of the source and the range, but becomes much larger if we expand everything that can be expanded. We estimate that the number of tokens, after full expansion, is roughly 16000.

In Version 4, we change out mind and decided that pairs are defined by an axiom. This means that J is unexpandable. This reduces the size of almost everythings. For instance, this is the normal form of P y.

```
chooseT
   (fun x : Set =>
    (exists x0 : Set, exists y0 : Set, y = J x0 y0) ->
    exists y0 : Set, y = J x y0 &
    ((exists x0 : Set, exists y0 : Set, y = J x0 y0) -> False) ->
    x = emptyset) (nonemptyT_intro emptyset)
```

In this new version, 1 becomes small enough in order to study its stucture.

It contains 197 *exists*, 184 *Set*, 117 *Ro*, 96 *J*, 90 *graph*, 64 *fun*, 75 primitives like *IM*, 59 *emptyset*, 42 keywords (like *return*), 28 *chooseT*, 28 *False*, 21 *forall*, 724 identifiers (like x, y), 967 punctuation signs (like ->), 255 operators (like =) (parentheses and commas are not counted). Total: 3100 tokens.

Later on, we removed the axiom of the pair, and the call to the axiom of choice in the definition of the projectors. For instance $P = \bigcup\bigcap$. We show here the normal form of P. The normal form of one with this definition is huge.

```
P= union (intersection x)
= IM (fun i : Union_integral (IM
     (fun z : Zorec (fun a : IM
        (fun i : Union_integral x => Ro
           (let (Union_param, Union_elt) as u return
              (Ro (let (Union_param, _) := u in Union_param)) := i in
                   Union_elt)) =>
```

```
            forall z : Set,
              (exists a0 : x, Ro a0 = z) ->
              exists a0 : z, Ro a0 = Ro a) =>
      let (a, _) := z in Ro a)) =>
Ro (let (Union_param, Union_elt) as u
       return (Ro (let (Union_param, _) := u in Union_param)) :=
       i in
    Union_elt))
```

Preliminary computations. Let's try to count exactly the numbers of signs of one in Bourbaki. Remember that the empty set is the following list of signs:

$$\tau\neg\neg\neg\in\tau\neg\neg\in\square\square\square$$

It is easy to count them: there are 12 signs. The definition of the empty set is $\tau_x((\forall y)(y\notin x))$. If P is a formula we denote by L[P] its length; assume that P depends on z and has α signs and β other signs. Remember that $(\exists z)P(z)$ is $(\tau_z P|z)P$, this is the formula obtained by replacing all z by $\tau_z P$. Thus we have

$$L[(\exists z)P(z))] = (\alpha+1)(\alpha+\beta).$$

Since $(\forall z)P(z)$ is $\neg(\exists z)(\neg(Pz))$, we get

$$L[(\forall z)P(z)] = 1+(\alpha+1)(\alpha+\beta+1).$$

In the case of $(\forall y)(y\notin x)$, we have $\alpha=1$ and $\beta=4$, the length is 11, hence $L[\emptyset]=12$. The following formulas are obvious:

$$L[A \text{ or } B] = 1+L[A]+L[B]$$

$$L[A \text{ and } B] = 4+L[A]+L[B]$$

$$L[A \implies B] = 2+L[A]+L[B]$$

$$L[A \iff B] = 8+2L[A]+2L[B]$$

The set of all x such that $P(z)$ is $\tau_y((\forall z)(z\in y) \iff P)$ hence

$$L[\{z,P(z)\}] = 4\alpha^2+36\alpha+47+(4\alpha+6)\beta.$$

The length of $z=x$ or $z=y$ is $3+x+y+2z$, and since $\{x,y\}$ is the set of all z such that $z=x$ or $z=y$ we get

$$L[\{x,y\}] = 177+14x+14y.$$

Since $\{\emptyset\}=\{\emptyset,\emptyset\}$ we get

$$L[\{\emptyset\}] = 513.$$

Assume that W has length $\alpha+\beta U+\gamma Z+u$. The size of $(\exists U)W$ is

$$(\beta+1)(\alpha+\beta+\gamma Z+u);$$

the size of $(\exists u)(\exists U)W$ is

$$(\beta+1)(\beta+2)(\alpha+\beta+\gamma Z+1);$$

hence the size of one is

$$1+(\beta+1)(\beta+2)(\alpha+\beta+\gamma+1).$$

Size of a triple. We estimate here the length of the expression $u = (\mathrm{U},\mathrm{Y},\mathrm{Z})$, assuming that the triple is the pair of pairs $((\mathrm{U},\mathrm{Y}),\mathrm{Z})$. Since the pair (x,y) is $\{\{x\},\{x,y\}\}$ we have

$$\mathrm{L}[(x,y)] = 5133 + 588x + 196y.$$

$$\mathrm{L}[((x,y),z)] = 3023337 + 345744x + 115248y + 196z$$

$$\mathrm{L}[u = (\mathrm{U},\mathrm{Y},\mathrm{Z})] = 62\,145\,562 + 345\,744\mathrm{U} + 196\mathrm{Z} + u$$

Let α be the constant that appears here, and β the coefficient of U. Since the size of one is at least $\alpha\beta^2$, this formula shows that the size is a bit larger than what Bourbaki claims. Instead of "several tens of thousands", it is at least 10^{18}.

The English version of Bourbaki (as well as the 1956 French edition) has the special symbol that looks like $\supset$ and creates a pair. We use here the notation L', when we count the size of the English one; we have

$$\mathrm{L}'[u = (\mathrm{U},\mathrm{Y},\mathrm{Z})] = u + \mathrm{U} + \mathrm{Z} + 516.$$

Size of a graph. We try to find the size of expression P2, namely $\mathrm{U} \subset \mathrm{Y} \times \mathrm{Z}$. If B is $\mathrm{Y} \times \mathrm{Z}$, this expression is $(\forall z)(z \in \mathrm{A} \implies z \in \mathrm{B})$, its size is $22 + 3\mathrm{A} + 3\mathrm{B}$. The size of "$z = (x,y)$ and $x \in \mathrm{Y}$ and $y \in \mathrm{Z}$" is $z+2x+2y+\mathrm{Y}+\mathrm{Z}+12$. If this is P, then $\mathrm{Y} \times \mathrm{Z}$ is the set of all z so that there exists x such that there exists y such that P. Quantifying $\exists y$ gives $42 + 6x + 3\mathrm{Y} + 3\mathrm{Z} + 3z$, quantifying $\exists x$ gives $336 + 21(\mathrm{Y} + \mathrm{Z} + z)$. Taking the set of all z gives $32807 + 1890(\mathrm{Y} + \mathrm{Z})$.

$$\mathrm{L}'[\mathrm{U} \subset \mathrm{Y} \times \mathrm{Z}] = 98443 + 3\mathrm{U} + 5670(\mathrm{Y} + \mathrm{Z});$$

$$\mathrm{L}'[\mathrm{P2}] = 3\,007\,153 + 3\mathrm{U} + 5670\mathrm{Z}.$$

For the French version, the length of P is

$$589x + 5144 + 197y + z + \mathrm{Y} + \mathrm{Z}.$$

Quantifying over y gives

$$1057518 + 198\mathrm{Y} + 198\mathrm{Z} + 116622x + 198z.$$

Quantifying over x gives

$$136931729220 + 23091354(\mathrm{Y} + \mathrm{Z} + z).$$

Taking the set of all z gives

$$\mathrm{L}[\mathrm{Y} \times \mathrm{Z}] = 12649889797944532895 + 2132842656761388(\mathrm{Y} + \mathrm{Z});$$

$$\mathrm{L}[\mathrm{U} \subset \mathrm{Y} \times \mathrm{Z}] = 37949669393833598707 + 3\mathrm{U} + 6398527970284164(\mathrm{Y} + \mathrm{Z});$$

$$\mathrm{L}[\mathrm{P2}] = 41232114242589374839 + 3\mathrm{U} + 6398527970284164\mathrm{Z}.$$

This number is so big that it is impossible to put in a computer the full expansion of $\mathrm{U} \subset \{\emptyset\} \times \mathrm{Z}$.

Size of a bijection. Let's try to find the size of the expressions P2, P3, P4, P5 and P6. They are similar. We start with $(x,y) \in \mathrm{U}$, the size is

$$5134 + 588x196y + \mathrm{U};\ \text{or}\ 2 + x + y + \mathrm{U}.$$

The size is $(\exists y)((x,y) \in \mathrm{U})$ is

$$1050010 + 197\mathrm{U} + 115836x \text{ or } 6 + 2\mathrm{U} + 2x.$$

The size of $x \in \mathrm{Y} \implies (\exists y)((x,y) \in \mathrm{U})$ is

$$1050013 + 197\mathrm{U} + 115837x + \mathrm{Y} \text{ or } 9 + 2\mathrm{U} + 3x + \mathrm{Y}.$$

The size of $(\forall x)(x \in \mathrm{Y} \implies (\exists y)((x,y) \in \mathrm{U}))$ is

$$135049848139 + 22820086\mathrm{U} + 115838\mathrm{Y} \text{ or } 53 + 8\mathrm{U} + 4\mathrm{Y}.$$

The size of P3 is

$$135109273033 + 22820086\mathrm{U} \text{ or } 2105 + 8\mathrm{U}.$$

The size of $(x,y) \in \mathrm{U}$ and $(x,y') \in \mathrm{U}$ is

$$10272 + 1176x + 196y + 196y' + 2\mathrm{U} \text{ or } 8 + 2x + y + y' + 2\mathrm{U};$$

The size of $(x,y) \in \mathrm{U}$ and $(x,y') \in \mathrm{U} \implies y = y'$ is

$$10275 + 1176x + 197y + 197y' + 2\mathrm{U} \text{ or } 11 + 2x + 2y + 2y' + 2\mathrm{U};$$

The size of $(\forall y')((x,y) \in \mathrm{U}$ and $(x,y') \in \mathrm{U} \implies y = y')$ is

$$2073655 + 396\mathrm{U} + 34848x + 39006y \text{ or } 43 + 6\mathrm{U} + 6x + 6y.$$

The size of $(\forall y)(\forall y')((x,y) \in \mathrm{U}$ and $(x,y') \in \mathrm{U} \implies y = y')$ is

$$82408606635 + 15446772\mathrm{U} + 9082701936x \text{ or } 351 + 42\mathrm{U} + 42x.$$

Finally the size of P4 is

$$830988285585569103965 + 14029842596479736\mathrm{U} \text{ or } 16943 + 1806\mathrm{U}.$$

The size of $(\exists x)((x,y) \in \mathrm{U})$ is

$$3370258 + 589\mathrm{U} + 115444y \text{ or } 6 + 2\mathrm{U} + 2y.$$

The size of $y \in \mathrm{Z} \implies (\exists x)((x,y) \in \mathrm{U})$ is

$$3370261 + 589\mathrm{U} + 115445y + \mathrm{Z} \text{ or } 9 + 2\mathrm{U} + 3y + \mathrm{Z}.$$

Hence the size of P5 is

$$402410930323 + 67997694\mathrm{U} + 115446\mathrm{Z} \text{ or } 53 + 8\mathrm{U} + 4\mathrm{Z}.$$

The size of $(x,y) \in \mathrm{U}$ and $(x',y) \in \mathrm{U} \implies x = x'$ is

$$10275 + 589x + 589x' + 392y + 2\mathrm{U} \text{ or } 11 + 2x + 2x' + 2y + 2\mathrm{U}.$$

The size of $(\forall y)((x,y) \in \mathrm{U}$ and $(x',y) \in \mathrm{U} \implies x = x')$ is

$$4192525 + 786\mathrm{U} + 231477(x + x') \text{ or } 43 + 6\mathrm{U} + 6x + 6x'.$$

The size of $(\forall x')(\forall y)((x,y) \in \mathrm{U}$ and $(x',y) \in \mathrm{U} \implies x = x')$ is

$$1024059366435 + 181941708\mathrm{U} + 53581833006x \text{ or } 351 + 42\mathrm{U} + 42x.$$

Hence the size of P6 is

$$57741990789964425582095 + 9748770215064355956\mathrm{U} \text{ or } 16943 + 1806\mathrm{U}.$$

The size of the expressions P3 to P6 is hence

$$58572979076087514889416 + 98890686411199711 00\mathrm{U} + 115446\mathrm{Z} \text{ or } 36044 + 3628\mathrm{U} + 4\mathrm{Z}.$$

This gives, for the French version $\alpha = 58614211190330166409837\ \beta = 9889068641120316847$ $\gamma = 6398527970399806$, and for the English version: This gives $\alpha = 3033733$, $\beta = 36732$ and $\gamma = 5675$.

Conclusion. The statement, at the start of the chapter, quoted from the 1956 edition of Bourbaki, translated in English in 1968, is grossly wrong since the size is not several thousands, but in fact

$$4\,150\,763\,939\,024\,663.$$

Moreover, the 1970 decision to remove the axiom of the ordered pair had a dramatic effect on the size of one, that increases to

$$573306704401798033758237640367224116154353941968147665929668 9.$$

Chapter 10

Exercises

There are 111 exercises for this chapter (plus 9 concerning direct and inverse limits). We give here the number of lines of the code of all these.

Section 1. 1 (12), 2 (350), 3 (671), 4 (149), 5 (114), 6 (977), 7 (128), 8 (29), 9 (145), 10 (17), 11 (364), 12 (29), 13 (78), 14 (76), 15 (320) 16 (294), 17 (304), 18 (536), 19 (127), 20 (391), 21 (514), 22 (539), 24 (153*).

Section 2. 1 (231), 2 (107), 3 (18), 4 (96), 5 (11), 6 (231), 7 (637).

Section 6. 1 (147), 2 (59), 3 (64), 4 (63).

This is example 1 page 148: *Let* $E = \{\alpha,\beta\}$ *be a set whose elements are distinct. It is easily verified that the subset* $\{(\alpha,\alpha),(\beta,\beta),(\alpha,\beta)\}$ *of* $E \times E$ *is the graph of a well-ordering on* E.

The example is now part of the main text; we nevertheless give a proof.

```
Definition example_worder :=
  union2(doubleton (J TPa TPa) (J TPb TPb)) (singleton (J TPa TPb)).

Lemma example_worder_related : forall x y,
    related  example_worder x y =
    ( (x= TPa & y = TPa) \/  (x= TPb & y = TPb) \/  (x= TPa & y = TPb)).
Proof.
move=> x y; rewrite /gle/related/example_worder.
apply iff_eq.
  move=> h; case (union2_or h)=> h'.
  case (doubleton_or h') => h''; move:(pr1_def h'')(pr2_def h'') => p1 p2; auto.
  awi h';  move:(pr1_def h')(pr2_def h') => p1 p2; auto.
case; [| case]; move=> [r1 r2]; rewrite r1 r2;
    try solve [apply union2_first; fprops].
apply union2_second; fprops.
Qed.

Lemma substrate_example_worder : substrate example_worder = two_points.
Proof.
have gc:is_graph example_worder.
  rewrite /example_worder=> t tu.
  case (union2_or tu);last by aw=>->; fprops.
  move=> h;case  (doubleton_or h) => ->; fprops.
set_extens1 x.
  rewrite  (inc_substrate_rw x gc); move=> h; case h;move=> [y].
  move: (canonical_doubleton_order_pr x y).
```

```
  rewrite /gle/related; move=> ->; intuition; ue.
  move: (canonical_doubleton_order_pr y x).
  rewrite /gle/related; move=> ->; intuition; ue.
rewrite  two_points_pr.
case => h;  have aux: (related example_worder x x)
     by rewrite example_worder_related; auto.
substr_tac.
substr_tac.
Qed.


Lemma example_is_worder: worder example_worder.
Proof.
have gc:is_graph example_worder.
  rewrite /example_worder=> t tu.
  case (union2_or tu);last by aw=>->; fprops.
  move=> h;case  (doubleton_or h) => ->; fprops.
move: (substrate_example_worder) => subs.
have oc: (order example_worder).
  hnf; ee; split=>//.
      move=> y; rewrite subs example_worder_related  two_points_pr.
      intuition.
    move=> x y z; rewrite 3! example_worder_related; intuition.
    elim two_points_distinct; ue.
  move=> x y; rewrite 2! example_worder_related.
  move => h1.
  case; first by move=> []->->.
  case; first by move=> []->->.
  move=> [p1 p2]; move: h1.
  case; first by move=> []->->.
  case; first by move=> []->->.
  by rewrite p1 p2;  move=> []->->.
split=>//.
move=> x xt nex.
case (inc_or_not TPa x)=> hyp.
  exists TPa; red; aw; split=>//.
    move=> y yx; aw; rewrite /gle example_worder_related.
    move: (xt _  yx); rewrite subs two_points_pr; intuition.
move: nex=> [y yx]; exists y; split; aw =>//.
have p: forall t, inc t x -> t = TPb.
  move=> t tx; move: (xt _ tx); rewrite  subs two_points_pr.
  by case =>// ta; rewrite ta in tx; elim hyp.
move => t tx; aw;  rewrite /gle example_worder_related.
by rewrite (p _ yx)(p _ tx); intuition.
Qed.
```

Let $\mathfrak{F}$ be a set of subsets of A, ordered by inclusion, and such that for every totally ordered subset $\mathfrak{G}$ of $\mathfrak{F}$ the union of the sets of $\mathfrak{G}$ belongs to $\mathfrak{F}$. Then $\mathfrak{F}$ is inductive with respect to the relation $\subset$.

```
Lemma inductive_example1: forall A F, sub A (powerset F) ->
  (forall S, (forall x y, inc x S -> inc y S -> sub x y \/ sub y x) ->
    inc (union S) A) ->
  inductive_set (inclusion_suborder A).
Proof.
move=> A F Ap ih X Xs [oX tX]; exists (union X).
```

```
have oi: order (inclusion_suborder A) by fprops.
awi tX =>//.
have uA: inc (union X) A.
  apply ih; move=> x y xX yX; move: (tX _ _ xX yX); rewrite /gge; aw.
  case; intuition.
by split;aw; move=> y yX; aw; ee; [ awi Xs; apply Xs| apply union_sub].
Qed.
```

This is a simple (160 lines) proof of Zermelo's theorem. It says that, for any set E, there is a well-ordering whose support is E; it assumes existence of some function r, such that, whenever $A \subset E$ and A is nonempty, then $r(A) \in A$. The ordrding we construct will be such that $r(A)$ is the least element of A. We define $p(A) = A' = A - \{r(A)\}$. If A is the interval $[x, \rightarrow [$, then A' is the interval $]x, \rightarrow [$. A chain is a subset F of the powerset of E (each element of F is thus a suset of E) such that $E \in F$, $A' \in F$ whenever $A \in F$ and F is stable by intersection. We shall define Ω as the intersection of all chains. This wiil be the set of all intervals $[x, \rightarrow [$. We introduce the property $m(a)$ that says that forall $x \in \Omega$, we have $a \subset x$ or $x \subset a$. If $m(a)$ holds for all elements of Ω, then $\supset$ is a total ordering on Ω. We define $p(x)$ to be the intersection of all elementrs of Ω that contains x. Assume x non-empty, and let y be the least element of x. Then $p(x)$ is the interval $[y, \rightarrow [$. Finally, we define $R(x)$ to be $p(\{\})$. This is the interval $[y, \rightarrow [$. We shall not proove the claims listed above; they are just hints for tezh construction of teh ordering.

```
Lemma Zermelo_bis: forall E, exists r, worder r & substrate r = E.
Proof.
move=> E.
set (p := fun a => complement a (singleton (rep a))).
set (chain:= fun F => sub F (powerset E) & inc E F &
    (forall A, inc A F -> inc (p A) F)
    & (forall A, sub A F -> nonempty A -> inc (intersection A) F)).
set (om:= intersection (Zo (powerset (powerset E)) chain)).
set (m:= fun a => forall x, inc x om -> sub x a \/ sub a x).
set (d:= fun p => intersection (Zo om (fun x => sub p x))).
set (R:= fun x => d (singleton x)).
```

We have $A' \subset A$, and if $A = \emptyset$, then $A' = A$.

```
have pe: p emptyset = emptyset.
  by empty_tac x xe; move: xe; rewrite /p; srw; case; case; case.
have sp: forall a, sub (p a) a by move=> t; rewrite /p; apply sub_complement.
```

We show here that the powerset of E is a chain. This will have as a consequence that Ω is well-defined.

```
have cp:chain (powerset E).
  split; fprops; split; first  by apply powerset_inc; fprops.
  split; first by  move=> A; aw; move=>  AE; apply sub_trans with A.
  move=> A AP [x xA]; move: (AP _ xA); aw.
  move=> xE t tA; exact (xE _ (intersection_forall tA xA)).
```

We show here that Ω is a chain.

```
have co :chain om.
  have aux: (nonempty (Zo (powerset (powerset E)) chain)).
    by exists (powerset E); apply Z_inc; aw; fprops.
  rewrite /om; red; ee.
        by rewrite /om; apply intersection_sub; apply Z_inc; aw; fprops.
      apply intersection_inc=>//; move=>y; rewrite Z_rw/chain; aw; intuition.
    move=> A Ai; apply intersection_inc=>//.
    move=> y yi; move: (yi); rewrite Z_rw; move=> [_ [_[_ [q _]]]]; apply q.
    apply (intersection_forall Ai yi).
  move=> A sAi neA;  apply intersection_inc=> //.
  move=> y yi; move: (yi); rewrite Z_rw; move=> [_ [_[_ [_ q]]]]; apply q=>//.
  move=>t tA; move: (sAi _ tA) => ti.
  by apply (intersection_forall ti yi).
```

We show hete that if C is any chain, then $\Omega \subset C$.

```
move: (co)=> [sop [Eo [po io]]].
have cio: forall x, chain x -> sub om x.
  move=> x xc;  rewrite / om; apply intersection_sub; apply Z_inc =>//.
  by aw; move:xc; rewrite /chain; intuition.
```

Now comes a big part of the proof. We pretend that $m(A)$ holds forall elements of Ω. In fact, the set of elements that satisfy m is a chain, thus has to be Ω. The non-obvious point is to show that $m(A)$ implies $m(A')$. In fact, let T be the set of elements B of Ω such that $B \subset A'$ or $A \subset B$. It contains E and is stable by intersection (if for all i, $A \subset T_i$, then $A \subset \cap T_i$, otherwise there is i such that $T_i \subset A'$ and $\cap T_i \subset A'$). Assume $B \in T$. If $B \subset A'$, then $B' \subset A'$. Since B' is in Ω, we have either $A \subset B'$ or $B' \subset A$. In the first case, $A' \subset B'$. Now we have $B' \subset A \subset B$. Let b' be $r(B')$. If $b' \in A$ then $B \subset A$, then $A = B$, hence $B' \subset A'$. Otherwise $A \subset B'$. In summary, T is a chain, thus must be Ω.

```
have am: om = Zo om m.
  apply extensionality; last by apply Z_sub.
  apply cio; red; ee.
        apply sub_trans with om; [by apply Z_sub| apply sop].
      by apply Z_inc=>//; move=> x xom; left;move: (sop  _ xom); aw.
    move=> A; rewrite Z_rw; move=> [Aom mA].
    apply Z_inc; first by apply (po _ Aom).
    have aux: (sub om (Zo om (fun x=> sub x (p A) \/ sub A x))).
      apply cio; red; ee.
            by apply sub_trans with om; first by apply Z_sub.
          by apply Z_inc=>//; right; move: (sop _ Aom); aw.
        move => B; rewrite Z_rw; move => [Bom ors].
        apply Z_inc; first by apply (po _ Bom).
        case ors => orsi.
          left;  apply sub_trans with B =>//; apply sp.
        case (mA _ (po _ Bom)) => aux; last by auto.
        case (inc_or_not (rep B) A)=> aux2.
          have BA: (sub B A).
            rewrite /p in aux2; move=> t tB.
            case (equal_or_not t (rep B)).
              by move=> ->.
            by move=> trB; apply aux;  rewrite /p; srw;  aw.
          have: (B = A) by apply extensionality.
          by move=> ->; intuition.
        right; move=> t tA; rewrite /p; srw; aw.
```

```
      split; [ by apply orsi| dneg trB; ue].
    move=> B sB neB; apply Z_inc.
      apply io =>//;  apply: sub_trans; [eexact sB| apply Z_sub].
    case (p_or_not_p (exists x, inc x B & sub x (p A))) => aux.
      move: aux=> [x [xB xp]]; left; move=> t ti; apply xp.
      apply (intersection_forall ti xB).
    right; move=> t tA;  apply intersection_inc=>//.
    move=> y yB; move: (sB _ yB); rewrite  Z_rw; move=> [yom ors].
    case ors; last by apply.
    by move => sy; elim aux; ex_tac.
  move=> x xom; case (mA _ xom) => hyp.
    move: (aux _ xom);  rewrite Z_rw; move=> [_ xpA].
    case xpA=>xpB; first by auto.
    have Ax: (A = x) by apply extensionality.
    rewrite Ax; right; apply sp.
  by right;  apply sub_trans with A=>//; apply sp.
move=> A sAZ neA; apply Z_inc.
  apply io =>//; apply sub_trans with (Zo om m) =>//; apply Z_sub.
move=> x xom.
case (p_or_not_p (exists y, inc y A & sub y x)) => hyp.
  move: hyp=> [y [yA yx]]; right; move => t ti.
  apply yx; apply (intersection_forall ti yA).
left; move=> t tx; apply intersection_inc=>//.
move=> y yA; move: (sAZ _ yA); rewrite Z_rw; move=> [yom my].
case (my _ xom);[ by apply | move=> yx; elim hyp; ex_tac; apply Z_sub].
```

Consequence: if A and B are in Ω, then A ⊂ B or B ⊂ A.

```
have st: forall a b, inc a om -> inc b om -> sub a b \/ sub b a.
  move=> a b; rewrite {2} am Z_rw; move=> aom [bom ba]; apply (ba _ aom).
```

We pretend here that $d(p)$ is the least element of Ω that contains p; this amounts to the three following conditions: if $p \subset \mathrm{E}$, then $d(p) \in \Omega$, $p \subset d(p)$, and if $p \subset q$, where $q \in \Omega$, then $d(p) \subset q$.

```
have dpo: forall q, sub q E -> inc (d q) om.
  by move=> q qE; rewrite /d; apply io;[ apply Z_sub| exists E;apply Z_inc].
have pdp: forall q, sub q E -> sub q (d q).
  rewrite /d=> q qE t tq; apply intersection_inc.
    by exists E; apply Z_inc.
  by move => y; rewrite Z_rw; move=>[_]; apply.
have dpq: forall q r, inc r om -> sub q r -> sub q E-> sub (d q) r.
  by rewrite /d=> q r rom qr qE; apply intersection_sub; apply Z_inc.
```

Fix a set $\mathrm{P} \subset \mathrm{E}$ and consider $q = r(d(\mathrm{P}))$. We pretend that if $q \in d(\mathrm{P})$, then $q \in \mathrm{P}$. Recall that $d(\mathrm{P})' = d(\mathrm{P}) - \{q\}$. If q is not in P, we get $\mathrm{P} \subset d(\mathrm{P})'$, and by minimality, $d(\mathrm{P}) = d(\mathrm{P})'$, absurd. *We assume from now on that r is a choice function:* i.e., $r(\mathrm{X}) \in \mathrm{X}$, whenever X is a nonempty subset of E. Thus $q \notin \mathrm{P}$ implies that $d(\mathrm{P}) = \emptyset$, hence that P is empty (since $\mathrm{P} \subset d(\mathrm{P})$).

```
have rdq: forall q, sub q E -> nonempty q -> inc (rep (d q)) q.
  move=> q qE neq; case (inc_or_not (rep (d q)) q)=>// ni.
  have aux: (sub q (p (d q))).
    by rewrite /p=> t tq; srw; aw; split; [ apply (pdp _ qE)| dneg tr; ue].
  move: (dpq _ _ (po _ (dpo _ qE)) aux qE).
  rewrite /p; case  (emptyset_dichot (d q)).
```

```
    move=> dqe; rewrite dqe pe in aux.
    by move: neq=> [t tq]; elim (emptyset_pr (x:= t)); apply aux.
  move=> ned; move: (nonempty_rep ned) => rd dc; move: (dc _ rd); srw.
  aw; intuition.
```

We have seen that if $Q = d(P)$ and P is non-empty, then $r(Q) \in P$. Conversely, if $Q \in \Omega$ and $q = r(Q) \in P$, then $Q = d(P)$. In fact, if $d(P) \subset Q'$, one gets $q \in P \subset d(P) \subset Q'$, absurd; so that $Q' \subset d(P) \subset Q$. The conclusion follows since $q \in d(P)$.

```
have qdp: forall q r, inc r om -> sub q r -> inc (rep r) q -> r = d q.
  move => q r rom qr rq.
  have spE: sub q E.
    by apply sub_trans with r=>//;  apply powerset_sub; apply sop.
  move: (dpq _ _ rom qr spE) => sdqr.
  case (st _ _  (dpo _ spE) (po _ rom)) => ch.
    move: (pdp _ spE) => qdp.
    have:(inc (rep r) (p r)) by apply ch; apply qdp.
    rewrite /p; srw;  aw; intuition.
  apply extensionality =>// t tr.
  case (equal_or_not t (rep r)). by move=> ->; apply (pdp _ spE).
  by move=> tnr; apply ch; rewrite /p; srw; aw; auto.
```

Denote by $R(a)$ the quantity $d(\{x\})$. For $x \in E$, this is in Ω and contains x. The choice function has no choice here: $r(R(x)) = x$. As a consequence R is injective.

```
have Rp: forall x, inc x E ->
    (inc (R x) om & inc x (R x) &  rep (R x) = x).
  rewrite /R=> x xE.
  have p1: sub (singleton x) E by apply sub_singleton.
  move: (pdp _ p1) => p2; ee.
  have nesi: (nonempty (singleton x)) by fprops.
  by move: (rdq _ p1 nesi); aw.
have Ri:forall x y, inc x E -> inc y E -> R x = R y -> x = y.
  move=> x y xE yE; move: (Rp _ xE)(Rp _ yE).
  by move=> [_ [_ p1]][_[_ p2]] p3; rewrite -p3 in p2; rewrite -p1 -p2.
```

We have $R(r(Q)) = Q$ for $Q \in \Omega$, by uniqueness (property Hl above).

```
have Rrq: forall q, inc q om -> nonempty q -> R (rep q) = q.
  move=> a qom neq; rewrite /R; symmetry; apply qdp =>//.
  by move=> t; aw; move=> ->;  apply nonempty_rep.
  fprops.
```

Fix two elements x and y and define $D = \{x, y\}$. We have $r(R(y)) \in D$ since $r(R(y)) = y$. Assume $D \subset R(y)$. This implies $R(y) = d(D)$. Thus $D \subset R(y)$ and $D \subset R(x)$ implies $R(x) = R(y)$. Lemma: if $x \in R(y)$, then $R(x) \subset R(y)$. The assumption says $D \subset R(y)$. Since Ω is totally ordered by inclusion, we have either our conclusion or $R(y) \subset R(x)$. But this implies $D \subset R(x)$ hence $R(x) = R(y)$.

```
have sRR: forall x y, inc x E -> inc y E -> inc x (R y) -> (sub (R x) (R y)).
  move=> x y xE yE xRy.
  move: (Rp _ xE)(Rp _ yE) => [Rom [xRx rR]] [Rom' [yRy rR']].
 case (st _ _ Rom Rom') =>// hyp.
  have p1:(sub (doubleton x y) (R y)).
```

```
    by move=> t td; case (doubleton_or td)=>->.
  have p2: (sub (doubleton x y) (R x)) by apply sub_trans with (R y).
  have p3: (inc (rep (R x)) (doubleton x y)) by rewrite rR; fprops.
  have p4: (inc (rep (R y)) (doubleton x y)) by rewrite rR'; fprops.
  move: (qdp _ _ Rom p2 p3) (qdp _ _ Rom' p1 p4).
  move=> -> ->;  fprops.
```

Since R is injective, the relation $x \leq y$ whenever $\mathrm{R}(y) \subset \mathrm{R}(x)$ is an ordering on E. The previous remarks says: if $x \in \mathrm{R}(y)$ then $y \leq x$.

```
set (r := graph_on (fun x y => (sub (R y) (R x))) E).
have or:order r.
  rewrite /r; apply order_from_rel1.
      by move=> x y z /= xy yz; apply sub_trans with (R y).
    by move=> u v uE vE vu uv; apply Ri=>//; apply extensionality.
  move => u ue; fprops.
have sr: substrate r = E.
  rewrite /r substrate_graph_on //;  move => u ue; fprops.
```

Given any nenempty subset A of E, the quantity $x = r(d(\mathrm{A}))$ is in A. Let $y \in \mathrm{A}$, we have $x \leq y$ since $y \in \mathrm{A} \subset d(\mathrm{A}) = \mathrm{R}(x)$.

```
exists r.
split=>//;split=>//.
move=> x xsr nex.
rewrite sr in xsr.
move: (rdq _ xsr nex) => rdx.
exists (rep (d x)); split.
  by aw;rewrite  sr.
aw; move=> a ax; aw; last by ue.
rewrite /r /gle graph_on_rw1;ee.
apply sRR; try apply xsr=>//.
move: ((pdp _ xsr) _ ax)=> adx.
have ne: (nonempty (d x)) by exists a.
rewrite Rrq //.
by apply dpo.
Qed.
```

The set $\Phi(\mathrm{E},\mathrm{F})$ of mappings of subsets of E into subsets of F is inductive, with respect to the order "v extends u" between u and v (i.e., the opposite of the extension ordering), because there is a common extension on a totally ordered subset.

We give two variants of the Exercise, showing the advantages of a function to be a set. Here is the old version.

```
Lemma inductive_graphs: forall a b,
  inductive_set (opposite_order (extension_order a b)).
Proof.

 ir. cp (extension_is_order a b). red. ir. nin H1. ee. awii H2. awii H0.
  assert (Hd: forall i j, inc i X -> inc j X ->
   agrees_on (intersection2 (source i) (source j)) i j).
  ir. cp (H0 _ H3). cp (H0 _ H4).  bwi H5; bwi H6. ee.
```

```
  cp (H2 _ _ H3 H4). ufi gge H11. red. ee.
  app intersection2sub_first. app intersection2sub_second.
  awii H11.nin H11; ee. ir. app W_extends. inter2tac.
  ir. sy. app W_extends. inter2tac.
  assert (He:forall i, inc i X -> function_prop i (source i) b).
  ir. red.  cp (H0 _ H3). bwi H4. eee.
  cp (extension_covering _ _ He Hd). nin H3. clear H4. nin H3. ee.
  red in H3. ee. assert (sub (source x) a). rw H5.
  red. ir. nin (unionf_exists H7). nin H8. cp (H0 _ H8). awi H10. ee.
  bwi H10. ee. app H11.
  assert (inc x (set_of_sub_functions a b)). bw. eee.
  exists x. red. ee. aw. ir.  aw. eee. red. cp (H0 _ H9). bwi H10. ee. am.
  am.  cp (H4 _ H9). red in H13. ee.
  red. ir. cp (in_graph_W H10 H16). rwi H17 H16.
  cp (inc_pr1graph_source H10 H16). rw H17. wr (H15 _ H18).  app W_pr3.
  app H13. rw H6. rw H12. fprops.
  app opposite_is_order.
Qed.
```

Here is the new version.

```
Lemma inductive_graphs: forall a b,
  inductive_set (opposite_order (extension_order a b)).
Proof.
move => a b.
move: (extension_is_order a b)=> or.
move => X;aw; move=> Xsr [oX ].
have Xs:sub X (substrate (extension_order a b)) by aw.
have oo: order (opposite_order (extension_order a b))
   by  apply opposite_is_order =>//.
aw => aux.
have aag: forall i j, inc i X -> inc j X ->
   agrees_on (intersection2 (source i) (source j)) i j.
  move=> i j iX jX; move: (Xsr _ iX) (Xsr _ jX); bw.
  move=> [fi [si ti]][fj [sj tj]].
  red; ee; [ apply intersection2sub_first | apply intersection2sub_second | ].
  move=> c; aw; move=> [csi csj].
  move: (aux _ _ iX jX); rewrite /gge; aw.
  case;move=> [_ [_ ext]]; [ | symmetry]; apply W_extends =>//.
have afp:forall i, inc i X -> function_prop i (source i) b.
  move=> i iX; move: (Xsr _ iX); bw; red; ee.
move: (extension_covering afp aag) => [[g [[fg [sg tg]]] gp] _].
have ssg: (sub (source g) a).
  rewrite sg => t; aw; srw; move =>[y [yX tsy]].
  by move: (Xsr _ yX); bw; move=> [_ [ssy _]]; apply ssy.
have gsab: (inc g (set_of_sub_functions a b)) by bw; ee.
exists g; red; ee; aw.
move=> y yX; move: (Xsr _ yX); bw; move=> [fy [ssy ty]].
aw; ee.
red; ee; last by rewrite tg ty.
move=> w wg; move: (in_graph_W fy wg) => pg.
rewrite pg in wg; move: (inc_pr1graph_source fy  wg) => ps.
move: (gp _ yX); rewrite /agrees_on; move => [w1 [_ w3]].
by rewrite pg -(w3 _ ps); apply W_pr3 =>//; apply w1.
Qed.
```

The objective here is to give a proof of the Cantor-Bernstein theorem that says that if there is an injection from A into B and an injection from B into A, there re is a bijection. We start with a lemma. Assume that $g : \mathfrak{P}(E) \to \mathfrak{P}(E)$ is an increasing function. Then g has a fixed-point. Indeed, let A the the set of all x such that $x \subset g(x)$, and U the union of A. If $x \in A$ then $x \subset U \subset g(U)$). This gives $U \subset g(U)$. It implies $g(U) \in S$. But this is equivalent to $g(U) \subset \cup S$, so that $U = g(U)$.

```
Lemma Cantor_Bernstein_aux: forall E (g: Set -> Set),
  (forall x, sub x E -> sub (g x) E) ->
  (forall x y, sub x y -> sub y E -> sub (g x) (g y)) ->
  (exists m, sub m E & g m = m).
Proof. ir. set (A := Zo (powerset E)(fun x=> sub x (g x))).
  assert (sub (union A) E). app sub_union. uf A. ir. Ztac. app powerset_sub.
  exists (union A). split. am.
  assert (forall x, inc x A -> sub x (g (union A))). ir. ufi A H2. Ztac.
  apply sub_trans with (g x). am. app H0. app union_sub.
  assert (sub (union A) (g (union A))). red. ir. nin (union_exists H3).
  ee. cp (H2 _ H5). app H6. app extensionality. red. ir.
  apply union_inc with (g (union A)). am.
  set (T:=union A) in *.  assert (sub (g T) E). app H. cp (H0 _ _ H3 H5).
  uf A. Ztac. app powerset_inc.
Qed.
```

Consider two injections $f : E \to F$ and $g : F \to E$. For $X \subset E$, consider $h(X) = E - g\langle F - f\langle X\rangle\rangle$. Since taking the complementary of a set by a function is decreasing and the composition of two decreasing functions is increasing, this function is increasing. If has a fixed point M. We have $E - M = g\langle F - f\langle M\rangle\rangle$.

```
Lemma Cantor_Bernstein: forall f g,
  injective f -> injective g -> source f = target g -> source g = target f ->
  equipotent (source f)(source g).
Proof. ir. set (E:= source f) in *. set (F:=source g) in *.
  set (h:= fun x => complement E (image_by_fun g
    (complement F (image_by_fun f x)))).
  assert (Ha:is_function f). nin H; am.
  assert (forall x, sub x E -> sub (h x) E). ir.  uf h. app sub_complement.
  assert (forall x y, sub x y -> sub y E -> sub (h x) (h y)). ir. uf h.
  red. ir. srwi H6. nin H6. srw. split. am. red. ir. elim H7. awi H8. nin H8.
  srwi H8. ee. aw. exists x1. srw. ee. am. red. ir. elim H10.
  awi H11. nin H11. aw. exists  x2. ee. app H4. am. am.
  apply sub_trans with y. am. am. am. nin H0; am. app sub_complement.
  nin H0; am. app sub_complement.
  nin (Cantor_Bernstein_aux _ H3 H4). nin H5. ufi h H6.
  set (T:= image_by_fun g (complement F (image_by_fun f x))) in *.
  assert (sub T E). uf T. red. ir. awi H7. nin H7. nin H7. rw H8. rw H1.
  app inc_W_target. nin H0; am. srwi H7. nin H7. am. nin H0; am.
  app sub_complement. cp (double_complement H7). rwi H6 H8.
```

Let $T = g\langle F - f\langle M\rangle\rangle$. This is the complementary of M. Every element in T is of the form $g(y)$ for some y not of the form $f(x)$ for $x \in T$. We use the choice function in order to get a function f_2. Note that the choice is limited, this y is unique by injectivity of g. We consider the function f_1 whose value is f on M and f_2 on T.

```
  set (f1:= fun a=> choose(fun y=> inc y F & ~ inc y (image_by_fun f x) &
    a = W y g)).
  assert (forall a, inc a T -> (inc (f1 a) F & ~ inc (f1 a) (image_by_fun f x) &
   a = W (f1 a) g)).
  ir. uf f1. app choose_pr. app H9.
  set (f2:= fun a=> Yo (inc a T) (f1 a) (W a f)).
```

This function is injective. Assume $f_1(x) = f_1(y)$. If one of x, y is in M and the other one is in T, one image is in f(M), and the other is in the complement, absurd. If both elements are in M, we use injectivity of f. Otherwise $f_1(x) = x'$ where $x = g(x')$ and $f_1(y) = y'$ where $y = g(y')$. We use injectivity of g. This function is clearly surjective, thus is a bijection E → F.

```
  assert (transf_axioms f2 E F). red. ir. uf f2. nin (inc_or_not c T).
  ir. rw Y_if_rw. cp (H10 _ H12). ee. am. am. ir. rww Y_if_not_rw.  rw H2.
  app inc_W_target.
  exists (BL f2 E F). ee. split.  app injective_af_function. ir.
  ufi f2 H14. nin (inc_or_not u T); [ rwii Y_if_rw H14 | rwii Y_if_not_rw H14] ;
  nin (inc_or_not v T). cp (H10 _ H15). cp (H10 _ H16). ee. rwii Y_if_rw H14.
  rw H22; rw H20; rww H14. cp (H10 _ H15). ee.  rwii Y_if_not_rw H14.
  elim H18. aw. exists v. rw H14. split. wr H6. srw. split; am. tv.
  rwii Y_if_rw H14.  cp (H10 _ H16). ee. elim H18. aw. exists u. split.
  wr H6. srw. split. am. am. sy; am. rwii Y_if_not_rw H14.
  nin H. app H17.
  app (surjective_af_function). ir. nin (inc_or_not y (image_by_fun f x)).
  awi H13. nin H13. nin H13. exists x0. split. app H5. uf f2.
  rw Y_if_not_rw. am. wri H6 H13. srwi  H13.  nin H13; am. am. am.
  exists (W y g). split.  rw H1. app inc_W_target. nin H0;am.
  uf f2. assert (inc (W y g) T). uf T.  aw. exists y. split. srw. split.
  am. am. tv. nin H0;am. app sub_complement. rww Y_if_rw. cp (H10 _ H14).
  ee. nin H0. app H18. tv. tv.
Qed.
```

We give now a variant of the theorem. In fact, we show that if v an injection A → B and B ⊂ A, then A and B are equipotent. Consider two functions f and g, let A be the source of f and B the image of g. We have the canonical decomposition of g as a function $\overline{g}$: F → B, the canonical injection i of B into A. By assumption $\overline{g}$ is bijective. The composition of $\overline{g} \circ f$ is an injection A → B, hence there is a bijection v : A → B, and $\bar{g}^{-1} \circ v$ is a bijection E → F. We leave the details to the reader.

We consider the set A of subsets of E invariant by F, the set B which is the complementary of F in E, the set C of all elements of A that contain B, and the intersection D of C. This intersection is well-defined, since E ∈ C. It is the smallest element of C.

```
Lemma Cantor_Bernstein_aux2: forall (E F: Set) (f:Set -> Set) ,
  sub F E ->
  (forall x, inc x E -> inc (f x) F) ->
  (forall x y, inc x E -> inc y E -> f x = f y  -> x = y) ->
  equipotent E F.
Proof. ir.
  set (A := Zo (powerset E)(fun x=> forall y, inc y x -> inc (f y) x)).
  set (B:= complement E F).
  set (C:= Zo A (fun x=> sub B x)).
  set (D:= intersection C).
  assert (sub A (powerset E)). uf A. app Z_sub.
```

```
  assert (inc E C). uf C. Ztac. uf A.
  Ztac. app powerset_inc. fprops. uf B.  app sub_complement.
  assert (forall x, inc x C -> sub D x). ir. red. ir. ufi D H5.
  ap (intersection_forall H5 H4).
  assert (inc D A). uf A. Ztac. app powerset_inc. red. uf D. ir.
  cp (intersection_forall H4 H3). am. ir. uf D. app intersection_inc.
  exists E. am. ir. assert (inc y y0). ufi D H4.
  ap (intersection_forall H4 H5). ufi C H5.  Ztac. ufi A H6. clear H5.
  ufi A H7. Ztac. app H9.
  assert (inc D C). uf C. Ztac. uf D. red. ir. app intersection_inc. exists E.
  am. ir. ufi C H6. Ztac. app H8.
  assert (sub B D). ufi C H5. Ztac. am.
  assert (complement E B=F). uf B. app double_complement.
```

We consider the function g whose value is f on D, the identity elsewhere. The relation $B \subset D$ implies $E - D \subset F$, so that g has value in F. Since $f(D) \subset D$ and f is injective, the function g is clearly injective.

```
  set (g:=fun y => Yo (inc y D) (f y) y).
  assert (transf_axioms g E F). red. ir. uf g. nin (inc_or_not c D).
  rww Y_if_rw. app H0. rww Y_if_not_rw. wr H8. srw. split. am.
  red. ir. elim H10. app H7.
  exists (BL g E F). ee. red. split. app injective_af_function. ir.
  ufi g H12. nin (inc_or_not u D); [rwi Y_if_rw H12 | rwi Y_if_not_rw H12];
  nin (inc_or_not v D). rwi Y_if_rw H12. app H1. am. rwi Y_if_not_rw H12.
  elim H14. wr H12. ufi A H5. Ztac. app H16. am. am. am.
  rwi Y_if_rw H12. elim H13. rw H12. ufi A H5. Ztac. app H16. am.
  rwi Y_if_not_rw H12. am. am. am. am.
```

Let's show surjectivity. We must show that if $y \in F$ and $y \in D$, then $y = f(x)$ for some x in D. Let $T = D - \{y\}$. This contains B (since $B \subset D$ and $y \notin B$). It is not in A (otherwise, it would be in C, hence contain D, which is impossible since $y \in D$ and $y \notin T$). This means that T is not invariant by f. Consider $x \in T$ such that $f(x) \notin T$. We have $f(x) = y$.

```
  app surjective_af_function. ir. uf g. nin (inc_or_not y D).
  assert (Ha:sub D E). ufi A H5. Ztac. app powerset_sub.
  set (T:= complement D (singleton y)).
  assert (Hb: sub B T). red. ir. uf T. srw.  split. app H7. ufi B H12.
  srwi H12. nin H12. red. ir. elim H13. rw (singleton_eq H14). am.
  assert (~ (inc T A)). red. ir. assert (inc T C). uf C. Ztac.
  cp (H4 _ H13). ufi T H14. cp (H14 _ H11).
  srwi H15. nin H15. elim H16. fprops.
  assert (inc T (powerset E)). app powerset_inc. apply sub_trans with D. uf T.
  app sub_complement. am. assert (exists x, inc x T & ~ (inc (f x) T)).
  app exists_proof. red. ir. elim H12. uf A. Ztac. ir.
  nin (inc_or_not (f y0) T). am. elim (H14 y0). split;am. nin H14.
  ufi T H14. nin H14. srwi H14. ee. exists x. split. app Ha. rww Y_if_rw.
  srwi H15. nin (equal_or_not y (f x)). am. elim H15. split.
  ufi A H5. Ztac. app H19. red. ir. elim H17. sy; app singleton_eq.
  exists y. ee. app H. rww Y_if_not_rw. tv. tv.
Qed.
```

10.1 Section 1

1. *Let* E *be an ordered set in which there exists at least one pair of distinct comparable elements. Show that if* R$\{x, y\}$ *denotes the relation "*$x \in$ E *and* $y \in$ E *and* $x < y$*", then* R *satisfies the first two conditions of no. 1 but not the third.*

```
Lemma Exercise1_1: forall r, let E:= substrate r in
  let s := fun x y => (inc x E & inc y E & glt r x y) in
    order r ->  (exists x, exists y, x <> y & related r x y)
    -> (transitive_r s &  antisymmetric_r s & ~(reflexive_rr s)).
Proof.
move=> r E s or [x [y [xy rxy]]]; ee.
- move => a b c [aE [bE ab]] [_ [cE bc]] ;red; ee; order_tac.
- move => a b  [aE [bE [ab nab]]] [_ [_ ba]]; order_tac.
- have sxy : s x y by red; rewrite /E;ee; try substr_tac; split.
  move => ref; move: (ref _  _ sxy) => [[_ [_ [_ xx]]] _].
  by elim xx.
```

---

2. *(a) Let* E *be a preordered set and let* S$\{x, y\}$ *be an equivalence relation on* E*. Let* R$\{$X, Y$\}$ *denote the relation "*X $\in$ E/S *and* Y $\in$ E/S *and for each each* $x \in$ X *there exists* $y \in$ Y *such that* $x \leq y$*". Show that* R *is a preorder relation on* E/S*, called the* quotient *by* S *of the relation* $x \leq y$*. The quotient* E/S*, endowed with this preorder relation, is called (by abuse of language; cf Chapter IV, § 2, no. 6) the quotient by* S *of the preordered set* E*.*

In what follows, we shall denote by $\leq_E$ the given preorder, and by $x \leq_E y$ the associated preorder relation; in the same way, $\leq_Q$ is the quotient preorder and $x \leq_Q y$ its associated relation. In other words, $x \leq_Q y$ is the relation R$\{x, y\}$ defined in the text and $\leq_Q$ is the graph of this relation on E/S.

```
Definition quotient_order_r r s X Y :=
  inc X (quotient s) & inc Y (quotient s) &
  forall x, inc x X -> exists y, inc y Y & gle r x y.
Definition quotient_order r s := graph_on (quotient_order_r r s) (quotient s).
```

First part: we show that $x \leq_Q y$ is a preorder relation.

```
Lemma Exercise1_2a: forall r s,
  is_equivalence s -> preorder r -> substrate s = substrate r ->
  preorder_r (quotient_order_r r s).
Proof.
move=> r s es [[_ rr] [gr tr]] sssr;red; split.
  move=> a b c [aq [bq abp]] [_ [cq bcp]];red; ee.
  move => x xa; move: (abp _ xa) => [y [yb xy]].
  move: (bcp _ yb) => [z [zc yz]]; ex_tac; apply (tr _ _ _ xy yz).
move=> a b  [aq [bq abp]].
by split; red; ee; move=> x xs; ex_tac; apply rr; rewrite -sssr;
  apply (inc_in_quotient_substrate es xs).
Qed.
```

We now show that $\leq_Q$ is a preorder on E/S, associated to the relation $x \leq_Q y$.

```
Lemma quotient_order_pr: forall r s x y,
  related (quotient_order r s) x y = quotient_order_r r s x y.
Proof.
move=> r s x y; rewrite /quotient_order /graph_on /related Z_rw; aw.
apply iff_eq; [ | rewrite / quotient_order_r ]; ee.
Qed.

Lemma quotient_is_preorder: forall r s,
  is_equivalence s -> preorder r -> substrate s = substrate r ->
  preorder (quotient_order r s).
Proof.
move=> r s es pr srsr; rewrite/ quotient_order; apply preorder_from_rel.
by apply Exercise1_2a.
ir. uf quotient_order. app preorder_from_rel. app Exercise2_2a.
Qed.

Lemma substrate_quotient_order: forall r s,
  is_equivalence s -> preorder r -> substrate s = substrate r ->
  substrate (quotient_order r s) = quotient s.
Proof.
move=> r s es pr sssr.
move: (quotient_is_preorder es pr sssr)=> pq.
set_extens1 x; rewrite preorder_reflexivity // quotient_order_pr.
  by move=> [xs _].
move=> xs; red; ee. move=> y yx; ex_tac.
rewrite  /gle - preorder_reflexivity // -sssr.
apply (inc_in_quotient_substrate es yx xs).
Qed.
```

(b) Let ϕ *be the canonical mapping of* E *onto* E/S*. Show that if* g *is a mapping of the preordered quotient set* E/S *into a preordered set* F *such that* $g \circ \phi$ *is an increasing mapping, then* g *is an increasing mapping. The mapping* ϕ *is increasing if and only if* S *satisfies the following condition*

(C) the relations $x \leq y$ *and* $x \equiv x' \pmod{S}$ *in* E *imply that there exists* $y' \in E$ *such that* $y \equiv y' \pmod{S}$ *and* $x' \leq y'$*.*

Every equivalence relation S *which is* compatible *(in* x*) with the preorder relation* $x \leq y$ *(Chapter II, § 6, no. 3) is a fortiori weakly compatible (in* x *and* y*) with this relation.*

We consider now a function g such that $g \circ \phi$ is an increasing mapping, and show that g is increasing. Assume $X \leq_Q Y$, and that we want to show $g(X) \leq_F g(Y)$. Since X is a class, there is $x \in X$ hence there is $y \in Y$ such that $x \leq_E y$. Thus $g(\phi(x)) \leq_F g(\phi(y))$. It remains to show that $X = \phi(x)$ and $Y = \phi(y)$.

```
Definition increasing_pre f r r':=
  is_function f & preorder r & preorder r' & substrate r = source f
  & substrate r' = target f  &
  forall x y,  gle r x y  -> gle r' (W x f) (W y f).

Lemma Exercise1_2b1: forall r s g r',
  is_equivalence s -> substrate s = substrate r ->
  is_function g -> quotient s = source g ->
  increasing_pre (compose g (canon_proj s)) r r' ->
  increasing_pre g (quotient_order r s) r'.
Proof.
move=> r s g r' es sssr fg qs [fc [pr [pr' [sq [sr' ale]]]]].
```

```
red; ee.
      apply quotient_is_preorder =>//.
    rewrite substrate_quotient_order//.
  move: sr'; aw.
have cc: (composable g (canon_proj s)) by  red; ee; symmetry; aw.
move => x y; rewrite /gle quotient_order_pr//; move=> [xq [yq h]].
move: (h _ (inc_rep_itself es xq)) => [y0 [y0y aux]].
have h1: (inc y0 (substrate s)) by rewrite sssr; order_tac.
have h2: (inc (rep x) (substrate s)) by rewrite sssr; order_tac.
have h0: source (canon_proj s) = substrate s by aw.
move: (ale _ _ aux); aw; try ue.
move: (related_rep_in_class es yq y0y).
rewrite related_rw//; move => [_ [_]] <-.
rewrite class_rep// class_rep//.
Qed.
```

The converse is true if ϕ is increasing, this is equivalent to the following condition.

```
Definition weak_order_compatibility r s:=
  preorder r & is_equivalence s &  substrate s = substrate r &
  forall x y x', gle r x y -> related s x x' -> exists y',
    (related s y y'  & gle r x' y').
```

We show that strong compatibility implies weak compatibility (assume $x \leq y$ and $x \equiv x'$ (mod S) in E, we can take $y' = y$ in $y \equiv y'$ (mod S) and $x' \leq y'$ since y is in the substrate of S).

```
Lemma strong_order_compatibility: forall r s,
  preorder r -> is_equivalence s -> substrate s = substrate r ->
  (forall x x' y,  gle r x y -> related s x x' -> gle r x' y) ->
  weak_order_compatibility r s.
Proof.
move=> r s pr es ss aux; red; ee.
move=> x y x' xy sxx'; exists y; split.
  apply reflexivity_e => //. red in xy;rewrite ss; substr_tac.
apply aux with x=> //.
Qed.
```

Let's show that the canonical projection is increasing.

```
Lemma compatibility_proj_increasing: forall r s,
  is_equivalence s -> preorder r -> substrate s = substrate r ->
  weak_order_compatibility r s =
  increasing_pre (canon_proj s) r  (quotient_order r s).
Proof.
move=> r s es pr sr.
move: (quotient_is_preorder es pr sr) => pq.
rewrite /increasing_pre  -sr substrate_quotient_order //; aw.
have gs: is_graph s by fprops.
apply iff_eq.
  move=> [_ [_ [_ woc]]]; ee.
  move=> x y lexy.
  have xss: inc x (substrate s) by  rewrite sr; order_tac.
  have yss: inc y (substrate s) by  rewrite sr; order_tac.
  rewrite /gle quotient_order_pr /quotient_order_r; aw; ee.
  move=> z; bw => xz; move: (woc _ _ _ lexy xz) => [y' [syy' rzy']].
```

```
  exists y'; bw;ee.
move=>[fc [_ [_ [_ [_ ci]]]]]; red; ee.
move=> x y x' rxy sxx'. move: (ci _ _ rxy).
rewrite /gle quotient_order_pr /quotient_order_r.
have xss: inc x (substrate s) by  rewrite sr; order_tac.
have yss: inc y (substrate s) by  rewrite sr; order_tac.
move=> [_ [_]]; aw=> h.
rewrite -inc_class // in sxx'; move: (h _ sxx') => [y'].
by rewrite inc_class //; exists y' .
Qed.
```

(c) Let E_1 *and* E_2 *be two preordered sets. Show that if* S_1 *is the equivalence relation* $\mathrm{pr}_1 z = \mathrm{pr}_1 z'$ *on* $\mathrm{E}_1 \times \mathrm{E}_2$*, then* S_1 *is weakly compatible in* z *and* t *with the product preorder relation* $z \leq t$ *on* $\mathrm{E}_1 \times \mathrm{E}_2$ *(but is not usually compatible with this relation in* z *or* t *separately); moreover if* ϕ_1 *is the canonical mapping of* $\mathrm{E}_1 \times \mathrm{E}_2$ *onto* $(\mathrm{E}_1 \times \mathrm{E}_2)/\mathrm{S}_1$*, and if* $\mathrm{pr}_1 = f_1 \circ \phi_1$ *is the canonical decomposition of* pr_1 *with respect to the equivalence relation* S_1*, then* f_1 *is an isomorphism of* $(\mathrm{E}_1 \times \mathrm{E}_2)/\mathrm{S}_1$ *into* E_1*.*

We show here weak compatibility of the product of two preorder relations on E_1 and E_2 and the equivalence S_1 in $\mathrm{E}_1 \times \mathrm{E}_2$ defined by $\mathrm{pr}_1 z = \mathrm{pr}_1 z'$.

```
Lemma exercise1_2c1: forall r1 r2,
  preorder r1 -> preorder r2 ->
  weak_order_compatibility (product2_order r1 r2)
  (first_proj_eq (substrate r1) (substrate r2)).
Proof. ir. red. ee. app product2_order_preorder.
  ap first_proj_equivalence. rw first_proj_substrate.
  rww product2_order_preorder_substrate.  ir.
  rwi product2_order_pr H1. ee. rwi first_proj_eq_related H2.
  ee. exists (J (P y) (Q x')).
  rw product2_order_pr. rw first_proj_eq_related. awi H2. awi H6.
  aw. awi H3. eee.  wrr preorder_reflexivity.
Qed.
```

If S_1 is compatible with $\leq$, then if $x \leq x$ and if x and y are related by S_1, we have $x \leq y$ or $y \leq x$ (depending on whether S_1 is compatible with the first or second argument). Consider the case of the product order and the first projection. If we take $x = (a, b)$ and $y = (a, c)$, then $x \leq y$ or $y \leq x$, hence $b \leq c$ or $c \leq b$. This means that all elements in E_2 are related. Note that if E_1 is empty, the product is empty and the condition is vacuous.

```
Lemma exercise1_2c2: forall r1 r2,
  let compatibility r s :=
   (forall x x' y,  gle r x y -> related s x x' -> gle r x' y) in
  preorder r1 -> preorder r2 -> nonempty (substrate r1) ->
  compatibility (product2_order r1 r2)
  (first_proj_eq (substrate r1) (substrate r2)) ->
  r2 = coarse (substrate r2).
Proof. ir. assert (preorder (product2_order r1 r2)).
  app product2_order_preorder. set_extens. uf coarse.
  assert (is_pair x). nin H0. nin H0. app H0. app product_inc.
  substr_tac. substr_tac. ufi coarse H4. awi H4. ee. nin H1.
  set (x1:= J y (Q x)). set (x2:= J y (P x)).
  assert (inc x1 (product (substrate r1) (substrate r2))). uf x1. fprops.
  assert (inc x2 (product (substrate r1) (substrate r2))). uf x2. fprops.
  assert (gle (product2_order r1 r2) x1 x1). wrr preorder_reflexivity.
```

```
  rww product2_order_preorder_substrate.
  set (s:=first_proj_eq (substrate r1) (substrate r2)).
  assert (related s x1 x2). uf s. rw first_proj_eq_related.
  split. am. split. am. uf x1. uf x2. aw.
  fold s in H2. cp (H2 _ _ _ H9 H10).
  rwi product2_order_pr H11.  eee. ufi x1 H14;  ufi x2 H14.
  red in H14. awi H14. red in H14.   awi H14. am. am.
Qed.
```

Same proof, with definitions of x_1 and x_2 exchanged.

```
Lemma exercise1_2c3: forall r1 r2,
  let compatibility r s :=
   (forall x y y',  gle r x y -> related s y y' -> gle r x y') in
  preorder r1 -> preorder r2 -> nonempty (substrate r1) ->
  compatibility (product2_order r1 r2)
  (first_proj_eq (substrate r1) (substrate r2)) ->
  r2 = coarse (substrate r2).
Proof. ... Qed.
```

We show that $\mathrm{pr}_1 = f_1 \circ \phi_1$ implies that f_1 is an isomorphism; we first introduce the definition of a preorder isomorphism. If E_2 is empty, then pr_1 is in general not surjective, thus f_1 is not surjective; this explains why we add the condition $\mathrm{E}_2 \neq \emptyset$. If $f_1(x) = f_1(y)$, where x and y are the classes of x' and y', then $\mathrm{pr}_1 x' = \mathrm{pr}_1 y'$, hence $x' \equiv y'$ and $x = y$, so that f_1 is injective. With the same notations, $f_1(x) \leq f_1(y)$ if and only if $\mathrm{pr}_1 x' \leq \mathrm{pr}_1 y'$.

```
Definition preorder_isomorphism f r r' :=
  (order r) & (order r') &
  (bijective f) & (substrate r = source f) &  (substrate r' = target f) &
  (forall x y, inc x (source f) -> inc y (source f) ->
    gle r x y = gle r' (W x f) (W y f)).

Lemma exercise1_2c4: forall r1 r2 f,
  let s := (first_proj_eq (substrate r1) (substrate r2)) in
    let r:= product2_order r1 r2 in
  is_function f -> source f = quotient s -> target f = (substrate r1) ->
  preorder r1 -> preorder r2 -> nonempty (substrate r2) ->
  compose f (canon_proj s)=(first_proj (product (substrate r1) (substrate r2)))
  ->  preorder_isomorphism f (quotient_order r s) r1.
Proof. ir. set (E1:= substrate r1) in *. set (E2:= substrate r2) in *.
  assert (Ha:substrate r = product E1 E2). uf r.
  rww product2_order_preorder_substrate.
  assert(Hb:substrate s =  product E1 E2). uf s.
  rww first_proj_substrate.
  assert (Hc:is_equivalence  s). uf s; app first_proj_equivalence.
  assert (composable f (canon_proj s)). red. eee; aw; app function_canon_proj.
  assert (Hd:preorder r).  uf r. app product2_order_preorder.
  assert (source (canon_proj s)= product E1 E2). aw.
  red. ee. app quotient_is_preorder. ue.  am. red. ee. red. ee. am. ir.
  assert (inc x (quotient s)).  wrr H0. cp (canon_proj_show_surjective Hc H11).
  assert (inc y (quotient s)).  wrr H0. cp (canon_proj_show_surjective Hc H13).
  wr (related_rep_rep Hc H11 H13). uf s.
  rw first_proj_eq_related. eee.
  wri H12 H10. wri H14 H10. assert (inc (rep x) (source (canon_proj s))).
  rww H7. ue. wri (compose_W H6 H15) H10.
```

```
  assert (inc (rep y) (source (canon_proj s))). rww H7. ue.
  wri (compose_W H6 H16) H10. rwi H5 H10.
  rwii first_proj_W H10. rwii first_proj_W H10. fprops. ue. fprops. ue.
  app surjective_pr6. rw H1. ir.  nin H4. exists (W (J y y0) (canon_proj s)).
  assert (inc (J y y0) (source (canon_proj s))). ue.
  ee. assert (source f = target (canon_proj s)). aw.
  rw H10. app inc_W_target. app canon_proj_function. wr (compose_W H6 H9).
  rw H5. rw first_proj_W. aw. fprops. fprops. sy; rww substrate_quotient_order.
  rww Ha. sy. am. ir. set (u:= W x f). set (v:= W y f).
  assert (inc x (quotient s)).  wrr H0. cp (canon_proj_show_surjective Hc H10).
  assert (inc y (quotient s)).  wrr H0. cp (canon_proj_show_surjective Hc H12).
  assert (u = W x f). tv. wri H11 H14. assert (v = W y f). tv. wri H13 H15.
  assert (inc (rep x) (source (canon_proj s))). rww H7. ue.
  wri (compose_W H6 H16) H14.
  assert (inc (rep y) (source (canon_proj s))). rww H7. ue.
  wri (compose_W H6 H17) H15. rwi H5 H14. rwi H5 H15.
  rwi first_proj_W H14.  rwi first_proj_W H15.  rw H14; rw H15.
  rw quotient_order_pr. uf quotient_order_r. app iff_eq. ir. ee.
  assert (inc (rep x) x). app (inc_rep_itself Hc H10). nin (H20 _ H21).
  nin H22. ufi r H23. rwi product2_order_pr H23. ee.
  cp (is_class_pr Hc H22 H19). assert (inc x0 (substrate s)).
  app (inc_in_quotient_substrate  Hc H22 H19). cp (related_rep_class Hc H28).
  wri H27 H29. ufi s H29. rwi first_proj_eq_related H29. eee.
  ir. eee. ir.
  assert(inc (J v (Q x0)) (product (substrate r1) (substrate r2))). aw. ee.
  fprops. rwi H7 H17. awi H17. eee.
  cp (inc_in_quotient_substrate Hc H19 H10). rwi Hb H20; awi H20.  eee.
  assert (inc x0 (product (substrate r1) (substrate r2))).
  cp (inc_in_quotient_substrate Hc H19 H10).  rwi Hb H21. am.
  exists (J v (Q x0)). ee. wr (class_rep Hc H12). bw. uf s.
  rw first_proj_eq_related. ee. wr H7. am. am. sy. aw.
  uf r. rw  product2_order_pr. red. eee. aw. rw H15.
  assert (inc (rep x) x). app (inc_rep_itself Hc H10).
  cp (is_class_pr Hc H19 H10). assert (inc x0 (substrate s)).
  app (inc_in_quotient_substrate  Hc H19 H10). cp (related_rep_class Hc H24).
  wri H23 H25. ufi s H25. rwi first_proj_eq_related H25. eee.
  aw. red in H3. ee. red in H22. ee. nin H3. app H24. awi H21; eee. fprops. ue.
  fprops. ue.
Qed.
```

(d) With the hypothesis of (a), suppose that E *is an* ordered *set and that the following condition is satisfied:*

(C') The relations $x \leq y \leq z$ *and* $x \equiv z \pmod{S}$ *in* E *imply* $x \equiv y \pmod{S}$.

Show that $R\{x, y\}$ *is then an* order *relation between* X *and* Y *in* E/S.

```
Definition quotient_order_axiom r s:=
  forall x y z, gle r x y -> gle r y z -> related s x z -> related s x y.

Lemma order_is_preorder: forall r,
  order r -> preorder r.
Proof. ir. assert (is_graph r). fprops.  red in H. red. intuition.
Qed.


Lemma Exercise1_2d: forall r s,
  is_equivalence s -> order r -> substrate s = substrate r ->
```

```
  quotient_order_axiom r s ->
  order (quotient_order r s).
Proof. ir. cp (order_is_preorder H0).
  cp (quotient_is_preorder H H3 H1). red in H4. ee. red. intuition.
  red. ee.  nin H4. am. ir. rwi quotient_order_pr H7. rwi quotient_order_pr H6.
  red in H6. red in H7. ee. set_extens. nin (H11 _ H12). ee.
  nin (H9 _ H13). ee. assert (related s x0 x2). rw in_class_related.
  exists x. eee. wrr inc_quotient. am.
  cp (H2 _ _ _ H14 H16 H17). cp (is_class_pr H H13 H10). rw H19.
  bw. app symmetricity_e.
  nin (H9 _ H12). ee. nin (H11 _ H13). ee. assert (related s x0 x2).
  rw in_class_related. exists y. eee. wrr inc_quotient. am.
  cp (H2 _ _ _ H14 H16 H17). cp (is_class_pr H H13 H6). rw H19.
  bw. app symmetricity_e.
Qed.
```

(e) Give an example of a totally ordered set E *with four elements and an equivalence relation* S *and* E *such that neither of the conditions (C) and (C') is satisfied, but such that* E/S *is an ordered set.*

Assume that E is a totally ordered finite set, and consider two classes X and Y; they have a greatest element x and y. The condition $x \leq y$ is equivalent to $X \leq Y$, so that the quotient is totally ordered. Assume $a < b < c$, S is such that a and c are related by S, but no other pair of distinct pairs are related. Then (C′) is false. In (C), take a, b and c for x, y and y'. Since $y \equiv y'$ implies $y = y'$, condition (C) is false. This gives an example with three elements. But one can add a fourth one.

(f) Let E *be an ordered set, let* f *be an increasing mapping of* E *into an ordered set* F*, and let* $S\{x, y\}$ *be the equivalence relation* $f(x) = f(y)$ *on* E*. Then the condition (C') is satisfied. Moreover the condition (C) is satisfied if and only if the relations* $x \leq y$ *and* $f(x) = f(x')$ *imply that there exists* $y' \in E$ *such that* $x' \leq y'$ *and* $f(y) = f(y')$*. Let* $f = g \circ \phi$ *be the canonical decomposition of* f*. Then* g *is an isomorphism of* E/S *onto* f(E) *if and only if this condition is satisfied and, in addition, the relation* $f(x) \leq f(y)$ *implies that there exists* x', y' *such that* $f(x) = f(x')$, $f(y) = f(y')$, *and* $x' \leq y'$.

```
Lemma Exercise1_2f1: forall r r' f, increasing_fun f r r' ->
  quotient_order_axiom r (equivalence_associated f).
Proof. ir. red. ir. red in H. ee. cp (H7 _ _ H0).  cp (H7 _ _ H1).
  rww ea_related. rwii ea_related H2. ee. am.  wr H5. order_tac.
  wri H11 H9. order_tac.
Qed.

Lemma Exercise1_2f2: forall r r' f, increasing_fun f r r' ->
  weak_order_compatibility r (equivalence_associated f) =
  (forall x y x', gle r x y -> inc x' (source f) -> W x f = W x' f ->
    exists y', inc y' (source f) & gle r x' y' &  W y f = W y' f).
Proof. ir. uf weak_order_compatibility. red in H. ee. ap iff_eq. ir. ee.
  assert (related (equivalence_associated f) x x'). rw ea_related.  ee.
  wr H2. order_tac. am. am. am. nin (H11 _ _ _ H6 H12).
  rwi ea_related H13. ee. exists x0. intuition. am. ir. ee.
  app order_is_preorder. app graph_ea_equivalence. rww graph_ea_substrate.
  sy; am. ir. rwi ea_related H7.  ee. nin (H5 _ _ _ H6 H8 H9). exists x0.
  rw ea_related. intuition. wr H2.  order_tac. am. am.
Qed.
```

The next point is straightforward, but a bit longish. The idea is the following. If X is in the quotient, and if x stands for *rep X*, then $g(\mathrm{X}) = f(x)$. The equivalence relation is defined by $a \in \mathrm{X}$ if and only if $f(x) = f(a)$.

```
Lemma Exercise1_2f3: forall r r' f g,
  let CC:= forall x y x', gle r x y -> inc x' (source f) -> W x f = W x' f ->
    exists y', inc y' (source f) & gle r x' y' &  W y f = W y' f in
  let DD:= forall x y, inc x (source f) -> inc y (source f) ->
    gle r' (W x f) (W y f) -> exists x', exists y',
      W x f = W x' f & W y f = W y' f & gle r x' y' in
  increasing_fun f r r' ->
  composable g (canon_proj (equivalence_associated f)) ->
  f = compose g (canon_proj (equivalence_associated f)) ->
  order_morphism g (quotient_order r (equivalence_associated f)) r' = (CC & DD).
Proof. ir. cp (Exercise1_2f1 H). rename H2 into Hc.
  red in H. ee. set (s:= equivalence_associated f) in *.
  assert (Ha:is_equivalence s). uf s. app graph_ea_equivalence.
  assert (Hb: substrate s = substrate r). uf s. sy; rww graph_ea_substrate.
  assert (Hd:preorder r). app order_is_preorder.
  assert (He: forall x, inc x (quotient s)  ->( inc (rep x) (source f) &
    W (rep x) f = W x g)). ir. cp (inc_rep_substrate Ha H7). split. wr H4.
  wrr Hb. cp (canon_proj_W Ha H8). rwii class_rep H9.
  assert (W (W (rep x) (canon_proj s)) g = W x g). rww H9. wri compose_W H10.
  wri H1 H10. am. am. aw.
  assert (Hf:forall x a, inc x (quotient s) ->
    inc a x = (inc (rep x)(source f) &inc a(source f) & W (rep x) f = W a f)).
  ir. ap iff_eq. ir. cp (related_rep_in_class Ha H7 H8). ufi s H9.
  rwi ea_related H9. am. am. ir. wr (class_rep Ha H7). bw.
  uf s. rw ea_related. am. am.
```

Assume that $g(a) \leq g(b)$ implies $a \leq b$. We have: if x and y in E, X and Y are their equivalence classes, then $f(x) \leq f(y)$ implies $\mathrm{X} \leq \mathrm{Y}$. Expanding the last relation, we get: if x' is in the class of x, there exists y' such that $f(y) = f(y')$ and $x' \leq y'$.

```
  app iff_eq. ir.
  assert (forall x y x', inc x (source f) -> inc y (source f) ->
    gle r' (W x f) (W y f) -> inc x' (class s x) ->
    exists y', inc y' (source f) &  gle r x' y' & W y f = W y' f).
  ir. assert(inc x (substrate s)). ue. ue.
  assert (inc x (class s x)). app inc_itself_class.
  assert (inc (class s x) (quotient s)). gprops.
  rwi (Hf _ x H14) H13. cp (He _ H14). ee. wri H18 H10. rwi H16 H10.
  assert(inc y (substrate s)). ue. ue.
  assert (inc y (class s y)). app inc_itself_class.
  assert (inc (class s y) (quotient s)). gprops.
  rwi (Hf _ y H21) H20. cp (He _ H21). ee. wri H25 H10. rwi H23 H10.
  red in H7. ee. wri H30 H10.  rwi quotient_order_pr H10. red in H10. ee.
  nin (H32 _ H11). exists x0. nin  H33. rwi (Hf _ x0 H21) H33. ee. am. am.
  assert (inc y (class s y)). app inc_itself_class. rwi (Hf _ y H21) H37.
  ee. wrr H39. red in H7. ee. wr H28. rww substrate_quotient_order.
  red in H7. ee. wr H28. rww substrate_quotient_order.
```

It is now easy to show that if g is a morphism, then the two conditions are true.

```
  ee. red. ir.
```

```
  cp (H6 _ _ H9). rwi H11 H12.
  assert (inc x' (class s x')). app inc_itself_class.  ue. ue.
  assert (inc y (source f)). wr H4. order_tac.
  nin (H8 _ _ _ H10 H14 H12 H13). exists x0. am.
  red. ir. assert (inc x (class s x)). app inc_itself_class. ue. ue.
  cp (H8 _ _ _ H9 H10 H11 H12).  nin H13. exists x. exists x0. eee.
```

We now show the converse.

```
  red. ir. assert (order (quotient_order r s)). app Exercise1_2d.
  ufi CC H7; ufi DD H7. nin H7.
  assert (source g = quotient s). red in H0. ee. awi H11. am.
  assert(substrate (quotient_order r s) = source g).  rw H10.
  rww substrate_quotient_order.
  assert (forall x y, inc x (source g) -> inc y (source g) ->
    gle (quotient_order r s) x y = gle r' (W x g) (W y g)). ir.
  rwi H10 H12. rwi H10 H13. nin (He _ H12). nin (He _ H13).
  wr H15. wr H17. ap iff_eq. ir. rwi quotient_order_pr H18. red in H18. ee.
  nin (H20 _ (inc_rep_itself Ha H18)). nin H21. rwi (Hf y x0 H13) H21. ee.
  rw H24. app H6. ir.  rw quotient_order_pr. red. eee.
  cp (H9 _ _ H14 H16 H18). nin H19; nin H19. ee. ir.
  rwi (Hf x x2 H12) H22. nin H22. nin H23. rwi H19 H24.
  nin (H7 _ _ _ H21 H23 H24). exists x3. ee. rw Hf. eee.
  am. am. red in H0. eee. red. ee. am. ir.
  assert (gle r' (W y g) (W y g)). order_tac. rw H5. rw H1. aw.
  app inc_W_target.  assert (gle r' (W x g) (W y g)). rww H17.
  assert (gle r' (W y g) (W x g)). rww H17.
  wri H12 H19. wri H12 H20. order_tac. am. am. am. am. rww H1.  aw.
Qed.
```

---

3. *Let* I *be an ordered set and let* $(E_\iota)_{\iota\in I}$ *be a family of non-empty ordered sets indexed by* I.

(a) Let F *be the* sum *(Chapter II, § 4, no. 8) of the family* $(E_\iota)_{\iota\in I}$*; for each* $x\in F$*, let* $\lambda(x)$ *be the index* ι *such that* $x\in E_\iota$*; and let* G *be the graph consisting of all the pairs* $(x,y)\in F\times F$ *such that either* $\lambda(x)<\lambda(y)$ *or else* $\lambda(x)=\lambda(y)$ *and* $x\le y$ *in* $E_{\lambda(x)}$*. Show that* G *is the graph of an ordering on* F*. The set* F *endowed with this ordering is called the* ordinal sum *of the family* $(E_\iota)_{\iota\in I}$ *(relative to the ordering on* I*) and is denoted* $\sum_{\iota\in I} E_\iota$*. Show that the equivalence relation corresponding to the partition* $(E_\iota)_{\iota\in I}$ *of* F *satisfies conditions (C) and (C') of Exercise 2, and that the quotient ordered set (Exercise 2) is canonically isomorphic to* I.

We have defined the ordinal sum in section 8.1, and shown that is an ordered set. We show here the claim of the last sentence.

Let's consider three variables, r, f and g. We define some quantities associated to them, prefixed by *E13*. Assumption H1 says that f is a family of sets (the family E_ι), r is an ordering on the index set, g is a family of orderings on E_ι. We shall denote the ordinal sum by F and its support (the disjoint union) by *sF* or $\bar{F}$. The quantity *lam* will be the function λ, i.e., the surjective function on $\bar{F}$ defined by $\lambda(x)=\mathrm{pr}_2 x$. Finally S the equivalence associated to λ. Assumption H2 says that the sets E_ι are non-empty.

```
Section Exercise1_3a.
Variables r f g: Set.
```

```
Definition E13_lam := second_proj (disjoint_union f).
Definition E13_S:= equivalence_associated (second_proj (disjoint_union f)).
Definition E13_F:= ordinal_sum r f g.
Definition E13_sF:= disjoint_union f.
Definition E13_H1:= ordinal_sum_axioms r f g.
Definition E13_H2:= is_graph f &
  (forall i, inc i (domain f) -> nonempty (V i f)).
```

We show, successively, that $\bar{F}$ is a graph, that pr_2 can be considered as a surjective function on F, that its target is I if H2 is true, and that it is an increasing function.

```
Lemma Exercise1_3a1: is_graph E13_sF.
Proof. ir. uf E13_sF. uf disjoint_union. uf
disjoint_union_fam. red. ir.
  srwi. nin H. nin H. bwi H0. awi H0. nin H0; am. bwi H. am.
Qed.

Lemma Exercise1_3a0: is_function E13_lam.
Proof. uf E13_lam. app function_second_proj. ap Exercise1_3a1.
Qed.

Lemma Exercise1_3a2: surjective E13_lam.
Proof. ap surjective_pr6. ap Exercise1_3a0. cp Exercise1_3a1. uf E13_lam.
  ir. ufi second_proj H0. awi H0. nin H0. exists (J x y). ee. uf second_proj.
  aw.  rw W_second_proj. aw. am. am. am.

Lemma Exercise1_3a3: E13_H2 -> domain f = target E13_lam.
Proof. ir. nin H. uf E13_lam.  uf second_proj. aw.
  set_extens. nin (H0 _ H1). aw. exists y. uf disjoint_union.
  uf disjoint_union_fam.
  srw. bw. ex_tac. bw. fprops. ap Exercise1_3a1.
  awi H1. nin H1. ufi disjoint_union H1. ufi disjoint_union_fam H1.
  srwi H1. nin H1. nin H1. bwi H1.  bwi H2. awi H2. ee. ue. am.
  ap Exercise1_3a1.
Qed.

Lemma Exercise1_3a4: E13_H1 -> E13_H2 -> increasing_fun E13_lam E13_F r.
Proof. ir. red in H. cp H; red in H. red. ee. ap Exercise1_3a0.
  uf E13_F. fprops. am. uf E13_F; uf E13_lam;  uf second_proj. aw.
  rw H2. app Exercise1_3a3.
  uf E13_lam. ir. cp Exercise1_3a1.  ufi E13_F H8. awi H8. ee.
  rww W_second_proj. rww W_second_proj. nin H11.
  nin H11; am. nin H11. rw H11. order_tac. cp (du_index_pr H10); eee. am.
Qed.
```

Consider now the "equivalence relation corresponding to the partition $(\mathrm{E}_\iota)_{\iota\in\mathrm{I}}$ of F". The set $\bar{\mathrm{F}}$ is the union of f', the graph of $\iota \mapsto \mathrm{E}_\iota \times \{\iota\}$, where f is the graph of $\iota \mapsto \mathrm{E}_\iota$. Consider now a function f'' whose graph is f'. This function defines a partition. By abuse of notations, F is identified with $\bar{\mathrm{F}}$, $\mathrm{E}_\iota \times \{\iota\}$ with E_ι and f'' with f'. Let S' be the equivalence associated to f''. We show here that x and y are related by S' (resp. S) if and only if x and y are in the disjoint union and $\mathrm{pr}_2 x = \mathrm{pr}_2 y$. This shows $\mathrm{S}' = \mathrm{S}$.

```
Definition disjoint_union_function f :=
  corresp (domain f)(range (disjoint_union_fam f))(disjoint_union_fam f).
```

```
Lemma disjoint_union_function_pr:
  is_function (disjoint_union_function f) &
  graph (disjoint_union_function f) =  (disjoint_union_fam f).
Proof. ir. uf disjoint_union_function. split. app is_function_pr.
  uf disjoint_union_fam. gprops. fprops.  uf disjoint_union_fam. bw. aw.
Qed.

Lemma Exercise1_3a5: forall x y, fgraph f ->
  related (partition_relation (disjoint_union_function f) (disjoint_union f))
  x y = (inc x E13_sF & inc y E13_sF & Q x = Q y).
Proof. ir. uf E13_sF. rename H into Ha. cp disjoint_union_function_pr.  ee.
  rw partition_relation_pr. uf in_same_coset. uf W. uf disjoint_union.
  uf disjoint_union_function. aw. uf disjoint_union_fam. ap iff_eq.
  ir. nin H1. ee. srw. bw. ex_tac. union_tac. bw.
  bwi H2.  bwi H3. awi H2; awi H3. eee. am. am.
  ir. ee. srwi H1. srwi H2. nin H1;nin H2. ee. bwi H1. bwi H2.
  assert (x0 = x1). bwi H5; bwi H4. awi H5; awi H4. ee. wr H9. ue. am. am. am.
  ex_tac. split. ue. am. am. rw H0. app partion_union_disjoint.
Qed.

Lemma Exercise1_3a6: forall x y,
  related E13_S x y = (inc x E13_sF & inc y E13_sF & Q x = Q y).
Proof. ir. cp Exercise1_3a1. assert (is_function E13_lam). ap Exercise1_3a0.
  uf E13_S. rw related_ea.
  assert (source (second_proj (disjoint_union f)) = E13_sF).
  uf second_proj. aw. rw H1.
  app iff_eq. ir. eee. rwii W_second_proj H4. rwii W_second_proj H4.
  ir. nin H2. nin H3. eee. rww W_second_proj. rww W_second_proj.  am.
Qed.
```

We show that the classes of S are the sets $E_\iota \times \{\iota\}$. If ϕ is the canonical projection of $\bar{F}$ on $\bar{F}/S$, there exists a function g defined by $\lambda = g \circ \phi$ (canonical decomposition) because of the surjectivity of λ, it is unique. It is the function induced by λ by passing on the quotient. We can restate it as: an element X of the quotient is of the form $\iota \mapsto E_\iota \times \{\iota\}$, and $g(X) = \iota$. We show that g is a function, that $g \circ \phi$ exists and that g is bijective.

```
Lemma Exercise1_3a7: is_equivalence E13_S.
Proof. uf E13_S. app equivalence_graph_ea. ap Exercise1_3a0. Qed.

Lemma Exercise1_3a8: forall a, E13_H2 ->
  inc a (quotient E13_S) = exists i,
    inc i (domain f) & a = product (V i f)  (singleton i).
Proof. ir. cp Exercise1_3a7. rww inc_quotient. rww is_class_rw.
  assert (substrate E13_S = disjoint_union f). uf E13_S. rww substrate_graph_ea.
  uf second_proj. aw.
  app function_second_proj. ap Exercise1_3a1.
  ap iff_eq. ir. ee. nin H2. assert (inc y (disjoint_union f)). wr H1. app H3.
  cp (du_index_pr H5). ee.  ex_tac. set_extens.
  rwi (H4 y x H2) H9. ufi E13_S H9. rwi Exercise1_3a6 H9. ee.
  cp (du_index_pr H10). ee. app product_inc: ue.
  rw (H4 _ x H2). uf E13_S. awi H9. ee. cp H11. rw Exercise1_3a6.
  ee. am. uf E13_sF. uf disjoint_union. srw.
  uf disjoint_union_fam. bw. ex_tac. bw. aw. au. sy; am.
  ir. nin H2. nin H2. assert (sub a (disjoint_union f)). red. ir.
  uf disjoint_union.  srw. uf disjoint_union_fam. bw.  ex_tac. bw. ue.
  ee. nin H. ee. cp (H5 _ H2). nin H6. exists (J y x). ue.  ue.
```

```
  ir. uf E13_S. rw Exercise1_3a6. ap iff_eq. ir. ee. app H4. app H4.
  rwi H3 H5. awi H5. rwi H3 H6. awi H6. ee. ue.
  ir. ee.  rw H3. cp (du_index_pr H7).
  rwi H3 H5. awi H5.  nin H5. nin H10. rwi H11 H8.
  aw. eee.
Qed.

Lemma Exercise1_3a9: is_function(fun_on_quotient E13_S E13_lam).
Proof. ir. cp Exercise1_3a0. app function_foqc. ap Exercise1_3a7.
  uf E13_S. rww substrate_graph_ea.
Qed.

Lemma Exercise1_3a10:
  composable  (fun_on_quotient E13_S E13_lam) (canon_proj E13_S).
Proof.  red. ee. ap  Exercise1_3a9. ap function_canon_proj. ap  Exercise1_3a7.
   uf fun_on_quotient. uf section_canon_proj. aw.
Qed.

Lemma Exercise1_3a11:
  E13_lam =  compose (fun_on_quotient E13_S E13_lam) (canon_proj E13_S).
Proof. ap (canonical_decomposition_surj2 Exercise1_3a2).
Qed.

Lemma Exercise1_3a12: forall x,  E13_H2 ->
  inc x (quotient E13_S) -> exists i,
    inc i (domain f) & x = product (V i f)  (singleton i) &
    W x (fun_on_quotient E13_S E13_lam) = i.
Proof. ir. cp Exercise1_3a0. cp  Exercise1_3a7.
  assert (source E13_lam = substrate E13_S).
  uf E13_S. sy; app substrate_graph_ea.
  assert (Q (rep x) = W (rep x) E13_lam). uf E13_lam. rww W_second_proj.
  ap Exercise1_3a1. ufi E13_lam H3.  ufi second_proj H3. rw H3.
  app inc_rep_substrate. rww W_foqc. wr H4.
  rwi (Exercise1_3a8 x H) H0. nin H0. nin H0. ex_tac.
  assert (inc (rep x) x). app nonempty_rep. nin H. ee. nin (H6 _ H0).
  exists (J y x0). rw H5. aw. ee.  fprops. am. fprops. set (t:=rep x) in *.
  rwi H5 H6. awi H6. eee.
Qed.

Lemma Exercise1_3a13: E13_H2 ->  bijective (fun_on_quotient E13_S E13_lam).
Proof. ir.
  assert (Ha: source (fun_on_quotient E13_S E13_lam) = quotient E13_S).
  uf fun_on_quotient. uf section_canon_proj. aw.
  cp  Exercise1_3a9. red. split. red. rw Ha. split. am. ir.
  ir. nin (Exercise1_3a12 H H1).  nin (Exercise1_3a12 H H2). ee.
  rw H8; rw H6; wr H9; rw H3;rw H7. tv.
  app surjective_pr6. rw Ha. cp (Exercise1_3a3 H). ir.
  ufi fun_on_quotient H2. awi H2.
  ir.  set (a:=product (V y f)  (singleton y)).
  assert (inc a  (quotient E13_S)). rww Exercise1_3a8. exists y. rw H1. au.
  ex_tac. nin (Exercise1_3a12 H H3). ee. rw H6.
  cp Exercise1_3a7. cp (inc_rep_itself H7 H3).  set (t:= rep a) in H8.
  cp H8. ufi a H8. awi H8. rwi H5 H9. awi H9. eee.
Qed.
```

We apply Exercise 2f; the two conditions are easily satisfied. We deduce that g is an isomorphism on its image. Since we know that g is bijective, it is an order isomorphism between F/S and I.

```
Lemma Exercise1_3a14: E13_H1 -> quotient_order_axiom E13_F E13_S.
Proof. ir. red. uf E13_F. uf E13_S. ir. assert (order (ordinal_sum r f g)).
  red in H; fprops. rw Exercise1_3a6. rwi Exercise1_3a6 H2. awi H0. awi H1.
  eee. nin H. nin H9. nin H7. assert (glt r (Q x) (Q z)). order_tac.
  nin H11. elim H12.  am. nin H7. ue. nin H9. ue. am. am.
Qed.

Lemma Exercise1_3a15: E13_H1 ->  E13_H2 ->
  order_isomorphism  (fun_on_quotient E13_S E13_lam)
  (quotient_order E13_F E13_S) r.
Proof. ir.  cp Exercise1_3a1. assert (order (ordinal_sum r f g)). fprops.
  assert (order_morphism (fun_on_quotient E13_S E13_lam)
    (quotient_order E13_F (equivalence_associated E13_lam)) r).
  rw Exercise1_2f3.
  assert (source E13_lam = disjoint_union f). uf E13_lam. uf second_proj. aw.
  rw H3. uf E13_F. split. ir. awi H4.  ee.
  assert (W x E13_lam = Q x). uf E13_lam. rww W_second_proj.
  assert (W x' E13_lam = Q x'). uf E13_lam. rww W_second_proj.
  assert (W y E13_lam = Q y). uf E13_lam. rww W_second_proj.
  nin H8. exists y.  ee. am.  aw. ee. am. am. left. wr H10. wr H6. rww H9. tv.
  exists x'. ee. am. order_tac. rww substrate_ordinal_sum.
  wr H6. rw H11. rw H9. sy. am. am.
  ir. awi H6. ee. exists x. nin (equal_or_not (W x E13_lam) (W y E13_lam)).
  exists x. split.  tv.  split. sy. am. order_tac.
  rww substrate_ordinal_sum. exists y. eee. aw. eee. left.
  assert (W x E13_lam = Q x). uf E13_lam. rww W_second_proj.
  assert (W y E13_lam = Q y). uf E13_lam. rww W_second_proj.
  wr H8; wr H9; red; split;am.
  app Exercise1_3a4. app Exercise1_3a10. app Exercise1_3a11.
  cp Exercise1_3a13. red in H3. red. intuition.
Qed.
End Exercise1_3a.
```

(b) If the set I *is the ordinal sum of a family* $(J_\lambda)_{\lambda\in L}$ *of ordered sets, where* L *is an ordered set, show that the ordered set* $\sum_{\iota\in I} E_\iota$ *is canonically isomorphic to the ordinal sum* $\sum_{\lambda\in L} F_\lambda$ *where* $F_\lambda = \sum_{\iota\in I_\lambda} E_\iota$ *("associativity of the ordinal sum"). If* I *is the linearly ordered set* $\{1,2\}$*, we write* $E_1 + E_2$ *for the ordinal sum of* E_1 *and* E_2*. Show that* $E_2 + E_1$ *and* $E_1 + E_2$ *are not necessarily isomorphic.*

The associativity formula has been shown in the main text. Consider two ordered sets E_1 and E_2 that we identify as subsets of the sum $E_1 + E_2$. Let x be the greatest element of the sum $E_1 + E_2$. Assume E_2 not empty; thus $y \in E_2$. Since $y \le x$, we deduce $x \in E_2$. Obviously, for all $z \in E_2$ we have $z \le x$. Thus, we have shown: if the sum $E_1 + E_2$ has a greatest element, then E_2 has a greatest element. This shows that $E_1 + E_2$ is not always isomorphic to $E_2 + E_1$.

```
Lemma ordinal_sum2_order_greatest: forall r r' x,  order r -> order r' ->
  nonempty (substrate r') ->
  greatest_element  (ordinal_sum2 r r') x -> greatest_element r' (P x).
Proof. ir. nin H1. red in H2. nin H2.
  assert (inc (J y (singleton emptyset)) (substrate (ordinal_sum2 r r'))).
```

```
  rw substrate_ordinal_sum2. rw canonical_du2_pr. split. fprops. right.
  aw. au. am. am. cp (H3 _ H4).
  rwi substrate_ordinal_sum2 H2. rwi canonical_du2_pr H2. nin H2. nin H6.
  nin H6. cp (related_ordinal_sum2_order_spec H H0 H6 H1).
  assert (x = J (P x) emptyset). app pair_extensionality. fprops. aw. aw.
  wri H9 H8. elim (not_le_gt (order_ordinal_sum2 H H0) H5 H8).
  ee. red. ee. am.  ir.
  assert (inc (J x0 (singleton emptyset)) (substrate (ordinal_sum2 r r'))).
  rw substrate_ordinal_sum2. rw canonical_du2_pr. split. fprops. right.
  aw. au. am. am. cp (H3 _ H9). rwi related_ordinal_sum2_order H10.
  ee. nin H12. nin H12. awi H12. elim one_not_zero. am. nin H12. nin H12.
  nin H13. awi H14. am. nin H12.  awi H12.  elim one_not_zero. am. am. am.
  am. am.
Qed.
```

(c) An ordinal sum $\sum_{\iota\in I} E_\iota$ *is right directed if and only if* I *is right directed and* E_ω *is right directed for each* ω *of* I.

(d) An ordinal sum $\sum_{\iota\in I} E_\iota$ *is totally ordered if and only if* I *and each* E_ι *is totally ordered.*

(e) An ordinal sum $\sum_{\iota\in I} E_\iota$ *is a lattice if and only if the following conditions are satisfied:*

> *(I) The set* I *is a lattice, and for each pair* (λ,μ) *of non-comparable indices in* I, $E_{\sup(\lambda,\mu)}$ *(resp.* $E_{\inf(\lambda,\mu)}$*) has a least (resp. greatest) element.*
>
> *(II) For each* $\alpha\in I$ *and each pair* (x,y) *of elements of* E_α *such that the set* $\{x,y\}$ *is bounded above (resp. bounded below) in* E_α*, the set* $\{x,y\}$ *has a least upper bound (resp. greatest lower bound) in* E_α*.*
>
> *(III) For each* $\alpha\in I$ *such that* E_α *contains a set of two elements which has no upper bound (resp. no lower bound) in* E_α*, the set of indices* $\lambda\in I$ *such that* $\lambda>\alpha$ *(resp.* $\lambda<\alpha$*) has a least element (resp. a greatest element)* β*, and* E_β *has a least element (resp. greatest element).*

The French edition corrects point (c) as: "il faut et il suffit que I soit filtrant à droite et que, pour tout élément maximal ω de E, E_ω soit filtrant à droite." This can be translated as: An ordinal sum $\sum_{\iota\in I} E_\iota$ is right directed if and only if I is right directed and E_ω is right directed for each maximal element ω of I.

The proof is as follows. We identify E_k with a subset of the disjoint union. Assume the sum directed. For every pair of indices i and j, there is $x\in E_i$ and $y\in E_j$, and if $z\in E_k$ is an upper bound for x and y, then k is an upper bound for i and j. Let k be a maximal index, x and y in E_k. Every upper bound of x and y in the sum must be in E_k. Conversely, let $x\in E_i$ and $y\in E_j$ be in the disjoint union; assume I right directed, so that there is $k\in I$ with $i\le k$ and $j\le k$. If k is one of i and j, but not both, then $x<y$ or $y<x$. If $i<k$ and $j<k$, there is $z\in E_k$ with $x<z$ and $y<z$. If k is not maximal, there is k' with $k<k'$, and the same conclusion holds. Finally, if $i=j=k$ and k is maximal, we have an upper bound since E_k is right directed.

```
Lemma directed_ordinal_sum_order: forall r f g,
  ordinal_sum_axioms1 r f g ->
  right_directed (ordinal_sum r f g) = (right_directed r &
    forall i, maximal_element r i -> right_directed (V i g)).
Proof. ir. red in H.  ee. cp H. red in H1. ee.
```

```
  assert (Ha: order (ordinal_sum r f g)). fprops.
  ap iff_eq. ir. rwi right_directed_pr H8. awi H8. ee. rw right_directed_pr.
  ee. am. ir. rwi H2 H10; rwi H2 H11. nin (H0 _ H10). nin (H0 _ H11).
  cp (inc_disjoint_union H10 H12). cp (inc_disjoint_union H11 H13).
  nin (H9 _ _ H14 H15). ee. exists (Q x0). cp (du_index_pr H16). ee.
  rw H2. am.
  cp (related_ordinal_sum_order_id H H17). awi H22. am.
  cp (related_ordinal_sum_order_id H H18). awi H22. am. ee.
  ir. rw right_directed_pr. red in H10. ee. app H6. wrr H2. ir. rwi H2 H10.
  rwi (H7 _ H10) H12. cp (inc_disjoint_union H10 H12).
  rwi (H7 _ H10) H13. cp (inc_disjoint_union H10 H13).
  nin (H9 _ _ H14 H15). ee. cp (related_ordinal_sum_order_id H H17). awi H19.
  cp (du_index_pr H16). ee. cp (H11 _ H19).
  awi H17; awi H18. ee. exists (P x0). ee.  rww H7. wrr H23. nin H27.
  nin H27.  elim H28. sy; am. ee; am. nin H25. nin H25. elim H28. sy;am. eee.
  am. am. am. am.
  ir. ee. rwi right_directed_pr H8. ee. rw right_directed_pr. ee. am.
  aw. ir. cp (du_index_pr H11). cp (du_index_pr H12). ee. rwi H2 H10.
  nin (H10 _ _  H13 H14). ee.
  nin (equal_or_not (Q x) x0). nin (equal_or_not (Q y) x0).
  nin (p_or_not_p (maximal_element r x0)). cp (H9 _ H24).
  rwi right_directed_pr H25. ee. rwi H7 H26. rwi H22 H17; rwi H23 H15.
  nin (H26 _ _ H17 H15).  ee. exists (J x1 x0). ee. app inc_disjoint_union.
  aw. ee. am.  app inc_disjoint_union.  right. eee.
  aw. ee. am.  app inc_disjoint_union.  right. eee. am.
  ufi maximal_element H24.
  assert (exists x1, inc x1 (substrate r) & glt r x0 x1). app exists_proof.
  dneg. ee. rww H2. ir. nin (equal_or_not x0 x1).  sy; am.
  elim (H25 x1).  ee. order_tac. split; am. nin H25. ee. rwi H2 H25.
  nin (H0 _ H25).
  cp (inc_disjoint_union H25 H27).  ex_tac. aw. eee.
  exists x. ee. am. order_tac. aw.
  aw. eee. left. rw H22. red. eee.
  nin (equal_or_not (Q y) x0). exists y. ee. am. aw. eee.
  left. rw H23. order_tac. order_tac. aw.
  nin (H0 _ H19). cp (inc_disjoint_union H19 H24). ex_tac.
  ee. aw. eee. left. order_tac. aw. eee. left. order_tac.
Qed.
```

We show that if an ordinal sum is a lattice, so is the index set I. The idea is the following. Consider two indices a and b. If they are comparable, there is a supremum. Otherwise, let $x \in \mathrm{E}_a$, $y \in \mathrm{E}_b$ and $z = \sup(x, y)$. We have $z \in \mathrm{E}_c$, and c is an upper bound of a and b. Let d be another upper bound. Since a and b are incomparable, we have $a < d$ and $b < d$. For every $t \in \mathrm{E}_d$ we have $x < t$ and $y < t$ hence $z \leq t$ hence $c \leq d$.

```
Lemma ordinal_sum_pr1: forall r f g,
  ordinal_sum_axioms1 r f g ->
  forall i, inc i (domain f) -> exists y, inc y (V i f) &
    inc (J y i) (substrate (ordinal_sum r f g)).
Proof. ir. red in H. ee. nin (H1 _ H0). ex_tac.
  set (inc_disjoint_union H0 H2). aw.
Qed.


Lemma ordinal_sum_lattice1: forall r f g,
  ordinal_sum_axioms1 r f g ->
```

```
  lattice (ordinal_sum r f g) -> (lattice r).
Proof.  ir. set (F:= ordinal_sum r f g) in H0.
  assert (Ha:forall i, inc i (domain f) -> exists y, inc y (V i f) &
    inc (J y i) (substrate F)). ir. uf F. app ordinal_sum_pr1. nin H.
  assert (order F). uf F. fprops. cp H; red in H;ee.
  assert (Hb: substrate F = disjoint_union f). uf F. aw.
   red. ee. am. ir. rwi H4 H10; rwi H4 H11. nin (Ha _ H10). nin (Ha _ H11).
  ee. cp (lattice_sup_pr H0 H15 H14).  fold F in H16. ee. red.
  nin (p_or_not_p (gle r x y)). exists y. app sup_comparable. ir.
  nin (p_or_not_p (gle r y x)). exists x. rw doubleton_symm.
  app sup_comparable. ir.
  set (z:= (sup F (J x0 x) (J x1 y))) in *. cp (inc_arg2_substrate H16).
  rwi Hb H21. cp (du_index_pr H21). ee. exists (Q z).
  app least_upper_bound_doubleton. ufi F H16.
  cp (related_ordinal_sum_order_id H3 H16). awi H25. am.
  ufi F H1. cp (related_ordinal_sum_order_id H3 H17). awi H25. am.
  ir. cp (inc_arg2_substrate H25). rwi H4 H27. nin (Ha _ H27). nin H28.
  assert (gle F (J x0 x) (J x2 t)). uf F. aw. eee. left.
  order_tac. wri H30 H26. contradiction.
  assert (gle F (J x1 y) (J x2 t)). uf F. aw. eee. left.
  order_tac. wri H31 H25. contradiction.  cp (H18 _ H30 H31).
  ufi F H32. cp (related_ordinal_sum_order_id H3 H32). awi H33. am.
  cp (lattice_inf_pr H0 H15 H14).  fold F in H16. ee. red.
  nin (p_or_not_p (gle r x y)). exists x. app inf_comparable. ir.
  nin (p_or_not_p (gle r y x)). exists y. rw doubleton_symm.
  app inf_comparable. ir.
  set (z:= (inf F (J x0 x) (J x1 y))) in *. cp (inc_arg2_substrate H16).
  rwi Hb H21. cp (du_index_pr H21). ee. exists (Q z).
  app greatest_lower_bound_doubleton. ufi F H16.
  cp (related_ordinal_sum_order_id H3 H16). awi H25. am.
  ufi F H1. cp (related_ordinal_sum_order_id H3 H17). awi H25. am.
  ir. cp (inc_arg1_substrate H25). rwi H4 H27. nin (Ha _ H27). nin H28.
  assert (gle F (J x2 t) (J x0 x)). uf F. aw. eee. left.
  order_tac. rwi H30 H26. contradiction.
  assert (gle F (J x2 t) (J x1 y)). uf F. aw. eee. left.
  order_tac. rwi H31 H25. contradiction. cp (H18 _ H30 H31).
  ufi F H32. cp (related_ordinal_sum_order_id H3 H32). awi H33. am.
Qed.
```

With the same notations as before, assume that a and b cannot be compared, let $x \in \mathrm{A}_a$, $y \in \mathrm{E}_b$ and $z \in \mathrm{E}_c = \sup(x, y)$. Then $c = \sup(a, b)$. Every z' in E_c is an upper bound of x and y. Hence $\sup(x, y)$ is the smallest element of E_c.

```
Lemma ordinal_sum_lattice2: forall r f g,
  ordinal_sum_axioms1 r f g ->
  lattice (ordinal_sum r f g) ->
  (forall i j, inc i (domain f) -> inc j (domain f) ->
    (gle r i j \/ gle r j i \/ (exists u, exists v,
      least_element (V (sup r i j) g) u &
      greatest_element (V (inf r i j) g) v))).
Proof. ir. cp (ordinal_sum_lattice1 H H0).  set (F:= ordinal_sum r f g) in H0.
  assert (Ha:forall i, inc i (domain f) -> exists y, inc y (V i f) &
    inc (J y i) (substrate F)). ir. uf F. app ordinal_sum_pr1. nin H.
  assert (order F). uf F. fprops. cp H; red in H;ee.
  assert (Hb: substrate F = disjoint_union f). uf F. aw.
```

```
  nin (p_or_not_p (gle r i j)). au.
  nin (p_or_not_p (gle r j i)). au. right. right.
  nin (Ha _ H1). nin (Ha _ H2). ee.
  cp (lattice_sup_pr H0 H18 H17). cp (lattice_inf_pr H0 H18 H17).
  set (A:= inf F (J x i) (J x0 j)) in H20. set (a:= inf r i j).
  set (B:= sup F (J x i) (J x0 j)) in H19. set (b:= sup r i j). ee.
  wri H7 H1.  wri H7 H2.  cp (lattice_inf_pr H3 H1 H2). fold a in H25. ee.
  cp (lattice_sup_pr H3 H1 H2). fold b in H28. ee.
  assert (Hc:Q A = a).
  assert (gle r (Q A) a). ufi F H20. cp (related_ordinal_sum_order_id H6 H20).
  ufi F H21. cp (related_ordinal_sum_order_id H6 H21). awi H31. awi H32.
  app H27. cp (inc_arg1_substrate H25). rwi H7 H32. nin (Ha  _  H32). ee.
  assert (gle F (J x1 a) (J x i)). uf F. aw. eee. left.
  order_tac. rwi H35 H26. contradiction.
  assert (gle F (J x1 a) (J x0 j)). uf F. aw. eee. left.
  order_tac. rwi H36 H25. contradiction.
  cp (H22 _  H35 H36).  uf F.  cp (related_ordinal_sum_order_id H6 H37).
  awi H38. order_tac.
  assert (Hd:Q B = b).
  assert (gle r b (Q B)). ufi F H19. cp (related_ordinal_sum_order_id H6 H19).
  ufi F H23. cp (related_ordinal_sum_order_id H6 H23). awi H31. awi H32.
  app H30. cp (inc_arg2_substrate H28). rwi H7 H32. nin (Ha  _  H32). ee.
  assert (gle F (J x i) (J x1 b)). uf F. aw. eee. left.
  order_tac. wri H35 H29. contradiction.
  assert (gle F (J x0 j)  (J x1 b)). uf F. aw. eee. left.
  order_tac. wri H36 H28. contradiction.
  cp (H24 _  H35 H36).  uf F.  cp (related_ordinal_sum_order_id H6 H37).
  awi H38. order_tac.
  cp (inc_arg2_substrate H23). rwi Hb H31. cp (du_index_pr H31). ee.
  wri (H12 _ H32) H33. rwi Hd H33.
  cp (inc_arg1_substrate H21). rwi Hb H35. cp (du_index_pr H35). ee.
  wri (H12 _ H36) H37. rwi Hc H36. exists (P B). exists (P A). ee.
  red. ee. am. ir. rwii H12 H39.
  assert (inc (J x1 b) (disjoint_union f)). app inc_disjoint_union. wrr Hd.
  assert (gle F (J x i) (J x1 b)). uf F. aw. eee. left.
  order_tac. wri H41 H29. contradiction.
  assert (gle F (J x0 j)  (J x1 b)). uf F. aw. eee. left.
  order_tac. wri H42 H28.  contradiction.
  cp (H24 _  H41 H42). ufi F H43. awi H43. ee.  nin H45. red in H45. ee.
  contradiction. ee. wrr H45. am. ue.
  red. ee. ue. ir. rwii H12 H39.
  assert (inc (J x1 a) (disjoint_union f)). app inc_disjoint_union.
  assert (gle F (J x1 a) (J x i) ). uf F. aw. eee. left.
  order_tac. rwi H41 H26. contradiction.
  assert (gle F (J x1 a) (J x0 j)). uf F. aw. eee. left.
  order_tac. rwi H42 H25. contradiction.
  cp (H22 _  H41 H42). ufi F H43. awi H43. ee.  nin H45. red in H45. ee.
  elim H46. sy. am. ee. am. am.
Qed.
```

Assume $a = b$ and t is an upper bound for x and y in E_a. Then $z \leq t$, thus $c \leq a$, hence $a = c$. This means $z \in \mathrm{E}_a$, so that x and y have a least upper bound in E_a.

```
Lemma ordinal_sum_lattice3: forall r f g i x y t,
  ordinal_sum_axioms1 r f g ->   lattice (ordinal_sum r f g) ->
  inc i (domain f) -> gle (V i g) x t -> gle (V i g) y t ->
```

```
    has_supremum (V i g) (doubleton x y).
Proof. ir. set (F:= ordinal_sum r f g) in H0.
  assert (Ha:forall i, inc i (domain f) -> exists y, inc y (V i f) &
    inc (J y i) (substrate F)). ir. uf F. app ordinal_sum_pr1. nin H.
  assert (order F). uf F. fprops. cp H; red in H;ee.
  assert (Hb: substrate F = disjoint_union f). uf F. aw.
  cp (inc_arg1_substrate H2). cp (inc_arg2_substrate H2).
  cp (inc_arg1_substrate H3).
  assert (inc (J x i) (substrate F)). uf F. aw. app inc_disjoint_union. wrr H12.
  assert (inc (J y i) (substrate F)). uf F. aw. app inc_disjoint_union. wrr H12.
  assert (inc (J t i) (substrate F)). uf F. aw. app inc_disjoint_union. wrr H12.
  cp (lattice_sup_pr H0 H17 H16). ee.
  assert (gle F (J y i) (J t i)). uf F. aw. eee.
  assert (gle F (J x i) (J t i)). uf F. aw. eee.
  cp (H21 _ H22 H23). set (z:= sup F (J y i) (J x i)) in *.
  cp (related_ordinal_sum_order_id H6 H19). awi H25.
  cp (related_ordinal_sum_order_id H6 H24). awi H26.
  cp (order_antisymmetry H H25 H26).
  exists (P z). ap least_upper_bound_doubleton. app H11.
  ufi F H20. awi H20. ee. nin H29. red in H29. nin H29. contradiction. eee.
  am. ufi F H19. awi H19. ee. nin H29. red in H29. nin H29. contradiction.
  eee. am. ir.
  assert (inc (J t0 i) (substrate F)). uf F. aw. app inc_disjoint_union.
  wrr H12. cp (inc_arg2_substrate H28). am.
  assert (gle F (J y i) (J t0 i)). uf F. aw. eee.
  assert (gle F (J x i) (J t0 i)). uf F. aw. eee.
  cp (H21 _ H31 H32).  ufi F H33. awi H33. ee. nin H35.
  red in H35. ee. elim H36. sy; am. ee. rwi H35 H36. am. am.
Qed.

Lemma ordinal_sum_lattice4: forall r f g i x y t,
  ordinal_sum_axioms1 r f g ->   lattice (ordinal_sum r f g) ->
  inc i (domain f) -> gle (V i g) t x -> gle (V i g) t y ->
    has_infimum (V i g) (doubleton x y).
Proof. ir. set (F:= ordinal_sum r f g) in H0.
  assert (Ha:forall i, inc i (domain f) -> exists y, inc y (V i f) &
    inc (J y i) (substrate F)). ir. uf F. app ordinal_sum_pr1. nin H.
  assert (order F). uf F. fprops. cp H; red in H;ee.
  assert (Hb: substrate F = disjoint_union f). uf F. aw.
  cp (inc_arg2_substrate H2). cp (inc_arg1_substrate H2).
  cp (inc_arg2_substrate H3).
  assert (inc (J x i) (substrate F)). uf F. aw. app inc_disjoint_union. wrr H12.
  assert (inc (J y i) (substrate F)). uf F. aw. app inc_disjoint_union. wrr H12.
  assert (inc (J t i) (substrate F)). uf F. aw. app inc_disjoint_union. wrr H12.
  cp (lattice_inf_pr H0 H17 H16). ee.
  assert (gle F (J t i) (J y i)). uf F. aw. eee.
  assert (gle F (J t i) (J x i)). uf F. aw. eee.
  cp (H21 _ H22 H23). set (z:= inf F (J y i) (J x i)) in *.
  cp (related_ordinal_sum_order_id H6 H19). awi H25.
  cp (related_ordinal_sum_order_id H6 H24). awi H26.
  cp (order_antisymmetry H H25 H26).
  exists (P z). ap greatest_lower_bound_doubleton. app H11.
  ufi F H20. awi H20. ee. nin H29. red in H29. nin H29. contradiciton. wr H27.
  eee. am. ufi F H19. awi H19. ee. nin H29. red in H29. nin H29. contradiction.
  wr H27.  eee. am. ir.
  assert (inc (J t0 i) (substrate F)). uf F. aw. app inc_disjoint_union.
```

```
  wrr H12. cp (inc_arg1_substrate H28). am.
  assert (gle F (J t0 i) (J y i)). uf F. aw. eee.
  assert (gle F (J t0 i) (J x i)). uf F. aw. eee.
  cp (H21 _ H31 H32).  ufi F H33. awi H33. ee. nin H35.
  red in H35. ee. elim H36. sy; am. ee. rwi H35 H36. wrr H27. am.
Qed.
```

Assume again $a = b$, but now, suppose no upper bound for x and y exists in E_a. Then the interval $]a, c[$ is empty and z is the least element of E_c. The first condition can be restated as: c is the least element of $]a, \rightarrow[$.

```
Lemma ordinal_sum_lattice5: forall r f g i x y,
  ordinal_sum_axioms1 r f g ->   lattice (ordinal_sum r f g) ->
  inc i (domain f) ->  inc x (V i f) ->  inc y (V i f) ->
  (forall t, inc t (V i f) -> ~ (gle (V i g) x t & gle (V i g) y t)) ->
  exists j, inc j (domain f) &
    least_element (induced_order r (Zo (domain f) (fun k=> glt r i k))) j &
    exists z, least_element  (V j g) z.
Proof. ir. set (F:= ordinal_sum r f g) in H0.
  assert (Ha:forall i, inc i (domain f) -> exists y, inc y (V i f) &
    inc (J y i) (substrate F)). ir. uf F. app ordinal_sum_pr1. nin H.
  assert (order F). uf F. fprops. cp H; red in H;ee.
  assert (Hb: substrate F = disjoint_union f). uf F. aw.
  assert (inc (J x i) (substrate F)). uf F. aw. app inc_disjoint_union.
  assert (inc (J y i) (substrate F)). uf F. aw. app inc_disjoint_union.
  cp (lattice_sup_pr H0 H14 H15). ee. set (Z:=sup F (J x i) (J y i)) in *.
  ufi F H16. ufi F H17. awi H16; awi H17. ee.
  assert (inc (Q Z) (domain f)).  cp (du_index_pr H21).  eee.
  assert (glt r i (Q Z)). nin H22. am.  nin H20. am. ee.
  cp (du_index_pr H21). ee. wri H20 H27. elim (H4 _ H27). eee.
  assert (sub (Zo (domain f) (fun k  => glt r i k)) (substrate r)).
  red. ir. Ztac. red in H27. ee.  order_tac.
  exists (Q Z). ee. am. red. ee.  aw. Ztac.  ir. awi H26. Ztac.
  clear H26. aw. nin (Ha _ H27). ee.
  assert (gle F (J x i) (J x1 x0)). uf F. aw. eee.
  assert (gle F (J y i) (J x1 x0)). uf F. aw. eee.
  cp (H18 _ H30 H31). cp (related_ordinal_sum_order_id H7 H32). awi H33. am.
  Ztac. Ztac. am. am. exists (P Z). red. ee. rww H13.
  cp (du_index_pr H19). ee. am.  ir.  rwi H13 H26.
  assert (inc (J x0 (Q Z)) (disjoint_union f)). app inc_disjoint_union.
  assert (gle F (J x i) (J x0 (Q Z))). uf F. aw. eee.
  assert (gle F (J y i) (J x0 (Q Z))). uf F. aw. eee.
  cp (H18 _ H28 H29). ufi F H30. awi H30. ee.  nin H32. red in H32. ee.
  elim  H33. tv. ee. am. am. am. am. am. am.
Qed.

Lemma ordinal_sum_lattice6: forall r f g i x y,
  ordinal_sum_axioms1 r f g ->   lattice (ordinal_sum r f g) ->
  inc i (domain f) ->  inc x (V i f) ->  inc y (V i f) ->
  (forall t, inc t (V i f) -> ~ (gle (V i g) t x & gle (V i g) t y)) ->
  exists j, inc j (domain f) &
    greatest_element (induced_order r (Zo (domain f) (fun k=> glt r k i))) j &
    exists z, greatest_element  (V j g) z.
Proof. ir. set (F:= ordinal_sum r f g) in H0.
  assert (Ha:forall i, inc i (domain f) -> exists y, inc y (V i f) &
    inc (J y i) (substrate F)). ir. uf F. app ordinal_sum_pr1. nin H.
```

```
  assert (order F). uf F. fprops. cp H; red in H;ee.
  assert (Hb: substrate F = disjoint_union f). uf F. aw.
  assert (inc (J x i) (substrate F)). uf F. aw. app inc_disjoint_union.
  assert (inc (J y i) (substrate F)). uf F. aw. app inc_disjoint_union.
  cp (lattice_inf_pr H0 H14 H15). ee. set (Z:=inf F (J x i) (J y i)) in *.
  ufi F H16. ufi F H17. awi H16; awi H17. ee.
  assert (inc (Q Z) (domain f)).  cp (du_index_pr H16).  ee. am.
  assert (glt r (Q Z) i). nin H22. am.  nin H20. am. ee.
  cp (du_index_pr H16). ee. rwi H20 H27. elim (H4 _ H27). wr H22. eee.
  assert (sub (Zo (domain f) (fun k  => glt r k i)) (substrate r)).
  red. ir. Ztac. red in H27. ee. order_tac.
  exists (Q Z). ee. am. red. ee.  aw. Ztac.  ir. awi H26. Ztac.
  clear H26. aw. nin (Ha _ H27). ee.
  assert (gle F (J x1 x0) (J x i)). uf F. aw. eee.
  assert (gle F (J x1 x0) (J y i)). uf F. aw. eee.
  cp (H18 _ H30 H31). cp (related_ordinal_sum_order_id H7 H32). awi H33. am.
  Ztac. Ztac. am. am. exists (P Z). red. ee. rww H13.
  cp (du_index_pr H17). ee. am.  ir.  rwi H13 H26.
  assert (inc (J x0 (Q Z)) (disjoint_union f)). app inc_disjoint_union.
  assert (gle F (J x0 (Q Z))(J x i)). uf F. aw. eee.
  assert (gle F (J x0 (Q Z)) (J y i)). uf F. aw. eee.
  cp (H18 _ H28 H29). ufi F H30. awi H30. ee.  nin H32. red in H32. ee.
  elim  H33. tv. ee. am. am. am. am. am. am.
Qed.
```

We are now ready for the main result.

```
Lemma ordinal_sum_lattice: forall r f g,
  ordinal_sum_axioms r f g ->
  lattice (ordinal_sum r f g) =
  ((lattice r) &
    (forall i j, inc i (domain f) -> inc j (domain f) ->
      (gle r i j \/ gle r j i \/ (exists u, exists v,
        least_element (V (sup r i j) g) u &
        greatest_element (V (inf r i j) g) v))) &
    (forall i x y t, inc i (domain f) -> gle (V i g) x t -> gle (V i g) y t ->
      has_supremum (V i g) (doubleton x y)) &
    (forall i x y t, inc i (domain f) -> gle (V i g) t x -> gle (V i g) t y ->
      has_infimum (V i g) (doubleton x y)) &
  (forall i x y,
      inc i (domain f) ->  inc x (V i f) ->  inc y (V i f) ->
      (forall t, inc t (V i f) -> ~ (gle (V i g) x t & gle (V i g) y t)) ->
      exists j, inc j (domain f) &
        least_element (induced_order r (Zo (domain f) (fun k=> glt r i k))) j &
        exists z, least_element  (V j g) z) &
    (forall i x y, inc i (domain f) ->  inc x (V i f) ->  inc y (V i f) ->
      (forall t, inc t (V i f) -> ~ (gle (V i g) t x & gle (V i g) t y)) ->
      exists j, inc j (domain f) &
        greatest_element (induced_order r (Zo (domain f) (fun k=> glt r k i)))
        j &
        exists z, greatest_element  (V j g) z)).
```

We start with some preliminary assertions. One half of the proof is obvious; it suffices to collect the previous results.

```
Proof. ir. assert (Ht:=H). nin H. set (F:= ordinal_sum r f g).
```

```
assert (order F). uf F. fprops. cp H; red in H;ee.
assert (Hb: substrate F = disjoint_union f). uf F. aw.
assert (Ha:forall i, inc i (domain f) -> exists y, inc y (V i f) &
  inc (J y i) (substrate F)). ir.  nin (H0 _ H9).
exists y. ee. am. cp (inc_disjoint_union H9 H10). ue.
app iff_eq. ir. cp (ordinal_sum_lattice1 Ht H9). ee. am.
ir.  app (ordinal_sum_lattice2 Ht).
ir.  app (ordinal_sum_lattice3 Ht H9 H11 H12 H13).
ir.  app (ordinal_sum_lattice4 Ht H9 H11 H12 H13).
ir.  app (ordinal_sum_lattice5 Ht H9 H11 H12 H13 H14).
ir.  app (ordinal_sum_lattice6 Ht H9 H11 H12 H13 H14).
```

Let's show the existence of a supremum.

```
ir. ee. red.  ee.  ir. am. ir. ee. red. rwi Hb H15. rwi Hb H16.
cp (du_index_pr H15). cp (du_index_pr H16). ee. wri H3 H17. wri H3 H18.
cp (lattice_sup_pr H9  H17 H18). ee. set (a:= sup r (Q x) (Q y)) in *.
assert (forall z, gle F x z -> gle F y z -> gle r a (Q z)).
ir. ufi F H26. cp (related_ordinal_sum_order_id H2 H26).
ufi F H26. cp (related_ordinal_sum_order_id H2 H27). cp (H25 _ H28 H29). am.
rwi H3 H17. rwi H3 H18.
nin (equal_or_not (Q x) (Q y)).  wri H27 H19.
nin (p_or_not_p (exists t, gle (V (Q x) g) (P x) t &gle (V (Q x) g) (P y) t)).
nin H28. nin H28.  cp (H11 _ _ _ _ H17 H28 H29).
wri H8  H21;wri H8  H19. cp (sup_pr (H7 _ H17) H21 H19 H30).
ee.  set (z:= sup (V (Q x) g) (P x) (P y)) in *.
assert (inc (J z (Q x)) (substrate F)). rw Hb. app inc_disjoint_union. wr H8.
app (inc_arg2_substrate  H31). am. exists (J z (Q x)).
ap least_upper_bound_doubleton. am. uf F. aw. eee.
uf F. aw. eee. ir.
ufi F H35; ufi F H36; awi H35; awi H36. ee. uf F. aw. eee.
nin H40. au. nin H38. left. ue. right. ee. am. app H33. ue.
am. am. am. am. am. am.
assert (forall t, inc t (V (Q x) f) -> ~ (gle (V (Q x) g) (P x) t &
  gle (V (Q x) g) (P y) t)). ir. red. ir. elim H28.  exists t.  am.
nin (H13 _ _ _ H17 H21 H19 H29). ee. nin H32. nin H32. ee.
assert (inc (J x1 x0) (disjoint_union f)). app inc_disjoint_union. wrr H8.
assert(Hc:sub (Zo (domain f) (fun k => glt r (Q x) k)) (substrate r)).
rw H3. app Z_sub.  red in H31. ee. awi H35. awi H31. Ztac.  clear H31.
exists  (J x1 x0). ap least_upper_bound_doubleton. am. uf F. aw. eee.
uf F. aw. eee. ir.
ufi F H31; ufi F H38; awi H31; awi H38. ee. uf F. aw. eee.
nin (equal_or_not x0 (Q t)). right. ee. am. app H33. cp (du_index_pr H39).
ee. rw H43. rww H8. left. nin (p_or_not_p (glt r (Q x) (Q t))). ir.
assert (inc (Q t) (Zo (domain f) (fun k  => glt r (Q x) k))). Ztac.
wr H3. order_tac. cp (H35 _ H45). awi H46.
red. ee. am. am. cp (inc_arg1_substrate H46). awi H47. am. am. am. am.
ir.  nin H42. elim H44. am. nin H40. elim H44. rww H27. ee.
cp (du_index_pr  H41).  ee. wri H42 H48. elim (H29 _ H48). ee. am. rww H27.
am. am. am. am. am. am. am.
cp (H10 _ _ H17 H18). nin (equal_or_not (Q y) a).
assert (gle F x y). uf F. aw.  eee. left. red. ee. rww H29. am.
exists y. app sup_comparable. nin (equal_or_not (Q x) a).
assert (gle F y x). uf F.  aw.  eee. left. red. ee. rww H30. intuition.
exists x. rw doubleton_symm. app sup_comparable.
nin H28. assert (gle r  (Q y) (Q y)). order_tac. ue.
cp (order_antisymmetry H H24  (H25 _ H28 H31)). elim H29; am.
```

```
nin H28. assert (gle r  (Q x) (Q x)). order_tac. ue.
cp (order_antisymmetry H H23 (H25 _ H31 H28)). elim H30. am.
nin H28; nin H28; nin H28. fold a in H28. red in H28. ee.
assert (inc (J x0 a) (disjoint_union f)). app inc_disjoint_union. wr H3.
order_tac. wrr H8. wr H3. order_tac.
exists (J x0 a).  ap least_upper_bound_doubleton. am. uf F. aw. eee.
left. red. au. uf F. aw. eee. left. red. au.
ir.  cp (H26 _ H34 H35).
ufi F H34. awi H34. ufi F H35. awi H35. uf F. aw. eee.
nin (equal_or_not a (Q t)). right. ee. am. app H32. rw H8. rw H41.
cp (du_index_pr H37). ee; am.  wr H3. order_tac.
left. order_tac am. am.
```

Let's show the existence of an infimum.

```
ee. red.  rwi Hb H15. rwi Hb H16.
cp (du_index_pr H15). cp (du_index_pr H16). ee. wri H3 H17. wri H3 H18.
cp (lattice_inf_pr H9  H17 H18). ee. set (a:= inf r (Q x) (Q y)) in *.
assert (forall z, gle F z x-> gle F z y -> gle r (Q z) a).
ir. ufi F H26. cp (related_ordinal_sum_order_id H2 H26).
ufi F H26. cp (related_ordinal_sum_order_id H2 H27). cp (H25 _ H28 H29). am.
rwi H3 H17. rwi H3 H18.
nin (equal_or_not (Q x) (Q y)). wri H27 H19.
nin (p_or_not_p (exists t, gle (V (Q x) g) t(P x) &gle (V (Q x) g) t (P y))).
nin H28. nin H28.  cp (H12 _ _ _ _ H17 H28 H29).
wri H8  H21;wri H8  H19. cp (inf_pr (H7 _ H17) H21 H19 H30).
ee.  set (z:= inf (V (Q x) g) (P x) (P y)) in *.
assert (inc (J z (Q x)) (substrate F)). rw Hb. app inc_disjoint_union. wr H8.
order_tac. ue. exists (J z (Q x)).
ap greatest_lower_bound_doubleton. am. uf F. aw. eee.
uf F. aw. eee. ir.
ufi F H35; ufi F H36; awi H35; awi H36. ee. uf F. aw. ee. am.  wrr Hb.
nin H40.  left. am. nin H38. left. rww H27. right. ee. am. rw H40. app H33.
rww H27. wr H38. am. wr H40. am. am. am. am. am. am. am.
assert (forall t, inc t (V (Q x) f) -> ~ (gle (V (Q x) g) t (P x) &
  gle (V (Q x) g) t (P y))). ir. red. ir. elim H28.  exists t.  am.
nin (H14 _ _ _ H17 H21 H19 H29). ee. nin H32. nin H32. ee.
assert (inc (J x1 x0) (disjoint_union f)). app inc_disjoint_union. wrr H8.
assert(Hc:sub (Zo (domain f) (fun k => glt r k (Q x))) (substrate r)).
rw H3. app Z_sub.  red in H31. ee. awi H35. awi H31. Ztac.  clear H31.
exists  (J x1 x0). ap greatest_lower_bound_doubleton. am. uf F. aw. eee.
uf F. aw. eee. ir.
ufi F H31; ufi F H38; awi H31; awi H38. ee. uf F. aw. ee.
nin (equal_or_not x0 (Q t)). right. ee. sy;am. wr H43. app H33.
cp (du_index_pr H38).  ee. rw H43. rww H8. left.
nin (p_or_not_p (glt r (Q t) (Q x))). ir.
assert (inc (Q t) (Zo (domain f) (fun k  => glt r k (Q x)))). Ztac.
wr H3. order_tac. cp (H35 _ H45). awi H46.
red. ee. am. intuition. am. cp (inc_arg2_substrate H46). awi H47. am. am.
am. ir.  nin H42. elim H44. am. nin H40. elim H44. rww H27. ee.
cp (du_index_pr  H31). ee. rwi H42 H48. elim (H29 _ H48). ee. wrr H42.
wrr H42.  am. am. am. am. am. am. am.
cp (H10 _ _ H17 H18). nin (equal_or_not (Q y) a).
assert (gle F y x). uf F. aw.  eee. left. red. ee. rww H29. intuition.
exists y.  rw doubleton_symm. app inf_comparable.  nin (equal_or_not (Q x) a).
assert (gle F x y). uf F.  aw.  eee.left. red. ee. rww H30. am.
exists x.  app inf_comparable.
```

```
  nin H28. assert (gle r  (Q x) (Q x)). order_tac. rww H3.
  cp (order_antisymmetry H H23  (H25 _ H31 H28)). elim H30; sy;am.
  nin H28. assert (gle r  (Q y) (Q y)). order_tac. rww H3.
  cp (order_antisymmetry H H24 (H25 _ H28 H31)). elim H29. sy;am.
  nin H28; nin H28; nin H28. fold a in H31. red in H31. ee.
  assert (inc (J x1 a) (disjoint_union f)). app inc_disjoint_union. wr H3.
  order_tac. wrr H8. wr H3. order_tac.
  exists (J x1 a).  ap greatest_lower_bound_doubleton. am. uf F. aw. eee.
  left. red. ee. am. intuition. uf F. aw. ee. am. am. left. red. au.
  ir.  cp (H26 _ H34 H35).
  ufi F H34. awi H34. ufi F H35. awi H35. uf F. aw. eee.
  nin (equal_or_not a (Q t)). right. ee. sy;am. wr H41. app H32. rw H8. rw H41.
  cp (du_index_pr H35). ee; am.   wr H3. order_tac.
  left.  order_tac. symmetry in H42. contradiction. am. am.
Qed.
```

---

4. * *Let* E *be an ordered set, and let* $(E_\iota)_{\iota\in I}$ *be the partition of* E *formed by the connected components of* E *(Chapter II, § 6, Exercise 10) with respect to the reflexive and symmetric relation "either $x = y$ or x and y are not comparable".*

(a) Show that if $\iota \neq \kappa$ *and if* $x \in E_\iota$ *and* $y \in E_\kappa$*, then x, y are comparable; and that if, for example,* $x \leq y$, $y' \in E_\kappa$*, and if* $y' \neq y$*, then also* $x \leq y'$ *(use the fact that there exists no partition of* E_κ *into two sets* A *and* B *such that every element of* A *is comparable with every element of* B*).*

(b) Deduce from (a) that the equivalence relation S *corresponding to the partition* (E_ι) *of* E *is compatible (in x and y) with the order relation* $x \leq y$ *on* E*, and that the quotient ordered set* E/S *(Exercise 2) is totally ordered.*

(c) What are the connected components of an ordered set $E = F \times G$ *which is the product of two totally ordered sets?* *

We first recall some facts about connected components. We shall denote by R the relation "either $x = y$ or x and y are not comparable", and by S the equivalence relation associated; the sets E_ι are the equivalence classes of S.

```
Definition not_comp_rel r := fun x y =>
  inc x (substrate r) & inc y (substrate r) &
  x = y \/ ~ ((gle r x y) \/ (gle r y x)).

Definition ncr_equiv r :=
  Exercice1.Sgraph (not_comp_rel r) (substrate r).
Definition ncr_component r :=
  Exercice1.connected_comp (not_comp_rel r) (substrate r).

Lemma ncr_properties: forall r, order r ->
  (is_equivalence (ncr_equiv r) &
  substrate  (ncr_equiv r) =  substrate r &
  (forall x, inc x (substrate r) -> class (ncr_equiv r) x = ncr_component r x)&
  (forall x y, not_comp_rel r x y -> related (ncr_equiv r) x y)).
Proof. ir. uf ncr_equiv. uf ncr_component.
  assert (reflexive_r (not_comp_rel r) (substrate r)).
  red. ir. uf not_comp_rel. app iff_eq. ir. eee.
```

```
  ir. ee. am.
  assert (symmetric_r (not_comp_rel r)). red. uf not_comp_rel. ir. ee.
  nin H3. left. sy; am. right. dneg. intuition.
  uf not_comp_rel.
  assert (forall x y, not_comp_rel r x y ->  inc x (substrate r)).
  ir. red in H2. ee; am.
  ee. app Exercice1.equivalence_Sgraph. app Exercice1.substrate_Sgraph.
  ir. app  Exercice1.connected_comp_class.
  ir. red. uf Exercice1.Sgraph. uf graph_on. ee. Ztac. fprops. red.
  aw. exists (Exercice1.chain_pair x y). eee. simpl. intuition.
Qed.
```

We restate "$\iota \neq \kappa$ and if $x \in E_\iota$ and $y \in E_\kappa$, then x, y are comparable" as: either the classes (mod S) of x and y are equal, or x, y are comparable. This can also be restated as $R\{x, y\}$ implies $S\{x, y\}$.

```
Lemma Exercice1_4a1: forall r x y, order r ->
  inc x (substrate r) -> inc y (substrate r) ->
  gle r x y \/ gle r y x \/ class (ncr_equiv r) x = class (ncr_equiv r) y.
Proof. ir. nin (p_or_not_p ((gle r x y) \/ (gle r y x))). nin H2. au.
  au. right. right. assert (not_comp_rel r x y). red. eee.
  cp (ncr_properties H). ee. wrr related_class_eq. app H7.
  app reflexivity_e. rww H5.
Qed.
```

Consider the hint of the second claim. Consider a class C modulo S that is the union of two sets A and B such that each element of A is comparable with each element of B. Assume neither set empty, let $a \in A$ and $b \in B$. Then a and b are related by S. This means that there is a chain $(x_i)_i$, relating a and b, thus a chain $(x_i)_i$ with head in A and tail in B; by induction, there exist elements $a' \in A$ and $b' \in B$ related by R (if a chain is non-trivial, its head a' is related to another chain with head b'; these elements are in the same equivalence class (namely C) because they are related by R; if $b' \in A$, the result follows by induction, otherwise $b' \in B$ and we have a solution). Since a' and b' are comparable, and related by R, they are equal. Thus the intersection of A and B is nonempty.

```
Lemma Exercice1_4a2: forall r y, order r -> inc y (substrate r) ->
  forall a b, union2 a b = class (ncr_equiv r) y ->
    (forall u v, inc u a -> inc v b -> gle r u v \/ gle r v u) ->
    a = emptyset \/ b = emptyset \/ nonempty (intersection2 a b).
Proof. ir. nin (emptyset_dichot a). left. am. right. nin (emptyset_dichot b).
  ue. right. nin H3. nin H4. cp (ncr_properties H). ee.
  assert (Ha:is_class (ncr_equiv r) (union2 a b)). rw H1.
  app is_class_class. ue.
  assert (related (ncr_equiv r) y0 y1). rw in_class_related.
  exists (union2 a b). ee. am. app union2_first. app union2_second. am.
  red in H9. ufi ncr_equiv H9. ufi Exercice1.Sgraph H9. ufi graph_on H9. Ztac.
  awi H11. red in H11. nin H11. ee.
  assert (inc (Exercice1.chain_head x) a). ue. clear H12.
  assert (inc (Exercice1.chain_tail x) b). ue. clear H13.

  assert (exists u, exists v, inc u a & inc v b & (not_comp_rel r) u v).
  nin x. simpl in H11. simpl in H12. simpl in H14. exists P. exists P0.
  intuition. simpl in H11. simpl in H14. simpl in H12. ee.
  nin (inc_or_not (Exercice1.chain_head x) a). ir. app IHx.
  ir. set (v:= (Exercice1.chain_head x)) in *.
```

```
  cp (H8 _ _ H11). rwi is_class_rw Ha. ee. wri H19 H16. assert (inc v b).
  nin (union2_or H16). elim H15. ee. am. am. exists P. exists v.
  intuition. app union2_first. am. nin H13. nin H13. ee. red in H16.
  ee. nin H18. exists x0. app intersection2_inc. rww H18. elim H18. app H2.
Qed.
```

Let C be a class for S, $x \in E - C$, A the set of all $y \in C$ such that $x \leq y$, B the set of $y' \in C$ such that $y' \leq x$. We have shown $C = A \cup B$. Obviously, if $y \in A$ and $y' \in B$ then $y' \leq y$, and $y \neq y'$ (this would imply $y = x$ hence $x \in C$). As a consequence one of A and B is empty.

```
Lemma Exercice1_4a3: forall r x y y', order r ->
  inc x (substrate r) -> related (ncr_equiv r) y y' -> gle r x y ->
  related (ncr_equiv r) x y \/ gle r x y'.
Proof. ir.  cp (ncr_properties H). ee. set (C:= class (ncr_equiv r) y).
  set (A:=Zo C (fun z => gle r x z)). set (B:=Zo C (fun z => gle r z x)).
  assert (forall u v, inc u A -> inc v B -> gle r u v \/ gle r v u).
  ir. ufi A H7. Ztac.  clear H7. ufi B H8. Ztac. right. order_tac.
  nin (p_or_not_p (related (ncr_equiv r) x y)). left. am. right.
  assert(inc y (substrate r)). app (inc_arg2_substrate H2).
  assert (union2 A B = class (ncr_equiv r) y). set_extens. nin (union2_or H10).
  ufi A H11. Ztac. am. ufi B H11. Ztac. am.
  assert (inc x0 (substrate r)). wr H4. app (sub_class_substrate H3 H10).
  nin (Exercice1_4a1 H H0 H11). app union2_first. uf A. Ztac.
  nin H12. app union2_second. uf B. Ztac. elim H8.
  apply transitivity_e with x0. am. rww related_class_eq.
  app reflexivity_e. rww H4. app symmetricity. wr inc_class.  am.  am.
  cp (Exercice1_4a2 H H9 H10 H7). nin H11.
  elim (emptyset_pr (x:=y)). wr H11. uf A. Ztac. uf C. bw.
  app reflexivity_e.  rww H4.
  nin H11. assert (inc y' C). uf C. bw.
  ufi C H12. wri H10 H12. nin (union2_or H12). ufi A H13. Ztac. am.
  rwi H11 H13. elim (emptyset_pr H13). nin H11.
  cp (intersection2_first H11). ufi A H12. Ztac. clear H12.
  cp (intersection2_second H11). ufi B H12. Ztac. elim H8.
  rw (order_antisymmetry H H14 H16). app symmetricity. wr inc_class. am. am.
Qed.
```

Let's show the compatibility of the equivalence and the order. We must show that if x and x' are in the same class, if y and y' are in the same class, then $x \leq y$ is the same as $x' \leq y'$. The assumption is that the classes are distinct. We know that $x' \leq y'$, $y' \leq x'$ or $R(x', y')$. The first possibility is the desired result, the last one says that the classes are the same. We have shown that $y' \leq x'$ implies $y' \leq x$ and $x \leq y$ implies $x \leq y'$. Thus, the second possibility says $x = y'$, absurd.

```
Lemma Exercice1_4b1: forall r x y x' y', order r ->
  related (ncr_equiv r) x x' -> related (ncr_equiv r) y y' ->
  class (ncr_equiv r) x <> class (ncr_equiv r) y ->
  gle r x y -> gle r x' y'.
Proof. ir. cp (ncr_properties H). ee.
  assert (inc x' (substrate r)). wr H5. app (inc_arg2_substrate H0).
  assert (inc y' (substrate r)). wr H5. app (inc_arg2_substrate H1).
  assert (inc x (substrate r)). wr H5. app (inc_arg1_substrate H0).
  assert (inc y (substrate r)). wr H5. app (inc_arg1_substrate H1).
  nin (Exercice1_4a1 H H8 H9). am. nin H12.
  assert (Ha:related (ncr_equiv r) x x). app reflexivity_e. rww H5.
```

```
  nin (Exercice1_4a3 H H10 H1 H3). elim H2. wrr related_class_eq.
  assert (related (ncr_equiv r) x' x).  app symmetricity.
  nin (Exercice1_4a3 H H9 H14 H12). elim H2.
  wrr related_class_eq. apply transitivity_e with  x'. am. am.
  app symmetricity.  apply transitivity_e with y'. am . am. am.
  wri (order_antisymmetry H H13 H15) H1. elim H2. wrr related_class_eq.
  app symmetricity.  elim H2. wrr related_class_eq.
  apply transitivity_e with x'. am. am. apply transitivity_e with y'. am.
  rww related_class_eq.  app reflexivity_e. rww H5. app symmetricity.
  app reflexivity_e. rww H5.
Qed.
```

The quotient order is an order, according to condition C'. To show it, assume $x \leq y \leq z$, x and z in the same class. If x and y are not in the same class, then $x \leq y$ implies $z \leq y$. This says $y = z$, absurd. The quotient order is total, since, given two distinct classes, the representatives are comparable. Now, if $x \in \mathrm{X}$ and $y \in \mathrm{Y}$, the compatibility condition says $\mathrm{X} \leq \mathrm{Y}$ in the quotient if and only if $x \leq y$ in the substrate.

```
Lemma Exercice1_4b: forall r, order r ->
  total_order(quotient_order r (ncr_equiv r)).
Proof. ir. cp (ncr_properties H). ee.
  assert (substrate (quotient_order r (ncr_equiv r)) = quotient (ncr_equiv r)).
  rww substrate_quotient_order. app order_is_preorder.
  assert (order (quotient_order r (ncr_equiv r))).
  app Exercise1_2d. red. ir. nin (p_or_not_p (related (ncr_equiv r) x y)).
  am. assert (related (ncr_equiv r) y y). app reflexivity_e. rww H1.
  order_tac.
  assert (class (ncr_equiv r) x <> class (ncr_equiv r) y). dneg.
  app symmetricity. rww related_class_eq. sy; am.
  cp (Exercice1_4b1 H H7 H9 H10 H5). rww (order_antisymmetry H H6 H11).
  red. ee. am. rw H4. ir. uf gge.
  nin (equal_or_not x y). left. rw H8. order_tac. ue.
  rw quotient_order_pr.  rw quotient_order_pr. uf quotient_order_r.
  cp (inc_rep_itself H0 H6).  cp (inc_rep_itself H0 H7).
  assert (inc (rep x) (substrate r)). wr H1. app inc_rep_substrate.
  assert (inc (rep y) (substrate r)). wr H1. app inc_rep_substrate.
  assert (Ha:class (ncr_equiv r) (rep x) <> class (ncr_equiv r) (rep y)).
  red. ir. rwi (inc_quotient x H0) H6.  rwi (inc_quotient y H0) H7. elim H8.
  red in H6; red in H7. ee. rw H17; rw H15. am.
  cp (Exercice1_4a1 H H11 H12). nin H13. left. eee. ir.
  exists (rep y). ee. am.
  assert (related (ncr_equiv r) (rep x) x0).
  app related_rep_in_class. assert (related (ncr_equiv r) (rep y)(rep y)).
  app related_rep_in_class. ap (Exercice1_4b1 H H15 H16 Ha H13).
  nin H13. right. eee. ir.  exists (rep x). ee. am.
  assert (related (ncr_equiv r) (rep y) x0).
  app related_rep_in_class. assert (related (ncr_equiv r) (rep x)(rep x)).
  app related_rep_in_class.
  assert (class (ncr_equiv r) (rep y) <> class (ncr_equiv r) (rep x)).
  intuition. ap (Exercice1_4b1 H H15 H16 H17 H13).  contradiction.
Qed.
```

Consider a product of two totally ordered sets $\mathrm{E} = \mathrm{F} \times \mathrm{G}$. Let's determine the set of components. We have $x \leq y$ if and only if $\mathrm{pr}_1 x \leq \mathrm{pr}_1 y$ and $\mathrm{pr}_2 x \leq \mathrm{pr}_2 y$. We exclude the case where the sets are empty. If G is a singleton, then $\mathrm{pr}_2 x \leq \mathrm{pr}_2 y$ is always true, and the product is

totally ordered. The set of components is then isomorphic to E. If a and a' are the least elements of F and G, $l = (a, a')$, then l is the least element of E, and $\{l\}$ is a component. In the same fashion, if g is the greatest element of E then $\{g\}$ is a component.

We assume now that E has at least two elements b and c, F has at least two elements b' and c'. We pretend that the class C of (b, c'), contains all elements of E, with the possible exception of l and g. It contains obviously (c, b'). 2 or 3 components. Let $y = (b, b')$, and assume that $y \neq l$. Then $y \in$ C. In fact, if $a < b$, then R$\{(a, c'), (b, b')\}$ and R$\{(a, c'), (c, b')\}$, so that (b, b') and (c, b') are in the same class. On the other hand, if $a' < b'$, then R$\{(c, a'), (b, b')\}$ and R$\{(c, a'), (b, c')\}$, so that (b, b') and (b, c') are in the same class. In the same way, (c, c') is the class if it is not the greatest element. Assume $c < d$. Then (c, b') and (d, b') are related to (b, c'), Then (c, b') and (d, b') are related to (b, c'), and (b, c') and (c, c') are related to (d, b'). This shows that the six elements of the product $\{b, c, d\} \times \{b', c'\}$, minus the greatest and least elements are in the same class. By symmetry, if $d' \in$ F, the six elements of the product $\{b, c, d\} \times \{b', c', d'\}$, minus the greatest and least elements are in the same class.

It follows (the Coq code is still missing), that $x \in$ E and $y \in$ E are related if neither of them is the least or greatest element of E.

---

5. *Let* E *be an ordered set. A subset* X *of* E *is said to be* free *if no two distinct elements of* X *are comparable. Let* $\mathfrak{I}$ *be the set of free subsets of* E. *Show that, on* $\mathfrak{I}$, *the relation "given any* $x \in$ X, *there exists* $y \in$ Y *such that* $x \leq y$*" is an order relation between* X *and* Y, *written* X $\leq$ Y. *The mapping* $x \to \{x\}$ *is an isomorphism of* E *onto a subset of the ordered set* $\mathfrak{I}$. *If* X $\subset$ Y *where* X $\in \mathfrak{I}$ *and* Y $\in \mathfrak{I}$, *show that* X $\leq$ Y. *The ordered set* $\mathfrak{I}$ *is totally ordered if and only if* E *is totally ordered, and then* $\mathfrak{I}$ *is canonically isomorphic to* E.

We restate the definition as: if $x \in$ X and $y \in$ X and $x \leq y$ then $x = y$.

```
Definition free_subset r X := forall x y, inc x X -> inc y X ->
  gle r x y -> x = y.
Definition set_of_free_subsets r:=
  Zo (powerset (substrate r)) (fun X=> free_subset r X).
Definition free_subset_compare r X Y:=
  inc X (set_of_free_subsets r) & inc Y (set_of_free_subsets r) &
  forall x, inc x X -> exists y, inc y Y & gle r x y.
Definition free_subset_order r:=
  graph_on (free_subset_compare r) (set_of_free_subsets r).
```

The relation $\forall x \in$ X, $\exists y \in$ Y, $x \leq y$ is clearly a preorder relation. Antisymmetry follow from the fact that if $x \in$ X, $y \in$ Y, $x \leq y$ and $x' \in$ X, $y \leq x'$ w have $x \leq x'$ by transitivity, then $x = x'$ when X is free, and $x = y$ by antisymmetry.

```
Lemma Exercise1_5a: forall r, order r ->
  order_r (free_subset_compare r).
Proof. ir. red. uf free_subset_compare. split. red. ir. eee.
  ir. nin (H5 _ H6).  nin H7. nin (H3 _ H7). nin H9. ex_tac. order_tac.
  split. red. ir. ee.
  set_extens; [nin (H5 _ H6) |  nin (H3 _ H6)]; nin H7;
  [nin (H3 _ H7) |  nin (H5 _ H7) ]; nin H9; cp (order_transitivity H H8 H10);
  [ ufi set_of_free_subsets H0 |ufi set_of_free_subsets H4];  Ztac;
   wri (H13 _ _ H6 H9 H11) H10; rww (order_antisymmetry H  H8 H10).
  red. ir. eee. ir. ex_tac. order_tac.
```

```
  ufi set_of_free_subsets H0. Ztac. rwi powerset_inc_rw H4. app H4.
  ir. ex_tac. order_tac.
  ufi set_of_free_subsets H1. Ztac. rwi powerset_inc_rw H4. app H4.
Qed.

Lemma fs_order_pr: forall r x y,
  related (free_subset_order r) x y = free_subset_compare r x y.
Proof. ir. uf free_subset_order. uf graph_on. uf related. ap iff_eq. ir. Ztac.
  awi H1. am. ir. Ztac. red in H. ee. fprops. aw.
Qed.

Lemma fs_is_order: forall r,
  order r -> order (free_subset_order r).
Proof. ir. uf free_subset_order. app order_from_rel. app Exercise1_5a.
Qed.

Lemma substrate_fs_order: forall r,
  order r -> substrate (free_subset_order r) = set_of_free_subsets r.
Proof. ir. cp (fs_is_order H). set_extens. rwi order_reflexivity H1.
  rwi fs_order_pr H1. red in H1. nin H1. am.
  am. rw order_reflexivity. rw fs_order_pr. red. split. am. split. am.
  ir. exists x0. split. am. wr order_reflexivity.  ufi set_of_free_subsets H1.
  Ztac. rwi powerset_inc_rw H3. app H3. am. am.
Qed.
```

A singleton is free; the mapping $x \mapsto \{x\}$ is an isomorphism onto its range. If E is a totally ordered set, then the only nonempty-free subsets are the singletons, so that the range is $\mathfrak{I} - \{\emptyset\}$.

```
Lemma Exercise1_5b: forall r x, order r ->
  inc x (substrate r) -> inc (singleton x) (set_of_free_subsets r).
Proof. ir. uf set_of_free_subsets. Ztac. app powerset_inc. red. ir.
  rw (singleton_eq H1). am. red. ir. rw (singleton_eq H1).
  rww (singleton_eq H2).
Qed.

Lemma Exercise1_5c: forall r x y, order r ->
  inc x (substrate r) ->inc y (substrate r) ->
  gle r x y = gle (free_subset_order r) (singleton x) (singleton y).
Proof.  ir. rw fs_order_pr. app iff_eq. ir. red. ee. app Exercise1_5b.
  app Exercise1_5b. ir. rw (singleton_eq H3). exists y. split. fprops. am.
  ir. red in H2. ee. assert (inc x (singleton x)). fprops. cp (H4 _ H5). nin H6.
  nin H6. wrr (singleton_eq H6).
Qed.
Lemma Exercise1_5d: forall r, order r ->
  order_morphism (BL singleton (substrate r)  (set_of_free_subsets r))
  r (free_subset_order r).
Proof. ir.
  assert (transf_axioms singleton (substrate r) (set_of_free_subsets r)).
  red. ir. app Exercise1_5b.
  assert (is_function (BL singleton (substrate r) (set_of_free_subsets r))).
  app bl_function. red. aw. ee. am. app fs_is_order.
  app injective_bl_function.  ir. app singleton_inj. tv. rww substrate_fs_order.
  ir. rww W_bl_function. rww W_bl_function. app Exercise1_5c.
Qed.
```

```
Lemma Exercise1_5e: forall r X, total_order r ->
  inc X (set_of_free_subsets r)-> small_set X.
Proof. ir. red. ir.  red in H. ee. ufi set_of_free_subsets H0. Ztac.
  rwi powerset_inc_rw H4. red in H5.  nin (H3 _ _ (H4 _ H1) (H4 _ H2)).
  app H5. red in H6. sy. app H5.
Qed.
```

If $X \subset Y$ then $X \leq Y$ (this is trivial). If E is totally ordered so is $\mathfrak{I}$. In fact, the empty set is the least element of $\mathfrak{I}$; two non-empty sets are singletons, and singletons are compared according to there elements. The converse is trivial. Bourbaki says that $\mathfrak{I}$ is canonically isomorphic to E in this case. This is obviously wrong: as noted above $x \mapsto \{x\}$ is an isomorphism between E and $\mathfrak{I} - \{\emptyset\}$.

```
Lemma Exercise1_5f: forall r X Y, order r ->
  inc X (set_of_free_subsets r) -> inc Y (set_of_free_subsets r) ->
  sub X Y ->  gle (free_subset_order r) X Y.
Proof. ir. rw fs_order_pr. red. eee. ir. exists x. split. app H2.
  wr order_reflexivity. ufi set_of_free_subsets H0. Ztac.
  rwi powerset_inc_rw H4. app H4. am.
Qed.

Lemma Exercise1_5g: forall r, total_order r ->
  total_order (free_subset_order r).
Proof. ir. cp H. nin H. red. split. app fs_is_order. rww substrate_fs_order.
  assert (inc emptyset (set_of_free_subsets r)). uf set_of_free_subsets.
  Ztac. app powerset_inc. app sub_emptyset_any. red. ir. elim (emptyset_pr H2).
  ir. cp (Exercise1_5e H0 H3). cp (Exercise1_5e H0 H4).
  nin (emptyset_dichot x). rw H7. left. app  Exercise1_5f. app sub_emptyset_any.
  nin (emptyset_dichot y). rw H8. right. red. app  Exercise1_5f.
  app sub_emptyset_any.  nin H7. nin H8.
  ufi set_of_free_subsets H3; Ztac. clear H3. rwi powerset_inc_rw H9.
  ufi set_of_free_subsets H4; Ztac. rwi powerset_inc_rw H3.
  cp (H9 _ H7). cp (H3 _ H8).
  assert (x = singleton y0). set_extens. red in H5. rw (H5 _ _ H7 H14). fprops.
  rw (singleton_eq H14). am.
  assert (y = singleton y1). set_extens. red in H6. rw (H6 _ _ H8 H15). fprops.
  rw (singleton_eq H15). am.  rw H14; rw H15.
  nin (H1 _ _ H12 H13). left. wrr Exercise1_5c. right. red. wrr Exercise1_5c.
Qed.

Lemma Exercise1_5h: forall r, order r ->
  total_order (free_subset_order r) -> total_order r.
Proof. ir. red. split. am. nin H0. ir. rwi substrate_fs_order H1.
  nin (H1 _ _  (Exercise1_5b H H2)  (Exercise1_5b H H3)). left.
  rww Exercise1_5c. right.  red. red in H4. rww Exercise1_5c. am.
Qed.
```

---

6. *Let* E *and* F *be two ordered sets and let* $\mathscr{A}$(E,F) *be the subset of the product ordered set* F^E *consisting of the* increasing *mappings of* E *into* F.

We change the definition of $\mathscr{A}$(E,F): it will be a subset of $\mathscr{F}$(E;F), the set of mappings from E to F, that is canonically isomorphic to F^E. It will be ordered by the ordering on functions.

```
Definition set_of_increasing_mappings r r' :=
  Zo (set_of_functions (substrate r) (substrate r'))
  (fun z=> increasing_fun z r r').
Definition increasing_mappings_order r r' :=
  induced_order (function_order (substrate r) (substrate r') r')
  (set_of_increasing_mappings r r').
```

We give here some trivial lemmas expliciting the definitions.

```
Lemma set_of_increasing_mappings_pr: forall r r' f,  order r -> order r' ->
  inc f (set_of_increasing_mappings r r') =
    (is_function f & source f = (substrate r) & target f = substrate r'
    & increasing_fun f r r').
Proof. ir. uf set_of_increasing_mappings. app iff_eq. ir. Ztac. awi H2. eee.
  ir. ee. Ztac. aw. eee.
Qed.
Hint Rewrite  set_of_increasing_mappings_pr : aw.

Lemma imo_order: forall r r', order r -> order r' ->
  order (increasing_mappings_order r r').
Proof. ir. uf increasing_mappings_order.  app order_induced_order.
  rw substrate_function_order. uf set_of_increasing_mappings. app Z_sub. am.
  tv. app order_function_order.
Qed.
Lemma imo_substrate: forall r r', order r -> order r' ->
  substrate (increasing_mappings_order r r') = set_of_increasing_mappings r r'.
Proof. ir. uf increasing_mappings_order.  rw substrate_induced_order. tv.
  app order_function_order. uf set_of_increasing_mappings.
  rw substrate_function_order. app Z_sub. am. tv.
Qed.

Lemma imo_pr: forall r r' f g, order r -> order r' ->
  gle (increasing_mappings_order r r') f g =
  (inc f (set_of_increasing_mappings r r') &
    inc g (set_of_increasing_mappings r r') &
    function_order_r (substrate r) (substrate r') r' f g).
Proof. ir. app iff_eq. ir. cp (imo_order H H0).
  cp (inc_arg1_substrate H1). cp (inc_arg2_substrate H1). rwi imo_substrate H3.
  rwi imo_substrate H4. ufi increasing_mappings_order H1. awi H1.
  ufi function_order H1. ufi graph_on H1. eee. red in H1. red in H1. Ztac.
  clear H1. awi H5. ee. awi H6. am.  am. am.  am. am. am. am.
  ir. ee. uf increasing_mappings_order. aw.
  ufi set_of_increasing_mappings H1. Ztac. clear H1.
  ufi set_of_increasing_mappings H2. Ztac. clear H2.
  uf function_order. red. red. uf graph_on. Ztac. aw. awi H4. awi H1. eee. aw.
Qed.
```

(a) Show that if E, F, G *are three ordered sets, then the ordered set* $\mathscr{A}(E, F \times G)$ *is isomorphic to the product ordered set* $\mathscr{A}(E,F) \times \mathscr{A}(E,G)$.

Given a function $f : E \to F \times G$, we can consider the two projections $f_1 : E \to F$ and $f_2 : E \to G$. We first show that $f \mapsto (f_1, f_2)$ is a bijection $\mathscr{F}(E; F \times G)$ onto $\mathscr{F}(E;F) \times \mathscr{F}(E;G)$.

```
Definition first_projection f a b:= BL (fun z=> P (W z f)) a b.
```

```
Definition secnd_projection f a c:= BL (fun z=> Q (W z f)) a c.
Definition two_projections a b c :=
  BL (fun z => (J (first_projection z a b)
    (secnd_projection z a c)))
  (set_of_functions a (product b c))
  (product (set_of_functions a b) (set_of_functions a c)).

Lemma Exercice1_6a: forall f a b c, is_function f -> source f =a ->
  target f = product b c ->
  (transf_axioms (fun z=> P (W z f)) a b &
    transf_axioms (fun z=> Q (W z f)) a c &
    is_function (first_projection f a b) &
    is_function (secnd_projection f a c) &
    (forall x, inc x a -> W x (first_projection f a b) = P (W x f)) &
    (forall x, inc x a -> W x (secnd_projection f a c) = Q (W x f))).
Proof. ir. assert (transf_axioms (fun z=> P (W z f)) a b). red. ir. wri H0 H2.
  cp (inc_W_target H H2). rwi H1 H3. awi H3; eee.
  assert (transf_axioms (fun z=> Q (W z f)) a c). red. ir. wri H0 H3.
  cp (inc_W_target H H3). rwi H1 H4. awi H4; eee. uf first_projection.
  uf secnd_projection. ee. am. am. app bl_function. app bl_function.
  ir. aw. ir. aw.
Qed.

Lemma Exercice1_6b: forall a b c,
  transf_axioms
    (fun z => (J (first_projection z a b)
      (secnd_projection z a c)))
    (set_of_functions a (product b c))
    (product (set_of_functions a b) (set_of_functions a c)).
Proof. ir.  red. ir. awi H. ee.
  cp (Exercice1_6a H H0 H1). ee. aw. split. fprops.
  eee; uf first_projection; uf secnd_projection; aw.
Qed.

Lemma Exercice1_6c: forall a b c, bijective  (two_projections a b c).
Proof. ir. cp (Exercice1_6b (a:=a) (b:=b)(c:=c)).
  uf two_projections. app bijective_bl_function.
  (* injectivity *)
  ir. awi H0. awi H1. cp (pr1_injective H2). cp (pr2_injective H2). ee.
  app funct_extensionality. ue. ue. rw H7. ir.
  cp (Exercice1_6a H0 H7 H8). cp (Exercice1_6a H1 H5 H6). ee.
  app pair_extensionality. assert (inc (W x u) (product b c)).  wr H8.
  app inc_W_target. ue.  awi H22; eee.
  assert (inc (W x v) (product b c)).  wr H6. app inc_W_target. ue.
  awi H22; eee. wr H20. wr H15. rww H3. am. am.
  wr H16. wr H21. rww H4. am. am.
  (* surjectivity *) ir. awi H0. ee.
  set (f:= BL (fun z=> J (W z (P y)) (W z (Q y))) a (product b c)).
  assert (transf_axioms (fun z=> J (W z (P y)) (W z (Q y))) a (product b c)).
  red. ir. aw. ee. fprops. wr H6. app inc_W_target. ue.
  wr H4. app inc_W_target. ue.
  assert (is_function f). uf f. app bl_function.
  assert (source f = a).  uf f. aw.  assert (target f = product b c). uf f. aw.
  cp (Exercice1_6a H8 H9 H10). ee. exists f. ee. aw. eee. sy.
  app pair_extensionality. fprops. aw. app funct_extensionality.
  uf first_projection. aw. uf first_projection. aw. rw H5. ir. rww H15.
```

```
  uf f. aw. aw. app funct_extensionality. uf secnd_projection. aw.
  uf secnd_projection. aw. rw H3. ir. rww H16. uf f. aw.
Qed.
```

Now we show that this operation is compatible with the order: if f is increasing, both projections are increasing; the converse is equally true. This means that $f \mapsto (f_1, f_2)$ induces a bijection $\mathscr{A}(E, F \times G)$ onto $\mathscr{A}(E, F) \times \mathscr{A}(E, G)$. Finally, we show that it is an order isomorphism.

```
Hint Rewrite substrate_order_product2_order: bw.

Lemma Exercice1_6d: forall f r r' r'', order r -> order r' -> order r'' ->
  increasing_fun f r (product2_order r' r'') ->
  (increasing_fun (first_projection f (substrate r) (substrate r')) r r' &
    increasing_fun (secnd_projection f (substrate r) (substrate r'')) r r'').
Proof. ir. red in H2. nin H2; nin H3; nin H4; nin H5; nin H6.
  symmetry in H5; symmetry in H6. bwi H6.
  cp (Exercice1_6a H2 H5 H6). ee. red. eee. uf first_projection. aw.
  uf first_projection. aw.
  ir. rw H12. rw H12. cp (H7 _ _ H14). rwi product2_order_pr H15.
  red in H15. ee. am.  order_tac. order_tac.
  red. eee. ir. uf secnd_projection. aw. uf secnd_projection. aw. ir.
  rw H13. rw H13. cp (H7 _ _ H14).
  rwi product2_order_pr H15. red in H15. ee. am.  order_tac. order_tac. am. am.
Qed.

Definition two_projections_increasing r r' r'' :=
  restriction2 (two_projections (substrate r) (substrate r')(substrate r''))
  (set_of_increasing_mappings r (product2_order r' r''))
  (product (set_of_increasing_mappings r r')
    (set_of_increasing_mappings r r'')).

Lemma Exercice1_6e: forall  r r' r'', order r -> order r' -> order r'' ->
  (restriction2_axioms
    (two_projections (substrate r) (substrate r')(substrate r''))
    (set_of_increasing_mappings r (product2_order r' r''))
    (product (set_of_increasing_mappings r r')
      (set_of_increasing_mappings r r'')))).
Proof. ir.
  cp  (Exercice1_6c (substrate r) (substrate r') (substrate r'')).
  cp (Exercice1_6b  (a:=substrate r) (b:=substrate r') (c:=substrate r'')).
  red. ee. fct_tac. uf two_projections. aw. uf set_of_increasing_mappings.
  bw. app Z_sub.  uf two_projections. aw.
  app product_monotone. uf set_of_increasing_mappings. app Z_sub.
  uf set_of_increasing_mappings. app Z_sub.
  uf image_by_fun. red. ir. cp (bij_is_function H2). awi H4. nin H4. nin H4.
  wr (W_pr H5 H6). uf two_projections. awi H4. ee.
  assert (inc x0
   (set_of_functions (substrate r) (product (substrate r') (substrate r'')))).
  aw. eee. bw.  rww W_bl_function.
  nin (Exercice1_6d H H0 H1 H9). bwi H8.
  cp (Exercice1_6a H4 H7 H8). ee. aw. eee.
  uf first_projection; aw. uf first_projection; aw.
  uf secnd_projection; aw. uf secnd_projection; aw. am. am. am.
  app order_product2_order.
Qed.
```

```
Lemma Exercice1_6f: forall  r r' r'', order r -> order r' -> order r'' ->
  bijective (two_projections_increasing r r' r'').
Proof. ir.  cp (Exercice1_6e H H0 H1).
  cp (Exercice1_6c (substrate r) (substrate r') (substrate r'')). nin H3.
  assert (is_function (two_projections_increasing r r' r'')).
  uf two_projections_increasing. app function_restriction2.
  red. split. uf two_projections_increasing. app injective_restriction2.
  uf two_projections_increasing. app surjective_pr6. ir.
  assert (inc y (target  (two_projections (substrate r) (substrate r')
    (substrate r'')))). ufi restriction2 H6. awi H6. uf two_projections. aw.
  eee. am. am. am. am. nin (surjective_pr2 H4 H7). nin H8.
  ufi two_projections H8. awi H8. aw. ee.
  cut (inc x (set_of_increasing_mappings r (product2_order r' r'')). ir.
  exists x. split. uf restriction2. rww corresp_source. rww W_restriction2.
  ufi restriction2 H6.  rwi corresp_target H6.
  ufi two_projections H9. rwii W_bl_function H9. aw. eee. bw. awi H6. ee.
  wri H9 H16.  awi H16. wri H9 H19. awi H19.
  cp (Exercice1_6a H8 H10 H11). ee.  red. ir. eee. app order_product2_order.
  bw. ir. rw product2_order_pr. red. wr H11.
  assert (inc x0 (substrate r)). order_tac.
  assert (inc y0 (substrate r)). order_tac. ee. app inc_W_target.  ue.
  app inc_W_target. ue. wrr H24. wrr H24. red in H19. ee. app H33.
  wrr H25. wrr H25. red in H16. ee. app H33. am. am. am. am.
  app order_product2_order. app Exercice1_6b. aw. eee.
Qed.

Lemma Exercice1_6g: forall  r r' r'', order r -> order r' -> order r'' ->
  order_isomorphism (two_projections_increasing r r' r'')
  (increasing_mappings_order r (product2_order r' r''))
  (product2_order (increasing_mappings_order r r')
    (increasing_mappings_order r r'')).
Proof. ir.  assert (order (product2_order r' r'')).  app order_product2_order.
  red. ee. app imo_order. app order_product2_order. app imo_order.
  app imo_order. app Exercice1_6f. uf two_projections_increasing.
  uf restriction2. aw. rww imo_substrate. bw. uf two_projections_increasing.
  uf restriction2. aw.  rww imo_substrate. rww imo_substrate.
  app imo_order. app imo_order. uf two_projections_increasing. ir.
  ufi restriction2 H3. ufi restriction2 H4.  rwi corresp_source H3.
  rwi corresp_source H4.  assert (Ha:= H3). assert (Hb:= H4). awi H3. awi H4.
  ee.  uf two_projections_increasing.
  cp (Exercice1_6b  (a:=substrate r) (b:=substrate r') (c:=substrate r'')).
  cp (Exercice1_6e H H0 H1). rww W_restriction2. rww W_restriction2. clear H12.
  bwi H9. cp (Exercice1_6a H3 H8 H9).  bwi H6. cp (Exercice1_6a H4 H5 H6). ee.
  uf two_projections. rww W_bl_function. rww W_bl_function.
  clear H11. rw product2_order_pr. uf product2_order_r. aw. do 3 rww imo_pr.
  set (f1 := first_projection x (substrate r) (substrate r')) in *.
  set (f2 := secnd_projection x (substrate r) (substrate r'')) in *.
  set (f3 := first_projection y (substrate r) (substrate r')) in *.
  set (f4 := secnd_projection y (substrate r) (substrate r'')) in *.
  repeat rww imo_substrate.
  cp (Exercice1_6d H H0 H1 H10). cp (Exercice1_6d H H0 H1 H7). ee. app iff_eq.
  (* direct *) ir.
  assert (source f1 = substrate r). uf f1. uf first_projection. aw.
  assert (source f2 = substrate r). uf f2. uf secnd_projection. aw.
  assert (source f3 = substrate r). uf f3. uf first_projection. aw.
```

```
  assert (source f4 = substrate r). uf f4. uf secnd_projection. aw.
  assert (target f1 = substrate r'). uf f1. uf first_projection. aw.
  assert (target f2 = substrate r''). uf f2. uf secnd_projection. aw.
  assert (target f3 = substrate r'). uf f3. uf first_projection. aw.
  assert (target f4 = substrate r''). uf f4. uf secnd_projection. aw.
  assert (inc f1 (set_of_increasing_mappings r r')).  uf f1. aw. eee.
  assert (inc f3 (set_of_increasing_mappings r r')).  uf f3. aw. eee.
  assert (inc f2 (set_of_increasing_mappings r r'')).  uf f2. aw. eee.
  assert (inc f4 (set_of_increasing_mappings r r'')).  uf f4. aw. eee.
  eee. red. eee. ir. rw H22. rw H17. red in H41. ee.  cp (H48 _ H42).
  rwi product2_order_pr H49. red in H49.  ee; am. am. am. red. eee.
  ir. rw H23. rw H18. red in H41. ee.  cp (H48 _ H42).
  rwi product2_order_pr H49. red in H49.  ee; am. am. am.
  (* converse *) ir. eee.  red. eee. bw. bw.
  ir. red in H34. ee. cp (H45 _ H39). rwi H22 H46. rwi H17 H46.
  red in H32. ee. cp (H52 _ H39). rwi H23 H53. rwi H18 H53.
  rw product2_order_pr. red. ee. wr H9. app inc_W_target. ue.
  wr H6. app inc_W_target. ue. am. am. am. am. am. am. aw. eee.  aw; eee. am.
  am. am. am.  am. am. am. am.
Qed.
```

(b) Show that if E, F, G *are three ordered sets, then the ordered set* $\mathscr{A}(E \times F, G)$ *is isomorphic to the ordered set* $\mathscr{A}(E, \mathscr{A}(F, G))$.

Let's show that $S = \mathscr{A}(E \times F, G)$ is isomorphic to the ordered set $T = \mathscr{A}(E, \mathscr{A}(F, G))$. We shall assume E and F non-empty. For otherwise, the product $E \times F$ is empty, case where there is a single element in S. If F is empty, $\mathscr{A}(F, G)$ has a single element (the empty function) and $\mathscr{A}(E, \mathscr{A}(F, G))$ has a single element (that maps everything to the empty function). Assume E empty. Then $\mathscr{A}(E, \mathscr{A}(F, G))$ contains only the empty function. Whatever the orderings on the sets S and T, since the sets are singletons, the orders are trivial and any function between S and T is an isomorphism.

In all our lemmas we consider orderings r, r' and r'', with substrate E, F and G. An increasing function $f : r \times r' \to r''$ if a function with source $E \times F$ and target G; the first lemma says, that if r and r' are orders, the substrate of $r \times r'$ is indeed the product of the substrates, namely $E \times F$, and if these two sets are non-empty, one can recover the sets from the product. The second lemma says that f_x is increasing for all $x \in E$. The third lemma says that the range of $x \mapsto f_x$ is a subset of the set of increasing functions.

```
Lemma Exercice1_6h: forall f r r' r'',  order r -> order r' ->
  nonempty (substrate r) -> nonempty (substrate r') ->
  increasing_fun f (product2_order r r') r'' ->
 ( (domain (source f)) = substrate r & (range (source f)) = substrate r').
Proof. ir. red in H3. ee. wr H6. bw. rw domain_product. tv. am.
  wr H6. bw. rw range_product. tv. am.
Qed.

Lemma Exercice1_6i: forall f x r r' r'',  order r -> order r' ->
  nonempty (substrate r) -> nonempty (substrate r') ->
  increasing_fun f (product2_order r r') r'' ->
  inc x (substrate r) -> increasing_fun (second_partial_fun f x) r' r''.
Proof. ir. nin (Exercice1_6h H H0 H1 H2 H3).
  red in H3. ee. assert (partial_fun_axioms f). red.  ee. am.
  wr H9. bw. exists (substrate r). exists (substrate r'). tv.
  wri H5 H4. cp (function_spf H12 H4). red. eee.  uf second_partial_fun. aw.
  sy; am. uf second_partial_fun. aw.
```

```
  ir.
  assert (inc (J x x0) (source f)). wr H9. bw. aw. eee.  order_tac.
  assert (inc (J x y) (source f)). wr H9. bw. aw. eee. order_tac.
  rww W_spf. rww W_spf.
  app H11. rw product2_order_pr. red. bwi H9. rw H9. eee. aw.
  wrr order_reflexivity. wrr H5. aw. am. am. rw H6. order_tac. rw H6. order_tac.
Qed.

Lemma Exercice1_6j: forall f r r' r'',  order r -> order r' ->
  nonempty (substrate r) -> nonempty (substrate r') ->
  increasing_fun f (product2_order r r') r'' ->
  (restriction2_axioms (second_partial_function f) (substrate r)
    (set_of_increasing_mappings r' r'')).
Proof. ir.  nin (Exercice1_6h H H0 H1 H2 H3). red in H3. ee.
  assert (partial_fun_axioms f). red. split. am. wr H8. bw.
  exists (substrate r). exists (substrate r'). tv.
  cp (function_spfa H11). red. ee. am.
  uf second_partial_fun. uf second_partial_fun. aw.
  uf second_partial_fun. uf second_partial_function. aw. ue.
  uf second_partial_fun. uf second_partial_function. aw.
  rw H5. uf set_of_increasing_mappings. wr H9. app Z_sub.
  red. ir. ufi image_by_fun H13. awi H13. nin H13. ee.
  cp (W_pr H12 H14). rwi W_spfa H15. wr H15. rw set_of_increasing_mappings_pr.
  eee. wr H15. app function_spf. rww H4.
  uf second_partial_fun. uf second_partial_fun. aw.
  uf second_partial_fun; uf second_partial_fun; aw.
  wr H15. app (Exercice1_6i (r:=r) (f:=f) (x:=x0)). red; intuition. tv. am. am.
  rww H4.
Qed.
```

We consider now Φ =*second_partial_map* (this is the bijection from $\mathscr{F}(\mathrm{E}\times\mathrm{F};\mathrm{G})$ onto $\mathscr{F}(\mathrm{E},\mathscr{F}(\mathrm{F};\mathrm{G}))$. For each $a\in\mathrm{E}$, $\Phi(f)(a)$ is in $\mathscr{F}(\mathrm{F};\mathrm{G})$); if f is increasing this is in $\mathscr{A}(\mathrm{F},\mathrm{G})$. Thus we can restrict $\Phi(f)$ as a function $\overline{\Phi(f)}$ from E into $\mathscr{A}(\mathrm{F},\mathrm{G})$. The mapping $f\mapsto\overline{\Phi(f)}$ will be our isomorphism. It is clearly injective; proving surjectivity is a bit longer.

If f_a and $a\mapsto f_a$ are increasing, then $a\le a'$ and $b\le b'$ implies $f_a(b)\le f_a(b')\le f_{a'}(b')$ so that f is increasing. Conversely, if f is increasing, then f_a is increasing since if $b\le b'$ then $f_a(b)\le f_a(b')$, using $a\le a$; and $a\mapsto f_a$ is increasing since if $a\le a'$ then $f_a(b)\le f_{a'}(b)$, using $b\le b$. These are the only properties of the order that are used here.

```
Definition second_partial_map2 r r' r'':=
  BL (fun f=> restriction2
    (second_partial_function f)
    (substrate r) (set_of_increasing_mappings r' r''))
  (set_of_increasing_mappings (product2_order r r') r'')
  (set_of_increasing_mappings r (increasing_mappings_order r' r'')).

Lemma Exercice1_6k: forall r r' r'',  order r -> order r' -> order r'' ->
  nonempty (substrate r) -> nonempty (substrate r') ->
  transf_axioms (fun f=> restriction2
    (second_partial_function f)
    (substrate r) (set_of_increasing_mappings r' r''))
  (set_of_increasing_mappings (product2_order r r') r'')
  (set_of_increasing_mappings r (increasing_mappings_order r' r'')).
Proof. ir. red. ir. awi H4. ee.
  cp (Exercice1_6j H H0 H2 H3 H7).
```

```
  nin (Exercice1_6h H H0 H2 H3 H7).
  assert (Ha: partial_fun_axioms c). red. split. am.
  rw H5.  exists (substrate r). exists (substrate r'). tv. bw.
  set (g:= restriction2 (second_partial_function c) (substrate r)
          (set_of_increasing_mappings r' r'')).
  assert (is_function g). uf g. app function_restriction2.
  aw. eee. uf g.  uf restriction2.  aw. rww imo_substrate. uf g.
  uf restriction2. aw. red. eee. app imo_order.  uf g.  uf restriction2. aw.
  rww imo_substrate. uf g. uf restriction2. aw.
  ir.
  assert (inc x (substrate r)). order_tac.
  assert (inc y (substrate r)). order_tac.
  uf g. rww W_restriction2. rww W_restriction2.
  rww W_spfa. rww W_spfa. rww imo_pr.
  uf function_order_r.
  cp (Exercice1_6i H H0 H2 H3 H7 H14). cp (Exercice1_6i H H0 H2 H3 H7 H13).
  cp H15. red in H15. cp H16. red in H16. aw. eee.
  ir. rww W_spf. rww W_spf. assert (inc (J x i) (source c)).
  rw H5. bw. aw. eee. assert (inc (J y i) (source c)).
  rw H5. bw. aw. eee. red in H7. ee. app H36.
  rw product2_order_pr. red. ee; aw. eee. eee. order_tac. ue. ue. ue. ue.
  ue. ue. app imo_order. app order_product2_order. am.
Qed.

Lemma Exercice1_6l: forall r r' r'',  order r -> order r' -> order r'' ->
  nonempty (substrate r) -> nonempty (substrate r') ->
  let f:= second_partial_map2 r r' r'' in
    (is_function f &
      source f = (set_of_increasing_mappings (product2_order r r') r'') &
      target f = (set_of_increasing_mappings r
        (increasing_mappings_order r' r'')) &
    bijective f).
Proof. ir. uf f.  uf second_partial_map2. aw. eee.
  app bl_function. app Exercice1_6k. split.
  (* injectivity *) app injective_bl_function.  app Exercice1_6k. ir.
  awi H4. awi H5. ee. app funct_extensionality. ue. ue.
  assert (Ha: partial_fun_axioms u). red. split. am.
  rw H10.  exists (substrate r). exists (substrate r').  bw.
  assert (Hb: partial_fun_axioms v). red. split. am.
  rw H7.  exists (substrate r). exists (substrate r'). bw.
  cp (Exercice1_6h H H0 H2 H3 H9). cp (Exercice1_6h H H0 H2 H3 H12). ee.
  assert (second_partial_function u = second_partial_function v).
  app funct_extensionality. app function_spfa. app function_spfa.
  uf second_partial_function. aw. ue. uf second_partial_function. aw. rw H16.
  rww H15. rw H11; rww H8. ir. ufi second_partial_function H17.
  ufi BL H17. rwi corresp_source H17. rwi H14 H17.
  transitivity (W x (restriction2 (second_partial_function u) (substrate r)
    (set_of_increasing_mappings r' r''))).
  rww W_restriction2. app Exercice1_6j. rw H6. rww W_restriction2.
  app Exercice1_6j. clear H6. ir. bwi H10.  rwi H10 H6. awi H6. ee.
  assert (inc (P x) (domain (source u))). ue. cp (W_spfa Ha H20).
  assert (inc (P x) (domain (source v))). rww H13. cp (W_spfa Hb H22).
  wri H17 H23. rwi H21 H23.
  assert (x = J (P x) (Q x)).  sy. aw.  rw H24.
  assert (inc (Q x) (range (source u))). ue. wr (W_spf Ha H20 H25).
  assert (inc (Q x) (range (source v))). ue.  rwi H14 H20; wri H13 H20.
```

```
  wr (W_spf Hb H20 H26). rww H23.
  am. am. app order_product2_order. am. app order_product2_order. am.
  (* surjectivity *)  app surjective_bl_function. app Exercice1_6k. ir.
  awi H4. nin H4. ee.
  assert (substrate (product2_order r r') =
    product (substrate r) (substrate r')). bw.
  assert (transf_axioms (fun z => W (Q z) (W (P z) y))
    (product (substrate r)(substrate r')) (substrate r'')).
  red. ir. awi H9. ee.  wri H5 H10. cp (inc_W_target H4 H10). rwi H6 H12.
  rwii imo_substrate H12. awii H12. ee. wr H14. app inc_W_target. ue.
  set (g:= BL (fun z => W (Q z) (W (P z) y))
    (product (substrate r)(substrate r')) (substrate r'')).
  assert (is_function g). uf g. app bl_function.
  assert (increasing_fun g (product2_order r r') r'').
  red. eee. app order_product2_order. uf g. aw. uf g. aw. ir.
  assert (inc x (source g)). uf g. uf BL. rw corresp_source. wr H8. order_tac.
  assert (inc y0 (source g)). uf g. uf BL. rw corresp_source. wr H8. order_tac.
  uf g.  ufi g H12. ufi BL H12. rwi corresp_source H12. ufi g H13. ufi BL H13.
  rwi corresp_source H13. rww W_bl_function. rww W_bl_function. awi H12.
  awi H13. ee. wri H5 H16. wri H5 H14.
  cp (inc_W_target H4 H16).  cp (inc_W_target H4 H14).
  rwi H6 H18. rwi H6 H19.
  rwii imo_substrate H18. rwii imo_substrate H19. awii H18; awii H19; ee.
  rwi product2_order_pr H11. red in H11. ee. red in H22. ee.
  cp (H33 _ _ H28). red in H7. ee. cp (H39 _ _ H27). rwii imo_pr H40.
  ee. red in H42. ee.
  assert (gle r'' (W (Q x) (W (P x) y)) (W (Q x) (W (P y0) y))).  app H48.
  order_tac.
  assert (source g = (product (substrate r) (substrate r'))). uf g. aw.
  assert (range (source g) = substrate r').  rw H12. rww range_product.
  assert (partial_fun_axioms g). red. split. am.  rw H12.
  exists (substrate r). exists (substrate r'). tv.
  exists g. split.  aw. ee. am.  bw. uf g. aw. am. app order_product2_order. sy.
  app funct_extensionality.   app function_restriction2.  app Exercice1_6j.
  uf restriction2. aw. uf restriction2. aw. rwi imo_substrate H6. am. am. am.
  ir. rw W_restriction2.
  assert (Ha: inc x (domain (source g))). uf g. aw. nin H3. exists y0. aw.
  eee. fprops. rw W_spfa. cp (inc_W_target H4 H15). rwi H6 H16.
  rwii imo_substrate H16. awi H16. ee.
  app funct_extensionality. app function_spf. uf second_partial_fun.  aw.
  rw H17. sy; am. uf second_partial_fun. rw H18. aw. uf g. aw.
  ir. rw W_spf. uf g. rw W_bl_function. aw. am. aw. eee. am. am. rww H13. ue.
  am. am. am. am.  app Exercice1_6j. ue. am.  app imo_order.
Qed.

Lemma Exercice1_6m: forall r r' r'',  order r -> order r' -> order r'' ->
  nonempty (substrate r) -> nonempty (substrate r') ->
  order_isomorphism (second_partial_map2 r r' r'')
  (increasing_mappings_order (product2_order r r') r'')
  (increasing_mappings_order r (increasing_mappings_order r' r'')).
Proof. ir. cp (Exercice1_6l H H0 H1 H2 H3). cbv zeta in H4. ee.
  assert (Hc:=Exercice1_6k H H0 H1 H2 H3).
  assert (Hd:order (increasing_mappings_order r' r'')). app imo_order.
  assert (Ha:order (product2_order r r')). app order_product2_order.
  red. eee.  app imo_order. app imo_order. rww imo_substrate.
  rww imo_substrate. rw H5. ir.
```

```
  rww imo_pr. rww imo_pr. wr H6. cp H8; cp H9. awii H8; awii H9. ee.
  assert (Hu:partial_fun_axioms x). red. split. am. rw H15. bw.
  exists (substrate r). exists  (substrate r'). tv.
  assert (Hv:partial_fun_axioms y). red. split. am. rw H12. bw.
  exists (substrate r). exists  (substrate r'). tv.
  set (u1:= inc (W x (second_partial_map2 r r' r''))
      (target (second_partial_map2 r r' r''))). assert u1. uf u1.
  app inc_W_target. ue.
  set (u2:= inc (W y (second_partial_map2 r r' r''))
      (target (second_partial_map2 r r' r''))). assert u2. uf u2.
  app inc_W_target. ue.
  uf second_partial_map2. aw. cp H18; cp H19.  ufi u1  H18; ufi u2 H19.
  unfold second_partial_map2 in H18, H19 |-*.
  rwii  W_bl_function H18.  rwii  W_bl_function H19. awi H18. awi H19.
  cp (Exercice1_6j H H0 H2 H3 H17). cp (Exercice1_6j H H0 H2 H3 H14).
  app iff_eq. ir. ee. am. am.
  red.  eee. ir. rww W_restriction2.  rww W_restriction2.
  assert (inc i (domain (source x))). rw H30. bw.  rww domain_product.
  assert (inc i (domain (source y))). rw H27. bw.  rww domain_product.
  assert (range (source x) = substrate r'). rw H30. bw.  rww range_product.
  assert (range (source y) = substrate r'). rw H27. bw.  rww range_product.
  rww W_spfa. rww W_spfa.
  rw imo_pr. aw. aw. cp (Exercice1_6i H H0  H2 H3 H29 H39).
  cp (Exercice1_6i H H0  H2 H3 H32 H39). red in H44; red in H45. eee.
  red. eee. red. eee. red. eee.
  ir. rww W_spf. rww W_spf.
  nin H26. ee. app H62. bw. aw.  eee. ue. ue. am. am.
  (* converse *) ir.  eee.  red. ir. eee. ir. bwi H33. awi H33. ee.
  assert(inc (P i) (domain (source x))). rw H15. bw. rww domain_product.
  assert(inc (P i) (domain (source y))). rw H12. wrr H15.
  assert(inc (Q i) (range (source x))). rw H15. bw. rww range_product.
  assert(inc (Q i) (range (source y))). rw H12. wrr H15.
  assert (i = J (P i) (Q i)). sy; app pair_recov. rw H40.
  red in H26. ee. cp (H46 _ H34). rwii W_restriction2 H47.
  rwii W_restriction2 H47. rwi W_spfa H47. rwi W_spfa H47. rwi imo_pr H47. ee.
  red in H49. ee. cp (H55 _ H35). rwii W_spf H56. rwii W_spf H56. am. am. am.
  am. am. am.  am. am. am. am. am. am.
Qed.
```

(c) If $\mathrm{E} \neq \emptyset$*, then* $\mathscr{A}(\mathrm{E},\mathrm{F})$ *is a lattice if and only if* F *is a lattice.*

We pretend that constant functions are increasing, and if $\mathrm{E} \neq \emptyset$, the constants $\cdot \mapsto x$ and $\cdot \mapsto x'$ compare as x and x'.

```
Lemma constant_increasing: forall r r', order r -> order r' ->
  forall y (Hy: inc y (substrate r')),
    (inc (constant_function (substrate r) (substrate r') y Hy)
      (set_of_increasing_mappings r r')).
Proof. ir. aw. eee.  app function_constant_fun. uf constant_function. aw.
  uf constant_function. aw. red. eee.
  app function_constant_fun. uf constant_function. aw.
  uf constant_function. aw. ir.
  rw W_constant. rw W_constant. order_tac. order_tac. order_tac.
Qed.


Lemma constant_increasing: forall r r', order r -> order r' ->
```

```
  forall y (Hy: inc y (substrate r')),
    (inc (constant_function (substrate r) (substrate r') y Hy)
      (set_of_increasing_mappings r r')).
Proof. ir. rw set_of_increasing_mappings_pr.
  exists (constant_function (substrate r) (substrate r') y Hy). eee.
  app function_constant_fun. red. eee.
  app function_constant_fun. ir.
  rw W_constant. rw W_constant. order_tac. order_tac. order_tac. am. am.
Qed.

Lemma constant_increasing1: forall r r', order r -> order r' ->
  nonempty (substrate r) ->
  forall y (Hy: inc y (substrate r'))  y' (Hy': inc y' (substrate r')),
    gle r' y y' =
    gle (increasing_mappings_order r r')
    (constant_function (substrate r) (substrate r') y Hy)
    (constant_function (substrate r) (substrate r') y' Hy').
Proof. ir. rw imo_pr. app iff_eq. ir. eee. app constant_increasing.
  app constant_increasing. red. ee. app function_constant_fun.
  app function_constant_fun. uf constant_function. aw. uf constant_function. aw.
  uf constant_function. aw. uf constant_function. aw.
  ir. rw W_constant. rw W_constant.
  am.  am. am. ir. ee. awi H4. nin H1. red in H4. ee. cp (H10 _ H1).
  rwi W_constant H11. rwi W_constant H11. am. am. am. am. am.
Qed.
```

Assume that F is a lattice; given two functions f and g we can consider $x \mapsto \sup(f(x), g(x))$ and $x \mapsto \inf(f(x), g(x))$. Since F is a lattice, if f and g are functions with target F, these are functions with target F.

```
Lemma Exercice1_6n: forall r r', order r -> order r'-> nonempty (substrate r) ->
  lattice r' = lattice (increasing_mappings_order r r').
Proof. ir. app iff_eq. clear H1. ir. uf lattice. split. app imo_order.
  rw imo_substrate. ir. assert (Hx:=H2).  assert (Hy:=H3).
  awi H2. awi H3.  set (E:=substrate r) in *;  set (E':=substrate r') in *.
  assert (transf_axioms (fun i=> sup r' (W i x) (W i y)) E E'). red. ir.
  ee. assert (inc (W c x) E'). wr H9.  app inc_W_target. ue.
  assert (inc (W c y) E'). wr H6. app inc_W_target. ue.
  nin (lattice_sup_pr H1 H11 H12). uf E'. order_tac.
  assert (transf_axioms (fun i=> inf r' (W i x) (W i y)) E E'). red. ir.
  ee. assert (inc (W c x) E'). wr H10.  app inc_W_target. ue.
  assert (inc (W c y) E'). wr H7. app inc_W_target. ue.
  nin (lattice_inf_pr H1 H12 H13). uf E'. order_tac.
  set (f1:= BL (fun i=> sup r' (W i x) (W i y)) E E').
  set (f2:= BL (fun i=> inf r' (W i x) (W i y)) E E').
  assert (is_function f1). uf f1. app bl_function.
  assert (is_function f2). uf f2. app bl_function.
```

The mappings $x \mapsto \sup(f(x), g(x))$ and $x \mapsto \inf(f(x), g(x))$ are increasing, hence in $\mathscr{A}(\mathrm{E}, \mathrm{F})$.

```
 assert (Ha:increasing_fun f1 r r'). red.  eee. uf f1.  aw. uf f1. aw. uf f1.
  ir. assert (inc x0 E). uf E. order_tac. assert (inc y0 E). uf E. order_tac.
  aw. set (a:= W x0 x). set (b:= W x0 y).
  assert (inc a E'). wr H12. uf a. app inc_W_target. ue.
  assert (inc b E'). wr H9. uf b. app inc_W_target. ue.
```

```
set (a':= W y0 x). set (b':= W y0 y).
assert (inc a' E'). wr H12. uf a'. app inc_W_target. ue.
assert (inc b' E'). wr H9. uf b'. app inc_W_target. ue.
cp (lattice_sup_pr H1 H17 H18). cp (lattice_sup_pr H1 H19 H20). ee.
ap H26. assert (gle r' a a').  uf a; uf a'. red in H13. ee. app H31.
order_tac.
assert (gle r' b b'). uf b; uf b'. red in H10. ee. app H31. order_tac.
assert (Hb:increasing_fun f2 r r'). red.  eee. uf f2.  aw. uf f2. aw. uf f2.
ir.  assert (inc x0 E). uf E. order_tac. assert (inc y0 E). uf E. order_tac.
aw. set (a:= W x0 x). set (b:= W x0 y).
assert (inc a E'). wr H12. uf a. app inc_W_target. ue.
assert (inc b E'). wr H9. uf b. app inc_W_target. ue.
set (a':= W y0 x). set (b':= W y0 y).
assert (inc a' E'). wr H12. uf a'. app inc_W_target. ue.
assert (inc b' E'). wr H9. uf b'. app inc_W_target. ue.
cp (lattice_inf_pr H1 H17 H18). cp (lattice_inf_pr H1 H19 H20). ee.
ap H24. assert (gle r' a a').  uf a; uf a'. red in H13. ee. app H31.
order_tac.
assert (gle r' b b'). uf b; uf b'. red in H10. ee. app H31. order_tac.
assert (inc f1 (set_of_increasing_mappings r r')). aw. eee; uf f1; aw.
assert (inc f2 (set_of_increasing_mappings r r')).  aw. eee; uf f2; aw.
```

The mapping $x \mapsto \sup(f(x), g(x))$ is the supremum.

```
ee. exists f1. rw least_upper_bound_pr. split. red. split.
rww imo_substrate. ir. nin (doubleton_or H16); rw H17; rw imo_pr. eee.
red. uf f1. eee. aw. aw. ir. aw.
assert (inc (W i x)  E'). wr H14. app inc_W_target. ue.
assert (inc (W i y) E'). wr H11. app inc_W_target. ue.
cp (lattice_sup_pr H1 H19 H20).  nin H21. am. am. am.
eee. red. uf f1.  eee. aw. aw. ir. aw.
assert (inc (W i x)  E'). wr H14. app inc_W_target. ue.
assert (inc (W i y) E'). wr H11. app inc_W_target. ue.
cp (lattice_sup_pr H1 H19 H20). ee. am. am. am.
ir. red in H16. nin H16. assert (inc x (doubleton x y)). fprops.
cp (H17 _ H18).  assert (inc y (doubleton x y)). fprops.  cp (H17 _ H20).
rw imo_pr.  rwi imo_pr H19; rwi imo_pr H21. ee.  am. am. aw. red. red in H23.
eee. uf f1. aw. uf f1; aw. ir. red in H25; ee. cp (H38 _ H32).
cp (H31 _ H32). uf f1. aw.
assert (inc (W i x) E'). wr H14. app inc_W_target. ue.
assert (inc (W i y) E'). wr H11. app inc_W_target. ue.
cp (lattice_sup_pr H1 H41 H42).  ee. app H45. am. am. am. am. am. am. am.
am. am. am. app imo_order. rw imo_substrate. red. ir.
nin (doubleton_or H16); rw H17; am. am. am.
```

The mapping $x \mapsto \inf(f(x), g(x))$ is the infimum.

```
exists f2. aw. split. red. split. rww imo_substrate. ir.
nin (doubleton_or H16); rw H17; rw imo_pr. eee.
red. uf f2. eee. aw. aw. ir. aw.
assert (inc (W i x)  E'). wr H14. app inc_W_target. ue.
assert (inc (W i y) E'). wr H11. app inc_W_target. ue.
cp (lattice_inf_pr H1 H19 H20).  nin H21. am. am. am.
eee. red. uf f2.  eee. aw. aw. ir. aw.
assert (inc (W i x)  E'). wr H14. app inc_W_target. ue.
```

```
assert (inc (W i y) E'). wr H11. app inc_W_target. ue.
cp (lattice_inf_pr H1 H19 H20). ee. am. am. am.
ir. red in H16. nin H16. assert (inc x (doubleton x y)). fprops.
cp (H17 _ H18).  assert (inc y (doubleton x y)). fprops.  cp (H17 _ H20).
rw imo_pr.  rwi imo_pr H19; rwi imo_pr H21. ee.  am. am. aw. red. red in H23.
eee. uf f2. aw. uf f2; aw. ir. red in H25; ee. cp (H38 _ H32).
cp (H31 _ H32). uf f2. aw.
assert (inc (W i x) E'). wr H14. app inc_W_target. ue.
assert (inc (W i y) E'). wr H11. app inc_W_target. ue.
cp (lattice_inf_pr H1 H41 H42).  ee. app H45. am. am. am. am. am. am. am.
am. am. am. app imo_order. rw imo_substrate. red. ir.
nin (doubleton_or H16); rw H17; am. am. am.   am. am. am. am. am. am.
```

Conversely. Given two values a and b in F, we can consider the two constant functions $x \mapsto a$ and $x \mapsto b$. They are increasing.

```
ir. red. split. am. assert (He:= H1).
set (E:=substrate r) in *;  set (E':=substrate r') in *. nin H1. ir.
set (f1:= constant_function E E' x H3).
set (f2:= constant_function E E' y0 H4).
assert (inc (corr_value f1) (set_of_increasing_mappings r r')).
uf f1. uf E. uf E'. app constant_increasing.
assert (inc (corr_value f2) (set_of_increasing_mappings r r')).
uf f2. uf E. uf E'. app constant_increasing.
assert (inc (corr_value f1) (substrate (increasing_mappings_order r r'))).
rw imo_substrate. am. am. am.
assert (inc (corr_value f2) (substrate (increasing_mappings_order r r'))).
rw imo_substrate. am. am. am.
```

Constant functions are in the same order as their value on a fixed point in E. The value c of the supremum of the two functions at the point is an upper bound for a and b. Given an upper bound c', we construct the constant function $x \mapsto c'$, and compare it, and get $c \leq c'$.

```
split. cp (lattice_sup_pr H2 H7 H8).
set (f3:= sup (increasing_mappings_order r r') f1 f2) in *.
ee. assert (inc f3 (substrate (increasing_mappings_order r r'))).
app (inc_arg2_substrate H9). rwi imo_substrate H12. awi H12. ee.
assert (inc (W y f3) E'). uf E'.  wr H14. app inc_W_target. rww H13.
exists (W y f3). rw least_upper_bound_pr. split.
red. ee.  am. ir. nin  (doubleton_or H17); rw H18.
rwi imo_pr H9. nin H9. nin H19. awi H20. red in H20. ee. cp (H26 _ H1).
ufi f1 H27. rwi W_constant H27. am. am. am. am.
rwi imo_pr H10. nin H10. nin H19. awi H20. red in H20. ee. cp (H26 _ H1).
ufi f2 H27. rwi W_constant H27. am. am. am. am.
ir. nin H17. set (f4:= constant_function E E' z H17).
assert (inc f4 (set_of_increasing_mappings r r')).
uf f4. uf E. uf E'. app constant_increasing.
assert (gle (increasing_mappings_order r r') f1 f4).
uf f1. uf f4. uf E. uf E'.  wr constant_increasing1. app H18. fprops. am.
am. am.
assert (gle (increasing_mappings_order r r') f2 f4).
uf f2. uf f4. uf E. uf E'.  wr constant_increasing1. app H18. fprops. am.
am. am. cp (H11 _ H20 H21). rwi imo_pr H22.  ee. red in H24. ee.
cp (H30 _ H1). ufi f4 H31.  rwi W_constant H31. am.
am. am. am. am. red. ir. nin (doubleton_or H17); rw H18; am. am. am. am. am.
```

Idem for the infimum.

```
  cp (lattice_inf_pr H2 H7 H8).
  set (f3:= inf (increasing_mappings_order r r') f1 f2) in *.
  ee. assert (inc f3 (substrate (increasing_mappings_order r r'))).
  app (inc_arg1_substrate H9). rwi imo_substrate H12. awi H12. ee.
  assert (inc (W y f3) E'). uf E'.  wr H14. app inc_W_target. rww H13.
  exists (W y f3). aw. split.
  red. ee.  am. ir. nin  (doubleton_or H17); rw H18.
  rwi imo_pr H9. nin H9. nin H19. awi H20. red in H20. ee. cp (H26 _ H1).
  ufi f1 H27. rwi W_constant H27. am. am. am. am.
  rwi imo_pr H10. nin H10. nin H19. awi H20. red in H20. ee. cp (H26 _ H1).
  ufi f2 H27. rwi W_constant H27. am. am. am. am.
  ir. nin H17. set (f4:= constant_function E E' z H17).
  assert (inc f4 (set_of_increasing_mappings r r')).
  uf f4. uf E. uf E'. app constant_increasing.
  assert (gle (increasing_mappings_order r r') f4 f1).
  uf f1. uf f4. uf E. uf E'.  wr constant_increasing1. app H18. fprops. am.
  am. am.
  assert (gle (increasing_mappings_order r r') f4 f2).
  uf f2. uf f4. uf E. uf E'.  wr constant_increasing1. app H18. fprops. am.
  am. am. cp (H11 _ H20 H21). rwi imo_pr H22.  ee. red in H24. ee.
  cp (H30 _ H1). ufi f4 H31.  rwi W_constant H31. am.
  am. am. am. red. ir. nin (doubleton_or H17); rw H18; am. am. am. am. am.
Qed.
```

(d) Suppose that E *and* F *are both non-empty. Then* $\mathscr{A}$(E, F) *is totally ordered if and only if one of the following conditions is satisfied:*
(α*)* F *consists in a single element;*
(β*)* E *consists in a single element and* F *is totally ordered;*
(γ*)* E *and* F *are both totally ordered and* F *has two elements.*

We first show that if $\mathscr{A}$(E, F) is totally ordered and E non-empty, then F is totally ordered, using constant functions as above.

```
Lemma Exercice1_6o: forall r r', order r -> order r'-> nonempty (substrate r)->
  total_order (increasing_mappings_order r r') ->
  total_order r'.
Proof. ir. nin H2. red. ee. am. ir.
  set (E:=substrate r) in *;  set (E':=substrate r') in *. nin H1.
  set (f1:= constant_function E E' x H4).
  set (f2:= constant_function E E' y H5).
  assert (substrate r = source f1). uf f1. uf constant_function. aw.
  assert (substrate r' = target f1). uf f1. uf constant_function. aw.
  assert (is_function f1). uf f1. app function_constant_fun.
  assert (inc f1 (set_of_increasing_mappings r r')). aw. eee. red. eee.
  uf f1. ir. rw W_constant. rw W_constant. wrr order_reflexivity.
  uf E; order_tac. uf E; order_tac.
  assert (substrate r = source f2). uf f2. uf constant_function. aw.
  assert (substrate r' = target f2). uf f2. uf constant_function. aw.
  assert (is_function f2). uf f2. app function_constant_fun.
  assert (inc f2 (set_of_increasing_mappings r r')). aw. eee. red. eee.
  uf f2. ir. rw W_constant. rw W_constant. wrr order_reflexivity.
  uf E; order_tac. uf E; order_tac.
  rwi imo_substrate H3. ufi gge H3. nin (H3 _ _ H9 H13); rwi imo_pr H14; ee.
```

```
  awi H16. red in H16. ee. cp (H22 _ H1). ufi f1 H23; ufi f2 H23.
  rwi W_constant H23. rwi W_constant H23. au. am. am. am. am.
  awi H16. red in H16. ee. cp (H22 _ H1). ufi f1 H23; ufi f2 H23.
  rwi W_constant H23. rwi W_constant H23. au. am. am. am. am. am. am.
Qed.
```

If F is a singleton, then $\mathscr{A}(E,F)$ has a single element, hence is totally ordered. If E is a singleton, all functions are constant and $\mathscr{A}(E,F)$ is isomorphic to F.

```
Lemma Exercice1_6p: forall r r', order r -> order r'->
  is_singleton (substrate r') ->
  total_order (increasing_mappings_order r r').
Proof. ir. red. ee. app imo_order. rww imo_substrate. ir. left.
  rw imo_pr. eee. rwi set_of_increasing_mappings_pr H2. awi H3. ee.
  red. aw. eee. ir. assert (inc (W i x)  (substrate r')). wr H8.
  app inc_W_target.  ue.  assert (inc (W i y)  (substrate r')). wr H5.
  app inc_W_target.  ue. nin H1. rwi H1 H11; rwi H1 H12.
  rw (singleton_eq H11).   rw (singleton_eq H12). wrr order_reflexivity.
  rw H1. fprops. am. am. am. am. am. am.
Qed.

Lemma Exercice1_6q: forall r r', order r -> order r'->
  is_singleton (substrate r) -> total_order r' ->
  total_order (increasing_mappings_order r r').
Proof. ir. red. ee. app imo_order. rww imo_substrate. nin H1.
  ir. uf gge. rww imo_pr. rww imo_pr. cp H3; cp H4; awii H3; awii H4. ee.
  assert (inc (W x y) (substrate r')). wr H8.  app inc_W_target. ue.
  rw H1. fprops.
  assert (inc (W x x0) (substrate r')). wr H11.  app inc_W_target. ue.
  rw H1. fprops.
  assert (forall u, inc u (substrate r) -> W u y =  (W x y)). ir.
  rwi H1 H15. rww (singleton_eq H15).
  assert (forall u, inc u (substrate r) -> W u x0 = (W x x0)). ir.
  rwi H1 H16. rww (singleton_eq H16).
  nin H2.  nin (H17 _ _ H13 H14). right. eee. red. eee. ir. rww H15. rww H16.
  left. eee. red. eee. ir. rww H15. rww H16.
Qed.
```

Assume that E and F are two totally ordered sets, and F has two elements. We may assume that these elements are distinct and satisfy $a < b$. Assume $f(u) < g(u)$. Then $f(u) = a$ and $g(u) = b$. If no such u exists, then $f \geq g$. Otherwise, consider v; if $f(v) > g(v)$ we get $f(v) = b$ and $g(v) = a$. Since u and v can be compared, this implies that one of f and g is non-increasing.

Converse. Assume $\mathscr{A}(E,F)$ totally ordered, both sets E and F non-empty. We may assume that F is totally ordered and that both sets have at least two elements. We may assume that $b < b'$ in F.

```
Lemma Exercice1_6s: forall r r', order r -> order r'->
  nonempty(substrate r) -> nonempty (substrate r') ->
  total_order (increasing_mappings_order r r') ->
  (is_singleton (substrate r') \/
   (is_singleton (substrate r) & total_order r') \/
```

```
   (total_order r' & total_order r
     & exists u , exists v, substrate r' = doubleton u v)).
Proof. ir. cp (Exercice1_6o H H0 H1 H3). nin H2.
  nin (equal_or_not (substrate r')(singleton y)). ir. left.
  exists y. tv. ir. right. nin H1.
  nin (equal_or_not (substrate r)(singleton y0)). ir. left. split.
  exists y0. tv. am. ir. right. split. am.
  assert (exists y1, inc y1 (substrate r') & y1 <> y). app exists_proof. red.
  ir. elim H5.  set_extens. nin (equal_or_not x y). rw H9. fprops.
  elim (H7 x). intuition. rw (singleton_eq H8). am. nin H7.
  assert (exists u,  exists v, inc u (substrate r') & inc v (substrate r')
    & glt r' u v). ee.  nin H4. nin (H9 _ _ H2 H7). exists y; exists x. eee.
  red. au. exists x; exists y. eee. red. eee.
  clear H2.  clear H5. clear H7. nin H8. nin H2.
  assert (exists y1, inc y1 (substrate r) & y1 <> y0). app exists_proof. red.
  ir. elim H6.  set_extens. nin (equal_or_not x2 y0). rw H8. fprops.
  elim (H5 x2). intuition. rw (singleton_eq H7). am. nin H5.
  rename y0 into a; rename x2 into a'; rename x0 into b; rename x1 into b'.
```

Consider the mapping f_u that associates b if $x \leq u$ and b' otherwise. Assume $x \leq x'$; the relation $f(x) \leq f(x')$ is true if one of the values is $f(x')$ is b'. Otherwise we have $x' \leq u$, thus $x \leq u$ and $f(x) = b$. Thus f_u is increasing. Consider similarly f_v. Since $\mathscr{A}(\mathrm{E},\mathrm{F})$ is totally ordered, we may compare the functions. If $f_u \leq f_v$ we have $f_u(v) \leq f_v(v) = b$. We cannot have $f_u(v) = b'$, hence $v \leq u$. Similarly if $f_v \leq f_u$ we have $u \leq v$.

```
  assert (total_order r). red. ee. am. ir.
  set (f1:= fun u=> Yo (gle r u x0) b b').
  assert (transf_axioms f1 (substrate r) (substrate r')).
  red. ir. uf f1. nin (p_or_not_p  (gle r c x0)). ir.
  rww Y_if_rw. ir. rww Y_if_not_rw.
  assert (inc (BL f1 (substrate r) (substrate r'))
    (set_of_increasing_mappings r r')).
  aw. eee. app bl_function. red. eee. app bl_function. aw. aw. ir.
  assert (inc x1 (substrate r)). order_tac.
  assert (inc y1 (substrate r)). order_tac.
  rww W_bl_function. rww W_bl_function. uf f1.
  nin (p_or_not_p (gle r x1 x0)). ir. rww Y_if_rw.
  nin (p_or_not_p (gle r y1 x0)). ir. rww Y_if_rw.
  order_tac. ir. rww Y_if_not_rw. nin H9. am.
  rww Y_if_not_rw. nin (p_or_not_p (gle r y1 x0)).
  rww Y_if_rw. elim H16. order_tac.
  rww Y_if_not_rw. order_tac.
  set (f2:= fun u=> Yo (gle r u y0) b b').
  assert (transf_axioms f2 (substrate r) (substrate r')).
  red. ir. uf f2. nin (p_or_not_p (gle r c y0)).  rww Y_if_rw. rww Y_if_not_rw.
  assert (inc (BL f2 (substrate r) (substrate r'))
    (set_of_increasing_mappings r r')). aw. eee. app bl_function.  red. eee.
  app bl_function. aw. aw. ir.
  assert (inc x1 (substrate r)). order_tac.
  assert (inc y1 (substrate r)). order_tac.
  rww W_bl_function. rww W_bl_function. uf f2.
  nin (p_or_not_p (gle r x1 y0)). rww Y_if_rw.
  nin (p_or_not_p (gle r y1 y0)). rww Y_if_rw.
  order_tac. rww Y_if_not_rw. nin H9. am.
  rww Y_if_not_rw. nin (p_or_not_p (gle r y1 y0)).
  rww Y_if_rw. elim H18. order_tac.
```

```
rww Y_if_not_rw.  order_tac.
nin H3. rwi imo_substrate H16. nin (H16 _ _ H13 H15).
rwi imo_pr H17. ee. red in H19. ee. awi H25. ufi f1 H25; ufi f2 H25.
cp (H25 _ H11). awi H26. nin (p_or_not_p (gle r y0 x0)). ir.
right. red. am. ir. rwi Y_if_not_rw H26.  rwi Y_if_rw H26. nin H9. elim H28.
order_tac. order_tac. am. am. am. am. am. am. am.
red in H17. rwi imo_pr H17. ee. red in H19. ee. awi H25.
ufi f1 H25; ufi f2 H25. cp (H25 _ H10). awi H26.
nin (p_or_not_p (gle r x0 y0)). ir. left. am. ir. rwi Y_if_not_rw H26.
rwi Y_if_rw H26. nin H9. elim H28.
order_tac. order_tac. am. am. am. am. am. am. am. am. am.
```

We pretend now that F has at most two elements. We use contradiction. We are reduced to analyze the case where F has three elements satisfying $x_0 < x_1 < x_2$.

```
split. am. nin (equal_or_not (substrate r')( doubleton b b')). ir.
exists b. exists b'. rw H8. tv. ir.
assert (exists b'', inc b'' (substrate r') & b'' <> b & b'' <> b').
app exists_proof. red. ir. elim H8. set_extens. nin (equal_or_not x0 b).
rw H11. fprops. nin (equal_or_not x0 b'). rw H12. fprops. elim (H9 x0).
intuition. ee. nin (doubleton_or H10); rw H14; am.
assert (exists u, exists v, exists w,
  inc u (substrate r') & inc v (substrate r') & inc w (substrate r') &
  glt r' u v & glt r' v w).  nin H9. ee. red in H4. ee. nin (H15 _ _ H9 H2).
exists x0; exists b; exists b'. eee. red. split. am. am.
nin (H15 _ _ H9 H13). exists b; exists x0; exists b'. eee.
red. red in H16. ee. am. intuition. red. ee. am. am.
exists  b; exists b'; exists x0. eee. red. ee.  red in H17.
am. intuition. clear  H2. clear H9. clear H8. nin H10. nin H2. nin H2. ee.
```

The set E has two elements $c < c'$; we could use the same argument as above, but we use here $\inf(a,a') < \sup(a,a')$. We consider two functions. First the constant function $f : x \mapsto x_1$; then the function g that associates x_0 if $x \leq c$ and $x2$ otherwise. The same arguments as above show that they are increasing. Since $\mathscr{A}(\mathrm{E},\mathrm{F})$ is totally ordered, we may compare them. We have $f(v) < g(v)$ and $f(u) > g(u)$.

```
set (f:= constant_function (substrate r) (substrate r') x1 H8).
assert (inc f (set_of_increasing_mappings r r')). uf f. aw. eee.
app function_constant_fun. uf constant_function. aw.
uf constant_function. aw. red. eee. app function_constant_fun.
uf constant_function. aw. uf constant_function. aw.
ir. rw W_constant. rw W_constant. order_tac. order_tac. order_tac.
cp (total_order_lattice H7). cp (lattice_sup_pr H14 H1 H5).
cp (lattice_inf_pr H14 H1 H5). set (u:=inf r a a').
assert (inc u (substrate r)).  nin H16. order_tac.
set (v:=sup r a a').
assert (inc v (substrate r)). nin H15. order_tac.
assert (glt r u v). red. ee. order_tac. red. ir.
ufi u H23; ufi v H23. rwi H23 H16. rwi H23 H19.
assert (gle r a a'). order_tac.  assert (gle r a' a). order_tac.
elim H12.  order_tac.
set (g:= fun x=> Yo (gle r x u) x0 x2).
assert (transf_axioms g (substrate r) (substrate r')).
red. ir. uf g. nin (p_or_not_p (gle r c u)). ir.
rww Y_if_rw. ir. rww Y_if_not_rw.
```

```
  assert (inc  (BL g (substrate r) (substrate r'))
    (set_of_increasing_mappings r r')).  aw. eee. app bl_function.
  red. eee. app bl_function. aw. aw. ir.
  assert (inc x3 (substrate r)). order_tac.
  assert (inc y0 (substrate r)). order_tac.
  rww W_bl_function. rww W_bl_function. uf g.
  nin (p_or_not_p (gle r x3 u)). ir. rww Y_if_rw.
  nin (p_or_not_p (gle r y0 u)). ir. rww Y_if_rw.
  order_tac. ir. rww Y_if_not_rw. nin H10. nin H11.
  order_tac. ir.
  rww Y_if_not_rw. nin (p_or_not_p  (gle r y0 u)). ir.
  rww Y_if_rw.  elim H28. order_tac. ir.
  rww Y_if_not_rw. order_tac.
  nin H3. rwi imo_substrate H22. nin (H22 _ _ H13 H21).
  rwi imo_pr H23. ee. red in H25. ee. awi H35. ufi f H35. awi H35.
  cp (H35 _ H17). awi H36. rwi W_constant H36. ufi g H36. rwi Y_if_rw H36.
  nin H10. elim H37. order_tac. order_tac.
  am. am. am. am. am. red in H23. rwi imo_pr H23. ee. red in H25. ee. awi H35.
  ufi f H35. awi H35. cp (H35 _ H18). awi H36. rwi W_constant H36. ufi g H36.
  rwi Y_if_not_rw H36. nin H11. elim H37. order_tac.
  red.  ir. nin H19. elim H38. order_tac.  am. am. am. am. am. am. am.
Qed.
```

---

7. *In order that every mapping of an ordered set* E *into an ordered set* F *with at least two elements, which is both an increasing and a decreasing mapping, should be constant on* E*, it is necessary and sufficient that* E *should be connected with respect to the reflexive and symmetric relation "x and y are comparable" (Chapter II, § 6, Exercise 10). This condition is satisfied if* E *is either left or right directed.*

If F is empty or has a single element, the all functions with values in F are constant. This explains why F is assume to have at least two elements. Denote the relation by $x \sim y$. We start, as in Exercise 4, with some properties of the connected components of this relation.

```
Definition comp_rel r:= fun x y => (gle r x y \/ gle r y x).

Definition cr_equiv r :=
  Exercice1.Sgraph (comp_rel r) (substrate r).
Definition cr_component r :=
  Exercice1.connected_comp (comp_rel r) (substrate r).

Lemma cr_properties: forall r, order r ->
  (is_equivalence (cr_equiv r) &
    (forall x y, comp_rel r x y -> (inc x (substrate r)& inc y (substrate r))) &
    substrate  (cr_equiv r) =  substrate r &
    (forall x, inc x (substrate r) -> class (cr_equiv r) x = cr_component r x) &
    (forall x y, comp_rel r x y -> related (cr_equiv r) x y)).
Proof. ir. uf cr_equiv. uf cr_component.
  assert (forall x y : Set,
    comp_rel r x y -> (inc x (substrate r) & inc y (substrate r))). ir. nin H0.
  split;  order_tac. split; order_tac.
  assert (reflexive_r (comp_rel r) (substrate r)).
  red. ir. app iff_eq.  uf comp_rel. ir. left. order_tac.
```

```
  ir.  nin (H0 _ _ H1). am.
  assert (symmetric_r (comp_rel r)). red. uf comp_rel. ir. intuition.
  assert (forall x y, comp_rel r x y ->  inc x (substrate r)).
  ir.  nin (H0 _ _ H3). am.
  ee. app Exercice1.equivalence_Sgraph. am. app Exercice1.substrate_Sgraph.
  ir. app  Exercice1.connected_comp_class.
  ir. red. uf Exercice1.Sgraph. uf graph_on. ee. Ztac. nin (H0 _ _ H4).
  fprops. red. aw. exists (Exercice1.chain_pair x y). au.
Qed.
```

Given two elements of E, if they have an upper bound or lower bound, this bound is related to both elements. Thus E is directed, it has a single component.

```
Lemma Exercice1_7a: forall r x, right_directed r ->
  inc x (substrate r) -> cr_component r x = substrate r.
Proof. ir. rwi right_directed_pr H. nin H. cp (cr_properties H). ee. wr H5.
  app extensionality. wr H4. app sub_class_substrate. red. ir. bw.
  nin (H1 _ _ H0 H7). nin H8. nin H9. assert (related (cr_equiv r) x x1).
  app H6. red. left. am. assert (related (cr_equiv r) x1 x0).
  app H6. red. right. am.  apply transitivity_e with x1. am. am.
  nin H2. am. nin H2; am.
Qed.

Lemma Exercice1_7b: forall r x, left_directed r ->
  inc x (substrate r) -> cr_component r x = substrate r.
Proof. ir. rwi left_directed_pr H. nin H. cp (cr_properties H). ee. wr H5.
  app extensionality. wr H4. app sub_class_substrate. red. ir. bw.
  nin (H1 _ _ H0 H7). nin H8. nin H9. assert (related (cr_equiv r) x x1).
  app H6. red. right. am. assert (related (cr_equiv r) x1 x0).
  app H6. red. left. am. apply transitivity_e with x1. am. am.
  nin H2. am. nin H2; am.
Qed.
```

If f is increasing and decreasing, x and y are related, then $f(x) = f(y)$.

```
Lemma Exercice1_7c: forall r r' f x y, increasing_fun f r r' ->
  decreasing_fun f r r' -> comp_rel r x y ->
  W x f = W y f.
Proof.  ir. assert (order r). nin H; eee. red in H. red in H0. ee.
  nin H1; cp (H12 _ _ H1); cp (H7 _ _ H1); red in H14; order_tac.
Qed.
```

If f is increasing and decreasing, then f is constant on chains. If E is the class of x, every element $u \in$ E is chained to x, $f(u) = f(x)$ hence f is constant.

```
Lemma Exercice1_7d: forall r r' f, increasing_fun f r r' ->
  decreasing_fun f r r' ->
  (exists x, inc x (substrate r) & cr_component r x = substrate r)
  -> (is_constant_function f).
Proof. ir.  assert (forall x y, comp_rel r x y ->  W x f = W y f). ir.
  app (Exercice1_7c H H0 H2).  red in H. ee.
  cp (cr_properties H3).  nin H1. ee.
  red. ee. am. ir. assert (source f= substrate r). au.
  assert (forall u, inc u (source f) -> W u f = W x f). ir. rwi H16 H17.
  wri H13 H17. wri (H11 _ H1) H17. bwi H17.
```

```
  assert (related (cr_equiv r) u x). apply symmetricity. am. am.
  ufi cr_equiv H18. ufi Exercice1.Sgraph H18. ufi graph_on H18. red in H18.
  Ztac. nin H20. awi H20. ee.
  assert (forall x1, Exercice1.chained_r (comp_rel r) x1 ->
    Exercice1.chain_tail x1 = x  ->
    W (Exercice1.chain_head x1) f = W x f). ir. nin x2. simpl. simpl in H24.
  simpl in H23. wr H24. app H2. simpl. simpl in H23. ee. cp (IHx2 H25 H24).
  simpl in H24. rww (H2 _ _ H23). wr (H23 _ H20 H22).  ue. am.
  rw (H17 _ H14). rw (H17 _ H15). tv.
Qed.
```

Converse. We assume that F is a set with at least two elements a and b, and that E contains c such that the component C of c is not E (if c' is another element of E, the components of c and c' are equal of disjoint, so that the component of c' cannot be E.). We consider the function g that maps x to a if $x \in$ C, and to be otherwise. This is a non-constant function since since is a nonempty strict subset of C.

```
Lemma Exercice1_7e: forall r r', order r -> order r' ->
  (exists u, exists v, inc u (substrate r') & inc v (substrate r')  & u <>v)
  -> (exists x, inc x (substrate r) & cr_component r x <> substrate r)
  -> exists f, increasing_fun f r r' &  decreasing_fun f r r' &
    ~(is_constant_function f).
Proof.  ir. nin H1. nin H1. nin H2. ee. cp (cr_properties H). ee.
  set (f:= (fun u => Yo (inc u (cr_component r x1)) x x0)).
  assert (transf_axioms f (substrate r) (substrate r')). red. ir.
  uf f. nin (p_or_not_p (inc c (cr_component r x1))). ir. rww Y_if_rw.
  ir. rww Y_if_not_rw.
  set (g:= BL f  (substrate r) (substrate r')).
  assert (is_function g). uf g. app bl_function.
  assert (~ is_constant_function g). red. ir. red in H13. ee.
  assert (inc x1 (source g)). uf g. aw. elim H3.  app extensionality.
  wr H9. wr H8. app sub_class_substrate. am. red. ir.
  assert (inc x2 (source g)). uf g. aw. cp (H14 _ _ H15 H17).
  ufi g H18. awi H18. ufi f H18. rwi Y_if_rw H18.
  nin (p_or_not_p (inc x2 (cr_component r x1))). tv.
  rwii Y_if_not_rw H18. contradiction. wrr H9. bw.
  app reflexivity_e. rww H8.  nin H6; am. am. am. am.
```

We pretend that if $x \leq y$ then either both or none of x and y are in C. This is equivalent to say that if $x \sim y$ then either both or none of x and y is related to c, and is a consequence of the transitivity of $\sim$. It follows $g(x) = g(y)$, thus $g(x) \leq g(y)$, and $g(x) \geq g(y)$. This means that g is increasing and decreasing.

```
  assert (forall a b,
    gle r a b -> (inc a (cr_component r x1) = inc b (cr_component r x1))).
  ir. wrr H9. bw. bw. assert (comp_rel r a b). red. left.
  am. cp (H10 _ _ H15). app iff_eq. ir.
  apply transitivity_e with a.  am. am. am.
  ir.  apply transitivity_e with b.  am. am.  app symmetricity.
  assert (forall a b,
    gle r a b ->  W a g = W b g). ir. uf g. aw. uf f.
  nin (p_or_not_p (inc a (cr_component r x1))). ir. rww Y_if_rw. rww Y_if_rw.
  wrr (H14 _ _ H15). ir.  rww Y_if_not_rw. rww Y_if_not_rw.
  wrr (H14 _ _ H15). order_tac. order_tac.
  assert (target g = substrate r'). uf g. aw.
```

```
  assert (source g = substrate r). uf g. aw.
  exists g. ee. red. eee. ir. rww (H15 _ _ H18). order_tac.
  wr H16. app inc_W_target. rw H17. order_tac.
  red. eee. ir. red. rww (H15 _ _ H18).  order_tac.
  wr H16. app inc_W_target. rw H17. order_tac. am.
Qed.
```

---

8. *Let* E *and* F *be two ordered sets, let f be an increasing mapping of* E *into* F*, and g an increasing mapping of* F *into* E*. Let* A *(resp.* B*) be the set of all $x \in$ E *(resp. $y \in$ F*) such that $g(f(x)) = x$ (resp. $f(g(y)) = y$). Show that the two ordered sets* A *and* B *are canonically isomorphic.*

The restriction of the function f is a bijection from A onto B. it is clearly increasing.

```
Proof. ir. assert (forall x, inc x A -> inc (W x f) B). ir. ufi A H1. Ztac.
  clear H1. uf B. Ztac.  red in H. ee. rw H6. app inc_W_target. wrr H5.
  rww H3. assert (forall x, inc x B -> inc (W x g) A).  ir. ufi B H2. Ztac.
  clear H2. uf A. Ztac. red in H0. ee. rw H7. app inc_W_target. ue. ue.
  assert (Ha: source (restriction2 f A B) = A). uf restriction2. aw.
  assert (Hb: target (restriction2 f A B) = B). uf restriction2. aw.
  set (h:=restriction2 f A B).  assert (restriction2_axioms f A B).
  red. red in H. ee.  am.  uf A. wr H5. app Z_sub. wr H6. uf B. app Z_sub.
  red. ir. ufi image_by_fun H8.  awi H8. nin H8. nin H8. red in H9.
  wr (W_pr H H9). app H1. fprops.
  assert (is_function h). uf h. app function_restriction2.
  assert (injective h). red. split. am. uf h. rw Ha.
  intros x y Hx Hy. rww W_restriction2. rww W_restriction2. ir.
  ufi A Hx; ufi A Hy. Ztac. wr H7. clear Hy. Ztac. wr H5. sy; am.
  assert (surjective h). app surjective_pr6. uf h. rw Hb. rw Ha. ir.
  exists (W y g).  ufi restriction2 H6. awi H6.  split. app H2.
  rww W_restriction2.  ufi B H6. Ztac. am.  app H2. exists h.
  red in H. ee. assert (sub A (substrate r)). uf A. app Z_sub.
  assert (sub B (substrate r')). uf B. app Z_sub.
  red. ee. fprops. fprops. red; split;am. aw. aw. uf h. sy;am. uf h. rww Hb.
  aw. uf h. rw Ha. ir. rww W_restriction2. rww W_restriction2.
  app iff_eq. ir. awi H16. aw. app H11.
  app H1. app H1. am. am.
  ir. cp (H1 _ H14). cp (H1 _ H15). awi H16. aw.
  ufi A H14. Ztac. clear H14.  ufi A H15. Ztac.
  wr H20. wr H21. red in H0; ee. app H26. am. am.
```

---

9. ** If* E *is a lattice, prove that*

$$\sup_j(\inf_i x_{ij}) \leq \inf_i(\sup_j x_{ij})$$

*for every finite "double" family (x_{ij}). **

In a lattice, if a set has a supremum or an infimum, the same is true if we add an element.

```
Lemma lattice_finite_sup1: forall r X x a, lattice r ->
  sub X (substrate r) -> least_upper_bound r X x -> inc a (substrate r) ->
  least_upper_bound r (tack_on X a) (sup r x a).
Proof. ir. assert (order r).  red in H;ee;am.
  assert (sub (tack_on X a) (substrate r)). app tack_on_sub.
  rwi (least_upper_bound_pr x H3 H0) H1.  ee.
  assert (inc x (substrate r)). red in H1. ee. am.
  cp (lattice_sup_pr H H6 H2). ee.
  rw (least_upper_bound_pr (sup r x a) H3 H4). ee. red. ee.
  order_tac.  ir. rwi tack_on_inc H10. nin H10.  red in H1.
  ee.  cp (H11 _ H10). order_tac. ue.
  ir. red in H10. app H9. ee. app H5. red. ee. am. ir. app H11. fprops. eee.
Qed.

Lemma lattice_finite_inf1: forall r X x a, lattice r ->
  sub X (substrate r) -> greatest_lower_bound r X x -> inc a (substrate r) ->
  greatest_lower_bound r (tack_on X a) (inf r x a).
Proof. ir. assert (order r).  red in H;ee;am.
  assert (sub (tack_on X a) (substrate r)). app tack_on_sub.
  rwi (greatest_lower_bound_pr x H3 H0) H1.  ee.
  assert (inc x (substrate r)). red in H1. ee. am.
  cp (lattice_inf_pr H H6 H2). ee.
  rw (greatest_lower_bound_pr (inf r x a) H3 H4). ee. red. ee.
  app (inc_arg1_substrate H7).  ir. rwi tack_on_inc H10. nin H10.  red in H1.
  ee.  cp (H11 _ H10). order_tac. ue.
  ir. red in H10. app H9. ee. app H5. red. ee. am. ir. app H11. fprops. eee.
Qed.
```

A proof by induction shows that a finite set has a supremum and an infimum.

```
Lemma lattice_finite_sup2: forall r x, lattice r ->
  is_finite_set x -> nonempty x -> sub x (substrate r) ->
  has_supremum r x.
Proof. ir. app (finite_set_induction2 (fun x => (sub x (substrate r)))
  (fun x => has_supremum r x)).  ir.
  assert (inc a (substrate r)). app H3. fprops. wr (doubleton_singleton a).
  red in H. ee. nin (H5 _ _ H4 H4).  am. ir.
  assert (inc b (substrate r)). app H5. fprops.
  assert (sub a (substrate r)). apply sub_trans with (tack_on a b). fprops.
  am. cp (H3 H7 H4). red in H8.  nin H8. exists (sup r x0 b).
  ap (lattice_finite_sup1 H H7 H8 H6).
Qed.
Lemma lattice_finite_inf2: forall r x, lattice r ->
  is_finite_set x -> nonempty x -> sub x (substrate r) ->
  has_infimum r x.
Proof. ir. app (finite_set_induction2 (fun x => (sub x (substrate r)))
  (fun x => has_infimum r x)).  ir.
  assert (inc a (substrate r)). app H3. fprops. wr (doubleton_singleton a).
  red in H. ee. nin (H5 _ _ H4 H4).  am. ir.
  assert (inc b (substrate r)). app H5. fprops.
  assert (sub a (substrate r)). apply sub_trans with (tack_on a b). fprops.
  am. cp (H3 H7 H4). red in H8.  nin H8. exists (inf r x0 b).
  ap (lattice_finite_inf1 H H7 H8 H6).
Qed.
```

If F is a finite subset of E, then $x \leq \inf F$ if and only if $x \leq y$ for all $y \in F$.

```
Lemma lattice_finite_sup3: forall r x y, lattice r ->
  is_finite_set x -> nonempty x -> sub x (substrate r) ->
  gle r (supremum r x) y = (forall z, inc z x -> gle r z y).
Proof. ir. cp (lattice_finite_sup2 H H0 H1 H2).
  assert (order r). red in H; ee; am.
  cp (supremum_pr H4 H2 H3). ee. app iff_eq. ir. red in H5. ee.
  cp (H9 _ H8).order_tac.
  ir. app H6. red. ee. nin H1. cp (H7 _ H1).order_tac. am.
Qed.
Lemma lattice_finite_inf3: forall r x y, lattice r ->
  is_finite_set x -> nonempty x -> sub x (substrate r) ->
  gle r y (infimum r x) = (forall z, inc z x -> gle r y z).
Proof. ir. cp (lattice_finite_inf2 H H0 H1 H2).
  assert (order r). red in H; ee; am.
  cp (infimum_pr H4 H2 H3). ee. app iff_eq. ir. red in H5. ee.
  cp (H9 _ H8).  order_tac.
  ir. app H6. red. ee. nin H1. cp (H7 _ H1).  order_tac. am.
Qed.
```

If f is a family, with source I and target E, if I is finite, then $x \leq \inf_{i\in \mathrm{I}} f(i)$ if and only if $x \leq f(i)$ for all $i \in \mathrm{E}$.

```
Lemma lattice_finite_sup4: forall r f  y, lattice r ->
  fgraph f -> is_finite_set (domain f) -> nonempty (domain f) ->
  sub (range f) (substrate r) ->
  gle r (sup_graph r f) y = (forall z, inc z (domain f) -> gle r (V z f) y).
Proof. ir. uf sup_graph. rw lattice_finite_sup3.
  app iff_eq. ir. app H4. fprops. ir. srwi H5.
  nin H5. ee.  rw H6. app H4. am.  am. app finite_range. nin H2.
  exists (V y0 f). fprops. am.
Qed.

Lemma lattice_finite_inf4: forall r f  y, lattice r ->
  fgraph f -> is_finite_set (domain f) -> nonempty (domain f) ->
  sub (range f) (substrate r) ->
  gle r y (inf_graph r f) = (forall z, inc z (domain f) -> gle r y (V z f)).
Proof. ir. uf inf_graph. rw lattice_finite_inf3.
  app iff_eq. ir. app H4. fprops. ir. srwi H5.
  nin H5. ee.  rw H6. app H4. am.  am. app finite_range. nin H2.
  exists (V y0 f). fprops. am.
Qed.
Lemma lattice_finite_sup5: forall r f, lattice r ->
  fgraph f -> is_finite_set (domain f) -> nonempty (domain f) ->
  sub (range f) (substrate r) ->
  inc (sup_graph r f) (substrate r).
Proof. ir. assert (has_sup_graph r f). uf has_sup_graph.
  app lattice_finite_sup2.  app finite_range. nin H2.  exists (V y f).
  fprops. red in H. ee. cp (is_sup_graph_pr1 H H3 H4).
  rwi least_upper_bound_pr H6.  ee.  red in H6. ee. am. am. am.
Qed.
Lemma lattice_finite_inf5: forall r f, lattice r ->
  fgraph f -> is_finite_set (domain f) -> nonempty (domain f) ->
  sub (range f) (substrate r) ->
  inc (inf_graph r f) (substrate r).
Proof. ir. assert (has_inf_graph r f). uf has_inf_graph.
  app lattice_finite_inf2.  app finite_range. nin H2.  exists (V y f).
```

```
  fprops. red in H. ee. cp (is_inf_graph_pr1 H H3 H4).
  rwi greatest_lower_bound_pr H6.  ee.  red in H6. ee. am. am. am.
Qed.
```

Applying the previous lemmas in one direction gives $\inf_i x_{ij} \leq x_{ij} \leq \sup_j x_{ij}$; apply it in the other direction gives the result.

```
Lemma Exercice1_9: forall I1 I2 r f,
  lattice r -> fgraph f -> domain f = product I1 I2 ->
  is_finite_set I1 -> is_finite_set I2 -> nonempty I1 -> nonempty I2 ->
  sub (range f) (substrate r) ->
  gle r
  (sup_graph r (L I2 (fun j => inf_graph r (L I1 (fun i => V (J i j) f)))))
  (inf_graph r (L I1 (fun i => sup_graph  r(L I2 (fun j => V (J i j) f))))).
Proof. ir.  assert (Ha: order r). red in H; ee; am.
  assert (Hb: forall i j, inc i I1 -> inc j I2 ->
    inc (V (J i j) f) (substrate r)). ir. app H6. app inc_V_range. rw H1.
  fprops.
  rw lattice_finite_sup4. ir. bwi H7. rw lattice_finite_inf4. ir.
  bwi H8. bw.
  apply order_transitivity with  (V (J z0 z) f). am.
  set (fa:= (L I1 (fun i : Set => V (J i z) f))).
  assert (fgraph fa). uf fa. gprops. assert (is_finite_set (domain fa)).
  uf fa. bw. assert (nonempty (domain fa)). uf fa. bw.
  assert(sub (range fa) (substrate r)).  uf fa. red. ir. srwi H12.
  nin H12. ee. bwi H12. bwi H13. rw H13. app Hb. am. gprops.
  cp (lattice_finite_inf4 (inf_graph r fa) H H9 H10 H11 H12).
  assert (gle r (inf_graph r fa) (inf_graph r fa)). order_tac.
  app lattice_finite_inf5. rwi H13 H14.
  assert (V (J z0 z) f = V z0 fa). uf fa. bw. tv.  rw H15. app H14. uf fa. bw.
  set (fb:= L I2 (fun j : Set => V (J z0 j) f)).
  assert (fgraph fb). uf fb. gprops. assert (is_finite_set (domain fb)).
  uf fb. bw. assert (nonempty (domain fb)). uf fb. bw.
  assert(sub (range fb) (substrate r)).  uf fb. red. ir. srwi H12.
  nin H12. ee. bwi H12. bwi H13. rw H13. app Hb. am. gprops.
  cp (lattice_finite_sup4 (sup_graph r fb) H H9 H10 H11 H12).
  assert (gle r (sup_graph r fb) (sup_graph r fb)). order_tac.
  app lattice_finite_sup5. rwi H13 H14.
  assert (V (J z0 z) f = V z fb). uf fb. bw. tv. rw H15. app H14. uf fb. bw.
  am. gprops. bw. bw. red. ir. rwi frange_inc_rw H8. nin H8. bwi H8.  ee.
  rw H9. app lattice_finite_sup5. gprops.  bw.  bw.
  red. ir. srwi H10.  nin H10. nin H10. bwi H10. bwi H11.
  rww H11. app Hb. am. gprops. ee;am. gprops. am. gprops.  bw. bw.
  red. ir. srwi H7.  nin H7. nin H7. bwi H7. bwi H8. rw H8.
  app  lattice_finite_inf5. gprops. bw. bw. red. ir.
  srwi H9. nin H9. nin H9. bwi H9. bwi H10. rw H10. app Hb.
  am. gprops. ee;am. gprops.
Qed.
```

---

10. *Let* E *and* F *be two lattices. Then a mapping* f *of* E *into* F *is increasing if and only if*

$$f(\inf(x,y)) \leq \inf(f(x), f(y))$$

for all $x \in \mathrm{E}$ *and* $y \in \mathrm{E}$.

* *Give an example of an increasing mapping* f *of the product ordered set* $\mathbf{N} \times \mathbf{N}$ *into the orders set* $\mathbf{N}$ *such that the relation*

$$f(\inf(x,y)) = \inf(f(x), f(y))$$

is false for at least one pair $(x,y) \in \mathbf{N} \times \mathbf{N}$.*

If a, b, c, and d are integers we have $(a,b) \leq (c,d)$ if and only if $a \leq c$ and $b \leq d$. The infimum of these two quantities is $(\inf(a,c), \inf(b,d))$. The function $f : (x,y) \mapsto x+y$ is increasing. If we take $x = (1,0)$ and $y = (0,1)$, then $f(x) = f(y) = 1$, the infimum is $(0,0)$ and the value is 0; this is the counter example. The main result is straightforward.

```
Lemma exercise1_10: forall r r' f,
  lattice r ->  lattice r' -> is_function f -> substrate r = source f ->
  substrate r' = target f ->
  (increasing_fun f r r') =
  (forall x y, inc x (substrate r) -> inc y (substrate r) ->
    gle r' (W (inf r x y) f) (inf r' (W x f) (W y f))).
Proof. ir. ap iff_eq. ir.  red in H4; ee.
  destruct (lattice_inf_pr H H5 H6) as [Ha [Hb _ ]].
  assert (inc (inf r x y) (source f)). wr H2. order_tac.
  rwi H2 H5; rwi H2 H6. cp (H11 _ _ Ha). cp (H11 _ _ Hb).
  simpl in H13; simpl in H14. cp (inc_W_target H4 H5). cp (inc_W_target H4 H6).
  wri H10 H15; wri H10 H16. cp (lattice_inf_pr H0 H15 H16). ee. app H19.
  ir. cp H; nin H. red. eee. nin H0;am. ir.
  rwi H2 H4. assert (inc x (source f)). wr H2. order_tac.
  assert (inc y (source f)). wr H2. order_tac.
  cp (H4 _ _ H8 H9). rwi (inf_comparable1 H H7) H10.
  cp (inc_W_target H1 H8). cp (inc_W_target H1 H9). wri H3 H11; wri H3 H12.
  destruct (lattice_inf_pr H0 H11 H12) as [_ [Hb _ ]].
  nin H0. order_tac.
Qed.
```

---

11. *A lattice* E *is said to be* complete *if every subset of* E *has a least upper bound and a greatest lower bound in* E*; this means in particular that* E *has a greatest and a least element.*

(a) Show that if an ordered set E *is such that every subset of* E *has a least upper bound in* E*, then* E *is a complete lattice.*

The first claim is obvious: it suffices to take the supremum of infimum of the empty set. The second claim is easy: Let X be a set and X′ be the set of lower bounds. If X′ has a supremum, this is the infimum of X.

```
Definition complete_lattice r := order r &
  forall X, sub X (substrate r) -> (has_supremum r X & has_infimum r X).

Lemma exercise1_11a: forall r, complete_lattice r ->
  ((exists a, greatest_element r a) & (exists b, least_element r b)).
Proof. ir. nin H. assert (sub emptyset (substrate r)). ap sub_emptyset_any.
  nin (H0 _ H1). nin H2. nin H3. rwi (least_upper_bound_emptyset x H) H2.
  rwi (greatest_lower_bound_emptyset x0 H) H3.
```

```
  split. exists x0; am. exists x;am.
Qed.


Lemma exercise1_11b: forall r, order r ->
  (forall X, sub X (substrate r) -> has_supremum r X) ->
  complete_lattice r.
Proof. ir. red. split. am. ir. split. nin ( H0 _ H1). exists x. am. red.
  set (Z := (Zo (substrate r) (fun z => lower_bound r X z))).
  assert (sub Z (substrate r)). uf Z; app Z_sub. nin (H0 _ H2).
  exists x. rwi (least_upper_bound_pr x H H2) H3. nin H3.
  rw (greatest_lower_bound_pr x H H1). split. red.  split.
  nin H3. am. ir. app H4. red.  split.  app H1. ir. ufi Z H6. Ztac.
  nin H8.  app H9. ir. nin H3. app H6. uf Z. Ztac.  nin H5. am.
Qed.
```

(b) A product of ordered sets is a complete lattice if only if each of the factors is a complete lattice.

Let X be a subset of $\prod \mathrm{E}_i$ and $\mathrm{X}_i = \mathrm{pr}_i\mathrm{X}$. If each E_i is a complete lattice, then $\sup \mathrm{X}_i$ is the supremum of the set X_i, and $i \mapsto \sup \mathrm{X}_i$ is the supremum of X. Conversely, if the product is a complete lattice, it is non-empty since it has a smallest element. If $\mathrm{X}_i \subset \mathrm{E}_i$ we can find a set X with $\mathrm{X}_i = \mathrm{pr}_i\mathrm{X}$. If x is the supremum of X then x_i is the supremum of X_i.

```
Lemma exercise1_11c: forall f g, axioms_product_order f g ->
  (forall i, inc i (domain f) -> complete_lattice (V i g)) ->
  complete_lattice (product_order f g).
Proof. ir. app exercise1_11b. fprops.
  assert (Ha:substrate (product_order f g) = productb f).
  app substrate_product_order. assert (Hb:=H).  red in H; ee. ir.
  set (Xi := fun i=> (image_by_fun (pr_i f i) X)).
  assert (forall i, inc i(domain f) ->  sub (Xi i) (substrate (V i g))).
  ir. rw H4. uf Xi.  assert (target (pr_i f i) = (V i f)). uf pr_i. aw. wr H7.
  uf image_by_fun.
  assert (is_function (pr_i f i)). app function_pri.
  apply sub_trans with (range  (graph (pr_i f i))).  app sub_image_by_graph.
  fprops. app range_correspondence. nin H8; am. am.
  set (v:= L (domain f) (fun i => supremum (V i g)(Xi i))).
  assert (forall i, inc i(domain f) -> least_upper_bound (V i g) (Xi i)(V i v)).
  ir. nin (H0 _ H7). cp (H6 _ H7). nin (H9 _ H10). uf v. bw. app supremum_pr1.
  assert (inc v (substrate (product_order f g))). rw Ha.
  rw productb_pr. ee. uf v. gprops.  uf v. bw. ir. ufi v H8. bwi H8.
  nin (H7 _ H8). awi H9. Ztac. wrr H4. app H3. app Z_sub. am.
  exists v. rw least_upper_bound_pr. split. red. split. am.
  ir. rww related_product_order. rwi Ha H5; rwi Ha H8. ee. app H5. am.
  ir. cp (H7 _ H10). rwi least_upper_bound_pr H11. nin H11. red in H11. nin H11.
  app H13. uf Xi. uf image_by_fun. assert (is_function (pr_i f i)).
  app function_pri. aw. exists y. split. am.
  assert (inc y (productb f)). app H5. wr (W_pri H H10 H15). app defined_lem.
  uf pr_i. aw.
  fprops. app H6. ir. nin H9.  rww related_product_order. eee.
  ir. cp (H7 _ H11). rwi least_upper_bound_pr H12. nin H12.
  app H13. red. split. rw H4. rwi Ha H9. rwi productb_pr H9. ee. app H15.
  ue. am. am. ir. ufi Xi H14. awi H14. ufi image_by_fun H14. awi H14.
  nin  H14. ee. cp (H10 _ H14). rwi related_product_order H16. ee.
```

```
  assert (y = V i x). wr H15. wrr (W_pri (f:=f)).  ue. app H18. am.
  app function_pri. uf pr_i. aw. ue.
  assert (is_function (pr_i f i)). app function_pri. app H3. app H6.
  app order_product_order.  am.
Qed.

Lemma exercise1_11d: forall f g, axioms_product_order f g ->
  complete_lattice (product_order f g) ->
  (forall i, inc i (domain f) -> complete_lattice (V i g)).
Proof. ir. cp (exercise1_11a H0). nin H2. clear H3. nin H2. nin H2. clear H3.
  awi H2; try am. assert (Ha:=H). nin H. ee. app exercise1_11b. app H5. ir.
  rwi productb_pr H2. ee.
  set (Y:= L(domain f) (fun j=> Yo(j=i) X (singleton (V j x)))).
  assert (sub (productb Y) (substrate  (product_order f g))). aw.
  app productb_monotone1. uf Y. gprops. uf Y. bw. ir. uf Y. ufi Y H10. bwi H10.
  bw. nin (equal_or_not i0 i). rw Y_if_rw. wrr H6. ue. am.
  rw Y_if_not_rw. red. ir. rw (singleton_eq H12). app H9. ue. am.
  nin H0. cp (H11 _ H10).  nin H12. nin H12.
  rwi least_upper_bound_pr H12. ee. red in H12. ee. awi H12.
  assert (W x0 (pr_i f i) = V i x0). rww W_pri.
  ee. exists (W x0 (pr_i f i)). rw least_upper_bound_pr. split. red. ee.
  rww H6. rw H16. rwi productb_pr  H12.  ee.  app H18. rww H17. am.
  ir. assert (inc (L(domain f)(fun j=> (Yo (j = i) y (V j x)))) (productb Y)).
  rw productb_pr. bw. ee. gprops. uf Y. bw. ir. bw.
  nin (equal_or_not i0 i). rw H19. rw Y_if_rw. uf Y. bw. rw Y_if_rw. am. tv.
  tv. uf Y. bw. rw Y_if_not_rw. rw Y_if_not_rw. fprops. am. am. uf Y. gprops.
  cp (H15 _ H18). rwi related_product_order H19.  ee. cp (H21 _ H1).
  bwi H22. rwi Y_if_rw H22.  tv. rw H16. am. tv. am. am.
  ir. set (w:=  (L(domain f)(fun j=> (Yo (j = i) z (V j x))))).
  assert (inc w (productb f)). uf w. rw productb_pr. bw. ee. gprops. tv. ir.
  bw. nin (equal_or_not i0 i). rw H19. rw Y_if_rw.  red in H17. nin H17. wrr H6.
  tv. rw Y_if_not_rw. app H9. rww H8. am. am.
  assert (upper_bound (product_order f g) (productb Y) w). red. split.
  aw. ir. rw related_product_order. ee. awi H10. app H10. am. am. ir.
  uf w. bw. rwi productb_pr H19. ee. rwi H21 H22. ufi Y H22. bwi H22.
  cp (H22 _ H20). bwi H23. nin (equal_or_not i0 i). rw Y_if_rw.
  rwi Y_if_rw H23. red in H17. ee. rw H24. app H25. wrr H24. am. am.
  rw Y_if_not_rw. rwi Y_if_not_rw H23. rw (singleton_eq H23).
  wr order_reflexivity. rww H6. app H9.  rww H8. app H5. am. am. am.
  uf Y; gprops. am. cp (H14 _ H19). rwi related_product_order H20. ee.
  rw H16. cp (H22 _ H1). ufi w H23. bwi H23. rwi Y_if_rw H23. am. tv. am.
  am. app H5. am. am. am. am. am.
Qed.
```

(c) An ordinal sum (Exercise 3) $\sum_{\iota\in I} E_\iota$ *is a complete lattice if and only if the following conditions are satisfied:*
(I) I *is a complete lattice*
(II) If J *is a subset of* I *which has no greatest element, and if* $\sigma = \sup J$*, then* E_σ *has a least element.*
(III) For each $\iota \in I$ *every subset of* E_ι *which has an upper bound in* E_ι *has a least upper bound in* E_ι*.*
(IV) For each $\iota \in I$ *such that* E_ι *has no greatest element, the set of all* $\kappa > \iota$ *has a least element* α *and* E_α *has a least element.*

Condition III has to be replaced by: "every non-empty subset of E_ι which has an upper bound has a supremum" Example. Consider the sum of two sets, a singleton and the reverse

order of **N**. The empty set is bounded in **N** but has no greatest lower bound. This is now the theorem we try to show.

```
Definition greatest_induced r X x := greatest_element (induced_order r X) x.
Definition least_induced r X x := least_element (induced_order r X) x.

Lemma exercise1_11e: forall r f g, ordinal_sum_axioms1 r f g->
  complete_lattice (ordinal_sum r f g) =
  (complete_lattice r
    & (forall j, sub j (substrate r) ->
      ~ (exists u, greatest_induced r j u) ->
      exists v, least_element (V (supremum r j) g) v)
    & (forall i x, inc i (substrate r) -> sub  x (substrate (V i g)) ->
      (exists u, upper_bound (V i g) x u) ->
      (exists u, least_upper_bound  (V i g) x u))
    & (forall i, inc i (substrate r) ->
      ~ (exists u, greatest_element (V i g) u ) ->
      exists v, least_induced r (Zo (substrate r) (fun j =>
        glt r i j)) v
      & exists w, least_element (V v g) w)).
```

We recall that the ordinal sum is the set of all (i, x_i) where $i \in \mathrm{I}$ and $x_i \in \mathrm{E}_i$, and $(i, x_i) < (j, x_j)$ if either $i < j$ (in I) or $i = j$ and $x_i < x_j$ (in the ordered set E_i). We assume E_i non-empty. In particular, this allows us to define a function k such that $k(i) \in \mathrm{E}_i$, hence consider I is a subset of the sum.

```
Proof. ir. set  (E:= substrate r). set (F:= disjoint_union f).
  nin H. rename H0 into Hb.
  assert (Hc:order (ordinal_sum r f g)). app  order_ordinal_sum.
  assert (Ha: substrate (ordinal_sum r f g) = F). rww substrate_ordinal_sum.
  app iff_eq. ir. cp H.
  assert (Hd:forall i, inc i (domain f) -> exists y, inc y (V i f) &
    inc (J y i) (substrate (ordinal_sum r f g))). ir. nin (Hb _ H2).
  exists y. split. am. cp (inc_disjoint_union H2 H3). aw.
  set (k:= fun i => choose(fun y => inc y (V i f) &
    inc (J y i) (substrate (ordinal_sum r f g)))).
  assert (forall i, inc i (domain f) -> (inc (k i) (V i f)&inc (J (k i) i)F)).
  ir. wr Ha.  uf k. app choose_pr. app Hd.  nin H1. nin H0.
```

Let J be a subset of I, and consider the least upper bound x_j of k(J). Then j is the supremum of J.

```
 assert (forall j, sub j E -> exists x,
    least_upper_bound (ordinal_sum r f g) (fun_image j (fun i => J (k i) i)) x
    & inc x F &  least_upper_bound r j (Q x)).
  ir. ee. set (Y:= fun_image j (fun i=> J (k i) i)).
  assert (sub Y F). uf Y. red. ir. awi H11. nin H11. nin H11.
  assert (inc x0 (domain f)). wr H3. app H5. nin (H2 _ H13). ue.
  rwi Ha H4. nin (H4 _ H11). nin H12.  exists x.  split. am. awi H12. nin H12.
  assert (inc x F). red in H12. eee. eee.
  cp (du_index_pr H15). ee. aw. split. red. eee. ir. red in H12. ee.
  assert (inc (J (k y) y) Y). uf Y. aw. exists y. auto.
  cp (H20 _ H21). cp (related_ordinal_sum_order_id H H22). awi H23. am.
  ir. red in H19. ee. assert (inc (J (k z) z) F).  rwi H3 H19. nin (H2 _ H19).
  am. assert (upper_bound (ordinal_sum r f g) Y (J (k z) z)).
```

```
red. rw Ha. eee. ir. aw. eee.
ufi Y H22. awi H22. nin H22. nin H22. cp (H20 _ H22). wr H23. aw.
nin (equal_or_not x0 z).  rw H25. right. split. tv. wrr order_reflexivity.
rwi H3 H19. nin (H2 _ H19). rww H10. app H9. wrr H3. uf glt. au.
cp (H14 _ H22). cp (related_ordinal_sum_order_id H H23).  awi H24. am.
am. rww Ha.
```

Consider a subset J of I and the least upper bound x_j of k(J). Then j is the least upper bound of J. This shows (I), namely that I is a complete lattice. The quantity j is the supremum of J. Assume that J has no greatest element, so that $j \notin$ J. Every element in E_j is an upper bound of k(J). Thus x_j must be the least element of E_j. This shows (II).

```
ee. app exercise1_11b. ir. nin (H5 _ H11). exists (Q x). ee. am.
ir. nin (H5 _ H11). ee. assert (least_upper_bound r j ((supremum r j))).
uf supremum.  app choose_pr. exists (Q x). am.
rw (supremum_unique H1 H16 H15). clear H16. assert (~ inc (Q x) j).
red. ir. elim H12. exists (Q x). red. split. aw. aw. ir. aw. awi H15. ee.
nin H15. app H19. am. am. exists (P x). awi H13. ee.
cp (du_index_pr H14). ee. red. ee. rww H10. ir. rwi H10 H21.
assert (inc (J x0 (Q x)) F). uf F. app inc_disjoint_union.
assert (upper_bound (ordinal_sum r f g)
  (fun_image j (fun i : Set => J (k i) i)) (J x0 (Q x))).
red. aw. ee. am. ir.  aw. ee. awi H23. nin H23. ee. wr H24.
app inc_disjoint_union. wr H3. app H11. assert (inc x1 (domain f)).
wr H3. app H11. nin (H2 _ H25). am. am. awi H23.  nin H23. nin H23.
wr H24. aw. assert (x1 <> Q x). red. ir. elim H16. wrr H25. awi H15.
nin H15.  nin H15. left. split. app H27. am. am. am.
cp (H17 _ H23). awi H24. ee. nin H26. nin H26. elim H27. tv. nin H26. am.
am. am. am.  red. ir. awi H17. rw Ha. nin H17. nin H17. wr H18.
assert (inc x1 (domain f)). wr H3. app H11. nin (H2 _ H19). am.
```

Consider now a non-empty bounded subset X_i of E_i. We can consider this as a subset X of the sum. It has a least upper bound x_k. Since X_i is non-empty, there is $x_i \in \mathrm{X} \cap \mathrm{E}_i$ hence $i \leq k$. Since X_i has an upper bound, X has an upper bound in E_i hence $k \leq i$. Thus $k = i$. It follows that x_k can be considered as an element of X_i and is hence the supremum. This is point (III).

```
ir. assert (inc i (domain f)). wrr H3. rwi H10 H12.
nin H13. assert (inc  (J x0 i) F). uf F. app inc_disjoint_union.
red in H13. ee. wrr H10.
set (X := product x (singleton i)). assert (sub X F). red.
ir. ufi X H17. awi H17. ee. assert (J (P x1) (Q x1)= x1). app pair_recov.
wr H20. uf F. app inc_disjoint_union. rw H19. am.
rw H19. app H12. rwi Ha H4. nin (H4 _ H17).
nin H18. awi H18. ee. assert (upper_bound (ordinal_sum r f g) X (J x0 i)).
red. ee. rw Ha. am. ir. aw. ee. app H17. am. ufi X H21. awi H21. ee.
rw H23. right. split. tv. red in H13. ee. app H24.
assert (Q x1 = i).  cp (H20 _ H21). cp (related_ordinal_sum_order_id H H22).
awi H23. nin H14. assert (inc (J y i) X). uf X. aw. ee. fprops. am. fprops.
red in H18. ee. cp (H25 _ H24). cp (related_ordinal_sum_order_id H H26).
awi H27. order_tac. exists (P x1).
aw. nin H18.  rwi Ha H18. cp (du_index_pr H18). ee. red. ee. rww H10.
wrr H22. ir. assert (inc (J y i) X). uf X. aw.  ee. fprops. am. fprops.
cp (H23 _ H28). awi H29. ee. nin H31.  red in H31. ee.  elim H32. sy. am.
nin H31. am. am. ir. red in H27. ee. rwi H10 H27.
```

```
assert (inc (J z i) F). uf F. app inc_disjoint_union.
assert (upper_bound (ordinal_sum r f g) X (J z i)).
red. ee. rww Ha. ir. aw. ee. app H17. am. ufi X H30. awi H30. ee.
rw H32. right. split. tv. app H28. cp (H20 _ H30). awi H31.
ee. rwi H22 H33. nin H33. nin H33. elim H34. tv. nin H33. am. am. am.
app H9. rww H10. am. rw Ha. am. am.
```

Consider finally point (IV). Let $i \in \mathrm{E}$. Consider J, the subset of I formed of indices $j > i$; consider E_i as a subset X of the sum. Let x_k be its supremum. Since E_i is nonempty, there is $y_i \leq x_k$, hence $i \leq k$. If $i = k$, then y_k is the greatest element of E_i. Let's assume that E_i has no greatest element. This implies $k \in \mathrm{J}$. For every $j \in \mathrm{J}$, any element of E_j is an upper bound of X, this $k \leq j$. This means that k is the least element of J. Take $j = k$. Then x_k is the least element of E_k. This proves (IV).

```
ee. ir. set (X:= product (substrate (V i g)) (singleton i)).
assert (sub X F). red. ir. uf F. ufi X H13. awi H13. ee.
assert (J (P x) (Q x)= x). app pair_recov.
wr H16. app inc_disjoint_union.  rw H15. wrr H3.
rww H15. wrr H10. wrr H3. rwi Ha H4. nin (H4 _ H13).
nin H14. awi H14. nin H14. ufi E H11. rwi H3 H11. nin (Hb _ H11).
assert (inc (J y i) X). uf X. aw. ee.  fprops.  rww H10. fprops. nin H14.
set (Ii:=Zo E (fun j : Set => glt r i j)). assert (inc (Q x) Ii).
cp (H19 _ H18). awi H20. ee. uf Ii. nin H22. Ztac.  nin H22.
uf E. order_tac. nin H22. elim H12. exists (P x).
red. split. uf E. order_tac. ir.
assert (inc (J x0 i) X). uf X. aw. eee.
cp (H19 _ H25). awi H26. ee. nin H28. nin H28. elim H29. am. nin H28. am.
am. am. exists (Q x). split. red. red. aw. split. am. ir. aw. ufi Ii H21.
Ztac. ufi E H22. rwi H3 H22. nin (H2 _ H22).
assert (upper_bound (ordinal_sum r f g) X  (J (k x0) x0)). red. rw Ha.
split. am. ir. aw. ee. app H13. am. left. ufi X H26. awi H26. ee.
rww H28.  cp (H16 _ H26).
cp (related_ordinal_sum_order_id H H27). awi H28. am. uf Ii. app Z_sub.
exists (P x). red.  split. rwi Ha H14. cp (du_index_pr H14). ee.
rww H10. rw H10. ir.
assert (inc (J x0 (Q x)) F). uf F. app inc_disjoint_union. ufi Ii H20.
Ztac. wr H3. am.
assert (upper_bound (ordinal_sum r f g) X  (J x0 (Q x))). red. rw Ha.
split. am. ir. aw. ee. app H13. am. left. ufi X H23. awi H23. ee.
rww H25. ufi Ii H20. Ztac. am. cp (H16 _ H23). awi H24.
ee. nin H26. nin H26. elim H27. tv. nin H26. aw. am.  ufi Ii H20. Ztac.
wrr H3. am. rww Ha.
```

Assume the four assumptions true. Take a subset X. We have to show that it has a least upper bound. Let J be the set of indices i of all $(x_i) \in \mathrm{X}$. It has a least upper bound i, by assumption (I).

```
ir. ee. app  exercise1_11b. ir.
rwi Ha H4. cp H. red in H. ee.
set (j:= Zo E (fun i=> exists x, inc x X & i = Q x)).
assert (sub j E). uf j. app Z_sub.
nin H0. nin (H13 _ H12). clear H15. cp (supremum_pr1 H H12 H14).
```

Assume that J has a greatest element j. Each upper bound x_k of X satisfies $k \geq j$. Let X_j be the set of all elements x_k of X such that $k = j$.

```
nin (p_or_not_p (exists u, greatest_induced r j u)).
nin H16.  rename x into k. red in H16. red in H16. awi H16; try am. ee.
assert (Hd:forall z, upper_bound (ordinal_sum r f g) X z -> gle r k (Q z)).
ir. red in H18. nin H18. rwi Ha H18. ufi j H16. Ztac. nin H21. nin H21.
cp (H19 _ H21). rw H22. app (related_ordinal_sum_order_id H5 H23).
assert (He: inc k (domain f)). wr H6. app H12.
set (Xj:= Zo (substrate (V k g))
  (fun  y=> exists x, inc x X & y = P x & k = Q x)).
```

The set X_j is non-empty. Assume that it has an upper bound. We apply assumption (III). It says that X_j has a least upper bound x. Consider this as x_j in the sum. If $y_k \in X$, then $k \in J$ hence $k \leq j$. If $k < j$ then $y_k < x_j$, and if $k = j$ we have $y_k \leq x_j$ in X_j. Conversely, consider an upper bound of y_k of X. If $k > j$ then $x_j < y_k$. Otherwise, $y_k \in X_j$ and is an upper bound of X_j hence $x_j \leq y_k$.

```
assert (Hf:nonempty Xj).  ufi j H16. Ztac. clear H16. nin H19.
exists (P x). uf Xj. Ztac. ee. cp (du_index_pr (H4 _ H16)). ee.
rw H11. rww H19. rww H19.  exists x. eee.
nin (p_or_not_p (exists u, upper_bound (V k g) Xj u)).
ir.  assert (sub Xj (substrate (V k g))). uf Xj. app Z_sub.
nin (H2 _ _ (H12 _ H16) H19 H18). rwi least_upper_bound_pr H20. ee.
assert (inc (J x k) F). uf F. app inc_disjoint_union. nin H20. wrr H11.
exists (J x k). rw least_upper_bound_pr.
split. red. aw. split. am. ir. aw. ee. app H4. am.
cp (du_index_pr (H4 _ H23)). ee. assert (inc (Q y) j). uf j.
Ztac. ee. uf E. rww H6. exists y. auto. cp (H17 _ H27).
awi H28. nin (equal_or_not (Q y) k). right. split. am. nin H20.
rw H29. app H30. uf Xj. Ztac. rww H11. wrr H29.
exists y. auto. uf glt. au. am. am.
ir. aw. cp (Hd _ H23). nin H23. rwi Ha H23. fold F. eee.
nin (equal_or_not k (Q z)). right. split. am. app H21. red.
nin (du_index_pr H23). nin H28. rw H11. split. rww H26. ir.
ufi Xj H30. Ztac. nin H32. ee. cp (H25 _ H32). awi H35. ee.
wri H34 H37. wri H26 H37. nin H37. nin H37. elim H38. tv. rw H33. nin H37.
am. am. wr H6. app H12. uf glt. au. am. rw Ha. am. app H10. am. am.
```

Assume that X_j has no upper bound in E_j. Then E_j has no greatest element, and we can use (IV). It asserts existence of an index k, the least index such that $k > i$ and a least element x in E_k. This is obviously an upper bound of X. If we consider another upper bound z_t, we must have $t > j$, thus $t \geq k$. If $t = k$, we use $x \leq z_t$.

```
ir. nin (p_or_not_p (exists u, greatest_element (V k g) u)). ir.
nin H19. elim H18. exists x. nin H19. red. split. am. ir. app H20.
ufi Xj H21. Ztac. am. ir. nin (H3 _ (H12 _ H16) H19). nin H20. nin H21.
red in H21. rwi H11 H21. nin H21. nin H20. awi H20.
Ztac.  clear H20. awi H23.
assert (inc (J x0 x) F). uf F. app inc_disjoint_union. wrr H6.
exists (J x0 x). rw least_upper_bound_pr. split. red. rw Ha. split. am. ir.
aw. split. app H4. split. am. assert (inc (Q y) j). uf j.
nin (du_index_pr (H4 _ H26)). ee. Ztac. uf E. rw H6. am. exists y. auto.
cp (H17 _ H27). awi H28. left. nin H25. cp (order_transitivity H H28 H25).
split. am. red. ir. rwi H31 H28. elim H29.
order_tac. am. am.
ir. cp (Hd _ H26). nin H26. rwi Ha H26. nin (du_index_pr H26). nin H30.
aw. ee. am. am.
```

```
  assert (inc (Q z) (Zo E (fun j : Set => glt r k j))). Ztac. uf E. rww H6.
  split. am. red. ir. elim H18. exists (P z). split. rw H32. rww H11.
  ir. ufi Xj H33. Ztac. nin H35. ee. cp (H28 _ H35). awi H38. ee. nin H40.
  nin H40. elim H41. wr H32; wr H37; tv. nin H40. wri H37 H41. wri H36 H41. am.
  am. cp (H23 _ H32). awi H33. nin (equal_or_not x (Q z)). right. split. am.
  wri H34 H30. cp (H22 _ H30). awi H35. am. uf glt. au.
  clear H32. Ztac. am. am. rww Ha. am. app Z_sub. am. app Z_sub. wr H6.
  red in H20.  red in H20. nin H20. awi H20. Ztac. am. am. app Z_sub.
```

Assume finally that J has no greatest element. By assumption (II), the set E_i has a least element x. Consider the pair $x' = (i, x)$. It is a strict upper bound of X since i is a strict upper bound of J. Consider another upper bound, say $z' = (j, z)$. We have $i \leq j$. If $i < j$ it follows $x' < z'$. If $i = j$, we have $x \leq z$ in E_i hence $x' \leq z'$.

```
  nin (H1 _  H12 H16).
  set (a:= supremum r j). assert (inc a (domain f)). nin H15. awi H15. Ztac.
  wrr H6. am. app Z_sub.
  set (b:= J x a). assert (inc b F). uf F. uf b. app inc_disjoint_union.
  wrr H11. nin H17; ee; am. exists b. rw least_upper_bound_pr. split.
  red. split. rww Ha. ir. aw. ee.  app H4. am. left.
  uf b. aw. assert (inc (Q y) j). uf j. assert (inc y F). app H4.
  cp (du_index_pr H21). Ztac. uf E. rww H6. eee.  exists y. eee.
  rwi least_upper_bound_pr H15. ee. nin H15. split. app H23.
  red. ir. rwi H24 H21. elim H16. exists a. red. red. split. aw. aw. ir. aw.
  app H23. am. am. aw. ir. nin H20. ee.  rwi Ha H20.  cp (du_index_pr H20).
  rwi least_upper_bound_pr H15. ee.  assert (upper_bound r j (Q z)). red.
  ir. ee. rww H6. ir. ufi j H26. Ztac. nin H28. ee. rw H29. cp (H21 _ H28).
  ap (related_ordinal_sum_order_id H5 H30). cp (H25 _ H26). aw. eee.
  nin (equal_or_not (Q b) (Q z)). right. eee. uf b. aw. nin H17.
  app H29. rww H11. wri H28 H23. ufi b H23. awi H23. am. left. split.
  uf b. aw.  am. am. am. am. rww Ha.
Qed.
```

(d) The ordered set $\mathscr{A}(E,F)$ of increasing maps of an ordered set E *into an ordered set* F *(Exercise 6) is a complete lattice if and only if* F *is a complete lattice.*

Assume that $\mathscr{A}(E,F)$ is a complete lattice. If E is non-empty, then F is a complete lattice, see Exercise 6 (c).

```
Lemma exercise1_11f: forall r r', order r -> order r' ->
  nonempty (substrate r) ->
  complete_lattice  (increasing_mappings_order r r') -> complete_lattice r'.
Proof. ir. app exercise1_11b. ir. red.
  set (E:= substrate r). set (E':=substrate r').
  set (Y:= Zo (substrate (increasing_mappings_order r r'))
    (fun f => exists y, exists Hy: inc y X,
      f = constant_function E E' y (H3 y Hy))).
  assert (sub Y (substrate (increasing_mappings_order r r'))). uf Y. app Z_sub.
  nin H2. nin (H5 _ H4). nin H6. awi H6. nin H6.
  nin H6. rwi imo_substrate H6. cp H1. nin H1.
  rwi set_of_increasing_mappings_pr H6. nin H6. ee.
  set (u:= W y x). assert (inc u E').  uf u. uf E'. wr H12. app inc_W_target.
  rww H11. exists u. aw.  split. red.  ee. am. ir.
  set (f:= constant_function E E' y0 (H3 y0 H15)).
  assert (inc f Y). uf Y. Ztac. rw imo_substrate.
  uf f. uf E; uf E'. app constant_increasing. am. am.  exists y0. exists H15.
```

```
  tv. cp (H9 _ H16). rwi imo_pr H17. ee. red in H19. ee.
  cp (H25 _ H1). ufi f H26. awi H26. rwi W_constant H26. am. am. am. am.
  ir. red in H15. ee.
  set (f:= (constant_function (substrate r) (substrate r') z H15)).
  assert (inc f (set_of_increasing_mappings r r')). uf f.
  app constant_increasing.
  assert (upper_bound (increasing_mappings_order r r') Y f). red. ee.
  rww imo_substrate. ir. ufi Y H18. Ztac. nin H20. nin H20. rw H20.
  uf f. uf E. uf E'. wrr constant_increasing1. app H16. cp (H8 _ H18).
  rwi imo_pr H19. ee. red in H21. ee.  cp (H27 _ H1). ufi f H28. awi H28.
  rwi W_constant H28. am. am. am. am. am. am. am. am. am. uf Y. app Z_sub.
Qed.
```

Let's show the converse. We consider a subset X of $\mathscr{A}(E,F)$, and for each $x \in E$ the set G_x of all $f(x)$ for $f \in X$. If F is a complete lattice, this set has a lest upper bound, say f_x. This gives us a function $f : x \mapsto f_x$.

```
Lemma exercise1_11g: forall r r', order r -> order r' ->
  complete_lattice r' -> complete_lattice  (increasing_mappings_order r r').
Proof. ir. app exercise1_11b. ir. app imo_order. rww imo_substrate. ir.
  set (E:=substrate r). set (E':=substrate r').
  set (img:= fun x=> fun_image X (fun f => W x f)).
  assert (forall x,  inc x E -> sub (img x) E'). ir. uf img. red. ir. awi H4.
  nin H4. ee. wr H5. cp (H2 _ H4). awi H6. ee. uf E'.  wr H8. aw.
  app inc_W_target. rww H7. am. am.
  set (f:= fun x=> supremum r' (img x)).
  assert (forall x,  inc x E -> least_upper_bound r' (img x) (f x)).
  ir. uf f. app supremum_pr1. app H3. nin H1. nin (H5 _ (H3 _ H4)). am.
  assert (transf_axioms f E E'). red. ee. ir. cp (H4 _ H5). awi H6. nin H6.
  red in H6. nin H6. am. am. app H3.
  assert (is_function (BL f E E')). app bl_function.
```

This function f is increasing. Assume $a \le b$. We have $g(b) \le \sup_h h(b)$ if the supremum is over X and $g \in X$. Thus $g(a) \le g(b) \le f(b)$. Taking the supremum over g gives $f(a) \le f(b)$.

```
  assert (inc (BL f E E') (set_of_increasing_mappings r r')). aw. eee.
  red. eee. aw. aw. ir. assert (inc x E). uf E; order_tac.
  assert (inc y E). uf E; order_tac.
  rww W_bl_function. rww W_bl_function. cp (H4 _ H8). cp (H4 _ H9).
  awi H10. awi H11. ee. ap H13. red. red in H11. ee. am. ir.
  ufi img H15. awi H15. nin H15. ee.
  set (t:= W y x0). assert (inc t (img y)). uf img. aw.
  exists  x0. split. am. tv. cp (H14 _ H17). cp (H2 _ H15). awi H19. ee.
  red in H22. ee. cp (H27 _ _ H7). wr H16. order_tac.
  ufi t H18. am. am.  am. app H3. am. app H3.
```

It is easy to show that the function f is the supremum.

```
  exists (BL f E E'). aw. split. red. split. rww imo_substrate.
  ir. rww imo_pr. ee. app H2. am.
  cp (H2 _ H8). awii H9. nin H9. ee.
  red. eee. aw. aw. ir. rww W_bl_function. cp (H4 _ H13).
  awi H14.  ee. red in H14.  ee. app H16. uf img. aw. ex_tac. am. app H3.
  ir. red in H8. ee. rww imo_pr. rwii imo_substrate H8. eee.  awi H8.
  ee. red. eee. aw. aw. ir.  rw W_bl_function. cp (H4 _ H13). awi H14. ee.
```

```
  app H15. red. split. wr H11. app inc_W_target. rww H10. ir. ufi img H16.
  awi H16. nin H16. nin H16. cp (H9 _ H16). rwi imo_pr H18. ee. red in H20.
  ee. cp (H26 _ H13). ue. am.  am. am. app H3. am. am. am. am.
  app imo_order. rww imo_substrate.
Qed.
```

---

12. *Let* Φ *be a mapping of a set* A *into itself. Let* $\mathfrak{F}$ *be the subset of* $\mathfrak{P}(A)$ *consisting of all* $X \subset A$ *such that* $f(X) \subset X$ *for each* $f \in \Phi$*. Show that* $\mathfrak{F}$ *is a complete lattice with respect to the relation of inclusion.*

Let $X \subset \mathfrak{F}$. We show that $\bigcup X \in \mathfrak{F}$ and $\bigcap X \in \mathfrak{F}$ (if X is empty, the intersection is replaced by E).

```
Lemma Exercise1_12: forall E f, is_function f -> source f = E ->
  target f = E ->
  complete_lattice (inclusion_suborder (Zo (powerset E) (fun X =>
    sub (image_by_fun f X) X))).
Proof. ir. set (F:=Zo (powerset E) (fun X : Set => sub (image_by_fun f X) X)).
  assert (order (inclusion_suborder F)). ap subinclusion_is_order. red. split.
  am. ir. rwi substrate_subinclusion_order H3.
  assert (sub F (powerset E)). uf F. ap Z_sub. assert (sub X (powerset E)).
  apply sub_trans with F. am. am.
  assert (Ha:sub (union X) E). red. ir.
  nin (union_exists H6). nin H7. cp (H5 _ H8). rwi powerset_inc_rw H9. app H9.
  set (v:= Yo (nonempty X) (intersection X) E).
  assert (Hb: sub v E). uf v. nin (p_or_not_p (nonempty X)).
  rww Y_if_rw. nin H6. cp (intersection_sub H6). apply sub_trans with y. am.
  app powerset_sub. app H5. rww Y_if_not_rw. fprops.
  assert (inc v F). uf F. Ztac. app powerset_inc.
  uf v.  nin (p_or_not_p (nonempty X)). rww Y_if_rw. cp H6.  nin H6.
  red. ir. awi H8. nin H8. nin H8. app intersection_inc. ir.
  cp (H3 _ H10). ufi F H11. Ztac. app H13. aw. exists x0. split.
  app (intersection_forall H8 H10). am. rw H0. app powerset_sub. am. rw H0.
  assert (v = intersection X). uf v. rww Y_if_rw. wrr H9.
  rww Y_if_not_rw. red. ir. awi H7. nin H7. ee. wr H8. wri H0 H7. wr H1.
  fprops. am. rw H0. fprops.
  assert (inc (union X) F). uf F. Ztac. app powerset_inc. red. ir.
  awi H7. nin H7. nin H7. nin (union_exists H7). nin H9. cp (H3 _ H10).
  ufi F H11. Ztac. apply union_inc with x1. app H13. aw. exists x0. split.
  am. am.  rww H0.  app powerset_sub. am. am.  rww H0.
  split. red. exists (union X). app (union_is_sup1 H4 H3 H7).
  red. exists v. app (intersection_is_inf1 H4 H3 H6).
Qed.
```

---

13. *Let* E *be an ordered set. A mapping* f *of* E *into itself is said to be a* closure *if it satisfies the following conditions: (1)* f *is increasing, (2) for each* $x \in E$, $f(x) \geq x$, *(3) for each* $x \in E$, $f(f(x)) = f(x)$*. Let* F *be the set of elements of* E *which are invariant under* f*.*

(a) Show that for each $x \in E$ *the set* F_x *of elements* $y \in F$ *such that* $x \leq y$ *is not empty and has a least element, namely* $f(x)$*. Conversely, if* G *is a subset of* E *such that, for each* $x \in E$,

the set of all $y \in$ G *such that* $x \leq y$ *has a least element* $g(x)$*, then* g *is a closure and* G *is the set of elements of* E *that are invariant under* g*.*

(b) Suppose that E *is a complete lattice. Show that the greatest lower bound in* E *of any non-empty subset of* F *belongs to* F*.*

(c) Show that if E *is a lattice, then* $f(\sup(x,y)) = f(\sup(f(x), f(y)))$ *for each pair of elements* x*,* y *of* E*.*

We start with some definitions.

```
Definition is_closure  f r :=
  increasing_fun f r r  &
  (forall x, inc x (substrate r) -> gle r x (W x f)) &
  (forall x, inc x (substrate r) -> W (W x f) f = W x f).

Definition set_of_invariants f := Zo (source f) (fun x => W x f = x).
Definition set_of_upper_bounds F r x := Zo F (fun y => gle x y).
```

First part of (a) is trivial.

```
Lemma Exercise1_13a: forall f r x, is_closure f r ->
  let F := set_of_invariants f in
    inc x (source f) ->
    least_element (induced_order r (set_of_upper_bounds F r x)) (W x f).
Proof. ir. red in H. ee. red in H. ee. assert (inc (W x f) F). uf F.
  uf set_of_invariants. Ztac.  wr H5; rw H6; fprops. ap H2. rww H5.
  assert (sub (set_of_upper_bounds F r x) (substrate r)).
  uf set_of_upper_bounds. apply sub_trans with F. app Z_sub.
  uf F. uf set_of_invariants. wr H5. app Z_sub.
  uf  least_element. aw.
  assert (inc (W x f) (set_of_upper_bounds F r x)). uf set_of_upper_bounds.
  Ztac. app H1. rww H5. split. am. ir. aw. ufi set_of_upper_bounds H11.
  Ztac. ufi F H12. ufi set_of_invariants H12.  Ztac. wr H15.
  ap (H7 _ _ H13).
Qed.
```

Second part is a bit more complicated. Consider a set G. Let G_x be the set of upper bounds of x that are in G. Let $P_G(x,y)$ be the property that y is the least upper bound G_x. We have an assumption that for each x there is an y satisfying $P(x,y)$. We use the axiom of choice and define a function $y = g_G(x)$. This assumption says that $g_G(x) \in G_x$ and is the smallest element of this set (assertions Hb and Hc). The first assumption says that, if $y = g_G(x)$, then y belongs to E (so that there is a function $g : E \to E$), it belongs to G and $x \leq y$. If $x \in G$, then $x \in G_x$, assumption Hc says $y \leq x$, hence $y = x$. Conversely, if $x = y$ then $x \in G$ (since $y \in G$) Assume now $a \leq b$. Since $b \leq g(b)$ we have $g(b) \in G_a$, hence $g(a) \leq g(b)$. This shows that the function is increasing, thus is a closure.

```
Lemma Exercise1_13b: forall r G, order r -> sub G (substrate r) ->
  let g:= fun x => choose (fun y => least_element (induced_order r
    (set_of_upper_bounds G r x)) y) in
  (forall x, inc x (substrate r) -> exists y,
    least_element (induced_order r (set_of_upper_bounds G r x)) y) ->
  (is_closure (BL g (substrate r) (substrate r)) r &
    (G = set_of_invariants (BL g (substrate r) (substrate r)))).
Proof. ir.  assert (forall x,  inc x (substrate r) ->
  least_element (induced_order r (set_of_upper_bounds G r x)) (g x)).
```

```
  ir. uf g. app choose_pr. app H1. clear H1.
  set (E:= substrate r) in *.
  assert (Ha:forall x, inc x E -> sub (set_of_upper_bounds G r x) E).
  ir. uf set_of_upper_bounds. apply sub_trans with G. app Z_sub. am.
  assert (Hb:forall x, inc x E -> (inc (g x) (set_of_upper_bounds G r x))). ir.
  cp (H2 _ H1). red in H3.  ee. awi H3. am. am. app Ha.
  assert (Hc:forall x y, inc x E ->  inc y (set_of_upper_bounds G r x) ->
    gle r (g x) y).
  ir.  cp (H2 _ H1). red in H4.  ee. awi H5. cp (H5 _ H3). awi H6. am.
  awi H4. am. am.  app Ha. am. am. app Ha.
  assert (Hd:forall x, inc x E -> (inc (g x) E & inc (g x) G & gle r x (g x))).
  ir. cp (Hb _ H1). ufi set_of_upper_bounds H3. Ztac. app H0. am. am.
  assert(He:forall x, inc x G ->  (g x) = x). ir.
  assert (gle r (g x) x). app Hc. app H0. uf set_of_upper_bounds. Ztac.
  order_tac. app H0. assert (inc x E). app H0. cp (Hd _  H4).
  ee. order_tac.
  assert (transf_axioms g E E). red. ir. nin (Hd _ H1). am.
  assert (forall x, inc x E -> W x (BL g E E) = g x). ir. app W_bl_function.
  split. red. split. red. eee. app bl_function. aw. aw.
  ir. assert (inc y E). uf E. order_tac.
  assert (inc x  E). uf E. order_tac. rw H3. rw H3.
  cp (Hd _ H5). ee. app Hc. uf set_of_upper_bounds.
  Ztac. order_tac. am. am. split. ir. rw H3.
  cp (Hd _ H4). ee; am. am. ir.
  set (y:= W x (BL g E E)).  assert (y = g x). uf y. rww H3.
  cp (Hd _ H4). wri H5 H6. ee. rww H3. app He.
  uf set_of_invariants. set_extens. Ztac. aw. app H0. Ztac. rw H3. rww He.
  app H0. Ztac. awi H5. cp (Hd _ H5). rwi H3 H6. ee.  wrr H6. am.
Qed.
```

Converse. Assume that x is a lower bound of $\mathrm{E} \subset \mathrm{F}$. Then $x \leq y$ for all $y \in \mathrm{E}$, hence $f(x) \leq f(y)$. But $f(y) = y$ so that $f(x)$ is also a lower bound. If y is the least upper bound, we get $f(y) \leq y$; since $y \leq f(y)$ we get $f(y) = y$, hence $y \in \mathrm{F}$.

```
Lemma Exercise1_13c: forall f r E, is_closure f r -> complete_lattice r ->
  let F := set_of_invariants f in
    sub E F -> nonempty E -> inc (infimum r E) F.
Proof. ir.  nin H0. nin H. ee. nin H. ee.
  assert (sub F (substrate r)). uf F. uf set_of_invariants. rw H8. app Z_sub.
  assert (sub E (substrate r)). apply sub_trans with F. am. am.
  nin (H3 _ H12). cp (infimum_pr1 H0 H12 H14). rwi greatest_lower_bound_pr H15.
  ee. set (y:= infimum r E) in *.  assert (inc y (substrate r)). nin H15. am.
  assert (inc (W y f) (substrate r)). rw H9. app inc_W_target. wrr H8.
  assert (lower_bound r E (W y f)). red. split. am. ir. red in H15. ee.
  cp (H20 _ H19). cp (H10 _ _ H21).
  assert (inc y0 F). app H1. ufi F H23. ufi set_of_invariants H23. Ztac.
  wrr H25. cp (H16 _ H19). cp (H4 _ H17). uf F. uf set_of_invariants. Ztac.
  wrr H8. order_tac. am. am.
Qed.
```

Consider a set with the following elements eight elements t', x, y, z, x', y', z', T and z'. Look at the following table.

	t'	x	y	z	x'	y'	T	z'
t'	$\bar{.}$	<	<	<	<	<	<	<
x	>	.		<	<		<	<
y	>		.	<		<	<	<
z	>	>	>	.			<	<
x'	>	$\bar{>}$			$\bar{.}$		<	<
y'	>		$\bar{>}$			$\bar{.}$	<	<
T	>	>	>	>	>	>	.	<
z'	>	>	>	$\bar{>}$	>	>	$\bar{>}$	$\bar{.}$

The diagonal contains dots, and if there is < at position (a,b) we have > at position (b,a). Let's ignore for a moment the little bars.

Define the relation $a \prec b$ if $a = b$ or (a,b) contains <. This is a reflexive and antisymmetric relation. It is transitive. Proof. Assume $a \prec b$ and $b \prec c$. Let's show $a \prec c$. The result is true if $a = t$ or $c = z'$ (the smallest element is t', the greatest is z'). Let's ignore these elements. Then the result is true if $c =$ T (since T is the second greatest element in the list). There are four remaining pairs, (x,z), (x,x'), (y,y'), (y,z). Notice that it is not possible to find a, b such that one pair is (a,b) and another one is (b,c).

We pretend that the set is a lattice. If $a = b$, $a < b$ or $a > b$, then the pair (a,b) has an infimum and a supremum. There are twelve other pairs (a,b). In each case, t is a lower bound. It most cases, it is the unique lower bound, hence is the least upper bound. In the case of (x',z) there is a second lower bound x, which is the infimum. In the case of (y',z) there is a second lower bound y which is also the infimum. For each pair, T and z' are upper bounds; in general there are no other upper bounds so that T is the supremum. The only exception is (x,y) whose supremum is z.

For each column a there is a unique row b such that element (b,a) contains a little bar. This allows us to define a function $f : a \mapsto b$. Note that the bar is over a dot or a >, so that $f(a) \geq a$. Note that b is a primed letter, and if a is primed we have $a = b$. This is equivalent to $f(f(x)) = f(x)$. Note that f is increasing. If $a \leq b$ we have $f(a) \leq f(b)$. We can forget the cases $b = z$, $b = z'$ and $b =$ T since $f(b) = z'$, as well as the case $a = t'$. Remaining cases are trivially checked.

We have $f(\sup(x,y)) = f(z) = z'$ and $\sup(f(x),f(y)) = \sup(x',y') =$ T. More generally, if $z = \sup(x,y)$ and $\mathrm{T} = \sup(f(x),f(y))$, one has $z \leq \mathrm{T} \leq f(z)$, from which we can deduce $f(z) = f(\mathrm{T})$, but not $f(z) =$ T.

---

14. *Let* A *and* B *be two sets, and let* R *be any subset of* A × B. *For each subset* X *of* A *(resp. each subset* Y *of* B*) let* ρ(X) *(resp.* σ(Y)*) denote the set of all* $y \in$ B *(resp.* $x \in$ A*) such that* $(x,y) \in$ R *for all* $x \in$ A *(resp.* $(x,y) \in$ R *for all* $y \in$ B*). Show that* ρ *and* σ *are decreasing mappings and that the mapping* X → σ(ρ(X)) *and* Y → ρ(σ(Y)) *are closures (Exercise 13) in* $\mathfrak{P}$(A) *and* $\mathfrak{P}$(B) *respectively (ordered by inclusion).*

The definition has to be corrected as: "For each subset X of A (resp. each subset Y of B) let ρ(X) (resp. σ(Y)) denote the set of all $y \in$ B (resp. $x \in$ A) such that $(x,y) \in$ R for all $x \in$ X (resp. $(x,y) \in$ R for all $y \in$ Y)."

```
Lemma Exercice1_14: forall A B R,
  let rho := fun X => Zo B (fun y => forall x,inc x X -> inc (J x y) R) in
    let sigma := fun Y => Zo A (fun x => forall y, inc y Y -> inc (J x y) R) in
      let fr:=BL rho (powerset A) (powerset B) in
        let fs:= BL sigma (powerset B) (powerset A) in
          let iA := inclusion_order A in
            let iB := inclusion_order B in
              sub R (product A B) ->
              ( decreasing_fun fr iA iB & decreasing_fun fs iB iA &
                is_closure (compose fs fr) iA & is_closure (compose fr fs) iB).
```

We start by showing that σ, ρ, $\sigma \circ \rho$ and $\rho \circ \sigma$ are functions.

```
Proof. ir. assert (transf_axioms rho (powerset A) (powerset B)).
  red. ir. uf rho. app powerset_inc. app Z_sub.
  assert (transf_axioms sigma (powerset B) (powerset A)).
  red. ir. uf sigma. app powerset_inc. app Z_sub.
  assert (is_function fr). uf fr. app bl_function.
  assert (is_function fs). uf fs. app bl_function.
  assert (composable fs fr). red. uf fs. uf fr. aw.
  assert (composable fr fs). red. uf fs. uf fr. aw.
  assert (is_function (compose fs fr)). app is_function_compose.
  assert (is_function (compose fr fs)). app is_function_compose.
```

We show these four functions are monotone.

```
  assert (forall u v,  sub u v -> sub (rho v) (rho u)).
  ir. uf rho. red. ir. Ztac. clear H9. Ztac.
  assert (forall u v, sub u v -> sub (sigma v) (sigma u)).
  ir. uf sigma. red. ir. Ztac. clear H10. Ztac.
  assert (forall u v,  sub u v -> sub (sigma (rho u)) (sigma (rho v))).
  ir. app H9. app H8.
  assert (forall u v,  sub u v -> sub (rho (sigma u)) (rho (sigma v))).
  ir. app H8. app H9.
  assert (order iA). uf iA. app inclusion_is_order.
  assert (order iB). uf iB. app inclusion_is_order.
  assert (substrate iA = powerset A). uf iA. rww substrate_inclusion_order.
  assert (substrate iB = powerset B). uf iB. rww substrate_inclusion_order.
  split. red. eee. uf fr. aw. uf fr. aw. ir. ufi iA H16. awi H16. ee.
  uf fr. assert (inc x (powerset A)). app powerset_inc.
  assert (inc y (powerset A)). app powerset_inc.
  rww W_bl_function. rww W_bl_function.
  red. uf iB. aw. ufi iA H16. awi H16. ee. app powerset_sub. app H0.
  app powerset_sub. app H0. app H8.
  split. red. eee. . uf fs. aw. uf fs. aw. ir. ufi iB H16. awi H16. ee.
  uf fs. assert (inc x (powerset B)). app powerset_inc.
  assert (inc y (powerset B)). app powerset_inc.
  rww W_bl_function. rww W_bl_function. red. uf iA. aw.
  ee. app powerset_sub. app H1.
  app powerset_sub. app H1. app H9.
  assert (Ha: forall x, sub x A -> sub x (sigma  (rho x))).
  ir. red. ir. uf sigma. uf rho. Ztac. ir. Ztac. app H20.
  assert (Hb: forall x, sub x B -> sub x (rho  (sigma x))).
  ir. red. ir. uf sigma. uf rho. Ztac. ir. Ztac. app H20.
  assert (increasing_fun (compose fs fr) iA iA). red. eee.
  uf fr. aw. uf fs. aw.
```

```
  ir.  ufi iA H16. awi H16. ee.
  assert (inc x (powerset A)). app powerset_inc.
  assert (inc y (powerset A)). app powerset_inc.
  rww compose_W. rww compose_W. uf fr. aw.
  assert (inc (rho y) (powerset B)). app H0.
  assert (inc (rho x) (powerset B)). app H0. uf fs. aw. uf iA. aw.
  ee. app powerset_sub. app H1. app powerset_sub. app H1. app H10.
  uf fr. aw. uf fr. aw.
  assert (increasing_fun (compose fr fs) iB iB). red. eee.
  uf fs. aw. uf fr. aw.
  ir. ufi iB H17. awi H17. ee.
  assert (inc x (powerset B)). app powerset_inc.
  assert (inc y (powerset B)). app powerset_inc.
  rww compose_W. rww compose_W. uf fs. aw.
  assert (inc (sigma y) (powerset A)). app H1.
  assert (inc (sigma x) (powerset A)). app H1. uf fr. aw. uf iB. aw.
  ee. app powerset_sub. app H0. app powerset_sub. app H0. app H11.
  uf fs. aw. uf fs. aw.
```

We have $x \subset \sigma(\rho(x))$ and $y \subset \rho(\sigma(y))$. This implies $\rho\sigma\rho = \rho$ and $\sigma\rho\sigma = \sigma$ (cf. Proposition 2 of § 1, no. 5). The conclusion follows.

```
  split. red. ee. am. ir. rww compose_W. uf fr. rwi H14 H18. rww W_bl_function.
  uf fs. assert (inc (rho x) (powerset B)). app H0. rww W_bl_function.
  uf iA. aw. ee. app powerset_sub. red. ir. ufi sigma H20.  Ztac. am.
  app Ha. app powerset_sub.  uf fr. aw. wrr H14. ir.
  set (y:= W x (compose fs fr)). rwi H14 H18.
  assert (y  = sigma (rho x)). uf y. rw compose_W. uf fr. uf fs. aw.
  aw. app H0. am.  uf fr. aw.
  assert (inc y (powerset A)). rw H19. app H1. app H0.
  assert (rho y = rho x). rw H19. app extensionality. app H8. app Ha.
  app powerset_sub. app Hb. app powerset_sub. app H0.
  assert (sigma (rho y) = sigma (rho x)). rww H21.
  rw compose_W. uf fr. uf fs. aw. sy. rww H22. aw. app H0. am. uf fr. aw.
  red. ee. am. ir. rww compose_W. uf fs. rwi H15 H18. rww W_bl_function.
  uf fr. assert (inc (sigma x) (powerset A)). app H1. rww W_bl_function.
  uf iB. aw. ee. app powerset_sub. red. ir. ufi rho H20. Ztac. am.
  app Hb. app powerset_sub.  uf fs. aw.  wrr H15. ir.
  set (y:= W x (compose fr fs)). rwi H15 H18.
  assert (y  = rho (sigma x)). uf y. rw compose_W. uf fr. uf fs. aw.
  aw. app H1. am.  uf fs. aw.
  assert (inc y (powerset B)). rw H19. app H0. app H1.
  assert (sigma y = sigma x). rw H19. app extensionality. app H9. app Hb.
  app powerset_sub. app Ha. app powerset_sub. app H1.
  assert (rho (sigma y) = rho (sigma x)). rww H21.
  rw compose_W. uf fr. uf fs. aw. sy. rww H22. aw. app H1. am.  uf fs. aw.
Qed.
```

---

15. *(a) Let* E *be an ordered set, and for each subset* X *of* E *let* $\rho(X)$ *(resp.* $\sigma(X)$*) denote the set of upper (resp. lower) bounds of* X *in* E. *Show that, in* $\mathfrak{P}(E)$, *the set* $\tilde{E}$ *of subsets* X *such that* $X = \sigma(\rho(X))$ *is a complete lattice, and that the mapping* $i : x \to \sigma(\{x\})$ *is an isomorphism (called* canonical*) of* E *onto an ordered subset* E' *of* $\tilde{E}$ *such that, if a family* (x_ι) *of elements*

of E *has a least upper bound (resp. greatest lower bound) in* E*, the image of this least upper bound (resp. greatest lower bound) is the least upper bound (resp. greatest lower bound) in* $\tilde{\mathrm{E}}$ *of the family of images of the* x_ι*.* $\tilde{\mathrm{E}}$ *is called the* completion *of the ordered set* E*.*

(b) Show that, for every subset X *of* E, $\sigma(\rho(\mathrm{X}))$ *is the least upper bound in* $\tilde{\mathrm{E}}$ *of the subset* $i(\mathrm{X})$ *of* $\tilde{\mathrm{E}}$*. If* f *is any increasing mapping of* E *into a complete lattice* F*, there exists a unique increasing mapping* $\bar{f}$ *of* $\tilde{\mathrm{E}}$ *into* F *such that* $f = \bar{f} \circ i$ *and* $\bar{f}(\sup \mathrm{Z}) = \sup(\bar{f}(\mathrm{Z}))$ *for every subset* Z *of* $\tilde{\mathrm{E}}$*.*

(c) If E *is totally ordered, show that* $\tilde{\mathrm{E}}$ *is totally ordered.*

Let's start with some definitions.

```
Definition set_of_up_bounds r X :=
  Zo (substrate r)(fun z => upper_bound r X z).
Definition set_of_lo_bounds r X :=
  Zo (substrate r)(fun z => lower_bound r X z).
Definition set_of_uplo_bounds r X := set_of_lo_bounds r (set_of_up_bounds r X).
Definition completion r:=
  Zo (powerset (substrate r)) (fun z => z = set_of_uplo_bounds r z).
Definition completion_order r := inclusion_suborder (completion r).
```

Let $f = \sigma \circ \rho$. As a composition of two decreasing functions, it is increasing.

```
Lemma Exercise1_15a1: forall r A B, sub A B ->
  sub (set_of_up_bounds r B) (set_of_up_bounds r A).
Proof. ir. uf set_of_up_bounds. red. ir. Ztac. clear H0. Ztac.
  nin H2. split. am. ir. ap (H2 _ (H _ H3)).
Qed.

Lemma Exercise1_15a2: forall r A B, sub A B ->
  sub (set_of_lo_bounds r B) (set_of_lo_bounds r A).
Proof. ir. uf set_of_lo_bounds. red. ir. Ztac. clear H0. Ztac.
  nin H2. split. am. ir. ap (H2 _ (H _ H3)).
Qed.

Lemma Exercise1_15a3: forall r A B, sub A B ->
  sub (set_of_uplo_bounds r A) (set_of_uplo_bounds r B).
Proof. ir. uf set_of_uplo_bounds. app  Exercise1_15a2.
  app Exercise1_15a1.
Qed.
```

The function f is a closure (see Exercise 13). This means $x \subset f(x)$ and $f(f(x)) = f(x)$. If fact, we have $\sigma\rho\sigma = \sigma$.

```
Lemma Exercise1_15a4: forall r A, sub A (substrate r) ->
  sub A (set_of_uplo_bounds r A).
Proof. ir. red. ir. uf set_of_uplo_bounds. uf set_of_lo_bounds.
  Ztac. split. app H. ir. ufi set_of_up_bounds H1. Ztac. nin H3. app H4.
Qed.

Lemma Exercise1_15a5: forall r A, sub A (substrate r) ->
  set_of_lo_bounds r (set_of_up_bounds r (set_of_lo_bounds r A)) =
  (set_of_lo_bounds r A).
Proof. ir. app extensionality. app Exercise1_15a2.
  uf set_of_up_bounds. red. ir. Ztac. split. app H. ir.
  ufi set_of_lo_bounds H1. Ztac. nin H3. app H4.
```

```
  assert (sub (set_of_lo_bounds r A) (substrate r)). uf set_of_lo_bounds.
  app Z_sub. ap (Exercise1_15a4 H0).
Qed.

Lemma Exercise1_15a6: forall r A, sub A (substrate r) ->
  set_of_uplo_bounds r (set_of_uplo_bounds r A) =
  (set_of_uplo_bounds r A).
Proof. ir. uf set_of_uplo_bounds. rww Exercise1_15a5. uf set_of_up_bounds.
  app Z_sub.
Qed.
```

A first consequence if that, if X ⊂ E, then σ(X) and f(X) are in $\tilde{E}$.

```
Lemma Exercise1_15a7: forall r A, sub A (substrate r) ->
  inc (set_of_uplo_bounds r A) (completion r).
Proof. ir. uf completion. Ztac. uf set_of_uplo_bounds. uf set_of_lo_bounds.
  app powerset_inc. app Z_sub. rww Exercise1_15a6.
Qed.
Lemma Exercise1_15a8: forall r A, sub A (substrate r) ->
  inc (set_of_lo_bounds r A) (completion r).
Proof. ir. uf completion. Ztac. uf set_of_lo_bounds.
  app powerset_inc. app Z_sub. uf set_of_uplo_bounds. rww Exercise1_15a5.
Qed.
```

Assume now that E is an ordered set, and assume that A has a least upper bound α. Then σ(A) is the set of all x such that $x \leq \alpha$. If B has a least upper bound β, then σ(A) = σ(B) is α = β. In particular, if A and B are singletons we get A = B.

```
Lemma Exercise1_15a9: forall r x y,  order r ->
  inc x (substrate r) -> inc y (substrate r) ->
  (set_of_lo_bounds r (singleton x) = set_of_lo_bounds r (singleton y))
    -> x = y.
Proof. ir. assert (inc x (set_of_lo_bounds r (singleton y))).
  wr H2. uf set_of_lo_bounds. Ztac. split. am. ir. awi H3. rw H3. order_tac.
  assert (inc y (set_of_lo_bounds r (singleton x))).
  rw H2. uf set_of_lo_bounds. Ztac. split. am. ir. awi H4. rw H4. order_tac.
  ufi set_of_lo_bounds H3.  ufi set_of_lo_bounds H4. Ztac. clear H4.
  Ztac. nin H6. nin H7.  assert (inc x (singleton x)). fprops.
  cp (H8 _ H10). assert (inc y (singleton y)). fprops. cp (H9 _ H12).
  order_tac.
Qed.
```

If E has a least element e, then $\{e\}$ is the least element of $\tilde{E}$, otherwise ∅ is the least element $\tilde{E}$. In any case E is the greatest element.

```
Lemma Exercise1_15_comp_pr: forall r x, order r -> sub x (substrate r) ->
  inc (set_of_uplo_bounds r x) (completion r).
Proof. ir. uf completion. Ztac. app powerset_inc. uf set_of_uplo_bounds.
  uf set_of_lo_bounds. app Z_sub. sy. app Exercise1_15a6.
Qed.

Lemma Exercise1_15a10: forall r e, order r ->
  least_element r e ->
  least_element (completion_order r) (singleton e).
Proof. ir. assert (Ha:order (completion_order r)). uf completion_order. fprops.
```

```
  assert (Hb: substrate  (completion_order r) = completion r).
  uf completion_order. aw.
  assert (sub (singleton e) (substrate r)). red. ir. awi H1. rw H1. nin H0. am.
  assert(inc (singleton e) (completion r)).
  uf completion. Ztac. app powerset_inc. app extensionality.
  app Exercise1_15a4. red. ir. ufi set_of_uplo_bounds H2.
  ufi set_of_lo_bounds H2. Ztac. clear H2. nin H4.
  assert (least_element r x). nin H0. split. am. ir. app H4.
  uf set_of_up_bounds. Ztac. split. am. ir. awi H7. rw H7. app H5.
  rw (unique_least H H5 H0). fprops.
  split. ue. rw Hb. ir. uf completion_order. aw. eee. red. ir. awi H4.
  rw H4. ufi completion H3. Ztac.  clear H3. rw H6.
  uf set_of_uplo_bounds. uf set_of_lo_bounds. Ztac. nin H0; am.
  split. nin H0. am. ir. nin H0. app H7.
  ufi set_of_up_bounds H3. Ztac. am.
Qed.

Lemma Exercise1_15a11: forall r, order r ->
  ~ (exists e, least_element r e) ->
  least_element (completion_order r) emptyset.
Proof. ir. assert (Ha:order (completion_order r)). uf completion_order. fprops.
  assert (Hb: substrate  (completion_order r) = completion r).
  uf completion_order. aw.
  assert (inc emptyset (completion r)). uf completion. Ztac.
  app powerset_inc. app sub_emptyset_any. sy. app is_emptyset. ir. dneg.
  exists y. ufi set_of_uplo_bounds H1. ufi set_of_lo_bounds H1. Ztac.
  clear H1. split. am. ir. nin H3. app H4. uf set_of_up_bounds. Ztac.
  split. am. ir. elim (emptyset_pr H5).
  split. ue. rw Hb. ir. uf completion_order. aw. eee.
  app sub_emptyset_any.
Qed.

Lemma Exercise1_15a12 : forall r, order r ->
  exists x, least_element (completion_order r) x.
Proof. ir. nin (p_or_not_p (exists e, least_element r e)).
  nin H0. exists  (singleton x). app  Exercise1_15a10.
  exists emptyset. app Exercise1_15a11.
Qed.

Lemma Exercise1_15a13: forall r, order r ->
  greatest_element (completion_order r) (substrate r).
Proof. ir. assert (inc (substrate r) (completion r)). uf completion. Ztac.
  app powerset_inc. fprops. app extensionality. app Exercise1_15a4. fprops.
  uf set_of_uplo_bounds. uf set_of_lo_bounds. red. ir. Ztac. am.
  split. uf completion_order. aw. uf completion_order. aw. ir. aw. eee.
  ufi completion H1. Ztac. app powerset_sub.
Qed.
```

We show that the completion is a complete lattice. Given a family X_i, the least upper bound is $f(\bigcup X_i)$, the greatest lower bound is $f(\bigcap X_i)$. This is because f is increasing and $f(X_i) = X_i$. In the case of intersection, we have to consider the case where the family is empty; we show that E is the greatest element of $\tilde{E}$.

```
Lemma Exercise1_15a14: forall r X, order r -> sub X (completion r) ->
  least_upper_bound (completion_order r) X (set_of_uplo_bounds r (union X)).
Proof. ir. assert (Ha:order (completion_order r)). uf completion_order. fprops.
```

```
  assert (Hb: substrate  (completion_order r) = completion r).
  uf completion_order. aw.
  set (v :=set_of_uplo_bounds r (union X)).
  assert (inc v (completion r)). uf v. app Exercise1_15a7. red. ir.
  nin (union_exists H1). nin H2. cp (H0 _ H3). ufi completion H4. Ztac.
  rwi powerset_inc_rw H5. app H5.
  rw least_upper_bound_pr. split. split. ue. ir.
  uf completion_order. aw. ee. app H0. am. uf v.
  cp (H0 _ H2). ufi completion H3.  Ztac. rw H5. app Exercise1_15a3.
  app union_sub. ir. nin H2. uf completion_order. aw. rwi Hb H2. eee.
  uf v.  ufi completion H2. Ztac. rwi powerset_inc_rw H4. rw H5.
  app Exercise1_15a3. red. ir. nin (union_exists H6). nin H7.
  cp (H3 _ H8). ufi completion_order H9. awi H9. ee. app H11. am. ue.
Qed.

Lemma Exercise1_15a15: forall r X, order r -> sub X (completion r) ->
  nonempty X ->
  greatest_lower_bound (completion_order r) X
  (set_of_uplo_bounds r (intersection X)).
Proof. ir. assert (Ha:order (completion_order r)). uf completion_order. fprops.
  assert (Hb: substrate  (completion_order r) = completion r).
  uf completion_order. aw.
  set (v :=set_of_uplo_bounds r (intersection X)).
  assert (inc v (completion r)). uf v. app Exercise1_15a7. red. ir.
  nin H1. cp (intersection_forall H2 H1). cp (H0 _ H1). ufi completion H4.
  Ztac. rwi powerset_inc_rw H5. app H5.
  rw greatest_lower_bound_pr. split. split. ue. ir.
  uf completion_order. aw. ee. am. app H0. uf v. cp (H0 _ H3).
  ufi completion H4. Ztac. rwi powerset_inc_rw H5. rw H6.
  app Exercise1_15a3. red. ir. ap (intersection_forall H7 H3).
  ir. red in H3. ee. rwi Hb H3.  uf completion_order. aw. eee.
  ufi completion H3. Ztac. rw H6. uf v.  app Exercise1_15a3.
  red. ir. app intersection_inc. ir. cp (H4 _ H8). ufi completion_order H9.
  awi H9. ee. app H11. am. ue.
Qed.

Lemma Exercise1_15a16: forall r, order r ->
  complete_lattice (completion_order r).
Proof. ir. split. uf completion_order. fprops. ir.
  assert (Hb: substrate  (completion_order r) = completion r).
  uf completion_order. aw. split.
  exists (set_of_uplo_bounds r (union X)). app Exercise1_15a14. ue.
  nin (emptyset_dichot X). rw H1.
  exists (substrate r). rww greatest_lower_bound_emptyset. app Exercise1_15a13.
  uf completion_order. fprops.
  exists (set_of_uplo_bounds r (intersection X)). app Exercise1_15a15. ue.
Qed.
```

Let's study the property of $i(x) = \sigma(\{x\})$. We know that it is injective from E into $\tilde{E}$. Let's show that it is an order isomorphism on its image.

```
Lemma Exercise1_15a17: forall r, order r ->
  transf_axioms (fun z => set_of_lo_bounds r (singleton z))
  (substrate r) (substrate (completion_order r)).
Proof. ir. uf completion_order. aw. red. ir. app Exercise1_15a8.
  red. ir. awi H1. ue.
Qed.
```

```
Lemma Exercise1_15a18: forall r, order r ->
  order_morphism (BL  (fun z => set_of_lo_bounds r (singleton z))
    (substrate r) (substrate (completion_order r)))
  r (completion_order r).
Proof. ir. cp (Exercise1_15a17 H). ufi completion_order H0. awi H0.
  red. uf completion_order. aw. ee. am. fprops.
  app injective_bl_function. ir. app (Exercise1_15a9 H H1 H2 H3). bw. bw.
  simpl. ir. aw. ap iff_eq. ir. ee. app Exercise1_15a8. red. ir. awi H4. ue.
  app Exercise1_15a8. red. ir. awi H4. ue.
  red. ir. ufi set_of_lo_bounds H4. Ztac.  nin H6.
  uf set_of_lo_bounds. clear H4. Ztac. split. am. ir. awi H4. rw H4.
  assert (inc x (singleton x)). fprops. cp (H7 _ H8). order_tac.
  ir. ee. assert (inc x (set_of_lo_bounds r (singleton x))).
  uf set_of_lo_bounds. Ztac. split. am. ir. awi H6. rw H6. order_tac.
  cp (H5 _ H6). ufi set_of_lo_bounds H7. Ztac.  nin H9. app H10. fprops.
Qed.
```

Let's study the links between i and bounds.

```
Lemma Exercise1_15a19: forall r X x, order r ->
  sub X (substrate r) -> least_upper_bound r X x ->
  least_upper_bound  (completion_order r)
  (fun_image X (fun z => set_of_lo_bounds r (singleton z)))
  (set_of_lo_bounds r (singleton x)).
Proof. ir. set (Y:= fun_image X (fun z => set_of_lo_bounds r (singleton z))).
  assert (sub Y (completion r)). red. ir. ufi Y H2. awi H2. nin H2. nin H2.
  wr H3. app Exercise1_15a8. red. ir. awi H4. rw H4. app H0.
  cp (Exercise1_15a14 H H2).
  rwi least_upper_bound_pr H1. nin H1.
  set (t := set_of_lo_bounds r (singleton x)).
  assert (sub (union Y) t). red. ir. nin (union_exists H5). nin H6.
  ufi Y H7.  awi H7. nin H7. nin H7. wri H8 H6. ufi set_of_lo_bounds H6.
  Ztac. nin H10. nin H1. cp (H12 _ H7).
  assert (inc x2 (singleton x2)). fprops. cp (H11 _  H14).
  uf t. uf set_of_lo_bounds. clear H6. Ztac. split. order_tac. ir.
  awi H6. rw H6. order_tac.
  assert (sub (set_of_uplo_bounds r (union Y)) t).
  assert (set_of_uplo_bounds r t = t). uf t. uf set_of_uplo_bounds.
  rww Exercise1_15a5. red. ir. awi H6. ue. nin H1. am. wr H6.
  app Exercise1_15a3.
  assert (t = set_of_uplo_bounds r (union Y)). app extensionality.
  red. ir. ufi t H7. ufi set_of_lo_bounds H7. Ztac. clear H7.  nin H9.
  assert (inc x (singleton x)). fprops. cp (H9 _ H10).
  uf set_of_uplo_bounds. uf set_of_lo_bounds.  Ztac. split. order_tac.
  ir. ufi set_of_up_bounds H12. Ztac. nin H14.
  cut (gle r x y). ir. order_tac. app H4. split. am. ir. app H15.
  assert (inc y0 (set_of_lo_bounds r (singleton y0))).
  uf set_of_lo_bounds. clear H12. Ztac.  split. app H0. ir. awi H12. rw H12.
  order_tac. app H0.
  apply union_inc with (set_of_lo_bounds r (singleton y0)). am. uf Y.  aw.
  exists y0. ee. am. tv. ue. am. am.
Qed.
```

Case of greatest lower bound. If the family is empty and has a sup g, this is the greatest element of E, and $\sigma(\{g\}) = E$.

```
Lemma Exercise1_15a20: forall r X x, order r ->
  sub X (substrate r) -> greatest_lower_bound r X x ->
  greatest_lower_bound  (completion_order r)
  (fun_image X (fun z => set_of_lo_bounds r (singleton z)))
  (set_of_lo_bounds r (singleton x)).
Proof. ir. set (Y:= fun_image X (fun z => set_of_lo_bounds r (singleton z))).
  assert (sub Y (completion r)). red. ir. ufi Y H2. awi H2. nin H2. nin H2.
  wr H3. app Exercise1_15a8. red. ir. awi H4. rw H4. app H0.
  nin (emptyset_dichot Y). assert (X = emptyset). app is_emptyset. ir. red.
  ir. elim (emptyset_pr (x:=set_of_lo_bounds r (singleton y))). wr H3. uf Y.
  aw. ex_tac. rwi H4 H1. rwi greatest_lower_bound_pr H1. nin H1.
  assert(set_of_lo_bounds r (singleton x) = substrate r). ap extensionality.
  uf set_of_lo_bounds. app Z_sub. red. ir. uf set_of_lo_bounds. Ztac.
  split. am. ir. awi H7. rw H7. app H5. split. am. ir. elim (emptyset_pr H8).
  rw H6. rw H3. rw greatest_lower_bound_emptyset. app Exercise1_15a13.
  uf completion_order. fprops. am. app sub_emptyset_any.

  cp (Exercise1_15a15 H H2 H3).
  rwii greatest_lower_bound_pr H1. nin H1.
  set (t := set_of_lo_bounds r (singleton x)).
  assert (sub t (intersection Y)). red. ir. app intersection_inc.
  ufi t H6. ufi set_of_lo_bounds H6. Ztac. clear H6. nin H8.
  ir. ufi Y H9. awi H9. nin H9. nin H9. wr H10. uf set_of_lo_bounds. Ztac.
  split. am. ir. awi H11. rw H11. assert (inc x (singleton x)). fprops.
  cp (H8 _ H12). nin H1. cp (H14 _ H9). order_tac.
  assert (sub t (set_of_uplo_bounds r (intersection Y))).
  assert (set_of_uplo_bounds r t = t). uf t. uf set_of_uplo_bounds.
  rww Exercise1_15a5. red. ir. awi H7. ue. nin H1. am. wr H7.
  app Exercise1_15a3.
  assert (t = set_of_uplo_bounds r (intersection Y)). app extensionality.
  red. ir. ufi set_of_uplo_bounds H8. ufi set_of_lo_bounds H8. Ztac.
  clear H8. nin H10. uf t. uf set_of_lo_bounds. Ztac. split. am.
  ir. awi H11. rw H11. app H10. uf set_of_up_bounds. Ztac. nin H1. am.
  split. nin H1. am. ir. app H5. split.
  nin H3. cp (intersection_forall H12 H3). ufi Y H3. awi H3. nin H3. nin H3.
  wri H14 H13. ufi set_of_lo_bounds H13. Ztac. am.
  ir. assert (inc (set_of_lo_bounds r (singleton y1)) Y). uf Y. aw.
  ex_tac. cp (intersection_forall H12 H14). ufi set_of_lo_bounds H15. Ztac.
  nin H17. app H18. fprops. ue.
Qed.
```

Let's show $\sigma(\rho(X)) = \sup i\langle X\rangle$. Its proof is exactly the same as lemma 1.15a19.

```
Lemma Exercise1_15b1: forall r X, order r ->
  sub X (substrate r) ->
  least_upper_bound (completion_order r)
  (fun_image X (fun z => set_of_lo_bounds r (singleton z)))
  (set_of_uplo_bounds r X).
Proof. ir. set (Y:= fun_image X (fun z => set_of_lo_bounds r (singleton z))).
  assert (sub Y (completion r)). red. ir. ufi Y H1. awi H1. nin H1. nin H1.
  wr H2. ap Exercise1_15a8. red. ir. awi H3. rw H3. app H0.
  assert (set_of_uplo_bounds r (union Y) = set_of_uplo_bounds r X).
  uf set_of_uplo_bounds; uf set_of_lo_bounds; uf set_of_up_bounds. set_extens.
  Ztac. clear H2. nin H4. Ztac. split. am. ir. Ztac. clear H5. app H4. Ztac.
  nin H7. split. am. ir. nin (union_exists H8). nin H9.  ufi Y H10.
  awi H10. nin H10. nin H10. wri H11 H9. ufi set_of_lo_bounds H9. Ztac.
```

```
  nin H13. cp (H7 _ H10). assert (inc x1 (singleton x1)). fprops.
  cp (H14 _ H16). order_tac.
  Ztac. clear H2. nin H4. Ztac. split. am. ir. Ztac. nin H7. app H4.
  clear H5. Ztac. split. am. ir. app H8.
  apply union_inc with (set_of_lo_bounds r (singleton y0)).
  uf set_of_lo_bounds. Ztac. split. app H0. ir. awi H9. rw H9. order_tac.
  app H0. uf Y. aw. ex_tac. wr H2. ap (Exercise1_15a14 H H1).
Qed.
```

Consider now point (b): every increasing function f can be extended to $\bar{f}$ such that $f = \bar{f} \circ i$ and $\bar{f}(\sup Z) = \sup(\bar{f}(Z))$. This is wrong.

Assume that $E = \{a, b, c\}$ with, $a \leq c$ and $b \leq c$. Then $\tilde{E}$ has four elements, the singletons $\alpha = \{a\} = i(a)$ and $\beta = \{b\} = i(b)$, the least element $\emptyset$ and the greatest element E. Note that $E = i(c)$. The first assumption on $\bar{f}$ gives $\bar{f}(\alpha) = f(a)$, $\bar{f}(\beta) = f(b)$ and $\bar{f}(E) = f(c)$. The second assumption, for $Z = \{\alpha, \beta\}$, reads $f(\sup(a, b)) = \sup(f(a), f(b))$. This is not necessarily the case, so that $\bar{f}$ does not always exist.

Consider now (c): the set $\tilde{E}$ if tottally ordered whenever E is totally ordered. This follows from that fact that, if $X \in \tilde{E}$, $a \in X$ and $b \leq a$ then $b \in X$.

```
Lemma Exercise1_15c: forall r, total_order r ->
  total_order (completion_order r).
Proof. ir. nin H. split. uf completion_order. fprops. uf completion_order.
  assert (forall x a b, inc x (completion r) ->
    inc a x -> gle r b a -> inc b x).
  ir.  ufi completion H1. Ztac. rw H5. uf set_of_uplo_bounds.
  uf set_of_lo_bounds. uf set_of_lo_bounds. clear H1. Ztac. order_tac.
  split. order_tac. ir. ufi set_of_up_bounds H1. Ztac. nin H7. cp (H8 _ H2).
  order_tac.
  aw. ir. uf gge. aw. nin (p_or_not_p (sub x y)). left. eee. right. eee.
  red. ir. nin (inc_or_not x0 x). am. elim H4. red. ir. nin (inc_or_not x1 y).
  am. assert (inc x0 (substrate r)). ufi completion H3. Ztac.
  rwi powerset_inc_rw H9. app H9.
  assert (inc x1 (substrate r)). ufi completion H2. Ztac.
  rwi powerset_inc_rw H10. app H10. nin (H0 _ _ H9 H10).
  cp (H1 _ _ _ H2 H7 H11). contradiction.
  cp (H1 _ _ _ H3 H5 H11). contradiction.
Qed.
```

¶ 16. *A lattice* E *is said to be* distributive *if it satisfies the following two conditions*

(D') $$\sup(x, \inf(y, z)) = \inf(\sup(x, y), \sup(x, z))$$

(D") $$\inf(x, \sup(y, z)) = \sup(\inf(x, y), \inf(x, z))$$

for all x, y, z in E. *A totally ordered set is a distributive lattice.*

(a) Show that each of the conditions (D'), (D") separately implies the condition

(D) $$\sup(\inf(x, y), \inf(y, z), \inf(z, x)) = \inf(\sup(x, y), \sup(y, z), \sup(z, x))$$

for all x, y, z in E.

(b) Show that the condition (D) implies the condition

(M) $$\textit{If } x \geq z, \textit{ then } \sup(z, \inf(x,y)) = \inf(x, \sup(y,z))).$$

Deduce that (D) implies each of (D') and (D"), and hence that the three axioms (D), (D') and (D") are equivalent (to show, for example, that D *implies* D′*, take the least upper bound of x and each side of (D) and use (M)).*

(c) Show that each of the two conditions

(T') $$\inf(z, \sup(x,y)) \leq \sup(x, \inf(y,z)),$$

(T") $$\inf(\sup(x,y), \sup(z, \inf(x,y))) = \sup(\inf(x,y), \inf(y,z), \inf(z,x)),$$

for all x, y, z in E *is necessary and sufficient for* E *to be distributive. (To show that (T') implies (D"), consider the element*

$$\inf(z, \sup(x, \inf(y,z))).)$$

Let's introduce the following definitions.

```
Definition distributive_lattice1 r :=
  forall x y z, inc x (substrate r) ->inc y (substrate r) ->
    inc z (substrate r) ->
    sup r x (inf r y z) = inf r (sup r x y) (sup r x z).
Definition distributive_lattice2 r :=
  forall x y z, inc x (substrate r) ->inc y (substrate r) ->
    inc z (substrate r) ->
    inf r x (sup r y z) = sup r (inf r x y) (inf r x z).
Definition distributive_lattice3 r :=
  forall x y z, inc x (substrate r) ->inc y (substrate r) ->
    inc z (substrate r) ->
    sup r (inf r x y) (sup r (inf r y z) (inf r z x)) =
    inf r (sup r x y) (inf r (sup r y z) (sup r z x)).
Definition distributive_lattice4 r :=
  forall x y z, inc x (substrate r) ->inc y (substrate r) ->
    inc z (substrate r) ->
    gle r z x -> sup r z (inf r x y) = inf r x (sup r y z).
Definition distributive_lattice5 r:=
  forall x y z, inc x (substrate r) ->inc y (substrate r) ->
    inc z (substrate r) ->
    gle r (inf r z (sup r x y)) (sup r x (inf r y z)).
Definition distributive_lattice6 r :=
  forall x y z, inc x (substrate r) ->inc y (substrate r) ->
    inc z (substrate r) ->
    inf r (sup r x y) (sup r z (inf r x y))
    = sup r (inf r x y) (sup r (inf r y z) (inf r z x)).
```

Let's show the following trivial facts. In particular, we state associativity of sup and inf, which are implicit in the formulation (D).

```
Lemma lattice_props: forall r, lattice r ->
  let E := substrate r in
    ( (forall x y, inc x E->  inc y E -> inc (sup r x y) E)
      & (forall x y, inc x E->  inc y E -> inc (inf r x y) E)
      & (forall x y, inc x E->  inc y E -> sup r x y = sup r y x)
```

```
      & (forall x y, inc x E->  inc y E -> inf r x y = inf r y x)
      & (forall x y, inc x E->  inc y E -> sup r (inf r x y) y = y)
      & (forall x y, inc x E->  inc y E -> inf r (sup r x y) y = y)
      & (forall x y z, inc x E->  inc y E -> inc z E ->
        sup r x (sup r y z) = sup r (sup r x y) z)
      & (forall x y z, inc x E->  inc y E -> inc z E ->
        inf r x (inf r y z) = inf r (inf r x y) z)).
Proof. ir. uf E. ee.
  ir. cp (lattice_sup_pr H H0 H1). ee. order_tac.
  ir. cp (lattice_inf_pr H H0 H1). ee. order_tac.

  ir. cp (lattice_sup_pr H H0 H1). ee.
  cp (lattice_sup_pr H H1 H0). ee. nin H.
  assert (gle r (sup r x y) (sup r y x)). app H4.
  assert (gle r (sup r y x) (sup r x y)). app H7.
  order_tac.
  uf E. ir. cp (lattice_inf_pr H H0 H1). ee.
  cp (lattice_inf_pr H H1 H0). ee. nin H.
  assert (gle r (inf r x y) (inf r y x)). app H7.
  assert (gle r (inf r y x) (inf r x y)). app H4.
  order_tac.
  ir. app sup_comparable1. nin H. am. cp (lattice_inf_pr H H0 H1). ee. am.
  ir. cp (lattice_sup_pr H H0 H1). ee.
  assert (inc (sup r x y) (substrate r)). order_tac.
  cp (lattice_inf_pr H H5 H1). ee.
  assert (gle r y (inf r (sup r x y) y)). app H8. wrr order_reflexivity.
  nin H. am. nin H. order_tac.
```

Associativity of sup.

```
  ir.  cp (lattice_sup_pr H H0 H1). cp (lattice_sup_pr H H1 H2). ee.
  assert (inc  (sup r y z) (substrate r)). order_tac.
  assert (inc  (sup r x y) (substrate r)). order_tac.
  cp (lattice_sup_pr H H0 H9). cp (lattice_sup_pr H H10 H2). ee. nin H.
  assert (gle r (sup r x (sup r y z))(sup r (sup r x y) z)). ap H16.
  apply order_transitivity with(sup r x y). am. ap H3. ap H12. ap H6.
  apply order_transitivity with(sup r x y). am. ap H7. ap H12. ap H13.
  assert (gle r (sup r (sup r x y) z) (sup r x (sup r y z))). ap H14.
  ap H8. ap H11. apply order_transitivity with(sup r y z). am. ap H4. ap H15.
  apply order_transitivity with(sup r y z). am. ap H5. ap H15.
  order_tac.
```

Associativity of inf.

```
  ir.  cp (lattice_inf_pr H H0 H1). cp (lattice_inf_pr H H1 H2). ee.
  assert (inc  (inf r y z) (substrate r)).  order_tac.
  assert (inc  (inf r x y) (substrate r)).  order_tac.
  cp (lattice_inf_pr H H0 H9). cp (lattice_inf_pr H H10 H2). ee. nin H.
  assert (gle r (inf r x (inf r y z))(inf r (inf r x y) z)). ap H14.
  ap H8. ap H11. apply order_transitivity with(inf r y z). am. ap H15. ap H4.
  apply order_transitivity with(inf r y z). am. ap H15. ap H5.
  assert (gle r (inf r (inf r x y) z) (inf r x (inf r y z))). ap H16.
  apply order_transitivity with(inf r x y). am. ap H12. ap H3. ap H6.
  apply order_transitivity with(inf r x y). am. ap H12. ap H7. ap H13.
  order_tac.
Qed.
```

Let's show that (D') implies (D). We just expand and simplify, ad libitum.

```
Lemma Exercise1_16a: forall r, lattice r ->
  ( (distributive_lattice1 r -> distributive_lattice3 r) &
    (distributive_lattice2 r -> distributive_lattice3 r)).
Proof. ir.  set (E:= substrate r).
  cp (lattice_props H). simpl in H0.  fold E in H0. ee.
  rename H6 into sup_assoc. rename H7 into inf_assoc.
  ir. red in H6. red. fold E. fold E in H6. ir.
  set (sxy :=sup r x y). set (syz:=sup r y z). set (szx:= sup r z x).
  cp (H1 _ _ H8 H9). rw (H6 _ _ _ H10 H9 H7). rw (H2 _ _ H10 H7).
  rw (H6 _ _ _ H7 H8 H9). rw (H2 _ _ H7 H9). fold szx. fold sxy.
  rw (H4 _ _ H8 H9).
  assert (inf r z (inf r sxy szx) = inf r z sxy). cp (H0 _ _ H7 H8).
  cp (H0 _ _ H9 H7). uf sxy. uf szx. rw (H3 _ _ H11 H12).
  rw (inf_assoc _ _ _ H9 H12 H11). rw (H3 _ _ H9 H12). rw (H2 _ _ H9 H7).
  rw (H5 _ _ H7 H9). tv. rw H11. cp (H0 _ _ H7 H8). fold sxy in H12.
  rw (H6 _ _ _ (H1 _ _ H7 H8) H9 H12). rw (H2 _ _ (H1 _ _ H7 H8) H9).
  rw (H6 _ _ _ H9 H7 H8). fold szx. rw (H2 _ _  H9 H8). fold syz.
  rw (H2 _ _ (H1 _ _ H7 H8) H12). rw (H6 _ _ _ H12 H7 H8).
  assert (inf r (sup r sxy x) (sup r sxy y) = sxy).
  uf sxy.  cp (lattice_sup_pr H H7 H8). ee. ufi sxy H12.
  assert (sup r (sup r x y) x = (sup r x y)). rw (H2 _ _ H12 H7).
  app sup_comparable1. nin H;am. rw H16.
  assert (sup r  (sup r x y) y =  (sup r x y)). rw (H2 _ _ H12 H8).
  app sup_comparable1. nin H;am. rw H17. app inf_comparable1. nin H. am.
  wrr order_reflexivity. nin H; am. rw H13. rw H3. ap uneq. rw H3.
  app refl_equal. uf szx. app H0. uf syz. app H0. app H1. uf szx. app H0.
  uf syz. app H0. am.
```

Let's show that (D") implies (D). Same as above, exchanging sup and inf.

```
  rename H6 into sup_assoc. rename H7 into inf_assoc.
  ir. red in H6. red. fold E. fold E in H6. ir.
  set (sxy :=inf r x y). set (syz:=inf r y z). set (szx:= inf r z x).
  cp (H0 _ _ H8 H9). rw (H6 _ _ _ H10 H9 H7). rw (H3 _ _ H10 H7).
  rw (H6 _ _ _ H7 H8 H9). rw (H3 _ _ H7 H9). fold szx. fold sxy.
  rw (H5 _ _ H8 H9).
  assert (sup r z (sup r sxy szx) = sup r z sxy). cp (H1 _ _ H7 H8).
  cp (H1 _ _ H9 H7). uf sxy. uf szx. rw (H2 _ _ H11 H12).
  rw (sup_assoc _ _ _ H9 H12 H11). rw (H2 _ _ H9 H12). rw (H3 _ _ H9 H7).
  rw (H4 _ _ H7 H9). tv. rw H11. cp (H1 _ _ H7 H8). fold sxy in H12.
  rw (H6 _ _ _ (H0 _ _ H7 H8) H9 H12). rw (H3 _ _ (H0 _ _ H7 H8) H9).
  rw (H6 _ _ _ H9 H7 H8). fold szx. rw (H3 _ _  H9 H8). fold syz.
  rw (H3 _ _ (H0 _ _ H7 H8) H12). rw (H6 _ _ _ H12 H7 H8).
  assert (sup r (inf r sxy x) (inf r sxy y) = sxy).
  uf sxy.  cp (lattice_inf_pr H H7 H8). ee. ufi sxy H12.
  assert (inf r (inf r x y) x = (inf r x y)).
  app inf_comparable1. nin H;am. rw H16.
  assert (inf r (inf r x y) y = (inf r x y)).
  app inf_comparable1. nin H;am. rw H17. app sup_comparable1. nin H. am.
  wrr order_reflexivity. nin H; am. rw H13.
  assert (sup r syz szx= sup r szx syz). rw H2.  app refl_equal.
  uf syz. app H1.  uf szx. app H1. rw H14. rw H2.  app refl_equal. am.
  ap H0. uf szx. app H1. uf syz. app H1.
Qed.
```

Let's show that (D) implies (M). If $z \leq x$, we can evaluate $\sup(x,z)$ and $\inf(x,z)$ as z and x, then simplify further.

```
Lemma Exercise1_16b: forall r, lattice r ->
  ( (distributive_lattice3 r -> distributive_lattice4 r) &
    (distributive_lattice3 r -> distributive_lattice1 r) &
    (distributive_lattice3 r -> distributive_lattice2 r)).
Proof. ir.  set (E:= substrate r).
  cp (lattice_props H). simpl in H0.  fold E in H0.
  assert (distributive_lattice3 r -> distributive_lattice4 r). ee.
  rename H6 into sup_assoc. rename H7 into inf_assoc. ir. red in H6. red.
  fold E in H6. fold E. ir. cp (H6 _ _ _ H7 H8 H9).
  assert (inf r z x = z). app inf_comparable1. nin H; am. rwi H12 H11.
  assert (sup r z x = x). app sup_comparable1. nin H; am. rwi H13 H11.
  assert (sup r (inf r y z) z = z). app sup_comparable1. nin H; am.
  cp (lattice_inf_pr H H8 H9). ee. am. rwi H14 H11.
  assert (inf r (sup r y z) x = inf r x (sup r y z)). app H3. app H0.
  rwi H15 H11. rwi inf_assoc H11. assert (inf r (sup r x y) x = x).
  rww H3. app inf_comparable1. nin H; am.  cp (lattice_sup_pr H H7 H8).
  ee; am. app H0.  rwi H16 H11. wr H11. app H2. app H1. app H0. am. app H0.
```

Let's show that that (D) implies (D'). Write (D) as $\alpha = \beta$. Set $a = \sup(x,\alpha)$. This the supremum of four terms, two of them being smaller than x. After simplification, we see that a is the LHS of (D').

```
  ir. cp (H1 H9). rename H10 into HM. ufi  distributive_lattice3 H9.
  ufi  distributive_lattice4 HM. uf distributive_lattice1; fold E.
  set (a:= sup r x (sup r (inf r x y) (sup r (inf r y z) (inf r z x)))).
  set (b:= sup r x (inf r (sup r x y) (inf r (sup r y z) (sup r z x)))).
  assert (a= b). uf a; uf b. rww H9.
  set (c:= sup r (inf r y z) (inf r z x)).
  assert (a = sup r x (sup r (inf r x y) c)). uf a.  fold c. app refl_equal.
  rwi H7 H14. assert (sup r x (inf r x y) = x). rww H3. app  sup_comparable1.
  nin H; am. cp (lattice_inf_pr H H10 H11). ee. am. app H2. rwi H15 H14.
  ufi c  H14. clear H15. clear c.
  assert (sup r (inf r y z) (inf r z x) = sup r (inf r z x) (inf r y z)).
  app H3. app H2. app H2. rwi H15 H14. rwi H7 H14.
  assert (sup r x (inf r z x)=x). rww H3. app  sup_comparable1. nin H;am.
  cp (lattice_inf_pr H H12 H10). ee. am. app H2. rwi H16 H14. wr H14.
  clear H15. clear H16.  clear H14. rw H13. clear H13. clear a.
```

We must show that $\sup(x,\beta) = \inf(\sup(x,y),\sup(x,z))$, where β is a infimum. We can apply (M). We get $\sup(x,\beta) = \inf(\sup(x,y),d)$ where $d = \sup(\inf(\sup(y,z),\sup(z,x)),x)$. We can apply (M) again. We get $d = \inf(\sup(z,x),(\sup(x,y,z)))$. This simplifies to the first term.

```
  set (c:= inf r (sup r y z) (sup r z x)) in *.
  assert (inc (sup r x y) E). app H0. assert (gle r x (sup r x y)).
  cp (lattice_sup_pr H H10 H11). ee. am. assert (inc c E). uf c. app H2.
  app H0. app H0. uf b. rw (HM _ _ _ H13 H15 H10 H14). uf c.
  set (d:= sup r (inf r (sup r y z) (sup r z x)) x).
  assert (d= sup r x (inf r (sup r y z) (sup r z x))). rww H3.
  assert (d= sup r x (inf r (sup r z x) (sup r y z))). rw H16. rw H4.
  app refl_equal. app H0.  app H0. assert (inc (sup r z x) E). app H0.
  assert (inc (sup r y z) E). app H0. assert (gle r x (sup r z x)).
  cp (lattice_sup_pr H H12 H10). ee. am.
```

```
  rwi (HM _ _ _ H18 H19 H10 H20) H17.
  assert(d= (sup r z x)). rw H17. ap inf_comparable1. nin H; am. wr H7.
  set (e:=  sup r z x). assert (inc e E). uf e. app H0.
  cp (lattice_sup_pr H H11 H21). ee. am. am. am. am. rw H21.
  rw  (H3 _ _ H12 H10). app refl_equal. am. app H2. app H2. am. app H2.
  uf c. app H0. app H2. app H2.
```

The same argument show that (D) implies (D"). Step one. We simplify $\inf(x,\beta)$ to the RHS of (D").

```
  ir. cp (H1 H9). rename H10 into HM. ufi  distributive_lattice3 H9.
  ufi  distributive_lattice4 HM. uf distributive_lattice2; fold E.
  fold E in H9. fold E in HM. ir.
  set (a:= inf r x (sup r (inf r x y) (sup r (inf r y z) (inf r z x)))).
  set (b:= inf r x (inf r (sup r x y) (inf r (sup r y z) (sup r z x)))).
  assert (a= b). uf a; uf b. rww H9.
  set (c:= inf r (sup r y z) (sup r z x)).
  assert (b = inf r x (inf r (sup r x y) c)). uf b.  fold c. app refl_equal.
  rwi H8 H14. assert (inf r x (sup r x y) = x). rww H4.
  rww H4. app inf_comparable1. nin H; am. cp (lattice_sup_pr H H10 H11). ee. am.
  app H0. app H0. rwi H15 H14.  ufi c  H14. clear H15. clear c.
  assert (inf r (sup r y z) (sup r z x) = inf r (sup r z x) (sup r y z)).
  app H4. app H0. app H0. rwi H15 H14. rwi H8 H14.
  assert (inf r x (sup r z x)=x). ap inf_comparable1. nin H;am.
  cp (lattice_sup_pr H H12 H10). ee. am. rwi H16 H14. wr H14.
  clear H15. clear H16. clear H14. wr H13. clear H13. clear b.
```

Step two. We use (M) (in reverse order).

```
  set (c:= sup r (inf r y z) (inf r z x)) in *.
  assert (inc (inf r x y) E). app H2. assert (gle r (inf r x y) x).
  cp (lattice_inf_pr H H10 H11). ee. am. assert (inc c E). uf c. app H0.
  app H2. app H2. uf a. rw H3. wr (HM _ _ _ H10 H15 H13 H14). uf c.
  assert (inc (inf r y z) E). app H2. assert (gle r (inf r z x) x).
  cp (lattice_inf_pr H H12 H10). ee. am.  assert (inc (inf r z x) E). app H2.
  wr (HM _ _ _ H10 H16 H18 H17).
  assert (sup r (inf r z x) (inf r x (inf r y z)) = inf r x z).
  rw (H4 _ _ H10 H12). rw (H4 _ _ H11 H12). rww H8.  rw (H4 _ _ H10 H12).
  rw H3. ap sup_comparable1. nin H; am. cp (lattice_inf_pr H H18 H11). ee; am.
  am. app H2. rw H19. app refl_equal. am. am. am. app H0. app H0. am. app H0.
  uf c. app H2. app H0. app H0.
Qed.
```

Let's show that (D") implies (T') and conversely.

```
Lemma Exercise1_16c: forall r, lattice r ->
  (distributive_lattice3 r = distributive_lattice5 r).
Proof. ir. app iff_eq. ir. red. ir. cp (Exercise1_16b H). ee. cp (H6 H0).
  red in H7. rww H7. cp (lattice_props H).  simpl in H8.
  set (E:= substrate r) in *. ee. rw (H11 _ _ H3 H2).
  set (b:=inf r y z). assert (inc b E). uf b; app H9.
  assert (inc  (inf r z x) E). app H9. cp (lattice_sup_pr H H17 H16). ee.
  ap H20. assert (order r). nin H; am. assert (gle r (inf r z x)  x).
  cp (lattice_inf_pr H H3 H1). ee. am. assert (gle r x (sup r x b)).
  cp (lattice_sup_pr H H1 H16). ee. am. order_tac.
  cp (lattice_sup_pr H H1 H16). ee. am.
```

We show that (T') implies (D'). We first notice that the sup is smaller than the inf.

```
  ir. cut (distributive_lattice1 r). ir. cp (Exercise1_16a H). ee. ap H2. am.
  red. red in H0. cp (lattice_props H).  simpl in H1. ee.
  set (E:= substrate r) in *. ir. set (b:= sup r x (inf r y z)).
  assert (inc b E). uf b.  app H1. app H2.
  assert (gle r b (inf r (sup r x y) (sup r x z))).
  assert (inc (sup r x y) E). app H1.  assert (inc (sup r x z) E). app H1.
  cp (lattice_inf_pr H  H13 H14). ee. ap H17.  uf b.
  assert (inc (inf r y z) E). app H2. cp (lattice_sup_pr H H9 H18). ee.
  ap H21. cp (lattice_sup_pr H H9 H10). ee; am.
  assert (gle r (inf r y z) y). cp (lattice_inf_pr H H10 H11). ee; am.
  assert (gle r y (sup r x y)). cp (lattice_sup_pr H H9 H10). ee; am.
  nin H. order_tac. uf b.
  assert (inc (inf r y z) E). app H2. cp (lattice_sup_pr H H9 H18). ee.
  ap H21. cp (lattice_sup_pr H H9 H11). ee; am.
  assert (gle r (inf r y z) z). cp (lattice_inf_pr H H10 H11). ee; am.
  assert (gle r z (sup r x z)). cp (lattice_sup_pr H H9 H11). ee; am.
  nin H. order_tac.
```

Write (D') as $a = b$. We apply (T') to b and get $b \leq \sup(x, \inf(z, \sup(x, y)))$. According to (T'), the second term is $\leq \sup(x, \inf(y, z))$. This is a. Since sup is increasing, we get $b \leq \sup(x, a)$; but $a \geq x$, thus $b \leq a$.

```
  cp (H0 _ _ _ H9 H10 H11).
  cp (H0 _ _ _ H9 H11 (H1 _ _ H9 H10)).
  set (a:= inf r (sup r x y) (sup r x z)) in *.
  set (c:= inf r z (sup r x y)) in *.
  assert (sup r x b = b). ap sup_comparable1. nin H; am. uf b.
  cp (lattice_sup_pr H H9 (H2 _ _ H10 H11)). ee; am.
  assert (gle r (sup r x c) b). wr H16. assert (inc c E). uf c. app H2. app H1.
  cp (lattice_sup_pr H H9  H12). cp (lattice_sup_pr H H9 H17). ee.
  ap H21. ap H18. nin H. ap (order_transitivity H H14 H22).
  nin H. cp (order_transitivity H H15 H17).
  ap (order_antisymmetry H H13 H19).
Qed.
```

Let's show that (D") is equivalent to (T"). One implication is easy.

```
Lemma Exercise1_16d: forall r, lattice r ->
  (distributive_lattice3 r = distributive_lattice6 r).
Proof. ir.  cp (lattice_props H). ap iff_eq. ir.
  assert (distributive_lattice2 r). cp (Exercise1_16b H). ee. app H4.
  red in H2. red. simpl in H0. ir. ee. set (E:= substrate r) in *.
  cp (H0 _ _ H3 H4). cp (H6 _ _ H3 H4). rw (H2 _ _ _ H13 H5 H14).
  rw (H8 _ _ H13 H5). rw (H2  _ _ _ H5 H3 H4).  rw (H8 _ _ H4 H5).
  assert (inf r (sup r x y) (inf r x y) = inf r x y). rww H8.
  ap inf_comparable1. nin H; am.
  cp (lattice_sup_pr H H3 H4). cp (lattice_inf_pr H H3 H4). ee.
  nin H. order_tac. rw H15.
  set (xy:=inf r x y). set (zx:=inf r z x). set (zy:=inf r z y).
  rw H7. ap uneq. rw H7. ap uneq. tv. uf zx. app H6. uf zy. app H6.
  app H0.  uf zx. app H6. uf zy. app H6. app H14.
```

Conversely (T") (if the form $a = b$) implies (T') (of the form $a' \leq b'$), since $a' \leq a$ and $b \leq b'$ are clear.

```
  ir. rww Exercise1_16c.  red in H1. red. ir.
  simpl in H0. ir. ee. set (E:= substrate r) in *.
  set (a:=inf r z (sup r x y)). assert (a = inf r (sup r x y) z). rw H7. tv.
  app H0. am. assert (gle r a (inf r (sup r x y) (sup r z (inf r x y)))).
  rw H12. set (xy:= sup r x y). assert (inc xy E). uf xy. app H0.
  set (b:= inf r x y). assert (inc b E). uf b. app H5.
  assert (gle r z (sup r z b)). cp (lattice_sup_pr H H4 H14). ee. am.
  cp (lattice_inf_pr H H13 (H0 _ _ H4 H14)). ee.
  cp (lattice_inf_pr H H13 H4). ee.  ap H18. am. nin H.
  order_tac. rwi H1 H13.
  set (c:= sup r (inf r x y) (sup r (inf r y z) (inf r z x))) in *.
  assert (gle r c (sup r x (inf r y z))). uf c.
  set (xy:= inf r x y). assert (inc xy E). uf xy. app H5.
  set (yz:= inf r y z). assert (inc yz E). uf yz. app H5.
  set (zx:= inf r z x). assert (inc zx E). uf zx. app H5.
  assert (inc  (sup r yz zx) E). app H0.
  cp (lattice_inf_pr H H2 H3). fold xy in H18. ee.
  cp (lattice_inf_pr H H4 H2). fold zx in H21. ee.
  cp (lattice_sup_pr H H14 H17). ee. ap H26.
  cp (lattice_sup_pr H H2 H15). ee.  nin H. order_tac.
  cp (lattice_sup_pr H H2 H15). ee.
  cp (lattice_sup_pr H H15 H16). ee. ap H32. am.
  nin H. ap (order_transitivity H H22 H27).
  nin H. ap (order_transitivity H H13 H14).  am. am. am.
Qed.
```

¶ 17. *A lattice* E *which has a least element* α *is said to be* relatively complemented *if, for each pair of elements x, y of* E *such that* $x \leq y$*, there exists an element* x' *such that* $\sup(x, x') = y$ *and* $\inf(x, x,') = \alpha$*. Such an element* x' *is called a* relative complement *of x with respect to y.*

We define now a relatively complemented set, a Boolean lattice, the complement, and show that in a relatively complemented set, a complement does exists.

```
Definition relatively_complemented r:=
  lattice r & (exists u, least_element r u) &
  (forall x y, gle r x y -> exists x',
    (inc x' (substrate r)& sup r x x' = y & inf r x x' =  the_least_element r)).

Definition boolean_lattice r:=
  relatively_complemented r &  (exists u, greatest_element r u) &
  distributive_lattice3 r.

Definition the_complement r x y:=
   choose (fun x' =>  (inc x' (substrate r) &
     sup r x x' = y & inf r x x' =  the_least_element r)).

Lemma the_complement_pr: forall r x y,
  relatively_complemented r ->  gle r x y ->
  let x' := the_complement r x y in
    (inc x' (substrate r) & sup r x x' = y & inf r x x' =  the_least_element r).
Proof. ir. uf x'. uf the_complement. app choose_pr. red in H. ee. ir. app H2.
Qed.
```

Let's state a theorem that says what happens when we take the supremum or infimum of two elements, one of them being the least or greatest element.

```
Lemma least_greatest_pr: forall r, order r ->
  ((forall a, inc a (substrate r) -> (exists u, least_element r u) ->
    sup r (the_least_element r) a = a) &
    (forall a, inc a (substrate r) ->  (exists u, greatest_element r u) ->
      inf r a (the_greatest_element r) = a) &
    (forall a, inc a (substrate r) -> (exists u, least_element r u) ->
      inf r (the_least_element r) a = (the_least_element r)) &
    (forall a, inc a (substrate r) ->  (exists u, greatest_element r u)->
      sup r a (the_greatest_element r) = (the_greatest_element r))).
Proof. ir. ee.
  ir. app sup_comparable1. cp (the_least_element_pr H H1). nin H2. app H3.
  ir. app inf_comparable1. cp (the_greatest_element_pr H H1). nin H2. app H3.
  ir. app inf_comparable1. cp (the_least_element_pr H H1). nin H2. app H3.
  ir. app sup_comparable1. cp (the_greatest_element_pr H H1). nin H2. app H3.
Qed.
```

* *(a) Show that the set* E *of vector subspaces of a vector space of dimension* ≥ 2*, ordered by inclusion, is a relatively complemented lattice, but that if* x*,* y *are two elements of* E *such that* $x \leq y$*, there exists in general several distinct complements of* x *with respect to* y*.**

Assume that x is a subspace of y; consider a basis $(x_i)_{i\in \mathrm{I}}$ of y, such that for some $\mathrm{J} \subset \mathrm{I}$, $(x_i)_{i\in \mathrm{J}}$ is a basis of x. Let x' be the space spanned by $(x_i)_{i\in \mathrm{I}-\mathrm{J}}$. This is a relative complement. Fix $i \in \mathrm{I}-\mathrm{J}$. We may replace x_i by any element of y not in $x \cup x'$; this changes x', but it will remain a relative complement. An example of such an element is $x_i + x_j$ where $j \in \mathrm{J}$. It exists if x is neither of dimension 0 nor equal to y.

(b) If E *is distributive and relatively complemented, show that if* $x \leq y$ *in* E*, there exists a unique relative complement of* x *with respect to* y*.* E *is said to be a* Boolean lattice *if it is distributive and relatively complemented and if, moreover, it has a greatest element* ω*. For each* $x \in \mathrm{E}$*, let* x^* *be the complement of* x *with respect to* ω*. The the mapping* $x \to x^*$ *is an isomorphism of* E *onto the ordered set obtained by endowing* E *with the opposite ordering, and we have* $(x^*)^* = x$*. If* A *is any set, then the set* $\mathfrak{P}(\mathrm{A})$ *of all subsets* A*, ordered by inclusion, is a Boolean lattice.*

Let's show that in a distributive and relatively complemented set, there exists a unique complement. Consider x, complemented by x' and x'', and apply relation (D) to these three quantities. On the LHS, we have a supremum of infimums of three terms; two of them being the least element of E, the result is $\sup(x', x'')$. A similar argument says that the RHS is $\inf(x', x'')$, thus equality.

```
Lemma Exercice1_17a: forall r x y,  relatively_complemented r ->
  distributive_lattice3 r -> gle r x y ->
  exists_unique (fun x' => (inc x' (substrate r) &
      sup r x x' = y & inf r x x' = the_least_element r)).
Proof. ir. red. ee. exists (the_complement r x y). app the_complement_pr.
  ir. red in H. ee.  cp (lattice_props H). simpl in H10. red in H0.
  cp (inc_arg1_substrate H1). cp (inc_arg2_substrate H1).
  set (E:= substrate r) in *. ee.
  cp (H0 _ _ _ H3 H11 H2). rwi H6 H20. rwi H14 H4. rwi H4 H20. rwi H7 H20.
  rwi H15 H5. rwi H5 H20. assert (order r). nin H; am.
  cp (least_greatest_pr H21). ee.
  assert (sup r (the_least_element r) (inf r x0 y0) =  (inf r x0 y0)).
  rww H22. app H13. rwi H26 H20.  rwi H26 H20. clear H26.
```

```
  set (sxy:=sup r x0 y0). assert (inc sxy E). uf sxy. app H10.
  assert (inf r y (sup r x0 y0) = (sup r x0 y0)). rw H15. app inf_comparable1.
  cp (lattice_sup_pr H H2 H3). ee. app H29.
  cp (lattice_sup_pr H H11 H2). ee. wr H6. am.
  cp (lattice_sup_pr H H3 H11). ee.  wr H4. am. am. ufi sxy H26. am.
  rwi H27 H20. rwi H27 H20.
  cp (lattice_sup_pr H H2 H3). cp (lattice_inf_pr H H2 H3). ee.
  nin H. rwi H20 H29. rwi H20 H30.
  cp (order_transitivity H H28 H30).  cp (order_transitivity H H32 H29).
  ap (order_antisymmetry H H35 H36). am. am. am. am.
Qed.
```

Let's call "standard completion" and denote by x^* the completion with the greatest element of E. By uniqueness of completion and commutativity of supremum and infimum, we have $(x^*)^* = x$.

```
Definition standard_completion r x :=
  the_complement r x (the_greatest_element r).

Lemma standard_completion_pr: forall r x,
  let y := standard_completion r x in
    boolean_lattice r -> inc x (substrate r) ->
    (inc y (substrate r) & sup r x y = the_greatest_element r &
      inf r x y = the_least_element r).
Proof. ir. uf y.  uf standard_completion. app the_complement_pr.
  red in H; ee; am. red in H. ee. red in H. ee. red in H. ee.
  cp (the_greatest_element_pr H H1). red in H6. ee. app H7.
Qed.

Lemma standard_completion_involutive: forall r x,
  boolean_lattice r -> inc x (substrate r) ->
  standard_completion r (standard_completion r x) = x.
Proof. ir. cp (standard_completion_pr H H0). ee.
  set (y:=  standard_completion r x) in *.
  cp (standard_completion_pr H H1). ee.
  set (z:=  standard_completion r y) in *. red in H; ee.
  assert (lattice r).  red in H. ee. am.
  assert (gle r y (the_greatest_element r)).
  cp (lattice_sup_pr H9 H0 H1). ee.  wr H2. am.
  nin (Exercice1_17a H H8  H10). app H12. eee. eee.
  cp (lattice_props H9). simpl in H13. eee.
  cp (lattice_props H9). simpl in H13. eee.
Qed.
```

Consider two elements x and y, their standard completion a and b. Let $c = \inf(a,b)$. We have $\inf(y,c) = \alpha$. We have $\sup(y,c) = \sup(y,a)$ (we use (D′)). Assume $x \leq y$ so that $\sup(x,a) \leq \sup(y,a)$. Since $\sup(x,a) = \omega$ we deduce $\sup(y,c) = \omega$. As a consequence, c is the standard completion of y, hence $b = c = \inf(a,b)$. This shows $b \leq a$.

```
Lemma standard_completion_monotone: forall r x y,
  boolean_lattice r -> gle r x y ->
  gle r (standard_completion r y) (standard_completion r x).
Proof. ir. cp (inc_arg1_substrate H0).  cp (inc_arg2_substrate H0).
  cp (standard_completion_pr H H1). cp (standard_completion_pr H H2). ee.
  set (a:= standard_completion r x) in *.
  set (b:= standard_completion r y) in *.
```

```
  set (c := inf r a b). red in H; ee.
  assert (lattice r). red in H; ee; am. nin (lattice_props H11). ee.
  assert (inf r y c = the_least_element r). uf c. rw (H15 _ _ H3 H4).
  rw H19. rw H6. ap inf_comparable1. nin H11; am. nin H. ee. nin H.
  cp (the_least_element_pr H H20). red in H23. ee. app H24. am. am. am.
  assert (sup r y c = sup r y a). uf c. assert (distributive_lattice1 r).
  cp (Exercise1_16b H11).  ee. app H22. red in H21. rww H21. rw H5.
  nin H11. cp (least_greatest_pr H11). ee. app H24. app H12.
  assert (gle r (sup r x a) (sup r y a)).
  cp (lattice_sup_pr H11 H1 H3). cp (lattice_sup_pr H11 H2 H3). ee.
  ap H27. nin H11. order_tac. am. rwi H7 H22.
  assert (sup r y c = the_greatest_element r). rw H21. nin H11.
  nin (the_greatest_element_pr H11 H9).
  assert (inc (sup r y a) (substrate r)). app H12. cp (H25 _ H26).
  order_tac.
  cp (lattice_sup_pr H11 H2 H4). ee. rwi H5 H24.
  nin (Exercice1_17a H H10 H24). assert (c = b). ap H28. ee. uf c. app H13. am.
  am. ee. am. am. am. ufi c H29. cp (lattice_inf_pr H11 H3 H4). ee. wr H29.
  exact H30.
Qed.
```

We show that $x \mapsto x^*$ is an isomorphism. This is obvious since it is increasing (for the order on E and its reverse) and involutive.

```
Lemma Exercise1_17b: forall r, boolean_lattice r ->
  order_isomorphism (BL (standard_completion r) (substrate r)(substrate r))
  r (opposite_order r).
Proof. ir.
  assert (transf_axioms (standard_completion r) (substrate r) (substrate r)).
  red. ir. nin (standard_completion_pr H H0). am.
  assert (is_function (BL (standard_completion r) (substrate r) (substrate r))).
  app bl_function. assert (order r). red in H. ee. red in H; ee. nin H. am.
  red. ee. am. fprops. red. split. app injective_bl_function. ir.
  wr (standard_completion_involutive H H3).
  wr (standard_completion_involutive H H4). rww H5.
  app surjective_bl_function. ir.  cp (standard_completion_involutive H H3).
  exists  (standard_completion r y). split. app H0. am. aw. aw.
  aw. ir. rww W_bl_function.  rww W_bl_function. aw.
  app iff_eq. ir. app standard_completion_monotone. ir.
  wr (standard_completion_involutive H H3).
  wr (standard_completion_involutive H H4). app standard_completion_monotone.
Qed.
```

Let's show that $\mathfrak{P}(A)$ is a Boolean lattice. We also pretend that x^* is the complement of x in A. First step. We show that we have a least and a greatest element.

```
Lemma Exercise1_17c: forall a,
  (boolean_lattice (inclusion_order a) &
    (forall x, inc x (powerset a) ->
      standard_completion (inclusion_order a) x = complement a x)).
Proof. ir. set (r:=inclusion_order a). uf standard_completion.
  assert (substrate r = powerset a). uf r. rww substrate_inclusion_order.
  assert (order r). uf r. fprops.
  assert (least_element r emptyset). red. split. rw H. app powerset_inc.
  app sub_emptyset_any. ir. uf r. aw. rwi H H1. ee. app sub_emptyset_any.
  app powerset_sub. app sub_emptyset_any.
```

```
assert (exists u, least_element r u). exists emptyset. am.
assert (the_least_element r = emptyset). cp (the_least_element_pr H0 H2).
app (unique_least H0 H3 H1).
assert (greatest_element r a). red. split. rw H. app powerset_inc.
fprops. ir. uf r. aw. rwi H H4. rwi powerset_inc_rw H4. ee. am. fprops. am.
assert (exists u, greatest_element r u). exists a. am.
assert (the_greatest_element r = a). cp (the_greatest_element_pr H0 H5).
app (unique_greatest H0 H6 H4). rw H6.
```

Let's show that the powerset is a lattice, where sup and inf are union and intersection (note that we know that the powerset is a complete lattice).

```
assert (forall x y, inc x (powerset a) -> inc y (powerset a) ->
  least_upper_bound r (doubleton x y) (union2 x y)).
ir. assert (sub x a). app powerset_sub.  assert (sub y a). app powerset_sub.
assert (sub (union2 x y) a). red. ir. nin (union2_or H11). app H9. app H10.
app least_upper_bound_doubleton. uf r. aw. eee.
red. ir. app union2_first. uf r. aw. eee. red. ir. app union2_second.
ir. uf r. aw. ee. am. ufi r H12. awi H12. ee. am. red. ir.
nin(union2_or H14).  ufi r H12. awi H12. ee. app H17. ufi r H13. awi H13.
ee. app H17.
assert (forall x y, inc x (powerset a) -> inc y (powerset a) ->
  greatest_lower_bound r (doubleton x y) (intersection2 x y)).
ir. assert (sub x a). app powerset_sub.  assert (sub y a). app powerset_sub.
assert (sub (intersection2 x y) a). red. ir. app H10.
ap (intersection2_first H12).
app greatest_lower_bound_doubleton. uf r. aw. eee.
app intersection2sub_first. uf r. aw. eee. app intersection2sub_second.
red. ir. uf r. aw. ee. ufi r H13. awi H13. ee. am. red. ir.
ufi r H13. awi H13. ee. ap H16. ap (intersection2_first H15).
ufi r H13. awi H13. ufi r H14. awi H14. ee. red. ir. ap intersection2_inc.
app H18. app H16.
assert (forall x y, inc x (powerset a) ->inc y (powerset a) ->
  (has_supremum r (doubleton x y) & has_infimum r (doubleton x y))).
ir. split. exists (union2 x y). app H7. exists  (intersection2 x y). app H8.
assert (forall x y, inc x (powerset a) ->inc y (powerset a) ->
  sup r x y = union2 x y). ir.
cp (H7 _ _ H10 H11). assert (least_upper_bound r (doubleton x y) (sup r x y)).
uf sup. app supremum_pr1. red. rw H. ir. nin (doubleton_or H13).
rww H14. rww H14. exists (union2 x y). am. app (supremum_unique H0 H13 H12).
assert (forall x y, inc x (powerset a) ->inc y (powerset a) ->
  inf r x y = intersection2 x y). ir.
cp (H8 _ _ H11 H12). assert (greatest_lower_bound r (doubleton x y)
(inf r x y)). uf inf. app infimum_pr1. red. rw H. ir. nin (doubleton_or H14).
rww H15. rww H15. exists (intersection2 x y). am.
app (infimum_unique H0 H14 H13).
assert (Ha: lattice r). red. ee. am. rww H.
```

We show that the set is relatively complemented.

```
assert (Hd:forall x y, sub x y -> sub y a ->
  (inc  (complement y x) (substrate r) & sup r x  (complement y x) = y &
    inf r x  (complement y x) =  the_least_element r)).
ir. assert (sub x a). apply sub_trans with y. am. am.
assert (inc (complement y x) (powerset a)). app powerset_inc. red. ir.
srwi H15. ee. app H13. split. rw H. am. rw H3. split. rw H10.
```

```
  app union2_complement.  app powerset_inc. am. rww H11.
  app intersection2_complement. app powerset_inc.
  assert (Hb:relatively_complemented r). red. ee. am. am. ir.
  ufi r H12. awi H12. exists  (complement y x). app Hd. eee. eee.
```

We show distributivity via (T′).

```
  assert (Hc:distributive_lattice3 r).
  rww Exercise1_16c.  red. ir. rwi H H12; rwi H H13; rwi H H14.
  rw (H10 _ _ H12 H13). assert (inc  (union2 x y) (powerset a)).
  app powerset_inc. red. ir. nin  (union2_or H15).
  assert (sub x a). app powerset_sub. app H17.
  assert (sub y a). app powerset_sub. app H17.
  rw (H11 _ _ H14 H15).
  rw (H11 _ _ H13 H14). assert (inc (intersection2 y z) (powerset a)).
  app powerset_inc. red. ir. assert (sub y a). app powerset_sub. app H17.
  ap (intersection2_first H16). rw (H10 _ _ H12 H16). uf r. aw. ee.
  assert (sub z a). app powerset_sub. red. ir. app H17.
  ap (intersection2_first H18). red. ir. nin (union2_or H17).
  assert (sub x a). app powerset_sub. app H19.
  assert (sub (intersection2 y z) a). app powerset_sub. app H19.
  red. ir. nin (intersection2_both H17). nin (union2_or H19).
  app union2_first. app union2_second. app intersection2_inc.
```

The conclusion is now obvious.

```
  split. red. eee.
  ir. assert (gle r x a). uf r. aw. rwi powerset_inc_rw H12. ee. am. fprops. am.
  nin (Exercice1_17a Hb Hc H13). ee. cp (the_complement_pr Hb H13).
  nin H16. app H15. eee. rw H3. wr H3.
  assert (sub x a). app powerset_sub. assert (sub a a). fprops.
  ap (Hd _ _ H18 H19).
Qed.
```

(c) If E *is a* complete *Boolean lattice (Exercise 11), show that for each family* (x_λ) *of elements of* E *and each* $y \in$ E *we have*

$$\inf(y, \sup_\lambda(x_\lambda)) = \sup_\lambda(\inf(y, x_\lambda)).$$

(Reduce to the case $y = \alpha$*, and use the fact that if* $\inf(z, x_\lambda) = \alpha$ *for every index* λ*, then* $z^* \geq x_\lambda$ *for every* λ*).*

I do not understand the sentence "Reduce to the case $y = \alpha$", since if $y = \alpha$ the result is trivial. We start with four formulas of the type $\inf(y, \sup(y^*, x) = \inf(y, x)$.

```
Lemma Exercise1_17d: forall r x y, boolean_lattice r ->
  inc x (substrate r) ->  inc y (substrate r) ->
  let ys := (standard_completion r y) in
  (inf r y (sup r ys x) = inf r y x &
  sup r y (inf r ys x) = sup r y x &
  inf r ys (sup r y x) = inf r ys x &
  sup r ys  (inf r y x) = sup r ys x).
Proof. ir. cp H. nin H. nin H3. assert (order r). nin H. nin H. am.
  assert (lattice r). nin H; am. cp (lattice_props H6). simpl in H7.
  assert (forall x y,  inc x (substrate r) -> inc y (substrate r) ->
```

```
  inf r y (sup r (standard_completion r y) x) = inf r y x). ee. ir.
  cp (standard_completion_pr H2 H16). ee.
  set (z:=standard_completion r y0) in *. red in H. ee. cp (Exercise1_16b H).
  ee. cp (H24 H4). red in H25. rww H25. rw H19.
  cp  (least_greatest_pr H5). ee. rww H26. app H8.
  assert (forall x y,  inc x (substrate r) -> inc y (substrate r) ->
  sup r y (inf r (standard_completion r y) x) = sup r y x). ee. ir.
  cp (standard_completion_pr H2 H17). ee.
  set (z:=standard_completion r y0) in *. red in H. ee. cp (Exercise1_16b H).
  ee. cp (H24 H4). red in H26. rww H26. rw H19.
  cp (least_greatest_pr H5). ee. rww H11. rww H28. app H7.
  cp (the_greatest_element_pr H5 H3). red in H31. ee. am. app H7.
  ee.  uf ys. app H8. uf ys. app H9. cp (standard_completion_involutive H2 H1).
  fold ys in H17. wr H17. app H8. cp (standard_completion_pr H2 H1). ee. am.
  cp (standard_completion_involutive H2 H1).
  fold ys in H17. wr H17. app H9. cp (standard_completion_pr H2 H1). ee. am.
Qed.
```

Let $u = \sup_\lambda(\inf(y, x_\lambda))$ and $v = \inf(y, \sup_\lambda(x_\lambda)$. We consider two functional graphs f and g associated to x_λ and $\inf(y, x_\lambda)$, and state the properties of the sup and inf.

```
Lemma Exercise1_17e: forall r f y, boolean_lattice r -> complete_lattice r ->
  inc y (substrate r) -> fgraph f ->  sub (range f) (substrate r)->
  inf r y (sup_graph r f)
   = sup_graph r (L (domain f) (fun x => inf r y (V x f))).
Proof. ir. set (v:= inf r y (sup_graph r f)).
  set (g:= L (domain f) (fun x : Set => inf r y (V x f))).
  set (u:= sup_graph r g).
  red in H0. ee. assert (has_sup_graph r f). red. nin (H4 _ H3). am.
  assert (fgraph g). uf g. gprops.
  assert (sub (range g) (substrate r)). red. uf g. ir. rwi create_range H7.
  awi H7. nin H7. ee. wr H8. nin H. nin H. cp (lattice_props H).  simpl in H11.
  ee. app H12. app H3. aw. exists x0. app fdefined_lem. nin H2. am.
  assert (has_sup_graph r g). nin (H4 _ H7). am.
  cp (is_sup_graph_pr1 H0 H3 H5).
  cp (is_sup_graph_pr (sup_graph r f) H0 H3 H2). ufi  is_sup_graph H10.
  rwi H10 H9. clear H10.
  cp (is_sup_graph_pr1 H0 H7 H8).
  cp (is_sup_graph_pr (sup_graph r g) H0 H7 H6). ufi is_sup_graph H11.
  rwi H11 H10. clear H11. fold u in H10. ee.
  assert (lattice r). nin H. nin H. am.
  cp (lattice_inf_pr H15 H1 H9). fold v in H16. ee.
```

We show here $u \le v \le y$. This is rather obvious.

```
  assert (gle r u v). app H18. app H12. ir. uf g. ufi g H19.  bwi H19.
  rw create_V_rewrite. assert (inc (V a f) (substrate r)). app H3. aw.
  exists a. ap fdefined_lem. am. am. nin H2;am. cp (lattice_inf_pr H15 H1 H20).
  ee. am. am.  app H12. ir. uf g.  ufi g H19.  bwi H19. rw create_V_rewrite.
  assert (inc (V a f) (substrate r)). app H3. aw. exists a. ap fdefined_lem.
  am. am. nin H2;am. cp (lattice_inf_pr H15 H1 H20). ee.
  assert (gle r (V a f) (sup_graph r f)). app H13.
  order_tac. am.
```

Define $z = \sup(y^*, u)$. We have $\inf(y, z) = \inf(y, \sup(y^*, u)) = \inf(y, u) = u$.

```
set (z:= sup r (standard_completion r y) u).
assert (inf r y z = u). uf z. cp (Exercise1_17d H H10 H1). simpl in H20.
ee. rw H20. cp (lattice_props H15). simpl in H24. ee. rw H27.
app inf_comparable1. order_tac. am. am.
```

Let $z' = \sup(y^*, \sup_\lambda(x_\lambda))$. We have $z = \sup(y^*, \sup_\lambda(\inf(y, x_\lambda)))$. We have $z' \geq z$; we have also $z \geq z'$; this is because $z = \sup_\lambda(\sup y^*, \inf(y, x_\lambda))$, and we can simplify the expression.

```
assert (z = sup r (standard_completion r y) (sup_graph r f)).
uf z. set (ys:=  standard_completion r y) in *. uf u.
assert (inc ys (substrate r)). cp (standard_completion_pr H H1). ee. am.
cp (lattice_sup_pr H15 H21 H10).  cp (lattice_sup_pr H15 H21 H9). ee.
assert (gle r (sup r ys (sup_graph r g)) (sup r ys (sup_graph r f))).
ap H27. ap H23. ap H12. cp (lattice_props H15). simpl in H28. ee. app H28.
ir. assert (gle r (V a g) (sup_graph r f)).
assert (gle r  (V a g) (V a f)). uf g. ufi g H28. bwi H28. bw.
assert (inc (V a f) (range f)). fprops. cp (H3 _ H29).
cp (lattice_inf_pr H15 H1 H30). ee. am.  ufi g H28. bwi H28. cp (H13 _ H28).
order_tac.  order_tac.
assert (gle r (sup r ys (sup_graph r f)) (sup r ys (sup_graph r g))).
ap H25. ap H22. ap H14. cp (lattice_props H15). simpl in H29. ee. app H29.
ir. assert (gle r (inf r y (V a f)) (sup_graph r g)).
assert (inf r y (V a f) = V a g). uf g. bw. rw H30. app H11. uf g. bw.
assert (gle r (sup r ys (inf r y (V a f))) (sup r ys (sup_graph r g))).
cp (lattice_sup_pr H15 H21 (inc_arg1_substrate H30)).
cp (lattice_sup_pr H15 H21 (inc_arg2_substrate H30)). ee. ap H36. ap H32.
order_tac.
assert (inc (V a f) (range f)). fprops. cp (H3 _ H32).
cp (Exercise1_17d H H33 H1). simpl in H34. fold ys in H34. ee. rwi H37 H31.
cp (lattice_sup_pr H15  (inc_arg1_substrate H23) H33). ee.
order_tac. order_tac.
```

We have $\inf(y, z') = \inf(y, \sup(y^*, \sup(x_\lambda))) = \inf(y, \sup(x_\lambda)) = v$. Since $z' = z$, the conclusion is obvious.

```
wr H20. rw H21.  cp (Exercise1_17d H H9 H1). simpl in H22. ee. rw H22.
fold v. app refl_equal.
```

¶ 18. * *Let* A *be a set with at least three elements, let $\mathscr{P}$ be the set of all partitions of* A*, ordered by the relation "ϖ is finer than ϖ'" between ϖ and ϖ' (no 1, Example 4). Show that $\mathscr{P}$ is a complete lattice (Exercise 11), is not distributive (Exercise 17), but is relatively complemented (To prove the last assertion, well-order the sets belonging to a partition.)*$_*$

To each partition we associate an equivalence relation. Then "ϖ is finer than ϖ'" is the same as $x \sim y \implies x \sim' y$. The greatest lower bound is defined by the equivalence $x \sim y$ and $x \sim' y$. The least upper bound is the set of connected components of the relation $x \sim y$ or $x \sim' y$ (see Part I, exercise 6.10). From this; one could characterize the least upper bound of a family of partitions (this is a bit tricky, but hopefully, not needed here). We start by a lemma that says that if every family has a greatest lower bound, then the set is a complete lattice. We also show that if $\mathscr{P}$ is a complete lattice, then $\mathscr{P}$ with the opposite order is a complete lattice. As a consequence, we shall show that "ϖ is coarser than ϖ'" makes $\mathscr{P}$ is a complete lattice.

```
Lemma exercise1_11h: forall r, order r ->
  (forall X, sub X (substrate r) -> has_infimum r X) ->
  complete_lattice r.
Proof. ir. red. split. am. ir. split.
  set (Z := (Zo (substrate r) (fun z => upper_bound r X z))).
  assert (sub Z (substrate r)). uf Z; app Z_sub. nin (H0 _ H2).
  exists x. rwi (greatest_lower_bound_pr x H H2) H3. nin H3.
  rw (least_upper_bound_pr x H H1). split. red. split.
  nin H3. am. ir. app H4. red. split. app H1. ir. ufi Z H6. Ztac.
  nin H8. app H9. ir. nin H3. app H6. uf Z. Ztac. nin H5. am. ap (H0 _ H1).
Qed.

Lemma exercise1_11i: forall r,
  complete_lattice r -> complete_lattice (opposite_order r).
Proof. ir. red in H. ee. red. ee. fprops. rww substrate_opposite_order.
  ir. cp (H0 _ H1). nin H2. ee. nin H3. exists x. wrr inf_sup_opp.
  nin H2. exists x. wrr sup_inf_opp.
Qed.
```

Consider the set of all $A \cap B$ for $A \in \mathfrak{w}$ and $B \in \mathfrak{w}'$ We have shown (in Part I of this report) that this is the least upper bound (for the "coarser" ordering for coverings). When we remove the empty set, we get the least upper bound of two partitions. We first recall the property of the covering intersection.

```
(*
  Lemma intersection_covering2_pr: forall x y z,
    inc z (intersection_covering2 x y) =
    exists a, exists b, inc a x & inc b y & intersection2 a b = z.
*)

Definition intersection_partition2 u v :=
  Zo (intersection_covering2 u v) (fun z => nonempty z).

Lemma intersection_is_partition2: forall u v x,
  partition u x -> partition v  x ->
  partition (intersection_partition2 u v) x.
Proof. ir. uf intersection_partition2. red. red in H; red in H0.
  ee. set_extens. nin (union_exists H5).
  ee. Ztac. rwi intersection_covering2_pr H8.
  nin H8; nin H8. ee. wr H. apply union_inc with x2. wri H11 H6.
  inter2tac. am. cp H5.
  wri H H5. wri H0 H6.  nin (union_exists H5).  nin (union_exists H6).
  ee. assert (inc x0 (intersection2 x1 x2)). app intersection2_inc.
  apply union_inc with (intersection2 x1 x2). am.
  Ztac. rw intersection_covering2_pr. exists x1. exists x2. eee. ex_tac.
  ir. Ztac. am. ir. Ztac. clear H6. Ztac. clear H5.
  rwi intersection_covering2_pr H6. rwi intersection_covering2_pr H7.
  nin H6. nin H5. nin H7. nin H6. ee. wr H12; wr H10.
  nin (H4 _ _ H5 H6). nin (H2 _ _ H7 H11). rw H13; rww H14. au.
  right. app disjoint_pr. ir. red in H14. elim (emptyset_pr (x:= u0)).
  wr H14. app intersection2_inc. inter2tac. inter2tac.
  right. app disjoint_pr. ir. red in H13. elim (emptyset_pr (x:= u0)).
  wr H13. app intersection2_inc. inter2tac. inter2tac.
Qed.
```

We show here that this defines the supremum of two elements.

```
Lemma intersection_is_inf2: forall u v x,
  partition u x -> partition v  x ->
  least_upper_bound (coarser x)(doubleton u v)(intersection_partition2 u v).
Proof. ir. app least_upper_bound_doubleton. app coarser_order.
  rw related_coarser. ee. am. app intersection_is_partition2.
  red. ir. ufi intersection_partition2 H1. Ztac.
  rwi intersection_covering2_pr H2. nin H2. nin H2. ee. ex_tac. wr H5.
  app intersection2sub_first.
  rw related_coarser. ee. am. app intersection_is_partition2.
  red. ir. ufi intersection_partition2 H1. Ztac.
  rwi intersection_covering2_pr H2. nin H2. nin H2. ee. ex_tac. wr H5.
  app intersection2sub_second.
  intro. do 3 rw related_coarser. ir. ee. app intersection_is_partition2.
  am. red. ir. red in H6; red in H4.
  cp (H6 _ H7). cp (H4 _ H7).  nin H8. nin H9. ee.
  assert (sub a (intersection2 x0 x1)). red. ir. app intersection2_inc.
  app H11. app H10.
  exists (intersection2 x0 x1). uf intersection_partition2. ee. Ztac.
  rw intersection_covering2_pr. exists x0; exists x1. eee.
  red in H5. ee. nin (H13 _ H7). cp (H12 _ H15). ex_tac. am.
Qed.
```

This is the generalization to a general family F_i of partitions of X. Assume $x_i \in F_i$, (this is equivalent to $(x_i)_i \in \prod F_i$). Then $\bigcap x_i \subset E$, and these sets form a partition of X when we remove the empty set from the list.

```
Definition intersection_partition f :=
  complement (fun_image (productb f) intersectionb) (singleton (emptyset)).

Lemma intersection_is_partition: forall f x,
  fgraph f -> (forall u, inc u (domain f) -> partition (V u f) x) ->
  nonempty (domain f) ->
  partition (intersection_partition f) x.
Proof. ir. uf intersection_partition. red. ee. set_extens.
  nin (union_exists H2). ee. srwi H4. ee. awi H4. nin H4. ee. wri H6 H3.
  rwi productb_pr H4. ee. wri H7 H1. nin H1. cp (H8 _ H1).
  cp (intersectionb_forall H3 H1). wri H7 H0. cp (H0 _ H1). red in H11. ee.
  wr H11. union_tac. am.
  set (g := fun u => choose (fun v =>  inc x0 v & inc v (V u f))).
  assert (forall u, inc u (domain f) -> (inc x0 (g u) & inc (g u) (V u f))).
  ir. uf g. app choose_pr. cp (H0 _ H3). red in H4. ee. wri H4 H2.
  ap (union_exists H2).
  set (h:= L (domain f) g). assert (inc x0 (intersectionb h)).  uf h.
  app intersectionb_inc. nin H1.  exists (J y (V y h)). uf h. app fdefined_lem.
  gprops. bw.  bw. ir. bw. nin (H3 _ H4). am.
  assert (inc h (productb f)). rw productb_pr. uf h. ee. gprops. bw. bw.
  ir. bw. nin (H3 _ H5). am. am. apply union_inc with (intersectionb h).
  am. srw. ee. aw. ex_tac. red. ir. rwi (singleton_eq H6) H4.
  elim (emptyset_pr H4). ir. srwi H2. ee. nin (emptyset_dichot a). elim H3.
  rw H4. fprops. am. ir. srwi H2. srwi H3. ee. awi H2. nin H2. awi H3. nin H3.
  ee. wr H7. wr H6. nin (equal_or_not x0 x1). rww H8. au. right.
  app disjoint_pr. ir. elim H8. rw (productb_extensionality H H2 H3). ir.
  rwi productb_pr H2. rwi productb_pr H3. ee.
  assert (inc u (V i x0)). app intersectionb_forall. ue.
  assert (inc u (V i x1)). app intersectionb_forall. ue.
  cp (H0 _ H11). red in H18. ee. rwi H14 H15. rwi H12 H13.
```

```
  nin (H20 _ _ (H15 _ H11) (H13 _ H11)). am. red in H21.
  elim (emptyset_pr (x:=u)). wr H21. app intersection2_inc. am. am.
Qed.
```

We show that the intersection is the least upper bound of the family.

```
Lemma intersection_is_inf: forall f x,
  fgraph f -> (forall u, inc u (domain f) -> partition (V u f) x) ->
  nonempty (domain f) ->
  least_upper_bound (coarser x) (range f) (intersection_partition f).
Proof. ir. rw least_upper_bound_pr. ee. red. rw substrate_coarser. split.
  rw set_of_partition_pr. app intersection_is_partition. ir.
  rwi frange_inc_rw H2. nin H2. ee. rw related_coarser. ee. rw H3. app H0.
  app intersection_is_partition. red. ir. ufi intersection_partition H4.
  srwi H4. ee. awi H4. nin H4. ee. rwi productb_pr H4. ee.
  exists (V x0 x1). ee. rw H3. app H8. ue. wr H6. red. ir.
  wri H7 H2. ap (intersectionb_forall H9 H2). am. am.
  ir. red in H2. ee. rwi substrate_coarser H2. rwi set_of_partition_pr H2.
  rw related_coarser. split.  app intersection_is_partition.  split. am.
  red. ir.
  set (g:= fun u => choose (fun z => inc z (V u f) & sub a z)).
  assert (forall u, inc u (domain f) -> (inc (g u) (V u f) & sub a (g u))).
  ir. assert (inc (V u f) (range f)). app inc_V_range. cp (H3 _ H6).
  rwi related_coarser H7. uf g. app choose_pr. ee. red in H9. app (H9 _ H4).
  assert (sub a (intersectionb (L (domain f) g))). red. ir.
  app intersectionb_inc. nin H1. exists (J y (V y (L (domain f) g))).
  app fdefined_lem. gprops. bw. bw. ir. bw. nin (H5 _ H7). app H9.
  exists (intersectionb (L (domain f) g)). split.
  uf intersection_partition. srw. ee. aw. exists (L (domain f) g). ee.
  rw productb_pr. ee. gprops. bw. bw. ir. bw.  nin (H5 _ H7). am. am. tv.
  red. ir. rwi (singleton_eq H7) H6. red in H2.  ee. nin (H8 _ H4).
  cp (H6 _ H10). elim (emptyset_pr H11). am. app coarser_order.
  rw substrate_coarser. red. ir. rw set_of_partition_pr.
  rwi frange_inc_rw H2. nin H2. ee. rw H3. app H0. am.
Qed.
```

We prove now the first claim. The previous result show that any non-empty family of partitions has a least upper bound. If the family is empty, the result is true as well, the coarsest partition being the one that has a single element E. Note that, if E is empty, there is only one partition, the empty set.

```
Lemma exercise1_18a: forall E,  complete_lattice (coarser E).
Proof. ir. cp (coarser_order E).
  app exercise1_11b. ir. nin (emptyset_dichot X). rw H1.
  nin (emptyset_dichot E).  red. exists emptyset.
  rw least_upper_bound_emptyset. red.
  assert (partition emptyset E). red. ee. rw H2.
  app is_emptyset. red. ir. nin (union_exists H3). ee. elim (emptyset_pr H5).
  ir.  elim (emptyset_pr H3). ir.  elim (emptyset_pr H3).
  rw substrate_coarser. ee. rw set_of_partition_pr. am.
  ir. rwi set_of_partition_pr H4. rw related_coarser. ee. am. am.
  red. ir. red in H4. ee. cp (H6 _ H5). nin H8. assert (inc y (union x)).
  union_tac. rwi H4 H9. rwi H2 H9. elim (emptyset_pr H9). am.
  red. exists (smallest_partition E). rw least_upper_bound_emptyset. red.
  rw substrate_coarser. ee. rw set_of_partition_pr.
  app partition_smallest. ir. rwi set_of_partition_pr H3.
```

```
  rw related_coarser. ee. app partition_smallest.  am.
  red. ir. exists E. ee. uf smallest_partition. fprops. red in H3. ee.
  wr H3. app union_sub. am.
  set (f := diagonal X). assert (fgraph f). uf f. app fgraph_diagonal.
  assert (range f = X). uf f. app range_diagonal.
  assert (domain f = X). uf f. app domain_diagonal. wri H4 H1.
  assert (forall u, inc u (domain f) -> partition (V u f) E). rw H4.
  ir. uf f. rww V_diagonal. cp (H0 _ H5). rwi substrate_coarser H6.
  rwi set_of_partition_pr H6.  am.
  cp (intersection_is_inf H2 H5 H1). rwi H3 H6.
  exists (intersection_partition f). am.
Qed.
```

Let's now show that $\mathscr{P}$ is not distributive. It is clear that we can use the coarser or finer order indifferently. If E is empty, or is a singleton, there is a unique partitions. If E has two elements, there are two partitions, the least and the greatest ones. In these cases, the set is distributive. Assume now that the set has three elements x, y and z. Let P_x be the partition with the singleton $\{x\}$ and the doubleton with the remaining elements, likewise for P_y and P_z. Let α be the partition formed of the three singletons and ω the partition containing only $\{x, y, z\}$. If E has more than three elements, we add the complement of $\{x, y, z\}$ to these sets. We have $\alpha \leq \mathrm{P}_x \leq \omega$ (for the finer ordering). The infimum (resp. supremum) of two distinct elements among P_x, P_y and P_z is α (resp. ω). This contradicts distributivity.

We first show that we can convert a partition of $\{x, y, z\}$ into a partition of E.

```
Definition big_set3 E :=
 (exists x, exists y, exists z, inc x E & inc y E & inc z E & x <> y & x <> z
   & y <> z).

Lemma exercise1_18b: forall E, big_set3 E ->
  ~ (distributive_lattice2 (coarser E)).
Proof. ir. destruct H as [ x [y [z  H]]]. ee.
  set (o:= tack_on (doubleton x y) z).
  assert (forall i, inc i o = (i = x \/ i = y \/ i = z)).
  ir. ap iff_eq. uf o. ir. nin (tack_on_or H5). nin (doubleton_or H6); au. au.
  uf o. ir. nin H5. rw H5. app inc_tack_on_y. fprops.
  nin H5. rw H5. app inc_tack_on_y. fprops. rw H5. fprops.
  set (F:=complement E o).
  set (with_F := fun u => complement (tack_on u F) (singleton emptyset)).
  assert (forall u, ~ (inc emptyset u) -> F = emptyset -> with_F u = u).
  ir. uf with_F. set_extens.  srwi H8. ee.  nin (tack_on_or H8). am.
  elim H9.  ue. ue. srw. ee. fprops. dneg. wrr (singleton_eq H9).
  assert (forall u, ~ (inc emptyset u) -> F <> emptyset ->
    with_F u = tack_on u F).
  ir. uf with_F. set_extens. srwi H9. ee; am. srw. ee. am. red. ir.
  rwi (singleton_eq H10) H9. nin (tack_on_or H9). contradiction.
  symmetry in H11. contradiction.
 assert (sub o E). red. ir. rwi H5 H8. nin H8. ue. nin H8; ue.
  assert (forall u, partition u o -> partition (with_F u) E).
  ir. red. ee. set_extens. nin (union_exists H10). ee. ufi with_F H12. srwi H12.
  ee. nin (tack_on_or H12). red in H9. ee. app H8. wr H9. union_tac.
  rwi H14 H11. ufi F H11. srwi H11. ee. am. nin H9. ee.
  assert (~ inc emptyset u). red. ir. cp (H11 _ H13). nin H14.
  elim (emptyset_pr H14). nin (equal_or_not F emptyset). rw H6. rw H9.
  nin (inc_or_not x0 o). am. elim (emptyset_pr (x:=x0)). wr H14. uf F. srw.
```

```
eee. am. am. rww H7. nin (inc_or_not x0 o). wri H9 H15.
nin (union_exists H15). ee. apply union_inc with x1. am. fprops.
apply union_inc with F.  uf F. srw. eee. fprops. uf with_F. ir.
srwi H10. ee. nin (emptyset_dichot a). elim H11. rw H12. fprops. am.
uf with_F. ir. srwi H10. srwi H11. ee. nin (equal_or_not a b). au. right.
red in H9. ee. nin (tack_on_or H10). nin (tack_on_or H11).
nin (H16 _ _ H17 H18). contradiction. am. app disjoint_pr. ir.
assert (inc u0 o). wr H9. union_tac. assert (inc u0 F). ue. ufi F H22.
srwi H22. ee. contradiction.  nin (tack_on_or H11).  app disjoint_pr. ir.
assert (inc u0 o). wr H9. union_tac. assert (inc u0 F). ue. ufi F H22.
srwi H22. ee. contradiction. elim H14. ue.
```

We now define the five partitions.

```
assert (forall a b c, union2  (singleton a) (doubleton b c) = o ->
  a <> b -> a <> c -> b <> c ->
  partition (doubleton (singleton a) (doubleton b c)) o).
ir. red. ee. am. ir. nin (doubleton_or H14). rw H15. fprops.
rw H15. app nonempty_doubleton. ir. nin (doubleton_or H14); rw H16.
nin (doubleton_or H15); rw H17; au. right.  app disjoint_pr. ir.
rwi (singleton_eq H18) H19. nin (doubleton_or H19). contradiction.
contradiction.
nin (doubleton_or H15); rw H17; au. right.  app disjoint_pr. ir.
rwi (singleton_eq H19) H18. nin (doubleton_or H18). contradiction.
contradiction.
assert (Ha:partition (doubleton (singleton x) (doubleton y z)) o).
app H10. set_extens. rw H5.
nin (union2_or H11). rw (singleton_eq H12). au.
nin (doubleton_or H12); rw H13; au. rwi H5 H11. nin H11. rw H11.
app union2_first. fprops. nin H11; rw H11;  app union2_second; fprops.
assert (Hb: partition (doubleton (singleton y) (doubleton x z)) o).
app H10. set_extens. rw H5.
nin (union2_or H11). rw (singleton_eq H12). au.
nin (doubleton_or H12); rw H13; au. rwi H5 H11. nin H11. rw H11.
app union2_second. fprops. nin H11; rw H11.  app union2_first; fprops.
app union2_second; fprops. intuition.
assert (Hc:partition (doubleton (singleton z) (doubleton y x)) o).
app H10. set_extens. rw H5.
nin (union2_or H11). rw (singleton_eq H12). au.
nin (doubleton_or H12); rw H13; au. rwi H5 H11. nin H11. rw H11.
app union2_second; fprops. nin H11; rw H11. app union2_second; fprops.
app union2_first; fprops. intuition. intuition. intuition.
set (Px:= with_F (doubleton (singleton x) (doubleton y z))).
assert (partition Px E). uf Px. app H9.
set (Py:= with_F (doubleton (singleton y) (doubleton x z))).
assert (partition Py E). uf Py. app H9.
set (Pz:= with_F (doubleton (singleton z) (doubleton y x))).
assert (partition Pz E). uf Pz. app H9.
set (alpha:= with_F(largest_partition o)).
assert (partition alpha E). uf alpha. app H9. app partition_largest.
set (omega:= with_F(smallest_partition o)).
assert (partition omega E). uf omega. app H9. app partition_smallest.
uf o. exists z. fprops.
```

Let's show that (D′) is absurd for P_x, P_y and P_z. We consider

$$\sup(P_x, \inf(P_y y, P_z)) = \inf(\sup(P_x, P_y), \sup(P_x, P_z))$$

We first pretend that we can replace the sup by *intersection_partition2*.

```
red. ir. red in H16. rwi substrate_coarser H16.
assert (inc Px (set_of_partition_set E)). rw set_of_partition_pr. am.
assert (inc Py (set_of_partition_set E)). rw set_of_partition_pr. am.
assert (inc Pz (set_of_partition_set E)). rw set_of_partition_pr. am.
cp (H16 _ _ _ H17 H18 H19). clear H16; clear H17; clear H18; clear H19.
assert (forall u v, partition u E -> partition v E ->
  sup (coarser E) u v = (intersection_partition2 u v)). ir.
cp (intersection_is_inf2 H16 H17). set (w:= intersection_partition2 u v) in *.
assert (has_supremum  (coarser E) (doubleton u v)).  exists w. am.
uf sup. assert (sub (doubleton u v) (substrate (coarser E))).
rw substrate_coarser. red. ir. rw set_of_partition_pr.
nin (doubleton_or H21); rww H22. cp (supremum_pr1 (coarser_order E) H21 H19).
app (supremum_unique (coarser_order E) H22 H18).
```

We show $\sup(P_x, P_y) = \alpha$.

```
assert (sup (coarser E) Px Py = alpha). rww H16. uf intersection_partition2.
uf alpha. set_extens. Ztac. rwi intersection_covering2_pr H18. nin H18.
nin H18. ufi Px H18; ufi Py H18. ee. ufi with_F H18. srwi H18.
ufi with_F H21. srwi H21. ee. uf with_F. srw. ee. nin (tack_on_or H18).
assert (inc x0  (largest_partition o)). rw largest_partition_pr.
wri H22 H19. nin H19. nin (intersection2_both H19).
nin (doubleton_or H25). exists x. ee. rw H5; au. wr H28. wr H22.
set_extens. app intersection2_inc. rwi H28  H26. rwi H28  H29. awi H26.
awi H29. rw H29. ue. inter2tac. nin (tack_on_or H21).
nin (doubleton_or H29). exists y.  ee. rw H5; au. wr H30. wr H22.
set_extens. app intersection2_inc. rwi H30  H31. rwi H30  H27. awi H27.
awi H31. rw H31. ue. inter2tac. exists z. ee. rw H5; au. wr H22.
rw H28; rw H30.  set_extens. rw (singleton_eq H31). app intersection2_inc.
fprops. fprops. nin (intersection2_both H31). nin (doubleton_or H32).
nin (doubleton_or H33). rwi H35 H34. contradiction.  rw H35. fprops.
rw H34. fprops. rwi H28 H26. rwi H29 H27. ufi F H27. srwi H27. ee.
elim H30. rw H5. nin (doubleton_or H26); au. fprops.
nin (tack_on_or H21). wri H22 H19. nin H19. rwi H25 H19.
nin (intersection2_both H19). ufi F H27. srwi H27. ee. elim H29. rw H5.
nin (doubleton_or H26).  rwi H30 H28. awi H28.  rw H28. au.
rwi H30 H28. nin (doubleton_or H28); rw H31; au. rwi H25 H22. rwi H26 H22.
rwi intersection2idem H22. wr H22. fprops. red. ir. awi H25. rwi H25 H19.
nin H19. elim (emptyset_pr H19). Ztac. rw intersection_covering2_pr.
ufi with_F H17. srwi H17. ee. nin (tack_on_or H17).
rwi largest_partition_pr H19. nin H19. nin H19. rwi H5 H19. nin H19.
exists x0. exists (doubleton x z). ee. uf Px. uf with_F. srw. ee. wr H19.
rw H21. app inc_tack_on_y. fprops. am. uf Py. uf with_F. srw. ee.
app inc_tack_on_y. fprops. red. ir. awi H22. elim (emptyset_pr (x:=x)).
wr H22. fprops. wr H21. rw H19. set_extens. inter2tac. app intersection2_inc.
awi H22. rw H22. fprops. nin H19. exists (doubleton y z). exists x0.
ee. uf Px. uf with_F. srw. ee. app inc_tack_on_y. fprops. red. ir. awi H22.
elim (emptyset_pr (x:=y)). wr H22. fprops. uf Py. wr H21. wr H19.
uf with_F. srw. ee.  app inc_tack_on_y. fprops. rww H21. wr H21. rw H19.
set_extens. inter2tac. app intersection2_inc. awi H22. rw H22. fprops.
rwi H19 H21. exists (doubleton y z). exists (doubleton x z). ee.
uf Px. uf with_F. srw. ee. app inc_tack_on_y. fprops.  red. ir. awi H22.
elim (emptyset_pr (x:=y)). wr H22. fprops.
uf Py. uf with_F. srw. ee. app inc_tack_on_y. fprops.  red. ir. awi H22.
```

```
elim (emptyset_pr (x:=x)). wr H22. fprops. wr H21. set_extens.
nin (intersection2_both H22). nin (doubleton_or H23). nin (doubleton_or H24).
rwi H26 H25. contradiction. rw H26. fprops. rw H25. fprops. awi H22. rw H22.
app intersection2_inc. fprops. fprops.  exists F. exists F. ee.
uf Px. uf with_F. srw. ee. fprops.  ue. uf Py.  uf with_F. srw. ee. fprops.
ue. rw H19. rw intersection2idem. tv. ufi with_F H17. srwi H17. ee.
nin (emptyset_dichot x0). rwi H19 H18. elim H18. fprops. am.
rwi H17 H20. clear H17.
```

We show $\sup(P_x, P_z) = \alpha$.

```
assert (sup (coarser E) Px Pz = alpha). rww H16. uf intersection_partition2.
uf alpha. set_extens. Ztac. rwi intersection_covering2_pr H18. nin H18.
nin H18. ufi Px H18; ufi Pz H18. ee. ufi with_F H18. srwi H18.
ufi with_F H21. srwi H21. ee. uf with_F. srw. ee. nin (tack_on_or H18).
assert (inc x0  (largest_partition o)). rw largest_partition_pr.
wri H22 H19. nin H19. nin (intersection2_both H19).
nin (doubleton_or H25). exists x. ee. rw H5; au. wr H28. wr H22.
set_extens. app intersection2_inc. rwi H28  H26. rwi H28  H29. awi H26.
awi H29. rw H29. ue. inter2tac. nin (tack_on_or H21).
nin (doubleton_or H29). exists z.  ee. rw H5; au. wr H30. wr H22.
set_extens. app intersection2_inc. rwi H30  H31. rwi H30  H27. awi H27.
awi H31. rw H31. ue. inter2tac. exists y. ee. rw H5; au. wr H22.
rw H28; rw H30.  set_extens. rw (singleton_eq H31). app intersection2_inc.
fprops. fprops. nin (intersection2_both H31). nin (doubleton_or H32).
rw H34. fprops. nin (doubleton_or H33). rw H35. fprops. rwi H35 H34.
contradiction.  rwi H28 H26. rwi H29 H27. ufi F H27. srwi H27. ee.
elim H30. rw H5. nin (doubleton_or H26); au. fprops.
nin (tack_on_or H21). wri H22 H19. nin H19. rwi H25 H19.
nin (intersection2_both H19). ufi F H27. srwi H27. ee. elim H29. rw H5.
nin (doubleton_or H26).  rwi H30 H28. awi H28.  rw H28. au.
rwi H30 H28. nin (doubleton_or H28); rw H31; au. rwi H25 H22. rwi H26 H22.
rwi intersection2idem H22. wr H22. fprops. red. ir. awi H25. rwi H25 H19.
nin H19. elim (emptyset_pr H19). Ztac. rw intersection_covering2_pr.
ufi with_F H17. srwi H17. ee. nin (tack_on_or H17).
rwi largest_partition_pr H19. nin H19. nin H19. rwi H5 H19. nin H19.
exists x0. exists (doubleton y x). ee. uf Px. uf with_F. srw. ee. wr H19.
rw H21. app inc_tack_on_y. fprops. am. uf Pz. uf with_F. srw. ee.
app inc_tack_on_y. fprops. red. ir. awi H22. elim (emptyset_pr (x:=x)).
wr H22. fprops. wr H21. rw H19. set_extens. inter2tac. app intersection2_inc.
awi H22. rw H22. fprops. nin H19. exists (doubleton y z).
exists (doubleton y  x). ee. uf Px. uf with_F. srw. ee. app inc_tack_on_y.
fprops. red. ir. awi H22. elim (emptyset_pr (x:=y)). wr H22. fprops.
uf Pz. uf with_F. srw. ee.  app inc_tack_on_y. fprops. red. ir. awi H22.
elim (emptyset_pr (x:=y)). wr H22. fprops. wr H21. wr H19. set_extens.
nin (intersection2_both H22). nin (doubleton_or H23). rw H25. fprops.
nin (doubleton_or H24). rw H26. fprops. rwi H26 H25.  contradiction.
app intersection2_inc. awi H22. rw H22. fprops. awi H22.  rw H22. fprops.
exists (doubleton y z). exists x0. ee. uf Px. uf with_F. srw. ee.
app inc_tack_on_y. fprops. red. ir. awi H22. elim (emptyset_pr (x:=y)).
wr H22. fprops.  uf Pz. uf with_F. srw. ee. wr H21. rw H19.
app inc_tack_on_y. fprops.  am. wr H21. set_extens. inter2tac.
app intersection2_inc. awi H22. rw H22. rw H19. fprops.
exists F. exists F. ee. uf Px. uf with_F. srw. ee. fprops.  ue. uf Pz.
uf with_F. srw. ee. fprops.
ue. rw H19. rw intersection2idem. tv. ufi with_F H17. srwi H17. ee.
nin (emptyset_dichot x0). rwi H19 H18. elim H18. fprops. am.
```

```
rwi H17 H20. clear H17.
```

We simplify our condition. It becomes:

$$\sup(P_x, \inf(P_y, P_z) = \alpha$$

```
assert(inf (coarser E) alpha alpha = alpha). app inf_comparable1.
app coarser_order. wrr order_reflexivity. rw substrate_coarser.
rww set_of_partition_pr. app coarser_order. rwi H17 H20. clear H17.
```

We pretend $\omega \leq \varpi$ whenever ϖ is a partition of E where one component is F.

```
assert (forall u, partition u o -> gle (coarser E) omega (with_F u)).
ir. rw related_coarser. ee. am. app H9. red. ir. ufi with_F H18. srwi H18.
ee. nin (tack_on_or H18). exists o. uf omega. ee. uf with_F. srw.
uf smallest_partition. ee. fprops. red. ir. awi H22.
elim (emptyset_pr (x:=z)). wr H22. uf o. fprops. red in H17. ee. wr H17.
app union_sub. exists F. ee. uf omega. uf with_F. srw. ee. fprops. ue. ue.
```

We pretend $\omega = \inf(P_y, P_Z)$. All we have to do is check that if $a \leq P_y$ and $a \leq P_z$, then $a \leq \omega$. Note that such a partition contains two sets b and b' such that $\{x, y\} \subset b$ and $\{x, z\} \subset b'$. These sets are equal, since they cannot be disjoint.

```
set (w:= inf (coarser E) Py Pz) in *.
assert (w = omega).
assert (order (coarser E)). app coarser_order.
assert (inc Py (substrate (coarser E))). rw substrate_coarser.
rw set_of_partition_pr. am.
assert (inc Pz (substrate (coarser E))). rw substrate_coarser.
rw set_of_partition_pr. am.
assert (has_infimum (coarser E) (doubleton Py Pz)).
nin (exercise1_18a E). assert (sub (doubleton Py Pz) (substrate (coarser E))).
red. ir. nin (doubleton_or H24); ue. nin (H23 _ H24). am.
cp (inf_pr H18 H19 H21 H22). fold w in H23. ee.
assert (gle (coarser E) omega w). app H25. uf Py. app H17. uf Pz. app H17.
assert (gle (coarser E) w omega). rw related_coarser.
rwi related_coarser H23. ee. am. am. rwi related_coarser H24. ee.
red in  H28; red in H30; red. ir. ufi omega H31. ufi smallest_partition H31.
ufi with_F H31. srwi H31. ee. nin (tack_on_or H31). awi H33.
assert (inc (doubleton x z) Py). uf Py. uf with_F. srw. ee.
app inc_tack_on_y. fprops. red. ir. awi H34. elim (emptyset_pr (x:=x)).
wr H34. fprops. assert (inc (doubleton y x) Pz). uf Pz. uf with_F. srw. ee.
app inc_tack_on_y. fprops. red. ir. awi H35. elim (emptyset_pr (x:=x)).
wr H35. fprops.  nin (H28 _ H34). nin (H30 _ H35). ee.
red in H24. ee. nin (H41 _ _ H36 H37). exists x0. ee. am. rw H33.
red. ir. rwi H5 H43. nin H43. rw H43. app H39. fprops. nin H43.  rw H43.
rw H42. app H38. fprops. rw H43. app H39. fprops.
elim (emptyset_pr (x:=x)). red in H42. wr H42. app intersection2_inc.
app H39. fprops. app H38. fprops.  assert (inc F Py). uf Py. uf with_F.
srw. ee. fprops. ue. cp (H28 _ H34). rw H33. am. order_tac. rwi H18 H20.
```

The distributivity formula reads now: $\sup(P_x, \omega) = \alpha$. Since $\omega \leq P_x$, this simplifies to $P_x = \alpha$, absurd.

```
assert (sup  (coarser E) omega Px = Px). app sup_comparable1.
```

```
  app coarser_order. uf Px. app H17.
  assert (sup (coarser E) Px omega = Px). uf sup. rw doubleton_symm. am.
  rwi H21 H20. assert (inc (doubleton y z) alpha). wr H20. uf Px.
  uf with_F. srw. ee. app inc_tack_on_y. fprops. red. ir.
  awi H22. elim (emptyset_pr (x:=y)). wr H22. fprops. ufi alpha H22.
  ufi with_F H22. srwi H22. ee. nin (tack_on_or H22).
  rwi largest_partition_pr H24. nin H24. ee. elim H4.
  assert (inc y (singleton x0)). rw H25. fprops. awi H26. rw H26.
  assert (inc z (singleton x0)). rw H25. fprops. awi H27. rw H27. tv.
  assert (inc y F). wr H24. fprops. ufi F H25. srwi H25. ee. elim H26.
  rw H5. au.
Qed.
```

Let's characterize the finest partition (it is α such that for all x, x is coarser than α, or α is finer than x).

```
Lemma exercise1_18c: forall E,
  largest_partition E =  the_greatest_element (coarser E).
Proof. ir. sy. app the_greatest_element_pr2. app coarser_order. red.
  rw substrate_coarser. ee. rw set_of_partition_pr. app partition_largest.
  ir. rwi set_of_partition_pr H. rw related_coarser. ee. am.
  app partition_largest. red. ir. rwi largest_partition_pr H0. nin H0. ee.
  red in H. ee. wri H H0. nin (union_exists H0). ee. ex_tac. wr H1. red. ir.
  awi H6. ue.
Qed.
```

Let's show that the set is relatively complemented for the "finer" ordering. If ≤ denotes "coarser" we have to show: for all x and y such that $y \leq x$, there exists x' such that $\inf(x, x') = y$ and $\sup(x, x') = \alpha$.

For $a \in x$, we select an element $e_a \in a$; this is possible since $a \neq \emptyset$. We consider the set of all singletons $\{b\}$ that are not of the form e_a. We consider also the sets P_v formed of all e_a for $a \subset v$. All these sets form a partition of E. There is no need to well-order the sets (but this required the axiom of choice).

```
Lemma exercise1_18d: forall E x y,
  let r := coarser E in
    gle r y x -> exists x',
      inc x' (substrate r) & inf r x x' = y & sup r x x' = largest_partition E.
Proof. ir. ufi r H. rwi related_coarser H. ee. red in H1.
  set (X1 := Zo (powerset E) (fun z=> exists a, z = singleton a &
    ~(exists u, inc u x & a = rep u))).
  set (X2 := fun_image y (fun v => Zo E
    (fun z => exists u, inc u x & sub u v & z = rep u))).
  set (z:=union2 X1 X2).
  assert (partition z E). uf z. red. ee. set_extens. nin (union_exists H2).
  ee. nin (union2_or H4). ufi X1 H5. Ztac. nin H7. ee. rwi H7 H3. awi H3.
  rwi powerset_inc_rw H6. app H6. rw H7. ue. ufi X2 H5. awi H5. nin H5. ee.
  assert (sub x1 E). wr H6. app Z_sub. app H7. assert (Ha: inc x0 E). am.
  red in H0. ee. wri H0 H2. nin (union_exists H2). ee.
  nin (equal_or_not x0 (rep x1)). cp (H1 _ H6). nin H8. ee.
  set (w:= Zo E (fun z => exists u , inc u x & sub u x2 & z = rep u)).
  assert (inc x0 w). uf w. Ztac. exists x1. au.
  apply union_inc with w. am. app union2_second. uf X2. aw. exists x2. ee. am.
  tv. apply union_inc with (singleton x0). fprops. app union2_first. uf X1.
  Ztac. app powerset_inc. red. ir. awi H8. rw H8. am. exists x0. ee. tv.
```

```
red. ir. nin H8. ee. assert (inc x0 x2). rw H9. ap nonempty_rep. app H3.
nin (H4 _ _ H6 H8). elim H7. ue. red in H11. elim (emptyset_pr (x:= x0)).
wr H11. app intersection2_inc. ir. nin (union_exists H2). ee.
nin (doubleton_or H4). rwi H5 H3. ufi X1 H3. Ztac. nin H7. ee. rw H7. fprops.
rwi H5 H3. ufi X2 H3. awi H3. nin H3. ee. wr H6.
red in H; ee. nin (H7 _ H3). assert (inc y0 E). wr H. union_tac.
red in H0. ee.  wri H0 H10. nin (union_exists H10). ee. nin (H1 _ H14).
ee. nin (H8 _ _ H3 H15). rw H17. cp (H11 _  H14).
exists (rep x2). Ztac.  wr H0. apply union_inc with x2. app nonempty_rep. am.
exists x2. au. elim (emptyset_pr (x:=y0)). wr H17. app intersection2_inc.
app H16. ir. nin (union2_or H2); nin (union2_or H3). ufi X1 H4; ufi X1 H5.
Ztac; clear H5; Ztac. nin H7; nin H8. ee. rw H7; rw H8.
nin (equal_or_not x1 x0). rw H11. au. right. app disjoint_pr. ir. app H11.
awi H12; awi H13. ue. ufi X1 H4; Ztac. nin H7. ufi X2 H5. awi H5. nin H5. ee.
right. wr H9. rw H7. app disjoint_pr. ir. Ztac. nin H13. awi H10. elim H8.
wr H10. exists x2. eee. right. ufi X1 H5; Ztac. nin H7. ufi X2 H4. awi H4.
nin H4. ee. app disjoint_pr. ir. elim H8. rwi H7 H11. awi H11. wr H11.
wri H9 H10. Ztac. nin H13. exists x2. eee.
ufi X2 H4; ufi X2 H5. awi H4; awi H5; nin H4; nin H5. ee.
red in H. ee. nin (H9 _ _ H4 H5). left.  wr H7; wr H6; rww H10.
right. app disjoint_pr. ir. wri H7 H11. Ztac. wri H6 H12. Ztac. nin H14.
nin H16. ee. red in H0. ee.
elim (emptyset_pr (x:=u)). wr H10. app intersection2_inc.
app H19. rw H20. app nonempty_rep. app H21. app H17. rw H18.
app nonempty_rep. app H21. uf r. rw substrate_coarser. exists z. ee.
rww set_of_partition_pr.
```

We show $\inf(x, x') = y$.

```
assert (order (coarser E)). app coarser_order.
assert (inc x (substrate (coarser E))). rw substrate_coarser.
rw set_of_partition_pr. am.
assert (inc z (substrate (coarser E))). rw substrate_coarser.
rw set_of_partition_pr. am.
assert (has_infimum (coarser E) (doubleton x z)).
nin (exercise1_18a E). assert (sub (doubleton x z) (substrate (coarser E))).
red. ir. nin (doubleton_or H8); ue. nin (H7 _ H8). am.
cp (inf_pr H3 H4 H5 H6). set (w:= (inf (coarser E) x z)) in *. ee.
assert (gle (coarser E) y w). app H9. rw related_coarser. au.
rw related_coarser. eee. red. ir. ufi z H10. nin (union2_or H10).
ufi X1 H11. Ztac. nin H13. ee. rw H13. assert (inc x0 E).
rwi powerset_inc_rw H12. app H12. rw H13. fprops. red in H. ee. wri H H15.
nin (union_exists H15). ee. ex_tac. red. ir. awi H20. rww H20.
ufi X2 H11. awi H11. nin H11. ee. exists x0. ee. am. wr H12. red. ir. Ztac.
nin H15. ee. rw H17. app H16. app nonempty_rep. red in H0. ee. app H18.
assert (gle (coarser E) w y). rw related_coarser.
rwi related_coarser H7. ee. am. am. rwi related_coarser H8. ee.
red in H12. red in H14. red. ir.
set (z1:=  Zo E (fun z  => exists u, inc u x & sub u a & z = rep u)).
assert (inc z1 z). uf z. app union2_second. uf X2. aw. exists a. ee. am. tv.
nin (H14 _ H16). exists x0. ee. am. red. ir.
assert (inc x1 E). red in H. ee. wr H. union_tac. red in H0. ee. wri H0 H20.
nin (union_exists H20). ee. nin (equal_or_not x1 (rep x2)).
app H18. uf z1. Ztac. wrr H0. exists x2. ee. am. nin (H1 _ H24). ee.
red in H. ee. nin (H29 _ _ H15 H26). rww H30. elim (emptyset_pr (x:=x1)).
wr H30. app intersection2_inc. app H27. am.
```

```
  nin (H12 _ H24). ee. red in H7. ee. nin (H29 _ _ H17 H26). rw H30. app H27.
  elim (emptyset_pr (x:= rep x2)). wr H30. app intersection2_inc. app H18.
  uf z1. Ztac. wr H0. assert (inc (rep x2) x2). app nonempty_rep. app H21.
  union_tac. exists x2. ee. am. nin (H1 _ H24). ee. red in H. ee.
  nin (H34 _ _ H15 H31). rww H35. elim (emptyset_pr (x:=x1)). wr H35.
  app intersection2_inc. app H32. tv. app H27. app nonempty_rep. app H21.
  order_tac.
```

We show $\sup(x, x') = \alpha$.

```
  assert (sup (coarser E) x z  = (intersection_partition2 x z)). ir.
  cp (intersection_is_inf2 H0 H2).
  assert (has_supremum  (coarser E) (doubleton x z)).
  exists  (intersection_partition2 x z). am.
  uf sup. assert (sub (doubleton x z) (substrate (coarser E))).
  rw substrate_coarser. red. ir. rw set_of_partition_pr.
  nin (doubleton_or H5); rww H6. cp (supremum_pr1 (coarser_order E) H5 H4).
  app (supremum_unique (coarser_order E) H6 H3). rw H3.
  set_extens. rw largest_partition_pr. ufi intersection_partition2 H4.
  Ztac. clear H4. rwi intersection_covering2_pr H5. nin H5. nin H4. ee.
  ufi z H5. nin (union2_or H5). ufi X1 H8. Ztac. nin H10. ee.
  assert (inc x3 E). rwi powerset_inc_rw H9. ap H9. ue. exists x3. ee. am.
  wr H7. wr H10. set_extens. app intersection2_inc. rwi H10 H13. awi H13.
  rw H13. nin H6. wri H7 H6. nin (intersection2_both H6).  rwi H10 H15.
  awi H15. ue. inter2tac. ufi X2 H8. awi H8. nin H8. ee.
  nin (H1 _ H4). ee. nin H6. wri H7 H6. nin (intersection2_both H6).
  exists y0. ee. red in H0. ee. wr H0. union_tac. set_extens. awi H14.
  rw H14. wr H7. app intersection2_inc. aw. wri H9 H13. Ztac. nin H16.
  wri H7 H14. nin (intersection2_both H14). wri H9 H18. Ztac. nin H20.
  ee. red in H0. ee. nin (H26 _ _ H16 H20). rw H22; rw H24; rww H27.
  assert (inc y0 x6). rw H24. app nonempty_rep. app H25.
  nin (H26 _ _ H4 H16).
  assert (inc x5 x7). rw H22. app nonempty_rep. app H25.
  nin (H26 _ _ H4 H20).
  elim (emptyset_pr (x:=x5)). wr H27. app intersection2_inc. ue.
  elim (emptyset_pr (x:=x5)). wr H31. app intersection2_inc.
  elim (emptyset_pr (x:=y0)). wr H29. app intersection2_inc.
  rwi largest_partition_pr H4.  nin H4. ee.
  red in H0. ee. cp H4.  wri H0 H8. nin (union_exists H8). ee.
  nin (H1 _ H10). ee. uf intersection_partition2. Ztac.
  rw intersection_covering2_pr. exists x2.
  nin (equal_or_not x1 (rep x2)).
  set (w:=  Zo E (fun z  => exists u, inc u x & sub u x3 & z = rep u)).
  exists w. ee. am. uf z. app union2_second. uf X2. aw. exists x3. eee. wr H5.
  set_extens. aw. nin (intersection2_both H14). ufi w H16. Ztac. nin H18.
  ee. rw H20. rw H13. nin (H7 _ _ H10 H18).  ue. elim (emptyset_pr (x:=x4)).
  wr H21. app intersection2_inc. rw H20. app nonempty_rep. app H6.
  awi H14. rw H14. app intersection2_inc. uf w. Ztac. exists x2. au.
  exists (singleton x1). ee. am. uf z. app union2_first. uf X1. Ztac.
  app powerset_inc. red. ir. awi H14. rww H14.  exists x1. ee. tv.
  red. ir. nin H14. ee. assert (inc x1 x4). rw H15. app nonempty_rep. app H6.
  nin (H7 _ _ H10 H14). elim H13. ue. elim (emptyset_pr (x:= x1)).
  wr H17. app intersection2_inc. wr H5. set_extens. inter2tac.
  app intersection2_inc. awi H14. ue. wr H5. fprops.
Qed.
```

---

19. *An ordered set* E *is said to be* without gaps *if it contains two distinct comparable elements and if, for each pair of elements x, y such that $x < y$, the open interval $]x, y[$ is not empty. Show that the ordinal sum $\sum_{\iota\in I} E_\iota$ (Exercise 3) is without gaps if and only if the following conditions are satisfied:*

(I) Either I *contains two distinct comparable elements, or else there exists $\iota \in I$ such that E_ι contains two distinct comparable elements.*

(II) Each E_ι which contains at least two distinct comparable elements is without gaps.

(III) If α, β are two elements of I *such that $\alpha < \beta$ and if the interval $]\alpha, \beta[$ in* I *is empty, then either E_α has no maximal element or else E_β has no minimal element.*

In particular, every ordinal sum $\sum_{\iota\in I} E_\iota$ of sets without gaps is itself without gaps, provided that no E_ι has a maximal element (or provided that no E_ι has a minimal element). If I *is without gaps, and if each E_ι is either without gaps or contains no two distinct comparable elements, then $\sum_{\iota\in I} E_\iota$ is without gaps.*

Let's start with the definitions and the main property.

```
Definition without_gaps r :=
  order r & (exists x, exists y, glt r x y) &
  (forall x y, glt r x y -> exists z, glt r x z & glt r z y).

Lemma Exercise1_19a: forall r f g,
  ordinal_sum_axioms1 r f g ->
  (without_gaps (ordinal_sum r f g) =
    (((exists i, exists j, glt r i j) \/
      (exists i, exists x, exists  y, inc i (substrate r) & glt (V i g) x y))
    & (forall i x y, inc i (substrate r) ->  glt (V i g) x y ->
      without_gaps (V i g))
    & (forall i j, glt r i j ->
      (exists k, glt r i k & glt r k j)
      \/ (forall u, ~ (maximal_element (V i g) u))
      \/ (forall u, ~ (minimal_element (V j g) u))))).
```

Condition (I) is says that the ordinal sum has two distinct comparable elements.

```
Proof. ir.  nin H. rename H0 into Hw.  cp H. red in H. ee.
  ap iff_eq. ir. red in H7. ee. nin H8. nin H8. nin H8.
  awi H8. ee. nin H12. left. exists (Q x). exists (Q x0). am. right.
  exists (Q x). exists (P x). exists (P x0). ee.
  nin (du_index_pr H8). ue. split. am. dneg.
  nin (du_index_pr H8). nin (du_index_pr H11). ee. app pair_extensionality. am.
```

Condition II is obvious.

```
  ir. red. ee. app H5. ue. exists x; exists y; am. ir.
  assert (glt (ordinal_sum r f g) (J x0 i) (J y0 i)).  nin H12. split.
  aw. ee. app inc_disjoint_union. ue. wr H6. order_tac. ue.
  app inc_disjoint_union. ue. wr H6. order_tac. ue. right. ee. tv. am.
  dneg. inter2tac. nin (H9 _ _ H13). ee. nin H14. nin H15. awi H14. awi H15.
  ee. exists (P x1). nin H21. nin H19. order_tac. nin H19. rwi H19 H21.
  nin H21. elim H23. tv. nin H19. nin H21. rwi H21 H19. nin H19.
```

```
 elim H23. tv. nin H21. nin H19. ee. split. am. red. ir. elim H16.
 nin (du_index_pr H20). ee. app pair_extensionality. fprops. aw. aw.
 split. rww H21. red. ir. elim H17.
 nin (du_index_pr H20). ee. app pair_extensionality. fprops. aw. aw. am. am.
```

Consider now condition (III). Consider a maximal element x of E_α, a minimal element y of E_β. We have $x < y$ (considered as elements of the sum); hence there is z such that $x < y < z$. It index cannot be α not β , hence there is a γ such that $\alpha < \gamma < \beta$.

```
ir. nin (p_or_not_p (forall u : Set, ~ maximal_element (V i g) u)). au.
 assert (exists u, maximal_element (V i g) u). app exists_proof.
 nin (p_or_not_p (forall u : Set, ~ minimal_element (V j g) u)). au.
 assert (exists u, minimal_element (V j g) u). app exists_proof.
 clear H11. clear H13. nin H12. nin H14. nin H11. nin H12. ee.
 assert (inc i (domain f)). wr H1. order_tac.
 assert (inc j (domain f)). wr H1. order_tac.
 assert (inc (J x i) (disjoint_union f)). app inc_disjoint_union. wrr H6.
 assert (inc (J x0 j) (disjoint_union f)). app inc_disjoint_union. wrr H6.
 assert (glt (ordinal_sum r f g) (J x i) (J x0 j)). split. aw. ee. am. am.
 au. nin H10.  dneg. inter2tac. nin (H9 _ _ H19). ee. nin H20. nin H21.
 awi H20. awi H21. ee. nin H27. nin H25. left. exists (Q x1). au.
 nin H25. assert (inc (P x1) (substrate (V j g))). rwi H25 H28.
 cp (H5 _ H16). order_tac. rwi H25 H28. cp (H14 _  H28).
 elim H23. cp (du_index_pr H21). ee. app pair_extensionality. fprops. aw. aw.
 ee.  assert (inc (P x1) (substrate (V i g))). cp (H5 _ H15). order_tac.
 cp (H13 _ H28). elim H22. cp (du_index_pr H26). ee.
 app pair_extensionality. fprops. sy.  aw. aw. am. am.
```

Converse. We use part (I) to show that there are at least two comparable elements.

```
ir. ee. red. ee. fprops.  nin H7.  nin H7. nin H7.
 assert (inc x (domain f)). wr H1. order_tac.
 assert (inc x0 (domain f)). wr H1. order_tac.
 nin (Hw _ H10). nin (Hw _ H11). exists (J y x). exists (J y0 x0).
 split. aw. ee. app inc_disjoint_union.  app inc_disjoint_union.  au.
 nin H7. dneg. inter2tac. nin H7. nin H7. nin H7. ee.
 exists (J x0 x). exists (J x1 x). rwi H1 H7. split. aw. ee.
 app inc_disjoint_union. wr H6. order_tac. am. app inc_disjoint_union. wr H6.
 order_tac. am. right. nin H10. au.  nin H10. dneg. inter2tac.
```

We use condition (III). If $x < y$ but have different indices, the interval is non-empty.

```
ir. nin H10. awi H10. ee. nin H13. nin (H9 _ _ H13). nin H14.
 assert (inc x0 (domain f)). ee. wr H1. order_tac. nin (Hw _ H15).
 assert (inc (J y0 x0) (disjoint_union f)). ap inc_disjoint_union. am. am.
 exists (J y0 x0). ee. split. aw. eee. red. ir. nin H14.  elim H20. rw H19.
 aw.  split. aw. eee. red. ir. nin H18. elim H20. wr H19. aw.
 nin H14. cp (H14 (P x)).  ufi maximal_element H15.
 assert (exists z, inc z (substrate (V (Q x) g)) & glt (V (Q x) g) (P x) z).
 app exists_proof. red. ir. elim H15. ee. cp (du_index_pr H10). ee. rww H6.
 ir. nin (equal_or_not x0 (P x)). am.  elim (H16 x0). ee. order_tac.
 split. am. au.
 nin H16. assert (inc (J x0 (Q x)) (disjoint_union f)).
 app inc_disjoint_union. cp (du_index_pr H10). ee. am. ee. wrr H6. wr H1.
 order_tac. exists (J x0 (Q x)). ee; split. aw. ee. am. am. right. ee. tv.
```

```
  nin H18. am. nin H18. red. ir. elim H19. rw H20. aw. aw. ee. am. am. au.
  red. ir. nin H18. elim H11. wr H19. nin H13. elim H21. wr H19. aw.
  cp (H14 (P y)).  ufi minimal_element H15.
  assert (exists z, inc z (substrate (V (Q y) g)) & glt (V (Q y) g) z (P y)).
  app exists_proof. red. ir. elim H15. ee. cp (du_index_pr H12). ee. rww H6.
  ir. nin (equal_or_not x0 (P y)). am.  elim (H16 x0). ee. order_tac.
  split. am. au.
  nin H16. assert (inc (J x0 (Q y)) (disjoint_union f)).
  app inc_disjoint_union. cp (du_index_pr H12). ee. am. ee. wrr H6. wr H1.
  order_tac. exists (J x0 (Q y)). ee; split. aw. ee. am. am. au.
  nin H13. red. ir. elim H19. rw H20. aw. aw. ee. am. am. right. ee. tv.
  nin H18. am. nin H18. red. ir. elim H19. wr H20. aw.
```

We conclude with (II).

```
  assert (glt (V (Q x) g) (P x) (P y)). red. ee. am.  dneg.
  cp (du_index_pr H10);   cp (du_index_pr H12);ee. app pair_extensionality.
  assert (inc (Q x) (substrate r)). cp (du_index_pr H10); ee. ue.
  nin (H8 _ _ _ H15 H14). ee. nin (H18 _ _ H14).
  assert (inc (J x0 (Q x)) (disjoint_union f)). app inc_disjoint_union. ue.
  nin H20.  wr H6. order_tac. ue. exists (J x0 (Q x)). ee; split.
  aw. eee. right. ee. tv. nin H20. am. nin H20. red. ir. elim H23. rw H24. aw.
  aw. nin H22. eee. nin H22. red. ir.  elim H23. wr H24. aw. am.
Qed.
```

Second claim. We assume the index set non-empty.

```
Lemma Exercise1_19b: forall r f g,
  ordinal_sum_axioms1 r f g -> nonempty(substrate r) ->
  (forall i u, ~ (maximal_element (V i g) u)) ->
  (forall i, inc i (substrate r) -> without_gaps (V i g)) ->
  without_gaps (ordinal_sum r f g).
Proof. ir. rww Exercise1_19a. ee. right. nin H0. cp (H2 _ H0). red in H3. ee.
  nin H4. nin H4. exists y. exists x. exists x0. eee. ir.  app H2.
  ir. right. left. ir. app H1.
Qed.

Lemma Exercise1_19c: forall r f g,
  ordinal_sum_axioms1 r f g -> nonempty(substrate r) ->
  (forall i u, ~ (minimal_element (V i g) u)) ->
  (forall i, inc i (substrate r) -> without_gaps (V i g)) ->
  without_gaps (ordinal_sum r f g).
Proof. ir. rww Exercise1_19a. ee. right. nin H0. cp (H2 _ H0). red in H3. ee.
  nin H4. nin H4. exists y. exists x. exists x0. eee. ir.  app H2.
  ir. right. right. ir. app H1.
Qed.
```

Last claim. The condition "Each E_ι is either without gaps or contains no two distinct comparable elements", is nothing else than (II).

```
Lemma Exercise1_19d: forall r f g,
  ordinal_sum_axioms1 r f g ->
  without_gaps r ->
    (forall i, inc i (substrate r) ->
      (without_gaps (V i g) \/
        (forall x y, inc x (V i f) -> inc y (V i f) -> ~ (glt (V i g) x y))))
```

```
    ->  without_gaps (ordinal_sum r f g).
Proof. ir.  rww Exercise1_19a. ee. left. nin H0. ee. am. ir.
  nin (H1 _ H2). am. nin H. red in H. ee. assert (inc x (V i f)). wr H11.
  order_tac. ue. assert (inc y (V i f)). wr H11. order_tac. ue.
  cp (H4 _ _ H12 H13). contradiction. ir. red in H0. ee. au.
Qed.
```

¶ 20. *An ordered set* E *is said to be* scattered *of no ordered subset of* E *is without gaps (Exercise 19). Every subset of a scattered set is scattered. *Every well-ordered set of more than one element is scattered.*$_*$

The first claim is obvious.

```
Definition scattered r := order r &
  (forall x, sub x (substrate r) -> ~ (without_gaps (induced_order r x))).

Lemma Exercise1_20a : forall r x,
  sub x (substrate r) -> scattered r -> scattered (induced_order r x).
Proof. ir. red in H0; red. ee. fprops. ir. awii H2.
  wrr induced_trans. app H1. apply sub_trans with x. am. am.
Qed.
```

Every well-ordered set is scattered. In fact, if $F \subset E$, and has at least two elements, let x be the least element of F and y the least of $F - \{x\}$. Then $]x, y[$ is empty.

```
Lemma Exercise1_20b : forall r, worder r -> scattered r.
Proof. ir. red. ee. nin H ;am. ir. red. ir.
  assert (worder (induced_order r x)). app worder_restriction.
  set (y:= induced_order r x) in *.  red in H1. ee. nin H3. nin H3.
  assert (inc x0 (substrate y)). order_tac.
  assert (nonempty (substrate y)). exists x0. am.
  nin (worder_least H2 H6).
  set (z:= complement (substrate y) (singleton x2)).
  assert (sub z (substrate y)). uf z. app sub_complement.
  assert (inc x1 z). uf z. srw. ee. order_tac. aw. red. ir. rwi H9 H3.
  red in H7. ee. cp (H10 _ H5). order_tac.
  assert (nonempty z). exists x1. am. nin H2. nin (H11 _ H8 H10).
  assert (glt y x2 x3). nin H12. awi H12. nin H7. ufi z H12. srwi H12.
  nin H12. awi H15. split. app H14. au. am. am.
  nin (H4 _ _ H13). nin H14. assert (inc x4 z). uf z. srw. ee. order_tac.
  aw. nin H14. au. nin H12. awii H17. cp (H17 _ H16). awii H12. awii H18.
  order_tac.
Qed.
```

(a) Suppose that E *is scattered. Then if x, y are two elements of* E *such that $x < y$, there exists two elements x', y' of* E *such that $x \leq x' < y' \leq y$, and such that the interval $]x', y'[$ is empty. *Give an example of a totally ordered set which satisfies this condition and is not scattered (consider Cantor's triadic set).*$_*$ We prove that $\exists x, \exists y,$ P is the negation of $\forall x, \forall y, \neg$P.

```
Lemma exists_proof2 : forall p : EEP, ~ (forall x y: Set, ~ p x y)
  -> (exists x, exists y, p x y).
Proof. ir. nin (p_or_not_p (exists x, exists y, p x y)). am. elim H. ir.
```

```
  red. ir. elim H0. exists x.  exists y. am.
Qed.
```

We just say that $[x, y]$ is not without gaps.

```
Lemma  Exercise1_20c: forall r x y,
  scattered r -> glt r x y ->
  exists x', exists y', glt r x' y'
    & (forall z, ~ (glt r x' z & glt r z y')).
Proof. ir. nin H. set (F:= interval_cc r x y).  assert (sub F (substrate r)).
  uf F. uf interval_cc. app Z_sub. cp (H1 _ H2).
  app exists_proof2. dneg. clear H3.
  split. fprops. split.  exists x; exists y. nin H0. split. aw.
  uf F. uf interval_cc. Ztac. order_tac. ee. order_tac. order_tac. am.
  uf F. uf interval_cc. Ztac. order_tac. ee. am. order_tac. order_tac. am.
  ir. app exists_proof. assert (glt r x0 y0).
  ap (related_induced_order2 H3). cp (H4 x0 y0). dneg.
  ee. am. ir. cp (H7 z). dneg. nin(related_induced_order4 H3).
  nin H9. nin H9; nin H12.
  assert (inc z F). uf F. uf interval_cc. Ztac. order_tac. ee.
  ufi F H10. ufi interval_cc H10.  Ztac. ee. order_tac.
  ufi F H11. ufi interval_cc H11.  Ztac. ee. order_tac.
  uf glt. ee. aw. am. aw. am.
Qed.
```

Let's consider the set of all function $\mathrm{N} \to \{a, b,\}$ ordered lexicographically, where $a < b$. The Cantor set is defined by $a = 0$ and $b = 2$, and to each function f, one associates the value $\sum_i f(i)/3^i$, considered as a real number. The lexicographic ordering is compatible with the ordering of the real numbers. We do not need real numbers here. We first show that the set is totally ordered.

```
Definition cantor_tri_order:=
  lexicographic_order Bnat_order
  (L Bnat (fun _: Set => two_points))
  (L Bnat (fun _: Set => example_worder)).

Lemma cantor_tri_order_axioms: lexicographic_order_axioms Bnat_order
  (L Bnat (fun _: Set => two_points))
  (L Bnat (fun _: Set => example_worder)).
Proof. red. ee; bw;gprops. app worder_Bnat_order. app substrate_Bnat_order.
  ir. bw. nin example_is_worder. am. ir. bw. rww substrate_example_worder.
Qed.

Lemma cantor_tri_order_total : total_order cantor_tri_order.
Proof. uf cantor_tri_order. app total_lexicographic_order.
  app cantor_tri_order_axioms. ir. bwi H. bw. app worder_total.
  app example_is_worder. rww inc_Bnat.
Qed.
```

We pretend that $f \leq g$ if and only if, if i is the least index for which $f(i) \neq g(i)$ we have $f(i) = a$ and $g(i) = b$. We give a simple condition for $f < g$.

```
Lemma related_cantor_tri_order: forall x x',
  related cantor_tri_order x x' =
```

```
  (inc x (productb  (L Bnat (fun _: Set => two_points)))
    & inc x' (productb  (L Bnat (fun _: Set => two_points)))
    & forall j, least_element (induced_order Bnat_order (Zo Bnat
      (fun i =>  V i x <> V i x'))) j -> (V j x = TPa & V j x' = TPb)).
Proof. ir. uf cantor_tri_order. rw related_lexicographic_order.
  assert (Ha: forall j p, least_element (induced_order Bnat_order (Zo Bnat p)) j
    -> inc j Bnat). ir. red in H. ee. awi H. Ztac. am. nin worder_Bnat_order.
  am.  rw substrate_Bnat_order. app Z_sub.
  ap iff_eq; ir; eee. ir. bwi H1. cp (H1 _ H2). bwi H3.
  nin H3. rwi example_worder_related H3. nin H3. nin H3. elim  H4. ue.
  nin H3. nin H3. elim H4. ue. am.  app (Ha _ _ H2).
  ir. bwi H2.  cp (H1 _ H2).  split.  bw. rw example_worder_related. au.
  app (Ha _ _ H2). nin H3. rw H3; rw H4. fprops. app cantor_tri_order_axioms.
Qed.


Lemma glt_cantor_tri_order: forall x x',
  glt cantor_tri_order x x' =
  (inc x (productb  (L Bnat (fun _: Set => two_points)))
    & inc x' (productb  (L Bnat (fun _: Set => two_points)))
    & exists j, inc j Bnat &
      (forall i, inc i Bnat -> cardinal_lt i j -> V i x = V i x')
      & (V j x = TPa & V j x' = TPb)).
Proof. ir.  set (Z:=Zo Bnat (fun i : Set => V i x <> V i x')).
  assert (Hb: order Bnat_order).  nin (worder_Bnat_order). am.
  assert (Hc:sub Z (substrate Bnat_order)). rw substrate_Bnat_order. uf Z.
  app Z_sub.
  assert (Ha: forall j p, least_element (induced_order Bnat_order (Zo Bnat p)) j
    -> inc j Bnat). ir. red in H. ee. awi H. Ztac. am. am.
  rw substrate_Bnat_order. app Z_sub.
  uf glt. rw related_cantor_tri_order. ap iff_eq; ir; eee.
  nin (emptyset_dichot Z). elim H0.
  assert (fgraph  (L Bnat (fun _ : Set => two_points))). gprops.
  rw (productb_extensionality H4 H H1). bw. ir.
  nin (p_or_not_p (V i x = V i x')). am. elim (emptyset_pr (x:=i)).
  wr H3. uf Z. Ztac. fold Z in H2.   nin (worder_Bnat_order).
  nin (H5 _ Hc H3). cp (H2 _ H6). exists x0. ee.
  app (Ha _ _ H6). ir. red in H6. awii H6. ee.
  nin (p_or_not_p (V i x = V i x')). am. assert (inc i Z). uf Z. Ztac.
  cp (H11 _ H13). cp  (related_induced_order1 H14).
  assert (cardinal_le x0 i). rwi related_Bnat_order H15. red in H15. ee; am.
  co_tac. am. am.
  ir. nin H1. nin H1. nin H3. red in H2. fold Z in H2. awii H2. nin H2.
  assert (inc x0 Z). uf Z. Ztac. nin H4. rw H4; rw H6. fprops.
  cp (H5 _ H6). cp (related_induced_order1 H7). assert (cardinal_le j x0).
  rwi related_Bnat_order H8. red in H8. ee; am.
  assert (is_cardinal j). red in H9; eee.
  assert (is_cardinal x0). red in H9; eee.
  nin (cardinal_le_total_order2 H11 H10). assert (x0 = j). co_tac. wrr H13.
  assert (inc j Bnat).  ufi Z H2. Ztac. am. cp (H3 _ H13 H12).
  ufi Z H2. Ztac. contradiction. nin H1. ee. red. ir.  assert (TPa = TPb).
  wr H3; wr H4; wrr H5. fprops.
Qed.
```

Consider two functions $f < g$. Assume that they agree up to (but excluded) index i. Consider f' and g', that agree with f and g up to i, whose value is constantly b or a respectively, for indices $> i$. Then $f \leq f' < g' \leq g$, and the interval $]f', g'[$ is empty.

```
Lemma Exercise1_20d:   let r:=  cantor_tri_order in
  forall x y, glt r x y ->
  exists x', exists y', glt r x' y'
    & (forall z, ~ (glt r x' z & glt r z y')).
Proof. ir. cp H. ufi r H0. rwi glt_cantor_tri_order H0. ee. nin H2. ee.
  set (f:= fun i=> Yo (cardinal_le i x0) (V i x) TPb).
  set (g:= fun i=> Yo (cardinal_le i x0) (V i y) TPa).
  assert (fgraph (L Bnat (fun _ : Set => two_points))). gprops.
  assert (inc (L Bnat f)  (productb (L Bnat (fun _ : Set => two_points)))).
  rww productb_pr. ee. gprops. bw. bw. ir. bw. uf f.
  nin (p_or_not_p (cardinal_le i x0)). rww Y_if_rw. rwi productb_pr H0. ee.
  bwi H9. rwi H9 H10. cp (H10 _ H7).  bwi H11. am. am. gprops.
  rww Y_if_not_rw.  fprops.
  assert (inc (L Bnat g)  (productb (L Bnat (fun _ : Set => two_points)))).
  rww productb_pr. ee. gprops. bw. bw. ir. bw. uf g.
  nin (p_or_not_p (cardinal_le i x0)). rww Y_if_rw. rwi productb_pr H1. ee.
  bwi H10. rwi H10 H11. cp (H11 _ H8).  bwi H12. am. am. gprops.
  rww Y_if_not_rw. fprops.
  exists (L Bnat f). exists (L Bnat g). ee. rw glt_cantor_tri_order. eee.
  exists x0. ee. am. ir. nin H10. uf f; uf g. bw. rww Y_if_rw. rww Y_if_rw.
  app H3. split; am. uf f. bw. rww Y_if_rw. fprops. uf g. bw.
  rww Y_if_rw. fprops. red. ir. ee.
  rwi glt_cantor_tri_order H9. rwi glt_cantor_tri_order H10. ee.
  nin H14. nin H12. ee.
  assert (is_cardinal x0). fprops.
  assert (is_cardinal x1). fprops.
  assert (is_cardinal x2). fprops.
  nin (cardinal_le_total_order2 H23 H21).
  nin (cardinal_le_total_order2 H22 H21).
  nin (cardinal_le_total_order1 H22 H23). assert (TPa = TPb).  wr H16.
  wr H20. rww H26. fprops. nin H26. cp (H15 _ H14 H26). rwi H20 H27.
  assert (cardinal_lt x1 x0). co_tac. ufi g H27. bwi H27.  cp (H3 _ H14 H28).
  nin H28. rwii Y_if_rw  H27. ufi f H19. bwi H19. rwii Y_if_rw  H19.
  wri H27 H29. rwi H19 H29. fprops. am. am.
  assert (cardinal_lt x2 x0). co_tac. ufi g H17. bwi H17.  cp (H3 _ H12 H27).
  nin H27. rwii Y_if_rw  H17. rwi H17 H28. cp (H18 _ H12 H26). rwi H16 H30.
  ufi f H30. bwi H30. rwii Y_if_rw  H30. rwi H30 H28. fprops. am. am.
  ufi f H19. bwi H19. rwi Y_if_not_rw H19. fprops. red. ir. co_tac. am.
  ufi g H17. bwi H17. rwi Y_if_not_rw H17. fprops. red. ir. co_tac. am.
Qed.
```

The Cantor set is not scattered. Let F be the set of sequences not of the form $x_i = a$ for $i > i_0$. We pretend that F has at least two elements.

```
Lemma  Exercise1_20e: ~ (scattered cantor_tri_order).
Proof. red. ir. red in H. ee.
  set (w:= productb (L Bnat (fun _ : Set => two_points))).
  assert (w = substrate cantor_tri_order). uf cantor_tri_order.
  rww substrate_lexicographic_order. app cantor_tri_order_axioms.
  set (all_a := Zo w (fun z => exists i, inc i Bnat & forall j,
    inc j Bnat -> cardinal_le i j -> V j z = TPa)).
  set (s:= complement w all_a). assert (sub s w). uf s. app sub_complement.
```

```
set (zb:= L Bnat (fun _ : Set => TPb)). assert (inc zb s). uf s. srw.
ee. uf w. rw productb_pr. uf zb. ee.  gprops. bw. bw. ir. bw. fprops.
gprops. red. ir. ufi all_a H3. Ztac. nin H5. ee.
assert (cardinal_le x x). fprops. cp (H6 _ H5 H7). ufi zb H8. bwi H8.
fprops. am.
set (zab:= L Bnat (fun i : Set => Yo (i=card_zero) TPa TPb)).
assert (inc zab s). uf s. srw.
ee. uf w. rw productb_pr. uf zab. ee.  gprops. bw. bw. ir. bw.
nin (equal_or_not i card_zero).  rww Y_if_rw. fprops. rww Y_if_not_rw. fprops.
gprops. red. ir. ufi all_a H4. Ztac. nin H6. ee.
assert (cardinal_le x (succ x)). app is_less_than_succ. fprops.
assert (inc (succ x) Bnat). fprops.  cp (H7 _ H9 H8). ufi zab H10. bwi H10.
rwi Y_if_not_rw H10. fprops. app succ_nonzero. fprops.
assert (glt cantor_tri_order zab zb). rw glt_cantor_tri_order.
fold w. ee. ufi s H4. srwi H4. ee; am.  ufi s H3. srwi H3. ee; am.
exists card_zero. ee. fprops. ir. elim (zero_smallest1 H6).
uf zab. bw. rww Y_if_rw. fprops. uf zb. bw. fprops.
```

Assume now $f < g$ and $g \in$ F. For some index we have $f(i) = a$ and $g(i) = b$. There is a least $j > i$ such that $g(j) = b$, since $g \in$ F. Let h be like g with $h(j) = a$. This is in F.

```
wri H1 H0. cp (H0 _ H2). elim H6. split. rwi H1 H2. fprops. ee.
exists zab. exists zb. nin H5. split. aw. am.
ir. cp (related_induced_order2 H7). nin H7. red in H7. red in H7.
ufi induced_order H7. cp (intersection2_second H7). clear H7.
ufi coarse H10. awi H10. ee. clear H7. rwi glt_cantor_tri_order H8.  ee.
nin H12. ee. set (u:= Zo Bnat (fun i => cardinal_lt x0 i & V i y = TPb)).
nin (emptyset_dichot u). ufi s H11. srwi H11. nin H11. elim H17. uf all_a.
Ztac. exists (succ x0). ee. fprops. ir.
rwi lt_n_succ_le0 H19.  assert (inc (V j y) two_points).
ufi w  H11. rwi productb_pr H11. ee. bwi H20. rwi H20 H21. cp (H21 _ H18).
bwi H22. am. am. gprops. rwi two_points_pr H20. nin H20. am.
elim (emptyset_pr (x:=j)). wr H16. uf u. Ztac. fprops. fprops.
nin worder_Bnat_order.  assert (sub u (substrate Bnat_order)).
rw substrate_Bnat_order. uf u. app Z_sub. nin (H18 _ H19 H16).
red in H20. awii H20. clear H18. nin H20.
set (z:= L Bnat (fun i=> Yo (i= x1) TPa (V i y))).
assert (inc z s). uf z. uf s. srw. ee. uf w. rw productb_pr. ee. gprops. bw.
bw.  ir. bw. nin (equal_or_not i x1).  rww Y_if_rw. fprops. rw Y_if_not_rw.
ufi s H11. srwi H11. nin H11. ufi w H11. rwi productb_pr H11. ee.
bwi H24. rwi H24 H25. cp (H25 _ H21). bwi H26. am. am. gprops. am. gprops.
red. ir. ufi all_a H21. Ztac. nin H23. ee. ufi s H11. srwi H11. nin H11.
elim H25.  uf all_a. clear H21. Ztac.
assert (inc x1 Bnat). ufi u H18. srwi H18. Ztac.  am.
set (a:= succ (card_plus x1 x2)). assert (inc a Bnat). uf a.  fprops.
assert (cardinal_le x2 a). uf a. uf succ.
assert (cardinal_le x2 (card_plus x1 x2)). rw card_plus_commutative.
app sum_increasing3. fprops. fprops.
assert (cardinal_le (card_plus x1 x2) (card_plus (card_plus x1 x2) card_one)).
app sum_increasing3. fprops. fprops. co_tac.
exists a. ee. am.  ir.  assert (cardinal_le x2 j). co_tac.
cp (H24 _ H28 H30). bwi H31. rwi Y_if_not_rw H31. am. red. ir.
rwi H32 H29. assert (cardinal_lt x1 a). uf a. rw lt_is_le_succ.
app sum_increasing3. fprops. fprops. fprops. co_tac. am.
```

We have $f < h < g$. Remember that g and h agree everywhere but at j, where $h(j) < g(j)$; thus $h < g$. This implies that f and h agree up to $i-1$, $f(i) < g(i) = h(i)$, hence $f < h$.

```
  assert (glt cantor_tri_order x z). rw glt_cantor_tri_order. fold w.
  ee. ufi s H10. srwi H10. ee; am.  ufi s H21. srwi H21. ee; am.
  exists x0. ee. am. ir. rww H13. uf z. bw. rw Y_if_not_rw. tv.
  ufi u H18. Ztac. ee. assert (cardinal_lt i x1). co_tac. nin H27. am.
  am. uf z. bw.  rw Y_if_not_rw. tv.
  ufi u H18. Ztac. ee. nin H23. am.
  assert (glt cantor_tri_order z y). rw glt_cantor_tri_order. fold w.
  ee. ufi s H21. srwi H21. ee. am. ufi s H11. srwi H11. ee; am.
  exists x1. ee. ufi u H18. Ztac. am. ir. uf z. bw. rw Y_if_not_rw. tv.
  nin H24. am. uf z. bw. rww Y_if_rw.  ufi u H18. Ztac. am.
  ufi u H18. Ztac.  ee. am.
  exists z. nin H22; nin H23. uf glt. ee. aw. am. aw.  am.
```

(b) An ordinal sum $\sum_{\iota\in I} E_\iota$ *(where neither* I *nor any* E_ι *is empty) is scattered if and only if* I *and each* E_ι *is scattered. (Note that* E *contains a subset isomorphic to* I *and that every subset* F *of* E *is the ordinal sum of those sets* $F \cap E_\iota$ *which are non-empty; finally use Exercise 19.)*

Bourbaki specifies here $E_\iota \neq \emptyset$. This condition was also needed in Exercise 19. He says I nonempty, this is not needed: if I is empty, the sum is empty, I and the sum are scattered, the condition "E_ι is scattered" is trivially true. Assume the sum scattered. We prove I is scattered, because for each $\alpha \in I$, there is $e_\alpha \in E_\alpha$; if this consider this as an element of the disjoint union, we a subset of E isomorphic to I.

```
Lemma  Exercise1_20f:  forall r f g,
  ordinal_sum_axioms1 r f g ->
  scattered (ordinal_sum r f g) =
  (scattered r & forall i, inc i (domain f) -> scattered (V i g)).
Proof. ir. cp H. nin H. cp H. red in H.  ee.
  ap iff_eq. ir. ee. red. ee. am. ir. red. ir.
  set (w:= fun_image x (fun i => (J (rep (V i f)) i))).
  assert (sub w (substrate (ordinal_sum r f g))). red. ir. aw.
  ufi w H12. awi H12. nin H12.  ee. wr H13. app inc_disjoint_union. ue.
  app nonempty_rep. app H1. ue. nin H9. cp (H13 _ H12).
  elim H14. red in H11. ee. red. ee. fprops. nin H15. nin H15.
  cp (related_induced_order4 H15). ee.  cp (related_induced_order2 H15).
  assert (inc x0 (domain f)). wr H3. app H10.
  assert (inc x1 (domain f)). wr H3. app H10.
  exists (J (rep (V x0 f)) x0); exists (J (rep (V x1 f)) x1).
  split. aw. ee.  app inc_disjoint_union. app nonempty_rep. app H1.
  app inc_disjoint_union. app nonempty_rep. app H1. au.
  uf w. aw. exists x0.  eee. uf w. aw.
  exists x1. eee. nin H19. dneg. inter2tac. ir.
  cp (related_induced_order2 H17). nin H18.
  nin H17. ufi induced_order  H17. cp (intersection2_second H17).
  ufi coarse H21. awi H21. ee. ufi w H22. awi H22. nin H22.
  ufi w H23. awi H23. nin H23. ee. awi H18. ee. nin H27.
  wri H25 H27. wri H24 H27. awi H27.
  assert (glt (induced_order r x) x1 x2). nin H27. split. aw. am.
  nin (H16 _ _ H28). assert (inc (J (rep (V x3 f)) x3) w).
  uf w. aw. nin H29. cp (related_induced_order4 H29).
  cp (related_induced_order4 H30). ee.
  exists x3. ee. am. tv. awi H12. nin H29.
  cp (related_induced_order2 H29). cp (related_induced_order2 H31).
  exists (J (rep (V x3 f)) x3). ee. uf glt. ee. aw.
  ee. am. app H12. wr H25. aw.  au. wr H25. uf w. aw. exists x1. eee.
  wr H25. nin H32. dneg. inter2tac. split. aw. ee. app H12. am.
```

```
wr H24. aw. au. wr H24. uf w. aw. exists  x2. eee. wr H24. nin H33.
dneg. inter2tac. am. elim H19. wr H25. wr H24.  nin H27.
wri H25 H27. wri H24 H27. awi H27. ue. am.
```

Assume that the sum scattered. We show that E_α is scattered, since this is naturally a subset of E.

```
ir. split. app H7. ir. set (y:= fun_image x (fun u => (J u i))).
assert (sub y (substrate (ordinal_sum r f g))). aw.  red. ir. ufi y H12.
awi H12. nin H12. nin H12.  wr H13. app inc_disjoint_union. wr H8. app H11.
am. nin H9. cp (H13 _ H12). dneg. nin H15. nin H16. nin H16; nin H16.
nin (related_induced_order4 H16). nin (related_induced_order2 H16).
red. ee.  fprops. exists (J x0 i). exists (J x1 i).
split. aw. ee. app inc_disjoint_union. wrr H8. order_tac.
app  inc_disjoint_union. wrr H8. order_tac. right. au. uf y. aw.
exists x0. au. uf y. aw. exists x1. au. dneg. inter2tac.
ir.  nin (related_induced_order4 H22). nin (related_induced_order2 H22).
ufi y H23. awi H23. nin H23. ufi y H24. awi H24. nin H24. ee.
wri H28 H25; wri H27 H25; awi H25. ee. nin H30. nin H30. elim H31. tv.
nin H30. assert (glt (induced_order (V i g) x) x3 x4). split. aw. dneg.
wr  H28; wr H27. ue. nin (H17 _ _ H32). nin H33.
nin (related_induced_order4 H33). nin (related_induced_order2 H33).
nin (related_induced_order4 H34). nin (related_induced_order2 H34).
assert (inc (J x5 i) y). uf y. aw.  exists x5. au.
assert (inc (J x5 i) (disjoint_union f)). app inc_disjoint_union.
wr H8. order_tac. am.  wr H28. wr H27. exists (J x5 i). uf glt. ee; aw. eee.
uf y. aw. exists x3. au. clear H42. dneg. inter2tac. eee. uf y.
aw. exists x4. au. dneg. inter2tac. am.
```

We now prove that the sum is scattered. We consider a subset E' of the ordinal sum E. We pretend that this is an ordinal sum of a restriction.

```
ir. red. ee. fprops. ir. red. ir.
set (ns := Zo (substrate r) (fun z => exists i, inc i x & z = Q i)).
set (r' := induced_order r ns).
assert (sub ns (substrate r)). uf ns. app Z_sub.
assert (order r'). uf r'. fprops.
assert  (substrate r' = ns). uf r'.  app substrate_induced_order.
set (f':= L ns (fun i=> Zo (V i f) (fun u => inc (J u i) x))).
assert (forall i, inc i ns -> sub (V i f') (V i f)). ir. uf f'. bw.
app Z_sub.
set (g' := L ns (fun i=> induced_order (V i g) (V i f'))).
assert (nonempty x). nin H12.  nin H17. nin H17. nin H17.
nin (related_induced_order4 H17).  exists x0. am.
assert (ordinal_sum_axioms1 r' f' g'). uf f'; uf g'. split. red. ee. am.
rw H15. bw. gprops. gprops. bw. bw. ir. bw. cp (H13 _ H18). rwi H3 H19.
cp (H7 _ H19). app order_induced_order.  rw H8. app H16. am. bw. ir. bw.
cp (H13 _ H18). rwi H3 H19. aw. uf f'. bw. app H7.  rww H8.  app H16.
bw. ir. bw. ufi ns H18. Ztac. nin H20. nin H20. clear H18.
cp (H11 _ H20). awi H18. nin (du_index_pr H18). nin H23.
exists (P x0). Ztac. rww H21. rww H21. aw. am.
assert (x =  substrate (ordinal_sum r' f' g')). aw.
set_extens. awi H11. cp (H11 _ H19). nin (du_index_pr H20). nin H22.
assert (x0 = J (P x0) (Q x0)). aw.  assert (inc (Q x0) ns). uf ns.
Ztac. rww H3. exists x0. au. rw H24. app inc_disjoint_union. uf f'.
bw. uf f'. bw. Ztac. wrr H24. am. cp (du_index_pr H19). ee.
```

```
ufi f' H21. bwi H21. Ztac. awi H24. am. am. ufi f' H20. bwi H20. am.
nin H18. am.
```

We pretend that the ordering is the same.

```
assert (induced_order (ordinal_sum r f g) x = (ordinal_sum r' f' g')).
uf induced_order. set_extens. nin (intersection2_both H20).
ufi coarse H22. awi H22. ee. set (a:= P x0) in *. set (b := Q x0) in *.
assert (x0 = J a b). uf a; uf b; aw.  rw H25.
change (gle (ordinal_sum r' f' g') a b). aw. uf f'.
rwi H25 H21.  change (gle (ordinal_sum r f g) a b) in H21. awi H21.
nin H21; nin H26.  nin (du_index_pr H21). nin H29.
nin (du_index_pr H26). nin H32.  assert (a = J (P a) (Q a)). aw.
assert (b = J (P b) (Q b)). aw.
assert (inc (Q a) ns). uf ns. Ztac. rww H3. exists a. au.
assert (inc (Q b) ns). uf ns. Ztac. rww H3. exists b. au.
ee. rw H34. app inc_disjoint_union. bw. bw. Ztac. wrr H34.
rw H35. app inc_disjoint_union. bw. bw. Ztac. wrr H35.
uf r'; uf g'. nin H27. left. nin H27. split. aw. am. right. ee. am.
red. red. bw. uf f'. bw. uf induced_order. app intersection2_inc.
uf coarse. aw. ee. fprops. Ztac. wrr H34. Ztac. rw H27. am. rw H27. wrr H35.
am. nin H18; am. assert (is_graph  (ordinal_sum r' f' g')).
app order_is_graph. nin H18; fprops. assert (is_pair x0). app H21.
assert (x0 = J (P x0) (Q x0)). aw. rw H23. rwi H23 H20.
change (gle  (ordinal_sum r' f' g') (P x0) (Q x0)) in H20.
assert (inc (P x0) (substrate (ordinal_sum r' f' g'))). order_tac.
assert (inc (Q x0) (substrate (ordinal_sum r' f' g'))). order_tac.
wri H19 H24; wri H19 H25. uf coarse. app intersection2_inc.
set (a:= P x0) in *. set (b := Q x0) in *.
change (gle (ordinal_sum r f g) a b). awi H20. ufi f' H20; ufi g' H20.
ufi r' H20. ee. cp (du_index_pr H20). bwi H28. ee.
cp (du_index_pr H26). bwi H31. ee.  Ztac.  clear H32. Ztac. clear H29.
assert (a = J (P a) (Q a)). aw. assert (b = J (P b) (Q b)). aw.
aw. ee. rw H29. app inc_disjoint_union. wr H3. app H13. rw H37.
app inc_disjoint_union. wr H3. app H13. nin H27. left.
ap (related_induced_order2 H27). right. ee. am.
bwi H38. cp (related_induced_order1 H38). am. am. ee. am. ee; am.
nin H18; am. aw. ee; am.
```

We have now the assumption that E' is without gaps. If E'_α has two elements $x < y$, then there is z such that $x < z < y$ in the sum, hence in E'_α, contradiction with condition (II). In particular all elements are maximal and minimal, so that condition (III) becomes: if $\alpha < \beta$ in I', then the interval $]\alpha,\beta[$ is non-empty. The conclusion follows.

```
rwi H20 H12. rwi (Exercise1_19a H18) H12. ee.
assert (forall i x y, inc i (substrate r') -> ~ glt (V i g') x y).
ir. nin (p_or_not_p (glt (V i g') x0 y)). cp (H21 _ _ _ H23 H24).
assert (inc i (domain f)). wr H3. app H13. wrr H15. cp (H10 _ H26).
ufi g' H25. bwi H25. nin H27. assert (sub (V i f') (substrate (V i g))).
rww H8.  app H16.  wrr H15. elim (H28 _ H29). am. wrr H15. am.
red in H9. ee. cp (H24 _ H13). elim H25. red. ee. fprops.
nin H12. nin H12. nin H12. exists x0; exists x1. am.
nin H12. nin H12. nin H12. ee. elim (H23 x0 x1 x2 H12). am.
ir. fold r' in H26. fold r'. nin (H22 _ _ H26). am.
nin H27. nin H18. assert (inc x0 (domain f')). uf f'. bw. wr H15. order_tac.
nin (H28 _ H29). elim (H27 y0).  split.  red in H18. ee. rww H36. ir.
```

```
  nin (equal_or_not x1 y0). am. assert (glt (V x0 g') y0 x1). split. am.
  intuition. elim (H23 x0 y0 x1). rww H15. ufi f' H29.  bwi H29. am. am.
  assert (inc y ns). wr H15. order_tac. assert (inc y (domain f')). uf f'. bw.
  nin H18.  nin (H30 _ H29). elim (H27 y0). split. red in H18. ee. rww H37. ir.
  nin (equal_or_not x1 y0). am. assert (glt (V y g') x1 y0). split; am.
  elim (H23 y x1 y0). rww H15. am.
Qed.
```

---

21. *Let* E *be a non-empty totally ordered set, and let* S$\{x, y\}$ *be the relation "the closed interval with endpoints x, y is scattered" (Exercise 20). Show that* S *is an equivalence relation which is weakly compatible (Exercise 2) in x and y with the order relation on* E*, that the equivalence classes with respect to* S *are scattered sets, and that the quotient ordered set* E/S *is either without gaps or else consists of a single element. Deduce that* E *is isomorphic to an ordinal sum of scattered sets whose index set is either without gaps or else consists of a single element.*

We start with a complement to Exercise 1.2. Assume that both conditions (C) and (C') are satisfied and that $\leq$ is a total order on E. Then the quotient order E/S is totally ordered, and $X \leq Y$ in the quotient is equivalent to $x \leq y$ whenever $x \in X$ and $y \in Y$, and we may take the representatives of X and Y for x and y.

```
Lemma Exercise1_2g: forall r s, weak_order_compatibility r s->
  quotient_order_axiom r s -> total_order r ->
  let r' := (quotient_order r s) in
    forall x y, gle r' x y = (inc x (quotient s) & inc y (quotient s)
      & gle r (rep x) (rep y)).
Proof. ir. rename H1 into Ht. red in H. red in H0. ee.
  assert (forall u, inc u (quotient s) -> inc (rep u) u). ir.
  app (inc_rep_itself H1 H4).
  uf r'. rw quotient_order_pr. uf quotient_order_r.
  ap iff_eq; ir; eee; ir;cp (H4 _ H5); cp (H4 _ H6).
  nin Ht. assert (inc (rep x) (substrate r)). wr H2.
  app (inc_in_quotient_substrate H1 H8 H5).
  assert (inc (rep y) (substrate r)). wr H2.
  app (inc_in_quotient_substrate H1 H9 H6).
  nin (H11 _ _ H13 H12). nin (H7 _ H8). ee.  assert (gle r (rep y) x0).
  order_tac. assert (related s (rep y) x0). rwi inc_quotient H6.
  rwi is_class_rw H6. ee. wrr (H19 (rep y) x0 H9). am. am.
  cp (H0 _ _ _ H14 H16 H18). rwii related_rep_rep H19. rw H19. order_tac. am.
  assert (related s (rep x) x0). rw in_class_related. exists x. eee.
  wrr inc_quotient. am. nin (H3 _ _ _ H7 H11). exists  x1. eee.
  rwi inc_quotient H6. rwi is_class_rw H6. ee. rww (H15 (rep y) x1 H10). am. am.
Qed.

Lemma Exercise1_2h: forall r s, weak_order_compatibility r s->
  quotient_order_axiom r s -> total_order r ->
  total_order (quotient_order r s).
Proof. ir. cp (Exercise1_2g H H0 H1). cp H. cp H1. red in H. red in H1. ee.
  cp (Exercise1_2d H6 H1 H7 H0). split. am. ir.
  rwi substrate_quotient_order H10. rwi substrate_quotient_order H11.
  assert (inc (rep x) (substrate r)). wr H7. app inc_rep_substrate.
  assert (inc (rep y) (substrate r)). wr H7. app inc_rep_substrate.
```

```
  nin (H5 _ _ H12 H13). left. rww Exercise1_2g. au. right.
  change (gle (quotient_order r s) y x).  rww Exercise1_2g. au.
  am. am. am. am. am. am.
Qed.
```

We prove now the last part of the exercise. We assume that S is an equivalence on E that satisfies the assumptions of the previous lemma. Let I be the quotient set E/S. Consider the identity function on I, and write it $i \mapsto \mathrm{E}_i$. This gives an ordinal sum $\sum_{i\in\mathrm{I}} \mathrm{E}_i$. The mapping $x \mapsto (x, \bar{x})$ is an order isomorphism, where $\bar{x}$ denotes the class of x for S. The ordinal sum is a disjoint union, and there is a natural equivalence S′; two elements x and y of E are related by S′ if and only if $f(x)$ and $f(y)$ are related by S′. Let Q be the quotient $\sum \mathrm{E}_i/\mathrm{S}'$; exercise 1-3 shows that this quotient is isomorphic to I (ordered by the quotient orders wrt S′ and S).

```
Lemma Exercise1_2i: forall r s,
  let q := quotient s in
    let r' := (quotient_order r s) in
      let f' := L q (fun z: Set => z) in
        let g' := L q (fun z => induced_order r z) in
          let du := disjoint_union f' in
            let f :=  BL (fun x => J x (class s x)) (substrate r) du in
              weak_order_compatibility r s->
              quotient_order_axiom r s -> total_order r ->
              (ordinal_sum_axioms1 r' f' g'
                & substrate (ordinal_sum r' f' g') = du
                & (forall x y, inc x (substrate r) -> inc y (substrate r) ->
                  related s x y =
                  related (equivalence_associated (second_proj du))
                  (W x f) (W y f))
                & bijective f
                & order_isomorphism f r (ordinal_sum r' f' g')).
Proof. ir. cp (Exercise1_2g H H0 H1). cp (Exercise1_2h H H0 H1).
  nin H. nin H4. nin H5. nin H1.
  assert (substrate r' = q). uf r'. rww substrate_quotient_order.
  assert (forall i, inc i q -> sub i (substrate r)). red. ir.
  wr H5. app (inc_in_quotient_substrate H4 H10 H9).
  assert (Ha: ordinal_sum_axioms1 r' f' g').
  red. ee. red. ee. nin H3. am. uf f'. bw. uf f'. gprops. uf g'. gprops.
  uf f'; uf g'; bw. uf f'; uf g'. bw. ir. bw.  cp (H9 _ H10). fprops.
  uf f'; uf g'. bw. ir. bw. cp (H9 _ H10). aw.
  uf f'. bw. ir. bw. app (non_empty_in_quotient H4 H10).
  assert (Hb: transf_axioms (fun x => J x (class s x)) (substrate r) du).
  red. ir. uf du. app inc_disjoint_union. uf f'. bw. uf q.
  app inc_class_quotient. ue. uf f'. bw. wri H5 H10. equiv_tac.
  wri H5 H10. uf q. gprops.
  assert (Hc: bijective f). uf f. app bijective_bl_function. ir. inter2tac.
  ir. ufi du H10. cp (du_index_pr H10). ufi f' H11 . bwi H11. ee.
  exists (P y). ee. wr H5. app (inc_in_quotient_substrate H4 H12 H11).
  app pair_extensionality. ee. fprops. aw. aw. sy. app is_class_pr. nin H11. am.
  ee. am. aw. nin Ha; am.
  ir. cp Exercise1_3a6.
  change (related s x y  = related (E13_S f') (W x f) (W y f)). rw H12.
  uf E13_sF. fold du. ap iff_eq. ir. ee.  assert (du = target f). uf f. aw.
  rw H14. app inc_W_target. fct_tac.  uf f. aw. assert (du = target f). uf f.
  aw. rw H14.  app inc_W_target. fct_tac. uf f. aw. uf f. aw.
  app related_class_eq1. ir.
  ee. ufi f H15. awi H15. rww related_class_eq. wri H5 H10. equiv_tac.
```

```
  am. am. am. am. am.
  red. nin Ha. eee. aw. uf f. aw. uf f. aw. uf f. aw. ir. aw. ap iff_eq. ir. ee.
  app Hb. app Hb. uf g'. bw. nin (p_or_not_p (related s x y)). right. ee.
  app related_class_eq1. aw. bw.  assert (inc x (substrate r)). order_tac.
  wri H5 H16.  equiv_tac. bw. left. split.  uf r'. rw quotient_order_pr.
  red. ee. wri H5 H12. gprops. wri H5 H13. gprops. ir.
  assert (related s x x0). bwi H16. am. am.  nin (H6 x y x0 H14 H17).
  exists x1.  ee. bw. am. dneg. rww related_class_eq.
  wri H5 H12. equiv_tac.  wri H5 H12. uf q. gprops.
  ir. ee. nin H16. nin H16. ufi r' H16. rwi quotient_order_pr H16. red in H16.
  ee. nin (H7 _ _ H12 H13). am. assert (inc x (class s x)). bw. wri H5 H12.
  equiv_tac. nin (H19 _ H21). ee. red in H0.
  assert (related s y x0). bwi H22. am. am. cp (H0 _ _ _ H20 H23 H24).
  elim H17. app related_class_eq1. equiv_tac. ee. ufi g' H17. bwi H17.
  ap (related_induced_order1 H17). uf q. wri H5 H12. gprops.
Qed.
```

Let's write WG instead of "without gaps", and split it into WG1 and WG2. Consider an ordered set $\bar{E}$, with substrate E, and $V \subset E$. We write $\bar{V}$ when we consider the ordering induced on V by $\bar{E}$. By definition $\bar{V}$ is scattered if, for no $U \subset \bar{V}$, $\bar{U}$ is without gaps. The first lemma shows that in $\bar{U}$, we can use the ordering induced by $\bar{V}$ or $\bar{E}$ indifferently.

```
Definition scattered_rel r x y :=
  (gle r  x y &  scattered (induced_order r (interval_cc r x y)))
  \/ (gle r  y x &  scattered (induced_order r (interval_cc r y x))).

Definition scattered_equiv r := graph_on (scattered_rel r) (substrate r).

Lemma Exercise1_21a : forall r v, order r ->
  sub v (substrate r) ->
  scattered (induced_order r v) =
  (forall u, sub u v ->  ~ without_gaps (induced_order r u)).
Proof. ir. uf scattered. aw. ir. ap iff_eq. ir. ee. cp (H3 _ H2).
  wri (induced_trans H H2 H0) H4. am. ir. ee. fprops.
  ir. wr (induced_trans H H2 H0). app H1.
Qed.
```

Condition WG1 is: the set U has two distinct comparable elements. If the ordering is total, we can rewrite it as "there are two distinct elements in U", and its negation as "there is at most one element in U".

```
Lemma Exercise1_21b : forall r u, total_order r ->
  sub u  (substrate r) ->
  (exists x, exists y, glt (induced_order r u) x y)=
  (exists x, exists y, inc x u & inc y u & x<> y).
Proof. ir. ap iff_eq; ir; nin H1; nin H1.
  nin (related_induced_order4 H1). nin (related_induced_order2 H1).
  exists  x; exists x0. eee. nin H. ee.
  nin (H2 _ _ (H0 _ H1) (H0 _ H3)). exists  x; exists x0. split. aw.
  am. exists x0; exists x. split. aw. au.
Qed.

Lemma Exercise1_21c : forall r u, total_order r ->
  sub u  (substrate r) ->
  (forall a b, inc a u -> inc b u -> a = b) =
```

```
  ~ (exists x, exists y, glt (induced_order r u) x y).
Proof. ir. rww Exercise1_21b. ap iff_eq. ir. red. ir. nin H2. nin H2.
  ee. elim H4. app H1. ir. nin (equal_or_not a b). am. elim H1.
  exists a; exists b; au.
Qed.
```

We can now say: U is not without gaps if and only if either U is a small set, or it contains $a < b$ such that $]a, b[$ is empty.

```
Lemma Exercise1_21d : forall r u, total_order r ->
  sub u (substrate r) ->
  (~ without_gaps (induced_order r u)) =
  ((forall a b, inc a u -> inc b u -> a = b)
      \/ (exists a, exists b, inc a u & inc b u & glt r a b &
        (forall z, inc z u -> gle r z a \/ gle r b z))).
Proof. ir. ap iff_eq. ir.
  nin (p_or_not_p  (forall a b : Set, inc a u -> inc b u -> a = b)). au.
  nin (p_or_not_p (exists a, exists b, inc a u & inc b u &
    glt r a b & (forall z : Set, inc z u -> gle r z a \/ gle r b z))). au.
  elim H1. red. ee. nin H. fprops. rwi (Exercise1_21c H H0) H2.
  app excluded_middle. ir. app exists_proof. dneg.
  nin (related_induced_order4 H4). cp (related_induced_order2 H4).
  exists x; exists y. eee. ir. nin H.
  nin (H10 _ _ (H0 _ H9) (H0 _ H6)). au.
  nin (H10 _ _ (H0 _ H7) (H0 _ H9)). au. nin (equal_or_not x z). left.
  rw H13. cp (H0 _ H9). order_tac.  nin (equal_or_not z y). right.
  wr H14. cp (H0 _ H9). order_tac. elim (H5 z). uf glt. ee; aw.
  ir. red. ir. red in H2. ee. nin H1. rwi (Exercise1_21c H H0) H1.
  contradiction. nin H1. nin H1. ee. assert (glt (induced_order r u) x x0).
  nin H6. split. aw. am. nin (H4 _ _ H8). ee.
  nin (related_induced_order4 H9). cp (related_induced_order2 H9).
  nin (related_induced_order4 H10). cp (related_induced_order2 H10).
  nin H. nin (H7 _ H12). order_tac. order_tac.
Qed.
```

Assume that U is without gaps, and consider two elements a, b of U such that $a < b$. The intersection $U \cap [a, b]$ is without gaps.

```
Lemma Exercise1_21e: forall r u a b, total_order r ->
  let v:= intersection2 u (interval_cc r a b) in
  sub u (substrate r) -> without_gaps (induced_order r u) ->
  (exists x, exists y,  inc x v & inc y v & glt r x y) ->
  without_gaps (induced_order r v).
Proof. ir. cp H. nin H.
  assert (sub v u). uf v. app intersection2sub_first.
  assert (sub v (interval_cc r a b)). uf v. app intersection2sub_second.
  assert (sub v (substrate r)). apply sub_trans with u. am. am.
  nin (p_or_not_p (without_gaps (induced_order r v))). am.
  assert (~ ~ without_gaps (induced_order r u)). red. ir. contradiction.
  rwi (Exercise1_21d H3 H7) H8. rwi (Exercise1_21d H3 H0) H9. elim H9. clear H9.
  nin H8. nin H2. nin H2. ee. nin H10. elim H11. app H8.
  right. nin H8. nin H8. ee. exists x; exists x0. ee. app H5. app H5. am.
  ir. nin (H4 _ _ (H0 _ H12) (H0 _ (H5 _ H8))). au. nin (equal_or_not x z).
  rw H14. left. order_tac. ap  (H0 _ H12).
  nin (H4 _ _  (H0 _ (H5 _ H9)) (H0 _ H12)). au. nin (equal_or_not z x0).
  rw H16. right. order_tac. order_tac. assert (inc z v). uf v.
```

```
  app intersection2_inc. uf interval_cc. Ztac. ee. ufi v H8.
  cp (intersection2_second H8). ufi interval_cc H17. Ztac.
  change (gle r x z) in H13. ee; order_tac. ufi v H9.
  cp (intersection2_second H9). ufi interval_cc H17. Ztac.
  change (gle r z x0) in H15. ee; order_tac. app H11.
Qed.

Lemma Exercise1_21f: forall r a b u, total_order r ->
  sub u (substrate r) ->
  inc a u -> inc b u -> glt r a b -> without_gaps (induced_order r u) ->
  without_gaps (induced_order r (intersection2 u (interval_cc r a b))).
Proof. ir.  app  Exercise1_21e. exists a. exists b. nin H. ee.
  app intersection2_inc. uf interval_cc. Ztac.  nin H3. ee.  order_tac.
  order_tac. am.  app intersection2_inc. uf interval_cc. Ztac.  nin H3. ee.
  am. order_tac. order_tac. am.
Qed.
```

The union of two scattered intervals $[x,y]$ and $[y,z]$ is scattered. Proof by contradiction. Let u be a subset without gaps of the union, consider the intersection u_1 and u_2 with the two intervals. They are not without gaps.

```
Lemma Exercise1_21g : forall r x y z, total_order r ->
  gle r  x y ->  gle r y z->
  scattered (induced_order r (interval_cc r x y)) ->
  scattered (induced_order r (interval_cc r y z)) ->
  scattered (induced_order r (interval_cc r x z)).
Proof. ir. assert (gle r x z). nin H. order_tac.
  assert (Ha: order r). nin H; am.
  assert (Hb:sub (interval_cc r x z) (substrate r)). uf interval_cc. app Z_sub.
  assert (Hc:sub (interval_cc r x y) (substrate r)). uf interval_cc. app Z_sub.
  assert (Hd:sub (interval_cc r y z) (substrate r)). uf interval_cc. app Z_sub.
  rwii Exercise1_21a H2. rwii Exercise1_21a H3. rww Exercise1_21a.
  ir. assert (sub u (substrate r)). apply sub_trans with  (interval_cc r x z).
  am. am.
  nin (p_or_not_p (forall a b : Set, inc a u -> inc b u -> a = b)).
  rww Exercise1_21d. au. rwi (Exercise1_21c H H6) H7.
  nin (excluded_middle H7). nin H8.
  set (u1 := intersection2 u (interval_cc r x y)).
  assert (sub u1 (interval_cc r x y)). uf u1. app intersection2sub_second.
  set (u2 := intersection2 u (interval_cc r y z)).
  assert (sub u2 (interval_cc r y z)). uf u2. app intersection2sub_second.
  cp (H2 _ H9); cp (H3 _ H10).
  nin (p_or_not_p (exists x2, exists y0 , inc x2 u1 & inc y0 u1 & glt r x2 y0)).
  red. ir. elim H11. uf u1. app Exercise1_21e.
  nin (p_or_not_p (exists x2, exists y0 , inc x2 u2 & inc y0 u2 & glt r x2 y0)).
  red. ir. elim H12. uf u2. app Exercise1_21e.

  assert (He:forall t, inc t u -> gle r t y -> inc t u1). ir. uf u1.
  app intersection2_inc. uf interval_cc. Ztac. ee. cp (H5 _ H15).
  ufi interval_cc H17. Ztac. ee. am. am.
  assert (Hf:forall t, inc t u -> gle r y t -> inc t u2). ir. uf u2.
  app intersection2_inc. uf interval_cc. Ztac. ee. am. cp (H5 _ H15).
  ufi interval_cc H17. Ztac. ee. am.
  red. ir. red in H15. ee. nin H16; nin H16. nin (H17 _  _ H16). ee.
  nin (related_induced_order4 H18).  nin (related_induced_order2 H18).
  nin (related_induced_order4 H19).  nin (related_induced_order2 H19).
```

```
  nin H. assert (inc y (substrate r)). order_tac.
  nin (H28 _ _ (H6 _ H21) H29).
  assert (inc x2 u1). app He. order_tac. assert (inc x4 u1). app He. elim H13.
  exists x2; exists x4. eee. split;am.
  change (gle r y x4) in H30. assert (inc x4 u2). app Hf.
  assert (inc x3 u2). app Hf. order_tac.
  elim H14.  exists x4; exists x3. eee. split;am.
Qed.
```

Let's show that we have an equivalence relation. Symmetry is true by definition; reflexivity is a consequence of the fact that singletons are scattered.

```
Lemma Exercise1_21h : forall r, total_order r ->
  equivalence_re (scattered_rel r) (substrate r).
Proof. ir. cp H. nin H.
  split. red. ee. red.
  uf scattered_rel. ir. nin H2; au.
  assert (Ha: forall a b x y, gle r x a -> gle r b y ->
    sub (interval_cc r a b) (interval_cc r x y)).
  red. uf interval_cc. ir.  Ztac. clear H4.  aw. Ztac. ee. order_tac. order_tac.
  assert (Hb: forall a b x y, gle r x a -> gle r b y ->
    sub (interval_cc r a b) (substrate (induced_order r (interval_cc r x y)))).
  ir. aw. app Ha. uf interval_cc.  app Z_sub.
  red. uf scattered_rel. ir. nin H2; nin H3. left. ee. order_tac.
  app (Exercise1_21g H0 H2 H3 H5 H4). ee.
  assert (inc x (substrate r)). order_tac. assert (inc z (substrate r)).
  order_tac.  nin (H1 _ _ H6 H7). left. split. am. assert (gle r x x).
  order_tac. cp (Hb _ _ _ _ H9 H3). cp (Exercise1_20a H10 H5).
  wrii induced_trans H11. app Ha. uf interval_cc.  app Z_sub.
  right. split. am. assert (gle r z z). order_tac.
  cp (Hb _ _ _ _ H9 H2). cp (Exercise1_20a H10 H4).
  wrii induced_trans H11. app Ha. uf interval_cc.  app Z_sub.
  ee. assert (inc x (substrate r)). order_tac. assert (inc z (substrate r)).
  order_tac.  nin (H1 _ _ H6 H7). left. split. am. assert (gle r z z).
  order_tac. cp (Hb _ _ _ _ H2 H9). cp (Exercise1_20a H10 H4).
  wrii induced_trans H11. app Ha. uf interval_cc.  app Z_sub.
  right. split. am. assert (gle r x x). order_tac.
  cp (Hb _ _ _ _ H3 H9). cp (Exercise1_20a H10 H5).
  wrii induced_trans H11. app Ha. uf interval_cc.  app Z_sub.
  right. ee. order_tac.  app (Exercise1_21g H0 H3 H2 H4 H5).
  red. ir. uf scattered_rel. app iff_eq. ir. left.
  set (int := interval_cc r y y). assert (forall t, inc t int -> t = y).
  ir. ufi int H3. ufi interval_cc H3. Ztac. ee. order_tac.
  assert (sub int (substrate r)). red. ir. rww (H3 _ H4).
  assert (gle r y y). order_tac. ee. am. rww Exercise1_21a. ir.
  rww Exercise1_21d. left. ir. rw (H3 _ (H6 _ H7)). rww (H3 _ (H6 _ H8)).
  apply sub_trans with int. am. am.
  ir. nin H2;ee;order_tac.
Qed.
```

We have now an equivalence on our set.

```
Lemma Exercise1_21i : forall r, total_order r ->
  is_equivalence (scattered_equiv r).
Proof. ir. assert (is_graph (scattered_equiv r)).
  uf scattered_equiv. app is_graph_graph_on.
```

```
  cp (Exercise1_21h H).
  assert (is_graph_of  (scattered_equiv r) (scattered_rel r)).
  uf scattered_equiv. app equivalence_has_graph0. nin H1.
 app (equivalence_if_has_graph2 H0 H2 H1).
Qed.

Lemma Exercise1_21j : forall r, total_order r ->
  substrate (scattered_equiv r) = substrate r.
Proof. ir.  app extensionality.  uf scattered_equiv. app substrate_graph_on.
  red. ir. assert (scattered_rel r x x). cp ( Exercise1_21h H).
  red in H1. ee. wrr H2.  cp (Exercise1_21i H).
  assert (related (scattered_equiv r) x x). uf scattered_equiv.
  rww related_graph_on1. eee. substr_tac.
Qed.
```

We simplify a bit the definition of being related by this relation. We first consider a mix of 21a and 21d.

```
Definition scattered_aux r x y :=
  gle r x y &
  (forall u, sub u (interval_cc r x y) ->
    ((forall a b, inc a u -> inc b u -> a = b)
      \/ (exists a, exists b, inc a u & inc b u & glt r a b &
        (forall z, inc z u -> gle r z a \/ gle r b z)))).

Lemma Exercise1_21k : forall r x y, total_order r ->
  gle r  x y ->
  scattered (induced_order r (interval_cc r x y)) =
  (forall u, sub u (interval_cc r x y) ->
    ((forall a b, inc a u -> inc b u -> a = b)
      \/ (exists a, exists b, inc a u & inc b u & glt r a b &
        (forall z, inc z u -> gle r z a \/ gle r b z)))).
Proof. ir. set (i:=interval_cc r x y). assert (sub i (substrate r)).
  uf i. uf interval_cc. app Z_sub. rww Exercise1_21a.
  assert (forall u, sub u i -> sub u (substrate r)). ir.
  ap (@sub_trans _ _ _ H2 H1). app iff_eq. ir. wr Exercise1_21d. app H3. am.
  app H2. ir. rww Exercise1_21d. app (H3 _ H4). app H2. nin H. am.
Qed.

Lemma Exercise1_21l : forall r x y, total_order r ->
  related (scattered_equiv r) x y=
    (scattered_aux r x y \/ scattered_aux r y x).
Proof. ir. uf scattered_aux. uf scattered_equiv. rw related_graph_on1.
  ap iff_eq. ir. ee. nin H2. ee. left. ee. am. rwii Exercise1_21k H3.
  right. ee. am. rwii Exercise1_21k H3.
  ir. nin H0. ee. nin H. order_tac. nin H; order_tac. left. ee. am.
  rww Exercise1_21k. ee. nin H. order_tac. nin H; order_tac. right. ee. am.
  rww Exercise1_21k.
Qed.
```

Let's show weak compatibility. Assume that x and x' and related, and $x \leq y$. We want to find y such that $x' \leq y'$ and y is related to y'. If $x' \leq y$ we may choose $y = y'$. Otherwise we have $x \leq y \leq x'$, and we chose $y = x'$. The interval $[y, x']$ is scattered, as a subset of a scattered set.

```
Lemma Exercise1_21m : forall r, total_order r ->
```

```
  weak_order_compatibility r (scattered_equiv r).
Proof. ir. red. ee. nin H. app order_is_preorder. app Exercise1_21i.
  rww Exercise1_21j.  ir. cp H. nin H.
  assert (inc y (substrate r)). order_tac. cp (Exercise1_21i H2).
  assert (inc y (substrate (scattered_equiv r))).  rww Exercise1_21j.
  assert (related (scattered_equiv r) y y). equiv_tac.
  rwi (Exercise1_21l x x' H2) H1. nin H1. red in H1. ee.
  assert (inc x' (substrate r)). order_tac. nin (H3 _ _ H4 H9).
  exists x'. ee. rww Exercise1_21l. left. split. am. ir. app H8.
  red. ir. cp (H11 _ H12).  uf interval_cc. ufi interval_cc H13. Ztac.
  clear H13. ee. Ztac. ee. order_tac. am. order_tac. exists y.
  ee. am. am. exists y. ee. am. nin H1. ee. order_tac.
Qed.
```

Let's show that equivalence classes are scattered. Let X be a subset of an equivalence class, assumed without gaps. It has two elements a and b such that $a < b$; and for every $c < d$, there is e such that $c < e < d$. These elements a and b are related so that the interval $[a,b]$ is scattered. Let Y be the intersection of X and the interval $[a,b]$. It is not without gaps, but has two comparable elements, namely a and b, hence there exists c, d such that $]c,d[\cap$Y is empty. But there is an $e \in]c,d[\cap$X. This element is clearly in $[a,b]$ hence in Y, absurd.

```
Lemma Exercise1_21n: forall r x, total_order r -> inc x (substrate r) ->
  scattered (induced_order r (class (scattered_equiv r) x)).
Proof. ir. cp (Exercise1_21i H). cp (Exercise1_21j H). cp H. nin H.
  assert (sub (class (scattered_equiv r) x) (substrate r)).
  wr H2. app sub_class_substrate.  red. ee. fprops. ir. red. ir.
  awii H6. rwii induced_order_trans H7. nin H7. ee.
  assert (forall u, inc u x0 -> related (scattered_equiv r) u x). ir.
  cp (H6 _ H10). bwi H11. equiv_tac. am. nin H8. nin H8.
  nin (related_induced_order4 H8). cp (H10 _ H11); cp (H10 _ H12).
  assert (related (scattered_equiv r) x x2). equiv_tac.
  assert (related (scattered_equiv r) x1 x2). equiv_tac.
  clear H13; clear H14; clear H15. rwi Exercise1_21l H16. nin H16.
  nin H13.
  set (u:=intersection2 x0 (interval_cc r x1 x2)).
  assert (sub u (interval_cc r x1 x2)).  uf u. app intersection2sub_second.
  nin (H14 _ H15).  assert (inc x1 u). uf u. app intersection2_inc.
  uf interval_cc. Ztac. ee. order_tac. order_tac. am.
  assert (inc x2 u).  uf u. app intersection2_inc.
  uf interval_cc. Ztac. ee. am. order_tac. order_tac. nin H8. elim H19. app H16.
  nin H16. nin H16.  ee.
  assert (glt (induced_order r x0) x3 x4). nin H18. split. aw.
  ufi u H16. inter2tac. ufi u H17. inter2tac. am. nin (H9 _ _ H20). ee.
  nin (related_induced_order4 H21).  cp (related_induced_order2 H21).
  cp (related_induced_order2 H22).
  assert (inc x5 u). uf u. app intersection2_inc. uf interval_cc. Ztac. ee.
  ufi u H16. cp (intersection2_second H16). ufi interval_cc H27. Ztac. ee.
  nin H25. order_tac. ufi u H17. cp (intersection2_second H17).
  ufi interval_cc H27. Ztac. ee. nin H26. order_tac. nin (H19 _ H27).
  order_tac. order_tac. nin H13. cp (related_induced_order2 H8). order_tac. am.
Qed.
```

Since the equivalence S is weakly compatible with the order, it induces a preorder on the quotient. We show here that it is an order (cf. Exercise 1.2).

```
Lemma Exercise1_21o: forall r, total_order r ->
```

```
  quotient_order_axiom r (scattered_equiv r).
Proof. red. ir. rwii Exercise1_21l H2. cp H. nin H. nin H2.
  rww Exercise1_21l. left. nin H2. split. am. ir. app H5. ufi interval_cc H6.
  uf interval_cc. red. ir. cp (H6 _ H7). Ztac. clear H8. Ztac. ee. am.
  order_tac. nin H2.  assert (gle r y x). order_tac. assert (x = y). order_tac.
  rw H7. cp (Exercise1_21i H3). cp (Exercise1_21j H3).
  assert (inc y (substrate r)). order_tac. wri H9 H10. equiv_tac.
Qed.

Lemma Exercise1_21p: forall r, total_order r ->
  order (quotient_order r (scattered_equiv r)).
Proof. ir. app Exercise1_2d. ap (Exercise1_21i H). nin H. am.
  ap (Exercise1_21j H). ap  (Exercise1_21o H).
Qed.
```

Another extension to Exercise 1.2.

```
Lemma Exercise1_2j: forall r s, weak_order_compatibility r s->
  quotient_order_axiom r s -> total_order r ->
  let r' := (quotient_order r s) in
    forall x y, inc x (substrate r ) -> inc y (substrate r) ->
      ((gle r x y -> gle r' (class s x) (class s y))
      & (glt r' (class s x) (class s y)) ->glt r x y).
Proof. ir. assert (Ha:=Exercise1_2h H H0 H1).
  rename H1 into Ht. red in H. red in H0.
  assert (forall x y, inc x (substrate r ) -> inc y (substrate r) ->
    gle r x y -> gle r' (class s x) (class s y)).
  uf r'. ir. rw quotient_order_pr. uf quotient_order_r. nin H. nin H6. nin H7.
  wri H7 H1; wri H7 H4. ir. ee. gprops. gprops.
  ir. assert (inc x1 (substrate r)). wr H7.
  ap((sub_class_substrate H6 (x:= x0)) _ H9).
  nin Ht. rwi H7 H4. nin (H12 _ _ H10 H4). exists y0. ee. app inc_itself_class.
  rww H7. am. change (gle r y0 x1) in  H13.
  assert (related s x0 x1). bwi H9. am. am.
  cp (H0 _ _ _ H5 H13 H14). exists x1. ee. wrr (related_class_eq1 H6 H15).
  order_tac.
  split. app H1. ir. nin H4. split.
  ir. ee. nin Ht. nin (H10 _ _ H2 H3). am. change (gle r y x) in H11.
  cp (H1 _ _ H3 H2 H11). elim H5. nin Ha. order_tac. dneg. rww H6.
Qed.
```

Let's show that the quotient ordered set is either without gaps or a small set (empty or containing a single element). All we need to show is that if X < Y in the quotient, there is Z such that X < Z < Y. Let x and y be the representatives of X and Y. They are not related by the equivalence S, hence $[x, y]$ is not scattered. We use contradiction, assume there is a set U with at least two elements, such that no interval is empty. Let $a \in$ U. We have $x \leq a \leq y$, so that classes are in the same order. Let A be the class of a, so that X ≤ A ≤ Y. By assumption, one ≤ is equality. This implies $a \in$ X or $a \in$ Y.

```
Lemma Exercise1_21q: forall r, total_order r ->
  let r' := quotient_order r  (scattered_equiv r) in
    small_set (substrate r') \/ without_gaps r'.
Proof. ir. assert (Ha:=Exercise1_21m H).  assert (Hb:=Exercise1_21o H).
  cp (Exercise1_2g Ha Hb H). cp (Exercise1_2h Ha Hb H).
  set (s:= scattered_equiv r) in *.
```

```
assert (Hc:forall a b, gle r a b -> gle r' (class s a) (class s b)).
ir. cp H; nin H. assert (inc a (substrate r)). order_tac.
assert (inc b (substrate r)). order_tac. nin (Exercise1_2j Ha Hb H3 H5 H6).
app H7.
simpl in H0. fold r' in H1. fold r' in H0.
nin (p_or_not_p (small_set (substrate r'))). au. right. red. ee.
nin H1; am. app exists_proof2. dneg. red. ir. nin H1.
nin (equal_or_not u v). am. nin (H6 _ _ H4 H5). elim (H3 u v). split; am.
elim (H3 v u). split. am. au. clear H2. ir. assert (Hf:=H2).
red in Ha. ee.
assert (substrate r' = quotient s). uf r'. rww substrate_quotient_order.
nin H2. rwi H0 H2. ee.
assert (~ (related s (rep x) (rep y))). dneg. rwii related_rep_rep H11.
app exists_proof. dneg. uf s. uf scattered_equiv. rw related_graph_on1.
ee. order_tac. order_tac. uf scattered_rel. left. ee. am.
assert (sub (interval_cc r (rep x) (rep y)) (substrate r)). uf interval_cc.
app Z_sub. assert (order r). nin H; am. rww Exercise1_21a.
ir. red. ir. red in H16. ee. nin H17. nin H17.
assert (Hx:class s (rep x) = x). app class_rep.
assert (Hy:class s (rep y) = y). app class_rep.
assert (forall a, inc a u -> inc a x \/ inc a y).
ir. cp (H15 _ H19). ufi interval_cc H20. Ztac. ee.
cp (Hc _ _ H22). rwi Hx H24. nin (equal_or_not x (class s a)).
rw H25. left. app inc_itself_class. ue.
assert (glt r' x (class s a)). split;am.
cp (Hc _ _ H23). rwi Hy H27. nin (equal_or_not (class s a) y).
wr H28. right. app inc_itself_class. ue.
assert (glt r' (class s a) y). split;am. elim (H12 (class s a)). au.
```

Let X_1 and X_2 be the intersections of U with X and Y. These sets have at most one element, for otherwise, we could find an empty interval $]a,b[$. This interval has a point c in U, which is in X or in Y. If we consider X_1, $c \leq b$, $b \in X$ and $c \in Y$ would imply $Y \leq X$, absurd.

```
cp (class_rep H4 H2). cp (class_rep H4 H9).
assert (inc (rep x) (substrate r)). order_tac.
assert (inc (rep y) (substrate r)). order_tac.
assert (Hu:forall a, inc a x -> x = class s a). ir. app is_class_pr.
assert (Hv:forall a, inc a y -> y = class s a). ir. app is_class_pr.
cp (Exercise1_21n H H22). ufi s H20. rwi H20 H24. clear H20. clear H22.
cp (Exercise1_21n H H23). ufi s H21. rwi H21 H20. clear H21. clear H23.
rwi Exercise1_21a H24. rwi Exercise1_21a H20.
set (u1:= intersection2 u x). assert (sub u1 x). uf u1.
app intersection2sub_second. cp (H24 _ H21).
assert (sub u1 (substrate r)). red. ir. ufi u1 H23.
cp (intersection2_first H23). cp (H15 _ H25).  ufi interval_cc H26. Ztac. am.
rwi (Exercise1_21d H H23) H22. clear H23. clear H21.
set (u2:= intersection2 u y). assert (sub u2 y). uf u2.
app intersection2sub_second. cp (H20 _ H21).
assert (sub u2 (substrate r)). red. ir. ufi u2 H25.
cp (intersection2_first H25). cp (H15 _ H26).  ufi interval_cc H27. Ztac. am.
rwi (Exercise1_21d H H25) H23. clear H25. clear H21.
assert (~(exists a, exists b, inc a u1 & inc b u1 & glt r a b)).
red. ir. nin H21. nin H21. ee. nin H22. nin H26. elim H27. app H22.
nin H22. nin H22. ee.
assert (glt (induced_order r u) x4 x5). nin H28. split. aw. ufi u1 H22.
inter2tac. ufi u1 H27. inter2tac. am. nin (H18 _ _ H30). ee.
```

```
  nin (related_induced_order4 H31).  cp (related_induced_order2 H31).
  nin (related_induced_order4 H32).  cp (related_induced_order2 H32).
  assert (inc x6 u1). uf u1. app intersection2_inc.
  nin (H19 _ H36). am. nin H38. cp (Hc _ _ H38). wri (Hv _ H39) H41.
  ufi u1 H27. cp (intersection2_second H27). wri (Hu _ H42) H41.
  nin H1. order_tac. nin (H29 _ H39). order_tac. order_tac.
  assert (~(exists a, exists b, inc a u2 & inc b u2 & glt r a b)).
  red. ir. nin H25. nin H25. ee. nin H23. nin H27. elim H28. app H23.
  nin H23. nin H23. ee.
  assert (glt (induced_order r u) x4 x5). nin H29. split. aw. ufi u2 H23.
  inter2tac. ufi u2 H28. inter2tac. am. nin (H18 _ _ H31). ee.
  nin (related_induced_order4 H32).  cp (related_induced_order2 H32).
  nin (related_induced_order4 H33).  cp (related_induced_order2 H33).
  assert (inc x6 u2). uf u2. app intersection2_inc.
  nin (H19 _ H37). nin H36. cp (Hc _ _ H36). wri (Hu _ H40) H42.
  ufi u2 H23. cp (intersection2_second H23). wri (Hv _ H43) H42.
  nin H1. order_tac. am.  nin (H30 _ H40). order_tac. order_tac.
  nin (H18 _ _ H17). ee.
```

Now U has three points, and is the union of two sets with at most one point, absurd.

```
  nin (related_induced_order4 H26). nin (related_induced_order4 H27).
  cp (related_induced_order2 H26). cp (related_induced_order2 H27).
  nin (H19 _ H28). assert (inc x0 u1). uf u1. app intersection2_inc.
  nin (H19 _ H29). assert (inc x2 u1). uf u1. app intersection2_inc.
  elim H21. exists x0; exists x2; ee; am.
  nin (H19 _ H31). assert (inc x1 u1). uf u1. app intersection2_inc.
  elim H21. exists x0; exists x1; ee; tv. nin H32. nin H. order_tac.
  assert (inc x2 u2). uf u2. app intersection2_inc.
  assert (inc x1 u2). uf u2. app intersection2_inc.
  elim H25. exists x2; exists x1; ee; tv.
  assert (inc x0 u2). uf u2. app intersection2_inc.
  nin (H19 _ H29). assert (inc x2 u1). uf u1. app intersection2_inc.
  nin (H19 _ H31). assert (inc x1 u1). uf u1. app intersection2_inc.
  elim H21. exists x2; exists x1; ee; tv.
  assert (inc x1 u2). uf u2. app intersection2_inc.
  elim H25. exists x0; exists x1; ee; tv. nin H32. nin H. order_tac.
  assert (inc x2 u2). uf u2. app intersection2_inc.
  elim H25. exists x0; exists x2; ee; tv.
  am. wr H5. red. ir. app (inc_in_quotient_substrate H4 H21 H9).
  am. wr H5. red. ir. app (inc_in_quotient_substrate H4 H21 H2).
Qed.
```

¶ 22. *(a) Let* E *be an ordered set; A subset* U *of* E *is said to be* open *if for each* $x \in U$, U *contains the interval* $[x, \rightarrow [$. *An open set* U *is said to be* regular *if there exists no open set* $V \supset U$, *distinct from* U *such that* U *is cofinal in* V. *Show that every open set* U *is cofinal in exactly one regular open set* $\bar{U}$. *The mapping* $U \rightarrow \bar{U}$ *is increasing. If* U, V *are two open sets such that* $U \cap V = \emptyset$, *then also* $\bar{U} \cap \bar{V} = \emptyset$.

We start by showing the the union and intersection of open sets is open.

```
Definition open_o r u:=
```

```
  sub u (substrate r) & forall x y, inc x u -> gle r x y -> inc y u.
Definition open_r r u:=
  open_o r u & forall v, open_o r v -> sub u v ->
    cofinal_set (induced_order r v) u
    -> u = v.

Lemma  Exercise1_22a: forall r u1 u2, order r ->
  open_o r u1 -> open_o r u2 -> open_o r (union2 u1 u2).
Proof. ir. red. nin H0. nin H1. split. red. ir. nin (union2_or H4).
  app H0. app H1.
  ir. nin (union2_or H4). app union2_first. app (H2 _ _ H6 H5).
  app union2_second. app (H3 _ _ H6 H5).
Qed.

Lemma Exercise1_22b: forall r u, order r -> nonempty u ->
  (forall x, inc x u -> open_o r x) ->
  open_o r (intersection u).
Proof. ir. split. red. ir. nin H0. cp (H1 _ H0).  nin H3. app H3.
  app (intersection_forall H2 H0).
  ir. app intersection_inc. ir. nin (H1 _ H4).
  assert (inc x y0).  app (intersection_forall H2 H4).
  ap (H6 _ _ H7 H3).
Qed.

Lemma Exercise1_22c: forall r u, order r ->
  (forall x, inc x u -> open_o r x) ->
  open_o r (union u).
Proof. ir. split. red. ir. nin (union_exists H1).  nin H2. nin (H0 _ H3).
  app H4. ir.  nin (union_exists H1). nin H3. nin (H0 _ H4).
  cp (H6 _ _ H3 H2). union_tac.
Qed.
```

We show uniqueness of $\bar{U}$. If U is cofinal in U_1 and U_2, if $U_3 = U_1 \cup U_2$ regularity of U_1 shows $U_3 = U_1$. Similarly $U_3 = U_2$.

```
Lemma Exercise1_22d: forall r x u1 u2, order r ->
  open_o r x -> open_r r u1 -> open_r r u2 ->
  sub x u1 -> sub x u2 ->
  cofinal_set (induced_order r u1) x -> cofinal_set (induced_order r u2) x
  -> u1 = u2.
Proof. ir. set (u3:= union2 u1 u2).
  assert (open_o r u3). uf u3. app Exercise1_22a. nin H1. ee; am.
  nin H2. ee;am. nin H1. assert (sub u1 u3). uf u3. red. ir. inter2tac.
  assert (cofinal_set (induced_order r u3) u1). red.
  assert (sub u3 (substrate r)). nin H7. am. aw. ee. am.
  ir. assert (exists a, inc a x & gle r x0 a). ufi u3 H11.
  nin (union2_or H11). nin H5.  awi H13. nin (H13 _ H12).  exists x1. ee. am.
  ap (related_induced_order1 H15). am. nin H1; am.
  nin H6. awi H13. nin (H13 _ H12).  exists x1. ee. am.
  ap (related_induced_order1 H15). am. nin H2. nin H2; am. nin H12. nin H12.
  exists x1. ee. app H3. aw. uf u3. app union2_first. app H3.
  cp (H8 _  H7 H9  H10).
  nin H0. assert (sub u2 u3). uf u3. red. ir. inter2tac.
  assert (cofinal_set (induced_order r u3) u2). red.
  assert (sub u3 (substrate r)). nin H7. am. aw. ee. am.
  ir. wri H11 H15. nin H5. awi H16. nin (H16 _ H15). nin H17.
  cp (related_induced_order1 H18). exists x1. ee.  app H4. aw. app H9.
```

```
  app H13. app H4. am. nin H1;am. nin H2. rw H11.  rww (H15 _ H7 H13 H14).
Qed.
```

Let $\bar{U}$ be the union of all open sets V containing U in which U is cofinal (since V = U is possible, we have $U \subset \bar{U}$). U is cofinal in $\bar{U}$.

```
Definition bar1_22 r u :=
  union (Zo (powerset(substrate r))
    (fun z => open_o r z &  cofinal_set (induced_order r z) u)).

Lemma Exercise1_22e: forall r u,
  order r ->  open_o r u ->  sub u (bar1_22 r u).
Proof. ir. cp H0; nin H0. red. ir. uf bar1_22. apply union_inc with u. am.
  Ztac. app powerset_inc. ee. am. split. aw. fprops. aw. ir. exists x0.
  ee. am. aw. order_tac. app H0.
Qed.

Lemma Exercise1_22f: forall r u,
  order r ->  open_o r u ->  sub (bar1_22 r u) (substrate r).
Proof. ir. uf bar1_22. red. ir.
  nin (union_exists H1). ee. Ztac. rwi powerset_inc_rw H4. app H4.
Qed.

Lemma Exercise1_22g: forall r u,
  order r ->  open_o r u ->
  cofinal_set (induced_order r (bar1_22 r u)) u.
Proof. ir. cp (Exercise1_22e H H0). cp (Exercise1_22f H H0).
  split. aw. aw. ir. ufi bar1_22 H3. nin (union_exists H3). nin H4. Ztac. ee.
  rwi powerset_inc_rw H6. nin H8. awii H9. nin (H9 _ H4). exists x1.
  ee. am. aw. ap (related_induced_order1 H11).  app H1.
Qed.
```

Consider $x \in E$, such that whenever $x \leq y$, y is bounded by an element of U. Then $x \in \bar{U}$. It suffices to consider $U \cup [x, \rightarrow [$.

```
Exercise1_22h: forall r u x,
  order r ->  open_o r u -> inc x (substrate r) ->
  (forall y, gle r x y -> exists z, inc z u &  gle r y z)
  -> inc x (bar1_22 r u).
Proof. ir.  assert (Ha:=Exercise1_22f H H0).
  set (t:=union2 (bar1_22 r u) (Zo (substrate r) (fun z=> gle r x z))).
  assert (sub t (substrate r)). red. uf t. ir. nin (union2_or H3).
  app Ha. Ztac. am.
  uf bar1_22. apply union_inc with t. uf t. app union2_second. Ztac.
  order_tac. Ztac. app powerset_inc. split.  uf t.
  app Exercise1_22a. red. split. am.
  assert (open_o r (bar1_22 r u)). uf bar1_22. app  Exercise1_22c.
  ir. Ztac. eee. nin H4. am.
  red. split. app Z_sub. ir. Ztac. clear H4. Ztac.
  order_tac. order_tac. split. aw. uf t. red. ir. app union2_first.
  app  (Exercise1_22e H H0). aw. ir. ufi t H4. nin (union2_or H4).
  cp (Exercise1_22g H H0). nin H6. awi H7. cp (H7 _ H5). nin H8.
  ee. exists x1. ee. am. aw. ap (related_induced_order1 H9). uf t.
  app union2_first. nin (related_induced_order3 H9). am. am.  am.
  Ztac. nin (H2 _ H7). exists x1. eee. aw. uf t.
  uf t. app union2_first. app (Exercise1_22e H H0).
Qed.
```

Assume $\bar{U}$ cofinal in V. If $x \in V$ and $x \leq y$, then $y \in V$ and is bounded by $z \in \bar{U}$, which in bounded by an element of U, hence $x \in \bar{U}$, and $\bar{U}$ is a regular open set.

```
Lemma Exercise1_22i: forall r u,
  order r ->  open_o r u ->
  open_r r (bar1_22 r u).
Proof. ir. assert (sub (bar1_22 r u) (substrate r)). nin H0. uf bar1_22.
  red. ir. nin (union_exists H2). ee. Ztac. rwi powerset_inc_rw H5. app H5.
  assert (open_o r (bar1_22 r u)). uf bar1_22. app  Exercise1_22c.
  ir. Ztac. eee.
  split.  am. ir. app extensionality. red. ir.
  app Exercise1_22h. nin H3. app H3. ir. nin H5. awi H8. nin H3.
  cp (H9 _ _ H6 H7). nin (H8 _ H10). nin H11.
  cp (related_induced_order1 H12). nin (Exercise1_22g H H0). awi H15.
  nin (H15 _ H11). nin H16. cp (related_induced_order1 H17). ex_tac. order_tac.
  am. am. am. nin H3. am.
Qed.
```

Assume $U \subset V$, and $x \in \bar{U}$, and $x \leq y$. Then $y \in \bar{U}$, hence is bounded by an element of U (thus V), and x is in $\bar{V}$. Hence $\bar{U} \subset \bar{V}$.

```
Lemma Exercise1_22j: forall r u v,
  order r ->  open_o r u -> open_o r v -> sub u v ->
  sub (bar1_22 r u)  (bar1_22 r v).
Proof. ir. red. ir. app Exercise1_22h. app (Exercise1_22f H H0).
  ir. assert (inc y (bar1_22 r u)). nin (Exercise1_22i H H0). nin H5.
  app (H7 _ _ H3 H4). nin (Exercise1_22g H H0).
  cp (Exercise1_22f H H0). awii H7.  nin (H7 _ H5). nin H9.
  exists x0. ee. app H2. ap (related_induced_order1 H10).
Qed.
```

Assume that U and V are open sets. Consider an element $a \in \bar{U} \cap \bar{V}$, say $a \in K_1$ and $a \in K_2$. There is $x \in U$, with $a \leq x$. Since K_2 is open, we have $x \in K_2$. Thus, there is $y \in V$ such that $x \leq y$. Since U is open, we have $y \in U \cap V$. Thus, if $U \cap V = \emptyset$ then $\bar{U} \cap \bar{V} = \emptyset$.

```
Lemma Exercise1_22k: forall r u v,
  order r ->  open_o r u -> open_o r v ->  intersection2 u v = emptyset ->
  intersection2 (bar1_22 r u)  (bar1_22 r v) = emptyset.
Proof. ir.  app is_emptyset. ir. red. ir. nin (intersection2_both H3).
  ufi bar1_22 H4. nin (union_exists H4). nin H6. Ztac. clear H7. ee.
  nin H9. nin H7.  awii H10. nin (H10 _ H6). nin H12.
  cp (related_induced_order1 H13).
  ufi bar1_22 H5. nin (union_exists H5). nin H15. Ztac. ee. nin H18.
  cp (H20 _ _ H15 H14). nin H19.  awii H22. nin (H22 _ H21). nin H23.
  cp (related_induced_order1 H24). nin H0.  cp (H26 _ _ H12 H25).
  elim (emptyset_pr (x:=x2)). wr H2. app intersection2_inc.
Qed.
```

(b) Show that the set R(E) *of regular open sets of* E*, ordered by inclusion, is a complete Boolean lattice (Exercise 17). For* R(E) *to consist of two elements, it is necessary and sufficient that* E *should be non-empty and* right directed*.*

Let's start with the last point. The set R(E) has a least and a greatest element, namely $\emptyset$ and E.

```
Lemma Exercise1_22m: forall r, order r -> open_r r emptyset.
Proof. ir. split. split.  app sub_emptyset_any. ir. elim (emptyset_pr H0).
  ir. nin H2. nin H0. awii H3.  nin (emptyset_dichot v). sy; am. nin H5.
  nin (H3 _ H5). nin H6. elim (emptyset_pr H6).
Qed.

Lemma Exercise1_22n: forall r, order r -> open_r r (substrate r).
Proof. ir. split. split.  fprops. ir.  order_tac. ir. nin H0.
  app extensionality.
Qed.
```

Let I_x denote the interval $[x, \rightarrow [$. Assume that our set is not right directed. This means that there exists x and y such that $I_x \cap I_y = \emptyset$. Let $U_x = \bar{I}_x$. We have $U_x \cap U_y = \emptyset$. These two sets are regular and nonempty (they contain x and y). Thus $U_x \neq U_y$.

```
Lemma Exercise1_22o: forall r, order r -> ~ (right_directed r) ->
  exists a, exists b, open_r r a & open_r r b & nonempty a & nonempty b& a <> b.
Proof. ir. set (i:= fun x => Zo (substrate r) (fun z => gle r x z)).
  assert (forall x, inc x (substrate r) -> open_o r (i x)).
  ir. split. uf i. app Z_sub.  uf i. ir.  Ztac. clear H2. Ztac.
  order_tac. order_tac.
  assert (forall x, inc x (substrate r) -> open_r r (bar1_22 r (i x))).
  ir.  app  Exercise1_22i. app H1.
  assert (forall x, inc x (substrate r) -> inc x (bar1_22 r (i x))).
  ir.  app Exercise1_22e. app H1. uf i. Ztac. order_tac.
  set (p := fun a b => inc a (substrate r) & inc b (substrate r) &
    intersection2 (i a) (i b) = emptyset).
  nin (p_or_not_p (exists a , exists b, p a b)). nin H4. nin H4. ufi p H4. ee.
  exists (bar1_22 r (i x)). exists (bar1_22 r (i x0)).
  cp (H1 _ H4).  cp (H1 _ H5). cp (Exercise1_22k H H7 H8 H6).
  ee. app H2. app H2.  exists x. app H3. exists x0. app H3. red. ir.
  rwi H10 H9. cp (H3 _ H5). elim (emptyset_pr (x:=x0)). wr H9.
  app intersection2_inc.
  elim H0. rw right_directed_pr. ee. am. ir. app exists_proof.  dneg.
  exists x;exists y. uf p. eee. app is_emptyset.  ir. cp (H7 y0). dneg.
  ufi i H9. nin (intersection2_both H9). Ztac. clear H11. Ztac. eee.
Qed.
```

We have $\bar{X} = X$ whenever X is regular.

```
Lemma Exercise1_22p: forall r x, order r ->
  open_r r x -> x = (bar1_22 r x) .
Proof. ir. cp H0. nin H1. cp (Exercise1_22g H H1). nin (Exercise1_22i H H1).
  ap (H2 _ H4 (Exercise1_22e H H1) H3).
Qed.
```

Let's consider the claim "R(E) consists in two elements". A big part of the proof here is to show that this is equivalent to "R(E) contains only E and $\emptyset$". If E is not right directed, there are at least 3 regular sets (U_x, U_y and $\emptyset$).

```
Definition set_of_reg_open r := Zo (powerset (substrate r))
  (fun z => open_r r z).

Lemma Exercise1_22q: forall r, order r ->
  (exists a, exists b, a <> b &  set_of_reg_open r = doubleton a b) =
```

```
  (nonempty (substrate r) & (right_directed r)).
Proof. ir.  ap iff_eq. ir. nin H0; nin H0. nin H0.
  assert (inc x (set_of_reg_open r)). rw H1. fprops.
  assert (inc x0 (set_of_reg_open r)). rw H1. fprops.
  ufi set_of_reg_open H2. Ztac. clear H2. ufi set_of_reg_open H3. Ztac.
  rwi powerset_inc_rw H4. rwi powerset_inc_rw H2. split.
  nin (emptyset_dichot x).  nin (emptyset_dichot x0). elim H0. ue.
  nin H8. exists y. app H2. nin H7. exists y. app H4.
  nin (p_or_not_p (right_directed r)). am. cp (Exercise1_22o H H7).
  nin H8. nin H8. clear H3. ee.
  assert (inc x1 (set_of_reg_open r)). uf set_of_reg_open. Ztac.
  app powerset_inc. nin H3. nin H3. am.
  assert (inc x2 (set_of_reg_open r)). uf set_of_reg_open. Ztac.
  app powerset_inc. nin H8. nin H8. am.
  assert (inc emptyset  (set_of_reg_open r)). uf set_of_reg_open. Ztac.
  app powerset_inc. app sub_emptyset_any. app Exercise1_22m. rwi H1 H14.
  rwi H1 H12. rwi H1 H13. nin H10; nin H9.
  nin (doubleton_or H14). nin (doubleton_or H13). elim (emptyset_pr (x:=y)).
  rww H15. wrr H16. nin (doubleton_or H12). elim (emptyset_pr (x:=y0)).
  rww H15. ue.  elim H11. ue.  nin (doubleton_or H13). nin (doubleton_or H12).
  elim H11. ue.  elim (emptyset_pr (x:=y0)). rww H15. ue.
  elim (emptyset_pr (x:=y)). rww H15. ue.
```

Converse. We must show that a non-empty element of R(E) is E.

```
  ir. ee. exists emptyset. exists (substrate r). ee. red. ir. nin H0.
  wri H2 H0. nin H0. elim x. sy. set_extens. nin (doubleton_or H2).
  rw H3.  uf set_of_reg_open. Ztac.
  app powerset_inc. app sub_emptyset_any. app Exercise1_22m. rw H3.
  uf set_of_reg_open. Ztac. app powerset_inc. fprops.  app Exercise1_22n.
  nin (emptyset_dichot x). rw H3. fprops. nin H3. ufi set_of_reg_open H2.
  Ztac. clear H2. cut (x = substrate r). ir.  rw H2. fprops.
```

Assume the set E, right directed, $x \in U$, $y \in E$. We have $y \in \bar{U}$, since whenever $y \leq y'$, there is a upper bound z for x and y'. It is in U, since U is open. If $U \in R(E)$ then $U = \bar{U}$, hence $y \in U$.

```
  rwi powerset_inc_rw H4. app extensionality. red. ir.
  assert (inc x0  (bar1_22 r x)). app Exercise1_22h. nin H5. am.
  ir. rwi right_directed_pr H1. nin H1. assert (inc y (substrate r)).
  app H4. assert (inc y0 (substrate r)). order_tac. nin (H7 _ _ H8 H9).
  exists x1. ee. nin H5. nin H5. app (H14 _ _ H3 H11). am.
  rww (Exercise1_22p H H5).
Qed.
```

Consider now the first part. We consider some properties of the ordering induced by inclusion on R(E). We first show that E and $\emptyset$ are the greatest and least elements. This is a trivial consequence of the fact that these sets are in R(E)

```
Definition reg_open_order r :=
  inclusion_suborder (set_of_reg_open r).

Lemma Exercise1_22r: forall r u v, order r ->
  gle (reg_open_order r) u v =
  (open_r r u & open_r r v& sub u v).
```

```
Proof. ir. uf reg_open_order. aw. uf set_of_reg_open. ap iff_eq. ir. ee.
  clear H1. Ztac. am. Ztac. am. am. ir. ee. Ztac. nin H0. nin H0.
  app powerset_inc. Ztac. nin H1. nin H1. app powerset_inc. am.
Qed.

Lemma Exercise1_22s: forall r, order r ->
  greatest_element (reg_open_order r) (substrate r).
Proof. ir. cp (Exercise1_22n H).
  split. uf reg_open_order. aw. uf set_of_reg_open. Ztac. app powerset_inc.
  fprops. ir. rw Exercise1_22r. ufi reg_open_order H1. awi H1.
  ufi set_of_reg_open H1. Ztac. ee.  am. am. nin H3. nin H3. am. am.
Qed.

Lemma Exercise1_22t: forall r, order r ->
  least_element (reg_open_order r) (emptyset).
Proof. ir.  cp (Exercise1_22m H). split. uf reg_open_order. aw.
  uf set_of_reg_open. Ztac.  app powerset_inc. app sub_emptyset_any.
  ir. rw Exercise1_22r. ufi reg_open_order H1. awi H1. ufi set_of_reg_open H1.
  Ztac. ee. am. am.  app sub_emptyset_any. am.
Qed.
```

The function $U \mapsto \bar{U}$ is a closure. As a consequence, the bar of the union of a family of elements of R(E) is the least upper bound. This shows that the set is a complete lattice, thus is a lattice. The sup and inf of two elements are $\overline{U \cup V}$ and $\overline{U \cap V}$.

```
Lemma inf_pr2: forall r x y z,
  order r -> gle r z x  -> gle r z y ->
  (forall t, gle r t x -> gle r t y -> gle r t z) ->
  inf r x y = z.
Proof. ir. cp (greatest_lower_bound_doubleton H H0 H1 H2).
  sy. uf inf. app infimum_pr2. red. ir. nin (doubleton_or H4); rw H5; order_tac.
Qed.

Lemma Exercise1_22u: forall r u v, order r ->
  open_r r u -> open_r r v ->
  inf (reg_open_order r) u v = bar1_22 r (intersection2 u v).
Proof. ir. set (z := bar1_22 r (intersection2 u v)).
  assert (open_o r (intersection2 u v)). uf intersection2. app Exercise1_22b.
  app nonempty_doubleton. ir. nin H0; nin H1. nin (doubleton_or H2); ue.
  cp (Exercise1_22i H H2). cp (Exercise1_22e H H2).
  assert (gle (reg_open_order r) z u). uf z. rw Exercise1_22r. ee. am. am.
  cp (Exercise1_22p H H0). rw H5. app Exercise1_22j. wr H5. am. nin H0. am.
  wr H5. app intersection2sub_first. am.
  assert (gle (reg_open_order r) z v). uf z. rw Exercise1_22r. ee. am. am.
  cp (Exercise1_22p H H1). rw H6. app Exercise1_22j. wr H6. am. nin H1. am.
  wr H6. app intersection2sub_second. am.
  assert (forall t, gle (reg_open_order r) t u -> gle (reg_open_order r) t v
    -> gle (reg_open_order r) t z). ir. rwi  Exercise1_22r H7.
  rwi Exercise1_22r H8. ee. rw Exercise1_22r. ee. am. am.
  apply sub_trans with (intersection2 u v). red. ir. app intersection2_inc.
  app H12. app H10. am. am. am. am.
  assert (order (reg_open_order r)). uf reg_open_order. fprops.
  ap (inf_pr2 H8 H5 H6 H7).
Qed.

Lemma Exercise1_22v: forall r X, order r ->
```

```
  sub X (substrate (reg_open_order r)) ->
  least_upper_bound (reg_open_order r) X (bar1_22 r (union X)).
Proof. ir. assert (order (reg_open_order r)). uf reg_open_order. fprops.
  assert (substrate (reg_open_order r) = (set_of_reg_open r)).
  uf reg_open_order. aw. aw.
  assert (open_o r (union X)). app Exercise1_22c. ir. cp (H0 _ H3). rwi H2 H4.
  ufi set_of_reg_open H4. Ztac. nin H6. am.
  assert (open_r r (bar1_22 r (union X))). app Exercise1_22i.
  assert (inc (bar1_22 r (union X)) (set_of_reg_open r)). uf set_of_reg_open.
  Ztac. app powerset_inc. app Exercise1_22f.
  split.  split. ue. ir. rw Exercise1_22r. rwi H2 H0. cp (H0 _ H6).
  ufi set_of_reg_open H7. Ztac. ee. am. am.
  apply sub_trans with (union X). app union_sub. app  Exercise1_22e. am.
  ir. nin H6. rwi H2 H6. ufi set_of_reg_open H6. Ztac. rw Exercise1_22r.
  ee. am. am. rw (Exercise1_22p H H9). app Exercise1_22j. nin H9. am.
  red. ir. nin (union_exists H10). nin H11. cp (H7 _ H12).
  rwi Exercise1_22r H13. ee. app H15. am. am.
Qed.

Lemma Exercise1_22w: forall r u v, order r ->
  open_r r u -> open_r r v ->
  sup (reg_open_order r) u v = bar1_22 r (union2 u v).
Proof. ir. assert (sub (doubleton u v) (substrate (reg_open_order r))).
  uf reg_open_order. aw. red. ir. uf set_of_reg_open.
  nin (doubleton_or H2); rw H3;  Ztac; app powerset_inc. nin H0; nin H0; am.
  nin H1; nin H1; am. cp (Exercise1_22v H H2). uf sup.
  sy; app supremum_pr2. uf reg_open_order. fprops.
Qed.

Lemma Exercise1_22x: forall r, order r ->
  complete_lattice (reg_open_order r).
Proof. ir. assert (order (reg_open_order r)). uf reg_open_order. fprops.
  app exercise1_11b. ir. exists (bar1_22 r (union X)). ap (Exercise1_22v H H1).
Qed.

Lemma Exercise1_22y: forall r, order r ->
  lattice (reg_open_order r).
Proof. ir. cp (Exercise1_22x H). red. nin H0. ee. am. ir.
  assert (sub  (doubleton x y) (substrate (reg_open_order r))).
  red. ir. nin (doubleton_or H4); rww H5. ap (H1 _ H4).
Qed.
```

Assume $X \subset Y$, where X and Y are regular open sets. Let Z be the set of all elements of Y not bounded by an element of X. This is an open set. Every element of Y is bounded by an element of X or Z. This implies that Y is a subset of $\bar{Z} \cup X$, hence $Y = \sup(\bar{Z}, X)$. Obviously, $Z \cap X = \emptyset$, thus $\bar{Z} \cap \bar{X} = \emptyset$. Since $X = \bar{X}$, we get $\inf(\bar{Z}, X) = \emptyset$. As a consequence, our set is relatively complemented.

```
Lemma Exercise1_22z: forall r, order r ->
  relatively_complemented (reg_open_order r).
Proof. ir. cp (Exercise1_22t H).
  assert (order  (reg_open_order r)). uf reg_open_order. fprops.
  assert (the_least_element (reg_open_order r) = emptyset).
  app the_least_element_pr2.
  red. ee. app Exercise1_22y. exists emptyset. am. rw H2.
  ir. rwi Exercise1_22r H3. ee.
```

```
  set (z:= Zo y (fun u => forall t, gle r u t -> ~ (inc t x))).
  assert (sub z (substrate r)). apply sub_trans with y. uf z; app Z_sub.
  nin H4. nin H4. am.
  assert (open_o r z). red. ee. am. ir. ufi z H7. Ztac. clear H7. uf z.
  Ztac.  nin H4. nin H4. app (H11 _ _ H9 H8). ir. app H10. order_tac.
  cp (Exercise1_22i H H7).
  assert (open_o r (union2 x (bar1_22 r z))). app  Exercise1_22a.
  nin H3; am. nin H8; am. exists (bar1_22 r z). split. uf reg_open_order.
  aw. uf set_of_reg_open. Ztac. app powerset_inc.  nin H8; nin H8; am. split.
  rw Exercise1_22w. app extensionality. rw (Exercise1_22p H H4).
  app Exercise1_22j. nin H4; am. red. ir. nin (union2_or H10). app H5.
  rw (Exercise1_22p H H4). nin H4. assert (sub z y). uf z. app Z_sub.
  app (Exercise1_22j H H7 H4 H13).
  red. ir. app Exercise1_22h. nin H4. nin H4. app H4. ir.
  assert (inc y0 y). nin H4; nin H4. app (H13 _ _ H10 H11).
  nin (p_or_not_p (exists t, inc t x & gle r y0 t)). nin H13. nin H13.
  exists x1. ee. inter2tac. am. assert (inc y0 z). uf z. Ztac. ir.
  nin (inc_or_not t x). elim H13. exists t. au. am. exists y0.  split.
  app union2_second. app  (Exercise1_22e H H7).  order_tac. order_tac. am.
  am. am.
  rw Exercise1_22u. assert (intersection2 x z = emptyset). app is_emptyset.
  ir. red.  ir. nin (intersection2_both H10). ufi z H12. Ztac.
  assert (gle r y0 y0). order_tac. nin H3. nin H3. app H3. elim (H14 _ H15).
  am. cp (Exercise1_22p H H3).  nin H3. cp (Exercise1_22k H H3 H7 H10).
  wri H11 H13. rw H13. sy. app Exercise1_22p. app Exercise1_22m. am. am. am. am.
Qed.
```

We have to show that our set is distributive. Condition (T') reads $\overline{Z\cap\overline{X\cup Y}}\subset\overline{X\cup\overline{Y\cap Z}}$. Write this as $\bar{A}\subset\bar{B}$. By 1.22h, we have to show that for any $x\in\bar{A}$, and $x\leq x'$ there is $y\in B$ such that $x'\leq y$. We have $x'\in\bar{A}$, hence there is $x''\in A$ such that $x'\leq x''$. Since A has the form $Z\cap\overline{X\cup Y}$ there is $y\in X\cup Y$. such that $x''\leq y$. We have also $y\in Z$. We have $Z\cap(X\cup Y)\subset X\cup(Y\cap Z)$ (the relation we want to prove, without the bars). Hence $y\in X\cup(Y\cap Z)\subset\bar{A}$.

```
Lemma Exercise1_22A: forall r, order r ->
  boolean_lattice (reg_open_order r).
Proof. ir. cp (Exercise1_22t H).
  assert (order  (reg_open_order r)). uf reg_open_order. fprops.
  split. app Exercise1_22z. split.
  exists (substrate r). app  (Exercise1_22s H).
  cp (Exercise1_22y H). rww (Exercise1_16c H2). red. ir.
  set (a1:= sup (reg_open_order r) x y).
  set (a2:= inf (reg_open_order r) y z).
  assert (substrate (reg_open_order r) = (set_of_reg_open r)).
  uf reg_open_order. aw. aw.
  assert (inc a1 (substrate (reg_open_order r))). uf a1.
  cp (lattice_sup_pr H2 H3 H4). ee. order_tac.
  assert (inc a2 (substrate (reg_open_order r))). uf a2.
  cp (lattice_inf_pr H2 H4 H5). ee. order_tac.
  rww (Exercise1_22r). ee.
  cp (lattice_inf_pr H2 H5 H7). ee.
  assert (inc (inf (reg_open_order r) z a1) (substrate (reg_open_order r))).
  order_tac. rwi H6 H12. ufi set_of_reg_open H12.  Ztac. am.
  cp (lattice_sup_pr H2 H3 H8). ee.
  assert (inc (sup (reg_open_order r) x a2) (substrate (reg_open_order r))).
  order_tac. rwi H6 H12. ufi set_of_reg_open H12. Ztac. am.
  rwi H6 H3; rwi H6 H4; rwi H6 H5; rwi H6 H7;rwi H6 H8.
```

```
  ufi set_of_reg_open H3;  ufi set_of_reg_open H4;  ufi set_of_reg_open H5.
  ufi set_of_reg_open H7; ufi set_of_reg_open H8.
  Ztac. clear H8. Ztac. clear H7. Ztac. clear H5. Ztac. clear H4. Ztac.
  clear H3. clear H8; clear H9; clear H7; clear H5; clear H4.
  rw (Exercise1_22u H H12 H11). rw (Exercise1_22w H H14 H10).
  assert (a1 = bar1_22 r (union2 x y)). uf a1. rww (Exercise1_22w H H14 H13).
  assert (a2 = bar1_22 r (intersection2 y z)). uf a2.
  rww  (Exercise1_22u H H13 H12).
  red. ir. ufi bar1_22 H5. nin (union_exists H5). clear H6. nin H7. Ztac.
  clear H7. clear H8. ee. assert (sub x1 (substrate r)). nin H7. am.
  clear H5. nin H8. awii H5. awii H8.
  app Exercise1_22h. app Exercise1_22a. nin H14. am. nin H10. am.
  nin H7. app H7. ir. assert (inc y0 x1). nin H7.  app (H16 _ _ H6 H15).
  nin (H8 _ H16). nin H17. nin (intersection2_both H17). rwi H3 H20.
  ufi bar1_22 H20. nin (union_exists H20). clear H20. nin H21. Ztac. clear H21.
  ee. nin H23. rwi powerset_inc_rw H22. awii H23. awii H24.
  cp (related_induced_order1 H18). nin (H24 _ H20). nin H26.
  cp (related_induced_order1 H27).
  assert (gle r y0 x4). order_tac. exists x4. ee. nin (union2_or H26).
  app union2_first.  app union2_second. rww H4. app Exercise1_22e.
  uf intersection2. app  Exercise1_22b. app nonempty_doubleton.
  ir. nin (doubleton_or H31); rw H32. nin H13; am. nin H12; am.
  app intersection2_inc. nin H12. nin H12. app (H32 _ _ H19 H28). am.
Qed.
```

(c) If F *is a cofinal subset of* E*, show that the mapping* $U \to U \cap F$ *is an isomorphism of* R(E) *onto* R(F)*.*

Assume U regular in E. Then $U \cap F$ is regular. In fact, assume $U \cap F$ cofinal in an open set V of F. Let $x \in V$. We must show $x \in U \cap F$. We have obviously $x \in F$. Assume $x \leq y$. Since F is cofinal, there is $z \in F$ such that $y \leq z$. Since V is open, we have $z \in V$, so there is $t \in U \cap F$ with $z \leq t$. Thus, there is $t \in U$ such that $y \leq t$. By 1.22h, thus says $x \in \bar{U}$, hence $x \in U$.

```
Lemma Exercise1_22B: forall r F x, order r ->
  cofinal_set r F -> open_r r x ->
  open_r (induced_order r F) (intersection2 x F).
Proof. ir.  assert (sub F (substrate r)). nin H0. am.
  assert (order (induced_order r F)). fprops.
  assert (open_o  (induced_order r F) (intersection2 x F)).
  ir. red. ee. aw. app intersection2sub_second.
  ir. nin (intersection2_both H4).
  cp (related_induced_order1 H5). nin (related_induced_order3 H5).
  app intersection2_inc. nin H1. nin H1. ap (H12 _ _ H6 H8).
  red. split. am. ir. app extensionality.
  red. ir. assert (inc x0 F). nin H5.  awi H5. app H5. am. am.
  app intersection2_inc. rw (Exercise1_22p H H1). cp (H2 _ H9).
  app Exercise1_22h. nin H1; am. ir. nin H0. assert (inc y (substrate r)).
  order_tac. nin (H12 _ H13). nin H14. nin H7. awii H16.
  assert (gle (induced_order r F) x0 x1). aw. order_tac.
  nin H5. cp (H18 _ _ H8 H17). nin (H16 _ H19). nin H20. exists x2. ee.
  inter2tac. cp (related_induced_order1 H21).
  cp (related_induced_order1 H22). order_tac. nin H5. am.
Qed.
```

Thus $U \mapsto U \cap F$ maps R(E) into R(F). Assume $U \cap F \subset U' \cap F$. Let $x \in U$, and $x \leq y$. There exists $z \in F$ such that $y \leq z$. We have $z \in U'$, and by 1.22h, this says $x \in \bar{U}'$, hence $U \subset U'$.

```
Lemma Exercise1_22C: forall r F U U', order r ->
  cofinal_set r F -> open_r r U -> open_r r U' ->
    sub (intersection2 U F) (intersection2 U' F) -> sub U U'.
Proof. ir. red. ir. rw (Exercise1_22p H H2).  app Exercise1_22h.
  nin H2. am. nin H1; nin H1. app H1. ir. assert (inc y (substrate r)).
  order_tac. nin H0. nin (H7 _ H6). nin H8. nin H1. nin H1.
  assert (gle r x x0). order_tac. cp (H11 _ _ H4 H12).
  assert (inc x0 (intersection2 U' F)). app H3. app intersection2_inc.
  cp (intersection2_first H14). ex_tac.
Qed.
```

Let g be the mapping $X \mapsto X \cap F$. We have shown that if $g(x) \subset g(y)$ then $x \subset y$. The converse is obvious. As a consequence, g is increasing and injecctive. It is also surjective. We use the same argument as before. If X is a regular open subset of F, Y be the elements $y \in E$ such that there is $x \in X$ with $x \leq y$, then $g(\bar{Y}) = X$.

```
Lemma Exercise1_22D: forall r F, order r ->
  cofinal_set r F ->
  order_isomorphism (BL (fun z => intersection2 z F) (set_of_reg_open r)
    (set_of_reg_open (induced_order r F)))
  (reg_open_order r)(reg_open_order (induced_order r F)).
Proof. ir. assert (sub F (substrate r)). nin H0. am.
  assert (order (induced_order r F)). fprops.
  set (r':=  induced_order r F) in *.
  assert (substrate r' = F). uf r'. aw.
  assert (forall x, open_r r x -> open_r r' (intersection2 x F)).
  ir. app (Exercise1_22B H H0 H4).
  assert (transf_axioms (fun z => intersection2 z F) (set_of_reg_open r)
        (set_of_reg_open r')).
  red. ir. ufi set_of_reg_open H5. Ztac. clear H5. cp (H4 _ H7).
  uf set_of_reg_open. Ztac. app powerset_inc. nin H5. nin H5;am.
  set (g:=BL (fun z : Set => intersection2 z F) (set_of_reg_open r)
        (set_of_reg_open r')).
  assert (is_function g). uf g. app bl_function.
  assert (bijective g). uf g. app bijective_bl_function.
  ir. ufi set_of_reg_open H7. Ztac. ufi set_of_reg_open H8. Ztac.
  app extensionality. app (Exercise1_22C (U:=u) (U':=v) H H0).
  rw H9; fprops.  app (Exercise1_22C (U:=v) (U':=u) H H0). rw H9; fprops.
  ir. ufi set_of_reg_open H7. Ztac. clear H7. clear H8.
  assert (sub y (substrate r)). nin H9. nin H7. apply sub_trans with F. ue. am.
  set (x1:= Zo (substrate r) (fun z => exists x, inc x y & gle r x z)).
  assert (sub x1 (substrate r)). uf x1. app Z_sub.
  assert (sub y x1). red. ir. uf x1. Ztac. exists x. ee. am. order_tac. app H7.
  assert (open_o r x1). red. ee. am. ir. ufi x1 H11. Ztac. clear H11. nin H14.
  uf x1. Ztac. order_tac. nin H11. ex_tac. order_tac.
  set (x2:= bar1_22 r x1). assert (sub y (intersection2 x2 F)). red. ir.
  app intersection2_inc. uf x2. app (Exercise1_22e H H11). app H10. nin H9.
  nin H9. wr H3. app H9. cp (Exercise1_22i H H11). exists (bar1_22 r x1).
  split. uf set_of_reg_open. Ztac. app powerset_inc. app Exercise1_22f.
  app extensionality. red. ir. nin (intersection2_both H14).
  rw (Exercise1_22p H2 H9). app Exercise1_22h. nin H9. am. ue. ir.
  ufi r' H17. cp (related_induced_order1 H17).
  assert (inc y0  (bar1_22 r x1)). nin H13. nin H13. ap (H20 _ _ H15 H18).
  ufi bar1_22 H19. nin (union_exists H19). nin H20. Ztac.
  nin H23. rwi powerset_inc_rw H22. nin H24. awii H25. nin (H25 _ H20).
  nin H26. cp (related_induced_order1 H27). ufi x1 H26.  Ztac. nin H30.
```

```
  nin H30. nin H0. nin (H32 _ H29). nin H33.
  assert (gle r' x4 x5). uf r'.  aw. order_tac. nin H9. nin H9. ue.
  exists x5.  ee. nin H9. nin H9.  app (H37 _ _ H30 H35). uf r'. aw. order_tac.
  nin (related_induced_order3 H17). am.
  red. ee. uf reg_open_order. fprops.  uf reg_open_order. fprops. am.
  uf g. uf reg_open_order. aw. uf reg_open_order. uf g. aw.  uf g. aw.
  ir. aw.
  uf reg_open_order. aw. ap iff_eq. ir. ee. app H5. app H5. red. ir.
  nin (intersection2_both H13). app intersection2_inc. app H12. ir.
  ee. am. am. ufi set_of_reg_open H8. Ztac.  ufi set_of_reg_open H9. Ztac.
  app (Exercise1_22C H H0 H14 H16 H12).
Qed.
```

(d) If E_1, E_2 *are two ordered sets, then every open set in* $E_1 \times E_2$ *is of the form* $U_1 \times U_2$, *where* U_i *is open in* E_i *(*$i = 1,2$*). The set* $R(E_1 \times E_2)$ *is isomorphic to* $R(E_1) \times R(E_2)$.

Let E be a set, ordered by the diagonal order ($x \leq y$ if and only if $x \leq y$). Then every subset is a regular open set. The product $E \times E$ is ordered by the diagonal order. As a consequence, there are open sets that are not products. The second part of the claim is wrong as well. The argument is the following. Assume that $E_1 = E_2$ has a single element. Then $R(E_1) = R(E_2)$ has two elements. The product $E_3 = E_1 \times E_2$ has a single element, and $R(E_3)$ has two elements. It cannot be isomorphic to $R(E_1 \times E_2)$ that has four elements.

```
Lemma Exercise1_22E: forall r r' X X', order r -> order r' ->
  open_o r X -> open_o r' X' -> open_o  (product2_order r r') (product X X').
Proof. ir. split. rww substrate_order_product2_order.
  app product_monotone. nin H1. am. nin H2. am. ir. rwi product2_order_pr H4.
  red in H4. ee. awi H3. ee. nin H1. cp (H10 _ _ H8 H6).
  nin H2. cp (H12 _ _ H9 H7).  awi H5. aw. eee.
Qed.


Lemma Exercise1_22F: forall E X,
  let r := diagonal E in
    sub X E-> open_r r X.
Proof. ir.  assert (forall x y, gle r x y -> x = y). uf gle.  uf related. uf r.
  ir. awi H0. ee. am.
  cp (diagonal_order E). split. split. uf r.
  rww substrate_diagonal. ir. wrr (H0 _ _ H3). ir.
  app extensionality. nin H4. awii H5. red. ir. nin (H5 _ H6). nin H7.
  awii H8. rww (H0 _ _ H8). nin (related_induced_order3 H8). am. nin H2; am.
Qed.

Lemma Exercise1_22G: forall E,
  let r := diagonal E in
    (product2_order r r = diagonal (product E E)).
Proof. ir.  assert (substrate r = E). uf r.  rww substrate_diagonal.
  assert (forall x y, related r x y -> x = y).  uf related. uf r.
  ir. awi H0. ee. am.
  uf product2_order. rw H. uf graph_on. set_extens. Ztac. nin H3. rwi H H3. ee.
  cp (H0 _ _ H5).  cp (H0 _ _ H6). rw inc_diagonal.
  assert (is_pair x). awi H2; ee; am. ee. am. am.  awi H3; awi H4. ee.
  app pair_extensionality.  rwi inc_diagonal H1. ee.  assert (Ha:= H2).
  awi H2. ee. Ztac. aw. ee. am. am. am. am. ue. ue. ue. red. rw H. wr H3.
  ee. am. am. uf r. red. rww inc_diagonal. ee. fprops. aw. aw.
  red. uf r. rww inc_diagonal. ee. fprops. aw. aw.
```

```
Qed.
```

¶ 23. *Let* E *be an ordered set and let* $R_0(E) = R(E) - \{\emptyset\}$ *(Exercise 22). For each* $x \in E$*, let* $r(x)$ *denote the unique regular open set in which the interval* $[x, \rightarrow [$ *(which is an open set) is cofinal. The mapping* r *so defined is called the canonical mapping of* E *into* $R_0(E)$*. Endow* $R_0(E)$ *with the order relation* opposite *to the relation of inclusion.*

We start with the definition of E_0 and its ordering.

```
Definition set_of_nreg_order r :=
  opposite_order (inclusion_suborder (set_of_nreg_open r)).

Lemma Exercise1_23a: forall r X,
  inc X (set_of_nreg_open r) = (open_r r X & nonempty X).
Proof. ir. uf set_of_nreg_open. ap iff_eq. ir. srwi H.  nin H.
  ufi set_of_reg_open H. Ztac. ee. am. nin (emptyset_dichot X). elim H0.
  rw H3. fprops. am. ir. ee. srw. split. uf set_of_reg_open. Ztac.
  app powerset_inc. nin H; nin H; am. red. ir. awi H1. rwi H1 H0.
  nin H0. elim (emptyset_pr H0).
Qed.

Lemma Exercise1_23b: forall r X Y, order r ->
  gle  (set_of_nreg_order r) X Y =
  (nonempty X & nonempty Y & open_r r X & open_r r Y & sub Y X).
Proof. ir. uf set_of_nreg_order. aw. rw  Exercise1_23a.  rw  Exercise1_23a.
  ap iff_eq. ir. eee. ir. eee. fprops.
Qed.
```

(a) Show that the mapping r *is increasing and that* $r(E)$ *is cofinal in* $R_0(E)$*.*

We have the following interesting property: $y \in r(x)$ if and only if, whenever $y \leq z$, there is a common upper bound to x and z. The mapping r is increasing, as the compsotion of two increasing functions. In general, it is not strictly increasing (if E is right directed it is constant).

```
Definition canonical_reg_open r x :=
  bar1_22 r (Zo (substrate r) (fun z => gle r x z)).

Lemma  Exercise1_23c: forall r x, order r ->
  open_o r  (Zo (substrate r) (fun z => gle r x z)).
Proof. ir. split. app Z_sub. ir. Ztac. clear H0. Ztac. order_tac. order_tac.
Qed.

Lemma Exercise1_23d1: forall r x, order r -> inc x (substrate r) ->
  inc x (canonical_reg_open r x).
Proof.  ir. uf canonical_reg_open. app Exercise1_22e. ap (Exercise1_23c x H).
  Ztac. order_tac.
Qed.

Lemma  Exercise1_23d2: forall r x, order r -> inc x (substrate r) ->
  inc (canonical_reg_open r x) (set_of_nreg_open r).
Proof. ir. rw Exercise1_23a. split. uf canonical_reg_open.
```

```
  cp (Exercise1_23c x H). app  Exercise1_22i.
  exists x. app Exercise1_23d1.
Qed.

Lemma Exercise1_23e:  forall r x y, order r ->
  inc x (substrate r) -> inc y (substrate r) ->
  (inc y (canonical_reg_open r x) =
    forall z, gle r y z -> exists t, gle r z t & gle r x t).
Proof. ir. uf canonical_reg_open. app iff_eq. ir. ufi bar1_22 H2.
  nin (union_exists H2). nin H4. Ztac. nin H7. nin H8. awii H9.
  assert (inc z x0). nin H7. ap (H10 _ _ H4 H3).
  nin (H9 _ H10). nin H11. Ztac.
  cp (related_induced_order1 H12). exists x1. au. nin H7. am.
  ir. app Exercise1_22h. app Exercise1_23c. ir. nin (H2 _ H3). nin H4.
  exists x0. ee. Ztac. order_tac. am.
Qed.

Lemma  Exercise1_23f: forall r x y, order r ->
  gle r x y -> gle  (set_of_nreg_order r)
  (canonical_reg_open r x)  (canonical_reg_open r y).
Proof. ir. assert (inc x (substrate r)). order_tac.
  assert (inc y (substrate r)). order_tac.
  cp (Exercise1_23d2 H H1).  cp (Exercise1_23d2 H H2).
  uf set_of_nreg_order. aw. ir. eee. uf canonical_reg_open.
  app Exercise1_22j. app Exercise1_23c. app Exercise1_23c.
  red. ir. Ztac. clear H5. Ztac. order_tac. fprops.
Qed.
```

If X is regular, then $X = \bar{X}$. If $x \in X$, then $[x, \rightarrow [\subset X$, thus $r(x) \subset X$. As a consequence the image of r is cofinal in R_0.

```
Lemma  Exercise1_23g: forall r, order r ->
  cofinal_set (set_of_nreg_order r)
   (fun_image (substrate r) (canonical_reg_open r)).
Proof. ir.
  assert (substrate (set_of_nreg_order r) = set_of_nreg_open r).
  uf set_of_nreg_order. aw. fprops.
  red. split. red. ir. awi H1. nin H1. nin H1. rw H0. wr H2.
  app Exercise1_23d2. fprops.
  ir. rwi H0 H1. rwi Exercise1_23a H1. nin H1.  nin H2.
  assert (inc y (substrate r)). nin H1. nin H1. app H1.
  cp (Exercise1_23d2 H H3). rwi Exercise1_23a H4.
  exists  (canonical_reg_open r y). split. aw. exists y. split. am. tv.
  rww Exercise1_23b. eee.  exists y. am.
  uf canonical_reg_open. rw (Exercise1_22p H H1). app Exercise1_22j.
  app Exercise1_23c. nin H1; am. red. ir. Ztac. nin H1. nin H1.
  ap (H10 _ _ H2 H8).
Qed.
```

(b) An ordered set E *is said to be* antidirected *if the canonical mapping* $r : E \rightarrow R_0(E)$ *is injective. For this to be so it is necessary and sufficient that the following two conditions should be satisfied.*

(I) If x *and* y *are two elements of* E *such that* $x < y$*, there exists* $z \in E$ *such that* $x < z$ *and such that the intervals* $[y, \rightarrow [$ *and* $[z, \rightarrow [$ *do not intersect.*

(II) If x and y are two non-comparable elements of E *then either there exists $x' \geq x$ such that the intervals $[x', \rightarrow [$ and $[y, \rightarrow [$ do not intersect, or else there exists $y' \geq y$ such that the intervals $[x, \rightarrow [$ and $[y', \rightarrow [$ do not intersect.*

Let $A(x, y)$ be the property that the intervals $[x, \rightarrow [$ and $[y, \rightarrow [$ do not intersect. We might replace $x < z$ in (I) by $x \leq z$, for $A(y, x)$ is false (since y is in the intersection). Our criterion 1.23e says: $a \in r(x)$ if and only if, forall b such that $a \leq b$, $A(b, x)$ is false.

Thus (I) can be written as: if $x < y$ then $x \notin r(y)$, and (II) as if x and y are non-comparable, then $x \notin r(y)$ or $y \notin r(x)$. Write this as "not ($x \in r(y)$ and $y \in r(x)$)". Since $x \in r(x)$ and $y \in r(y)$ this is $r(x) \neq r(y)$. Condition (II) becomes: if $r(x) = r(y)$, then x and y are comparable. But (I) excludes $x < y$ and $y < x$, so that (I) and (II) imply injectivity of r. Conversely, injectvivity implies (II). Assume now $x < y$. There is z such that $x \leq z$ and $A(y, z)$, since otherwise we would have $r(x) \subset r(y)$. But $x \leq y$ implies $r(x) \subset r(y)$, thus $r(x) = r(y)$. If the set if antidirected, we get $x = y$, absurd.

```
Lemma  Exercise1_23h: forall r, order r ->
  let aux := (fun x y => forall z, gle r x z -> gle r y z -> False) in
  (anti_directed r) =
  ((forall x y, glt r x y -> exists z, (glt r x z & aux y z))
    & forall x y, inc x (substrate r) -> inc y (substrate r) ->
      (gle r x y \/ gle r y x \/ (exists x', gle r x x' & aux x' y) \/
        (exists y', gle r y y' & aux x y'))).
Proof. ir. assert (forall x y, inc x (substrate r) -> inc y (substrate r) ->
  (inc y  (canonical_reg_open r x) = forall z, gle r y z -> ~ (aux x z))).
  ir. rww  Exercise1_23e. uf aux. ap iff_eq. ir. red. ir.
  nin (H2 _ H3). nin H5. app (H4 _ H6 H5). ir. cp (H2 _ H3).
  app exists_proof. dneg. ir. elim (H5 z0). au.
  assert (Hb:forall x y, gle r x y ->
    sub  (canonical_reg_open r y) (canonical_reg_open r x)).
  ir. cp (Exercise1_23f H H1).  rwi Exercise1_23b H2. ee; am. am.
  assert (Hc:forall x y, aux x y -> aux y x). uf aux. ir.
  app (H1 _  H3 H2).
  sy. app iff_eq. ir. red. ir. nin H1.
  assert (inc x (canonical_reg_open r x)). app (Exercise1_23d1).
  assert (inc y (canonical_reg_open r y)). app (Exercise1_23d1).
  nin (equal_or_not x y). am. nin (H5 _ _ H2 H3). assert (glt r x y). split;am.
  nin (H1 _ _ H10).  nin H11. rwi H4 H6. rwi H0 H6. nin H11. elim (H6 _ H11).
  am. am. am. nin H9.
  assert (glt r y x). split;au. nin (H1 _ _ H10).
  nin H11. wri H4 H7. rwi H0 H7. nin H11. elim (H7 _ H11). am. am. am.
  nin H9. nin H9. nin H9. rwi H4 H6.  rwi H0 H6. elim (H6 _ H9). app Hc.
  am. am. nin H9; nin H9. wri H4 H7. rwi H0 H7. elim (H7 _ H9). am. am. am.
  ir.
  assert (forall x y,  inc x (substrate r) ->  inc y (substrate r) ->
    (sub (canonical_reg_open r x) (canonical_reg_open r y)
    \/ (exists x' : Set, gle r x x' & aux x' y))).
  ir. nin (p_or_not_p (exists x' : Set, gle r x x' & aux x' y)). au. left.
  cp (Exercise1_23d2 H H3). rwi Exercise1_23a H5. nin H5.
  rw (Exercise1_22p H H5). uf canonical_reg_open. app Exercise1_22j.
  app Exercise1_23c. nin H5. am. red. ir. ufi canonical_reg_open H0. rw H0.
  Ztac. ir. nin (p_or_not_p (aux y z)). elim H4. exists z. ee. order_tac.
  app Hc. am. am. Ztac. am.
  cut ((forall x y : Set, glt r x y -> exists z : Set, glt r x z & aux y z)).
  ir. split. am. ir.
  nin (p_or_not_p (gle r x y)). left. am. right.
```

```
  nin (p_or_not_p (gle r y x)). left. am. right.
  nin (H2 _ _ H4 H5). nin (H2 _ _ H5 H4).
  assert (canonical_reg_open r x = canonical_reg_open r y). app extensionality.
  red in H1. rwi (H1 _ _ H4 H5 H10) H6. elim H6. order_tac. right. nin H9.
  nin H9. exists x0. ee. am. app Hc.  left. am.
  ir.
  assert (inc x (substrate r)). order_tac.
  assert (inc y (substrate r)). order_tac. nin H3. nin (H2 _ _ H4 H5).
  cp (Exercise1_23f H H3). rwi Exercise1_23b H8.  ee.
  assert (canonical_reg_open r x = canonical_reg_open r y). app extensionality.
  red in H1. rwi (H1 _ _ H4 H5 H13) H6. elim H6. tv. am. nin H7.
  exists x0. nin H7.  split. split. am. red. ir. wri H9 H8. red in H8.
  elim (H8 y). am. order_tac. app Hc.
Qed.
```

(c) Show that, for every ordered set E, $R_0(E)$ *is antidirected and that the canonical mapping of* $R_0(E)$ *into* $R_0(R_0(E))$ *is bijective (use Exercise 22(a)).*

The condition A(X,Y) in $R_0(E)$ says that there is no upper bound for X and Y. We know that R(E) is a complete lattice, so that the condition becomes: there is only one upper bound, namely the empty set (that is not in R_0). Since $x \in X$ implies $X \leq r(x)$, the condition becomes X and Y are disjoint.

```
Lemma Exercise1_23i: forall r x y, order r ->
  inc x y -> inc y  (set_of_nreg_open r) ->
    gle (set_of_nreg_order r) y (canonical_reg_open r x).
Proof. ir. rw Exercise1_23b. rwi Exercise1_23a H1. nin H1.
  assert (sub y (substrate r)). nin H1. nin H1. am.
  cp (Exercise1_23d2 H (H3 _ H0)).
  rwi Exercise1_23a H4. eee. uf canonical_reg_open.
  rw (Exercise1_22p H H1). app  Exercise1_22j. app  Exercise1_23c. nin H1; am.
  red. ir. Ztac. nin H1; nin H5. nin H1. ap (H10 _ _ H0 H8). am.
Qed.

Lemma Exercise1_23j: forall r, order r ->
  let r':= set_of_nreg_order r in
    (forall x y, inc x (substrate r') -> inc y (substrate r') ->
      (forall z, gle r' x z ->  gle r' y z -> False) =
      (disjoint  x y)).
Proof. ir.  assert (order r'). uf r'. uf set_of_nreg_order. fprops.
  assert (substrate r' = set_of_nreg_open  r). uf r'. uf set_of_nreg_order. aw.
  fprops.
  ap iff_eq. ir. app disjoint_pr. ir. rwi H3 H0; rwi H3 H1.
  ap (H4 _ (Exercise1_23i H H5 H0) (Exercise1_23i H  H6 H1)).
  ir. red in H4. ufi r' H5. rwi Exercise1_23b H5. ufi r' H6.
  rwi Exercise1_23b H6. ee. nin H11. elim (emptyset_pr (x:=y0)). wr H4.
  app intersection2_inc.  app H14. app H10. am. am.
Qed.
```

We know that R(E) is a Boolean lattice. Given X and Y we construct a regular open set Z, a subset of X that is disjoint from Y. (This is the complement (for the lattice) of Y in X whenever $Y \subset X$). If Z is empty, then $X \subset Y$. This can be restated as: if $X \subset Y$ is false, then $Z \in R_0(E)$. It follows that $R_0(E)$ is antidirected. The canonical mapping of $R_0(E)$ into $R_0(R_0(E))$ is hence injective.

```
Lemma Exercise1_23k: forall r, order r ->
```

```
  anti_directed (set_of_nreg_order r).
Proof. ir. set (r':= set_of_nreg_order r).
  assert (order r'). uf r'. uf set_of_nreg_order. fprops.
  set (aux:=(fun x y => forall z, gle r' x z ->  gle r' y z -> False)).
  assert (forall x y, inc x (substrate r') -> inc y (substrate r') ->
    (disjoint  x y) -> aux x y). ir. uf aux. uf r'. rww Exercise1_23j.
  assert (Hv:substrate r' = set_of_nreg_open  r).
  uf r'. uf set_of_nreg_order. aw.  fprops.
  set (i := fun x y => bar1_22 r
    (Zo x (fun u => forall t, gle r u t -> ~ (inc t y)))).
  assert (forall x y,  inc x (substrate r') -> inc y (substrate r') ->
    (open_r r (i x y) & sub (i x y) x & disjoint (i x y) y
    & (i x y = emptyset -> sub x y))). ir.
  set (z:= Zo x (fun u => forall t, gle r u t -> ~ (inc t y))).
  ufi r' H2; ufi r' H3. ufi set_of_nreg_order H2; ufi set_of_nreg_order H3.
  awi H2. awi H3. rwi Exercise1_23a H2. rwi Exercise1_23a H3. nin H2; nin H3.
  assert (sub z (substrate r)). apply sub_trans with x. uf z; app Z_sub.
  nin H2. nin H2. am.
  assert (open_o r z). red. ee. am. ir. ufi z H7. Ztac. clear H7. uf z.
  Ztac. nin H2. nin H2. app (H11 _ _ H9 H8). ir. app H10. order_tac.
  cp (Exercise1_22i H H7). uf i. fold z. split. am. split.
  rw (Exercise1_22p H H2). app Exercise1_22j. nin H2; am. uf z. app Z_sub.
  split. uf disjoint.  rw (Exercise1_22p H H3). app Exercise1_22k. nin H3; am.
  app is_emptyset. red; ir. nin (intersection2_both H9). ufi z H10. Ztac.
  assert (gle r y0 y0). order_tac. nin H3. nin H3. app H3. elim (H13 _ H14).
  am. ir. red. ir. rw (Exercise1_22p H H3). app Exercise1_22h. nin H3;am.
  nin H2; nin H2; app H2. ir. assert (inc y0 x). nin H2. nin H2.
  app (H13 _ _ H10 H11). app exists_proof. red. ir.
  elim (emptyset_pr (x:= y0)). wr H9. app Exercise1_22e. uf z. Ztac. ir.
  nin (inc_or_not t y).  elim (H13 t). au. am. fprops. fprops.
  assert (forall x y, inc x (substrate r') -> inc y (substrate r') ->
    (inc (i x y) (substrate r') \/ sub x y)). ir. cp (H2 _ _ H3 H4). ee.
  rw Hv. rw Exercise1_23a. nin (emptyset_dichot (i x y)). right. app H8.
  au. fprops. rw Exercise1_23h.
  split. ir. nin H4.
  assert (inc x (substrate r')). order_tac.
  assert (inc y (substrate r')). order_tac.
  ufi r' H4. rwi Exercise1_23b H4. ee. nin (H3 _ _ H6 H7).
  cp (H2 _ _ H6 H7). ee. exists  (i x y). split. split. uf r'.
  rww Exercise1_23b. eee. rwi Hv H12. rwi Exercise1_23a H12. ee; am.
  red. ir. nin H8. elim (emptyset_pr (x:=y0)). red in H15. wr H15.
  app intersection2_inc. wr H17. app H11. app H1. app disjoint_symmetric.
  elim H5. app extensionality. am. ir.
  nin (H3 _ _ H4 H5).  cp (H2 _ _ H4 H5). right. right. left.
  exists  (i x y). split. uf r'. rww Exercise1_23b.
  rwi Hv H4. rwi Exercise1_23a H4. rwi Hv H6. rwi Exercise1_23a H6. ee; tv.
  ee. app H1. right; left. uf r'.  rww Exercise1_23b.
  rwi Hv H4. rwi Exercise1_23a H4. rwi Hv H5. rwi Exercise1_23a H5. ee; tv. am.
Qed.
```

Let's study r in $R_0(E)$. We can rewrite 1.23i as: $y \in r(x)$ if and only if for all t, $y \leq t$ implies $t \cap x$ is non-empty. Let $a \in y$. We have $y \leq r(a)$, hence $r(a) \cap x$ is non-empty. This implies that there is $b \in x$, such that $a \leq b$. Conversely, assume every $a \in y$ bounded by an element of x; assume $y \leq t$. There is $a \in t \subset y$. If $a \leq b$ then $b \in t$. If moreover $a \in x$ the set $t \cap x$ is non-empty.

Hence $y \in r(x)$ is and only if every element of y is bounded by an element of x. It follows that if x non-empty and open, then $y \in r(\bar{x})$ is and only if every element of y is bounded by an element of x.

```
Lemma Exercise1_23l: forall r y, order r ->
  let r' := set_of_nreg_order r in
    inc y (set_of_nreg_open r') ->
    exists_unique (fun x => inc x (set_of_nreg_open r) &
      y =  canonical_reg_open r' x).
Proof. ir.  split. set (E:=set_of_nreg_open r). set (E':=set_of_nreg_open r').
  assert (order r'). uf r'. uf set_of_nreg_order. fprops.
  assert (E = substrate r'). uf r'. uf set_of_nreg_order. aw. fprops.
  assert (forall x t, inc x E -> inc t E -> inc t (canonical_reg_open r' x) =
    (forall u, gle r' t u -> nonempty (intersection2 u x))).
  ir. rww Exercise1_23e. ap iff_eq; ir; nin (H5 _ H6).  ufi r' H7.
  rwi  Exercise1_23b H7. rwi Exercise1_23b H7. ee. nin H9.
  exists y0. app intersection2_inc. app H16. app  H12. am. am.
  nin (intersection2_both H7). assert (inc z (substrate r')). order_tac.
  ufi r' H10.  ufi set_of_nreg_order H10; awi H10. cp (Exercise1_23i H H8 H10).
  rwi H2 H3. ufi r' H3. ufi set_of_nreg_order H3; awi H3.
  cp (Exercise1_23i H H9 H3). exists (canonical_reg_open r y0). au. fprops.
  fprops. ue. ue.
  assert (forall x t, inc x E -> inc t E -> inc t (canonical_reg_open r' x) =
    (forall a, inc a t -> (exists b, inc b x & gle r a b))).
  ir. rww H3. ap iff_eq. ir. cp (Exercise1_23i H H7 H5). cp (H6 _ H8).
  ufi canonical_reg_open H9. ufi E H4. rwi Exercise1_23a H4. nin H4.
  rwi (Exercise1_22p H H4) H9.
  nin (emptyset_dichot (intersection2 (Zo (substrate r)
    (fun z => gle r a z)) x)). nin H4. cp  (Exercise1_23c a H).
  rwi (Exercise1_22k H H13 H4) H9. nin H9. elim (emptyset_pr H9). am. nin H11.
  nin (intersection2_both H11). Ztac. exists y0. au.
  ir. ufi r' H7. rwi Exercise1_23b H7. ee. nin H8. nin (H6 _ (H11 _ H8)).
  nin H12. nin H10. nin H10. cp (H15 _ _ H8 H13). exists x0.
  app intersection2_inc. am.
  assert (forall x t, nonempty x -> open_o r x -> inc t E ->
    inc t (canonical_reg_open r' (bar1_22 r x)) =
    (forall a, inc a t -> (exists b, inc b x & gle r a b))).
  ir. assert (inc (bar1_22 r x) E). uf E. rw  Exercise1_23a. ee.
  app  Exercise1_22i. nin H5. exists y0. app  Exercise1_22e. rw (H4 _ _ H8 H7).
  ap iff_eq. ir. nin (H9 _ H10). ee. ufi bar1_22 H11. nin (union_exists H11).
  nin H13. Ztac. nin H16. nin H17. rwi powerset_inc_rw H1<5. awii H18.
  nin (H18 _ H13). nin H19. cp (related_induced_order1 H20).
  exists x2. split. am. order_tac. ir. nin (H9 _ H10). nin H11. exists x0.
  split. app  Exercise1_22e. am.
  assert (Hu:forall x, inc x y -> open_r r x). ir.  rwi  Exercise1_23a H0.
  nin H0. nin H0. nin H0. cp (H0 _ H6). wri H2 H10. ufi E H10.
  rwi Exercise1_23a H10. nin H10. am.
  assert (sub (union y) (substrate r)). red. ir. nin (union_exists H6).
  nin H7. nin (Hu _ H8). nin H9. app H9.
  assert (open_o r (union y)). app Exercise1_22c.  ir. nin (Hu _ H7). am.
```

Let Y be in $R_0(R_0(E))$. Proving surjectivity of r means showing existence of a set X such that every element of the union of the elements of Y is bounded by an element of X. Take $X = \cup Y$. We have $Y \subset r(\bar{X})$. Assume Y cofinal. Since Y is regular, this will imply $Y = r(\bar{X})$. Let X be a regular open set, such that each element of X is bounded by an element of the union of Y. We must show: There is $Z \in Y$ such that $Z \subset X$. Is this true?

```
  set (T:=Zo E (fun z => forall a, inc a z -> exists b,
    inc b (union y) & gle r a b)).
  set (y':= canonical_reg_open r' (bar1_22 r (union y))).
  assert (nonempty (union y)). rwi  Exercise1_23a H0.
  nin H0. nin H8. nin H0. nin H0. cp (H0 _ H8). wri H2 H11. ufi E H11.
  rwi Exercise1_23a H11. nin H11. nin H12. exists y1.  union_tac.
  assert (Hv:inc (bar1_22 r (union y)) E). uf E.
  rw Exercise1_23a. split. app Exercise1_22i. nin H8. exists y0.
  app  Exercise1_22e.
  assert (T = y').  cut (forall t, inc t E -> (inc t T = inc t y')). ir.
  set_extens. wr H9. am. ufi T H10. Ztac. am. rw H9. am. ufi y' H10.
  assert (inc  (bar1_22 r (union y)) (substrate r')). wr H2. am.
  cp (Exercise1_23d2 H1 H11).
  set (k := canonical_reg_open r' (bar1_22 r (union y))) in *.
  rwi Exercise1_23a H12. nin H12. nin H12. nin H12. rw H2. app H12.
  ir. uf y'. rw (H5 _ _ H8 H7 H9). ap iff_eq. ir. ufi T H10. Ztac.
  ap (H13 _ H11). ir. uf T. Ztac.
  assert (sub y y'). wr H9. uf T. red. ir. Ztac.  rwi  Exercise1_23a H0.
  nin H0. nin H0. nin H0. rw H2. ap (H0 _ H10). ir. exists a.
  assert (inc a (union y)). union_tac.  split. am. order_tac.  app H6.
  assert (open_o r' y'). uf y'.  rwi H2 Hv. cp (Exercise1_23d2 H1 Hv).
  rwi Exercise1_23a H11. nin H11. nin 11. nin H11. am. am.
  assert (sub y y'). wr H9. uf T. red. ir. Ztac.  rwi  Exercise1_23a H0.
  nin H0. nin H0. nin H0. rw H2. ap (H0 _ H10). ir. exists a.
  assert (inc a (union y)). union_tac.  split. am. order_tac.  app H6.
  assert (open_o r' y'). uf y'.  rwi H2 Hv. cp (Exercise1_23d2 H1 Hv).
  rwi Exercise1_23a H11. nin H11. nin 11. nin H11. am. am.
  assert (substrate (induced_order r' y') = y'). aw.  wrr H2. wr H9.
  uf T. app Z_sub.

(*
  set (T:= intersection(Zo (powerset (substrate r))
    (fun x => open_o r x & forall a, inc a (union y) -> exists b,
      inc b x & gle r a b))).
  assert (open_o r T). uf T. app Exercise1_22b. exists (union y). Ztac.
  app powerset_inc. split. app Exercise1_22c. ir. nin (Hu _ H7). am.
  ir. exists a. split. am. order_tac. app H6. ir. Ztac. nin H9; am.
  assert (nonempty T).
  maybe T is empty ??
*)
(* uniqueness is obvious
  assert (forall x y0 : Set, f x -> f y0 -> x = y0). uf f. ir.
  cp (Exercise1_23j H). red in H3. ee. fold r' in H3.
  assert (inc x (substrate r')). uf r'. uf set_of_nreg_order. aw. fprops.
  assert (inc y0 (substrate r')). uf r'. uf set_of_nreg_order. aw. fprops.
  cp (H3 _  _ H6 H7). app H8.  wrr H5; wrr H4.
  red. eee.
*)
Abort.
```

24. * (a) An ordered set E is said to be *branched* (on the right) if for each $x \in E$ there exist y, z in E such that $x \leq y$, $x \leq z$ and the intervals $[y, \rightarrow [$ and $[z, \rightarrow [$ do not intersect. An antidirected set with no maximal element (Exercise 23) is branched.

(b) Let E be the set of intervals in **R** of the form $[k.2^{-n},(k+1).2^{-n}]$ $(0 \leq k < 2^n)$, ordered by the relation $\supset$. Show that E is antidirected and has no maximal elements.

(c) Give an example of a branched set in which there exists no antidirected cofinal subset (Take the product of the set E defined in (b) with a well-ordered set which contains no countable cofinal subset, and use Exercise 22.)

(d) Give an example of an ordered set E which is not antidirected, but which has an antidirected cofinal subset (Note that an ordinal sum $\sum_{\xi\in E} F_\xi$ contains a cofinal subset isomorphic to E).$_*$

```
Definition branched r :=
  order r & (forall x, inc x (substrate r) ->
    exists y, exists z, gle r x y & gle r x z &
      (forall t, gle r y t -> gle r z t -> False)).

Lemma Exercise1_24a: forall r,
  order r -> anti_directed r ->
  (forall x, inc x (substrate r) -> ~ maximal_element r x)
  -> branched r.
Proof. ir. split. am. ir. rwi (Exercise1_23h H) H0. nin H0.
  cp (H1 _ H2). assert (exists y, glt r x y).  app exists_proof. dneg.
  split. am. ir. nin (equal_or_not x0 x). am. elim (H5 x0). split; au.
  nin H5. nin (H0 _ _ H5). nin H6. exists x0. exists x1. nin H5; nin H6.
  split; au.
Qed.
```

Consider point (b). Let E be the set of all intervals of the form

$$A_{kn} = [k.2^{-n},(k+1).2^{-n}]$$

Consider two intervals $x = A_{kn}$ and $y = A_{lm}$. Assume $n \leq m$ and write $m = n+p$. Write $l = k.2^p + i$ by Euclidean division. We have

$$x = [\frac{k2^p}{2^m}, \frac{(k+1)2^p}{2^m}] \qquad y = [\frac{k2^p+i}{2^m}, \frac{k2^p+i+1}{2^m}]$$

If $i = -1$ or $i = 2^p$, the intersection of the two intervals is reduced to a point, and x and y are non-comparable. If $i < -1$ or if $i > 2^p$, the intersection of x and y is empty, the elements are non-comparable. Finally, if $0 \leq i < 2^p$, y is a subset of x and $x \leq y$. Thus, either the intersection $x \cap y$ contains at most one point, and x and y are non-comparable, or the intersection contains at least two points, and they are comparable. In particular;, in the first case the two intervals $[x,\rightarrow[$ and $[y,\rightarrow[$ are disjoint (since $x \leq z$ and $y \leq z$ implies that z is a subset of the intersection).

Let's show that E is antidirected. Assume first $x < y$. Write $x = A_{kn}$ and $y = A_{k.2^p+l,n+p}$. We have $p > 0$ and $0 \leq l < 2^p$. Take $z = A_{k.2^p+m,n+p}$, where $m = 1$ (if $l = 0$) and $m = 0$ (otherwise). Then $x < z$ and $[y,\rightarrow[$ and $[z,\rightarrow[$ are disjoint. Condition (II) is trivial with $x' = x$ and $y' = y$.

The set E has no maximal element, since if $x = A_{kn}$, the two elemenst $y = A_{2k,n+1}$ and $y' = A_{2k+1,n+1}$ satisfy $x < y$ and $x < y'$. Thus E is thus branched.

Points (c) and (d) remain to do.

10.2 Section 2

Consider an ordered set E, such that, whenever a and b are in E, $a \leq b$ is equivalent to $a \subset b$. Consider a family A_i that has an upper bound S. Then S contains the union U of the family. If this union is in E, it is the least upper bound of the family.

```
Lemma induced_sub_pr1: forall r X x,
  (forall a b, gle r a b -> sub a b) ->
  upper_bound r X x -> sub (union X) x.
Proof. ir. red in H0. ee. red. ir. nin (union_exists H2). ee. cp (H1 _ H4).
  app (H _ _ H5).
Qed.

Lemma induced_sub_pr2: forall r X,
  order r ->
  (forall a b, inc a (substrate r) -> inc b (substrate r) ->
    (gle r a b = sub a b)) ->
  sub X (substrate r) -> inc (union X) (substrate r) ->
  least_upper_bound r X (union X).
Proof. ir. rww least_upper_bound_pr. split.
  red. ee. am. ir. rw H0. app union_sub. app H1. am. ir. rww H0.
  assert (forall a b, gle r a b -> sub a b). ir. wrr H0. order_tac. order_tac.
  app (induced_sub_pr1 H4 H3). nin H3. am.
Qed.

Lemma inc_coarse: forall a b E, inc a E -> inc b E ->
  inc (J a b) (coarse E).
Proof. ir. uf coarse. aw. eee. Qed.
```

1. *Show that, in the set of orderings on a set* E, *the minimal elements (with respect to the ordered relation "*Γ *is coarser than* Γ'*" between* Γ *and* Γ'*) are the total orderings on* E *and that if* Γ *is any ordering on* E, *the graph of* Γ *is the intersection of the graphs of the total orderings on* E *which are coarser than* Γ *(apply Theorem 2 of no. 4). Deduce that every ordered set is isomorphic to a subset of a product of totally ordered sets.*

The key to the exercise is the following result: if E is an ordered set containing two non-comparable elements x and y, we can extend the ordering by putting $a \leq b$ whenever $a \leq x$ and $y \leq b$.

```
Lemma Exercise2_1a: forall r x y,
  order r -> inc x (substrate r) -> inc y (substrate r) ->
  ~ (gle r x y \/ gge r x y) ->
  exists r', order r' & inc (J x y) r' & substrate r = substrate r' & sub r r'.
Proof. ir. set (E:= substrate r).
  set (a := union2 r (Zo (coarse E)(fun z=>  gle r (P z) x & gle r y (Q z)))).
  assert (is_graph a). red. ir. ufi a H3. nin (union2_or H3).
  assert (is_graph r). fprops. app H5. Ztac. ufi coarse H5.  awi H5. ee; am.
  assert (E = substrate a). set_extens. assert (inc (J x0 x0) a). uf a.
  app union2_first.  change (gle r x0 x0). order_tac. substr_tac.
  rwi (inc_substrate_rw x0 H3) H4. ufi a H4; ufi coarse H4.
  nin H4; nin H4; nin (union2_or H4). uf E. substr_tac. Ztac. awi H6;ee;am.
  uf E. substr_tac. Ztac; awi H6;ee;am.
  assert (order a).  red. ee; split;tv. wr H4. ir.
  assert (gle r y0 y0). order_tac. red. uf a. app union2_first.
```

```
  ir. red in H5. red in H6. ufi a H5. ufi a H6. nin (union2_or H5).
  nin (union2_or H6). assert (inc (J x0 z) r). change (related r x0 z).
  app (order_transitivity H H7 H8). red. uf a; inter2tac.
  Ztac. awi H10. red. uf a. app union2_second. clear H8. Ztac. app inc_coarse.
  uf E. substr_tac. ufi coarse H9. awi H9. ee; am. aw. ee.
  assert (gle r x0 y0). am. order_tac. am. Ztac.  awi H9. clear H7.
  red. uf a. app union2_second. Ztac. app inc_coarse. nin H9. uf E. order_tac.
  rw H4. assert (inc (J y0 z) a). am. substr_tac. aw. ee. am.
  nin (union2_or H6).  change (gle r y0 z) in H10. order_tac. Ztac. awi H12.
  nin H12. elim H2. right. change (gle r y x). order_tac.
  ir. red in H5. red in H6. ufi a H5. ufi a H6. nin (union2_or H5).
  nin (union2_or H6). change (gle r x0 y0) in H7. change (gle r y0 x0) in H8.
  order_tac. Ztac. awi H10. change (gle r x0 y0) in H7. ee.
  assert (gle r y y0). order_tac.  assert (gle r y x). order_tac. elim H2. au.
  Ztac. clear H7. awi H9. nin (union2_or H6). ee.
  change (gle r y0 x0) in H7.
  assert (gle r y x0). order_tac. assert (gle r y x). order_tac. elim H2. au.
  Ztac. awi H11. ee. assert (gle r y x).  order_tac. elim H2. au.
  exists  a. ee. am. uf a. app union2_second. Ztac. app inc_coarse. aw.
  ee; order_tac. am. uf a. app union2sub_first.
Qed.
```

We define here the set of orderings on E.

```
Definition set_of_orders x :=
  Zo (powerset (coarse x))(fun z => substrate z = x & order z).

Lemma set_of_orders_pr: forall x z,
  inc z (set_of_orders x) = (substrate z = x & order z).
Proof. ir. uf set_of_orders. app iff_eq. ir. Ztac. am. ir. Ztac. ee.
  app powerset_inc. wr H. app sub_graph_coarse_substrate. fprops.
Qed.
```

Note. In the main text we have defined “coarser” as an ordering between partitions, and *coarser_order* is a lemma that says that this is an ordering. We have defined *coarser_preorder* as an ordering on the set of preorders of E. The exercise considers the ordering induced on the set of orders of the *opposite* ordering of this one. We start by proving directly that this is an ordering.

```
Definition finer_order_order x :=
  inclusion_suborder (set_of_orders x).

Lemma order_finer_order: forall x,
  order (finer_order_order x).
Proof. ir. uf finer_order_order. app subinclusion_is_order.
Qed.

Lemma substrate_finer_order: forall x,
  substrate (finer_order_order x) = set_of_orders x.
Proof. ir. uf finer_order_order. app substrate_subinclusion_order.
Qed.

Hint Rewrite set_of_orders_pr substrate_finer_order : aw.

Lemma related_finer_order: forall x u v,
```

```
  related (finer_order_order x) u v =
  (order u & order v & substrate u = x & substrate v = x & sub u v).
Proof. ir. uf finer_order_order. rw subinclusion_order_rw. aw.
  ap iff_eq; ir; intuition.
Qed.


Lemma related_finer_order1: forall x u v,
  related (finer_order_order x) u v =
  (order u & order v & substrate u = x & substrate v = x &
  forall a b, inc a x -> inc b x -> gle u a b -> gle v a b).
Proof. ir.  uf gle. rw related_finer_order. ap iff_eq. ir. eee.
  ir. uf related. ufi related H6. app H3.
  ir. eee. red. ir. assert (is_graph u). fprops.
  assert (is_pair x0). app H5. assert (J (P x0) (Q x0) = x0). aw.
  wri H7 H4. assert (inc (P x0) x). wr H1.  substr_tac.
  assert (inc (Q x0) x). wr H1. substr_tac.
  ufi related H3. wr H7. ap (H3 _ _ H8 H9 H4).
Qed.

Lemma inc_coarse: forall a b E, inc a E -> inc b E ->
  inc (J a b) (coarse E).
Proof. ir. uf coarse. aw. eee. Qed.
```

A non-total ordering can be extended, hence is not is not maximal; this shows that maximal orderings are total orderings (the statement of the Exercise considers “minimal” orderings, but for the opposite ordering). The converse is obvious.

```
Lemma Exercise2_1b: forall E a,
  maximal_element (finer_order_order E) a = (total_order a & substrate a = E).
Proof. ir. ap iff_eq. ir. nin H. awi H. ee. split. am.
  ir.  nin (p_or_not_p(gle a x y \/ gge a x y)). am.
  nin (Exercise2_1a  H1 H2 H3 H4).
  assert (gle (finer_order_order E) a x0). rw related_finer_order. eee.
  cp (H0 _ H6). elim H4. left. ee. wrr H7. am.

  ir. red. split. aw. nin H. nin H. au. ir. rwi related_finer_order1 H0. ee.
  rw graph_extensionality. ir. ap iff_eq. ir.
  assert (inc u E). wr H3. substr_tac. assert (inc v E). wr H3. substr_tac.
  cp H7; cp H8.  nin H.  wri H2 H7; wri H2 H8. nin (H11 _ _ H7 H8). am.
  cp (H4 _ _ H10 H9 H12). change (gle x u v) in H6. change (gle x v u) in H13.
  assert (u = v). order_tac. rw H14. order_tac.
  ir. app H4. wr H2. substr_tac. wr H2. substr_tac. fprops. fprops.
Qed.
```

We show here that the set of orderings is inductive (for the “finer” ordering). Consider a totally ordered family X (each element being an ordering on E). Let U be the union of the family. This is an ordering. If X is not empty, the substrate of U is E. In the other case, all we need to do is to show that there exists at least ordering on E.

```
Lemma inductive_finer_order: forall E, inductive_set (finer_order_order E).
Proof. ir. red. ir. nin (emptyset_dichot X).
  exists (diagonal E). split. ee. aw. ee. rww substrate_diagonal.
  app diagonal_order. ir. rwi H1 H2. elim (emptyset_pr H2).
  nin H1. cp (H _ H1). awi H2. nin H2.
```

```
  exists (union X).  set (r:=finer_order_order E) in *.
  assert (order r). uf r. app order_finer_order.
  assert (forall a b, inc a (substrate r) -> inc b (substrate r) ->
    (gle r a b = sub a b)). ir. uf r. rw related_finer_order.
  ap iff_eq. ir. ee. am. ufi r H5. awi H5. ufi r H6. awi H6. ee. ir. au.
  assert (is_graph (union X)). red. ir. nin (union_exists H6). ee.
  cp (H _ H8). ufi r H9. awi H9. nin H9. assert (is_graph x). fprops. app H11.
  assert(substrate (union X) = E).
  set_extens. ufi substrate H7. nin (union2_or H7). awi H8. nin H8.
  nin (union_exists H8). ee. cp (H _ H10). ufi r H11. awi H11. nin H11. wr H11.
  substr_tac. am. awi H8. nin H8.  nin (union_exists H8). ee.
  cp (H _ H10). ufi r H11. awi H11. nin H11. wr H11. substr_tac. am.
  wri H2 H7. assert (inc (J x x) y).  order_tac.
  assert (inc (J x x) (union X)). union_tac. substr_tac.
  assert (inc (union X) (substrate r)). uf r. aw. ee. am.
  red. ee; split;tv. rw H7. ir. red. assert (inc (J y0 y0) y).
  wri H2 H8. order_tac. union_tac. ir. red in H8. red in H9.
  nin (union_exists H8). nin (union_exists H9). ee.
  nin H0. awii H14. nin (H14 _ _ H12 H13). ufi r H15. awii H15.
  rwi related_finer_order1 H15. ee.
  change (gle x0 x y0) in H10. assert (inc y0 E).  wr H17. substr_tac.
  assert (inc z E).  wr H17. substr_tac. cp (H19 _ _ H20 H21 H11).
  assert (gle x0 x z). order_tac. red in H23. red in H23. red. union_tac.
  change (gle (induced_order r X) x0 x1) in H15.  ufi r H15. awii H15.
  rwi related_finer_order1 H15. ee.
  change (gle x1 y0 z) in H11. assert (inc x E).  wr H17. substr_tac.
  assert (inc y0 E).  wr H17. substr_tac. cp (H19 _ _ H20 H21 H10).
  assert (gle x1 x z). order_tac. red in H23. red in H23. red. union_tac.
  ir. red in H8. red in H9.
  nin (union_exists H8). nin (union_exists H9). ee.
  nin H0. awii H14. nin (H14 _ _ H12 H13). ufi r H15. awii H15.
  rwi related_finer_order1 H15. ee.
  change (gle x0 x y0) in H10. assert (inc y0 E).  wr H17. substr_tac.
  assert (inc x E).  wr H18. order_tac. cp (H19 _ _ H20 H21 H11). order_tac.
  change (gle (induced_order r X) x0 x1) in H15.  ufi r H15. awii H15.
  rwi related_finer_order1 H15. ee.
  change (gle x1 y0 x) in H11. assert (inc x E).  wr H17. substr_tac.
  assert (inc y0 E).  wr H17. substr_tac. cp (H19 _ _ H20 H21 H10). order_tac.
  cp (induced_sub_pr2 H4 H5 H H8).
  rwi least_upper_bound_pr H9. ee. am. am. am.
Qed.
```

First Corollary to Zorn's lemma says that for any ordering Γ there is a maximal ordering Γ' such that $\Gamma \leq \Gamma'$. This ordering is total, and extends Γ.

```
Lemma order_total_extension: forall r, order r ->
  exists r', total_order r' & substrate r' = substrate r & sub r r'.
Proof. ir. set (E:= substrate r). set (R:= finer_order_order E).
  assert (order R). uf R. app order_finer_order.
  assert (inductive_set R). uf R. app inductive_finer_order.
  assert (inc r (substrate R)). uf R. aw. au.
  nin (inductive_max_greater H0 H1 H2). ufi R H3.  nin H3.
  rwi Exercise2_1b H3. exists x. eee. rwi related_finer_order H4. ee; am.
Qed.
```

We now prove that Γ is the intersection of all total orders Γ' such that $\Gamma \leq \Gamma'$. As in the previous lemma, we don't use the words "finer" nor "coarser". Let x be an element of the

intersection. There is at least one extension Γ_1 of Γ to a total ordering. Thus x is a pair (a,b) of elements of the common substrate. Assume that a and b are uncomparable. We can extend Γ into a total ordering Γ_2 for which $b < a$ (using two lemmas). But $x \in \Gamma_2$ says $a \leq b$, absurd. If $b < a$ (for Γ) we have $b < a$ for Γ_1, absurd. The only remaining case is $a \leq b$.

```
Lemma Exercise2_1c: forall r,
 order r -> r = intersection (Zo (set_of_orders (substrate r))
   (fun r' => total_order r' & sub r r')).
Proof. ir. set (bs:= Zo (set_of_orders (substrate r))
   (fun r' => total_order r' & sub r r')).
  nin (order_total_extension _ H). assert (inc x bs). uf bs. Ztac. aw. ee. am.
  nin H0. am. ee; am. set_extens. app intersection_inc. ex_tac.
  ir. ufi bs H3. Ztac. nin H5. app H6. cp (intersection_forall H2 H1).
  ufi bs H1.  Ztac. clear H1. nin H5.
  assert (is_pair x0). assert (is_graph x). nin H1. fprops. fprops.
  assert (x0 = J (P x0) (Q x0)). aw. awi H4. nin H4.
  assert (inc (P x0) (substrate r)). wr H4. rwi H7 H3. substr_tac.
  assert (inc (Q x0) (substrate r)). wr H4. rwi H7 H3. substr_tac.
  nin (equal_or_not (P x0) (Q x0)). rw H7. rw H11. order_tac.
  nin (p_or_not_p (gle r (Q x0) (P x0) \/ gge r (Q x0) (P x0))). nin H12.
  ee.  red in H12. red in H12. cp  (H5 _ H12). rwi H7 H3. elim H11. order_tac.
  red in H12. red in H12.  rww H7.
  nin (Exercise2_1a H H10 H9 H12). ee.
  nin (order_total_extension H13). ee.
  assert (inc x2 bs). uf bs. Ztac. aw. ee. ue. nin H19. am. ee. am.
  apply sub_trans with x1. am. am. cp (intersection_forall H2 H22).
  cp (H21 _ H14).  rwi H7 H23. elim H11. nin H19. order_tac.
Qed.
```

Consider an ordering on E, and let B be the set of total orderings on E that extend it. Consider the set of functions $B \to E$. If f and g are two such functions, we say $f \leq g$ if for all $i \in B$, $f(i) \leq_i g(i)$, where $\leq_i$ is i considered as an ordering. This gives an ordered set, the product of the total orders of B. For each $x \in E$, we consider the constant function f_x with value x. The previous lemma says $f_x \leq f_y$ if and only if $x \leq y$. Thus $x \mapsto f_x$ is the desired isomorphism (the range is the set of constant functions).

```
Lemma Exercise2_1d: forall r, order r -> exists f, exists g, exists h,
  axioms_product_order f g &
  (forall i, inc i (domain f) -> total_order (V i g)) &
  order_morphism h r (product_order f g).
Proof. ir. cp  (Exercise2_1c H).
  set (bs :=  Zo (set_of_orders (substrate r))
    (fun r' => total_order r' & sub r r')) in H0.
  set (f:= L bs (fun _: Set => (substrate r))).
  set (g:= L bs (fun z: Set => z)).
  assert (axioms_product_order f g). red. uf f; uf g. ee. gprops. gprops. bw.
  bw. ir. ufi bs H1. Ztac.  nin H3. nin H3. bw. bw. ir. ufi bs H1. Ztac.
  awi H2. nin H2. bw.
  assert(forall i, inc i (domain f) -> total_order (V i g)). uf f. uf g.
  bw. ir.  ufi bs H2. Ztac. nin H4. bw.
  set (h := BL (fun x => L bs (fun _:Set => x)) (substrate r) (productb f)).
  nin (order_total_extension H). assert (inc x bs). uf bs. Ztac. aw. ee. am.
  nin H3. am. ee. am. am.
  exists f. exists g. exists h. ee. am. am.
  assert (transf_axioms  (fun x : Set => L bs (fun _ : Set => x))
```

```
    (substrate r) (productb f)).
  red. ir. uf f. rw productb_pr. bw. ee. gprops. tv. ir. bw.  gprops.
  red. eee. uf h. app injective_bl_function. ir.
  cp (f_equal (fun  f=> V x f) H10). simpl in H11. bwi H11. am. am. am.
  ir. rww substrate_product_order. uf h. simpl. bw. ir. aw.
  uf f; uf g. ap iff_eq. ir. ee. app H7. app H7. ir. bwi H11. bw.
  ufi bs H11. Ztac. nin H13. red. app H14.
  bw. ir. ee. red. red. rw Exercise2_1c. fold bs. app intersection_inc.
  ex_tac. ir. cp (H12 _ H13). bwi H14. am. am. am. am. am.
Qed.
```

---

2. *Let* E *be an ordered set and let* $\mathfrak{B}$ *be the set of subsets of* E *which are well-ordered by the inducing ordering. Show that the relation "*X *is a segment of* Y*" on* $\mathfrak{B}$ *is an order relation between* X *and* Y *and that* $\mathfrak{B}$ *is inductive with respect to this order relation. Deduce that there exist well-ordered subsets of* E *which have no strict upper bound in* E*.*

Let's start with some preliminaries.

```
Lemma Exercise2_2a : forall r,
  let B:= Zo (powerset (substrate r)) (fun z=> worder (induced_order r z)) in
    let ss_order := Zo (coarse B)
      (fun z=> is_segment (induced_order r (Q z)) (P z)) in
      order r ->
      (order ss_order & substrate ss_order = B & inductive_set ss_order).
Proof. ir.
  assert (Ha: forall x y,  related ss_order x y =
    (inc x B & inc y B & is_segment (induced_order r y) x)).
  ir. uf ss_order. uf related. ap iff_eq. ir. Ztac. awi H1. awi H2.
  ufi coarse H1. awi H1. eee. ir. ee. Ztac. app inc_coarse. aw.
  assert (Hb: is_graph ss_order). uf ss_order. red. ir. Ztac.  ufi coarse H1.
  awi H1. ee. am.
  assert (Hc: forall x y, related ss_order x y -> sub x y). ir.
  rwi Ha H0. ee. red in H2. ee. awi H2. am. am.  ufi B H1. Ztac.
  rwi powerset_inc_rw H4. am.
  assert (Hd: forall x, inc x B ->gle ss_order x x).
  ir.  rw Ha. eee. red.  ufi B H0. Ztac. rwi powerset_inc_rw H1. ee. aw.
  fprops. ir. nin (related_induced_order3 H4). am.
  assert(He:substrate ss_order = B). set_extens. ufi substrate  H0.
  nin (union2_or H0). awi H1.  nin H1. change (related ss_order x x0) in H1.
  rwi Ha H1. ee. am. am. awi H1. nin H1. change (related ss_order x0 x) in H1.
  rwi Ha H1. ee. am. am. assert (inc (J x x) ss_order). app Hd. substr_tac.
  assert (Hf: forall x, inc x B -> sub x (substrate r)). uf B. ir. Ztac.
  rwi powerset_inc_rw H1. am.
```

Let's show that we have an order. Antisymmetry is trivial since $X \leq Y$ implies $X \subset Y$.

```
  assert (order ss_order). red. ee; split; tv. rw He. exact Hd.
  ir. rwi Ha H0; rwi Ha H1; rw Ha. eee. red in H5; red in H3. ee.
  awii H5. awii H3. red. ee. aw. apply sub_trans with y. am. am.  app Hf.
  ir. cp (H6 _ _ (H5 _ H8) H9).  cp (H5 _ H8).
  assert (gle (induced_order r y) y0 x0). aw. ap (related_induced_order1 H9).
  ap (H7 _ _ H8 H12).  app Hf. app Hf.
  ir. app extensionality. app Hc. app Hc. eee.
```

Let's show that the set $\mathfrak{B}$ is inductive. We consider a totally ordered family X_i. We pretend that its union X is an upper bound. The non trivial point is to show that the union is well-ordered. Consider Y be a nonempty subset of X. Let $a \in Y$, say $a \in X_i$. Let Z be the intersection of Y and X_i and b its least element. Let c be in Y, say $c \in X_j$. We have to show $b \leq c$. This is clear if $c \in X_i$, in particular when $X_j \subset X_i$. If this relation is false, then $X_i \subset X_j$, so that b and c are in X_j, thus are comparable. If $c \leq b$ we get $c \in X_i$, because X_i is a segment of X_j.

```
  red. ir. rwi He H1. assert (sub  (union X) (substrate r)).
  red. ir.  nin (union_exists H3). ee. cp (H1 _ H5). ufi B H6. Ztac.
  rwi powerset_inc_rw  H7. app H7.
  assert (worder (induced_order r (union X))). split. fprops.
  ir. awii H4. nin H5. cp (H4 _ H5). nin (union_exists H6). ee.
  set (Z := intersection2 x x0). assert (nonempty Z). exists y.
  uf Z. app intersection2_inc. cp (H1 _ H8). ufi B H10. Ztac.
  assert (sub Z x0). uf Z.  app intersection2sub_second.
  assert (sub x0 (substrate r)). app Hf.
  assert (sub Z (substrate r)). apply sub_trans with x0; am.
  nin H12. assert (sub Z (substrate (induced_order r x0))). aw.
  nin (H16 _ H17 H9). wrii induced_trans H18. red in H18.  awii H18. nin H18.
  exists x1. wrr induced_trans. red.
  assert (inc x1 x).  ufi Z H18. inter2tac.
  aw. split.  am. ir. aw.  cp (H4 _ H21). nin (union_exists H22). ee.
  nin H2. awii H25. nin (H25 _ _ H24 H8).
  cp ((Hc _ _ (related_induced_order1 H26)) _ H23). assert (inc x2 Z).
  uf Z. app intersection2_inc. cp (H19 _ H28). ap (related_induced_order1 H29).
  red in H26. cp (related_induced_order1 H26). cp ((Hc _ _ H27) _ (H13 _ H18)).
  rwi Ha H27. ee. red in H30. ee. ufi B H29. Ztac. cp (worder_total H33).
  nin H34. awi H35. nin (H35 _ _ H28 H23).  ap (related_induced_order1 H36).
  cp (H31 _ _  (H13 _ H18) H36). assert (inc x2 Z). uf Z. app intersection2_inc.
  cp (H19 _ H38). ap (related_induced_order1 H39). am. app Hf. rww He.
  apply sub_trans with (union X). am. am.
  assert (inc (union X) B). uf B. Ztac. app powerset_inc. eee.

  exists (union X). split. rww He. red. ir. rw Ha. eee. red. aw. ee.
  app union_sub. ir. cp (related_induced_order1 H8).
  nin (related_induced_order3 H8). nin H2. awi H12.
  nin (union_exists H10). nin H13. nin (H12 _ _ H14 H6).
  cp (related_induced_order1 H15). app (Hc _ _ H16).
  cp (related_induced_order1 H15). rwi Ha H16. ee. red in H18. ee.
  assert (gle (induced_order r x0) y0 x). aw. awi H18. app H18. am. app Hf.
  ap (H19 _ _ H7 H20). am. rww He.
Qed.
```

Let E be an ordered set. We apply Zorn's lemma to the ordering defined above. It says that we have a maximal element, i.e., a well-ordered subset X of E. Assume that X has an upper bound x and $Y = X \cup \{x\}$. This is a well-ordered set (adding of a greatest element to a well-ordered set), hence $Y \in \mathfrak{B}$, thus $x \in X$.

```
Lemma Exercise2_2b : forall r, order r ->
  exists x, sub x (substrate r) &  worder (induced_order r x) &
    forall z, upper_bound r x z -> inc z x.
Proof. ir. cp  (Exercise2_2a H).
  set (B:= Zo (powerset (substrate r)) (fun z=> worder (induced_order r z))) in
    H0.
  set (ss_order := Zo (coarse B)
```

```
    (fun z=> is_segment (induced_order r (Q z)) (P z))) in H0.
  ee. nin (Zorn_lemma H0 H2). red in H3. ee. exists x. rwi H1 H3. ufi B H3.
  Ztac. clear H3.  rwi powerset_inc_rw H5. eee. ir.
  set (y:= tack_on x z). assert (sub y (substrate r)). uf y. red. ir.
  rwi tack_on_inc H7. nin H7. app H5. rw H7. nin H3. am.
  cut (gle ss_order x y). ir. wr (H4 _ H8). uf y.  fprops.
  red. red. uf ss_order. Ztac. uf coarse. aw. ee. fprops. uf B. Ztac.
  app powerset_inc. uf B. Ztac. app powerset_inc. red. ee. fprops. aw.
  ir. set (t:= intersection2 x0 x).
  assert (Ha: sub t x). uf t. app intersection2sub_second.
  assert (Hb: sub t y).  apply sub_trans with x0. uf t.
  app intersection2sub_first.  am.
  assert (Hc: sub t (substrate r)). apply sub_trans with y. am. am.
  assert (Hd: sub x0 (substrate r)). apply sub_trans with y. am. am.
  nin (emptyset_dichot t). nin H9. assert (forall u, inc u x0 -> u = z).
  ir. cp (H8 _ H11).  ufi y H11. ufi y H12. rwi tack_on_inc H12. nin H12.
  elim (emptyset_pr (x:=u)). wr H10. uf t. app intersection2_inc. am.
  assert (x0 = singleton z). set_extens. rw (H11 _ H12). fprops. awi H12.
  rw H12. wrr (H11 _ H9). exists z. wrr induced_trans. red. aw. ee.
  rw H12. fprops. ir.  aw. rw (H11 _ H13).  order_tac. nin H3. am. rw H12.
  fprops. nin H6. awii H11.
  nin (H11 _ Ha H10). exists x1. wrii induced_trans H12. nin H12.
  awii H12. awii H13. wrr induced_trans. split. aw. ufi t H12.  inter2tac.
 ir. awii H14. aw. nin (equal_or_not x2 z). rw H15. nin H3. app H16. app Ha.
  assert (inc x2 t).  uf t. app intersection2_inc. cp (H8 _ H14). ufi y H16.
  rwi tack_on_inc H16. nin H16. am. contradiction. cp (H13 _ H16).
  ap (related_induced_order1  H17). ufi t H12. inter2tac.
  aw. red. aw. ee.  uf y. fprops. ir. nin H3.
  nin (related_induced_order3 H9). ufi y H11. rwi tack_on_inc H11. nin H11. am.
  rw H11. cp (related_induced_order1 H9). rwi H11 H13. cp (H10 _ H8).
  assert (z = x0). order_tac. rww H15.
Qed.
```

---

3. *Let* E *be an ordered set. Show that there exist two subsets* A, B, *of* E *such that* $A \cup B = E$ *and* $A \cap B = \emptyset$ *and such that* A *is well-ordered and* B *has no least element (for example, take* B *to be the union of those subsets of* E *which have no least element).* * *Give an example in which there are several partitions of* E *into two subsets having these properties.**

We start with the example. Let N denote the opposite ordering of the set of integers $\mathbf{N}$, and A a finite subset of N. It is well-ordered, and its complementary has no least element.

If B is as indicated and has a least element w, then $w \in W$ for some subset W of B that has no least element, absurd. If W is a nonempty subset of the complement of B, that has no least element, then $W \subset B$ by definition; if $a \in W$ we have $a \in B$ and $a \notin B$, absurd.

```
Lemma Exercice2_3: forall r, order r -> exists B,
  sub B (substrate r) &
  worder (induced_order r (complement (substrate r) B)) &
  (forall y, ~ (least_element (induced_order r B) y)).
Proof. ir.
  set (no_least := fun B =>forall y, ~ least_element (induced_order r B) y).
  set (B:= union (Zo (powerset (substrate r)) no_least)).
  assert (sub B (substrate r)). red. ir.
```

```
  nin (union_exists H0). nin H1. Ztac. rwi powerset_inc_rw H3. app H3.
  exists B. ee. am. assert (sub (complement (substrate r) B) (substrate r)).
  app sub_complement. split. fprops. aw. ir. wrr induced_trans.
  nin (p_or_not_p (exists y, least_element (induced_order r x) y)). am.
  assert (sub x B). red. ir. uf B. apply union_inc with x. am. Ztac.
  app powerset_inc. red. ir. cp (H2 _ H6). srwi H7. ee. am. red. ir.
  dneg. exists y. am. nin H3. cp (H2 _ H3). srwi H6. nin H6. elim H7. app H5.
  ir. red. ir. nin H1. awii H1. awii H2. ufi B H1. nin (union_exists H1).
  nin H3. Ztac. rwi powerset_inc_rw H5. elim (H6 y). split. aw. aw.
  ir. aw. assert (inc x0 B). uf B. union_tac. cp (H2 _ H8). awii H9.
Qed.
```

¶ 4. *An ordered set* F *is said to be* partially well-ordered *if every totally ordered subset of* F *is well-ordered. Show that in every ordered set* E *there exists a partially well-ordered subset which is cofinal in* E *(Consider the set* $\mathfrak{F}$ *of partially well-ordered subsets of* E*, and the order relation "*$X \subset Y$ *and no element of* $Y - X$ *is bounded above by any element of* X*" between* X *and* Y *of* $\mathfrak{F}$*. Show that* $\mathfrak{F}$ *is inductive with respect to this order relation).*

Consider an ordered set E. We have to find F, a partially well-ordered set (for the induced order). This explains why our definition involves the set F and the ordering on E.

Let's introduce the set $\mathfrak{F}$ and its ordering.

```
Lemma Exercice2_4: forall r, order r ->
  let  pworder :=  fun F => forall X,
     sub X F -> total_order (induced_order r X) -> worder (induced_order r X)
    in
    exists F, (pworder F & cofinal_set r F).
Proof. ir. set (FF:= Zo (powerset (substrate r)) pworder).
  set (r' := graph_on (fun x y => sub x y & forall a b, inc a x ->
    inc b (complement y x) -> ~ (gle r b a)) FF).
  set (f:= fun x y => forall a b, inc a x ->
    inc b (complement y x) -> ~ gle r b a).
  assert (forall x y, related r' x y = (inc x FF & inc y FF & sub x y & f x y)).
  ir. uf r'. uf graph_on. uf related. ap iff_eq. ir. Ztac. awi H2. awi H1.
  eee. ir. ee.  Ztac. aw. fprops. aw. eee.
```

We simplify the definition of partially well-ordered, by saying: every non-empty totally set has a a least element.

```
  assert (Hp: forall F, sub F (substrate r) ->  pworder F =
    (forall X, sub X F -> nonempty X -> total_order (induced_order r X)
      -> exists y, least_element (induced_order r X) y)).
  ir. ap iff_eq. ir. cp (H2 _ H3 H5). nin H6.  cp (sub_trans H3 H1).
  awii H7. assert (sub X X). fprops. nin (H7 _ H9 H4).
  wrii induced_trans H10. exists x. am. ir. red. ir. cp (sub_trans H3 H1).
  split. fprops. ir. awi H6. cp (sub_trans H6 H3).
  assert (total_order (induced_order r x)).  nin H4. awii H9. ee.
  cp (sub_trans H8 H1). split. fprops. aw. ir.
  cp (H6 _ H11). cp (H6 _ H12). ufi gge H9. aw. nin (H9 _ _ H13 H14); awi H15.
  au. am. am. right. uf gge.  aw. am. am. nin (H2 _ H8 H7 H9). exists x0.
  wrr induced_trans. am. am.
```

Let's show $F \in \mathfrak{F}$ implies $F \cup \{t\} \in \mathfrak{F}$ (a well-ordered set remains well-ordered by adding an element, if this new order remains total).

```
assert (Hf: forall x F, inc x (substrate r) -> inc F FF ->
  inc (tack_on F x) FF).
ir. ufi FF H2. Ztac. clear H2. rwi powerset_inc_rw H3. rwii Hp H4.
assert (sub (tack_on F x) (substrate r)). red. ir. rwi tack_on_inc H2.
nin H2. app H3. rww H2. uf FF. Ztac. app powerset_inc. rww Hp. ir.
assert (Hs:=sub_trans H5 H2). nin (inc_or_not x X).
nin (equal_or_not X (singleton x)).  exists x. split. aw.  aw. ir. aw.
rwi H9 H10. awi H10. rw H10. order_tac.
set (Y := intersection2 X F). assert (sub Y F). uf Y.
app intersection2sub_second.
nin (emptyset_dichot Y). elim H9. set_extens. cp (H5 _ H12).
rwi tack_on_inc H13. nin H13. elim (emptyset_pr (x:=x0)).
wr H11. uf Y. app intersection2_inc. ue. awi H12. rww H12.
assert (sub Y X). uf Y.  app intersection2sub_first. cp (sub_trans H10 H3).
assert (total_order (induced_order r Y)).
split. fprops. aw. nin H7. awii H14. ir. uf gge. aw. ufi gge H14.
cp (H12 _ H15). cp (H12 _ H16). nin (H14 _ _ H17 H18); awi H19; au.
nin (H4 _ H10 H11 H14).
assert (exists z, inc z X & gle r z x & gle r z x0).
nin H15. awii H15. ufi Y H15. cp (intersection2_first H15). nin H7.
awii H18. ufi gge H18. nin (H18 _ _ H8 H17); awi H19.  exists x. eee.
order_tac. am. am. exists x0. eee. order_tac. order_tac. am. am.
nin H16. ee. exists x1. split. aw. aw. ir. aw. nin H15. awi H20.
cp (H5 _ H19). rwi tack_on_inc H21. nin H21. assert (inc x2 Y).
uf Y. app intersection2_inc. cp (H20 _ H22).
cp (related_induced_order1 H23). order_tac. rww H21. am. am. app H4.
red. ir. cp (H5 _ H9). rwi tack_on_inc H10. nin H10. am. elim H8. ue.
```

Let's show that we have an ordering.

```
assert (order r'). uf r'. app order_from_rel. red. ee; red; ir; ee.
apply sub_trans with y. am. am. ir. srwi H6. nin H6.
nin (inc_or_not b y). app H4. srw. eee. app H3. app H1. srw. eee.
app extensionality. fprops. ir. srwi H4.
nin H4; contradiction. fprops. ir. srwi H4. nin H4; contradiction.
assert (substrate r' = FF). rw substrate_domain_order.
set_extens. awi H2. nin H2. ufi related H0. rwi H0 H2. ee; am. fprops.
aw. exists x. ufi related H0. rw H0. eee. uf f. ir. srwi H4.
nin H4; contradiction. fprops. am.
```

Let's show that $\mathfrak{F}$ is inductive. Consider a totally ordered family X_i of elements of $\mathfrak{F}$, the union X, and a non-empty totally ordered subset Y of X. We have $x \in Y \cap X_i$ for some x and i. Consider the least element y of $K = Y \cap X_i$, (it exists, by definition of X_i). It is the least element of Y. Consider y' in Y. It is in some X_j. If $X_j \subset X_j$, then $y' \in K$ and the conclusion follows. Otherwise we have $y \in X_i$, $y' \in X_j - X_i$, this implies that $y' \leq y$ is false; but since the ordering on Y is total, it implies $y \leq y'$. This argument show $X \in \mathfrak{F}$. From this, it is easy to deduce that X is an upper bound for X_i.

```
assert (inductive_set r'). red. ir. nin H4. awii H5.
assert (sub (union X) (substrate r)).  red. ir.
nin (union_exists H6). nin H7. cp (H3 _ H8). rwi H2 H9. ufi FF H9. Ztac.
rwi powerset_inc_rw H10. app H10.
```

```
  assert (inc (union X) FF). uf FF. Ztac. app powerset_inc. red. ir.
  cp (sub_trans H7 H6). split. nin H8. am. ir. awii H10. nin H11.
  cp (H10 _ H11). cp (H7 _ H12). nin (union_exists H13). nin H14.
  wrr induced_trans.
  set (K:=intersection2 x x0). cp (H3 _ H15). rwi H2 H16. ufi FF H16.
  Ztac. assert (sub K x0). uf K. app intersection2sub_second.
  assert (sub K X0). apply sub_trans with x. uf K. app intersection2sub_first.
  am. cp (sub_trans H20 H9).
  assert (total_order (induced_order r K)). split. fprops. aw. ir. nin H8.
  awii H24. uf gge. aw. cp  (H20 _ H22). cp (H20 _ H23). nin (H24 _ _ H25 H26).
  awii H27. au. ufi gge H27.  awii H27. au. cp (H18 _ H19 H22).
  nin H23. awii H24. assert (sub K K). fprops. assert (nonempty K). exists  y.
  uf K. app intersection2_inc. cp (H24 _ H25 H26). nin H27.
  wrii induced_trans H27. nin H27. awii H27. awii H28. cp (sub_trans H10 H9).
  exists x1. split. aw. ufi K H27. inter2tac. ir. awii H30. aw.
  cp (H10 _ H30). cp (H7 _  H31).  nin (union_exists H32). nin H33.
  nin (inc_or_not x2 x0).  assert (inc x2 K). uf K. app intersection2_inc.
  cp (H28 _ H36). awii H37. nin (H5 _ _ H34 H15). awii H36. rwi H0 H36. ee.
  elim H35. app H38. ufi gge H36. awii H36.  rwi H0 H36. ee. ufi f H39.
  cp (H19 _ H27). assert (inc x2 (complement x3 x0)). srw. ee. am. am.
  cp (H39 _ _ H40 H41). nin H8. awii H43. cp (H20 _ H27). nin (H43 _ _ H31 H44).
  awii H45. contradiction. red in H45. awii H45. ufi K H27. inter2tac.
  exists (union X). split. ue. ir. rw H0. ee. wr H2. app H3. am. app union_sub.
  uf f. ir. srwi H10. nin H10. nin (union_exists H10). nin H12.
  nin (H5 _ _ H8 H13). awii H14. rwi H0 H14. ee. ufi f H17. app H17.
  srw. eee. ufi gge H14.  awii H14. rwi H0 H14. ee. elim H11. app H16.
```

Let F be a maximal element. For no $t \notin F$ we have $F \leq F \cup \{t\}$. This says that F is cofinal in E.

```
  nin (Zorn_lemma H1 H3). nin H4. rwi H2 H4. cp H4. ufi FF H4. Ztac.
  rwi powerset_inc_rw H7. clear H4. exists x. split. am. split. am.
  ir. nin (inc_or_not x0 x). exists x0. ee. am. order_tac.
  app exists_proof. red. ir. assert (gle r' x (tack_on x x0)). rw H0. ee. am.
  app Hf. fprops. uf f. ir. srwi H12. red. ir. elim (H10 a). ee. am.
  rwi tack_on_inc H12. nin H12. contradiction. ue.
  cp (H5 _ H11). elim H9. wr H12. fprops.
Qed.
```

---

5. *Let* E *be an ordered set and let* $\mathfrak{I}$ *be the set of* free *subsets of* E*, ordered by the relation defined in § 1, Exercise 5. Show that, if* E *is inductive, then* $\mathfrak{I}$ *has a greatest element.*

This is a direct consequence of the first corollary to Zorn's lemma.

```
Lemma Exercise2_5: forall r, order r -> inductive_set r ->
  exists x, greatest_element (free_subset_order r) x.
Proof. ir. set (A:= Zo (substrate r) (maximal_element r)).
  assert (free_subset r A). red. ir. ufi A H1. Ztac. nin H5. sy. app H6.
  assert (inc A (set_of_free_subsets r)). uf set_of_free_subsets. Ztac.
  app powerset_inc. uf A. app Z_sub.
  cp (fs_is_order H). cp (substrate_fs_order H).
  exists A. split. ue. rw H4. ir. rw fs_order_pr. red. eee.
  ir. ufi set_of_free_subsets H5. Ztac. rwi powerset_inc_rw H7.
```

```
  nin (inductive_max_greater H H0 (H7 _ H6)).
  exists x1. ee. uf A. clear H5. Ztac. order_tac. am.
Qed.
```

¶ 6. *Let* E *be an ordered set and let* f *be a mapping from* E *into* E *such that* $f(x) \geq x$ *for all* $x \in$ E.

(a) Let $\mathfrak{S}$ *be the set of subsets* M *of* E *with the following properties: (1) the relation* $x \in$ M *implies* $f(x) \in$ M*; (2) if a non-empty subset of* M *has a least upper bound in* E*, then this least upper bound belongs to* M*. For each* $a \in$ E*, show that the intersection* C_a *of the sets of* $\mathfrak{S}$ *which contain* a *also belongs to* $\mathfrak{S}$*; that* C_a *is well-ordered; and that if* C_a *has an upper bound* b *in* E*, then* $b \in C_a$ *and* $f(b) = b$*.* C_a *is said to be the* chain *of* a *(with respect to the function* f*). (Consider the set* $\mathfrak{M}$ *whose elements are the empty set and the subsets* X *of* E *which contain* a *and have a least upper bound* m *in* E *such that* $m \notin$ X *or* $f(m) > m$*, and apply Lemma 3 of no. 3 to the set* $\mathfrak{M}$*.)*

(b) Deduce from (a) that if E *is inductive, then there exists* $b \in$ E *such that* $f(b) = b$*.*

We start with some definitions.

```
Definition Ex2_6S r f :=
  Zo (powerset (substrate r))
  (fun M => (forall x, inc x M -> inc (W x f) M) &
    (forall N x, sub N M -> nonempty N -> least_upper_bound r N x -> inc x M)).

Definition Ex2_6chain r f a :=
  intersection (Zo (Ex2_6S r f) (fun z => inc a z)).

Definition Ex2_6H r f := order r & is_function f & substrate r = source f
  & substrate r = target f & forall x, inc x (substrate r) -> gle r x (W x f).
```

We start with some trivial lemmas.

```
Lemma Exercise2_6a: forall r f, Ex2_6H r f ->
  (inc (substrate r) (Ex2_6S r f)).
Proof. ir. uf Ex2_6S. Ztac. app powerset_inc. fprops. ee. red in H. ee.
  ir. rwi H1 H4. rw H2. fprops. ir. nin H2.  awi H2. Ztac. am.
  nin H. am. app Z_sub.
Qed.

Lemma Exercise2_6b: forall r f a, Ex2_6H r f ->
  inc a (source f) -> nonempty (Zo (Ex2_6S r f) (fun z => inc a z)).
Proof. ir. exists (substrate r). Ztac. app Exercise2_6a.
  red in H. ee. ue.
Qed.

Lemma Exercise2_6c: forall r f a, Ex2_6H r f ->
  inc a (source f) -> inc a (Ex2_6chain r f a).
Proof. ir. uf Ex2_6chain. app intersection_inc. app Exercise2_6b.
  ir. Ztac. am.
Qed.
```

We deduce $C_a \in \mathfrak{S}$. If we correct[1] the third claim as "if C_a has a least upper bound b, then $b \in C_a$ and $f(b) = b$" it becomes trivial.

```
Lemma Exercise2_6d: forall r f a, Ex2_6H r f ->
  inc a (source f) -> inc (Ex2_6chain r f a) (Ex2_6S r f).
Proof. ir. cp (Exercise2_6b H H0). uf Ex2_6S. Ztac. app powerset_inc.
  uf Ex2_6chain. red. ir. nin H1. cp (intersection_forall H2 H1).  Ztac.
  ufi Ex2_6S H4. Ztac. rwi powerset_inc_rw H6. app H6.
  ee. ir. uf Ex2_6chain. app intersection_inc. ir. Ztac. ufi Ex2_6S H4. Ztac.
  nin H7. app H7. ap (intersection_forall H2 H3). ir. uf Ex2_6chain.
  app intersection_inc. ir. Ztac. ufi Ex2_6S H6. Ztac.
  nin H9. assert (sub N y). red. ir. cp (H2 _ H11).
  ufi Ex2_6chain H12. ap (intersection_forall H12 H5). app (H10 _ _ H11 H3 H4).
Qed.

Lemma Exercise2_6e: forall r f a b, Ex2_6H r f ->
  inc a (source f) -> least_upper_bound r  (Ex2_6chain r f a) b ->
  (inc b  (Ex2_6chain r f a) & W b f = b).
Proof. ir. cp (Exercise2_6c H H0). cp (Exercise2_6d H H0).
  set (s:= Ex2_6chain r f a) in *.
  ufi Ex2_6S H3. Ztac. nin H5. assert (sub s s). fprops.
  assert (nonempty s). exists a. am. assert (inc b s). app (H6 _ _ H7 H8 H1).
  split. am. Ztac. rwi powerset_inc_rw H10.
  cp (H5 _ H9). red in H. ee. awii H1. nin H1. nin H1.
  cp (H19 _ H12). cp (H17 _ H1). order_tac.
Qed.
```

Consider a nonempty well-ordered set E. If e is the least element, the segment with end-point e is empty. If the interval $]\leftarrow, a]$ is not the whole set, it is a segment with end-point b.

```
Lemma Exercise2_6f: forall r, worder r ->
  nonempty (substrate r) -> exists x,
    (inc x (substrate r) & segment r x = emptyset).
Proof. ir. nin H.  set (x:= substrate r) in H0.
  assert (sub x (substrate r)). uf x. fprops. cp (H1 _ H2 H0).
  nin H3.  nin H3. awii H3; awii H4. exists x0. ee. am.
  app is_emptyset. ir. red. ir. cp (inc_segment H5).
  assert (inc y x). uf x.  order_tac. cp (H4 _ H7). awii H8. order_tac.
Qed.

Lemma Exercise2_6g: forall r a, worder r -> inc a (substrate r) ->
  let m := Zo (substrate r) (fun z => glt r a z) in
    nonempty m -> exists b,
      inc b (substrate r) & (segment_c r a = segment r b).
Proof. ir. cp (worder_total H). nin H2. clear H2. rename H3 into Ht.
 nin H. assert (sub m (substrate r)). uf m. app Z_sub.
  nin (H2 _ H3 H1). nin H4. awii H4. awii H5. exists  x. split. app H3.
  set_extens. rww segment_rw.  rwii segmentc_rw H6. ufi m H4. Ztac. order_tac.
  app H3. rww segmentc_rw.  rwii segment_rw H6. ufi m H4. Ztac.
  assert (inc x0 (substrate r)). order_tac. nin (Ht _ _ H9 H0). am.
  nin (equal_or_not x0 a). rw H11; rwi H11 H10; am. assert (glt r a x0).
  split. am. au. assert (inc x0 m). uf m. clear H4. Ztac.  cp (H5 _ H13).
  awi H14. order_tac. am. am. app H3.
```

[1] as in the French edition of Bourbaki

```
Qed.
```

Consider now the second claim: C_a is well-ordered. For this purpose we consider some sets. $\mathfrak{M}_0$ is the set all subsets of E that contain a and have a least upper bound. If x is such a set, we have $\sup x \in E$ and $f(\sup x) \in E$. Denote these two quantities by $p_1(x)$ and $p_2(x)$. The set $\mathfrak{M}_1$ contains those x for which $\sup x \notin x$, and the set $\mathfrak{M}_2$ contains those x for which $\sup x < f(\sup x)$. We define $p(x)$ as $p_1(x)$ and $p_2(x)$, whichever applies (we use p_1 for the intersection). The set $\mathfrak{M}$ will be the union of $\mathfrak{M}_1$ and $\mathfrak{M}_2$, to which we adjoin the empty set, with $p(\emptyset) = a$. By construction $p(x) \in E - x$, whenever $x \in \mathfrak{M}$.

```
Lemma Exercise2_6h: forall r f a, Ex2_6H r f ->
  inc a (source f) -> worder (induced_order r  (Ex2_6chain r f a)).
Proof. ir. set (E:= substrate r).
  set(M0 :=  Zo (powerset E) (fun z => inc a z & exists b,
    least_upper_bound r z b)).
  assert (forall z, inc z M0 -> least_upper_bound r z (supremum r z)).
  ir. ufi M0 H1. Ztac. nin H3. nin H4. rwi powerset_inc_rw H2. nin H.
  app supremum_pr1. exists x. am.
  assert (forall x, inc x M0 -> (inc  (supremum r x) (substrate r)
    & inc (W (supremum r x) f) (substrate r))).
  ir. cp (H1 _ H2). red in H. nin H. ufi M0 H2. Ztac.  rwi powerset_inc_rw H5.
  awii H3. nin H3. nin H3. ee. am. rw H11. app inc_W_target. ue.
  set (sif := fun x => glt r x (W x f)).
  set (M1 := Zo M0 (fun z => ~ (inc (supremum r z) z))).
  set (M2 := Zo M0 (fun z => sif (supremum r z))).
  set(M:= tack_on (union2 M1 M2) emptyset).
  set (p:= fun x => Yo (x= emptyset) a
    (Yo (inc x M1)  (supremum r x) (W (supremum r x) f))).
  assert (Ha:sub (union2 M1 M2) M0). uf M1; uf M2. red. ir.
  nin (union2_or H3); Ztac;am.
  assert (forall x, inc x M -> ~ (inc (p x) x)). ir. red. ir. ufi p H4.
  nin (equal_or_not x emptyset). rwi Y_if_rw H4. rwi H5 H4.
  elim (emptyset_pr H4). am. rwii Y_if_not_rw H4. ufi M H3.
  rwi tack_on_inc H3. nin H3. nin (inc_or_not x M1). rwii Y_if_rw H4.
  ufi M1 H6. Ztac. contradiction. nin (union2_or H3). contradiction.
  rwii Y_if_not_rw H4. ufi M2 H7. Ztac. red in H9. cp (H1 _ H8).
  awii H10. nin H10. nin H10. cp (H12 _ H4). nin H. order_tac. nin H;am.
  ufi M0 H8. Ztac. app powerset_sub. contradiction.
  assert (forall x, inc x M -> inc (p x) E). ir. red in H. ee. uf p.
  nin (equal_or_not x emptyset). rww Y_if_rw.  uf E. ue. rww Y_if_not_rw.
  ufi M H4. rwi tack_on_inc H4. nin H4. nin (H2 _  (Ha _ H4)).
  nin (inc_or_not x M1). rww Y_if_rw. rww Y_if_not_rw. contradiction.
  assert (sub M (powerset E)). red. ir. ufi M H5. rwi tack_on_inc H5.
  nin H5. ufi M1 H5; ufi M2 H5; ufi M0 H5. nin (union2_or H5); Ztac.
  Ztac. am. Ztac. am. rw H5. app powerset_inc. app sub_emptyset_any.
  assert (forall x, inc x M -> (inc (p x) E & ~ (inc (p x) x))).
  ir. split. app H4. app H3.
```

We now apply the famous lemma. It asserts the existence of a well-ordered set M. We shall denote by $\leq_M$ the ordering on M, and use the ordering of E for expressions like $x \leq y$ and $\sup(z)$. The first property is $M \notin \mathfrak{M}$. This can be restated as: M is nonempty, and if it has a least upper m, then $m = f(m) \in M$.

The set $S(x)$ of all y such that $y <_M x$ is in $\mathfrak{M}$, and $p(S(x)) = x$. Taking for x the least element of M (it exists, since we have a non-empty well-ordered set) says that $a = p(\emptyset)$ is the

least element of M. Assume $y <_{\mathrm{M}} x$, so that $y \in \mathrm{S}(x)$. Since $\mathrm{S}(x) \in \mathfrak{M}$, it has a least upper bound c (hence $y \leq c$). We have $c \leq p(\mathrm{S}(x))$ by construction, hence $y < x$. This shows that the ordering extends that of E.

```
nin (Zermelo_aux _ H5 H6). nin H7. red in H7. ee.
set (q:= substrate x) in *. assert (inc a q). nin (emptyset_dichot q).
elim H8. uf M. rw H12. fprops. nin (Exercise2_6f H7 H12). nin H13.
cp (H11 _ H13). rwi H14 H15. ufi p H15. rwii Y_if_rw H15. rww H15.
assert (forall a b, gle x a b -> gle r a b). ir.
assert (inc b q). uf q. nin H. order_tac. assert (inc b E). app H9.
nin (equal_or_not a0 b). rw H16. nin H. order_tac.
assert (inc a0 (segment x b)). app segment_inc. split; am.
cp (H11 _ H14). cp (H10 _ H14). ufi M H19. rwi tack_on_inc H19. nin H19.
cp  (H1 _ (Ha _ H19)). nin H; ee.
assert (gle r a0 (supremum r (segment x b))). awii H20. nin H20. nin H20.
app H26.  apply sub_trans with q.  uf q. app sub_segment. am.
ufi p H18. rwi Y_if_not_rw H18. nin (inc_or_not (segment x b) M1).
rwii Y_if_rw H18. wrr H18. rwii Y_if_not_rw H18.
set (c:=supremum r (segment x b)) in *. assert (inc c (substrate r)).
order_tac. cp (H24 _ H27). wr H18. order_tac. red. ir. rwi H26 H17.
elim (emptyset_pr H17). rwi H19 H17. elim (emptyset_pr H17).
```

There are many functions satisfying $c \leq p(\mathrm{S}(x))$. Our function is not too big. Fix y, and let x be the least element such that $y <_{\mathrm{M}} x$ (if this does not exist, then y is the greatest element of M and $y = f(y)$). Then $\mathrm{S}(x)$ is the set of all $t \leq_{\mathrm{M}} y$ and has y as least upper bound. Since this is in $\mathfrak{M}$ we have $y < f(y)$. We chose $p(\mathrm{S}(x)) = p_2(\mathrm{S}(x)) = f(y)$. This shows that M is stable by f.

```
assert (forall x, inc x q -> inc (W x f) q). ir.
set (m:=Zo q (fun z => glt x x0 z)). nin (emptyset_dichot m).
assert (greatest_element x x0). split. am. ir.
cp (worder_total H7). nin H17. nin (H18 _ _ H14 H16).
nin (equal_or_not x0 x1). rw H20; rwi H20 H19; am.
elim (emptyset_pr (x:=x1)). wr H15. uf m. Ztac. split;am. am.
nin H. ee. assert (greatest_element (induced_order r q) x0). split. aw.
aw. ir. aw. app H13. nin H16. app H22. cp (greatest_is_sup H H9 H21).
ufi M H8. nin (inc_or_not (W x0 f) q). am. elim H8.
assert (inc q M0). uf M0. Ztac. app powerset_inc. ee. am. exists x0. am.
app inc_tack_on_y. app union2_second. uf M2. Ztac. uf sif.
wr (supremum_pr2 H H9 H22). split. app H20. ue. ue. dneg. wrr H25.
nin (Exercise2_6g H7 H14 H15). nin H16.
cp (H10 _ H16). cp (H11 _ H16). ufi M H18. rwi tack_on_inc H18. nin H18.
cp (H1 _ (Ha _ H18)).
nin H; nin H7. assert (sub (segment_c x x0) (substrate r)).
apply sub_trans with q. uf q. app sub_segmentc. am.
assert (greatest_element (induced_order r (segment_c x x0)) x0). split.
aw. app inc_bound_segmentc. aw. ir. aw. rwi segmentc_rw H24.
app H13. am. app  inc_bound_segmentc.
cp (greatest_is_sup H H23 H24). rwi H17 H25. cp (supremum_unique H H20 H25).
ufi p H19. rwi Y_if_not_rw H19. nin (inc_or_not (segment x x1) M1).
rwii Y_if_rw H19. ufi M1 H27. Ztac. rwi H26 H29. elim H29. wr H17.
app inc_bound_segmentc.  rwii Y_if_not_rw H19.  nin (union2_or H18).
contradiction. ufi M2 H28. Ztac. rwi H26 H19. rw H19. am.
red. ir. elim (emptyset_pr (x:=x0)). wr H27. wr H17. app inc_bound_segmentc.
elim (emptyset_pr (x:=x0)). wr H18. wr H17. app inc_bound_segmentc. nin H7;am.
```

Consider a non-empty subset N of M that has a least upper bound y. We pretend $y \in q$, proof by contradiction. Let Q be the set of elements $z \in M$ such that $y \leq z$. If Q is empty, we get $y = \sup M$, and the conclusion follows. In the other case, Q has a least element z. We may assume $z \notin N$, so that $N \subset S(z)$, where $S(z) \in \mathfrak{M}$ is the set of elements of M that are $< z$. This set has a least upper bound α with $y \leq \alpha \leq z$. Let's show $\alpha \leq y$. Consider x with $x <_M z$. Assume there is $t \in N$ such that $x < t$. Since $t \leq y$, we get $x \leq y$. If there is no such t, then x is an upper bound of N, and $y \leq x$. Thus $z \leq x$. This contradicts $x <_M z$. As a consequence $y = \alpha$.

We have $S(z) \in \mathfrak{M}_1$, for $\alpha \in S(z)$ implies $\alpha \in M$ thus $\alpha \in Q$, $z \leq \alpha$, $\alpha < z$, absurd. Our definition of p_1 gives now $\alpha = z$. Since $z \in M$, we have $y \in M$.

```
assert (forall N y, sub N q -> nonempty N -> least_upper_bound r N y ->
   inc y q).
nin (worder_total H7). clear H15. rename H16 into Ht.
assert (forall a b, inc a q -> inc b q -> gle r a b \/ glt r b a).
ir. nin H. nin (equal_or_not a0 b). rw H18. left.
order_tac. app H9. nin (Ht _ _ H15 H16). left. app H13. right. split.
app H13. au.
ir. red in H; nin H7. ee. cp (sub_trans H16 H9). awii H18. nin H18.
set (bN:=Zo q (fun z=> gle r y z)).
nin (emptyset_dichot bN).
assert (least_upper_bound r q y). aw. split. split. nin H18; am.
ir. nin H18. assert (exists t, inc t N & glt r y0 t). app exists_proof.
red. ir. elim (emptyset_pr (x:=y0)). wr H26. uf bN. Ztac. app H25.
split. app H9. ir. nin (H15 _ _ (H16 _ H30) H27). am. elim (H29 y1). au.
nin H29. nin H29. nin H30. cp (H28 _ H29). order_tac. ir. app H25.
split. nin H27;am. ir. nin H27. app H29. app H16. nin (inc_or_not y q). am.
elim H8. uf M. app inc_tack_on_y. app union2_first. uf M1. Ztac. uf M0.
Ztac. app powerset_inc. ee. am. exists y. am. wrr (supremum_pr2 H H9 H27).
assert (sub bN q). uf bN. app Z_sub. nin (H19 _ H27 H26). nin H28.
awii H28. awii H29. ufi bN H28. Ztac. clear H28.
assert (forall u, inc u q -> gle r y u -> gle x x0 u). ir.
assert (inc u bN). uf bN. Ztac. cp (H29 _ H33).
ap (related_induced_order1 H34). nin (inc_or_not y q). am.
nin (inc_or_not x0 N). nin H18. cp (H34 _ H33).
assert (x0 = y). order_tac. wrr H36.
assert (sub N (segment x x0)). red. ir. rww segment_rw. split. nin H18.
cp (H35 _ H34). assert (gle r x1 x0). order_tac.
nin (Ht _ _ (H16 _ H34) H30). am. cp (H13 _ _ H38).
assert (x0 = x1). order_tac. rw H40; rwi H40 H38; am. dneg. ue.
assert (nonempty (segment x x0)). nin H17. exists y0. app H34.
cp (H10 _ H30). cp (H11 _ H30). ufi M H36.
assert (inc (segment x x0) (union2 M1 M2)). rwi tack_on_inc H36. nin H36.
am.  rwi H36 H35. nin H35. elim (emptyset_pr H35). clear H36.
ufi p H37. nin (equal_or_not  (segment x x0) emptyset).
rwi H36 H35. nin H35. elim (emptyset_pr H35). rwii Y_if_not_rw H37.
clear H36.
cp (Ha _ H38). ufi M0 H36. Ztac. clear H36. rwi powerset_inc_rw H39.
clear H40.  cp (H1 _ (Ha _ H38)).
set (s:= supremum r (segment x x0)) in *.
assert (gle r y s). awii H36. ee. app H25. nin H36. split. am. ir.
nin H18. cp (H43 _ H42). app H41. app H34.
assert (gle r s x0). awii H36.  nin H36. app H41. split. order_tac. ir.
nin (inc_segment H42). app H13.
assert (~ (inc s (segment x x0))). red. ir. rwi segment_rw H42.
```

```
  assert (inc s q). uf q. order_tac. cp (H28 _ H43 H40). order_tac. am.
  assert (inc (segment x x0) M1). uf M1. Ztac. rwi Y_if_rw H37.
  assert (gle r s y). awii H36. nin H36. app H44. split. order_tac. ir.
  rwi segment_rw H45. assert (inc y0 q). uf q. order_tac.
  nin (p_or_not_p (exists u, inc u N & glt r y0 u)). nin H47. nin H47.
  nin H18. cp (H49 _ H47). nin H48. order_tac.
  assert (forall u, inc u N -> gle r u y0). ir. cp (H15 _ _ (H16 _ H48) H46).
  nin H49. am. elim H47. exists u. au.
  assert (upper_bound r N y0). split. app H9. am. cp (H25 _ H49).
  cp (H28 _ H46 H50). order_tac. am. assert (y =s). order_tac. rw H45; rw H37.
  am. am.
```

The properties shown above can be summarized as: M contains the chain of a. The chain is well-ordered as a subset of a well-ordered sets, with compatible orderings.

```
  assert (sub (Ex2_6chain r f a) q). uf Ex2_6chain. red. ir.
  apply (intersection_forall (y:=q) H16). Ztac. uf Ex2_6S. Ztac.
  app powerset_inc. cp (sub_trans H16 H9). split. ufi E H17. nin H. fprops.
  nin H. aw. ir.  cp (sub_trans H19 H16). cp (sub_trans H21 H9).
  nin H7. nin (H23 _ H21 H20). nin H24. awii H24. awii H25. exists x1.
  wrr induced_trans. split. aw. aw. ir. cp (H25 _ H26). awii H27. aw. app H13.
Qed.
```

Second part of the Exercise. We must show that f has a fixed point. Let P(E) be the property: there is a chain C_a that has a least upper bound. We have shown that this least upper is a fixed point. Conversely, if f has a fixed point b, then the chain of b has a least upper bound b. Thus P(E) is equivalent to the existence of a fixed point of f. Note that P is a consequence of Q(E) that says: every non-empty well-ordered subset of E has a least upper bound. Are we assumed to prove Q? this is unclear.

We give here a trivial argument, unrelated to the first part. Since $a \leq f(a)$, all maximal elements of E are fixed points of f. An inductive set has a maximal element.

```
Lemma Exercise2_6i: forall r f, Ex2_6H r f -> inductive_set r ->
  exists a, inc a (source f) & W a f = a.
Proof. ir.  nin H. nin (Zorn_lemma H H0). nin H2. ee.
  cp (H6 _ H2). cp (H3 _ H7). exists x. wr H4. au.
Qed.
```

¶ 7. *Let* E *be an ordered set and let* F *be the set of all closures (§ 1, Exercise 13) in* E. *Order* F *by putting* $u \leq v$ *whenever* $u(x) \leq v(x)$ *for all* $x \in$ E. *Then* F *has a least element* e, *the identity mapping of* E *onto itself. For each* $u \in$ F, *let* I(u) *denote the set of elements of* E *which are invariant under* u.

(a) Show that $u \leq v$ *in* F *if and only if* I(v) $\subset$ I(u).

(b) Show that if every pair of elements of E *has a greatest lower bound in* E, *then every pair of elements of* F *has a greatest lower bound in* F. *If* E *is a complete lattice, then so is* F *(§ 1, Exercise 11).*

(c) Show that if E *is inductive (with respect to the relation* $\leq$*), then every pair* u, v *of elements of* F *has a least upper bound in* F *(Show that if* $f(x) = v(u(x))$ *and if* $w(x)$ *denotes*

the greatest element of the chain of x, relative to f (Exercise 6), then w is a closure in E *and is the least upper bound of u and v.)*

Let's start with the definitions.

```
Definition set_of_closures r :=
  let E:=substrate r in
  Zo (set_of_functions E E) (fun z=> is_closure (sof_value E E z) r).
Definition closure_ordering r :=
  let E:=substrate r in
    induced_order (function_order E E r) (set_of_closures r).
```

Let's show some trivial properties.

```
Lemma Exercise2_7a: forall r f g, order r ->
  let E:=substrate r in
  gle (closure_ordering r) f g =
  (inc f (set_of_closures r) & inc g (set_of_closures r) &
    forall i, inc i (substrate r) ->
      gle r (W i (sof_value E E f)) (W i (sof_value E E g))).
Proof. ir. assert (sub (set_of_closures r) (substrate (function_order E E r))).
  rw substrate_function_order. uf set_of_closures. app Z_sub. am. tv.
  assert (order (function_order E E r)). app order_function_order.
  uf closure_ordering. app iff_eq. ir.
  cp (related_induced_order1 H2). cp (related_induced_order3 H2). eee.
  rwi function_order_pr H3. eee. am. tv. ir. ee. aw.
  ufi set_of_closures H2. Ztac. ufi set_of_closures H3. Ztac.
  rww function_order_pr. eee.
Qed.

Lemma Exercise2_7b: forall r, order r ->
  (order (closure_ordering r) &
  substrate (closure_ordering r) = (set_of_closures r)).
Proof. ir. uf closure_ordering. cp (refl_equal (substrate r)).
  cp (order_function_order (substrate r) H H0).
  cp (substrate_function_order (substrate r) H H0).
  assert (sub (set_of_closures r)
   (substrate (function_order (substrate r) (substrate r) r))). rw H2.
  uf set_of_closures. app Z_sub. split.  app order_induced_order. aw.
Qed.


Lemma Exercise2_7c: forall r, order r ->
  least_element (closure_ordering r) (corr_value (identity_fun (substrate r))).
Proof. ir. set (E:=substrate r). set (f:= identity_fun E).
  assert (f = sof_value (substrate r) (substrate r) (corr_value f)).
  uf corr_value. uf sof_value. aw.
  assert (inc (corr_value f) (set_of_closures r)). uf set_of_closures.
  Ztac. app inc_set_of_functionsc. uf f. app function_identity.
  wr H0. red. uf f. ee. red; eee.
  app function_identity. red. simpl. ir. rww W_identity. rww W_identity.
  uf E. order_tac. uf E. order_tac.
  ir. rww W_identity. order_tac. ir. repeat  rww W_identity.
  nin (Exercise2_7b H). split. ue. rw H3. ir. rw Exercise2_7a. eee. ir.
  wr H0. uf f. rww W_identity. ufi set_of_closures H4. Ztac.
  set (g:= sof_value (substrate r) (substrate r) x) in *.
  red in H7. ee. app H8. am.
```

```
Qed.
```

Let's consider (a). One implication is trivial, the other one is false. Example. Consider a set with E three elements a, b and c. Define $f(b) = g(b) = b$, $f(c) = g(c) = c$, $f(a) = b$ and $g(a) = c$. We have $I(f) = I(g) = \{b, c\}$. If $a \leq b$ and $a \leq c$ then f and g are closures. If (a) were true, it would imply $f = g$.

```
Lemma Exercise2_7d: forall r f g, order r ->
  let E:=substrate r in
    gle (closure_ordering r) f g ->
    sub (set_of_invariants (sof_value E E g))
    (set_of_invariants (sof_value E E f)).
Proof. ir. rwii Exercise2_7a H0. ee. ufi set_of_closures H0.
  ufi set_of_closures H1. Ztac. clear H1. Ztac. red in H4; red in H5. ee.
  uf set_of_invariants. red. uf E. ir. Ztac. app Z_inc.
  red in H4. red in H5. ee. wri H20 H11. cp (H6 _ H11).
  cp (H2 _ H11). rwi H12 H24. order_tac.
Qed.
```

Consider now (b). The infimum of a family of closures, if it exists, is a closure. We start with a family of two elements.

```
Lemma Exercise2_7e: forall r f g, order r ->
  (forall x y, inc x (substrate r) -> inc y (substrate r) ->
    has_infimum r (doubleton x y))
  -> inc f (set_of_closures r)
  -> inc g (set_of_closures r)
  -> has_infimum (closure_ordering r) (doubleton f g).
Proof. ir.
  assert (Ht:sub (doubleton f g) (set_of_closures r)). red. ir.
  nin (doubleton_or H3); rww H4.
  ufi set_of_closures H1. Ztac. clear H1.
  ufi set_of_closures H2. Ztac. clear H2.
  set (f1:= sof_value (substrate r) (substrate r) f) in *.
  set (g1:= sof_value (substrate r) (substrate r) g) in *. red in H4.
  red in H5. ee. set (E:= substrate r) in *.
  assert (Ha: forall x, inc x E -> inc (W x f1) E). ir. cp (H7 _ H9).
  uf E. order_tac.
  assert (Hb: forall x, inc x E -> inc (W x g1) E). ir. cp (H5 _ H9).
  uf E. order_tac.
  set (hh:= fun z=> inf r (W z f1) (W z g1)).
  assert (forall x, inc x E ->
    (gle r (hh x) (W x f1)  &  gle r (hh x) (W x g1) &
      forall z, gle r z (W x f1) ->  gle r z (W x g1) -> gle r z (hh x))).
  ir. cp (Ha _ H9). cp (Hb _ H9). cp (inf_pr H H10 H11 (H0 _ _ H10 H11)). eee.
  assert (transf_axioms hh E E). red. ir. nin (H9 _ H10). uf E. order_tac.
  assert (Hg: forall x, inc x E -> gle r x (hh x)). ir. nin (H9 _ H11). ee.
  app H14. app H7. app H5.
  assert (forall x y, gle r x y -> gle r (hh x) (hh y)).
  ir. nin H4. nin H2. ee. cp (H21 _ _ H11). cp (H17 _ _ H11).
  assert (inc x E). uf E. order_tac.  assert (inc y E). uf E. order_tac.
  cp (H9 _ H24). cp (H9 _ H25). ee. ap H29. order_tac. order_tac.
  set (h:= BL hh E E).  assert (is_function h). uf h. app bl_function.
  assert (is_closure h r). red. ee. red. eee. ir. uf h. aw. app H11.
  uf E; order_tac. uf E; order_tac.
```

```
  ir. uf h. aw. app Hg. ir. uf h. set (t:= hh x).
  assert (inc t E). uf t. app H10. assert (W x (BL hh E E) = t). aw.
  rw H15. assert (W t (BL hh E E) = hh t). ap W_bl_function. am. am.
  rw H16. clear H16. clear H15. cp (Hg _ H14). assert (gle r (hh t) t).
  cp (H9 _ H14). ee.  cp (H9 _ H13). ee. red in H4. ee.
  cp (H26 _ _ H19). red in H2. ee. cp (H32 _ _ H20). rwi (H8 _ H13) H27.
  rwi (H6 _ H13) H33. fold t in H27.  fold t in H33. ap H21.
  order_tac. order_tac. order_tac.
  assert (sof_value (substrate r) (substrate r) (corr_value h) = h).
  uf sof_value. uf corr_value. aw.
  nin (Exercise2_7b H).
  assert (inc (corr_value h) (substrate (closure_ordering r))). rw H16.
  uf set_of_closures. Ztac. app inc_set_of_functionsc. ue.
  exists (corr_value h). aw. ee. split. am. ir. rw  Exercise2_7a. ee. ue.
  app Ht.  ir. rw H14. uf h. aw.  cp (H9 _ H19). ee.
  nin (doubleton_or H18); rw H23. exact  H20.  exact  H21. am.
  ir. nin H18. assert (inc f (doubleton f g)). fprops. cp (H19 _ H20).
  assert (inc g (doubleton f g)). fprops. cp (H19 _ H22).
  rwi  Exercise2_7a H21.  rwi  Exercise2_7a H23. ee. rww  Exercise2_7a.
  ee. am. ue. ir. cp (H27 _ H28). cp (H25 _ H28). rw H14. uf h. aw.
  cp (H9 _ H28). ee.  app H33. am. am. ue.
Qed.
```

We consider here the case of a general family. We start with preliminaries.

```
Lemma Exercise2_7f: forall r, complete_lattice r ->
  complete_lattice (closure_ordering r).
Proof. ir. red in H. ee. nin (Exercise2_7b H).
  app exercise1_11h. ir. rwi H2 H3.
  set (ic:= fun f => sof_value (substrate r) (substrate r) f).
  assert (forall f, inc f X -> is_closure (ic f) r). ir. cp (H3 _ H4).
  ufi set_of_closures H5. Ztac. uf ic. aw.
  set (iv := fun x => fun_image X (fun z => W x (ic z))).
  assert (forall x, inc x (substrate r) -> sub (iv x) (substrate r)).
  ir. red. ir. ufi iv H6. awi H6. nin H6. nin H6. cp (H4 _ H6). wr H7.
  red in H8. ee. nin H8. ee. rw H14. app inc_W_target.
  set (hh := fun x => infimum r (iv x)).
  assert (forall x, inc x (substrate r) -> (lower_bound r (iv x) (hh x) &
    forall z, lower_bound r (iv x) z -> gle r z (hh x))).
  ir. cp (H5 _ H6). nin (H0 _ H7). uf hh. ap (infimum_pr H H7 H9).
```

Let's show that our family of functions has a greatest lower bound.

```
  assert (transf_axioms hh (substrate r) (substrate r)). red. ir.
  nin (H6 _ H7). nin H8. am.
  set (h:= BL hh (substrate r) (substrate r)).
  assert (is_function h). uf h. app bl_function.
  assert (forall x f, inc x (substrate r) -> inc f X ->
    gle r (W x h) (W x (ic f))). ir.
  nin (H6 _ H9). nin H11. uf h. aw. app H13. uf iv. aw. exists f. au.
  assert (forall x y, inc x (substrate r) ->  inc y (substrate r) ->
    (forall f, inc f X -> gle r y  (W x (ic f))) -> gle r y (W x h)).
  ir. nin (H6 _ H10). uf h. aw. app H14. split. am. ir. ufi iv H15.
  awi H15. nin H15. nin H15. wr H16. app H12.
```

Let's show that the greatest lower bound is a closure.

```
  assert (increasing_fun h r r). red. eee. ir.
  assert (inc x (substrate r)). order_tac.
  assert (inc y (substrate r)). order_tac. app H10.
  uf h. aw. app H7. ir. cp (H9 _ _ H12 H14). cp (H4 _ H14). red in H16.
  ee. red in H16. ee. cp (H23 _ _ H11). order_tac.
  assert (forall x, inc x (substrate r) -> gle r x (W x h)).
  ir. app (H10 _ _ H12 H12). ir. nin (H4 _ H13).  ee. app H15.
  assert (forall x, inc x (substrate r) -> W (W x h) h = W x h). ir.
  set (t := W x h) in *. assert (inc t (substrate r)).
  change (inc  (W x h) (target h)). app inc_W_target.
  assert (gle r t (W t h)). app H12.
  assert (gle r (W t h) t). uf t. app H10.
  change (inc (W (W x h) h) (target h)). app inc_W_target. ir.
  cp (H9 _ _ H13 H16).  cp (H9 _ _ H14 H16). fold t. fold t in H17.
  nin (H4 _ H16). ee. red in H19. ee. cp (H26 _ _ H17).
  rwi (H21 _ H13) H27. order_tac. order_tac.
  assert (is_closure h r). red. eee.
```

The conclusion is now easy.

```
  assert (sof_value (substrate r) (substrate r) (corr_value h) = h).
  uf sof_value. uf corr_value. aw.
  assert (inc (corr_value h) (substrate (closure_ordering r))). rw H2.
  uf set_of_closures. Ztac. app inc_set_of_functionsc. ue.
  exists (corr_value h). aw. ee. split. am. ir. rw Exercise2_7a. ee. ue.
  app H3. ir. rw H15. app H9. am. ir. nin H17. rwi H2 H17.
  rww  Exercise2_7a. ee. am. ue. ir. rw H15.
  set (k:= W i (sof_value (substrate r) (substrate r) z)).
  assert (inc k (substrate r)). ufi set_of_closures H17. Ztac.
  red in H21. ee. red in H21. ee. rw H27. uf k. app inc_W_target.
  app H10. ir.  cp (H18 _ H21). rwi  Exercise2_7a H22. ee. cp (H24 _ H19).
  fold k in H25. am. am. ue.
Qed.
```

We define $I_x(f)$ be the least element y such that $x \leq y$ and $f(y) = y$. We pretend that if this exists for $f = u \circ v$, then the pair of closures (u, v) has a least upper bound. We start with preliminaries.

```
Lemma Exercise2_7g: forall r u v,
  let fp := fun x f => Zo (substrate r) (fun z => gle r x z & W z f = z) in
    let f := compose (sof_value (substrate r) (substrate r) u)
      (sof_value (substrate r) (substrate r) v)
      in  order r -> inc u (set_of_closures r) -> inc v (set_of_closures r)
      -> (forall x, inc x (substrate r) -> exists y,
        least_element (induced_order r (fp x f)) y)
      -> has_supremum (closure_ordering r) (doubleton u v).
Proof. ir.
  assert (Ht:sub (doubleton u v) (set_of_closures r)). red. ir.
  nin (doubleton_or H3); rww H4.
  ufi set_of_closures H0. Ztac. clear H0.
  ufi set_of_closures H1. Ztac. clear H1.
  set (u1:= sof_value (substrate r) (substrate r) u) in *.
  set (v1:= sof_value (substrate r) (substrate r) v) in *. red in H4.
  red in H5. ee. set (E:= substrate r) in *.
  assert (Ha: forall x, inc x E -> inc (W x u1) E). ir. cp (H7 _ H9).
  uf E. order_tac.
```

```
assert (Hb: forall x, inc x E -> inc (W x v1) E). ir. cp (H5 _ H9).
uf E. order_tac.
assert (composable u1 v1). red. red in H4; red in H1; ee. am. am. ue.
assert (is_function f). uf f. fct_tac.
```

Let $g(x) = \text{least}(\mathrm{I}_x)$. If f is any function, and g is well-defined, then g is a closure.

```
set (gg := fun x => the_least_element (induced_order r (fp x f))).
assert (forall x, inc x E -> sub (fp x f) (substrate r)).
ir. uf fp. app Z_sub.
assert (forall x, inc x E -> substrate (induced_order r (fp x f))
  = (fp x f)). ir. aw. app H11.
assert (forall x, inc x E -> least_element (induced_order r (fp x f)) (gg x)).
ir. cp (H2 _ H13). set (F:= induced_order r (fp x f)) in *.
assert (order F). uf F. fprops. ap (the_least_element_pr H15 H14).
assert (transf_axioms gg E E). red. ir. nin (H13 _ H14). rwi H12 H15.
app (H11 _ H14). am.
assert (forall x, inc x E -> gle r x (gg x)). ir.
nin (H13 _ H15). rwi H12 H16. ufi fp H16. Ztac. ee. am. am.
assert (forall x, inc x E -> gg (gg x) = gg x). ir.
nin (H13 _ H16). rwi H12 H17. ufi fp H17. Ztac. ee.
nin (H13 _ H19). rwi H12 H23. rwi H12 H22. ufi fp H22. Ztac. ee.
assert (inc (gg x) (fp (gg x) f)).  uf fp. app Z_inc. eee. order_tac.
cp (H23 _ H27). cp (related_induced_order1 H28). order_tac. am. am. am.
assert (forall x y, gle r x y -> gle r (gg x) (gg y)).
ir. assert (inc x E). uf E. order_tac. assert (inc y E). uf E. order_tac.
cp (H15 _ H19). cp (H13 _ H18). nin H21. rwi  H12 H22.
assert (inc (gg y) (fp x f)). uf fp. Ztac. split. order_tac.
cp (H13 _ H19).  nin H23. rwi (H12 _ H19) H23.  ufi fp H23.  Ztac. ee; am.
cp (H22 _ H23). ap (related_induced_order1 H24). am.
set (g:= BL gg E E). assert (is_function g). uf g. app bl_function.
assert (is_closure g r). split. red. eee. ir.  uf g.  aw. app H17.
uf E. order_tac. uf E. order_tac. ee. ir. uf g. aw. app H15. ir.
uf g. aw. app H16. aw. app H14.
```

Showing that g is the least upper bound is now straightforward.

```
assert (sof_value (substrate r) (substrate r) (corr_value g) = g).
uf sof_value. uf corr_value. aw.
nin (Exercise2_7b H).
assert (inc (corr_value g) (set_of_closures r)). uf set_of_closures.
Ztac. app inc_set_of_functionsc. ue. exists (corr_value g).
aw. ee. split. rww H22. ir. rw  Exercise2_7a. ee. app Ht. am.
ir. rw H20. uf g. aw. set (a := gg i). cp (H15 _ H25). fold a in H26.
assert (gle r (W i v1) (W a v1)). red in H1. ee. app H31.
assert (gle r (W (W i v1) u1) (W (W a v1) u1)). red in H4. ee. app H32.
assert ((W (W a v1) u1) = a).  cp (H13 _ H25). nin H29. rwi H12 H29.
ufi fp H29. Ztac. nin H32. fold a in H33. ufi f H33. awi H33. am. am.
am. am. rwi H29 H28. assert (inc (W i v1) E). uf E; order_tac.
cp (H7 _ H30).  assert (gle r (W i v1) a). order_tac.
cp (H5 _ H25). assert (gle r i (W a v1)). order_tac.
assert (gle r (W i u1) (W (W a v1) u1)). red in H4. ee. app H39.
rwi H29 H35. nin (doubleton_or H24); rw H36. am. am. am.
ir. nin H24. rw Exercise2_7a. ee. am. ue. ir. rw H20.
assert (inc u (doubleton u v)). fprops. cp (H25 _ H27).
assert (inc v (doubleton u v)). fprops. cp (H25 _ H29).
```

```
  rwi  Exercise2_7a H28. rwi  Exercise2_7a H30. ee. uf g. aw.
  set (z1:= sof_value (substrate r) (substrate r) z) in *.
  set (j := W i z1). assert (inc j E). uf E. cp (H34 _ H26). order_tac.
  ufi set_of_closures H31. Ztac. clear H31. fold z1 in H37. red in H37. ee.
  fold E in H34. fold u1 in  H34.  fold E in H32. fold v1 in H32.
  cp (H7 _ H35). cp (H5 _ H35). cp (H34 _ H35). cp (H32 _ H35).
  cp (H38 _ H26). fold j in H43. rwi H43 H41; rwi H43 H42.
  assert (j = W j u1). order_tac.  assert (j = W j v1). order_tac.
  assert (j = W j f). uf f. aw. wr H45; wrr H44.
  cp (H13 _  H26). nin H47.  rwi (H12 _ H26) H48.
  assert (inc j (fp i f)). uf fp. Ztac. ee. ap (H37 _ H26). sy; am.
  cp (H48 _ H49). ap (related_induced_order1 H50). am. am. am. ue.
Qed.
```

Consider two closures u and v. Assume that w is an upper bound of u and v. We have then $\mathrm{I}(w) \subset \mathrm{I}(u) \cap \mathrm{I}(v)$. We shall construct in the sequel an example where $\mathrm{I}(u) \cap \mathrm{I}(v)$ is empty, and in this case there is no upper bound. If f is as above, we have $\mathrm{I}(f) \subset \mathrm{I}(u) \cap \mathrm{I}(v)$, and the previous exercise says that $\mathrm{I}(f)$ is non-empty whenever E is inductive. It also says that, for any x, there is y, a fixed point of u and v such that $u(x) \leq y$ and $v(x) \leq y$. Define $w(x) = y$. Since there are many choices for y this function is unlikely to satisfy the desired properties. If we choose y maximal, we have $w(y) = y$. This same holds if we choose y to be the least element that satisfies the property. However, if there is no least element, we have to choose y randonly, and w is unlikely to be increasing.

We give now a counter-example to (c). For simplicity, we choose a totally ordered set. Here inductive means that the set has a greatest element. Conversely, a set that has a greatest element is inductive. We show that the ordinal sum of two sets where the second has a grestest element is inductive.

```
Lemma Exercice2_7A1: forall r,
  (exists u, greatest_element r u) -> inductive_set r.
Proof. ir. red. ir. nin H. exists x. nin H. split. am. ir. app H2. app H0.
Qed.

Lemma Exercice2_7A2: forall r r', order r  -> order r' ->
  (exists u, greatest_element r' u)
  -> inductive_set  (ordinal_sum2 r r').
Proof. ir. app Exercice2_7A1. nin H1. exists (J x TPb).
  assert (inc (J x TPb) (canonical_du2 (substrate r) (substrate r'))).
  rw canonical_du2_pr.  aw. split. fprops. right. split. nin H1. am. tv.
  split. rww substrate_ordinal_sum2. ir. rwii substrate_ordinal_sum2 H3.
  rww related_ordinal_sum2_order. ee. am. am.
  rwi canonical_du2_pr H3. ee. nin H4. right. right. aw. ee. am. fprops.
  right. left. aw. nin H4. rw H5. ee. fprops. fprops. nin H1. app H6.
Qed.
```

We define here out set $\mathrm{E} = \mathrm{N}_1 + \mathrm{N}_2$ and show that it is inductive. Here N_1 is the set of natural integers, and N_2 the set of natural integers with the reverse ordering.

```
Definition NNstar :=
  ordinal_sum2 Bnat_order (opposite_order Bnat_order).

Lemma Exercice2_7A3:
  order NNstar & substrate NNstar =  canonical_du2 Bnat Bnat
  & (forall x x', gle NNstar x x' =
```

```
    (inc x (canonical_du2 Bnat Bnat) &
    inc x' (canonical_du2 Bnat Bnat) &
    ((Q x = TPa & Q x' = TPa & cardinal_le (P x) (P x'))
      \/ (Q x <> TPa & Q x' <> TPa & cardinal_le (P x') (P x))
      \/ (Q x = TPa & Q x' <> TPa)))).
Proof. ir. uf NNstar.
  cp substrate_Bnat_order. nin worder_Bnat_order. clear H1.
  cp (opposite_is_order H0).  cp (substrate_opposite_order H0).  rwi H H2.
  ee. app order_ordinal_sum2. rww substrate_ordinal_sum2. rw H. rww H2.
  ir. rww related_ordinal_sum2_order. rw H; rw H2. rww opposite_gle.
  rw related_Bnat_order. rw related_Bnat_order. uf Bnat_le.
  app iff_eq; ir; eee. nin H5.  left. ee; am. nin H5. right; left; ee; am.
  right;  right; ee; am.
  assert (inc (P x) Bnat). rwi canonical_du2_pr H3. nin H3. nin H6;ee; am.
  assert (inc (P x') Bnat). rwi canonical_du2_pr H4. nin H4. nin H7;ee; am.
  nin H5. left. eee. nin H5. right; left; eee. right; right; eee.
Qed.

Lemma Exercice2_7A4: inductive_set NNstar.
Proof. uf NNstar. nin (worder_Bnat_order). app  Exercice2_7A2.
  app opposite_is_order. exists card_zero. app least_reverse.
  cp substrate_Bnat_order.  split. rw H1. fprops. ir.
  rw related_Bnat_order.  red. rwi H1 H2. eee.
Qed.
```

If f is a function $N_1 \mapsto N_1$ it can be extended to E by putting $f(x) = x$ on N_2. If we have a closure, then the extension is a closure.

```
Definition extension_to_NNstar f :=
  BL (fun z=> Yo (Q z = TPa) (J (W (P z) f) TPa) z)
  (substrate NNstar) (substrate NNstar).

Lemma Exercice2_7A5: forall f, function_prop f Bnat Bnat ->
  ((forall x, inc x Bnat ->
    W (J x TPa) (extension_to_NNstar f) = (J (W x f) TPa)) &
  (forall x, inc x (substrate NNstar) -> Q x = TPb ->
    W x (extension_to_NNstar f) = x) &
  function_prop (extension_to_NNstar f) (substrate NNstar) (substrate NNstar)).
Proof. ir. red in H. destruct Exercice2_7A3 as [_ [H1 _]]. rw H1.
  assert (transf_axioms (fun z=> Yo (Q z = TPa) (J (W (P z) f) TPa) z)
    (substrate NNstar) (substrate NNstar)). rw H1. red. ir.
  cp H0. rwi canonical_du2_pr H0. nin H0. nin H3; nin H3. rw Y_if_rw.
  rw canonical_du2_pr.  aw. split. fprops. left. split. ee. wr H6.
  app inc_W_target. ue. tv. am. rw Y_if_not_rw. am. rw H4. fprops.
  ee. ir. uf extension_to_NNstar. aw. rww Y_if_rw. rww H1.
  rww canonical_du2_pr. aw. split. fprops. au.
  ir. uf extension_to_NNstar. aw. rw H5. rww Y_if_not_rw. fprops. ue.
  red. eee. uf extension_to_NNstar. app bl_function.
Qed.


Lemma Exercice2_7A6: forall f, is_closure f Bnat_order ->
  is_closure  (extension_to_NNstar f) NNstar.
Proof. ir. red in H. ee.  red in H. cp substrate_Bnat_order. ee.
  assert (function_prop f Bnat Bnat). red. eee.
  cp (Exercice2_7A5 H8). ee. red in H11. ee. cp Exercice2_7A3. ee.
  red. ee. red. eee. ir. rwi H16 H17. ee. rw H16. ee. wr H15.
```

```
  wr H13. app inc_W_target. rww H12. ue. wr H15. wr H13. app inc_W_target.
  rww H12. ue. nin H19. left. rwi canonical_du2_pr H17. nin H17. nin H20.
  nin H20. assert (x = J (P x) (Q x)). aw. rw H22. rw H21. rw (H9 _ H20).
  rwi canonical_du2_pr H18. nin H18. nin H23. nin H23.
  assert (y = J (P y) (Q y)). aw. rw H25. rw H24. rw (H9 _ H23). aw. ee.
  tv. tv. assert (gle Bnat_order (P x) (P y)). rw related_Bnat_order.
  red. eee. cp (H7 _ _ H28).  rwi related_Bnat_order H29. red in H29. ee; am.
  nin H23. nin H19. nin H25. rwi H25 H24. assert (TPa <> TPb). fprops.
  contradiction.  nin H20. nin H19. rwi H19 H21. assert (TPa <> TPb). fprops.
  contradiction. nin H19. right. left.
  assert (W x (extension_to_NNstar f) = x). app H10. ue.
  rwi  canonical_du2_pr H17.  nin H17. nin H20. ee. contradiction. ee; am.
  assert (W y (extension_to_NNstar f) = y). app H10. ue.
  rwi  canonical_du2_pr H18. nin H18. nin H21. ee. contradiction. ee; am.
  rw H20. rw H21. am. right. right.
  assert (W y (extension_to_NNstar f) = y). app H10. ue.
  rwi  canonical_du2_pr H18. nin H18. nin H20. ee. contradiction. ee; am.
  rw H20. rwi canonical_du2_pr H17. nin H17. nin H21.
  nin H21. assert (x = J (P x) (Q x)). aw. rw H23. rw H22. rw (H9 _ H21). aw.
  eee.  assert (TPa <> TPb). fprops. nin H21. nin H19. rwi H19 H23.
  contradiction.
  ir. assert (Ht:= H17). rw H16. ee. ue. wr H15. wr H13. app inc_W_target.
  rwi H15 H17.  rwi canonical_du2_pr H17. ee. nin H18. nin H18.
  assert (x = J (P x) (Q x)). aw. left. split. am. rw H20. rw H19.
  rw (H9 _ H18). aw. split. tv.
  assert (inc (P x) (substrate Bnat_order)). ue.  cp (H0 _ H21).
  rwi related_Bnat_order H22. red in H22. ee. am. right. left.
  assert (W x (extension_to_NNstar f) = x). app H10. nin H18;am. rw H19.
  nin H18. rw H20. ee. fprops. fprops. fprops.
  ir.  cp H17. rwi H15 H18. rwi  canonical_du2_pr H18. nin H18. nin H19.
  nin H19.  assert (x = J (P x) (Q x)). aw. rw H21. rw H20.
  rw (H9 _ H19). assert (inc (W (P x) f) Bnat). red in H8. ee. wr H23.
  app inc_W_target. rww H22. rw (H9 _  H22). rww H1. ue.
  assert (W x (extension_to_NNstar f) = x). app H10. nin H19;am. rw H20. am.
Qed.
```

We define here some properties of even and odd numbers. We say that a number is even if it is of the form $2n$ and odd otherwise. More precisely, we consider the remainder of the division of p by 2; if the remainder r is 0, the number is even. We have $r = 0$ or $r = 1$, so that $r = 1$ for an odd number. We prove here directly that $2n + 1 \neq 2m$.

```
Lemma even_odd_aux: forall n m,
  inc n Bnat -> inc m Bnat ->
  succ (card_mult n card_two) <> (card_mult m card_two).
Proof. ir. red. ir.
  assert (forall a, inc a Bnat -> cardinal_le (succ a) a -> False).
  ir. bwi H2. rwi lt_n_succ_le0 H3. nin H3. elim H4. tv. nin H2. am. am.
  assert (inc card_two Bnat). fprops. cp (card_two_not_zero).
  assert (cardinal_le card_two card_two). fprops.
  assert (is_cardinal n). fprops. assert (is_cardinal m). fprops.
  nin (cardinal_le_total_order2 H7 H6).
  cp (product_increasing2 H8 H5). wri H1 H9.
  assert (inc (card_mult n card_two) Bnat). fprops. app (H2 _ H10 H9).
  wri lt_n_succ_le0 H8. cp (product_increasing2 H8 H5). wri H1 H9. ufi succ H9.
  rwi cardinal_distrib_prod_sum2 H9. set (a:= card_mult n card_two) in *.
  awi H9. rwi card_two_pr H9. rwi card_plus_associative H9.
```

```
  set (b:= card_plus a card_one) in *. change (cardinal_le (succ b) b) in H9.
  assert (inc a Bnat). uf a. fprops. assert (inc b Bnat). uf b. fprops.
  app (H2 _ H11 H9). fprops. am. fprops.
Qed.
Lemma even_odd_aux1: forall n m,
  inc n Bnat -> inc m Bnat ->
  succ (card_mult card_two n) <> (card_mult card_two m).
Proof. ir. rw card_mult_commutative. rw (card_mult_commutative card_two m).
  app  even_odd_aux.
Qed.

Lemma less_than_two: forall a, cardinal_lt a card_two ->
  a = card_zero \/ a = card_one.
Proof. ir. rwi card_two_pr H. change (cardinal_lt a (succ card_one)) in H.
  assert (is_finite_c card_one). fprops. rwi (lt_is_le_succ a H0) H.
  nin (equal_or_not a card_one). au. assert (cardinal_lt a card_one).
  split; am. wri succ_zero H2. assert (is_finite_c card_zero). fprops.
  rwi (lt_is_le_succ a H3) H2. rw (zero_smallest2 H2). au.
Qed.
```

We define now even and odd integers, and show that adding one exchanges the property.

```
Definition even_int n := inc n Bnat & card_rem n card_two = card_zero.
Definition odd_int n := inc n Bnat & ~ (even_int n).


Lemma even_odd_succ: forall n,
  (even_int n -> odd_int (succ n)) & (odd_int n -> even_int (succ n)).
Proof. ir. assert (inc card_two Bnat). fprops.  cp (card_two_not_zero).
  split. ir. red in H1. ee. assert (inc (succ n) Bnat). fprops. split. am.
  red. ir. red in H4. ee.  cp (Bnat_division H1 H H0).
  cp (Bnat_division H3 H H0). ee. rwi H2 H11. rwi H5 H9.
  red in H11. red in H9. ee.
  set (q1:= card_quo n card_two) in H11.
  set (q2:= (card_quo (succ n) card_two)) in H9. awi H9. awi H11. rwi H11 H9.
  cp (even_odd_aux1  H10 H8).  elim H14. am. fprops. fprops.
  ir. nin H1. cp (Bnat_division H1 H H0). ee. red in H5. ee.
  ufi even_int H2. set (r := card_rem n card_two). assert (r <> card_zero).
  dneg. au. fold r in H6. nin (less_than_two H6). contradiction.
  fold r in H5. set (q:= card_quo n card_two) in H5. rwi H8 H5.
  split. fprops.
  assert (inc (succ n) Bnat). fprops. cp (Bnat_division H9 H H0). ee.
  assert (division_prop (succ n) card_two (succ q) card_zero).
  red. ee. aw. uf succ. rw H5. rw cardinal_distrib_prod_sum3. aw.
  wr card_plus_associative. wr card_two_pr. tv. fprops. fprops.
  split. app zero_smallest. fprops. fprops. assert (inc (succ q) Bnat). fprops.
  nin (division_unique H9 H H11 H10 H14 inc0_Bnat H0 H12 H13). am.
Qed.
```

Let $v(x)$ be the function defined on $\mathbf{N}$ by: if x is even, then $v(x) = x$, otherwise $v(x) = x+1$, and u the same function with "even" replaced by "odd". These are closures. They have no common fied point (this no upper bound in the set of closures of $\mathbf{N}$). We show here a simple property: there is no x such that $u(x) \leq x$ and $v(x) \leq x$.

```
Lemma Exercice2_7A7:
```

```
  is_closure (BL (fun z => Yo (even_int z) z (succ z)) Bnat Bnat) Bnat_order.
Proof. cp substrate_Bnat_order. nin worder_Bnat_order.
  assert (transf_axioms (fun z => Yo (even_int z) z (succ z)) Bnat Bnat).
  red. ir. nin (p_or_not_p  (even_int c)). rww Y_if_rw. rww Y_if_not_rw.
  fprops. red. ir. ee. red. eee. app bl_function. ir.
  rwi related_Bnat_order H3. red in H3. ee. rw related_Bnat_order. red.
  aw. ee. app H2. app H2. nin (p_or_not_p (even_int x)). rww Y_if_rw.
  nin (p_or_not_p (even_int y)). rww Y_if_rw. rww Y_if_not_rw.
  assert (is_cardinal y). fprops. cp (is_less_than_succ H8). co_tac.
  rww Y_if_not_rw. nin (p_or_not_p (even_int y)). rww Y_if_rw.
  assert (cardinal_lt x y). split. am. dneg. ue. srw. fprops. fprops.
  rww Y_if_not_rw. wrr lt_n_succ_le1. fprops. fprops. ir.
  rw related_Bnat_order. red. rwi H H3. aw. ee. am. app H2.
  nin (p_or_not_p  (even_int x)). rww Y_if_rw. fprops. rww Y_if_not_rw.
  app is_less_than_succ.  fprops. ir. rwi H H3.
  set (t:= W x (BL (fun z : Set => Yo (even_int z) z (succ z)) Bnat Bnat)).
  assert (t = Yo (even_int x) x (succ x)).  uf t. aw.
  nin (p_or_not_p (even_int x)). rwi Y_if_rw H4. rww H4.  am.
  rwi Y_if_not_rw H4. rw H4. aw. rw Y_if_rw.  tv.
  nin (even_odd_succ x). app H7. split. am. am. fprops. am.
Qed.

Lemma Exercice2_7A8:
  is_closure (BL (fun z => Yo (even_int z) (succ z) z) Bnat Bnat) Bnat_order.
Proof. cp substrate_Bnat_order. nin worder_Bnat_order.
  assert (transf_axioms (fun z => Yo (even_int z) (succ z) z) Bnat Bnat).
  red. ir. nin (p_or_not_p  (even_int c)). rww Y_if_rw. fprops.
  rww Y_if_not_rw.
  red. ir. ee. red. eee. app bl_function. ir.
  rwi related_Bnat_order H3. red in H3. ee. rw related_Bnat_order. red.
  aw. ee. app H2. app H2. nin (p_or_not_p (even_int x)). rww Y_if_rw.
  nin (p_or_not_p (even_int y)). rww Y_if_rw. wrr lt_n_succ_le1.
  fprops. fprops. rww Y_if_not_rw. assert (cardinal_lt x y). split. am. dneg.
  ue. srw. fprops. fprops.
  rww Y_if_not_rw. nin (p_or_not_p (even_int y)). rww Y_if_rw.
  assert (is_cardinal y). fprops. cp (is_less_than_succ H8). co_tac.
  rww Y_if_not_rw. fprops.
  ir. rw related_Bnat_order. red. rwi H H3. aw. ee. am. app H2.
  nin (p_or_not_p  (even_int x)). rww Y_if_rw. app is_less_than_succ.
  fprops. rww Y_if_not_rw.  fprops. ir. rwi H H3.
  set (t:= W x (BL (fun z : Set => Yo (even_int z) (succ z) z) Bnat Bnat)).
  assert (t = Yo (even_int x) (succ x) x).  uf t. aw.
  nin (p_or_not_p (even_int x)). rwi Y_if_rw H4. rw H4. aw.  rww Y_if_not_rw.
  nin (even_odd_succ x). nin (H6 H5). am. fprops. am.
  rwi Y_if_not_rw H4. rww H4. am.
Qed.

Lemma Exercice2_7A9: forall x w,
  let u :=BL (fun z => Yo (even_int z) (succ z) z) Bnat Bnat in
    let v :=BL (fun z => Yo (even_int z) z (succ z)) Bnat Bnat in
      inc x Bnat ->
      cardinal_le (W x u) w ->
      cardinal_le (W x v) w ->
      x <> w.
Proof. ir. cp H. bwi H. nin H.
  cp (is_less_than_succ H). dneg. ufi u H0. ufi v H1.
```

```
  nin (p_or_not_p (even_int x)). awi H0.
  rwi Y_if_rw H0. wri H5 H0. co_tac. am. red. ir.
  nin (p_or_not_p (even_int c)). rww Y_if_rw. fprops. rww Y_if_not_rw. am.
  awi H1. rwi Y_if_not_rw H1. wri H5 H1. co_tac. am. red. ir.
  nin (p_or_not_p (even_int c)). rww Y_if_rw. rww Y_if_not_rw. fprops. am.
Qed.
```

We construct now our counter example, by considering $E = N_1 + N_2$, and the two functions u' and v', the extensions of u and v.

```
Lemma Exercice2_7A10: exists r, exists u, exists v,
  order r & inductive_set r &
  inc u (set_of_closures r) & inc v (set_of_closures r)
  & ~  has_supremum (closure_ordering r) (doubleton u v).
Proof. cp Exercice2_7A9. simpl in H.
  cp Exercice2_7A7. cp Exercice2_7A8.
  set (u :=BL (fun z => Yo (even_int z) (succ z) z) Bnat Bnat) in *.
  set (v :=BL (fun z => Yo (even_int z) z (succ z)) Bnat Bnat) in *.
  cp (Exercice2_7A6 H0). cp (Exercice2_7A6 H1).
  set (u1:=  extension_to_NNstar u) in *.
  set (v1:=  extension_to_NNstar v) in *.
  exists NNstar.  exists (corr_value u1). exists (corr_value v1).
  nin Exercice2_7A3. cp Exercice2_7A4. split. am. split. am.
  assert (forall w, is_closure w NNstar ->
    inc (corr_value w) (set_of_closures NNstar)). ir.
  assert (w = sof_value (substrate NNstar) (substrate NNstar) (corr_value w)).
  nin H7. red in H7. ee.
  assert (w = sof_value (source w) (target w) (corr_value w)).
  uf corr_value. uf sof_value. aw. sy. app corr_propc.
  wri H12 H16; wri H13 H16. am. uf set_of_closures. Ztac. nin H7. red in H7. ee.
  app inc_set_of_functionsc. ue. cp (H7 _ H2).  cp (H7 _ H3). eee.
  red. ir. nin (Exercise2_7b H4).
```

Assume that the pair has a supremum. There is a closure w on $N_1 + N_2$ that is an upper bound for u' and v'.

```
  wri H13 H8; wri H13 H9. cp (sup_pr H12 H9 H8 H11).
  set (su := sup (closure_ordering NNstar) (corr_value u1) (corr_value v1))
    in *. ee. rwi  Exercise2_7a H14.  rwi  Exercise2_7a H15.
  ee.
  assert (u1 =sof_value (substrate NNstar) (substrate NNstar) (corr_value u1)).
  uf corr_value. uf sof_value. aw.
  assert (v1 =sof_value (substrate NNstar) (substrate NNstar) (corr_value v1)).
  uf corr_value. uf sof_value. aw.  wri H21 H20. wri H22 H18.
  ufi set_of_closures H19. Ztac. clear H19.
  set (w:= sof_value (substrate NNstar) (substrate NNstar) su) in *.
  red in H24. ee.
```

We have $w(x) \in E_2$ whenever $x \in E_1$. Proof by contradiction; otherwise $w(y) = y$ is an upper bound in E_1 of $u(y)$ and $v(y)$, contradicting 2_7A9.

```
  assert (function_prop u Bnat Bnat). red. ee. red in H1. ee. red in H1. ee.
  am. tv. tv. nin (Exercice2_7A5  H26).
  assert (function_prop v Bnat Bnat). red. ee. red in H0. ee. red in H0. ee.
  am. tv. tv. nin (Exercice2_7A5  H29).
```

```
assert (forall x, inc x Bnat -> Q (W (J x TPa) w) = TPb). ir.
assert (inc  (J x TPa) (substrate NNstar)). rw H5.
rw canonical_du2_pr. ee. fprops. aw. au.
assert (inc (W (J x TPa) w) (substrate NNstar)). red in H19. ee. rw H39.
app inc_W_target. rwi H5 H34. rwi canonical_du2_pr H34. ee. nin H35. ee.
assert (Hr:=H25 _ H33).
set (y := P (W (J x TPa) w)). assert (J y TPa = W (J x TPa) w).
app pair_extensionality. fprops. aw. aw. sy; am. wri H39 Hr.
assert (inc (J y TPa) (substrate NNstar)).  rw H5.
rw canonical_du2_pr. ee. fprops. aw. au.
cp (H20 _ H40). cp (H18 _ H40). ufi u1 H41. ufi v1 H42. fold y in H35.
rwi (H30 _ H35) H42.  rwi (H27 _ H35) H41.
rwi H10 H41. rwi H10 H42. ee. nin H46. awi H46. ee.
nin H44. awi H44. ee.
cp (Exercice2_7A9 H35 H48 H50).
rwi Hr H51. awi H51. elim H51. tv. ee. nin H44. ee. contradiction.
ee. contradiction. nin H46. ee. awi H46. elim H46. tv. nin H46.
rwi Hr H47. awi H47. elim H47. tv. nin H35. am.
```

We pretend that there is no least upper bound. If w is any closure, if $x \leq y \leq w(x)$ then $w(y) = w(x)$. We apply this to $x = 0$. Let $k = w(0)$ and $k' = k + 1$. Since $k \in \mathrm{N}_2$ we have $k' < k$. We consider the function w' that is like w but $w'(0) = k'$. The previous remark implies $w'(x) = k'$ whenever $x \leq k'$. We define $w'(x) = w(x)$ otherwise.

```
cp (H32 _ inc0_Bnat). set (k:= W (J card_zero TPa) w).
assert (inc (J card_zero TPa) (substrate NNstar)). rw H5. rw canonical_du2_pr.
ee. fprops. left. aw. eee.
assert (inc k (substrate NNstar)). red in H19. ee. rw H40.
uf k. app inc_W_target.
assert (inc (P k) Bnat). rwi H5 H35. rwi canonical_du2_pr H35. nin H35.
nin H36; nin H36; am.
assert (forall x, gle NNstar x k -> W x w = k).
ir. assert (gle NNstar (J card_zero TPa) x).  rw H10. ee. ue.  wr H5.
order_tac. aw. assert (inc x (substrate NNstar)). order_tac.
rwi H5 H40. rwi canonical_du2_pr H40. ee. nin H41. left. ee. tv. am.
app zero_smallest. fprops.  right. right. ee. tv. rw H42. fprops.
red in H19. ee. cp (H45 _ _ H38). cp (H45 _ _ H37). ufi k H47.
rwi (H25 _ H34) H47. uf k. order_tac.
set (k':= J (succ (P k)) TPb).
assert (inc k' (substrate NNstar)). uf k'. rw H5. rw canonical_du2_pr.
split. fprops. right. aw. split. fprops. tv.
assert (glt NNstar k' k). split.  rw H10. ee. ue. ue.
uf k'. aw. right. left. ee. fprops. uf k. rw H33. fprops.
app is_less_than_succ. fprops. uf k'. red. ir. cp (f_equal P H39).
awi H40. bwi H36. nin H36. symmetry in H40. contradiction.

set (ww := fun z=> Yo (gle NNstar z k') k' (W z w)).
assert (transf_axioms ww (substrate NNstar) (substrate NNstar)).
red. ir. uf ww. nin (p_or_not_p  (gle NNstar c k')). rww Y_if_rw.
rw Y_if_not_rw. red in H19. ee. rw H47. app inc_W_target. am.
set (w' := BL ww (substrate NNstar) (substrate NNstar)).
assert (is_function w'). uf w'. app bl_function.
```

We pretend that this function is an upper bound for u and v.

```
assert (forall x, inc x (substrate NNstar) ->
```

```
  (gle NNstar (W x u1)  (W x w') &  gle NNstar (W x v1)  (W x w'))).
ee. ir. cp H44. rwi H5 H44. rwi canonical_du2_pr H44. nin H44. nin H46.
nin H46. assert (x = J (P x) (Q x)). aw. rwi H47 H48.
cp (H27 _ H46). cp (H30 _ H46). fold u1 in H49. fold v1 in H50.
wri H48 H49. wri H48 H50. rw H49; rw H50. assert  (gle NNstar x k').
rw H10. ee. ue. ue. right. right. ee. am. uf k'. aw. fprops.
assert (W x w' = k'). uf w'. aw. uf ww. rww Y_if_rw. rw H52.
split. rw H10. ee. rw canonical_du2_pr. ee. fprops. left. aw. ee.
red in H26. ee. wr H54. app inc_W_target. tv. ue. right. right. uf k'.
aw. eee. rw H10. ee. rw canonical_du2_pr. ee. fprops. left. aw. ee.
red in H29. ee. wr H54. app inc_W_target. tv. ue. right. right. uf k'.
aw. eee. nin H46. nin (p_or_not_p  (gle NNstar x k')).
assert (W x w' = k'). uf w'. aw. uf ww. rww Y_if_rw. rw H49.
uf u1. rw (H28 _ H45 H47). uf v1.  rw (H31 _ H45 H47). au.
assert (W x w' = W x w). uf w'. aw. uf ww. rww Y_if_not_rw. rw H49.
ee. app H20. app H18.
```

We pretend that this is a closure.

```
assert (is_closure w' NNstar). red. ee. red. eee. ir.
assert (inc x (substrate NNstar)). order_tac.
assert (inc y (substrate NNstar)). order_tac. red in H19. ee.
cp (H52 _ _ H45). nin (p_or_not_p  (gle NNstar y k')).
assert (gle NNstar x k'). order_tac. uf w'. aw. uf ww. rww Y_if_rw.
rww Y_if_rw. order_tac. assert (W y w' = W y w).  uf w'. aw. uf ww.
rw Y_if_not_rw. tv. am. rw H55.
nin (p_or_not_p (gle NNstar x k')). uf w'. aw. uf ww. rww Y_if_rw.
assert (gle NNstar k' y). rw H10. ee. ue. ue.
rwi H5 H47. rwi canonical_du2_pr H47. nin H47. nin H57. nin H57.
elim H54. rw H10. ee. rw canonical_du2_pr.  au. ue. right. right. eee.
uf k'. aw. fprops. right. left. ee. uf k'. aw. fprops. rw H58. fprops.
assert (is_cardinal (P y)). fprops. assert (is_cardinal (P k')).
uf k'. aw. fprops. nin (cardinal_le_total_order3 H59 H60). am.
elim H54. rw H10. ee. rw canonical_du2_pr.  au. ue. right. left.
ee. rw H58. fprops. uf k'. aw. fprops. am.
cp (H52 _ _ H57). cp (H24 _ H38). order_tac. uf w'. aw. uf ww.
rww Y_if_not_rw.
ir. uf w'. aw. uf ww.  nin (p_or_not_p (gle NNstar x k')). rww Y_if_rw.
rww Y_if_not_rw. app H24. ir.
nin (p_or_not_p (gle NNstar x k')). assert (W x w' = k').
uf w'. aw. uf ww. rww Y_if_rw. rw H47. uf w'. aw. uf ww.  rww Y_if_rw.
order_tac. assert (W x w' = W x w). uf w'. aw. uf ww. rww Y_if_not_rw.
set (y :=  W x w'). assert (~ gle NNstar y k'). uf y. rw H47. dneg.
cp (H24 _ H45). order_tac. uf w'. aw. uf ww. rww Y_if_not_rw.
uf y.  rw H47. app H25. uf y. cp (H24 _ H45). rw H47. order_tac.
```

The conclusion is now clear.

```
cp (H7 _ H43).
assert (w' =sof_value (substrate NNstar) (substrate NNstar) (corr_value w')).
uf corr_value. uf sof_value. aw.
assert (gle (closure_ordering NNstar) (corr_value v1) (corr_value w')).
rw Exercise2_7a. ee. am. am. ir. wr H22; wr H45. nin (H42 _ H48). am. am.
assert (gle (closure_ordering NNstar) (corr_value u1) (corr_value w')).
rw Exercise2_7a. ee. am. am. ir. wr H21; wr H45. nin (H42 _ H49). am. am.
cp (H16 _ H47 H46). rwi Exercise2_7a H48. ee. cp (H50 _ H34).
```

```
  wri H45 H53. fold w in H53. fold k in H53. ufi w' H53. awi H53. ufi ww H53.
  rwi Y_if_rw H53. order_tac. rw H10. ee. ue. ue. uf k'. right. right. aw.
  ee. tv. fprops. am. am. am. am. am.
Qed.
```

¶ 8. An ordered set E is said to be *ramified* (on the right) if, for each pair of elements x, y of E such that $x < y$, there exists $z > x$ such that y and z are not comparable. E is said to be *completely ramified* (on the right) if it is ramified and has no maximal elements. Every antidirected set (§ 1, Exercise 22) is ramified.

(a) Let E be an ordered set and let a be an element of E. Let $\mathfrak{R}_a$ denote the set of ramified subsets of E which have a as least element. Show that $\mathfrak{R}_a$, ordered by inclusion, has a maximal element.

(b) If E is branched (§ 1, Exercise 24), show that every maximal element of $\mathfrak{R}_a$ is completely ramified.

(c) Given an example of a branched set which is not ramified. The branched set defined in § 1, Exercise 24 (c) is completely ramified.

(d) Let E be a set in which each interval $]\leftarrow, x]$ is totally ordered. Show that E has an antidirected cofinal subset (§ 1, Exercise 22) (use (b)).

The first point is trivial. Assume $x < y$ in an antidirected set. There exists z such that $x < z$ and the intervals $[y, \rightarrow [$ and $[z, \rightarrow [$ do not intersect. This implies that y and z are non-comparable and the set is ramified.

```
Definition is_ramified r :=
  forall x y, glt r x y -> exists z, glt r x z & ~ gle r y z&  ~ gle  r z y.

Definition is_ramified_c r :=
  is_ramified r & not (exists x, maximal_element r x).

Lemma Exercise2_8a: forall r, order r -> anti_directed r -> is_ramified r.
Proof. ir. rwi (Exercise1_23h H) H0. nin H0. red. ir. nin (H0 _ _ H2).
  nin H3. exists x0. ee. am.
  red. ir. assert (gle r x0 x0). order_tac. order_tac. app (H4 _ H5 H6).
  red. ir. assert (gle r y y). order_tac. order_tac. app (H4 _ H6 H5).
Qed.
```

Let $\mathfrak{R}_a$ be the set of all subsets Z of E such that the induced ordering is ramified and has a as least element. We rewrite this directly in terms of the ordering on E.

```
Definition  Exercise2_8a_R r a :=
  Zo (powerset (substrate r))
  (fun z => is_ramified (induced_order r z) &
    least_element  (induced_order r z) a).

Lemma Exercise2_8b: forall r a F, order r ->
  inc F (Exercise2_8a_R r a) =
  (sub F (substrate r)
    &  (forall x y, glt r x y -> inc x F -> inc y F ->
      exists z, glt r x z & ~ gle r y z&  ~ gle  r z y & inc z F)
```

```
    & inc a F
    & (forall z, inc z F -> gle r a z)).
Proof. ir. assert (forall F u v, sub F (substrate r) ->
  gle (induced_order r F) u v =   (inc u F & inc v F & gle r u v)).
  ir. ap iff_eq. ir. cp (related_induced_order1 H1).
  cp (related_induced_order3 H1). nin H3. au. ir. ee. aw.
  assert (forall F u v, sub F (substrate r) ->
  glt (induced_order r F) u v =   (inc u F & inc v F & glt r u v)).
  ir. uf glt. ap iff_eq. ir. rwi H0 H2. eee. am. ir. rww H0. eee.
  uf Exercise2_8a_R. uf is_ramified. ap iff_eq; ir; Ztac.
  nin H4. rwi powerset_inc_rw H3. red in H5. awii H5. nin H5.
  ee. am. ir. assert (glt (induced_order r F) x y). rww H1. au.
  nin (H4 _ _ H10). ee. exists x0.  rwi H1 H11. ee. am. clear H13. dneg.
  rww H0. au. dneg. rww H0. au. am. am. am. ir. cp (H6 _ H7).
  rwii H0 H8. eee.
  ap powerset_inc. eee. ee. ir. rwi H1 H6. ee. nin (H3 _ _ H8 H6 H7).
  exists x0. ee. rw H1. eee. am. clear H11. dneg. rwii H0 H11.  eee.
  dneg. rwii H0 H13.  eee. am. red. aw. ee. am. ir. rww H0. eee.
Qed.
```

Le's show that $\mathfrak{R}_a$ has a maximal element for the inclusion order. We use Zorn's Lemma. The nontrivial point is to prove that the $\cup \mathrm{A}_i \in \mathfrak{R}_a$ if $\mathrm{A}_i \in \mathfrak{R}_a$ and for each pair (i, j) either $\mathrm{A}_i \subset \mathrm{A}_j$ or $\mathrm{A}_k \subset \mathrm{A}_i$. This last condition implies that the union is ramified. The element a is obviously the least element of the union, except when the union is empty; the least upper bound of the empty set is the singleton $\{a\}$.

```
Lemma Exercise2_8c: forall r a, order r -> inc a (substrate r) ->
  exists A, maximal_element (inclusion_suborder  (Exercise2_8a_R r a)) A.
Proof. ir. app Zorn_lemma. fprops. red. aw. ir.
  nin H2. set (F := Exercise2_8a_R r a) in *.
  assert (order (inclusion_suborder F)). fprops.
  assert (sub X (substrate (inclusion_suborder F))). aw. awii H3.
  assert (forall x y, inc x X -> inc y X -> sub x y \/ sub y x).
  ir. nin (H3 _ _ H6 H7). awii H8. ee; au. red in H8. awii H8. ee; au.
  nin (emptyset_dichot X). exists (singleton a).
  red. split. aw.  uf F. rw Exercise2_8b. ee. app sub_singleton.
  ir. awi H9. awi H10. nin H8. elim H11. ue. fprops. ir. awi H8. rw H8.
  order_tac. am. ir. aw. rwi H7 H8. elim (emptyset_pr H8).
  assert (inc (union X) F). uf F. rw  Exercise2_8b. awi H5.
  ee. red. ir. nin (union_exists H8). nin H9. cp (H1 _ H10).
  ufi F H11. rwi  Exercise2_8b H11. ee. app H11. am.
  ir. nin (union_exists H9). nin (union_exists H10). ee.
  assert (exists z, inc z X & inc x z & inc y z). nin (H6 _ _ H13 H14).
  exists x0. eee. exists x1. eee. nin H15. ee.
  cp (H1 _ H15). ufi F H18.  rwi  Exercise2_8b H18. ee.
  nin (H19 _ _ H8 H16 H17). ee. exists x3. eee. union_tac. am.
  nin H7. cp (H1 _ H7).  ufi F H8. rwi  Exercise2_8b H8. ee. union_tac. am.
  ir. nin (union_exists H8). nin H9. cp (H1 _ H10).  ufi F H11.
  rwi Exercise2_8b H11. ee.  app H14. am. am.
  exists (union X). red. aw. split. am. ir. aw. eee. app union_sub.
Qed.
```

Needs to be completed.

9. *An ordinal sum* $\sum_{\iota\in I} E_\iota$ *(§ 1, Exercise 3) is well-ordered if and only if* I *and each* E_ι *is well-ordered.*

One implication can be found in the chapter 8. If the sum is well ordered, then the subset of I formed of all ι such that $E_\iota \neq \emptyset$ is well-ordered. We deduce that I is well-ordered is no E_ι is empty.

```
Lemma ordinal_sum_wo_pr : forall r f g,
  ordinal_sum_axioms1 r f g ->  worder (ordinal_sum r f g) ->
  (worder r &  (forall i, inc i (domain f) -> worder (V i g))).
Proof. ir. nin H. cp H. red in H. ee. split. am. ir.
  set (y := fun_image x  (fun z => (J (rep (V z f)) z))).
  assert (sub y (substrate (ordinal_sum r f g))). aw. red. ir.
  ufi y H11. awi H11. nin H11. nin H11. wr H12. app inc_disjoint_union.
  wr H3. app H9. app nonempty_rep. app H1.  wr H3. app H9.
  assert (nonempty y). nin H10. exists (J (rep (V y0 f)) y0). uf y. aw.
  ex_tac. nin H0. nin (H13 _ H11 H12). nin H14. awii H14. awii H15.
  assert (inc (Q x0) x). ufi y H14. awi H14. nin H14. nin H14. wr H16. aw.
  exists (Q x0). split. aw. aw. ir. aw.
  assert (inc  (J (rep (V x1 f)) x1) y). uf y. aw. ex_tac.
  cp (H15 _ H18). awi H19. ee. nin H21. nin H21. am. nin H21. rw H21.
  order_tac. app H9. am. am. am.
  ir. split. app H7. ir.
  set (y := fun_image x  (fun z => (J z i))).
  assert (sub y (substrate (ordinal_sum r f g))). aw. red. ir.
  ufi y H12. awi H12. nin H12. nin H12. wr H13. app inc_disjoint_union. wrr H8.
  app H10. assert (nonempty y). nin H11. exists (J y0 i). uf y. aw.
  ex_tac. nin H0. nin (H14 _ H12 H13). nin H15. awii H15. awii H16.
  assert (inc (P x0) x). ufi y H15. awi H15. nin H15. nin H15. wr H17. aw.
  exists (P x0). split. aw. app H7. aw. ir. aw.
  assert (inc (J x1 i) y). uf y. aw. ex_tac.
  cp (H16 _ H19). awi H20. ee. ufi y H15. awi H15. nin H15. nin H22. nin H15.
  wri H23 H22. awi H22. nin H22. elim H24. tv.  nin H22. wrr H22. am. am. am.
  app H7.
Qed.
```

---

10. *Let* I *be an ordered set and let* $(E_\iota)_{\iota\in I}$ *be a family of ordered sets, all equal to the same ordered set* E*. Show that the ordinal sum* $\sum_{\iota\in I} E_\iota$ *(§ 1, Exercise 3) is isomorphic to the lexicographic product of the sequence* $(F_\lambda)_{\lambda\in\{\alpha,\beta\}}$*, where the set* $\{\alpha,\beta\}$ *of two distinct elements is well-ordered by the relation whose graph is* $\{(\alpha,\alpha),(\alpha,\beta),(\beta,\beta)\}$*, and where* $F_\alpha = I$ *and* $F_\beta = E$*. This product is called the* lexicographic product *of* E *by* I *and is written* E.I*.*

This is lemma *ordinal_product_pr* of chapter 8.

¶ 11. * *Let* I *be a well-ordered set and let* $(E_\iota)_{\iota\in I}$ *be a family of ordered sets, each of which contains at least two distinct comparable elements. Then the lexicographic product of the* E_ι *is well-ordered if and only if each of the* E_ι *is well-ordered and* I *is* finite *(if* I *is infinite, construct a strictly decreasing infinite sequence in the lexicographic product of the* E_ι*).* $_*$

We start with an auxiliary result: if $f : I \to J$ is strictly decreasing, both sets are well-ordered, then I is finite. Since I is totally ordered, every non-empty subset K of I has a greatest

element (inverse image of the least element of $f(\mathrm{K})$). Since I is well-ordered, there is a morphism $\mathbf{N} \to s(\mathrm{I})$, or a morphism $\mathrm{I} \to s(\mathbf{N})$, where $s(\mathrm{K})$ is some segment of K. Let g be the morphism. In the first case, the image has a greatest element $g(a)$. We have a contradiction since $g(a+1) > g(a)$. In the second case, the morphism may be surjective; same contradiction as above. Otherwise its range is finite and I is finite.

```
Lemma worder_decreasing_finite: forall r r' f,
  worder r -> worder r' ->
  (forall i, inc i (substrate r) -> inc (f i) (substrate r')) ->
  (forall i j, glt r i j -> glt r' (f j) (f i)) ->
  is_finite_set (substrate r).
Proof. ir. assert (forall X, sub X (substrate r) -> nonempty X ->
  exists a, inc a X & forall b, inc b X -> gle r b a).
  ir. set (Y := fun_image X f). assert (sub Y (substrate r')).
  red. ir. ufi Y H5. awi H5. nin H5. nin H5. wr H6. app H1. app H3.
  assert (nonempty Y). nin H4. exists (f y). uf Y. aw. ex_tac.
  nin H0. nin (H7 _ H5 H6). red in H8. awii H8. nin H8. ufi Y H8. awi H8.
  nin H8. nin H8. exists x0. split. am. ir. assert (inc (f b) Y). uf Y.
  aw. ex_tac. cp (H9 _ H12).  cp (related_induced_order1 H13). wri H10 H14.
  cp (worder_total H). nin H15. nin (H16 _ _ (H3 _ H8) (H3 _ H11)).
  nin (equal_or_not x0 b). rw H18. order_tac. app H3.
  assert (glt r x0 b). split; am. cp (H2 _ _ H19). order_tac. am.
  cp  worder_Bnat_order.  nin (isomorphism_worder H4 H). nin H5. nin H5.
  nin H5. clear H6. nin H5. red in H7. ee. nin H9.
  assert (nonempty (range (graph x))).  exists (W card_zero x). aw.
  exists card_zero. graph_tac. wr H10. rw substrate_Bnat_order. fprops. fprops.
  nin (H3 _ H5 H14). nin H15. awi H15. nin H15.
  assert (inc x1 (source x)). graph_tac.
  assert (source x = Bnat). wr H10. rww substrate_Bnat_order.
  assert (inc (succ x1) (source x)). rwi H18 H17. rw H18. fprops.
  assert (gle Bnat_order x1 (succ x1)). rw related_Bnat_order. red. ee.
  wrr H18. wrr H18. app is_less_than_succ. rwi H18 H17. fprops.
  rwi (H12 _ _ H17 H19) H20.
  assert (x0 = W x1 x). sy.  app W_pr. wri H21 H20.
  assert (inc  (W (succ x1) x) (range (graph x))). aw. exists (succ x1).
  graph_tac. fprops. cp (H16 _ H22). assert ( x0 = W (succ x1) x). order_tac.
  rwi H21  H24. cp (H13 _ _ H17 H19  H24). rwi H18 H17. bwi H17.
  nin H17. contradiction. fprops.
  nin H5. nin H5. clear H6. nin H5. nin (well_ordered_segment H4 H5).
  rwi substrate_Bnat_order H7.
  cp (@sub_refl  (substrate r)).
  nin (emptyset_dichot (substrate r)). rw H9.  app emptyset_finite.
  nin (H3 _ H8 H9). nin H10. red in H6. ee. assert (inc (W x0 x) Bnat).
  wr H7. nin H13.  aw. exists x0. graph_tac. ue. fprops.
  assert (inc  (succ (W x0 x)) Bnat). fprops. cp H18. wri H7  H19. awi H19.
  nin H19.  assert (W x1 x =  (succ (W x0 x))). app W_pr. fct_tac.
  assert (inc x1 (substrate r)). rw H14. nin H13;graph_tac.
  cp (H11 _ H21). wri H14 H16. rwi (H16 _ _ H21 H10) H22.
  rwi related_Bnat_order H22. red in H22. ee. rwi H20 H24.
  rwi lt_n_succ_le H24. nin H24. elim H25. tv. am. am. nin H13. fprops.
  nin H7. nin H7.  rwi substrate_Bnat_order H7. rwii segment_Bnat_order H8.
  assert (is_finite_set (range (graph x))). rw H8.
  app finite_set_interval_co. nin H6. ee.  cp (equipotent_range H11).
  red in H9. red.
  assert (cardinal (substrate r) = cardinal (range (graph x))). rw H12. aw.
  rww H16.
```

```
Qed.
```

Assume that the product is well-ordered and that we have two families such that $x_\iota < y_\iota$. Fix κ, and let $a \in E_\kappa$. We consider the family $f(a)$ defined by $f(a)_\kappa = a$ and $f(a)_\iota = x_\iota$ for $\iota \neq \kappa$. Given a non-empty subset X of E_κ the set $f(X)$ has a least element, thus X has a least element.

```
Lemma worder_lexicographic_order_bis: forall r f g,
  lexicographic_order_axioms r f g ->
  worder (lexicographic_order r f g) ->
  (forall i, inc i (domain g) ->  exists x, exists y, glt (V i g) x y) ->
  ((forall i, inc i (domain g) -> worder (V i g)) &
  is_finite_set (substrate r)).
Proof. ir. cp H. red in H.
  set (f0 := fun i => choose (fun z =>  glt (V i g) (P z) (Q z))).
  assert (forall i, inc i (domain g) ->  glt (V i g) (P (f0 i)) (Q (f0 i))).
  ir. uf f0. app choose_pr. nin (H1 _ H3). nin H4. exists (J x x0). aw.
  ee. ir.  split. app H8. ue.
  ir.  set (f1 := fun j => P (f0 j)).
  assert (Ha: order (V i g)). app H8. ue.
  assert (forall j, inc j (domain f) -> inc (f1 j) (V j f)).
  ir. wr H9. rwi H7 H13. cp (H3 _ H13). order_tac. am.
  set (f3:= fun a  => L (domain f) (fun j => Yo (j = i) a (f1 j))).
  assert (forall a , inc a (V i f) ->
    inc (f3 a) (substrate (lexicographic_order r f g))).
  ir. aw. uf f3. rw productb_pr. ee. gprops. bw. bw. ir. bw.
  nin (equal_or_not i0 i). rww Y_if_rw. ue. rww Y_if_not_rw. app H13. am.
  set (X:= fun_image x f3).
  assert (sub X (substrate  (lexicographic_order r f g))).
  red. ir. ufi X H15. awi H15. nin H15. nin H15. wr H16. app H14. wrr H9.
  app H11. ue. assert (nonempty X). nin H12. exists (f3 y). uf X. aw. ex_tac.
  nin H0. nin  (H17 _ H15 H16). nin H18. awii H18. awii H19.
  ufi X H18. awi H18. nin H18. nin H18.
  exists x1. split. aw. ir. awii H21. aw.
  assert (inc (f3 x2) X). uf X. aw. ex_tac. cp (H19 _ H22).
  cp (related_induced_order1 H23). awi H24. ee.
  assert (Hb : V i x0 = x1). wr H20. uf f3. bw. rww Y_if_rw. ue.
  assert (Hc : V i (f3 x2) = x2). uf f3. bw. rww Y_if_rw. ue.
  nin (equal_or_not x1 x2). wr H27. order_tac. app H11. nin H.
  assert (least_element  (induced_order r
            (Zo (domain f) (fun i : Set => V i x0 <> V i (f3 x2)))) i).
  split. aw. Ztac. ue.  rw Hc. rw Hb. am. rw H4. app Z_sub. aw. ir.
  nin (equal_or_not x3 i). aw. rw H30.
  order_tac.  ue. ue. wrr H30. Ztac. wri H20 H32. ufi f3 H32. bwi H32.
  rwii Y_if_not_rw H32. rwii Y_if_not_rw H32. elim H32. tv. am. am.
  rw H4. app Z_sub. cp (H26 _ H29). nin H30. rwi Hb H30. rwi Hc H30. am. am.
```

Consider the sequence $z(\kappa)$ such that $z(\kappa)_\iota = x_\iota$, if $\iota < \kappa$ and $z(\kappa)_\iota = y_\iota$ otherwise. This is a strictly decreasing function of κ. The previous lemma then implies that I is finite.

```
  set (f1 := fun i =>  L (domain f)
    (fun z => Yo (glt r z i) (P (f0 z)) (Q (f0 z)))).
  assert (forall i, inc i (domain f) -> inc (f1 i) (productb f)).
  ir. rw productb_pr. uf f1. split. gprops. bw. split. tv. ir. bw.
  wrr H9. cp (H8 _ H11). rwi H7 H11. cp (H3 _ H11).
  nin (p_or_not_p (glt r i0 i)). rww Y_if_rw. order_tac. rww Y_if_not_rw.
```

```
order_tac. am.
assert (forall i j, glt r i j ->
  glt (lexicographic_order r f g) (f1 j) (f1 i)).
ir. nin H. assert (inc i (substrate r)). order_tac.
assert (inc j (substrate r)). order_tac.
assert (inc i (domain f)). ue. assert (inc j (domain f)). ue.
assert (V i (f1 i)  = (Q (f0 i))). uf f1. bw. rww Y_if_not_rw. red. ir.
nin H17. elim H18. tv.
assert (V i (f1 j)  = (P (f0 i))). uf f1. bw. rww Y_if_rw.
assert (inc i (domain g)). ue. cp (H3 _ H19).
split. aw. ee. app H10. app H10. ir.
assert (inc i (domain g)). ue.
set (E:= Zo (domain f) (fun i0  => V i0 (f1 j) <> V i0 (f1 i))) in *.
assert (sub E (substrate r)). rw H4. uf E. app Z_sub. red in H21.
awii H21. nin H21. assert (inc i E). uf E. Ztac. rw H17. rw H18. nin H20; am.
cp (H24 _ H25). nin (equal_or_not j0 i). rw H27. rw H17. rww H18.
cp (related_induced_order1 H26). assert (glt r j0  i). split. am. am.
cp (related_induced_order3 H26). nin H30. ufi E H30. Ztac.
ufi f1 H33. bwi H33. rwii Y_if_rw H33. rwii Y_if_rw H33.
elim H33. tv. order_tac. am. am. red. ir. wri H17 H20. wri H18 H20.
rwi H21 H20. nin H20. elim H22. tv.
assert (forall i, inc i (substrate r) ->
  inc (f1 i) (substrate (lexicographic_order r f g))).
ir. aw. app H10. ue.
app (worder_decreasing_finite _ H H0 H12 H11).
```

The proof of the converse is by induction on the cardinal of I; it can be found in chapter 8. If I is empty, the product has a single element, thus is well-ordered. Otherwise I has a least element ι. Let X be a non-empty subset of the product, and X_ι the ι-th projection. This is a non-empty subset of E_ι and has a least element α. Let $\bar{X}$ the set of elements of X for which the ι-component is equal to α. If $x \in \bar{X}$ and $y \in X - \bar{X}$ then $x < y$. Denote by f' the restriction of f to $I - \{\iota\}$. We denote by $\bar{X}'$ the set of restrictions. By induction, the restriction product is well-ordered and this set has a least element x', that is the restriction of some element $x \in \bar{X}$. If $y \in \bar{X}$ then $x \leq y$ if and only if $x' \leq y'$ in the restriction product. This x is the least element of $\bar{X}$, hence of X.

¶ 12. Let I be a totally ordered set and let $(E_\iota)_{\iota \in I}$ be a family of ordered sets indexed by I. Let $R\{\!\{x, y\}\!\}$ denote the following relation on $E = \prod_{\iota \in I} E_\iota$: "the set of indices $\iota \in I$ such that $\mathrm{pr}_\iota x \neq \mathrm{pr}_\iota y$ is well-ordered, and if κ is the least element of this subset of I, we have $\mathrm{pr}_\kappa x < \mathrm{pr}_\kappa y$". Show that $R\{\!\{x, y\}\!\}$ is an order relation between x and y on E. If the E_ι are totally ordered, show that the connected components of E with respect to the relation "x and y are comparable" (Chapter II, § 6, Exercise 10) are totally ordered sets. Suppose that each E_ι has at least two elements. Then E is totally ordered if and only if I is well-ordered and each E_ι is totally ordered (use Exercise 3); and E is then the lexicographic product of the E_ι.

13. (a) Let $\mathrm{Is}(\Gamma,\Gamma')$ be the relation "Γ is an ordering (on E) and Γ' is an ordering (on E'), and there exists an isomorphism of E, ordered by Γ, onto E', ordered by Γ'". Show that $\mathrm{Is}(\Gamma,\Gamma')$ is an equivalence relation on every set whose elements are orderings. The term $\tau_\Delta(\mathrm{Is}(\Gamma,\Delta))$ is an ordering called the *order-type* of Γ and denoted by $\mathrm{Ord}(\Gamma)$, or $\mathrm{Ord}(E)$ by abuse of notations. Two ordered sets are isomorphic if and only if their order-types are equal.

(b) Let $R\{\lambda,\mu\}$ be the relation: "λ is an order-type, and μ is an order-type and there exists an isomorphism of the set ordered by λ onto a subset of the set ordered by μ". Show that $R\{\lambda,\mu\}$ is a preorder relation between λ and μ. It will be denoted by $\lambda \prec \mu$.

(c) Let I be an ordered set and let $(\lambda_\iota)_{\iota\in I}$ be a family of order-types indexed by I. The order-type of the ordinal sum (§ 1, Exercise 3) of the family of sets ordered by the λ_ι ($\iota \in I$) is called the *ordinal sum* of the order-types λ_ι ($\iota \in I$) and is denoted by $\sum_{\iota\in I}\lambda_\iota$. If $(E_\iota)_{\iota\in I}$ is a family of ordered sets, the order type of $\sum_{\iota\in I}E_\iota$ is $\sum_{\iota\in I}\mathrm{Ord}(E_\iota)$. If I is the ordinal sum of a family $(J_\kappa)_{\kappa\in K}$, show that

$$\sum_{\kappa\in K}\Big(\sum_{\iota\in J_\kappa}\lambda_\iota\Big)=\sum_{\iota\in I}\lambda_\iota.$$

(d) Let I be a well-ordered set and $(\lambda_\iota)_{\iota\in I}$ be a family of order-types indexed by I. The order-type of the lexicographic product of the family of sets indexed by by the λ_ι ($\iota \in I$) is called the *ordinal product* of the order-types λ_ι ($\iota \in I$) and is denoted by $\prod_{\iota\in I}\lambda_\iota$. If $(E_\iota)_{\iota\in I}$ is a family of ordered sets, the order type of the lexicographic product of the family $(E_\iota)_{\iota\in I}$ is $\prod_{\iota\in I}\mathrm{Ord}(E_\iota)$. If I is the ordinal sum of a family of well-ordered sets $(J_\kappa)_{\kappa\in K}$ indexed by a well-ordered set K, show that

$$\prod_{\kappa\in K}\Big(\prod_{\iota\in J_\kappa}\lambda_\iota\Big)=\prod_{\iota\in I}\lambda_\iota.$$

(e) We denote by $\lambda+\mu$ (resp. $\mu\lambda$) the ordinal sum (resp. ordinal product) of the family $(\xi_\iota)_{\iota\in J}$ where $J=\{\alpha,\beta\}$ is a set with two distinct elements, ordered by the relation whose graph is $\{(\alpha,\alpha),(\alpha,\beta),(\beta,\beta)\}$, and where $\xi_\alpha=\lambda$ and $\xi_\beta=\mu$. Show that if I is a well-ordered set of order-type λ and if $(\mu_\iota)_{\iota\in I}$ is a family of order-types such that $\mu_\iota=\mu$ for each $\iota\in I$ then $\sum_{\iota\in I}\mu_\iota=\mu\lambda$. We have $(\lambda+\mu)+\nu=\lambda+(\mu+\nu)$, $(\lambda\mu)\nu=\lambda(\mu\nu)$, and $\lambda(\mu+\nu)=\lambda\mu+\lambda\nu$ (but in general $\lambda+\mu\neq\mu+\lambda$, $\lambda\mu\neq\mu\lambda$ and $(\lambda+\mu)\nu\neq\lambda\nu+\mu\nu$).

(f) Let $(\lambda_\iota)_{\iota\in I}$ and $(\mu_\iota)_{\iota\in I}$ be two families of order-types indexed by the same ordered set I. Show that if $\lambda_\iota\prec\mu_\iota$ for each $\iota\in I$, then $\sum_{\iota\in I}\lambda_\iota\prec\sum_{\iota\in I}\mu_\iota$ and (if I is well-ordered) $\prod_{\iota\in I}\lambda_\iota\prec\prod_{\iota\in I}\mu_\iota$. If J is a subset of I, show that $\sum_{\iota\in J}\lambda_\iota\prec\sum_{\iota\in I}\lambda_\iota$ and (if I is well-ordered and the λ_ι are non-empty) $\prod_{\iota\in J}\lambda_\iota\prec\prod_{\iota\in I}\lambda_\iota$.

(g) Let λ^* denote the order-type of the set ordered by the opposite of the ordering λ. Then we have

$$(\lambda^*)^*=\lambda \qquad \text{and} \qquad \Big(\sum_{\iota\in I}\lambda_\iota\Big)^*=\sum_{\iota\in I^*}\lambda_\iota^*$$

where I^* denotes the set I endowed with the opposite of the ordering given in I.

Note: part (g) not yet done.

¶ 14. An *ordinal* is the order-type of a well-ordered set (Exercise 13).

(a) Show that, if $(\lambda_\iota)_{\iota\in I}$ is a family of ordinals indexed by a well-ordered set I, then the ordinal sum $\sum_{\iota\in I}\lambda_\iota$ is an ordinal; *and that, if moreover, I is finite, then the ordinal product

$\sum_{\iota\in I}\lambda_\iota$ is an ordinal (Exercise 11).$_*$ The order-type of the empty set is denoted by 0, and that of a set with one element by 1 (by abuse of language, cf § 3). Show that

$$\alpha+0=0+\alpha=\alpha \qquad \text{and} \qquad \alpha.1=1.\alpha=\alpha$$

for every ordinal α.

(b) Show that the relation "λ is an ordinal and μ is an ordinal and $\lambda \prec \mu$" is a *well-ordering* relation, denoted by $\lambda \leq \mu$ (Note that, if λ are μ are ordinals, the relation $\lambda \prec \mu$ is equivalent to "λ is equal to the order-type of a segment of μ" (no. 5, Theorem 3, Corollary 3): given a family $(\lambda_\iota)_{\iota\in I}$ of ordinals, consider a well-ordering in I and take the ordinal sum of the family of sets ordered by the λ_ι; finally use Proposition 2 of no 1.).

(c) Let α be an ordinal. Show that the relation "ξ is an ordinal and $\xi \leq \alpha$" is collectivizing in ξ, and that the set O_α of ordinals $< \alpha$ is a well-ordered set such that $\mathrm{Ord}(O_\alpha) = \alpha$. We shall often identify O_α with α.

(d) Show that for every family $(\xi_\iota)_{\iota\in I}$ there exists a unique ordinal α such that the relation "λ is an ordinal and $\xi_\iota \leq \lambda$ for all $\iota \in I$ is equivalent to $\alpha \leq \lambda$. By abuse of language, α is called the *least upper bound* of the family of ordinals $(\xi_\iota)_{\iota\in I}$, and we write $\alpha = \sup_{\iota\in I}\xi_\iota$ (it is the greatest element of the union of $\{\alpha\}$ and the set of the ξ_ι). The least upper bound of the set of ordinals $\xi < \alpha$ is either α or an ordinal β such that $\alpha = \beta + 1$. In the latter case β is said to be the predecessor of α.

15. (a) Let α and β be two ordinals. Show that the inequality $\alpha < \beta$ is equivalent to $\alpha + 1 \leq \beta$, and that it implies the inequalities $\xi + \alpha < \xi + \beta$, $\alpha + \xi \leq \beta + \xi$, $\alpha\xi \leq \beta\xi$ for all ordinals ξ and $\xi\alpha < \xi\beta$ if $\xi > 0$.

(b) Deduce from (a) that there exists no set to which every ordinal belongs (use Exercise 14 (d)).

(c) Let α, β, μ be three ordinals. Show that each of the relations $\mu + \alpha < \mu + \beta$, $\alpha + \mu < \beta + \mu$ implies $\alpha < \beta$; and that each of the relations $\mu\alpha < \mu\beta$, $\alpha\mu < \beta\mu$ implies $\alpha < \beta$ provided that $\mu > 0$.

(d) Show that the relation $\mu + \alpha = \mu + \beta$, implies $\alpha = \beta$, and that $\mu\alpha = \mu\beta$, implies $\alpha = \beta$ provided that $\mu > 0$.

(e) Two ordinals α and β are such that $\alpha \leq \beta$ if and only if there exists an ordinal ξ such that $\beta = \alpha + \xi$. This ordinal is then unique and is such that $\xi \leq \beta$; it is written $(-\alpha) + \beta$.

(f) Let α, β, ζ be three ordinals such that $\zeta < \alpha\beta$. Show that there exists two ordinals ξ, η such that $\zeta = \alpha\eta + \xi$ and $\xi < \alpha$, $\eta < \beta$ (cf. No. 5, Theorem 3, Corollary 3). Moreover, ξ and η are uniquely determined by these conditions.

¶ 16. An ordinal $\rho > 0$ is said to be *indecomposable* if there exists no pair of ordinals ξ, η such that $\xi < \rho$, $\eta < \rho$ and $\xi + \eta = \rho$.

(a) An ordinal ρ is indecomposable if and only if $\xi + \rho = \rho$ for every ordinal ξ such that $\xi < \rho$.

(b) If $\rho > 1$ is an indecomposable ordinal and if α is any ordinal > 0, then $\alpha\rho$ is indecomposable, and conversely (use Exercise 15(f)).

(c) If ρ is indecomposable and if $0 < \alpha < \rho$, then $\rho = \alpha\xi$, where ξ is an indecomposable ordinal (use Exercise 15(f)).

(d) Let α be an ordinal > 0. Show that there exists a greatest indecomposable ordinal among the indecomposable ordinals $\leq \alpha$ (consider the decomposition $\alpha = \rho + \xi$, where ρ is indecomposable).

(e) If E is a set of indecomposable ordinals, deduced from (d) that the least upper bound of E (Exercise 14(d)) is an indecomposable ordinal.

¶ 17. Given an ordinal α_0, a term $f(\xi)$ is said to be an *ordinal functional symbol (with respect to ξ) defined for* $\xi \geq \alpha_0$ if the relation "ξ is an ordinal and $\xi \geq \alpha_0$" implies the relation "$f(\xi)$ is an ordinal"; $f(\xi)$ is said to be *normal* if the relation $\alpha_0 \leq \xi < \eta$ implies $f(\xi) < f(\eta)$ and if for each family $(\xi_\iota)_{\iota\in I}$ of ordinals $\geq \alpha_0$ we have $\sup_{\iota\in I} f(\xi_\iota) = f(\sup_{\iota\in I} \xi_\iota)$ (cf. Exercise 14(d)).

(a) Show that for each ordinal $\alpha > 0$, $\alpha + \xi$ and $\alpha\xi$ are ordinal functional symbols defined for $\xi \geq 0$ (use Exercise 15(f)).

(b) Let $w(\xi)$ be an ordinal functional symbol defined for $\xi \geq \alpha_0$ such that $w(\xi) \geq \xi$ and such that $\alpha_0 \leq \xi < \eta$ implies $w(\xi) < w(\eta)$. Also let $g(\xi,\eta)$ be a term such that the relation "ξ and η are ordinals such that $g(\xi,\eta) > \xi$". Define a term $f(\xi,\eta)$ with the following properties: (1) for each ordinal $\xi \geq \alpha_0$, $f(\xi,1) = w(\xi)$; for each ordinal $\xi \geq \alpha_0$ and each ordinal $\eta > 1$, $f(\xi,\eta) = \sup_{0<\zeta<\eta} g(f(\xi,\zeta),\xi)$ (use Criterion C60 of no. 2). Show that if $f_1(\xi,\eta)$ is another term with these properties then $f(\xi,\eta) = f_1(\xi,\eta)$ for all $\xi \geq \alpha_0$ and all $\eta \geq 1$. Prove that, for each ordinal $\xi \geq \alpha_0$, $f(\xi,\eta)$ is a normal functional symbol with respect to η (defined for $\eta \geq 1$). Show that $f(\xi,\eta) \geq \xi$ for all $\eta \geq 1$ and $\xi \geq \alpha_0$ and that $f(\xi,\eta) \geq \eta$ for all $\xi \geq \sup(\alpha_0, 1)$ and $\eta \geq 1$. Furthermore, for each pair (α,β) of ordinals such that $\alpha > 0$, $\alpha \geq \alpha_0$ and $\beta \geq w(\alpha)$ there exists a unique ordinal ξ such that

$$f(\alpha,\xi) \leq \beta < f(\alpha,\xi+1),$$

and we have $\xi \leq \beta$.

(c) If we take $\alpha_0 = 0$, $w(\xi) = \xi + 1$, $g(\xi,\eta) = \xi + 1$ then $f(\xi,\eta) = \xi + \eta$. If we take $\alpha_0 = 1$, $w(\xi) = \xi$, $g(\xi,\eta) = \xi + \eta$ then $f(\xi,\eta) = \xi\eta$.

(d) Show that if the relations $\alpha_0 \leq \xi \leq \xi'$, $\alpha_0 \leq \eta \leq \eta'$ imply $g(\xi,\eta) \leq g(\xi',\eta')$, then the relations $\alpha_0 \leq \xi \leq \xi'$, $1 \leq \eta \leq \eta'$ imply $f(\xi,\eta) \leq f(\xi',\eta')$. If the relations $\alpha_0 \leq \xi \leq \xi'$, $\alpha_0 \leq \eta < \eta'$ imply $g(\xi,\eta) < g(\xi,\eta')$ and $g(\xi,\eta) \leq g(\xi',\eta)$, then the relations $\alpha_0 \leq \xi < \xi'$ and $\eta \geq 0$ imply $f(\xi,\eta+1) \leq f(\xi',\eta+1)$.

(e) Suppose that $w(\xi) = \xi$ and that the relations $\alpha_0 \leq \xi \leq \xi'$, $\alpha_0 \leq \eta < \eta'$ imply $g(\xi,\eta) < g(\xi,\eta')$ and $g(\xi,\eta) \leq g(\xi',\eta)$. Suppose moreover, that for each $\xi \geq \alpha_0$, $g(\xi,\eta)$ is a normal functional symbol with respect to η (defined for $\eta \geq \alpha_0$), and that, whenever $\xi \geq \alpha_0$, $\eta \geq \alpha_0$, and $\zeta \geq \alpha_0$, we have the associativity relation

$$g(g(\xi,\eta),\zeta) = g(\xi,g(\eta,\zeta)).$$

Show that, if $\xi \geq \alpha_0$, $\eta \geq 1$, and $\zeta \geq 1$, we have then

$$g(f(\xi,\eta),f(\xi,\zeta)) = f(\xi,\eta+\zeta)$$

("distributivity" of g with respect to f) and

$$f(f(\xi,\eta),\zeta) = f(\xi,\eta\zeta)$$

("associativity" of f).

¶ 18. In the definition procedure defined in Exercise 17 (b), take $\alpha_0 = 1 + 1$ (denoted by 2 by abuse of language),

$$w(\xi) = \xi, \qquad g(\xi, \eta) = \xi\eta.$$

Denote $f(\xi, \eta)$ by ξ^η and define α^0 to be 1 for all ordinals α. Also define 0^β to be 0 and 1^β to be 1 for all ordinals $\beta \geq 1$.

(a) Show that if $\alpha > 1$ and $\beta < \beta'$, we have $\alpha^\beta < \alpha^{\beta'}$, and that, for each ordinal $\alpha > 1$, α^ξ is a normal functional symbol with respect to ξ. Moreover, if $0 < \alpha \leq \alpha'$, we have $\alpha^\beta \leq \alpha'^\beta$.

(b) Show that $\alpha^\xi.\alpha^\eta = \alpha^{\xi+\eta}$ and $(\alpha^\xi)^\eta = \alpha^{\xi\eta}$.

(c) Show that if $\alpha \geq 2$ and $\beta \geq 1$, $\alpha^\beta \geq \alpha\beta$.

(d) For each pair of ordinals $\beta \geq 1$ and $\alpha \geq 2$, there exists three ordinals ξ, γ, δ such that $\beta = \alpha^\xi\gamma + \delta$ where $0 < \gamma < \alpha$ and $\delta < \alpha^\xi$, and these ordinals are uniquely by these conditions.

19. * Let α and β be two ordinals and let E and F be two well-ordered sets such that $\mathrm{Ord}(E) = \alpha$ and $\mathrm{Ord}(F) = \beta$. In the set E^F of mappings of F into E consider the subset G of mappings g such that $g(y)$ is equal to the least element of E for all but a *finite* number of elements $y \in F$. if F^* is the ordered set obtained by endowing F with the opposite order, show that G is a connected component with respect to the relation "x and y are comparable" (Chapter II, § 6, Exercise 10) in the product E^{F^*} endowed with the ordering defined in Exercise 12, and show that G is well-ordered. Furthermore, prove that $\mathrm{Ord}(G) = \alpha^\beta$ (use the uniqueness property of Exercise17 (b)). *

¶ 20. A set X is said to be *transitive* if the relation $x \in X$ implies $x \subset X$.

(a) If Y is a transitive set, then so is $Y \cup \{Y\}$. If $(Y_\iota)_{\iota\in I}$ is a family of transitive sets, then $\bigcup_{\iota\in I} Y_\iota$ and $\bigcap_{\iota\in I} Y_\iota$ are transitive.

(b) A set X is a *pseudo-ordinal* if every transitive set Y such that $Y \subset X$ and $Y \neq X$ is an element of X. A set S is said to be *decent* if the relation $x \in S$ implies $x \notin x$. Show that every pseudo-ordinal is transitive and decent (consider the union of decent transitive subsets of X and use (a)). If X is a pseudo-ordinal, so is $X \cup \{X\}$.

(c) Let X be a transitive set and suppose that each $x \in X$ is a pseudo-ordinal. Then X is a pseudo-ordinal (note that, for each $x \in X$, $x \cup \{x\}$ is a pseudo-ordinal contained in X).

(d) Show that $\emptyset$ is a pseudo-ordinal and that every element of a pseudo-ordinal X is a pseudo-ordinal (Consider the union of the transitive subsets of X whose elements are pseudo-ordinals).

(e) If $(X_\iota)_{\iota\in I}$ is a family of pseudo-ordinals then $\bigcap_{\iota\in I} X_\iota$ is the least element of this family (with respect to the relation of inclusion). (Use (b).) Deduced that, if E is a pseudo-ordinal, the relation $x \subset y$ between elements x, y of E is a well-ordering relation.

(f) Show that for each ordinal α there exists a unique pseudo-ordinal E_α such that $\mathrm{Ord}(E_\alpha) = \alpha$ (use (e) and Criterion C60). In particular the pseudo-ordinals whose order-type are 0, 1, $2 = 1 + 1$, and $3 = 2 + 1$ are respectively

$$\emptyset, \quad \{\emptyset\}, \quad \{\emptyset, \{\emptyset\}\}, \quad \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$$

Note. The French version of the exercise has one more item. It says: if X and Y are two pseudo-ordinals, then either $X \in Y$ or $Y \in X$ or $X = Y$. The hint for item (c) is: note that if $Y \neq X$ is transitive, then $Y \subset x$.

10.3 Section 3

¶ 1. Let E and F be two sets, let f be an injection of E into F and let g be mapping of F into E. Show that there exist two subsets A, B of E such that $B = E - A$ and two subsets A', B' of F such that $B' = F' - A'$ for which $A' = f(A)$ and $B = g(B')$. (Let $R = E - g(F)$) and put $h = g \circ f$; take A to be the intersection of the subsets M of E such that $M \supset R \cup h(M)$.)

2. *If* E *and* F *are two distinct sets, show that* $E^F \neq F^E$. *Deduce that if* E *and* F *are the cardinals 2 and 4 (= 2 + 2), then at least one of the sets* E^F, F^E *is not a cardinal.*

Assume $E^F = F^E$. If E is non-empty and $x \in E$, then the graph of the constant function with source F and value x is in E^F. The domain of this graph is F; from $E^F = F^E$ we deduce that it is E, hence E = F. The same conclusion holds, by symmetry, if F is non-empty. The remaining case is $E = \emptyset = F$.

Denote by *pow*(F, E) the cardinal of $\mathscr{F}(E;F)$, the set of functions from F into E. We have shown (lemma *power_2_4*) that *pow*(2, 4) = *pow*(4, 2). Thus, if a is a cardinal equipotent to $\mathscr{F}(2;4)$ and b a cardinal equipotent to $\mathscr{F}(4;2)$, we have $a = b$. The sets 4^2 and 2^4 are equipotent; if they were cardinals we would deduce 2 = 4. From 4 = 2 + 2 we get 0 = 2, which is absurd.

it is (according to *bijective_graph_function*) the cardinal of F^E; it is F^E if F^E is a cardinal.

¶ 3. Let $(\mathfrak{a}_\iota)_{\iota\in I}$ and $(\mathfrak{b}_\iota)_{\iota\in I}$ be two families of cardinals such that $\mathfrak{b}_\iota \geq 2$ for each $\iota \in I$.

(a) Show that, if $\mathfrak{a}_\iota \leq \mathfrak{b}_\iota$ for each $\iota \in I$, then

$$\sum_{\iota\in I} \mathfrak{a}_\iota \leq \prod_{\iota\in I} \mathfrak{b}_\iota.$$

(b) Show that, if $\mathfrak{a}_\iota < \mathfrak{b}_\iota$ for each $\iota \in I$, then

$$\sum_{\iota\in I} \mathfrak{a}_\iota < \prod_{\iota\in I} \mathfrak{b}_\iota.$$

(Note that a product $\prod_{\iota\in I} E_\iota$ cannot be the union of a family $(A_\iota)_{\iota\in I}$ such that $\mathrm{Card}(A_\iota) < \mathrm{Card}(E_\iota)$ for all $\iota \in I$, by observing that $\mathrm{Card}(\mathrm{pr}_\iota(A_\iota)) < \mathrm{Card}(E_\iota)$.).

4. Let E be a set and let f be a mapping of $\mathfrak{P}(E) - \emptyset$ into E such that for each non-empty subset X of E we have $f(X) \in X$ ("Choice function").

(a) Let $\mathfrak{b}$ be cardinal and let A be the set of all $x \in E$ such that $\mathrm{Card}(\overset{-1}{f}(x)) \leq \mathfrak{b}$. Show that if $\mathfrak{a} = \mathrm{Card}(A)$, then $2^{\mathfrak{a}} \leq 1 + \mathfrak{a}\mathfrak{b}$ (note that if $Y \subset A$ and $Y \neq \emptyset$ then $f(Y) \in A$).

(b) Let B be the set of all $x \in E$ such that, for each non-empty subset X of $\overset{-1}{f}(x)$, $\mathrm{Card}(X) \leq \mathfrak{b}$. Show that $\mathrm{Card}(B) \leq \mathfrak{b}$.

5. Let $(\lambda_\iota)_{\iota\in I}$ be a family of order-types (§2, Exercise 13), indexed by an ordered set I. Show that

$$\mathrm{Card}\Big(\sum_{\iota\in I} \lambda_i\Big) = \sum_{\iota\in I} \mathrm{Card}(\lambda_i)$$

and that, if I is well-ordered,

$$\mathrm{Card}\Big(\prod_{\iota\in I} \lambda_i\Big) = \prod_{\iota\in I} \mathrm{Card}(\lambda_i)$$

6. Show that for every set E there exists $X \subset E$ such that $X \notin E$ (use Theorem 2 of no. 6).

10.4 Section 4

1. (a) Let E be a set and let $\mathfrak{F}(E)$ be the set of finite subsets of E. Show that $\mathfrak{F}(E)$ is the smallest subset $\mathfrak{G}$ of $\mathfrak{P}(E)$ satisfying the following conditions: (i) $\emptyset \in \mathfrak{G}$; (ii) the relation $X \in \mathfrak{G}$ and $x \in E$ imply $X \cup \{x\} \in G$.

(b) Deduce from (a) that the union of two finite subsets A and B is finite (consider the set of subsets X of E such that $X \cup A$ is finite; cf § 5; no 1 Proposition 1, Corollary 1).

(c) Deduce from (a) and (b) that for every finite set E the set $\mathfrak{P}(E)$ is finite (consider the set of subsets X of E such that $\mathfrak{P}(X)$ is finite; cf § 5; no 1 Proposition 1, Corollary 4).

2. Show that a set E is finite if and only every non-empty subset of $\mathfrak{P}(E)$ has a maximal element (with respect to inclusion). (To show that the condition is sufficient, apply it to the set $\mathfrak{F}(E)$ of finite subsets of E).

3. Show that if a well-ordered set E is such that the ordered set obtained by endowing E with the opposite ordering is also well-ordered, then E is finite (consider the greatest element x of E such that the segment S_x is finite.

4. Let E be a finite set with $n \geq 2$ elements, and let C be a subset of $E \times E$ such that, for each pair x, y of distinct elements of E, exactly one of the two elements (x, y), (y, x) of $E \times E$ belongs to C. Show that there is a mapping f of the interval $[1, n]$ onto E such that $(f(i), f(i+1)) \in C$ for $1 \leq i \leq n-1$ (use induction on n).

¶ 5. Let E be an ordered set for which there exists an integer k such that k is the greatest number of elements in a free subset X of E (§ 1, Exercise 5). Show that E can be partitioned into k totally ordered subsets (with respect to the induced ordering)[2]. The proof is in two steps:

(a) If E is finite and has n elements, use induction on n; let a be a minimal element of E and let $E' = E - \{a\}$. If there exists a partition of E' into k totally ordered sets C_i $(1 \leq i \leq k)$, let U_i be the set of all $x \in C_i$ which are $\geq a$. Show that there is at least one index i for which a free subset $E' - U_i$ has at most $k-1$ elements. The proof of this is by *reduction ad absurdum*. For each i, let S_i be a free subset of $E' - U_i$ which has k elements, let S be the union of the sets S_i, and let s_j be the least element of $S \cap C_j$ for each index $j \leq k$; show that the $k+1$ elements a, $s_1, \ldots, s_k$ form a free subset of E.

(b) If E is arbitrary, the proof is by induction on k, as follows. A subset C of E is said to be *strongly related* in E if for each finite subset F of E there exists a partition of F into at most k totally ordered sets such that $C \cap F$ is contained in one of them. Show that there exists a maximal strongly related subset C_0, and that every free subset of $E - C_0$ has at most $k-1$ elements (Argue by contradiction, and suppose that there is a free subset $\{a_1, \ldots, a_k\}$ of k elements in $E - C_0$; Consider each set $C_0 \cup \{a_i\}$ $(1 \leq i \leq k)$, and express the fact that it is not strongly related, thus introducing a finite subset F_i of E for each index i. Then consider the union F of then sets F_i and use the fact that C_0 is strongly related to obtain a contradiction).

[2] This is called Dilworth's theorem in the French edition

¶ 6. (a) Let A be a set and let $(X_i)_{1\leq i\leq m}$, $(Y_j)_{m+1\leq j\leq m+n}$ be two finite families of subsets of A. Let h be the least integer such that, for each integer $r \leq m-h$ and each subset $\{i_1,\dots,i_{r+h}\}$ of $r+h$ elements of $[1,m]$, there exists a subset $\{j_1,\dots,j_r\}$ of r elements of $[m+1,m+n]$ for which the union of the sets X_{i_α} $(1\leq\alpha\leq r+h)$ meets each of the sets Y_{j_β} $(1\leq\beta\leq r)$ (which implies that $m\leq n+h$). Show that there exists a finite subset B of A with at most $n+h$ elements such that every X_i $(1\leq i\leq m)$ and every Y_j $(m+1\leq j\leq m+n)$ meets B. (Consider the order relation on the interval $[1,m+n]$ whose graph is the union of the diagonal and the set of pairs (i,j) such that $1\leq i\leq m$ and $m+1\leq j\leq m+n$ and $X_i\cap Y_j\neq\emptyset$, and apply Exercise 5 to this ordered set.)

(b) Let E and F be two finite sets and let $x\to A(x)$ be a mapping of E into $\mathfrak{P}(F)$. Then there exists an injection f of E into F such that $f(x)\in A(x)$ for each $x\in E$ if and only if for each subset H of E, we have $\mathrm{Card}\big(\bigcup_{x\in H} A(x)\big)\geq \mathrm{Card}(H)$ (the method of proof is analogous to that of (a), with $h=0$).

(c) With the hypotheses of (b), let G be subset of F. Then there exists an injection f of E into F such that $f(x)\in A(x)$ for each $x\in E$ and such that $f(E)\supset G$ if and only if f satisfies the condition of (b) and for each subset L of G the cardinal of the set of all $x\in E$ such that $A(x)\cap L\neq\emptyset$ is $\geq \mathrm{Card}(L)$. (Let $(a_i)_{1\leq i\leq p}$ be the sequence of distinct elements of G, arranged in some order; let $(b_j)_{p+1\leq j\leq p+m}$ be the sequence of distinct elements of F, arranged in some order; and let $(c_k)_{p+m+1\leq k\leq P+M+n}$ be the sequence of distinct elements of E, arranged in some order. Consider the order relation on the set $[1,p+m+n]$ whose graph is the union of the diagonal and the set of pairs (i,j) such that either

$$1\leq i\leq p \text{ and } p+1\leq j\leq p+m \text{ and } a_i=b_j,$$

$$\text{or } 1\leq i\leq p \text{ and } p+m+1\leq j\leq p+m+n \text{ and } a_i\in A(c_j),$$

$$\text{or } p+1\leq i\leq p+m \text{ and } p+m+1\leq j\leq p+m+n \text{ and } b_i\in A(c_j);$$

then apply Exercise 5.)

7. An element a of a lattice E is said to *irreducible* if the relation $\sup(x,y)=a$ implies either $x=a$ or $y=a$.

(a) Show that in a finite lattice E every element a can be written as $\sup(e_1,\dots,e_k)$ where the e_i $(1\leq i\leq n)$ are irreducible.

(b) Let E be a finite lattice and let J be the set of its irreducible elements. For each $x\in E$ let $S(x)$ be the set of all $y\in J$ which are $\leq x$. Show that the mapping $x\to S(x)$ is an isomorphism of E onto a subset of $\mathfrak{P}(J)$, ordered by inclusion, and that $S(\inf(x,y))=S(x)\cap S(y)$.

¶ 8. (a) Let E be a distributive lattice (§ 1, Exercise 16). If a is irreducible in E (Exercise 7), show that the relation $a\leq\sup(x,y)$ implies $a\leq x$ or $a\leq y$.

(b) Let E be a finite distributive lattice and let J be the set of its irreducible elements, ordered by the induced ordering. Show that the isomorphism $x\to S(x)$ of E onto a subset of $\mathfrak{P}(J)$ defined in Exercise 7 (b) is such that $S(\sup(x,y))=S(x)\cup S(y)$ Deduced that if J^* is the ordered set obtained by endowing J with the opposite ordering, then E is isomorphic to the set $\mathscr{A}(J^*,I)$ of increasing mappings of J^* into $I=\{0,1\}$ (§ 1, Exercise 6).

(c) With the hypothesis of (b), let P be the set of elements of J other than the least element of E. For each $x\in E$, let $y_1,\dots,y_k$ be the distinct minimal elements of the interval $]x,\to[$ in E;

for each index i, let q_i be an element of P such that $q_i \notin S(x)$ and $q \in S(y_i)$. Show that no two elements $q_1, \dots, q_k$ are comparable.

(d) Conversely, let $q_1, \dots, q_k$ be k elements of P, no two of which are comparable. Let $u = \sup(q_1, \dots, q_k)$ and let

$$v_i = \sup_{1 \le j \le k, j \ne i} (q_j) \qquad (1 \le i \le k).$$

Show that $v_i < u$ for $1 \le i \le k$. Let $x = \inf(v_1, \dots, v_k)$ and let

$$y_i = \inf_{1 \le j \le k, j \ne i} (v_j)$$

Show that $x < y_i$ for each index i, and deduce that the interval $]x, \rightarrow[$ has at least k distinct minimal elements.

¶ 9. A subset A of a lattice E is said to be a *sublattice* if for each pair (x, y) of elements of A, $\sup_E(x, y)$ and $\inf_E(x, y)$ belong to A.

(a) Let $(C_i)_{1 \le i \le n}$ be a finite family of totally ordered sets and let $E = \prod_{i=1}^{n} C_i$ be their product. Let A be a sublattice of E. Show that A cannot have more than n irreducible elements (Exercise 7) no two of which are comparable (The proof is by *reductio ad absurdum*. Suppose that there exist $r > n$ irreducible elements $a_1, \dots, a_n$ in A, no two of which are comparable. Consider the elements $u = \sup(a_1, \dots, a_n)$ and

$$v_j = \inf_{1 \le j \le r, j \ne i} (a_j)$$

of A. By projecting onto the factors, show that $u = v_i$ for some index i, and hence that two of the a_i are comparable).

(b) Conversely, let F be a finite distributive lattice, let P be the set of irreducible elements of F other than the least element of F, and suppose that n is the greatest number of elements in a free subset of P (§ 1, Exercise 5). Show that F is isomorphic to a sublattice of a product of n totally ordered sets (Apply Exercise 5, which shows that P is the union of n totally ordered sets P_i with no elements in common. Let C_i be the totally ordered set obtained by adjoining a least element to P_i $(1 \le i \le n)$. With each $x \in F$ associate the family $(x_i)_{1 \le i \le n}$ where x_i is the least upper bound in C_i of the sets of elements of P_i which are $\le x$.)

¶ 10. (a) An ordered set E is isomorphic to a subset of a product of n totally ordered sets if and only if the graph of the ordering on E is the intersection of the graphs on n total orderings on E. (To show that the condition is necessary, show that if $F = \prod_{i=1}^{n} F_i$ is a product of n totally ordered sets, the the graph of the product ordering on F is the intersection of n graphs of lexicographic orderings on F.)

(b) An ordered set E is isomorphic to a subset of the product of two totally ordered sets if and only if the ordering Γ on E is such that there exists another ordering Γ' on E with the property that any two distinct elements of E are comparable with respect to exactly one of the orderings Γ and Γ'.

(c) Let A be a finite set of n elements. Let E be the subset of $\mathfrak{P}(A)$ consisting of all subsets $\{x\}$ and $A - \{x\}$ as x runs through A. Show that n is the smallest integer m such that E, ordered by inclusion, is isomorphic to a subset of a product of m totally ordered sets (use (a)).

¶ 11. Let A be a set and let $\mathfrak{R}$ be a subset of the set $\mathfrak{F}(A)$ of finite subsets of A. $\mathfrak{R}$ is set to be *mobile* if it satisfies the following condition:
(MO) If X, Y are two distinct elements of $\mathfrak{R}$ and if $z \in X \cap Y$, then there exists $Z \subset X \cap Y$ belonging to $\mathfrak{R}$ such that $z \notin Z$.
A subset P of A is then said to be *pure* if it contains no set belonging to $\mathfrak{R}$.

(a) Show that every pure subset of A is contained in a maximal pure subset of A.

(b) Let M be a maximal pure subset of A. Show that for each $x \in \complement M$ there exists a unique finite subset $E_M(x)$ of M such that $E_M(x) \cup \{x\} \in \mathfrak{R}$. Moreover, if $y \in E_M(x)$, the set $(M \cup \{x\}) - \{y\}$ is a maximal pure subset of A.

(c) Let M, N be two maximal pure subsets of A, such that $N \cap \complement M$ is finite. Show that Card(M) = Card(N). (Proof by induction on the cardinal of $N \cap \complement M$, using (b).)

(d) Let M, N be two maximal pure subsets of A, and put $N' = N \cap \complement M$, $M' = M \cap \complement N$. Show that $M' \subset \bigcap_{x \in N'} E_M(x)$. * Deduce that Card(M) = Card(N) (by virtue of (c), we are reduced to the case where N′ and M′ are infinite; show then that $\mathrm{Card}(M') \leq \mathrm{Card}(N')$). *

10.5 Section 5

1. Prove the formula

$$\sum_{k=q+1}^{n-p+q+1} \binom{n-k}{p-q-1}\binom{k-1}{q} = \binom{n}{p},$$

where $p \leq n$ and $q < p$ (generalize the argument of no. 8, Corollary to Proposition 14).

2. If $n \geq 1$, prove the relation

$$\binom{n}{0} - \binom{n}{1} + \binom{n}{2} - \ldots + (-1)^n \binom{n}{n} = 0$$

(Define a one-to-one correspondence between the set of subsets of $[1, n]$ which have an even number of elements, and the set of subsets of $[1, n]$ which have an odd number of elements. Distinguish between the cases n even and n odd.)

3. Prove the relations

$$\binom{n}{0}\binom{n}{p} + \binom{n}{1}\binom{n-1}{p-1} + \binom{n}{2}\binom{n-2}{p-2} + \cdots + \binom{n}{p}\binom{n-p}{0} = 2^p \binom{n}{p},$$

$$\binom{n}{0}\binom{n}{p} - \binom{n}{1}\binom{n-1}{p-1} + \binom{n}{2}\binom{n-2}{p-2} - \cdots + (-1)^p \binom{n}{p}\binom{n-p}{0} = 0.$$

(Consider the subsets of p elements of $[1, n]$ which contain a given subset of k elements ($0 \leq k \leq p$), and use Exercise 2 for the second formula.)

4. Prove Proposition 15 of no. 8 by defining a bijection of the set of mappings u of $[1, h]$ into $[0, n]$ such that

$$\sum_{i=1}^{h} u(x) \leq n$$

onto the set of strictly increasing mappings of $[1, h]$ into $[1, n + h]$.

5. * (a) Let E be a distributive lattice and let f be a mapping of E into a commutative semigroup M (written additively) such that

$$f(x)+f(y)=f(\sup(x,y))+f(\inf(x,y))$$

for all x, y in E. Show that for each finite subset I of E, we have

$$f(\sup(\mathrm{I}))+\sum_{2n\leq \mathrm{Card(I)}}\Big(\sum_{\mathrm{H}\subset\mathrm{I},\mathrm{Card(H)}=2n} f(\inf(\mathrm{H}))\Big)=\sum_{2n+1\leq \mathrm{Card(I)}}\Big(\sum_{\mathrm{H}\subset\mathrm{I},\mathrm{Card(H)}=2n+1} f(\inf(\mathrm{H}))\Big)$$

(By induction on Card(I)).) *

(b) In particular let A be a set, let $(\mathrm{B}_i)_{i\in\mathrm{I}}$ be a finite family of finite subsets of A and let B be the union of the B_i. For each subset H of I, put $\mathrm{B_H}=\bigcap_{i\in\mathrm{H}}\mathrm{B}_i$. Show that

$$\mathrm{Card(B)}+\sum_{2n\leq \mathrm{Card(I)}}\Big(\sum_{\mathrm{Card(H)}=2n}\mathrm{Card(B_H)}\Big)=\sum_{2n+1\leq \mathrm{Card(I)}}\Big(\sum_{\mathrm{Card(H)}=2n+1}\mathrm{Card(B_H)}\Big).$$

6. Prove the formula

$$\binom{n+h}{h}=1+\binom{h}{1}\binom{n+h-1}{h}-\binom{h}{2}\binom{n+h-2}{h}+\ldots+(-1)^h\binom{h}{h}\binom{n}{h}.$$

(If F denotes the set of mappings u of $[1,h]$ into $[0,n]$ such that $\sum_{x=1}^{h}u(x)\leq n$, consider for each subset H of $[1,h]$ the set of all $u\in\mathrm{F}$ such that $u(x)\geq 1$ for each $x\in\mathrm{H}$, and use Exercise 5.)

7. (a) Let $\mathrm{S}_{n,p}$ denote the number of mappings of $[1,n]$ *onto* $[1,p]$. Prove that

$$\mathrm{S}_{n,p}=p^n-\binom{p}{1}(p-1)^n+\binom{p}{2}(p-2)^n-+\ldots+(-1)^{p-1}\binom{p}{p-1}.$$

(Note that $p^n=\mathrm{S}_{n,p}=\binom{p}{1}\mathrm{S}_{n,p-1}+\binom{p}{2}\mathrm{S}_{n,p-2}+\ldots+\binom{p}{p-1}$ and use Exercise 3.)

(b) Prove that $\mathrm{S}_{n,p}=p(\mathrm{S}_{n-1,p}+\mathrm{S}_{n-1,p-1})$ (method of no. 8, Proposition 13).

(c) Prove that

$$\mathrm{S}_{n+1,n}=\frac{n}{2}(n+1)!\qquad\text{and}\qquad\mathrm{S}_{n+2,n}=\frac{n(3n+1)}{24}(n+2)!$$

(consider the elements r of $[1,n]$ whose inverse image consists of more than one element).

(d) If $\mathrm{P}_{n,p}$ is the number of partitions into p parts of a set of n elements, show that $\mathrm{S}_{n,p}=p!\mathrm{P}_{n,p}$.

8. Let p_n be the number of permutations of a set E with n elements such that $u(x)\neq x$ for all $x\in\mathrm{E}$. Show that

$$p_n=n!-\binom{n}{1}(n-1)!+\binom{n}{2}(n-2)!-\cdots+(-1)^n$$

* and hence that $p_n\sim n!/e$ as $n\to\infty$ * (same method as in Exercise 7 (a)).

9. (a) Let E be a set with qn elements. Show that the number of partitions of E into n subsets each of q elements is equal to

$$(qn)!/(n!(q!)^n).$$

(b) Suppose that E = $[1, qn]$. Show that the number of partitions of E into n subsets each of q elements, no one of which is an interval is equal to

$$\frac{(qn)!}{n!(q!)^n} - \frac{(qn-q+1)!}{1!(n-1)!(q!)^{n-1}} + \frac{(qn-2q+2)!}{2!(n-2)!(q!)^{n-2}} - \cdots + (-1)^n$$

(same method as in Exercises 7 and 8).

10. Let $q_{n,k}$ be the number of strictly increasing mappings u of $[1,k]$ into $[1,n]$ such that for each even (resp. odd) x, $u(x)$ is even (resp. odd). Show that $q_{n,k} = q_{n-1,k-1} + q_{n-2,k}$ and deduce that

$$q_{n,k} = \binom{[\frac{n+k}{2}]}{k}.$$

¶ 11. Let E be a set with n elements and let S be a set of signs such that S is the disjoint union of E and a set consisting of a single element f. Suppose that f has weight 2 and that each element of E has weight 0 (Chapter I, Appendix Exercise 3).

(a) Let M be the set of significant words in $L_0(S)$ which contain each element of E exactly once. Show that if u_n is the number of elements in M, then $u_{n+1} = (4n-2)u_n$, and deduce that

$$u_n = 2.6\ldots(4n-6) \qquad (n \geq 2)$$

(This is the number of products of n different terms with respect to a non-associative law of composition).

(b) let x_i be the ith of the elements of E which appear in a word of M. Show that the number v_n of words of M, for which the sequence x_i is given, is equal to $\binom{2n-2}{n-1}/n$ and satisfies the relation

$$v_{n+1} = v_1 v_n + v_2 v_{n-1} + \cdots + v_{n-1} v_2 + v_n v_1.$$

¶ 12. (a) let p and q be two integers ≥ 1, let $n = 2p+q$, let E be a set with n elements and let $N = \binom{n}{p} = \binom{n}{p+q}$. Let $(X_i)_{1\leq i\leq N}$ (resp $(Y_i)_{1\leq i\leq N}$) be the sequence of all subsets of E which have p (resp $p+q$) elements arranged in a certain order. Show that there exists a bijection ϕ of $[1,n]$ onto itself such that $X_{\phi(i)} \subset Y_i$ for all i. (The method is analogous to that of Exercise 6 of § 4: observe that for each $r \leq N$ the number of sets Y_j which contain at least one of X_1, …, X_r is $\geq r$).

(b) Let h, k be two integers ≥ 1, let n be an integer such that $2h+k<n$, let E be a set with n elements and let $(X_i)_{1\leq i\leq r}$ be a sequence of distinct subsets of E, each having h elements. Show that there exists a sequence $(Y_j)_{1\leq j\leq r+1}$ of distinct subsets of E, each having $h+k$ elements, such that each Y_j contains at least one X_i and each X_i is contained in at least one Y_j (by induction on n, using (a)).

¶ 13. Let E be set with $2m$ elements, let q be an integer $< m$, and let $\mathscr{F}$ be the set of all subsets $\mathfrak{S}$ of $\mathfrak{P}(E)$ with the following property: if X and Y are two distinct elements of $\mathfrak{S}$ such that $X \subset Y$, then $Y - X$ has at most $2q$ elements.

(a) Let $\mathfrak{M} = (A_i)_{1\leq i\leq p}$ be an element of $\mathscr{F}$ such that $p = \mathrm{Card}(\mathfrak{M})$ is as large as possible. Show that $m - q \leq \mathrm{Card}(A_i) \leq m + q$ for $1 \leq i \leq p$ (Argue by contradiction. Suppose, for example, that there exists indices i such that $\mathrm{Card}(A_i) < m - q$ and consider those of the A_i for which $\mathrm{Card}(A_i)$ has the least possible value $m-q-s$ (where $s \geq 1$). Let $A_1, \ldots, A_r$, say, these sets. Let $\mathfrak{G}$ be the set of subsets of E each of which is the union of some A_i $(1 \leq i \leq r)$ and a subset of $2q+1$ elements contained in $E - A_i$. Show that $\mathfrak{G}$ contains at least $r+1$ elements (cf. Exercise 12), and that if $B_1, \ldots, B_{r+1}$ are $r+1$ distinct elements of $\mathfrak{G}$, the set whose elements are B_j $(1 \leq j \leq r+1$ and A_i $(r+1 \leq i \leq p)$ belongs to $\mathscr{F}$, contrary to the hypothesis.)

(b) Deduce from (a) that the number of elements p of each $\mathfrak{S} \in \mathscr{F}$ satisfies the inequality

$$p \leq \sum_{k=0}^{2q} \binom{2m}{m-q+k}.$$

(c) Establish results analogous to those of (a) and (b) when $2m$ or $2q$ is replaced by an uneven number.

¶ 14. Let E be a finite set with n elements, let $(a_j)_{1\leq j\leq n}$ be the sequence of elements of E arranged in some order, and let $(A_i)_{1\leq i\leq m}$ be a sequence of subsets of E.

(a) For each index j, let k_j be the number of indices i such that $a_j \in A_i$, and let $S_i = \mathrm{Card}(A_i)$. Show that

$$\sum_{j=1}^{n} k_j = \sum_{i=1}^{m} s_i.$$

(b) Suppose that for each subset $\{x, y\}$ of two elements of E, there exists exactly one index i such that x and y are contained in A_i. Show that, if $a_j \notin A_i$, then $S_i \leq k_j$.

(c) With the hypotheses of (b), show that $m \geq n$ (Let k_n be the least of the numbers k_j. Show that we may suppose that, whenever $i \leq k_n$, $j \leq k_n$, and $i \neq j$, we have $a_j \notin A_i$ and $a_n \notin A_j$ for all $j \geq k_n$.)

(d) With the hypotheses of (b), show that $m = n$ if and only if one of the following two alternatives is true: (i) $A_1 = \{a_1, a_2, \ldots, a_{n-1}\}$, $A_i = \{a_{i-1}, a_n\}$ for $i = 2, \ldots, n$; (ii) $n = k(k-1)+1$; each A_i has k elements, and each element of E belongs to exactly k set A_i.

¶ 15. Let E be a finite set, let $\mathfrak{L}$ and $\mathfrak{C}$ be two disjoint non-empty subsets of $\mathfrak{P}(E)$, and let λ, h, k, l be four integers ≥ 1 with the following properties: (i) for each $A \in \mathfrak{L}$ and each $B \in \mathfrak{C}$, $\mathrm{Card}(A \cap B) \geq \lambda$; (ii) for each $A \in \mathfrak{L}$, $\mathrm{card}(A) \geq h$; (iii) for each $B \in \mathfrak{C}$, $\mathrm{card}(B) \leq k$; (iv) for each $x \in E$ the number of elements of $\mathfrak{L} \cup \mathfrak{C}$ which contain x is exactly l. Show that $\mathrm{Card}(E) \leq hk/\lambda$. (Let $(a_i)_{1\leq i\leq n}$ be the sequence of distinct elements of E arranged in some order, and for each i let r_i be the number of elements of $\mathfrak{L}$ to which a_i belongs. Show that, if $\mathrm{Card}(\mathfrak{L}) = s$ and $\mathrm{Card}(\mathfrak{C}) = t$, then we have

$$\sum_{i=1}^{n} r_i \leq sh, \qquad \sum_{i=1}^{n} (l - r_i) \leq tk, \qquad \sum_{i=1}^{n} r_i(l - r_i) \geq \lambda st.)$$

For $\mathrm{Card}(E)$ to be equal to hk/λ it is necessary and sufficient that for each $A \in \mathfrak{L}$ and each $B \in \mathfrak{C}$ we have $\mathrm{card}(A) = h$, $\mathrm{card}(B) = k$, $\mathrm{Card}(A \cap B) = \lambda$ and that there exists an $r \leq l$ such that for each $x \in E$ the number of elements of $\mathfrak{L}$ to which x belongs is equal to r.

16. Let E be a finite set with n elements, let $\mathfrak{D}$ be a non-empty subset of $\mathfrak{P}(E)$, and let λ, k, l be three integers ≥ 1 with the following properties: (i) if A and B are distinct elements of $\mathfrak{D}$, then $\mathrm{Card}(A \cap B) = \lambda$; (ii) for each $A \in \mathfrak{D}$, $\mathrm{card}(A) \leq k$; (iii) for each $x \in E$ the number of elements of $\mathfrak{D}$ to which x belongs is equal to l. Show that

$$n(\lambda - 1) \leq k(k-1)$$

and that if $n(\lambda-1) = k(k-1)$ then $\lambda = k$ and $\mathrm{Card}(\mathfrak{D}) = n$. (Given $a \in E$, let $\mathfrak{L}$ be the set of all $A - \{a\}$ where $A \in \mathfrak{D}$ and $a \in A$, and let $\mathfrak{C}$ be the set of all $A \in \mathfrak{D}$ such that $a \notin A$. Apply the results of Exercise 15 to $\mathfrak{L}$ and $\mathfrak{C}$.

¶ 17. Let i, h, k be three integers such that $i \geq 1$, $h \geq i$, $k \geq i$. Show that there exists an integer $m_i(h,k)$ with the following properties: for each finite set E with at least $m_i(h,k)$ elements, and each partition $(\mathfrak{X}, \mathfrak{Y})$ of the set $\mathfrak{F}_i(E)$ of subsets of i elements of E, it is impossible that every subset of h elements of E contains a subset $X \in \mathfrak{X}$ and that every subset of k elements of E contains a subset $Y \in \mathfrak{Y}$; in other words, if every subset of h elements of E contains some $X \in \mathfrak{X}$ there exists a subset A of k elements of E such that every subset of i elements of A belongs to $\mathfrak{X}$ (Proof by induction. Show that we may take $m_1(h,k) = h+k-1$, $m_i(i,k) = k$ and $m_i(h,i) = h$ and finally $m_i(h,k) = m_{i-1}(m_i(h-1,k), m_i(h,k-1))+1$. If E is a set with $m_i(h,k)$ elements if $a \in E$ and $E' = E - \{a\}$, show that if the proposition were false, then every subset of $m_i(h-1,k)$ elements of E' would contain a subset X' of $i-1$ elements such that $X' \cup \{a\} \in \mathfrak{X}$, and that every subset of $m_i(h,k-1)$ elements of E' would contain a subset Y' of $i-1$ elements such that $Y' \cup \{a\} \in \mathfrak{Y}$.

18. (a) Let E be a finite ordered set with p elements. If m, n are two integers such that $mn < p$, show that E has either a totally ordered subset of m elements or else a free subset (§ 1, Exercise 5) of n elements (use § 4, Exercise 5).

(b) Let h, k be two integers ≥ 1 and let $r(h,k) = (h-1)(k-1)+1$. Let I be a totally ordered set with at least $r(h,k)$ elements. Show that, for each finite sequence $(x_i)_{i\in I}$ of elements of a totally ordered set E, there exists either a subset H of h elements of I such that the sequence $(x_i)_{i\in H}$ is increasing, or else a subset K of k elements of I such that the sequence $(x_i)_{i\in H}$ is decreasing. (Use (a) applied to $I \times E$.)

10.6 Section 6.

1. *A set* E *is infinite if and only if each mapping* f *of* E *into* E *there exists a non-empty set* S *of* E *such that* $S \neq E$ *and* $f(S) \subset S$.

Assume E non-empty, and let $f : E \to E$. Fix $x \in E$, and define by induction $g(n+1) = f(g(n))$, with $g(0) = x$. Let G be the range (or target) of g and $S = f(G)$. Then G and S are non-empty and stable by f. If $x \notin S$, then $S \neq E$. Otherwise, $x = f(g(m)) = g(m+1)$. Let $n = m+1$. Then $x = g(n)$ and by induction on i, $g(i) = g(i+n)$. By induction on k, we have $g(i) = g(i+kn)$. By Euclidean division, every element of S has the form $g(i)$ for $i < n$. This show that S is finite. If E is infinite, we deduce $S \neq E$.

```
Lemma Exercise_6_1: forall  E, infinite_set E =
  (forall f, is_function f -> source f = E -> target f = E ->
    exists F, sub F E & nonempty F & F <> E & sub (image_by_fun f F) F).
```

```
Proof. ir. ap iff_eq. ir.
  nin (emptyset_dichot E). rwi H3 H. red in H. red in H. ee. elim H4.
  app is_finite0. nin H3.  assert (forall u, inc u E -> inc (W u f) E).
  ir. wr H2. app inc_W_target. ue. rename H4 into Ha.
  cp (induction_defined_pr (fun n : Set => W n f) y).
  cp (integer_induction_stable _ H3 Ha).
  set (g:=induction_defined (fun n : Set => W n f) y) in *. simpl in H4. ee.
  set (F:= target g).  nin H6.
  assert (inc y F). uf F. wr H7. app inc_W_target. ue.
  assert (sub (image_by_fun f F) F). uf F. red. ir. ufi image_by_fun H10.
  awi H11. nin H11. ee. wri H9 H11. ufi image_of_fun H11. awi H11. nin H11. ee.
  red in H13. wr H12. wr (W_pr H6 H13). wr H8.
  app inc_W_target. rw H4. rwi H4 H11.  fprops. wrr H4. fprops. ue.
  set (G:=image_by_fun f F).
  exists G. ee.  apply sub_trans with F. app H11. app H5. uf G.
  exists (W y f). aw. ex_tac. uf F. rww H1.
  red. ir. wri H12 H3. ufi G H3.
  ufi image_by_fun H3. awi H3. nin H3. nin H3.
  ufi F H3. wri H9 H3. ufi image_of_fun H3. awi H3. nin H3. ee.
  cp (W_pr H6 H14). cp (W_pr H0 H13). wri H15 H16. wri H8 H16. wri H7 H16.
  set (k:=  succ x0) in *. assert (inc k Bnat). uf k. rwi H4 H3. fprops.
  assert (forall i, inc i Bnat ->  W i g = W (card_plus i k) g).
  app cardinal_c_induction_v. wr H16. rww zero_unit_suml. fprops. ir. rw H8.
  rw H19.
  assert (card_plus (succ n) k = succ (card_plus n k)). uf succ.
  rw (card_plus_commutative n card_one).
  wr card_plus_associative. app card_plus_commutative. rw H20. rww H8. fprops.
  am.  assert (forall i, inc i Bnat -> forall j, inc j Bnat ->
  W i g = W (card_plus i (card_mult j k)) g). intros i H19.
  ap cardinal_c_induction_v. rw card_mult_commutative. rw zero_prod_absorbing.
  rww zero_unit_sumr. fprops. ir. rw card_mult_commutative.
  rw mult_via_plus. rw card_plus_associative. wr H18. rww card_mult_commutative.
  fprops. fprops.
  assert (forall z, inc z E -> exists m, cardinal_lt m k & z = W m g). ir.
  wri H12 H20. ufi G H20. ufi image_by_fun H20. awi H20. nin H20. nin H20.
  red in H21. wr (W_pr H0 H21). ufi F H20. wri H9 H20. ufi image_of_fun H20.
  awi H20. nin H20. nin H20. red in H22. wr (W_pr H6 H22). wr H8.
  assert (inc (succ x2) Bnat). rwi H4 H20. fprops. assert (inc k Bnat).
  uf k. rwi H4 H3. fprops.  assert (k <> card_zero). uf k.
  cp (succ_positive x0). red in H25. ee. intuition.
  cp (division_exists H23 H24 H25). nin H26. nin H26. ee. red in H28. ee.
  rw H28. rw card_plus_commutative. exists x4.  split. am.
  rww card_mult_commutative. wr H19. tv. am. am. wrr H4. fprops. fprops.
  assert (sub E (image_by_fun g (interval_co_0a k))). red. ir.
  assert (inc k Bnat). uf k. rwi H4 H3. fprops.  nin (H20 _ H21). ee.
  assert (inc x2 Bnat). red in H23. ee. rw inc_Bnat. rwi inc_Bnat H22.
  app (le_int_is_int H22 H23). uf image_by_fun. aw.
  exists x2. ee. rww interval_co_0a_pr2.
  red. rw H24. app defined_lem. rww H4. fprops.
  cp (cardinal_le8 H21). rwi cardinal_le3 H22.
  assert (is_finite_set (interval_co_0a k)). uf interval_co_0a.
  app finite_set_interval_co.
  assert (sub (interval_co_0a k) (source g)). rw H4. uf interval_co_0a.
  uf interval_co. rw substrate_Bnat_order. ap Z_sub.
  cp (finite_image_by H6 H24 H23). red in H25.
  set (w:= cardinal (image_by_fun g (interval_co_0a k)))  in *.
```

```
cp (le_int_is_int H25 H22). red in H. red in H. ee. elim H27. am. wrr H4.
fprops. fprops. uf image_by_fun. red. ir. awi H12. nin H12. nin H12.
ufi G H12. ufi image_by_fun H12. awi H12. nin H12. nin H12.
uf G. uf image_by_fun. aw. exists x0. split.  app H11. uf image_by_fun.
aw. exists x1. eee. am.
```

Converse. We assume that for every function $f : \mathrm{E} \to \mathrm{E}$ there is a non-trivial subset of E invariant by f. This implies E non-empty. Assume E finite. There is a bijection $\mathrm{T} : [0, n[\to \mathrm{E}$, where $n \neq 0$. Let f be the function $i \mapsto i+1$ (modulo n). This induces a function g on E, so that there is a set S such that $g(\mathrm{S}) \subset \mathrm{S}$. Since this set is not empty, it contains an element $\mathrm{T}(i)$. By induction it contains all $\mathrm{T}(i+j \pmod n)$, thus all elements of E.

```
ir. red. split. fprops. red. ir. set (n:= cardinal E).
assert (inc n Bnat). bw.
assert (equipotent E (interval_co_0a n)). wr cardinal_equipotent.
rw cardinal_interval_co_0a1. tv. am. red in H2. nin H2. ee.
set  (y:=inverse_fun x).
assert (nonempty E). set (f:=identity_fun E).
assert (is_function f). uf f.  app function_identity. assert (source f = E).
tv. assert (target f = E). tv. nin (H _ H5 H6 H7). ee.
nin H9. exists y0. app H8. assert (n <> card_zero). red. ee. ir. ufi n H6.
elim (cardinal_nonemptyset1 H5). am.
set (f:= fun i=> card_rem (succ i) n).
assert (Ha: forall i, inc i Bnat -> inc (card_rem i n) (interval_co_0a n)).
ir. rw interval_co_0a_pr2. cp (Bnat_division H7 H1 H6).
ee. red in H10; ee; am. am.
assert (Hb:sub  (interval_co_0a n)  Bnat). uf interval_co_0a.
uf interval_co. rw substrate_Bnat_order. app Z_sub.
assert (forall i, inc i (interval_co_0a n) -> inc (f i) (interval_co_0a n)).
ir. uf f. app Ha. cp (Hb _ H7). fprops.
cp (inverse_bij_is_bij H2). cp (bij_is_function H2).
cp (bij_is_function H8).
set (g:= fun u => W (f  (W u x)) y).
assert (transf_axioms g E E). red.  ir. uf g. uf y.
assert (E = target (inverse_fun x)). simpl. sy; am. rw H12.
app inc_W_target. simpl. rw H4. app H7. wr H4. app inc_W_target. ue.
set (g1:= BL g E E). assert (is_function g1). uf g1. app bl_function.
assert (source g1 = E). tv. assert (target g1 = E). tv.
nin (H _ H12 H13 H14). nin H15. nin H16. nin H17. nin H16.
set (i:=W y0 x). assert (inc i Bnat). assert (inc i (target x)).
uf i. app inc_W_target. ue. app Hb. ue.
assert (inc (W i y) x0). uf i. uf y.
set (xi:= W y0 x). assert (inc y0 (source x)). ue.
assert (xi =  W y0 x). tv. cp (W_inverse2 H2 H20 H21). ue.
assert (forall j, inc j Bnat -> inc (W (card_rem (card_plus i j)n) y) x0).
app cardinal_c_induction_v. rw  zero_unit_sumr.
assert (card_rem i n = i). cp (Bnat_division H19 H1 H6). ee.
assert (division_prop i n  card_zero i). red. split. rw zero_prod_absorbing.
rww  zero_unit_suml. fprops.  assert (inc i (target x)).
uf i. app inc_W_target. ue. rwi H4 H24.
rwi interval_co_0a_pr2 H24. am. am.
nin (division_unique H19 H1 H22 H21 (inc0_Bnat) H19 H6 H23 H24). am. ue.
fprops. ir.
set (u:= card_rem (card_plus i n0) n) in *.
assert (inc (W (W u y) g1) x0). app H18. aw. ex_tac.
ufi g1 H23. rwi W_bl_function H23. ufi g H23.
```

```
  assert (W (W u y) x = u). sy. app W_inverse. rw H4. uf u. app Ha. fprops.
  rwi H24 H23. ufi f H23.
  assert (card_rem (succ u) n = card_rem (card_plus i (succ n0)) n).
  uf u. assert (inc (card_plus i n0) Bnat). fprops.
  cp (Bnat_division  H25 H1 H6). ee. nin H28.
  set (q0:= card_quo (card_plus i n0) n) in H28.
  set (r0:= card_rem (card_plus i n0) n) in *. uf succ.
  rw card_plus_associative. rw H28. wr card_plus_associative.
  set (r1:= card_plus r0 card_one). set (r2:= card_plus (card_mult n q0) r1).
  assert (inc r1 Bnat). uf r1. fprops. assert (inc r2 Bnat). uf r2. fprops.
  cp (Bnat_division H30 H1 H6). cp (Bnat_division H31 H1 H6). ee.
  nin H37.
  assert (r2= card_plus (card_mult n q0) r1). tv. rwi H37 H39.
  rwi card_plus_associative H39. wri cardinal_distrib_prod_sum3 H39.
  set (q2:= (card_plus q0 (card_quo r1 n))).
  assert (division_prop r2 n q2 (card_rem r1 n)). red.  split. am. am.
  assert (inc q2 Bnat). uf q2. fprops.
  nin (division_unique H31 H1 H41 H32 H34 H33 H6 H40 H35). am. wr H25. am.
  am. app H15.
  elim H17. app extensionality. red. ir. assert (E = target y). sy; tv.
  rwi H23 H22. red in H8. ee. fold y in H24.
  cp (surjective_pr2 H24 H22). nin H25. ee. wr H26. ufi y H25.
  simpl in H25. rwi H4 H25.
  assert (inc i (target x)). uf i. app inc_W_target.  ue.
  rwi H4 H27.  rwi interval_co_0a_pr2 H27.
  set (j:= card_sub n i). nin H27. cp (card_sub_pr H1 H19 H27). fold j in H29.
  assert (inc j Bnat). uf j. fprops.
  set (k:= card_plus j x2).  assert (inc k Bnat). uf k. fprops.
  cp (H21 _ H31). assert (card_rem (card_plus i k) n = x2).
  assert (division_prop (card_plus i k) n card_one x2). red.
  rw one_unit_prodr. split. uf k. rw card_plus_associative. ue.
  rwi interval_co_0a_pr2 H25. eee. am. fprops.
  assert (inc (card_plus i k) Bnat). fprops.
  cp (Bnat_division H34 H1 H6). ee.
  cp (division_unique H34 H1 (inc1_Bnat) (Hb _ H25) H36 H35 H6 H33 H37).
  ee. sy; am.  rwi H33 H32. am. am.
Qed.
```

---

2. *Show that, if* $\mathfrak{a}$, $\mathfrak{b}$, $\mathfrak{c}$ *and* $\mathfrak{d}$ *are four cardinals such that* $\mathfrak{a} < \mathfrak{c}$ *and* $\mathfrak{b} < \mathfrak{d}$ *then* $\mathfrak{a}+\mathfrak{b} < \mathfrak{c}+\mathfrak{d}$ *and* $\mathfrak{a}\mathfrak{b} < \mathfrak{c}\mathfrak{d}$. *(cf. Exercise 21 (c)).*

We first assume $\mathfrak{c} \leq \mathfrak{d}$. If $\mathfrak{d}$ is finite, all quantities are finite and the result is obvious.

```
Lemma exercice6_2: forall a b c d, cardinal_lt a c ->
  cardinal_lt b d ->
  (cardinal_lt (card_plus a b) (card_plus c d) &
    cardinal_lt (card_mult a b) (card_mult c d)).
Proof. assert (forall a b c d, cardinal_le c d -> cardinal_lt a c ->
  cardinal_lt b d ->
  (cardinal_lt (card_plus a b) (card_plus c d) &
    cardinal_lt (card_mult a b) (card_mult c d))).
  ir.  assert (Ha:c <> card_zero). red. ir. rwi H2 H0.
  app (zero_smallest1 H0).
```

```
nin (p_or_not_p (is_finite_c d)). ir.
cp (le_int_is_int H2 H).
assert (is_finite_c a). nin H0. Bnat_tac.
assert (is_finite_c b). nin H1. Bnat_tac.
split. app finite_sum2_lt. bw. bw.  bw.  bw.
nin H0. am. app finite_prod2_lt.  bw. bw. bw. bw. nin H0. am.
```

We have now $\mathfrak{c}+\mathfrak{d}=\mathfrak{c}\mathfrak{d}=\mathfrak{d}$. Such a formula holds if one of $\mathfrak{a}$ or $\mathfrak{b}$ is infinite (the result is trivial is none is infinite). Note that if one of the cardinals is zero, so is the product, thus is $<\mathfrak{d}$.

```
ir. assert (is_infinite_c d). red. ee.  nin H.  nin H3. am. am.
rw (card_mult_commutative c d). rww (product2_infinite H H3 Ha).
rw (card_plus_commutative c d). rw (sum2_infinite H H3).
nin (p_or_not_p (is_finite_c a)). ir.
nin (p_or_not_p (is_finite_c b)). ir.
wri inc_Bnat H4; wri inc_Bnat H5.
assert (is_finite_c (card_plus a b)). fprops.
assert (is_finite_c (card_mult a b)). fprops.
split. app finite_lt_infinite. app finite_lt_infinite.
ir. assert (is_infinite_c b). red. ee. red in H1. ee.  red in H1; ee; am. am.
cp  (finite_le_infinite H4 H6).
split. rw card_plus_commutative. rw (sum2_infinite H7 H6). am.
nin (equal_or_not a card_zero). rw H8.
rw card_mult_commutative. rw zero_prod_absorbing.  app finite_lt_infinite.
wr inc_Bnat. fprops. rw card_mult_commutative.
rww (product2_infinite H7 H6 H8). ir. assert (is_infinite_c a). red. ee.
red in H0. ee.  red in H0; ee; am. am.
nin (p_or_not_p (is_finite_c b)). ir.
cp  (finite_le_infinite H6 H5). split. rw (sum2_infinite H7 H5).
app (cardinal_lt_le_trans H0 H).
nin (equal_or_not b card_zero). rw H8. rw zero_prod_absorbing.
app finite_lt_infinite. wr inc_Bnat. fprops.
rww (product2_infinite H7 H5 H8).  app (cardinal_lt_le_trans H0 H).
```

Here all cardinals are infinite. In particular, they are non-zero. We have $\mathfrak{a}+\mathfrak{b}=\mathfrak{a}\mathfrak{b}=\mathfrak{a}'$ where $\mathfrak{a}'<\mathfrak{d}$ is the greatest of the two cardinals.

```
ir. assert (is_infinite_c b). red. ee. red in H1. ee. red in H1; ee; am. am.
 assert (cardinal_le a b \/ cardinal_le b a).
 nin H0. nin H0. nin H1; nin H1. nin (cardinal_le_total_order1 H0 H1).
 left. rw H12. fprops. nin H12.  nin H12. left. am. nin H12. right. am.
 nin H8.  split. rw card_plus_commutative. rw (sum2_infinite H8 H7). am.
 nin (equal_or_not a card_zero). rw H9.
 rw card_mult_commutative. rw zero_prod_absorbing.  app finite_lt_infinite.
 wr inc_Bnat. fprops. rw card_mult_commutative.
 rww (product2_infinite H8 H7 H9). rw (sum2_infinite H8 H5).  split.
 app (cardinal_lt_le_trans H0 H). nin (equal_or_not b card_zero). rw H9.
 rw zero_prod_absorbing.  app finite_lt_infinite. wr inc_Bnat. fprops.
 rww (product2_infinite H8 H5 H9).  app (cardinal_lt_le_trans H0 H).
```

The result is now clear. If $\mathfrak{c}\geq\mathfrak{d}$, we use commutativity.

```
ir. assert (cardinal_le c d \/ cardinal_le d c).
red in H0. red in H1. ee. red in H0; red in H1; ee.
```

```
  nin (cardinal_le_total_order1 H6 H4).
  left. rw H8. fprops. nin H8.  nin H8. left. am. nin H8. right. am.
  nin H2. app H.
  rw (card_plus_commutative c d). rw (card_mult_commutative c d).
  rw (card_plus_commutative a b). rw (card_mult_commutative a b). app H.
Qed.
```

---

3. *If* E *is an infinite set, the subsets of* E *which are equipotent to* E *is equipotent to* $\mathfrak{P}(E)$ *(use Proposition 3 of no. 4).*

We know that E is equipotent to $E_1 \cup E_2$, where the union is disjoint and both sets are equipotent to E (this is not the hint given by Bourbaki). Let f be a bijection $E_1 \cup E_2 \to E$. Xe may assume $E_1 = E \times \{\alpha\}$. For each subset X of E, let $\bar{X}$ be X as a subset of E_1 (i.e., $\bar{X} = X \times \{\alpha\}$), and $g(X) = f(\bar{X} \cup E_2)$. This a subset of E, and its cardinal is at least the cardinal of $f(E_2)$, which is the cardinal of E, so that $g(X)$ is equipotent to E.

```
Lemma Exercise6_3: forall E, infinite_set E ->
  equipotent (powerset E) (Zo (powerset E) (fun z => equipotent z E)).
Proof. ir. set (Qo:= Zo (powerset E) (fun z : Set => equipotent z E)).
  assert (sub Qo (powerset E)). uf Qo. app Z_sub.
  cp (cardinal_le8 H0). rwi cardinal_le3 H1.
  set (n:= cardinal E). assert (card_plus n n = n).
  assert (cardinal_le n n). uf n. fprops. red in H.  fold n in H.
  ap (sum2_infinite H2 H).
  set (E1:= product E (singleton TPa)).  set (E2:= product E (singleton TPb)).
  assert (equipotent E1 n). eqtrans E.  eqsym. uf E1. fprops. uf n. fprops.
  assert (equipotent E2 n). eqtrans E.  eqsym. uf E2. fprops. uf n. fprops.
  assert (disjoint E1 E2). uf E1; uf E2. app disjoint_union2_pr. fprops.
  cp (card_plus_pr1 H5 H3 H4). rwi H2 H6. ufi n H6. awi H6. nin H6. ee.
  set (f:= fun X => image_by_fun x (union2 (product X (singleton TPa)) E2)).
  assert (forall X, sub X E -> sub (f X) E). red.  ir. ufi f H10.
  assert (is_function x). app bij_is_function.
  ufi image_by_fun H10. awi H10. nin H10. ee. red in H12. wr H8.
  app (inc_pr2graph_target H11 H12). fprops.
  assert (forall X, sub X E -> equipotent (f X) E). ir.
  cp (H9 _ H10).  cp (cardinal_le8 H11). rwi cardinal_le3 H12.
  set (b:= image_by_fun x E2). assert (sub E2 (source x)). rw H7.
  red. ir. app union2_second. cp (equipotent_restriction H13 H6).
  assert (sub (image_by_fun x E2) (f X)). uf f.  uf image_by_fun.
  app image_by_increasing. cp (bij_is_function H6). fprops. red. ir.
  app union2_second. cp (cardinal_le8 H15). rwi cardinal_le3 H16.
  wri cardinal_equipotent H14. wri H14 H16.
  assert (cardinal E2 = cardinal E). aw. eqtrans n. am. uf n. fprops.
  rwi H17 H16. wr cardinal_equipotent. app cardinal_antisymmetry1.
  assert (transf_axioms f (powerset E) Qo). red. ir.
  rwi powerset_inc_rw H11.  uf Qo. Ztac. app powerset_inc. app H9.
  set (F:=  BL f (powerset E) Qo).
  assert (injective F). uf F. app injective_bl_function. uf f. ir.
  cp (bij_is_function H6).
  set (T:= image_by_fun x (union2 (product u (singleton TPa)) E2)).
```

Let Q be the set of subsets of E equipotent to E. The mapping $g(X) = f(\bar{X} \cup E_2)$ is injective $\mathfrak{P}(E) \to Q$ because f is injective and the sets $\bar{X}$ and E_2 are disjoint. Since $Q \subset \mathfrak{P}(E)$, these two sets have the same cardinal.

```
  set_extens. set (y := W (J x0 TPa) x).
  assert (inc (J x0 TPa) (source x)). rw H7. app union2_first. uf E1. aw. ee.
  fprops. rwi powerset_inc_rw H12. app H12. fprops.
  assert (inc y T). uf T. uf image_by_fun. aw. exists (J x0 TPa). split.
  app union2_first. aw. ee. fprops. am. fprops. red.  uf y. app defined_lem.
  fprops. ufi T H18. rwi H14 H18. ufi image_by_fun H18.
  awi H18. nin H18. ee. red in H19. cp (W_pr H15 H19). unfold y in H20.
  assert (inc x1 (source x)). rw H7. nin (union2_or H18).  app union2_first.
  uf E1. awi H21. ee. aw. ee.  am. rwi powerset_inc_rw H13.  app H13. am.
  app union2_second. nin H6. nin H6.  symmetry in H20. cp (H23 _ _ H17 H21 H20).
  nin (union2_or H18).  awi H25.  ee. wri H24 H26. awi H26. am. ufi E2 H25.
  awi H25. ee. wri H24 H27. awi H27. cp H27.
  elim two_points_distinct. am. fprops.
  set (y := W (J x0 TPa) x).
  assert (inc (J x0 TPa) (source x)). rw H7. app union2_first. uf E1. aw. ee.
  fprops. rwi powerset_inc_rw H13. app H13. fprops.
  assert (inc y T). uf T. rw H14. uf image_by_fun. aw. exists (J x0 TPa).
  split. app union2_first. aw. ee. fprops. am. fprops. red.  uf y.
  app defined_lem. fprops. ufi T H18. ufi image_by_fun H18.
  awi H18. nin H18. ee. red in H19. cp (W_pr H15 H19). unfold y in H20.
  assert (inc x1 (source x)). rw H7. nin (union2_or H18).  app union2_first.
  uf E1. awi H21. ee. aw. ee.  am. rwi powerset_inc_rw H12.  app H12. am.
  app union2_second. nin H6. nin H6. symmetry in H20. cp (H23 _ _ H17 H21 H20).
  nin (union2_or H18).  awi H25.  ee. wri H24 H26. awi H26. am. ufi E2 H25.
  awi H25. ee. wri H24 H27. awi H27. cp H27.
  elim two_points_distinct. am. fprops.
  assert (cardinal_le (cardinal (powerset E)) (cardinal Qo)). red. ee.
  fprops. fprops. wr cardinal_le2. rw cardinal_le1. exists F. ee. am. uf F.
  tv. uf F. tv. wr cardinal_equipotent.
  app cardinal_antisymmetry1.
Qed.
```

---

4. *If* E *is an infinite set, the set of all partitions of* E *is equipotent to* $\mathfrak{P}(E)$ *(associate a subset of* $E \times E$ *with each partition of* E*).*

Let ω be a partition, and $\tilde{\omega}$ be the union of all $A \times A$ with $A \in \omega$. We know that $\tilde{\omega} \supset \tilde{\omega}'$ is an order (see chapter one), so that the function $\omega \mapsto \tilde{\omega}$ is injective. Let Q be the set of partitions; this shows $\mathrm{Card}(Q) \leq \mathrm{Card}(\mathfrak{P}(E \times E))$, hence $\mathrm{Card}(Q) \leq \mathrm{Card}(\mathfrak{P}(E)$.

```
Lemma Exercise6_4: forall E, infinite_set E ->
  equipotent (set_of_partition_set E)(powerset E).
Proof. ir.  set (q:=set_of_partition_set E).
  set (f:= BL (fun y=> partition_relation_set y E) q (powerset (coarse E))).
  assert (injective f). uf f. app injective_bl_function. red. ir.
  app powerset_inc. app sub_partition_relation_set_coarse. ufi q H0.
  ufi set_of_partition_set H0. Ztac. am. uf q. uf set_of_partition_set. ir.
  Ztac. clear H1. Ztac. sy.
  app (partition_relation_set_order_antisymmetric H4 H5).
```

```
assert (equipotent_to_subset q (powerset (coarse E))). rw cardinal_le1.
exists f. eee. rwi cardinal_le2 H1.
assert (cardinal_le  (cardinal q) (cardinal (powerset (coarse E)))).
uf cardinal_le. eee.
assert (cardinal (powerset (coarse E)) = cardinal (powerset E)).
rw card_powerset. rw card_powerset.  uf coarse. app card_pow_pr. fprops.
wr cardinal_equipotent.  wr card_mult_pr1.
transitivity (card_mult (cardinal E)(cardinal E)). app card_mult_pr2.
sy. fprops. sy. fprops. wr power_x_2. app equipotent_inf2_inf. fprops.
```

Conversely, consider the mapping $I \mapsto \{I, E - I\}$. This mapping is not injective for $I \subset E$, but it is injective for $I \subset F$, where F is strict subset of E. Since there is at least one element y in E, we may consider $F = E - singletony$.

```
rwi H3 H2. nin (emptyset_dichot E). rwi H4 H. nin H. elim H5.
wr inc_Bnat. app inc0_Bnat. nin H4.
set (F:= complement E (singleton y)).
set (g:= fun u => doubleton u (complement E u)).
assert (forall u v , sub u F -> sub v F -> g u = g v -> u = v). uf g. ir.
nin (doubleton_inj H7). nin H8. am. nin H8.
assert (inc y F).  app H5. rw H8. rw inc_complement. split. am. red. ir.
cp (H6 _ H10). ufi F H11. rwi inc_complement H11. ee. elim H12. fprops.
ufi F H10. rwi inc_complement H10. ee. elim H11. fprops.
```

If I is a non-empty subset of F then $I \mapsto \{I, E - I\}$ is a partition of E.

```
assert (forall u, sub u F -> nonempty u -> inc (g u) q). ir.
assert (sub u E). assert (sub F E). uf F. app sub_complement.
app (sub_trans H6 H8).
uf g. uf q. rw set_of_partition_pr. red. ee.  set_extens.
nin (union_exists H9). nin H10. nin (doubleton_or H11). app H8. wrr H12.
rwi H12 H10. srwi H10. nin H10; am.
nin (p_or_not_p (inc x u)). ir.
assert (inc u (doubleton u (complement E u))). fprops. ap (union_inc H10 H11).
ir.  assert (inc (complement E u) (doubleton u (complement E u))). fprops.
assert (inc x (complement E u)). srw.  intuition.
ap (union_inc H12 H11). ir. nin (doubleton_or H9). rww H10. rw H10.
exists y. srw. split. am. red. ir. cp (H6 _ H11).
ufi F H12. srwi H12. ee. elim H13. fprops.
ir.  nin (doubleton_or H9); nin (doubleton_or H10); rw H11; rw H12.
left. tv. right. app disjoint_complement.  right.
app disjoint_symmetric. app disjoint_complement. au.
```

All that remains to do is to show that $\mathrm{Card}(\mathfrak{P}(E - \{y\}) - \{\emptyset\}) = \mathrm{Card}(\mathfrak{P}(E))$.

```
set (T:= complement (powerset F) (singleton emptyset)).
assert (cardinal_le(cardinal T)  (cardinal q)).
wr cardinal_le3. rw cardinal_le1. exists (BL g T q). ee.
app injective_bl_function. red. ir. ufi T H7. rwi inc_complement H7. ee.
app H6.  app powerset_sub. nin (emptyset_dichot c).  elim H8. rw H9. fprops.
am. ir.  ufi T H7. rwi inc_complement H7.  ufi T H8. rwi inc_complement H8.
ee. app H5.  app powerset_sub. app powerset_sub.  tv. tv.
cp (cardinal_comp_singl_inf y H). fold F in H8.
assert (infinite_set (powerset F)). red. red.   split. fprops. red.
rw card_powerset. ir.
```

```
  assert (card_pow card_two F = card_pow card_two (cardinal F)).
  app card_pow_pr. fprops.  fprops. rwi H10 H9.
  assert (is_cardinal (cardinal F)). fprops. cp (cantor  H11). nin H12.
  cp (le_int_is_int H9 H12). wri H8 H14. nin H. ee. elim H15. am.
  cp (cardinal_comp_singl_inf emptyset H9). fold T in H10. wri H10 H7.
  assert (cardinal (powerset F) = cardinal (powerset E)).
  rw card_powerset. rw card_powerset.  app card_pow_pr. fprops.
  symmetry in H8. awi H8. am. rwi H11 H7. cp (cardinal_antisymmetry1 H2 H7).
  awi H12. am.
Qed.
```

5. If E is an infinite set, the set of all permutations of E is equipotent to $\mathfrak{P}(E)$. (Use Proposition 3 of No. 4 to shat that, for each subset A of E whose complementary does not consist of a single element; there exists a permutation f of E such that A is the set of elements of E which are invariant under f.)

6. Let E, F be two infinite sets such that Card(E) ≤ Card(F). Show that (i) the set of all mappings of E onto F, (ii) the set of all mappings of E into F, and (iii) the set of all mappings of subsets of E into F are all equipotent to $\mathfrak{P}(F)$.

7. Let E, F be two infinite sets such that Card(E) < Card(F). Show that the set of all subsets of F which are equipotent to E and the set of all injections of E into F are both equipotent to then set F^E of all mappings of E into F (for each mapping f of E into F, consider the injection $x \mapsto (x, f(x))$ of E into E × F).

8. Show that the set of well-orderings on an infinite set E (and *a fortiori* the set of orderings on E) is equipotent to $\mathfrak{P}(E)$ (Use Exercise 5).

9. Let E be a non-empty well-ordered set in which every element x other then the least element of E has a predecessor (the greatest element of $]\leftarrow, x[$). Show that E is isomorphic to either **N** or an interval $[0, n$ of **N** (remark that every segment ≠ E is finite by using Proposition 6 of No. 5; then use Theorem 3 of § 2, no. 5).

¶ 10. Let ω or ω_0 denote the ordinal Ord(**N**) (§ 2, Exercise 14). The set of all integers is then a well-ordered set isomorphic to the set of all ordinals < ω: For each integer n we denote again by n (by abuse of language) the ordinal Ord($[0, n[$).

(a) Show that for each cardinal $\mathfrak{a}$ the relation "ξ is an ordinal and Card(ξ) < $\mathfrak{a}$ is collectivizing (use Zermelo's theorem). Let W($\mathfrak{a}$) denote the set of all ordinals ξ such that Card(ξ) < $\mathfrak{a}$.

(b) For each ordinal α > 0 define a function f_α on the well-ordered set O′(α) or ordinals ≤ α by transfinite induction as follows: $f_\alpha(0) = \omega_0 = \omega$, and for each ordinal ξ such that $0 < \xi \leq \alpha$, $f_\alpha(\xi)$ is the least upper bound (§ 2, Exercise 14 (d)) of the set of ordinals ζ such that Card(ζ) ≤ Card($f_\alpha(\eta)$) for at least one ordinal η < ξ. Show that if $0 \leq \eta < \xi \leq \alpha$, then Card($f_\alpha(\eta)$) < Card($f_\alpha(\xi)$) and that, if $\xi \leq \alpha \leq \beta$, then $f_\alpha(\xi) = f_\beta(\xi)$. Put $\omega_\alpha = f_\alpha(\alpha)$; ω_α is said to be the *initial ordinal of index* α. Put $\aleph_\alpha$ = Card(ω_α); $\aleph_\alpha$ is said to be the *aleph of index* α. In particular, $\aleph_0$ = Card(**N**).

(c) Show that for each infinite cardinal $\mathfrak{a}$, the least upper bound λ of the set of ordinals W(α) is an initial ordinal ω_α, and that $\mathfrak{a} = \aleph_\alpha$ (consider the least ordinal μ such that $\omega_\mu \geq$

λ); in other words ω_α is the least ordinal ξ such that $\mathrm{Card}(\xi) = \aleph_\alpha$. For each ordinal α the mapping $\xi \mapsto \aleph_\xi$, defined on $O'(\alpha)$, is an isomorphism of the well-ordered set $O'(\alpha)$ onto the well-ordered set of cardinals $\leq \aleph_\alpha$; in particular $\aleph_{\alpha+1}$ is the least cardinal $> \aleph_\alpha$. Show that if α has no predecessor, then for every strictly increasing mapping $\xi \mapsto \sigma_\xi$ of an ordinal β into α such that $\alpha = \sup_{\xi<\beta} \sigma_\xi$, we have

$$\sum_{\xi<\beta} \aleph_{\sigma_\xi} = \aleph_\alpha.$$

(d) Deduce from (c) that ω_ξ is a normal ordinal functional symbol (§ 2, Exercise 17).

¶ 11. (a) Show that the ordinal ω_α is the least ordinal > 0 which has no predecessor, that ω is indecomposable (§ 2, Exercise 16), and that for each ordinal $\alpha > 0$, $\alpha\omega$ is the least indecomposable ordinal which is $> \alpha$ (note that $n\omega = \omega$ for each integer n). Deduced that

$$(\alpha + 1)\omega = \alpha\omega \text{ for each } \alpha > 0.$$

(b) Deduce from (a) that an ordinal is indecomposable if an only if it is of the form ω^β (use Exercise 18 (d) of § 2).

¶ 12. (a) Show that for each ordinal α and each ordinal $\gamma > 1$, there exists two finite sequences of ordinals (λ_i) and (μ_i) $(1 \leq i \leq k)$ such that

$$\alpha = \gamma^{\lambda_1}\mu_1 + \gamma^{\lambda_2}\mu_2 + \ldots + \gamma^{\lambda_k}\mu_k,$$

where $0 < \mu_i < \gamma$ for each i, and $\lambda_i > \lambda_{i+1}$ for $1 \leq i \leq k-1$ (use Exercise 18 (d) of § 2 and Exercise 3 of § 4). Moreover the sequences (λ_i), (μ_i) are uniquely determined by these conditions. In particular there exists a unique finite deceasing sequence $(\beta_j)_{1\leq j\leq m}$ such that

$$\alpha = \omega^{\beta_1} + \omega^{\beta_2} + \cdots + \omega^{\beta_m}.$$

Let $\phi(\alpha)$ denote the greatest ordinal ω^{β_1} in this sequence.

(b) For each integer n let $f(n) \leq n!$ be the greatest number of elements in the set of ordinals of the form $\alpha_{\sigma(1)} + \alpha_{\sigma(2)} + \cdots + \alpha_{\sigma(n)}$, where $(\alpha_i)_{1\leq i\leq n}$ is an arbitrary sequence of n ordinals and σ runs through the set of permutations of the interval $[1, n]$. Show that

$$f(n) = \sup_{1\leq k\leq n-1} (k.2^{k-1} + 1) f(n-k). \tag{1}$$

(consider first the case where all the $\phi(\alpha_i)$ are equal and show that the largest possible number of distinct ordinals of the desired form is equal to n, by using Exercise 16 (a) of § 2. Then use induction on the number of ordinals α_i for which $\phi(\alpha_i)$ takes the least possible value among the set of ordinals $\phi(\alpha_j)$ $(1 \leq j \leq n)$.) Deduce from (1) that for $n \geq 20$ we have $f(n) = 81 f(n-5)$.

(c) Show that the $n!$ ordinals $(\omega + \sigma(1))(\omega + \sigma(2))\ldots(\omega + \sigma(n))$ where σ runs through the set of permutations of the interval $[1, n]$, are all distinct.

¶ 13. (a) Let $w(\xi)$ be a, ordinal functional symbol (§2, Exercise 17), defined for $\xi \geq \alpha_0$ and such that the relation $\alpha_0 \leq \xi < \xi'$ implies $w(\xi) < w(\xi')$. Show that, if $\xi \geq \alpha_0$, then $w(\xi + \eta) \geq \xi(\xi) + \eta$ for every ordinal η (argue by contradiction). Deduce that there exists α such that $w(\xi) \geq \xi$ for all $\xi \geq \alpha$ (take α to be the least indecomposable ordinal $\geq \alpha_0$; cf Exercise 11 (a)).

(b) Let $f(\xi,\eta)$ be the ordinal functional symbol defined in § 2, Exercise 17(b). Suppose that the relations $\alpha_0 \leq \xi \leq \xi'$ and $\alpha_0 \leq \eta \leq \eta'$ imply $g(\xi,\eta) \leq g(\xi',\eta')$ so that the relations $\alpha_0 \leq \xi \leq \xi'$ and $1 \leq \eta \leq \eta'$ imply $f(\xi,\eta) \leq f(\xi',\eta')$ (§ 2, Exercise 17(d)). Show that for each ordinal β there exist at most a finite number of ordinals η for which the equation $f(\xi,\eta) = \beta$ has at least one solution (Note that if ξ_1 is the least solution of $f(\xi,\eta_1) = \beta$ and if ξ_2 is the least solution of $f(\xi,\eta_2) = \beta$ then the relation $\eta_1 < \eta_2$ implies $\xi_1 > \xi_2$.)

(c) A *critical ordinal* with respect to f is any infinite ordinal $\gamma > \alpha_0$ such that $f(\xi,\gamma) = \gamma$ for all ξ such that $\alpha_0 \leq \xi < \gamma$. Show that a critical ordinal (with respect to f) has no predecessor. If there exists a set A or ordinals such that $f(\xi,\gamma) = \gamma$ for all $\xi \in A$, and if γ is the least upper bound of A, show that γ is a critical ordinal.

(d) let $h(\xi) = f(\xi,\xi)$ (defined for $\xi \geq \alpha_0$); Defined inductively $\alpha_1 = \alpha_0 + 2$ $\alpha_{n+1} = h(\alpha_n)$ for $n \geq 1$. Show that that the least upper bound of the sequence (α_n) is a critical ordinal with respect to f.

(e) Show that the least upper bound of every set of critical ordinals with respect to f is again a critical ordinal, and that every critical ordinal is indecomposable (note that $f(\xi,\eta+1) \geq \omega(\xi)+\eta \geq \xi+\eta$ for all $\xi \geq \alpha_0$.

¶ 14. (a) Show that if $\alpha \geq 2$ and if β has no predecessor, then α^β is an indecomposable ordinal (cf § 2, Exercise16 (a)); if α is finite and if $\beta = \omega\gamma$, then $\alpha^\beta = \omega^\gamma$; if α is infinite and if π is the greatest indecomposable ordinal $\leq \alpha$, then $\alpha^\beta = \pi^\beta$ (use Exercise 11).

(b) An ordinal δ is critical with respect to the functional symbol $f(\xi,\eta) = \xi\eta$ if and only if, for each α such that $1 < \alpha \leq \delta$, the equation $\delta = \alpha^\xi$ has a solution; the unique solution ξ if this equation is then indecomposable (Use Exercise 13 (e), together with Exercise 18 (d) of § 2). Conversely, for each $\alpha > 1$ and each indecomposable ordinal π, α^π is a critical ordinal with respect to $\xi\eta$ (use Exercise 13 (c)). Deduced that δ is a critical ordinal with respect to $\xi\eta$ if and only if δ is of the form ω^{ω^μ} (cf. Exercise 11 (b)).

(c) For an ordinal ϵ to be critical with respect to the functional symbol $f(\xi,\eta) = \xi^\eta$, i.e. such that $\gamma^\epsilon = \epsilon$ for each γ satisfying $2 \leq \gamma \leq \epsilon$, it is sufficient that $2^\epsilon = \epsilon$. Show that the least critical ordinal ϵ_0 with respect to ξ^η is countable (cf. Exercise 13 (d)).

¶ 15. Let γ be an ordinal > 1, and for each ordinal α let $L(\alpha)$ denote the set of exponents λ_i in the expression for α given in Exercise 12 (a).

(a) Show that $\lambda_i \leq \alpha$ for each $\lambda_i \in L(\alpha)$ and that $\lambda_i = \alpha$ for one of these ordinal only if $\alpha = 0$ or if α is a critical ordinal with respect to ξ^η (Exercise 14 (c)).

(b) Define $L_n(\alpha)$ by induction on n as follows: $L_1(\alpha) = L(\alpha)$ and $L_n(\alpha)$ is the union of the sets $L(\beta)$ as β runs through $L_{n-1}(\alpha)$. Show that there exists an integer n_0 such that $l_{n+1}(\alpha) = L_n(\alpha)$ whenever $n \geq n_0$, and that the elements of $L_n(\alpha)$ are then either 0 or critical ordinals with respect to ξ^η (Argue by contradiction: for each n, consider the set $M_n(\alpha)$ of elements $\beta \in L_n(\alpha)$ such that $\beta \notin L(\beta)$, and assume that $L_n(\alpha)$ if not empty for any n; use (a) to obtain a contradiction.)

16. Every totally ordered set has a well-ordered cofinal subset (§ 2, Exercise 2). The least of the ordinals Ord(M) of the well-ordered cofinal subsets M of E is said the *final character* of E.

(a) An ordinal ξ is said to be *regular* if it is equal to its final character, and *singular* otherwise. Show that every infinite regular ordinal is an an initial ordinal ω_α (Exercise 10). Conversely, every initial ordinal ω_α, whose index α is either 0 or has a predecessor, is a regular

ordinal. An initial ordinal ω_α whose index α has no predecessor is singular if $0 < \alpha < \omega_\alpha$; in particular, ω_ω is the least infinite singular initial ordinal.

(b) An initial ordinal ω_α is said to be *inaccessible* if it is regular and its index α has no predecessor. Show that if $\alpha = 0$, then $\omega_\alpha = \alpha$; in other words, α is a critical ordinal with respect to the normal functional symbol ω_η (Exercise 10 (d) and 13 (c)). Let κ be the least critical ordinal with respect to this functional symbol. Show that ω_κ is singular, with final character ω (cf Exercise 13(d)). In other words, there exists no inaccessible ordinal ω_α such that $0 < \alpha \leq \kappa$ (At present, it is not known whether or not there exist inaccessible ordinals other than ω).

(c) Show that there exists only one regular ordinal which is cofinal in a given totally ordered set E; this ordinal is equal to the final character of E, and if E is not empty and has no greatest element, it is an initial ordinal. If $\omega_{\bar\alpha}$ is the final character of ω_α, then $\bar\alpha \leq \alpha$; and ω_α is regular if and only if $\alpha = \bar\alpha$.

(d) Let ω_α be a regular ordinal and let I be a well-ordered set such that $\mathrm{Ord}(I) < \omega_\alpha$. Show that, for each family $(\xi_\iota)_{\iota\in I}$ of ordinals such that $\xi_\iota < \omega_\alpha$ for all $\iota \in I$, we have $\sum_{\iota\in I} \xi_\iota < \omega_\alpha$.

17. A cardinal $\aleph_\alpha$ is said to be *regular* (resp. *singular*) if the initial ordinal ω_α is regular (resp. singular). For $\aleph_\alpha$ to be regular if is necessary and sufficient that for every family $(\mathfrak{a}_\iota)_{\iota\in I}$ of cardinals such that $\mathrm{Card}(I) < \aleph_\alpha$ and $\mathfrak{a}_\iota < \aleph_\alpha$ for all $\iota \in I$, we have

$$\sum_{\iota\in I} \mathfrak{a}_\iota < \aleph_\alpha.$$

$\aleph_\omega$ is the least singular cardinal.

¶ 18. (a) For each ordinal α and each cardinal $\mathfrak{m} \neq 0$ we have $\aleph_{\alpha+1}^{\mathfrak{m}} = \aleph_\alpha^{\mathfrak{m}} \cdot \aleph_{\alpha+1}$ (reduce to the case where $\mathfrak{m} < \aleph_{\alpha+1}$ and consider the mappings of the cardinal $\mathfrak{m}$ into the ordinal $\omega_{\alpha+1}$).

(b) Deduce from (a) that, for each ordinal γ such that $\mathrm{Card}(\gamma) \leq \mathfrak{m}$ we have $\aleph_{\alpha+\gamma}^{\mathfrak{m}} = \aleph_\alpha^{\mathfrak{m}} \cdot \aleph_{\alpha+\gamma}^{\mathrm{Card}(\gamma)}$ (by transfinite induction on γ).

(c) Deduce from (b) that, for each ordinal α such that $\mathrm{Card}(\alpha) \leq \mathfrak{m}$, we have $\aleph_\alpha^{\mathfrak{m}} = 2^{\mathfrak{m}} \cdot \aleph_\alpha^{\mathrm{Card}(\alpha)}$.

¶ 19. (a) Let α and β be two ordinals such that α has no predecessor, and let $\xi \to \sigma_\xi$ be a strictly increasing mapping of the ordinal ω_β into the ordinal α such that $\sup_{\xi<\omega_\beta} \sigma_\xi = \alpha$. Show that

$$\aleph_\alpha^{\aleph_\beta} = \prod_{\xi<\omega_\beta} \aleph_{\sigma_\xi}.$$

(With each mapping f of the ordinal ω_β into the ordinal ω_α associate an injective mapping $\bar f$ of ω_β into the set of all ω_{σ_ξ} ($\xi < \omega_\beta$) such that $f(\zeta) \leq \bar f(\zeta)$ for all $\zeta < \omega_\beta$. Calculate the cardinal of the set of mappings f associated with then same $\bar f$ and observe that

$$\mathfrak{m} = \prod_{\xi<\omega_\beta} \aleph_{\sigma_\xi} \geq 2^{\mathrm{card}(\omega_\beta)}$$

and $\mathfrak{m} \geq \aleph_\alpha$ (cf. § 3, Exercise 3).)

(b) Let $\bar\alpha$ be the ordinal such that $\omega_{\bar\alpha}$ is the final character of ω_α. Show that that $\aleph_\alpha^{\aleph_{\bar\alpha}} > \aleph_\alpha$ and that if there exists $\mathfrak{n}$ such that $\aleph_\alpha = \mathfrak{n}^{\alpha_\gamma}$ then $\gamma < \bar\alpha$ (use (a) and Exercise 3 of § 3).

(c) Show that if $\lambda < \bar\alpha$ then

$$\aleph_\alpha^{\aleph_\beta} = \sum_{\xi<\alpha} \aleph_\alpha^{\aleph_\lambda}$$

(argue as in Exercise 18 (a)).

¶ 20. (a) For a cardinal $\mathfrak{a}$ to be regular (Exercise 17) it is necessary that for every cardinal $\mathfrak{b} \neq 0$ we should have

$$\mathfrak{a}^{\mathfrak{b}} = \mathfrak{a}. \sum_{\mathfrak{m}<\mathfrak{a}} \mathfrak{m}^{\mathfrak{b}}.$$

(Use Exercise 19 and consider separately the cases (i) $\mathfrak{b}$ is finite, (ii) $\aleph_0 \leq \mathfrak{b} < \mathfrak{a}$, (iii) $\mathfrak{b} \geq \mathfrak{a}$; also use Exercise 3 of § 3.) The generalized continuum hypothesis implies that the above condition is also sufficient.

(b) Show that if a cardinal $\mathfrak{a}$ is such that $\mathfrak{a}^{\mathfrak{m}} = \mathfrak{a}$ for every cardinal $\mathfrak{m}$ such that $0 < \mathfrak{m} < \mathfrak{a}$, then $\mathfrak{a}$ is regular (use Exercise 3 of § 3).

(c) Show that the proposition "for every regular cardinal $\mathfrak{a}$ and every cardinal $\mathfrak{m}$ such that $0 < \mathfrak{m} < \mathfrak{a}$, we have $\mathfrak{a} = \mathfrak{a}^{\mathfrak{m}}$ is equivalent to the generalized continuum hypothesis (use (a)).

¶ 21. An infinite cardinal $\mathfrak{a}$ is said to be *dominant* if for each pair of cardinals $\mathfrak{m} < \mathfrak{a}$, $\mathfrak{n} < \mathfrak{a}$ we have $\mathfrak{m}^{\mathfrak{n}} < \mathfrak{a}$.

(a) For $\mathfrak{a}$ to be dominant it is sufficient that $2^{\mathfrak{m}} < \mathfrak{a}$ for every cardinal $\mathfrak{m} < \mathfrak{a}$.

(b) Define inductively a sequence $(\mathfrak{a}_n)$ of cardinals as follows: $\mathfrak{a}_0 = \aleph_0$, $\mathfrak{a}_{n+1} = 2^{\mathfrak{a}_n}$ Show that the sum $\mathfrak{b}$ of the sequence $\mathfrak{a}_n$ is a dominant cardinal. $\aleph_0$ and $\mathfrak{b}$ are the two smallest dominant cardinals.

(c) Show that $\mathfrak{b}^{\aleph_0} = \aleph_0^{\mathfrak{b}} = 2^{\mathfrak{b}}$ (Note that $2^{\mathfrak{b}} \leq \mathfrak{b}^{\aleph_0}$). Deduce that $\mathfrak{b}^{\aleph_0} = (2^{\mathfrak{b}})^{\mathfrak{b}}$, although $\mathfrak{b} < 2^{\mathfrak{b}}$ and $\aleph_0 < \mathfrak{b}$.

¶ 22. A cardinal $\aleph_\alpha$ is said to be *inaccessible* if the ordinal ω_α is inaccessible (Exercise 16 (b)). We have then $\omega_\alpha = \alpha$ if $\omega_\alpha \neq \omega_0$. A cardinal $\mathfrak{a}$ is said to be *strongly inaccessible* if it is inaccessible and dominant.

(a) The generalized continuum hypothesis implies that every inaccessible cardinal is strongly inaccessible.

(b) For a cardinal $\mathfrak{a} \geq 3$ to be strongly inaccessible it is necessary and sufficient that, for each family $(\mathfrak{a}_\iota)_{\iota \in I}$ of cardinals such that

$$\text{Card(I)} < \mathfrak{a} \text{ and } \mathfrak{a}_\iota < \mathfrak{a}$$

for all $\iota \in I$, we should have $\prod_{\iota \in I} \mathfrak{a}_\iota < \mathfrak{a}$.

(c) For an infinite cardinal $\mathfrak{a}$ to be strongly inaccessible it is necessary and sufficient that it should be dominant (Exercise 21) and that it should satisfy one of the following two conditions: (i) $\mathfrak{a}^{\mathfrak{b}} = \mathfrak{a}$ for every cardinal $\mathfrak{b}$ such that $0 < \mathfrak{b} < \mathfrak{a}$; (ii) $\mathfrak{a}^{\mathfrak{b}} = \mathfrak{a}.2^{\mathfrak{b}}$ for every cardinal $\mathfrak{b} > 0$ (Use Exercises 20 and 21).

¶ 23. Let α be an ordinal > 0. A mapping f of the ordinal α into itself is said to be *divergent* if for each ordinal $\lambda_0 < \alpha$ there exists an ordinal $\mu_0 < \alpha$ such that the relation $\mu_0 \leq \xi < \alpha$ implies $\lambda_0 \leq f(\xi) < \alpha$. (this condition may be written as $\lim_{\xi \to \alpha, \xi < \alpha} f(\xi) = \alpha$)

(a) Let ϕ be a strictly increasing mapping of an ordinal β into α such that

$$\phi(\sup_{\zeta<\gamma} \zeta) = \sup_{\zeta<\gamma} \phi(\zeta) \text{ for all } \gamma < \beta,$$

and such that

$$\sup_{\zeta<\beta} \phi(\zeta) = \alpha.$$

(if we extend ϕ be defining $\phi(\beta) = \alpha$, the conditions above signify that ϕ is continuous) Then there exists a divergent mapping f of α into itself, such that $f(\xi) < \xi$ for all ξ satisfying $0 < \xi < \alpha$, if and only if there exists a divergent mapping of β into itself of the same type.

(b) Deduce from (a) that there exists a divergent mapping of ω_α into itself, such that $f(\xi) < \xi$ for all ξ satisfying $0 < \xi < \alpha$ if and only if the final character of ω_α is ω_0. (If ω_α is a regular ordinal $> \omega_0$, defined inductively a strictly increasing sequence (η_n) as follows: $\eta_1 = 1$, and η_{n+1} is the least ordinal ζ such that $f(\xi) > \eta_n$ for all $\xi \geq \zeta$.)

(c) Let $\omega_{\bar{\alpha}}$ be the final character of ω_α (Exercise 16). Show that, if $\bar{\alpha} > 0$ and if f is a mapping of ω_α into itself such that $f(\xi) < \zeta$ for all ξ such that $0 < \xi < \omega_\alpha$, then there exists an ordinal λ_0 such that the set of solutions to the equation $f(\xi) = \lambda_0$ has a cardinal $\geq \aleph_{\bar{\alpha}}$.

¶ 24. Let $\mathfrak{F}$ be a set of subsets of a set E such that for every $A \in \mathfrak{F}$ we have $\mathrm{Card}(A) = \mathrm{Card}(\mathfrak{F}) = \mathfrak{a} \geq \aleph_0$. Show that E has a subset P such that $\mathrm{Card}(P) = \mathfrak{a}$ and such that no set of $\mathfrak{F}$ is contained in P (If $\mathfrak{a} = \aleph_\alpha$, define by transfinite induction two injective mappings $\xi \to f(\xi)$, $\xi \to g(\xi)$ of ω_α into E such that the sets $P = f(\omega_\alpha)$ and $Q = g(\omega_\alpha)$ do not intersect and such that each of them meets every subset $A \in \mathfrak{F}$.)

(b) Suppose, moreover, that for each subset $\mathfrak{G}$ of $\mathfrak{F}$ such that $\mathrm{Card}(\mathfrak{G}) < \alpha$, the complement in E of the union of the sets $A \in \mathfrak{G}$ has cardinal $\geq \mathfrak{a}$. Show that E then has a subset P such that $\mathrm{Card}(P) = \mathfrak{a}$ and such that, for each $A \in \mathfrak{G}$, $\mathrm{card}(P \cap A) < \mathfrak{a}$ (similar method).

¶ 25. (a) Let $\mathfrak{F}$ be a covering of an infinite set E. The *degree of disjointness* of $\mathfrak{F}$ is the least cardinal $\mathfrak{c}$ such that $\mathfrak{c}$ is *strictly greater* than the cardinals $\mathrm{Card}(X \cap Y)$ for each pair of distinct sets $X, Y \in \mathfrak{F}$. If $\mathrm{Card}(E) = \mathfrak{a}$ and $\mathrm{Card}(\mathfrak{F}) = \mathfrak{b}$, show that $\mathfrak{b} \leq \mathfrak{a}^{\mathfrak{c}}$ (note that a subset of E of cardinal $\mathfrak{c}$ is contained in at most one set of $\mathfrak{F}$).

(b) Let ω_α be an initial ordinal and let F be set such that $2 \leq \mathfrak{p} = \mathrm{Card}(F) < \aleph_\alpha$. Let E be the set of mappings of segments of ω_α, other then ω_α itself, into F. The we have $\mathrm{Card}(E) \leq \mathfrak{p}^{\aleph_\alpha}$. For each mapping f of ω_α into F, let K_f be the subset of E consisting of the restrictions of f to the segments of ω_α (other than ω_α itself). Show that the set $\mathfrak{F}$ of subsets of K_f is a covering of E such that $\mathrm{Card}(\mathfrak{F}) = \mathfrak{p}^{\aleph_\alpha}$ and that its degree of disjointness is equal to ω_α.

(c) Let E be an infinite set of cardinal $\mathfrak{a}$ and let $\mathfrak{c}$, $\mathfrak{p}$ be two cardinal > 1 such that $\mathfrak{p} < \mathfrak{c}$, $\mathfrak{p}^{\mathfrak{m}} < \mathfrak{a}$ for all $\mathfrak{m} < \mathfrak{c}$ and $\mathfrak{a} = \sum_{\mathfrak{m}<\mathfrak{c}} \mathfrak{p}^{\mathfrak{m}}$. Deduce from (b) that there exists a covering $\mathfrak{F}$ of E consisting of sets of cardinal $\mathfrak{c}$, with degree of disjunction equal to $\mathfrak{c}$ and such that $\mathrm{Card}(\mathfrak{F}) = \mathfrak{p}^{\mathfrak{c}}$. In particular, if E is countably infinite, there exists a covering $\mathfrak{F}$ of E by infinite sets such that $\mathrm{card}(\mathfrak{F}) = 2^{\aleph_0}$ and such that the intersection of any two sets of $\mathfrak{F}$ is *finite*.

¶ 26. Let E be an infinite set and let $\mathfrak{F}$ be a set of subsets of E such that for $A \in \mathfrak{F}$ we have

$$\mathrm{card}(A) = \mathrm{card}(\mathfrak{F}) = \mathrm{card}(E) = \mathfrak{a} \geq \aleph_0.$$

Show that there exists a partition $(B_\iota)_{\iota \in I}$ of E such that

$$\mathrm{card}(I) = \mathrm{card}(B_\iota) = \mathfrak{a}$$

for all $\iota \in I$ and such that $A \cap B_\iota \neq \emptyset$ for all $A \in \mathfrak{F}$ and all $\iota \in I$. (With the notation of Exercise 24 (a), consider first a surjective mapping f of ω_α into $\mathfrak{F}$ such that for each $A \in \mathfrak{F}$ the set of all $\xi \in \omega_\alpha$ such that $f(\xi) = A$ has cardinal equal to $\mathfrak{a}$. Then, by transfinite induction, define a bijection g of ω_α onto E such that $g(\xi) \in f(\xi)$ for every $\xi \in \omega_\alpha$.)

¶ 27. Let L be an infinite set and let $(E_\lambda)_{\lambda\in L}$ be a family of sets indexed by L. Suppose that for each integer $n > 0$ the set of $\lambda \in L$ such that $\mathrm{card}(E_\lambda) > n$ is equipotent to L. Show that there exists a subset F of the product $E = \prod_{\lambda\in L} E_\lambda$, such that $\mathrm{Card}(F) = 2^{\mathrm{Card}(L)}$, and such that F has the following property: for each finite sequence $(f_k)_{1\le k\le n}$ of distinct elements of F there exists $\lambda \in L$ such that the elements $f_k(\lambda) \in E_\lambda$ $(1 \le k \le n)$ are all distinct. (Show first that there exists a partition $(L_j)_{j\in\mathbf{N}}$ of L such that $\mathrm{Card}(L_j) = \mathrm{Card}(L)$ for all j, and such that $\mathrm{Card}(E_\lambda) \ge 2^j$ for each $\lambda \in L_j$. Hence reduce to the case where L is the sum of the countable family of sets X^j $(j \ge 1)$, where X is an infinite set, and $E_\lambda = 2^j$ for each $\lambda \in X^j$. With each mapping $g \in 2^X$ of X into 2, associate the element $f \in E$ such that $f(\lambda) = (g(x_1),\dots,g(x_j))$ whenever $\lambda = (x_k)_{1\le k\le j} \in X^j$; show that the set F of elements $f \in E$ so defined has the required property.)

¶ 28. Let E be an infinite set and let $(\mathfrak{X}_i)_{1\le i\le m}$ be a finite partition of the set $\mathfrak{F}_n(E)$ of subsets of E having n elements. Show that there exists an index i and an infinite subset F of E such that every subset of F with n elements belongs to $\mathfrak{X}_i$. (Proof by induction on n. For each $a \in E$ show that there exists an index $j(a)$ and an infinite subset $M(a)$ of $E - \{a\}$ such that, for every subset A of $M(a)$ with $n-1$ elements, $\{a\} \cup A$ belongs to $\mathfrak{X}_{j(a)}$. Then define a sequence (a_i) of elements of E as follows: a_1 is an arbitrary element of E, a_2 is an arbitrary element of $M(a_1)$, a_3 is defined in terms of $M(a_1)$ and a_2 in the same way as a_2 was defined in terms of E and a_1, and so on. Show that the set F of elements of a suitable subsequence of the sequence (a_i) satisfies the required conditions).

29. In an ordered set E, every finite union of Noetherian subsets (with respect to the induced ordering) is Noetherian.

(b) An ordered set E is Noetherian if and only if for each $a \in E$, the interval $]a, \to [$ is Noetherian.

(c) Let E be an ordered set such that the ordered set obtained by endowing E with the opposite ordering is Noetherian. Let u be a letter and let $T\{u\}$ be a term. Show that there exists a set U and a mapping f of E onto U such that for each $x \in E$ we have $f(x) = T\{f^{(x)}\}$, where $f^{(x)}$ denotes the mapping of $]\leftarrow, x[$ onto $]\leftarrow, f(x)[$ which coincides with f on this interval. Furthermore U and f are determined uniquely by this condition.

(d) Let E be a Noetherian ordered set such that every finite subset of E has a least upper bound in E. Show that, if E has a least element, then E is a complete lattice (§ 1, Exercise 11); and that if E has no least element, the set E' obtained by adjoining a least element to E (§ 1, no. 7, Proposition 3) is a complete lattice.

30. Let E be a lattice such that the set obtained by endowing E with the opposite ordering is Noetherian. Show that every element $a \in E$ can be written as $\sup(e_1, e_2, \dots, e_n)$ where $e_1,\dots,e_n$ are irreducible (§ 4, Exercise 7; show first that there exists an irreducible element e such that $a = \sup(e, b)$ if a is not irreducible). Generalize Exercise 7 (b) of § 4 to E; also generalize Exercises 8(b) and 9(b) of § 4.

¶ 31. Let A be an infinite set and let E be the set <of all infinite subsets of A, ordered by inclusion. Show that E is completely ramified (§ 2, Exercise 8) but not antidirected (§ 1, Exercise 23) and that E has an antidirected cofinal subset F. (Consider first the set $\mathfrak{D}(A)$ of countable infinite subsets of A (which is cofinal in E) and let $Z = R_0(\mathfrak{D}(A))$ (§ 1, Exercise 23). Write Z in the form $(z_\lambda)_{\lambda\in L}$, where L is a well-ordered set, and take F to be a set of countable subsets

X_n^λ, where λ runs through a suitable subset of L, $n \in \mathbf{N}$, $X_m^\lambda \supset X_n^\lambda$ whenever $m \leq n$, $X_n^\lambda - X_{n+1}^\lambda$ is infinite for all $n \geq 0$ and $\cup_{n\in\mathbf{N}} X_n^\lambda = \emptyset$; the X_n^λ, are to be defined by transfinite induction in such a way that the images of the sets X_n^λ, under the canonical mapping $r : \mathfrak{D}(A) \to Z$ (§ 1, Exercise 23) are mutually disjoint and form a cofinal subset of Z.)

¶ 32. * Let (M_n), (P_n) be two sequences of mutually disjoint finite sets (not all empty), indexed by the set $\mathbf{Z}$ of rational integers. Let $\alpha_n = \mathrm{Card}(M_n)$, $\beta_n = \mathrm{card}(P_n)$. Suppose that there exists an integer $k > 0$ such that for each $n \in \mathbf{Z}$ and each integer $l \geq 1$ we have

$$\alpha_n + \alpha_{n+1} \cdots + \alpha_{n+l} \leq \beta_{n-k} + \beta_{n-k+1} + \cdots + \beta_{n+l+k},$$

$$\beta_n + \beta_{n+1} \cdots + \beta_{n+l} \leq \alpha_{n-k} + \alpha_{n-k+1} + \cdots + \alpha_{n+l+k}.$$

Let M be the union of the family (M_n) and let P be the union of the family (P_n). Show that there exists a bijection ϕ of M onto P such that

$$\phi(M_n) \subset \bigcup_{i=n-k-1}^{n+k+1} P_i \text{ and } \overset{-1}{\phi}(P_n) \subset \bigcup_{i=n-k-1}^{n+k+1} M_i$$

for each $n \in \mathbf{Z}$ (consider a total ordering on each M_n (resp. P_n) and take M (resp. P) to be the ordinal sum (§ 1, Exercise 3) of the family $(M_n)_{n\in\mathbf{Z}}$ (resp. $(P_n)_{n\in\mathbf{Z}}$). If n_0 is an index such that $M_{n_0} \neq \emptyset$ consider the isomorphisms of M onto P which transform the least element of M_{n_0} into one of the elements of $\cup_{j=n_0-k}^{n_0+k} P_j$ and show that one of these isomorphisms satisfies the required conditions. Let δ be the least of the numbers

$$\beta_{n-k} + \beta_{n-k+1} + \cdots + \beta_{n+l+k} - (\alpha_n + \alpha_{n+1} \cdots + \alpha_{n+l}),$$

$$\alpha_{n-k} + \alpha_{n-k+1} + \cdots + \alpha_{n+l+k} - (\beta_n + \beta_{n+1} \cdots + \beta_{n+l})$$

for all $n \in \mathbf{Z}$ and all $l \geq 1$. If $n \in \mathbf{Z}$ and $l \geq 1$ are such that, for example $\beta_{n-k} + \beta_{n-k+1} + \cdots + \beta_{n+l+k} = \delta + \alpha_n + \alpha_{n+1} \cdots + \alpha_{n+l}$ we may take ϕ to be such that the least element of P_{n-k} is the image under ϕ of the least element of M_n. *

¶ 33. Soient $\mathfrak{a}$, $\mathfrak{b}$ deux cardinaux tels que $\mathfrak{a} \geq 2$, $\mathfrak{b} \geq 1$, l'un au moins des deux étant infini. Soient E un ensemble, $\mathfrak{F}$ une partie de $\mathfrak{P}(E)$, telle que $\mathrm{card}(\mathfrak{F}) > \mathfrak{a}^\mathfrak{b}$ et $\mathrm{card}(X) \leq \mathfrak{b}$ pour tout $X \in \mathfrak{F}$. On se propose de montrer qu'il existe une partie $\mathfrak{G} \subset \mathfrak{F}$ telle que $\mathrm{card}(\mathfrak{G}) > \mathfrak{a}^\mathfrak{b}$ et que deux quelconques des ensembles appartenant à $\mathfrak{G}$ aient *la même intersection.* On pourra procéder de la façon suivante.

a) soit $\mathfrak{c}$ le plus petit des cardinaux $> \mathfrak{a}^\mathfrak{b}$ et soit Ω le plus petit ordinal de cardinal $\mathfrak{c}$. On considère une application injective $\nu \mapsto X(\nu)$ de Ω dans $\mathfrak{F}$ et on pose $M = \cup_{\nu\in\Omega} X(\nu)$; on peut supposer que $\mathrm{card}(M) = \mathfrak{c}$, et il y a donc une bijection $\nu \mapsto x_\nu$ de Ω sur M, ordonnant M.

b) Pour tout $\nu \in \Omega$ soit ρ_ν l'ordinal type d'ordre du sous-ensemble $X(\nu)$ de M (on a $\mathrm{Card}(\rho_\nu) \geq \mathfrak{b}$) et soit $\mu \mapsto y_\mu^{(\nu)}$ l'unique application bijective croissante de ρ_ν sur $X(\nu)$. On note M_μ l'ensemble des $y_\mu^{(\nu)}$ lorsque ν parcourt Ω. Montrer qu'il existe au moins un ordinal μ tel que $\mathrm{Card}(M_\mu) = \mathfrak{c}$. On désigne par α le *plus petit* de ces ordinaux; la réunion de M_γ pour $\gamma < \alpha$ a un cardinal $\leq \mathfrak{a}^\mathfrak{b} < \mathfrak{c}$.

c) Montrer qu'il existe une partie $N_0 \subset \Omega$ telle que $\mathrm{Card}(N_0) = c$ et que l'application $\nu \mapsto y_\alpha^{(\nu)}$ de N_0 dans M soit injective. Montrer, par récurrence ur β, qu'il existe une partie $N_\beta \subset N_0$ de cardinal $\mathfrak{c}$ telle que l'élément $y_\lambda^{(\nu)} = z_\lambda$ soit indépendant de ν pour $\nu \in N_\beta$ et pour tout $\lambda \leq \beta$. Montrer que l'intersection N des N_β pout $\beta < \alpha$ a pour cardinal $\mathfrak{c}$ (considérer son complémentaire). Soit Q l'ensemble des z_λ pour $\lambda < \alpha$.

d) Pour tout $\nu \in N$, on définit par récurrence un ordinal λ_ν par la condition suivante : c'est le plus petit ordinal dans N tel que $y_\alpha^{(\lambda_\nu)}$ soit un majorant strict, dans M, de la réunion des $X(\lambda_\mu)$ pour $\mu < \nu$, $\mu \in N$. Montrer que pour $\mu < \nu$ dans N on a $(\lambda_\mu) \cap (\lambda_\nu) = Q$.

Chapter 11

Theorems, Notations, Definitions

List of all theorems of Bourbaki, with their Coq equivalent.

Theorems of Chapter 1

Proposition 1 (*order_cor_pr*) « A correspondence Γ between E and E is an ordering on E if and only if ...» , [12].

Proposition 2 (*decreasing_composition*) says that $u(v(x)) \geq x$ and $v(u(x)) \geq x$ and decreasing imply $u \circ v \circ u = u$ and $v \circ u \circ v = v$, [22].

Proposition 3 (*adjoin_greatest*) says that we can add a greatest element to an ordered set, [25].

Proposition 4 (*compare_inf_sup1* and *compare_inf_sup2*) characterizes the supremum and infimum of a subset, [31].

Proposition 5 (*sup_increasing* and *inf_decreasing*) says that $\sup_A$ and $\inf_A$ are increasing functions of the set A, [31].

Proposition 6 (*sup_increasing2* and *inf_decreasing2*) says that $\sup f$ and $\inf f$ are increasing functions of the function f, [32].

Proposition 7 (*sup_distributive1* and *inf_distributive2*) asserts associativity of sup, [32].

Proposition 8 (*sup_in_product* and *inf_in_product*) characterizes supremum and infimum in a product, [33].

Proposition 9 (*sup_induced2* and 3 variants) characterizes supremum of a subset of a subset, [34].

Proposition 10 (*right_directed_maximal* and *left_directed_minimal*) says that «in a right directed ordered set E, a maximal element a is the greatest element of E», [35].

Proposition 11 (*total_order_monotone_injective* and *total_order_increasing_morphism*) characterizes increasing functions and morphism on totally ordered sets, [37].

Proposition 12 (*sup_in_total_order* and *inf_in_total_order*) characterizes supremum and infimum in a totally ordered set, [37].

Proposition 13 (*intersection_interval*) says that in a lattice, the intersection of two intervals is an interval, [38].

Theorems of Chapter 2

Proposition 1 (*well_ordered_segment*) says that «in a well-ordered set E, every segment of E other than E itself is an interval $]\leftarrow, a[$, where $a \in$ E, [44].

Proposition 2 (*isomorphism_set_of_segments_iso* and *set_of_segments_worder*) studies $x \mapsto]\leftarrow, x[$ [45].

Proposition 3 (*worder_merge*) studies the supremum of compatible well-orderings, [46].

Lemma 1 (*order_merge1* and *order_merge2*) is a helper for Proposition 3.

Lemma 2 (*transfinite_principle1* and *transfinite_principle2*) is a helper for C59.

Criterion C59 (*transfinite_principle*) is the principle of transfinite induction, [47].

Criterion C60 (*transfinite_definition* and *transfinite_definition_stable*) (Definition of a mapping by transfinite induction), [48].

Lemma 3 (*Zermelo_aux*) is a helper for Theorem 1.

Theorem 1 (*Zermelo*) says that «every set E can be well-ordered», [51].

Proposition 4 (*Zorn_aux*) is a generalization of Zorn's lemma [51].

Theorem 2 (*Zorn_lemma*) says «every inductive ordered set has a maximal element», [51].

Corollary 1 (*inductive_max_greater*).

Corollary 2 (*maximal_in_powerset* and *minimal_in_powerset*).

Theorem 3 (*isomorphism_worder*) studies existence and uniqueness of an isomorphism between two well-ordered sets, [52].

Lemma 4 (*increasing_function_segments*), [53].

Corollary 1 (*unique_isomorphism_onto_segment*), [53].

Corollary 2 (*bij_pair_isomorphism_onto_segment*), [53].

Corollary 3 (*isomorphic_subset_segment*) [54].

Theorems of Chapter 3

Proposition 1 *cardinal_equipotent* «two sets X and Y are equipotent if and only if their cardinals are equal», [60].

Theorem 1 (*wordering_cardinal_le*) says that the ordering between cardinals is a well-ordering, [62].

Corollary 1 (*cardinal_le_total_order*).

Corollary 2 (*cardinal_antisymmetry2*).

Proposition 2 (*cardinal_supremum*) says that a family of cardinals has a supremum, [63].

Proposition 3 (*surjective_cardinal_le*) says «if there exists a surjection f of X onto Y, then Card(Y) ≤ Card(X), [64].

Proposition 4 (*cardinal_prod_pr* and *cardinal_sum_pr*) says that the cardinal sum or cardinal product of the family $\mathrm{Card}(E_\iota)$ is the cardinal of the sum or the product of the sets E_ι; [64].

Corollary (*cardinal_sum_pr1*).

Proposition 5 (*cardinal_sum_assoc*, *cardinal_prod_assoc*, *cardinal_sum_commutative*, *cardinal_prod_commutative* and *cardinal_distrib_prod_sum*) asserts commutativity, associativity and distributivity of sum and products, [64].

Corollary. Application to the case of 2 or 3 arguments.

Proposition 6 (*zero_unit_sum* and *one_unit_prod*) says that one can remove 0 in a sum and 1 in a product, [68].

Corollary 1 (*zero_unit_sumr*, *zero_unit_suml*, *one_unit_prodr*, *one_unit_prodl*).

Corollary 2 (*sum_of_ones* and *sum_of_same*).

Proposition 7 (*zero_cardinal_product*) says that a cardinal product is non-zero if and only if each factor is non-zero, [69].

Proposition 8 (*succ_injective*) asserts injectivity of the successor function, [69].

Proposition 9 (*cardinal_pow_pr1*) says that a^b remains unchanged if letters are replaced by equipotent sets, [69].

Proposition 10 *cardinal_pow_pr3* says that a^b is a product where all factors are the same [69].

Corollary 1 (*power_of_sum*).

Corollary 2 (*power_of_prod*).

Corollary 3 (*power_of_prod2*).

Proposition 11 (*power_x_0* and variants), states $\mathfrak{a}^0 = 1$, $\mathfrak{a}^1 = \mathfrak{a}$, $1^{\mathfrak{a}} = 1$, and $0^{\mathfrak{a}} = 0$, [70].

Proposition 12 (*cardinal_powerset*) says $\mathrm{Card}(\mathfrak{P}(X)) = 2^X$, [71].

Proposition 13 (*cardinal_le_when_complement*) states that «$\mathfrak{a} \geq \mathfrak{b}$ if and only if there exists a cardinal $\mathfrak{c}$ such that $\mathfrak{a} = \mathfrak{b} + \mathfrak{c}$», [71].

Proposition 14 (*sum_increasing* and *product_increasing*) says the sum and product are increasing functions, [71].

Corollary 1 (*sum_increasing1*, *product_increasing1*).

Corollary 2 (*power_increasing1*).

Theorem 2 (*cantor*) says $X < 2^X$, [72].

Corollary (*cantor_bis*).

Theorems of Chapter 4

Proposition 1 (*is_finite_succ*) says that «a cardinal a is finite if and only if $a+1$ is finite», [75].

Proposition 2 (*le_int_is_int*, *exists_prec*) says that (if n is an integer), if $a \leq n$ then a is an integer, if $n > 0$ there is a unique m with $m+1 = n$ and $a < m+1$ is equivalent to $a < n$, [76].

Corollary 1 (*sub_finite_set*).

Corollary 2 (*strict_sub_smaller*).

Corollary 3 (*finite_image*).

Corollary 4 (*bijective_if_same_finite_c_inj*, *bijective_if_same_finite_c_surj*).

Criterion C61 (*cardinal_c_induction* and variants) (principle of induction), [79]

Proposition 3 (*finite_subset_directed_bounded*, *finite_subset_lattice_inf*, *finite_subset_lattice_sup*, *finite_subset_torder_greatest*, *finite_subset_torder_least*) gives some properties of a finite subset of an ordered set [82].

Corollary 1 (*finite_set_torder_greatest*, *finite_set_torder_worder*)

Corollary 2 (*finite_set_maximal*).

Theorem 1 (*maximal_inclusion*) says that every nonempty set which is of finite character has a maximal element, [106].

Theorems of Chapter 5

Proposition 1 (*finite_sum_finite* and *finite_product_finite*) says that a finite sum or product of integers is an integer, [97].

Corollary 1 (*finite_union_finite*).

Corollary 2 (*finite_product_finite_set*).

Corollary 3 (*Bnat_stable_pow*).

Corollary 4 (*finite_powerset*).

Proposition 2 (*cardinal_lt_pr*) says that $a < b$ if and only if there is c such that $0 < c$ and $b = c + a$; [97].

Proposition 3 (*finite_sum_lt* and *finite_product_lt*) says that $\sum a_i < \sum b_i$ and $\prod a_i < \prod b_i$ if $a_i \leq b_i$ for each i and $a_j < b_j$ for some j, [98].

Corollary 1 (*finite_power_lt1*).

Corollary 2 (*finite_power_lt2*).

Corollary 3 (*plus_simplifiable_left* and variants).

Corollary 4 (*card_sub_pr* and others).

Proposition 4 (*restr_plus_interval_isomorphism*) says that $x \mapsto x + a$ is a bijection (order isomorphism) $[0, b] \to [a, a + b]$, [102].

Proposition 5 (*cardinal_interval*) gives the cardinal of $[a, b]$, [103].

Proposition 6 (*finite_ordered_interval*) asserts that every finite totally ordered set is isomorphic to a unique interval $[1, n]$, [103].

Proposition 7 (*char_fun_union* and others) states properties of the characteristic function of a set, [106].

Theorem 1 (*division_unique*, *division_exists*) asserts existence and uniqueness of Euclidean division, [106].

Proposition 8 is expansion to base b [109].

Proposition 9 (*shepherd_principle*) says that if f is a function from a set with cardinal $\mathfrak{a}$ onto a set with cardinal $\mathfrak{b}$, and if all set $f^{-1}\langle\{x\}\rangle$ have the same cardinal $\mathfrak{c}$, then $\mathfrak{a} = \mathfrak{b}\mathfrak{c}$, [113].

Proposition 10 (*number_of_injections_prop*) gives the number of injections from a finite set into another one, [114].

Corollary (*number_of_permutations*).

Proposition 11 (*number_of_partitions*) gives the number of partitions with p_i elements, [115].

Corollary 1 (*binomial7*).

Corollary 2 (*cardinal_set_of_increasing_functions*).

Proposition 12 (*sum_of_binomial*) $\sum_p \binom{n}{p} = 2^n$ [118].

Proposition 13 is the binomial formula (is a definition in Coq) [118].

Proposition 14 (*cardinal_pairs_lt* and *cardinal_pairs_le*) counts the number of pairs (i, j) such $1 \leq i \leq j \leq n$ or $1 \leq i < j \leq n$, [126].

Corollary (*sum_of_i*).

Proposition 15 counts the number of monomials, [127].

Theorems of Chapter 6

Theorem 1 «The relation 'x is an integer' is collectivizing » (see *inc_Bnat*), [75].

Criterion C62 (restatement on C61, not shown in Coq).

Criterion C63 (*integer_induction*), [134].

Lemma 1 (*infinite_greater_countable*) «Every infinite set, E contains a set equipotent to **N**».

Lemma 2 (*equipotent_N2_N*) «The set **N** × **N** is equipotent to **N**».

Theorem 2 (*equipotent_inf2_inf*) «for every infinite cardinal $\mathfrak{a}$, we have $\mathfrak{a} = \mathfrak{a}^2$» [136]

Corollary 1 (*power_of_infinite*).

Corollary 2 (*finite_family_product*).

Corollary 3 (*notbig_family_sum1*).

Corollary 4 (*sum2_infinite, product2_infinite*).

Proposition 1 (*countable_subset, countable_product countable_union*) states properties of countable sets [138].

Proposition 2 (*countable_finite_or_N*) «Every countable infinite set E is equipotent to **N**», [138].

Proposition 3 (*infinite_partition*) says that every infinite set E has a partition X_i where X_i is equipotent to E and the index set to **N**, [138].

Proposition 4 (*countable_inv_image*) says that if f is a function from E onto F, such that F is infinite and $f^{-1}\langle\{x\}\rangle$ is countable for any $x \in$ F, then F is equipotent to E, [138].

Proposition 5 (*infinite_finite_subsets*) says that the set of finite subsets of an infinite set E is equipotent to E, [138].

Proposition 6 (*increasing_stationary*) characterizes stationary sequences, [140].

Corollary 1 (*decreasing_stationary*).

Corollary 2 (*finite_increasing_stationary*).

Proposition 7 (*noetherian_induction*) (Principle of Noetherian induction), [140].

Symbols

$x \wedge y$ is often replaced by "and". The Coq equivalent is `/\`.

$x \vee y$ is often replaced by "or". The Coq equivalent is `\/`.

$\neg x$ is often replaced by "not". The Coq equivalent is `~`.

$(a|b)c$ is a Bourbaki notation, meaning the relation obtained by replacing b by a in c.

R$\{x\}$ is a Bourbaki notation, meaning that R is a relation that may depend on x. If R is a relation that depends on y, it is also $(x|y)$R.

τ_x(R) is a Bourbaki notation, it is the generic element satisfying R$\{x\}$.

$x \implies y$ is represented in Coq by `x -> y`.

$x \to y$ is a Coq notation meaning the type of functions from type a to type b.

$x = y$ is equality. We use it as synonym to $\iff$.

x : y is a Coq notation meaning that x is of type y.

$f(x)$ is the value of the function f at point x, parentheses are sometimes omitted.

$f\langle x\rangle$ is the value of f on the set x, see *fun_image, image_by_graph, image_by_fun.*

$f^{-1}\langle x\rangle$, see *inverse_image.*

$(\forall x)$P and *forall x, p* are similar constructions.

$(\exists x)$P and *exists x, p* are similar constructions.

$(\exists! x)$P means sometimes *exists_unique.*

$x \in y$, $x \ni y$ (is element of): see *inc* and *elt.*

$x \subset y$ (is subset of): see *sub.*

$\emptyset$ (empty set): see *emptyset.*

$\{x, \text{R}\}$ (set of x such that R): see *Zo.*

$\{x\}$, $\{x,y\}$: see *singleton* or *doubleton*.

$a-b$, $a\setminus b$, $\complement a$: see *complement*.

(x,y) (ordered pair): see *J*.

$\bigcup \mathrm{X}$, $\bigcup_{\iota\in \mathrm{I}} \mathrm{X}_\iota$, see *union*.

$a\cup b$, $a\cap b$, see *union2*, *intersection2*.

$\mathrm{A}\times\mathrm{B}$, $u\times v$, $\mathrm{R}\times\mathrm{R}'$, see *product*, *ext_to_prod*, *prod_of_relation*.

$f\circ g$, see *fcompose*, *gcompose*, *compose_graph*, *compose*, *composeC*.

Δ_{A}, see *diagonal*.

G^{-1} see *inverse_graph* , *inverse_fun* or *inverseC*.

$x\mapsto y$ or $x\to y$ is the function that maps x to y, for instance $x\mapsto \sin x$ (source and target are implicit).

$\boldsymbol{x}\to\boldsymbol{T}$ ($\boldsymbol{x}\in\boldsymbol{A}$, $\boldsymbol{T}\in\boldsymbol{C}$), is the function with source A, target C that maps x to T.

$(f_x)_{x\in\mathrm{A}}$ is a shorthand for $x\to f(x)$ ($x\in\mathrm{A}$); see above, the piece $\boldsymbol{T}\in\boldsymbol{C}$ is implicit.

$\hat{f}$, see *extension_to_parts*.

F^{E}, set of graphs of functions from E to F, see *set_of_gfunctions*.

$\mathscr{F}(\mathrm{E};\mathrm{F})$, set of functions from E to F, see see *set_of_functions*.

$\Phi(\mathrm{E},\mathrm{F})$, set of functions from a subset of E to F, see *set_of_sub_functions*.

f_x, f_y sometimes denotes the mappings $y\mapsto f((x,y))$ or $x\mapsto f((x,y))$, implemented as *first_partial_fun*, *second_partial_fun*.

$\tilde{f}$, sometimes denotes the mappings $x\mapsto f_x$ or $y\mapsto f_y$. Implemented as *first_partial_function*, *second_partial_function*.

$f\mapsto\tilde{f}$, implemented as *first_partial_map*, *second_partial_map*, is a bijection from $\mathscr{F}(\mathrm{B}\times\mathrm{C};\mathrm{A})$ into $\mathscr{F}(\mathrm{B};\mathscr{F}(\mathrm{C};\mathrm{A})$ or $\mathscr{F}(\mathrm{C};\mathscr{F}(\mathrm{B};\mathrm{A}))$.

$\prod_{\iota\in\mathrm{I}}\mathrm{X}_\iota$, product of a family of sets, see *producttt*.

$(x_\iota)_{\iota\in\mathrm{I}}$ denotes an element of a product indexed by I.

$x\overset{r}{\sim}y$ is sometimes used instead of $r(x,y)$, especially when r is the graph of an equivalence relation.

$g_{\mathrm{E}}(\sim)$, the graph of $\sim$ on E, see *graph_on*.

$\sim_f$ may denote *eq_rel_associated f*.

$\bar{x}$, may denote the equivalence class of x, see *class*.

$\hat{x}$ may denote a representative of the equivalence class x.

$\mathrm{E}/\sim$, E/R (quotient set of E) see *quotient*.

R/S (quotient of two equivalence relations) see *quotient_of_relations*.

X_f sometimes means $f^{-1}\langle f\langle\mathrm{X}\rangle\rangle$, see *inverse_direct_value*.

R_{A} see *induced_relation*.

$x\succ_r y$, $y\prec_r x$, notations used when x and y are related by a preorder relation,[7].

$x\leq_r y$, $y\geq_r x$, notations used when x and y are related by an order relation.

$x\leq y$, $y\geq x$, $x<y$, $x>y$: notations used when x and y are related by an order relation, see *gle*, *gge*, *glt*, *ggt*.

$f\mapsto \mathrm{G}_f$ see *graph_of_function*.

$\omega\mapsto\tilde{\omega}$ see *graph_of_partition*.

$\sup(x,y)$, $\sup_{\mathrm{E}}\mathrm{X}$, $\sup\mathrm{X}$, $\sup_{x\in\mathrm{A}} f(x)$ see *supremum*, *sup*, *sup_graph*.

$\inf(x,y)$, $\inf_{\mathrm{E}}\mathrm{X}$, $\inf\mathrm{X}$, $\inf_{x\in\mathrm{A}} f(x)$ see *infimum*, *inf*, *sup_graph*.

$[a,b]$, $[a,b[$, $]a,b]$, $]a,b[$, $[a,\rightarrow[$, $]a,\rightarrow[$, $]\leftarrow,b]$, $]\leftarrow,b]$, $]\leftarrow,\rightarrow[$, see interval.

$\mathfrak{r} \leq_{\mathrm{Card}} \mathfrak{n}$, order on cardinals, see *cardinal_le*.

$g^{(x)}$ is the restriction of g to $]\leftarrow,x[$ see *restriction_to_segment*

$\mathrm{Card}(x)$ is the cardinal of x, see *cardinal*.

0, 1, 2, 3, 4: see *card_zero* or *card_three*.

$\sum_{\iota\in I}\mathfrak{a}_\iota$, $\prod_{\iota\in I}\mathfrak{a}_\iota$: cardinal sum or cardinal product of a family of cardinals, see *cardinal_sum* and *cardinal_prod*.

$\mathfrak{a}+\mathfrak{b}$, $\mathfrak{a}.\mathfrak{b}$, $\mathfrak{a}\mathfrak{b}$, is the cardinal sum or cardinal product of two cardinals, see *card_plus card_mult*

$\mathrm{E}_1+\mathrm{E}_2$ denotes also the ordinal sum, see *ordinal_sum*.

$\mathrm{X}_{xy}(a,b)$ is the family $x\mapsto a$ and $y\mapsto b$, see page 59.

$\mathrm{X}(a,b)$ is $\mathrm{X}_{\alpha,\beta}(a,b)$, for some fixed α and β.

$a_x \cup b_y$ is the disjoint union of $\mathrm{X}_{xy}(a,b)$, i.e., $a\times\{x\}\cup b\times\{y\}$.

$\mathfrak{a}^{\mathfrak{b}}$ is the cardinal power of two cardinals, see *card_pow*.

a^b is the power of two integers, see *pow*.

$\mathbb{N}$, **N**, set of integers, see *Bnat*.

$a-b$ is the difference in **N**, see *card_sub*: can also means *minus*, the difference in $\mathbb{N}$.

$[a,b]$ is an interval on **N**, [101]

$\sum_{i=a}^{b} t_i$ is $\sum_{i\in[a,b]} t_i$.

a::b is *cons a b*, it is the list obtained by putting the element a in front of the list b.

ϕ_{A} is the characteristic function on E.

a/b is the quotient of a and b.

$n!$ is the factorial of n.

$\binom{n}{p}$, see binom.

Letters

$\mathscr{B}$ see *Bo*.

$\mathscr{C}_{\mathrm{C}}(a,b)$, $\mathscr{C}_{\mathrm{T}}(p,q)$, $\mathscr{C}(p)$: see *by_cases a b*, *chooseT* and *choose*.

$\mathrm{C}_{xy}a$ stands for *constant_function x y a*, it is the constant function from x to y with value a.

$\mathrm{C}_{\mathrm{R}}x$ may denote the equivalence class of x for R, see *class*.

$\mathrm{Coll}_{\boldsymbol{x}}\boldsymbol{R}$ says that R is collectivizing in x.

$\mathscr{E}$, see *Set*.

$\mathscr{E}_x(\mathrm{R})$ appears in the English version where $\{x,\mathrm{R}\}$ is used in the French version; see *Zo*.

I_{A}, see *identity*.

I_{xy} see *inclusionC*, *canonical_injection*.

$\mathscr{I}(\mathrm{E},\mathrm{T},f)$ says that f is defined by transfinite induction, see *transfinite_def*.

$\mathscr{L}_{\mathrm{X}}f$, $\mathscr{L}f$, $\mathscr{L}_{\mathrm{A;B}}f$ (creating functions): see *L*, *acreate*, *BL*.

$\mathscr{M}f$, $\mathscr{M}_{\mathrm{A;B}}f$ (inverse of $\mathscr{L}$) see *bcreate1* and *bcreate*.

$\mathbb{N}$, **N**, set of integers, see *Bnat*.

$\mathscr{N}$, is the bijection from $\mathbb{N}$ onto **N**, see *nat_to_B*.

$\mathfrak{P}(x)$, see *powerset*.

$\mathrm{pr}_1 z$, $\mathrm{pr}_2 z$, $\mathrm{pr}_\mathrm{I} f$, $\mathrm{pr}_\mathrm{J} f$ (projections), see P, Q, *pr_i*, *pr_j*.

$\mathscr{R} x$ see *Ro*.

$\mathrm{R}_{ab} f$ (restriction) see restriction2.

$\mathscr{V}(x, f)$, $\mathscr{V}_f x$ (value of a function): see *V*.

$\mathscr{W}_f x$ (value of a function): see *W*.

$\mathscr{X}(f, y)$, see *Xo*.

$\mathscr{Y}(\mathrm{P}, x, y)$ see *Yo*.

$\mathscr{Z}(x, \mathrm{P})$ see *Zo*.

¶ is not defined. We use it as a paragraph separator.

Words

acreate f, $\mathscr{L} f$, is the correspondence associated to the Coq function f.

agrees_on x ff', *agreeC x ff'* is the property that for all $a \in x$, $f(a)$ and $f'(a)$ are defined and equal.

antisymmetric_r r says that the relation r is antisymmetric, [7].

axioms_product_order f g is the condition under which *product_order f g* is an order, [17].

back_to_nat f n returns the value of f (defined on a subset of **N**) as is if f were defined on $\mathbb{N}$, [419].

bcreate f A B, $\mathscr{M}_{\mathrm{A;B}} f$, is a kind of inverse of $\mathscr{L}$.

bcreate1 f, $\mathscr{M} f$, is a kind of inverse of $\mathscr{L}$.

bijective f, *bijectiveC f*, means that f is a bijection.

binom n p, $\binom{n}{p}$, is the binomial coefficient, [118].

BL f a b, $\mathscr{L}_{\mathrm{A;B}} f$, *fun_function f a b*, is function from A to B whose graph is $\mathscr{L}_\mathrm{A} f$.

Bnat or **N** is the set of all integers, [75].

Bnat_divides b a says that $a = bq$ for some q, [107].

Bnat_order, *Bnat_le x y*, *Bnat_lt x y* is the ordering on **N**, and the relations $x \leq y$ or $x < y$ on **N**, [79].

Set or $\mathscr{E}$ is the type of sets.[1]

Bo, $\mathscr{B}$, is an inverse of $\mathscr{R}$.

bounded_above r X, *bounded_below r X*, *bounded_both r X*, mean that X is bounded for r (from above, below or both), [27].

by_cases a b, $\mathscr{C}_\mathrm{C}(a, b)$, defines an object by applying a if P is true, and b if P is false.

canonical_injection x y, I_{xy}, is the inclusion map on $x \subset y$.

canon_proj r, is the mapping $x \mapsto \bar{x}$ from E onto E/R, where E/R is the quotient set of r.

card_mult a b, $\mathfrak{a}.\mathfrak{b}$ or $\mathfrak{ab}$, is the cardinal product of a family of two elements, [66].

card_plus a b, $\mathfrak{a} + \mathfrak{b}$, is the cardinal sum of a family of two elements, [66].

card_quo a b and *card_rem a b* are the quotient and remainder in the division of a by b [107].

card_pow a b, $\mathfrak{a}^\mathfrak{b}$, is the cardinal is the set of functions (or graphs of functions) from y into x. [69].

card_sub a b, $a - b$ is the difference of the two cardinals. [99].

[1] Changed to Set in version 2

card_three, *card_four*, or 3 and 4, are the cardinals $2+1$ and $3+1$, [75]

card_zero, *card_one*,*card_two*, or 0, 1, 2, are the cardinals of the empty set, a singleton, or a doubleton with two distinct elements [59].

cardinal x, Card(x) is some set equipotent to x, [59].

cardinal_le x y, $x \leq_{\mathrm{Card}} y$, says that x and y are two cardinals such that x is equipotent to a subset of y, [60].

cardinal_lt x y is $x \leq_{\mathrm{Card}} y$ and $x \neq y$.

cardinal_prod x, $\prod_{\iota\in I} \mathfrak{a}_\iota$, is the cardinal of the product of the family of sets, [64].

cardinal_nat x maps every set equipotent to the n-th ordinal to the natural number n, [409].

cardinal_sum x, $\sum_{\iota\in I} \mathfrak{a}_\iota$, is the cardinal of the disjoint of the family of sets, [64].

class r x is the class of x for the equivalence relation r.

char_fun A B is the characteristic function of A, as a mapping from B into $\{0,1\}$, [105].

choose p, $\mathscr{C}(p)$, is some x such that $p(x)$ is true, the empty set if no x satisfies p.

choosef p is some function f such that $p(f)$ is true, the identity on the empty set if no f satisfies p.

choosenat p, $\mathscr{C}_{\mathbb{N}}(p)$, is some integer i such that $p(i)$ is true, zero if no i satisfies p, [409].

chooseT p q, $\mathscr{C}_{\mathrm{T}}(p,q)$, is our basic axiom of choice.

coarse x is $x \times x$.

coarser x is the order associated to *coarser_c*, [11].

coarser_c f g, *coarser_covering I f J g*, two definitions that say for all $j \in$ J there is $i \in$ I such that $g_j \subset f_i$ or for all $g_j \in g$ there is $f_i \in f$ such that $g_j \subset f_i$.

coarser_preorder is the order induced by $\subset$ on preorders, [17].

cofinal_set r A says that x in the substrate of r there is an $y \in$ A such that $x \leq_r y$, [26].

coinitial_set r A says that x in the substrate of r there is an $y \in$ A such that $x \geq_r y$, [26].

common_prolongation_order_axiom g are the conditions on g for which an order can be put on the union, [45].

common_prolongation_order g h says that h is an order that on the union of the substrate of the g_i that coincides with the restriction, [45].

common_worder_axiom g are the conditions on g for which an well-ordering can be put on the union of the family [46].

compatible_with_equiv_p p r means that $p(x)$ and $x \overset{r}{\sim} y$ implies $p(y)$.

compatible_with_equiv f r means that $x \overset{r}{\sim} y$ is equivalent to $f(x) = f(y)$.

compatible_with_equivs f r r' means that $x \overset{r}{\sim} y$ is equivalent to $f(x) \overset{r'}{\sim} f(y)$.

complement a b, $a - b$, $a \setminus b$, $\complement b$, is the set of element of a not in b.

composableC f g, *composable f g* is the condition on correspondences (resp. functions) f and g for $f \circ g$ to be a correspondence (resp. function).

compose_graph f g, $f \circ g$, composition of two graphs.

compose f g, *composeC f g*, $f \circ g$, is the composition of two functions.

constant_graph s x is the graph of the constant function with domain s and value x.

contraction A B L f v: assume $f : \mathrm{A} \times \mathrm{B} \to \mathrm{B}$ is a function, v an object of type B. If L is a list of type A C(L) is defined by $\mathrm{C}(a :: b) = f(a, \mathrm{C}(b))$ and $\mathrm{C}(nil) = v$,[420].

correspondenceC is a data type with three slots, source, target and graph.

corr_value f associates to a correspondence f its triple (G, A, B).

covering f x, *covering_f I f x*, *covering_s f x*, three variants of a family of sets (defined by f and I) whose union contains x.

cut x p is the set of all x that satisfy p.

cut r x is $r\langle x\rangle$.

decent_set x says that no element y of x satisfies $y \in y$, [83].

decreasing_map f r r', *decreasing_fun f r r'* is a function such that $x \leq_r y$ implies $f(x) \geq_{r'} f(y)$, [20].

decreasing_sequence f r says that f is a decreasing function with source **N** and target r, [139].

diagonal A, Δ_A, is the set of all (x, x) such that $x \in$ A.

diagonal_application A is the diagonal mapping $x \mapsto (x, x)$ of A into Δ_A.

diagonal_graphp I E is the set of graphs of constant functions from I to E.

disjoint x y means $x \cap y = \emptyset$.

disjoint_union f, *disjoint_union_fam f* are two variants of the disjoint union of the family of sets f.

domain f is the set of x for which there is an y with $(x, y) \in f$, it is $\mathrm{pr}_1\langle f\rangle$.

doubleton x y, $\{x, y\}$, is a set with elements x and y.

double_list_prop A L Q says that all elements x and y of the list L of type A satisfy the predicate Q(x, y), whenever x comes before y, [419].

EEE is a shorthand for the type *Set* → *Set* → *Set*.

EEP is a shorthand for the type *Set* → *Set* → *Prop*.

elt x y, $x \ni y$, is the same as $y \in x$.

empty_function, *empty_functionC* is the identity on $\emptyset$.

emptyset, $\emptyset$, is a set without elements.

eq_rel_associated f is the graph of the equivalence relation $f(x) = f(y)$.

equipotent x y means that there is a bijection from x into y.

equipotent_to_subset x y means that x is equipotent to s subset of x, [60].

equivalence_associated f is the equivalence relation $f(x) = f(y)$.

equivalence_associated_o r is the equivalence relation $x \prec_r y$ and $y \prec_r x$, [13]

equivalence_r r, *equivalence_re r x*, says that the relation r is an equivalence relation (in x).

equivalence_corr r says that the correspondence r is associated ton an equivalence.

exists_unique p,$(\exists!x)p$, (this notation is not in Bourbaki) means that there exists a unique x such that $p(x)$.

expansion_value f b assume that f is a functional graph, defined for $i < k$ and such that $f_i < b$; the value is $\sum f_i b^i$ [110].

extends g f, *extendsC g f* says $g(x) = f(x)$ whenever $f(x)$ is defined.

extends_in E F is the relation *extends* in Φ(E, F), [10].

extension_order E F is the order associated to *extends_in E F*, [10].

ext_map_prod I X Y g is the function $(x_\iota)_{\iota\in I} \mapsto (g_\iota(x_\iota))_{\iota\in I}$ from $\prod_I X_\iota$ into $\prod_I Y_\iota$.

ext_to_prod u v is the function $(x, y) \mapsto (u(x), v(y))$, sometimes denoted $u \times v$.

extension_to_parts f, denotes the function $x \mapsto f\langle x\rangle$, from $\mathfrak{P}$(A) into $\mathfrak{P}$(B).

factorial n, $n!$, is the factorial function, [113].

fct_to_list A f n is the list of type A containing $f(i)$ for $i < n$, [417]

fct_sum f n, *fct_prod f n*, is the sum or product of the values $f(k)$ for $k < n$, computed via *fct_to_list*, [423].

finer_equivalence s r, comparison of equivalences, $x \overset{s}{\sim} y$ implies $x \overset{r}{\sim} y$.

finite_int_fam f says that f is a functional graph, with a finite domain and whose range is a subset of the set of integers, [96].

first_proj g is the function $x \mapsto \mathrm{pr}_1 x$ $(x \in g)$.

first_proj_equiv x y, *first_proj_equivalence x y*, is the equivalence associated to *first_proj* on the set $x \times y$.

fcompose f g, $f \circ g$, composition of two graphs, without assumption.

fcomposable f g says that graphs g and $f \circ g$ have the same domain.

fgraph f says that f is a functional graph.

functional_graph f says that f is a functional graph.

function_order E F G, *function_order_r E F G*, is the order defined on $\mathscr{F}(\mathrm{E};\mathrm{F})$ by $\forall x, f(x) \prec_{\mathrm{G}} g(x)$, [18].

function_prop f s t, *function_prop_sub f s t*. This is the property that f is a function from s into t, or into a subset of t.

fun_image x f, $f\langle x\rangle$, is the value of f on the set x.

fun_on_quotient r f, *function_on_quotient r f b*, *function_on_quotients*, *fun_on_quotients r r' f*, the function obtained from f on passing to the quotient of r (or r and r').

fun_set_to_prod E X is the canonical bijection between $(\prod \mathrm{X}_\iota)^{\mathrm{E}}$ and $\prod \mathrm{X}_\iota^{\mathrm{E}}$.

gcompose f g, $f \circ g$, composition of two graphs, assumes that range g is a subset of domain f.

gge r x y, $x \geq y$, says that $y \leq x$, [8]

ggt r x y, $x > y$, says that $x \geq y$ and $x \neq y$, [8]

gle r x y, $x \leq y$, says that x and y are related by r, [8]

glt r x y, $x < y$, says $x \leq y$ and $x \neq y$, [8]

graph f is a part of a correspondence.

graph_of_function X Y is the function $f \mapsto \mathrm{G}_f$ defined on $\Phi(\mathrm{E},\mathrm{F})$, where G_f is the graph of f, [16].

graph_of_partition, is the function $\omega \mapsto \tilde{\omega}$, see *partition_relation_set*, [16]

graph_on r X is the graph of the relation r restricted to X.

graph_order E F G, *graph_order_r E F G* is the order defined on F^{E} by $\forall x, f(x) \prec_{\mathrm{G}} g(x)$, [18].

greatest_element r a is the property that a is the greatest (unique maximal) element of the substrate of the order r, [23].

greatest_lower_bound r X a is the property that a is the greatest element of the set of lower bounds of X, [27].

has_infimum r X says that X has a infimum, [28].

has_inf_graph r f says that the image of the graph f has an infimum, [30].

has_supremum r X says that X has an supremum, [28].

has_sup_graph r f says that the image of the graph f has a supremum, [30].

identity A, I_{A}, is is the graph of the identity function on the set A.

identity_fun A, I_{A}, is the identity function on the set A.

IM stands for the image of a function. Its axioms implement the Scheme of Selection and Union.

image_by_fun f A, $f\langle \mathrm{A}\rangle$, is $\{t, \exists x \in \mathrm{A}, t = f(x)\}$.

image_by_graph f x, $f\langle \mathrm{A}\rangle$, is $\{t, \exists x \in \mathrm{A}, (x,t) \in f\}$.

image_of_fun f, is the image of f.

inc x y or $x \in y$ means that x is an element of y.

inclusionC x y, I_{xy}, it is the inclusion map on $x \subset y$ as a Coq function.

inclusion_order A, is the order induced by $\subset$ on $\mathfrak{P}(\mathrm{A})$ [9].

inclusion_suborder A, is the order induced by $\subset$ on A [9].

increasing_map f r r', *increasing_fun f r r'* is a function such that $x \leq_r y$ implies $f(x) \leq_{r'} f(y)$, [20].

increasing_pre f r r' says that f is an increasing function for preorders r and r', [184].

increasing_sequence f r says that f is an increasing function with source **N** and target r, [139].

induced_relation R A, R_A, is the equivalence induced by R on A.

induced_order R A, R_A, is the order induced by R on A [9].

induction_defined s a is the function f defined by $f(0) = a$ and $f(n+1) = s(f(n))$, [134].

induction_defined1 E h a is the function f defined by $f(0) = a$ and $f(n+1) = h(n, f(n))$, [134].

induction_defined2 E h a p is the function f defined for $n < p$ by $f(0) = a$ and $f(n+1) = h(n, f(n))$, [134].

inductive_set r means that r is an order whose substrate is inductive, [51].

inf r x y, $\inf(x,y)$, is the greatest lower bound of pair $\{x,y\}$ (if it exists), [28].

infimum r X, $\inf_\mathrm{E} \mathrm{X}$, is the greatest lower bound of X (if it exists), [28].

infinite_set x means that x is a not a finite set, [75].

inf_graph r f, $\inf\limits_{x\in\mathrm{A}} f(x)$, is the greatest lower bound of the image of the graph f (if it exists), [30].

injective f, *injectiveC f*, means that f is an injection.

in_same_coset f is the relation "there exists i such that $x \in f(i)$ and $y \in f(i)$" between x and y.

intersection X, $\bigcap \mathrm{X}$, is the intersection of a set of sets.

intersectiont I f, *intersectionf x f*, *intersectiont g*, $\bigcap\limits_{\iota\in\mathrm{I}} \mathrm{X}_\iota$ is the set of elements a such that forall $\iota \in \mathrm{I}$ we have $a \in \mathrm{X}_\iota$.

intersection2 X Y, $\mathrm{X} \cap \mathrm{Y}$, is the intersection of two sets.

intersection_covering, intersection of coverings, .

interval_oo r a b, *interval_oc r a b*, *interval_ou r a*, *interval_co r a b*, *interval_cc r a b*, *interval_cu r a interval_uo r b*n, *interval_uc r b interval_uu r*; Intervals, [37].

interval_Bnat a b, *interval_co_0a c* is the interval $[a,b]$ or $[0,c[$ as a subset of **N**.[101]

interval_Bnato a b, *interval_Bnatco a* is the interval $[a,b]$ or $[0,c[$ as an ordered set.[102]

inverse_direct_value f X, X_f, is $f^{-1}\langle f\langle \mathrm{X}\rangle\rangle$.

inverse_graph G, $\overset{-1}{\mathrm{G}}$, inverse graph of the graph G.

inverse_fun f or *inverseC a b f H*, $\overset{-1}{f}$, inverse of the function f.

inverse_image x f, $\overset{-1}{f}\langle x\rangle$, is the inverse value of f on the set x.

inv_image_relation f r, is the inverse image of the relation r under the function f.

inv_image_by_graph f x, *inv_image_by_fun r x*, $\overset{-1}{f}\langle x\rangle$, direct image of a set by the inverse function

inv_corr_value t associates to a $t = (\mathrm{G}, \mathrm{A}, \mathrm{B})$ its correspondence f.

inv_graph_canon G is the bijection $(x, y) \mapsto (y, x)$ from G to G^{-1}.

is_antisymmetric r says that the graph r is antisymmetric, [7].

is_bounded_interval r x, see interval.

is_cardinal x says that x is of the form Card(x), [60].

is_class r x says that x is an equivalence class for r.

is_closed_interval r x, see interval.

is_countable_set x says that x is equipotent to **N**, [138].

is_correspondence f says that f is associated to a triple (G, A, B).

is_equivalence r says that the graph r is an equivalence.

is_expansion f b k say that f is a functional graph, defined for $i < k$ and such that $f_i < b$, [110].

is_finite_c x, *is_finite_set y*: a cardinal x is finite if $x \neq x+1$, a set is finite if its cardinal is finite, [75].

is_function f says that f is a function in the sense of Bourbaki.

is_graph f says that f is a set of pairs.

is_graph_of g r is true if g is the graph of the relation r.

is_infinite_c x means that x is a cardinal that is not finite, [75].

is_inf_fun r f x, *is_inf_graph r f x*, says that x is the supremum of the image of the function of graph f for the order r, [30].

is_interval r x, see interval.

is_left_inverse r f means that r is a retraction or left-inverse of f, and $r \circ f$ is the identity.

is_left_unbounded_interval r x, see interval.

is_open_interval r x, see interval.

is_reflexive r says that the graph r is reflexive.

is_restriction f g says that f is the restriction of g to some set.

is_right_inverse s f means that s is a section or right-inverse of f, and $f \circ s$ is the identity.

is_right_unbounded_interval r x, see interval.

is_segment r s, says that s is the interval $]\leftarrow, x[$ or the whole substrate of a well ordered relation r, [43].

is_semi_open_interval r x, see interval.

is_singleton x means that x is a singleton.

is_sup_fun r f x, *is_sup_graph r f x*, says that x is the supremum of the image of the function or graph f for the order r, [30].

is_symmetric r says that the graph r is symmetric.

is_unbounded_interval r x, see interval.

is_transitive r says that the graph r is transitive.

J x y, or (x, y), is an ordered pair, formed of two items x and y.

nil is the empty list.

L X f, *fcreate X f*, $\mathscr{L}_{\mathrm{X}} f$ is the graph formed of all $(x, f(x))$ with $x \in \mathrm{X}$.

largest_partition x is the set of all singletons of x.

lattice r, is a relation for which $\sup(x,y)$ and $\inf(x,y)$ exist, [35].

least_element r a ids the property that a is the least (unique minimal) element of the substrate of the order r, [23].

least_upper_bound r X a is the property that a is the least element of the set of upper bounds of X, [27].

left_directed r means that each doubleton is bounded below, [34].

left_inverseC, left inverse of a Coq function.

lexicographic_order r f g, *lexicographic_order_r r f g*, *lexicographic_order_axioms r f g*: assume that f is a family of sets with index I, g is a family of orders with index I such that the substrate of g_ι is f_ι, and r is a well-ordering on I; these conditions are the axioms; they allow to define an ordering and an order relation on the product of the family f, [54].

LHS is the left hand side of an equality.

list_range L is the smallest set containing all elements of the list, [420]

list_subset L E says that all elements of the list L belong to the set E, [420]

list_sum L, *list_prod L*, is the sum or product of the element of the list L [423].

list_to_fct L, *list_to_f L*, *list_to_fctB L*, *list_to_fB L E*, converts the list L into a mapping $\mathbb{N}\to\mathbb{N}$, or a function with source $[0,n[$ and target **N** or E.[417], [418].

lower_bound r X x says that for all $y\in$ X, we have $x\leq_r y$, [26].

Lvariant a b x y, *variant a x y*, *Lvariantc x y*, these are functions whose range is the doubleton $\{x,y\}$.

maximal_element r a says that $x\leq_r a$ implies $x=a$, [22].

merge_int n m is a bijection $\mathbb{N}\times\mathbb{N}\to\mathbb{N}$, [136].

minimal_element r a says that $x\geq_r a$ implies $x=a$, [22].

monotone_fun f r r' is a, increasing or decreasing function, [20].

mutually_disjoint f says that for all distinct i and j, $f(i)$ and $f(j)$ are disjoint.

nat, $\mathbb{N}$ is the type of natural integers in Coq.

nat_to_B, $\mathcal{N}$ is the canonical bijection from *nat* to the set of finite cardinals, [410].

natR n is maps a natural number onto a pseudo-ordinal, [408]

neq x y, $x\neq y$, is inequality.

Nquo a b and *Nrem a b* are the quotient and remainder in the division of a by b, [415].

number_of_injections b a is $a!/(a-b)!$, [114].

of_finite_character x says that x is of finite character, [83].

one_point is the basic singleton.

opposite_relation r is the relation $r(y,x)$ between x and y, [7].

opposite_order r is inverse graph of r, [7].

order r says that the graph r is an order, [8].

order_associated r is the order associated to a preorder by passing on the quotient of "$x\prec y$ and $y\prec x$", [13]

order_axioms r s is the condition on which r is an order on s [14].

order_c f says that the graph of the function r is an order, [8].

order_isomorphism f r r' says that f is an injective increasing function from the support of the order r into the support of the order r', [15].

order_morphism f r r' says that f is an injective increasing function from the support of the order r into the support of the order r', [15].

order_r r says that the relation r is an order, [7].

order_re r x says that the relation r is an order on x, [8].

order_with_greatest r a is the order obtained from r by adjoining a greatest element a, [25]

ordinal_sum r f g, $\sum_{\iota\in I} X_\iota$, is the ordinal sum of the family of sets $f(i)$, where each set is ordered by $g(i)$ and the index set by r. [142].

ordinal_sum2 r r', $E_1 + E_2$, is ordinal sum of two sets. [143].

P z, $\mathrm{pr}_1 z$ denotes x if z is the pair (x, y).

partial_fun1 f y, *partial_fun1 f x*, partial functions.

partition y x, *partition_s y x*,*partition_fam f x*, thee variants that say that y or f is a partition of x.

partition_fun_of_set Y X is the canonical injection from Y int $\mathfrak{P}(X)$, (if Y is a partition of X then $Y \in \mathfrak{P}(\mathfrak{P}(X))$) [11].

partition_relation f x is the equivalence relation associated to the partition f of x.

partition_relation_set y x, $\tilde{\omega}$, is the graph of the equivalence associated to the partition $y = \omega$ of x, [11].

partition_with_complement X A, is the partition of X formed of A and its complementary set.

partition_with_pi_elements p E f says that the sets $f(i)$ are of cardinal p_i, mutually disjoint and form a covering of E, [115].

pow x y, x^y, is the power function on the type *nat*, [411].

powerset x, $\mathfrak{P}(x)$, is the set of subsets of x.

$\mathrm{pr}_1 z$, $\mathrm{pr}_2 z$ stand for *pr1 z* and *pr2 z*. These are also denoted by P and Q. If z is the pair (x, y), these functions return x and y respectively.

pr_i f i, *pr_it f i*, $\mathrm{pr}_i f$, denotes a component of an element of a product.

pr_j f J, $\mathrm{pr}_J f$, is the function $(x_\iota)_{\iota\in I} \mapsto (x_\iota)_{\iota\in J}$

prec x is the finite cardinal y whose successor is y. It exists if x is a non-zero finite cardinal, [76].

preorder r is a reflexive and transitive graph, [12].

preorder_r is a reflexive and transitive relation, [12].

prod_assoc_map is the function whose bijectivity is the "theorem of associativity of products".

prod_of_function u v, is the function $x \mapsto (u(x), v(x))$.

prod_of_products_canon F F', is the bijection between $\prod F_\iota \times \prod F'_\iota$ and $\prod(F_\iota \times F'_\iota)$.

prod_of_relation R R', $R \times R'$, is the product of two equivalences.

product A B, $A \times B$, is the set of all pairs (a, b) with $a \in A$ and $b \in B$

producttt I X, *product b g* or *productf I f*, $\prod_{\iota\in I} X_\iota$ is the product of a family of sets.

product1 x a is the product of the family defined on the singleton $\{a\}$ via value x.

product1_canon x a is the canonical application from x into *product1 x a*.

product2 x y is the product of the family defined on the doubleton $\{a, b\}$ via value x and y.

product2_canon x y is the canonical application from $x \times y$ into *product2 x y*.

product_compose, auxiliary function used for change of variables in a product.

product_order f g, *product_order_r f g*, is the order on the product $\prod X_i$ induced by Γ_i (where f defines the family X_i and g defined the family Γ_i), [17].

pseudo_ordinal x says that x is a pseudo-ordinal, [83].

Q z, $\mathrm{pr}_2 z$ denotes y if z is the pair (x, y).

quotient R, E/R, is the set of equivalence classes of R

quotient_of_relations r s, R/S, is the quotient of two equivalences

quotient_order_r r s is the preorder relation induced in the quotient E/S by the preorder $\succ_R$, where E is the common substrate of R and S, [184].

range f is the set of y for which there is an x with $(x, y) \in f$, it is $\mathrm{pr}_2\langle f\rangle$.

reflexive_r r x says that the relation r is reflexive in x.

reflexive_rr r says that the relation r is reflexive, [7].

related r x y is a short-hand for $(x, y) \in r$.

relation_on_quotient p r is the relation induced by $p(x)$ on passing to the quotient (with respect to x) with respect to R.

rep x is an element y such that $y \in x$, whenever x is not empty.

representative_system s f x means that, for all i, $s \cap X_i$ is a singleton, where X_i is a partition of x associated to the function f.

representative_system_function g f x, means that g is an injection whose image is a system of representatives (see definition above).

restr x G is the restriction to x of the graph G.

rest_plus_interval a b, *rest_minus_interval a b* are the function $x \mapsto x+b$ and $x \mapsto x-b$ as bijections between $[0, a]$ and $[b, a+b]$, [102]

restricted_eq E is the relation "$x \in$ E and $y \in$ E and $x = y$".

restriction_function f x is like *restr*, but f and the restrictions are functions.

restriction2_axioms f x y is the condition: f is a function whose source contains x, whose target contains y, moreover $a \in x$ implies $f(a) \in y$.

restriction2 f x y, *restriction2C f x y*, restriction of f as a function $x \to y$.

restrictionC f H is the restriction to x of the function $f : a \to b$, where H proves $x \subset a$ implicitly.

restriction_product f j is the product of the restrictions of $\prod f$ to J.

restriction_to_image f is the restriction of the function f to its range.

restriction_to_segment r x g, $g^{(x)}$, is the restriction of g to the segment S_x defined by the order r, [47]

restriction_to_segment_axiom r x g is the property for *restriction_to_segment* to be well-behaved, [47]

retraction: see *is_left_inverse*.

RHS is the right hand side of an equality.

right_directed r means that each doubleton is bounded above, [34].

right_inverseC, right inverse of a Coq function.

Ro x or $\mathscr{R}x$ converts its argument x of type u to a set, which is an element of u.

saturated r x means: for every $y \in x$, the class of x for the relation r is a subset of x.

saturation_of r x is the saturation of x for r.

second_proj g is the function $x \mapsto \mathrm{pr}_2 x$ $(x \in g)$.

section: see *is_right_inverse*.

section_canon_proj R is the function from E/R into E induced by *rep*.

segment r x, S_x, is the interval $]\leftarrow, x[$, [43].

Set or $\mathscr{E}$ is the type of sets.

set_for_equipotent_inf2_inf E psi is a set used when proving that E is qequipoptent to $E \times E$ when E is infinite, [136].

set_of_correspondences A B means the set of triples

set_of_cardinals_le x is the set of all cardinals $\leq x$, [63].

set_of_endomorphisms E, is the set of triples (G, E, E) associated to functions from E into E.

set_of_finite_subsets x is the set of finite subset sof x, [81].

set_of_functions E F, denoted $\mathscr{F}(E;F)$, is the set of triples (G, E, F) associated to functions from E into F.

set_of_functions_sum_le E n, set_of_functions_sum_eq E n is the set of functions $f : E \to [0, n]$ such that the sum $\sum f(i)$ is $\leq n$ or $= n$, [127].

set_of_gfunctions E F, denoted F^E, is the set of graphs of functions from E to F

set_of_injections E F is the set of injective functions from E into F, [114].

set_of_majorants1 is used in an example.

set_of_partition p E is the set of all partitions X_i of E, where each X_i has p_i elements, [115].

set_of_partition_set X is the set of all partitions of X, [11].

set_of_permutations E F, is the set of injective functions from E onto itself, [114].

set_of_segments r, set_of_segments_strict r, is the set of all segments of an ordered set (with possible exclusion of the whole set), [45].

set_of_sub_functions E F, denoted $\Phi(E;F)$ is the set of triples (G, A, F) associated to functions from $A \subset E$ into F.

set_of_graphs E F, is the set of functional graphs from E to F, [16].

set_of_preorders E, is the set of preorders on E, [17].

singleton x, $\{x\}$, is a set with one element.

single_list_prop A L Q says that all elements of the list L of type A satisfy the predicate Q, [419].

sof_value x y z converts three elements into a correspondence,.

small_set x means that x hast at most one element.

smallest_partition x is the singleton $\{x\}$.

source f contains (resp. is equal to) the domain of the graph of a correspondence f (resp. function f).

stationary_sequence f says that the restriction of f to some interval $[n, \rightarrow [$ is constant, [139].

strict_decreasing_map f r r', strict_decreasing_fun f r r' is a function such that $x <_r y$ implies $f(x) >_{r'} f(y)$, [20].

strict_increasing_map f r r', strict_increasing_fun f r r' is a function such that $x <_r y$ implies $f(x) <_{r'} f(y)$, [20].

strict_monotone_fun f r r' is a strictly increasing or strictly decreasing function, [20].

strict_sub x y, $x \subsetneq y$, means $x \subset y$ and $x \neq y$.

sub x y, $x \subset y$, means that x is a subset of y.

substrate r is the union of the domain and range.

subsets_with_p_elements p E, is the set of subsets of E having p elements, [120].

succ x is $x+1$, [75].

sup r x y, $\sup(x,y)$, is the least upper bound of the pair $\{x,y\}$ (if it exists), [28].

supremum r X, $\sup_{\mathrm{E}} \mathrm{X}$, is the least upper bound of X (if it exists), [28].

sup_graph r f, $\sup_{x\in \mathrm{A}} f(x)$, is the least upper bound of the image of the graph f (if it exists), [30].

surjective f, *surjectiveC f*, means that f is a surjection.

symmetric_r r says that the relation r is symmetric.

target f contains the range of the graph of a correspondence f.

the_greatest_element r, *the_least_element_pr r* denoes the greatest or least element of the ordering r, [23].

transf_axioms f A B says that for all $x \in \mathrm{A}$ we have $f(x) \in \mathrm{B}$, case where $\mathscr{L}_{\mathrm{A};\mathrm{B}} f$ is a function.

transfinite_def r p f, $\mathscr{I}(\mathrm{E}, p, f)$, says that f is defined by transfinite induction on the set E, well-ordered by r, via the property p, [48].

transfinite_defined r p is the function defined by the property p by transfinite induction on the well-ordered set r, [48].

transitive_r r says that the relation r is transitive.

transitive_set x says that if $a \in b$ and $b \in x$ then $a \in x$, [83].

total_order r means that r is a total order, [36].

two_points is the basic doubleton.

union X, $\bigcup \mathrm{X}$, is the union of a set of sets.

uniont I f, *unionf x f*, *uniont g*, $\bigcup_{\iota\in \mathrm{I}} \mathrm{X}_\iota$ is the set elements a with $a \in \mathrm{X}_\iota$ for some $\iota \in \mathrm{I}$.

union2 a b, $a \cup b$, is the union of two sets.

upper_bound r X x says that for all $y \in \mathrm{X}$, we have $y \leq_r x$, [26].

V x f, $\mathscr{V}(x,f)$ or $\mathscr{V}_f x$, is the value at the point x of the graph f.

variant, see *Lvariant*.

W x f, $\mathscr{W}_f x$, is the value at the point x of the function f.

well-ordered set, well-ordering relation, well-ordering: see *worder*.

worder r says that r is a well-ordering, [41]

worder_r r says that r is a relation, that induces a well-ordering on each set where it is reflexive, [41].

Xo f y, $\mathscr{X}(f,y)$, this is $f(x)$ if $y = \mathscr{R} x$.

Yo P x y, $\mathscr{Y}(\mathrm{P},x,y)$, is a function that associates to z the value x is P is true, and y if P is false.

Zo x R, $\mathscr{Z}(x,\mathrm{R})$, $\mathscr{E}_x(\mathrm{R})$ or $\{x,\mathrm{R}\}$: it is the set of all x that satisfy R.

Chapter 12

Compatibility

This chapter explains the differences between the current version and the previous one.

12.1 Changes to Chapter 5

12.1.1 Pseudo-ordinals and the type nat

Our implementation of Bourbaki in Coq relies on the fact that a set is a type, and if a is a set, $a \in \mathrm{B}$ means $a = \mathscr{R}b$ for some b of type B (where $\mathscr{R}$ denotes *Ro*). This is an abstract construction. If we define a type A with two constructors B and C, then $\mathscr{R}\mathrm{B} \in \mathrm{A}$ and $\mathscr{R}\mathrm{C} \in \mathrm{A}$. We assume $\mathscr{R}$ injective; since $\mathrm{B} \neq \mathrm{C}$ by construction, the set A has two distinct elements $\mathscr{R}\mathrm{B}$ and $\mathscr{R}\mathrm{C}$. The only property of $\mathscr{R}\mathrm{B}$ is that it is one of the two elements of A. A property of the form $\mathscr{R}\mathrm{B} = \emptyset$ is undecidable.

Let's now define a more complicated type, *nat*, denoted $\mathbb{N}$; it has a constant constructor O, and another constructor S that is a function on $\mathbb{N}$. This means that, whenever x is of type $\mathbb{N}$, then $\mathrm{S}x$ is also of type $\mathbb{N}$. This set satisfies the principle of induction (that says under which condition a property is true for the elements of this set), and we can define a function by induction. Later on, Bourbaki introduces **N**, the set of finite cardinals. It satisfies the principle of induction (see section 5.4), and functions can be defined by induction (Chapter six, section 7.2), as a variant of definition by transfinite induction of the well-ordered set **N**. In the next section, we shall show that $\mathbb{N}$ and **N** are isomorphic; in this section we compare $\mathbb{N}$ and the collection of finite pseudo-ordinals (this is in fact a set, but it will not be used).

Since $\mathrm{S}n$ is of type $\mathbb{N}$ whenever n is of type $\mathbb{N}$, we can define a function s such that $s(a) \in \mathbf{N}$ whenever $a \in \mathbb{N}$: if $a \in \mathbb{N}$ and $a = \mathscr{R}(b)$ then $s(a) = \mathscr{R}(\mathrm{S}(b))$. As noted above, a property of the form $\mathscr{R}\mathrm{O} = \emptyset$ is undecidable. Although the exact value of $s(\mathscr{R}\mathrm{O})$ is unknown, we can show some properties of s: for instance, s is injective and not surjective (there is a unique way to construct an object of type $\mathbb{N}$; so that $\mathrm{S}x$ is never O and $\mathrm{S}x = \mathrm{S}y$ implies $x = y$). The Coq parser and pretty printer identify O and 0, SO and 1, SSO and 2. In order to avoid confusion, we shall write **0**, **1** and **2** for the cardinals (note that $\mathbf{0} = \emptyset$).

Let's define by induction a function f by $f(\mathbf{0}) = \mathscr{R}\mathrm{O}$ and $f(n+\mathbf{1}) = \mathscr{R}(\mathrm{S}(\mathscr{B}(f(n))))$ where $a = \mathscr{B}(f(n))$ is defined by $\mathscr{R}a = f(n)$. For instance $f(\mathbf{1}) = \mathscr{R}(1)$. The property "$f$ is the identity function" (more precisely $f = \mathscr{R}$) is undecidable (it cannot be proved; we hope that adding it as an axiom does not make the theory contradictory).

The type $\mathbb{N}$ is called *nat* in Coq; it has an order relation, noted $\leq$, and two operations $+$ and $*$, that correspond, via the bijection f, to comparison, sum and product of finite cardinals. We shall import all theorems about natural integers from the Coq library by identification of **N** and $\mathbb{N}$. Given that $\emptyset = f(\mathbf{0}) = \mathscr{R}0 = \mathscr{R}\mathrm{O}$, we may assume $\mathscr{R}\mathrm{O} = \emptyset$. The relation $f(\mathbf{1}) = \mathscr{R}(1)$ suggest that $\mathscr{R}(1)$ should be **1**, but this is a set defined via the axiom of choice, as a set with one element; it could be $\{\emptyset\}$, it could also be any other set. We will add the relation $\mathscr{R}(\mathrm{SO}) = \{\emptyset\}$ as axiom. As a consequence $\mathscr{R}(\mathrm{SO})$ is unlikely to be a cardinal, but it will allow us to construct a function card, such that $\mathrm{card}(x) = 1$ whenever x is a singleton, i.e., whenever $\mathrm{Card}(x) = \mathbf{1}$. The two axioms relating $\mathscr{R}$, S and O have been introduced by Carlos Simpson in the following way:

```
(*
   Axiom nat_realization_O : forall x : Set, ~ inc x (Ro 0).
   Axiom nat_realization_S :
          forall (n : nat) (x : Set),
          inc x (Ro (S n)) = (inc x (Ro n) \/ x = Ro n).
   Lemma nat_zero_emptyset : Ro 0 = emptyset.
*)
```

These axioms are useless, hence have been withdrawn. On the other hand, we can define a function that shares exactly the same properties. The first axioms defines a set $\mathscr{R}0$ that contains no element, hence is the emptyset. The second axioms defines $\mathscr{R}(\mathrm{S}n)$, that is equal (by extensionality) to $\mathrm{T}(\mathscr{R}n)$. Thus, we define *natR* denoted by $\mathscr{R}_{\mathbb{N}}$, via $\mathscr{R}_{\mathbb{N}}0 = \emptyset$ and $\mathscr{R}_{\mathbb{N}}(\mathrm{S}n) = \mathrm{T}(\mathscr{R}_{\mathbb{N}}n)$.

```
Fixpoint natR (n:nat) :=
  match n with 0 =>  emptyset
             | S p => tack_on (natR p) (natR p)
  end.
```

The conclusion of Exercise 20 is: *In particular the pseudo-ordinals whose order-type are* $0, 1, 2 = 1+1,$ *and* $3 = 2+1$ *are respectively*

$$\emptyset, \quad \{\emptyset\}, \quad \{\emptyset,\{\emptyset\}\}, \quad \{\emptyset,\{\emptyset\},\{\emptyset,\{\emptyset\}\}\}.$$

```
Lemma value_R_0: natR 0 = emptyset.
Lemma value_R_1: natR 1 = singleton emptyset.
Lemma value_R_2: natR 2 = doubleton (singleton emptyset) emptyset.
Lemma value_R_3: let tripleton a b c := tack_on (doubleton a b) c in
  natR 3 = tripleton (doubleton (singleton emptyset) emptyset)
  (singleton emptyset) emptyset.
```

If a is a pseudo-ordinal, then so is $\mathrm{T}(a)$. By induction, we deduce that if n is of type *nat*, $\mathscr{R}_{\mathbb{N}}n$ is a finite pseudo-ordinal.

```
Lemma pseudo_ordinal_Rnat: forall i,  pseudo_ordinal (natR i).
Lemma finite_Rnat: forall i, is_finite_set (natR i).
```

Define a relation $\leq$ on *nat* by the properties: $\forall x, x \leq x$ and $\forall x y,\ x \leq y \implies x \leq \mathrm{S}(y)$. This is reflexive and transitive. Showing that it is an order is not completely trivial (see the Coq library). One can show that the relation $x < y$, defined by $\mathrm{S}x \leq y$, is equivalent to $x \leq y$ and $x \neq y$.

It is clear by induction that if $x \leq y$ then $\mathscr{R}_{\mathbb{N}}x \subset \mathscr{R}_{\mathbb{N}}y$. If $x < y$ then $\mathscr{R}_{\mathbb{N}}(\mathrm{S}x) \subset \mathscr{R}_{\mathbb{N}}y$ hence $\mathscr{R}_{\mathbb{N}}x \in \mathscr{R}_{\mathbb{N}}y$. If $\mathscr{R}_{\mathbb{N}}x \subset \mathscr{R}_{\mathbb{N}}y$, then $x \leq y$, for otherwise we would have $y < x$, hence $\mathscr{R}_{\mathbb{N}}y \in \mathscr{R}_{\mathbb{N}}x$, hence $\mathscr{R}_{\mathbb{N}}x \in \mathscr{R}_{\mathbb{N}}x$, absurd. In the same fashion, $x < y$ is equivalent to $\mathscr{R}_{\mathbb{N}}(\mathrm{S}x) \subset \mathscr{R}_{\mathbb{N}}y$. Note that, if $\mathscr{R}_{\mathbb{N}}i = \mathscr{R}_{\mathbb{N}}j$, then $\mathscr{R}_{\mathbb{N}}i \subset \mathscr{R}_{\mathbb{N}}j$ and $\mathscr{R}_{\mathbb{N}}j \subset \mathscr{R}i$, this $i \leq j$ and $j \leq i$; hence $i = j$. This shows injectivity of $\mathscr{R}_{\mathbb{N}}$.

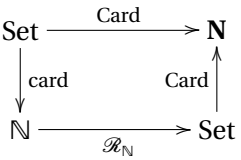
```
Lemma Rnat_le_implies_sub : forall i j,  i <= j -> sub (natR i) (natR j).
Lemma Rnat_lt_implies_inc : forall i j,  i < j -> inc (natR i) (natR j).
Lemma Rnat_lt_implies_strict_sub : forall i j,
  i < j -> strict_sub (natR i) (natR j).
Lemma Rnat_sub_le : forall i j,  sub (natR i) (natR j) = (i <= j).
Lemma Rnot_inc_itself: forall i, ~ (inc (natR i)(natR i)).
Lemma Rnat_inc_lt : forall i j,  inc (natR i) (natR j) = (i < j).
```

As a consequence, if $i < j$ then $\mathrm{Card}(\mathscr{R}_{\mathbb{N}}i) <_{\mathrm{Card}} \mathrm{Card}(\mathscr{R}_{\mathbb{N}}j)$, and if $\mathrm{Card}(\mathscr{R}_{\mathbb{N}}i) = \mathrm{Card}(\mathscr{R}_{\mathbb{N}}j)$, then $i = j$. From this, we deduce that each finite cardinal is of the form $\mathrm{Card}(\mathscr{R}_{\mathbb{N}}(i))$ for a unique i.

```
Lemma cardinal_Rnat_lt: forall i j,
  i< j -> cardinal_lt (cardinal (natR i)) (cardinal (natR j)).
Lemma  cardinal_Rnat_inj: forall i j,
  cardinal (natR i) = cardinal (natR j) -> i = j.
Lemma exists_nat_cardinal: forall a, is_finite_c a ->
  exists_unique(fun i:nat => cardinal (natR i)=a).
```

We can now introduce a function $\mathrm{card}(n)$ making the following diagram commutative

(Finite cardinals)



In this diagram, $\mathbb{N}$, Set and **N** are types; $\mathbb{N}$ is the set of natural numbers (as a Coq type), and **N** is the collection of cardinals (the objects x such that x is a cardinal do not form a set, neither a type, we call it a *collection*). We use the notation **N** to emphasize that if x is a finite set, then its cardinal is a member of the set of finite cardinals. If x is not finite, we define $\mathrm{card}(x)$ to be 0, in this case the diagram does not commute. If however n is finite we have $\mathrm{Card}(\mathscr{R}_{\mathbb{N}}(\mathrm{card}(n))) = \mathrm{Card}(n)$. By uniqueness, we have $\mathrm{card}(\mathscr{R}_{\mathbb{N}}(i)) = i$ for every nat i. If A is a finite set, then $\mathrm{Card}\,\mathrm{A} = \mathrm{Card}\,\mathrm{B}$ implies $\mathrm{card}\,\mathrm{A} = \mathrm{card}\,\mathrm{B}$. The converse is true if both sets are finite.

```
Definition cardinal_nat x := choosenat(fun i => cardinal (natR i) = x).
Lemma cardinal_nat_cardinal: forall x,
  cardinal_nat (cardinal x) = cardinal_nat x.
Lemma cardinal_nat_pr: forall x, is_finite_set x ->
  cardinal (natR (cardinal_nat x)) = cardinal x.
Lemma cardinal_nat_pr1: forall i, cardinal_nat(natR i) = i.
Lemma cardinal_nat_finite_eq: forall a b, is_finite_set a ->
  cardinal a = cardinal b -> cardinal_nat a = cardinal_nat b.
Lemma cardinal_nat_finite_eq1: forall a b,
  is_finite_set a -> is_finite_set b ->
  cardinal_nat a = cardinal_nat b -> cardinal a = cardinal b.
```

We have card $\mathbf{0} = 0$, card $\mathbf{1} = 1$ and card $\mathbf{2} = 2$. Note that, if s is the successor function, then $\mathbf{3} = s(\mathbf{2})$ and $3 = S2$, this implies card $\mathbf{3} = 3$.

```
Lemma cardinal_nat_emptyset: cardinal_nat emptyset = 0.
Lemma cardinal_nat_singleton: forall x, cardinal_nat (singleton x) = 1.
Lemma cardinal_nat_doubleton: forall x y,
  x <> y -> cardinal_nat (doubleton x y) = 2.
Lemma cardinal_nat_zero: cardinal_nat card_zero = 0.
Lemma cardinal_nat_one: cardinal_nat card_one = 1.
Lemma cardinal_nat_two: cardinal_nat card_two = 2.
```

12.1.2 Bijection between nat and the integers

Denote by $s(n)$ the successor of n. We have $s(0) = 1$, and $a + s(n) = s(a + n)$ (associativity of the sum). We have $a.s(b) = ab + a$ and $a.0 = 0$. By induction we deduced that the sum and product of two integers are integers.

```
Lemma plus_via_succ: forall a n,
  card_plus a (succ n) = succ (card_plus a n).
Lemma Bnat_stable_plus: forall a b, inc a Bnat -> inc b Bnat ->
  inc (card_plus a b) Bnat.
Lemma mult_via_plus: forall a b, is_cardinal a ->
  card_mult a (succ b) = card_plus (card_mult a b) a.
Lemma Bnat_stable_mult: forall a b, inc a Bnat -> inc b Bnat ->
  inc (card_mult a b) Bnat.
```

We define now by induction a function $\mathcal{N}$ that associates a cardinal to each *nat*; by construction $\mathcal{N}(0) = \mathbf{0}$ and $\mathcal{N}(Sn) = s(\mathcal{N}(n))$. This function converts addition and multiplication on *nat* to cardinal sum and cardinal product (induction on $\mathbb{N}$). This function is injective (because *succ* is injective). By induction on **N**, this function is surjective. More precisely, every finite cardinal has the form $\mathcal{N}(n)$. Assume $a \leq b$; then $\mathcal{N}(a) \leq_{\mathrm{Card}} \mathcal{N}(b)$; conversely, if this relation holds then $\mathcal{N}(a) + x = \mathcal{N}(b)$ for some x. By surjectivity $x = \mathcal{N}(c)$, by injectivity $a + c = b$ which implies $a \leq b$. By injectivity of $\mathcal{N}$ we deduce that $a < b$ and $\mathcal{N}(a) <_{\mathrm{Card}} \mathcal{N}(b)$ are equivalent.

Let g be the inverse of $\mathcal{N}$; such a function exists using the axiom of choice. By uniqueness this function is card, hence $\mathrm{card}(\mathcal{N}(x)) = x$ and $\mathcal{N}(\mathrm{card}(x)) = x$.

```
Fixpoint nat_to_B (n:nat) :=
  match n with 0 => card_zero | S m => succ (nat_to_B m) end.

Lemma nat_B_0: nat_to_B 0 = card_zero.
Lemma nat_B_1: nat_to_B 1 = card_one.
Lemma nat_B_2:  nat_to_B 2 = card_two.
Lemma nat_B_S: forall n, nat_to_B (S n) = succ (nat_to_B n).
Lemma inc_nat_to_B: forall n, inc (nat_to_B n) Bnat.
Lemma nat_B_plus: forall a b,
  nat_to_B (a+b) = card_plus (nat_to_B a) (nat_to_B b).
Lemma nat_B_mult: forall a b,
  nat_to_B (a*b) = card_mult (nat_to_B a) (nat_to_B b).

Lemma nat_B_inj: forall a b,
  nat_to_B a = nat_to_B b -> a = b.
```

```
Lemma nat_to_B_surjective: forall n, inc n Bnat -> exists m,
  nat_to_B m = n.
Lemma nat_B_le: forall a b,
  (a<= b) =  cardinal_le (nat_to_B a) (nat_to_B b).
Lemma nat_B_lt: forall a b,
  (a< b) =  cardinal_lt (nat_to_B a) (nat_to_B b).
Lemma nat_B_lt: forall a b,
  (a< b) =  cardinal_lt (nat_to_B a) (nat_to_B b).
Lemma nat_B_lt0: forall b,
  (0<b) = cardinal_lt card_zero (nat_to_B b).
Lemma nat_to_B_pr: forall n, inc n Bnat ->
  nat_to_B (cardinal_nat n) = n.
Lemma nat_to_B_pr1: forall n,
  cardinal_nat(nat_to_B n) = n.
```

We finish with the definition of the power function on $\mathbb{N}$.

```
Fixpoint pow (n m:nat) {struct m} : nat :=
  match m with
  | O => 1
  | S p => (pow n p) * n
  end
where "n ^ m" := (pow n m) : nat_scope.
```

12.2 Introduction to Chapter 6

Our current work is based on the ssreflect library, which redefined a certain number of functions on natural numbers. For instance the type of $a \leq b$ is now *bool* instead of *Prop*. Below is a theorem *zerop* that says that a number is zero or positive. This is restated in *ssrat* by the lemma *posnP* that says that the boolean value of $n ==$ or $0 < n$ are mutually exclusive (one is true, the other one is false).

We start this chapter with a list of some definitions and theorems, extracted from the Coq standard library implemening $\mathbb{N}$. Consider for instance *zerop*. We show its type, not its value which is irrelevant for the use we shall make of it; this value is a proof that for every n (of type *nat*), one of A or B is true. This is summarized by the notation {A} + {B} (certified disjoint union). The *heavyside* function is not part of the library, it is an example of how this construction can be used. The underscores in the definition represent the two proofs. The Coq parser and pretty-printer interpret this in the same fashion as *if zerop n then 0 else 1*. A property is decidable if it can be shown true of false. For us, all properties are decidable since we have an axiom that says so. It is however useful to know that equality and inequality are decidable. We state also some theorems such as if $n \leq m$ is false then $n > m$ is true. In this case, the result is a consequence of the fact that one of the properties is true.

```
(*
  Definition zerop n : {n = 0} + {0 < n}.
  Definition lt_eq_lt_dec n m : {n < m} + {n = m} + {m < n}.
  Definition gt_eq_gt_dec n m : {m > n} + {n = m} + {n > m}.
  Definition le_lt_dec n m : {n <= m} + {m < n}.
  Definition le_le_S_dec n m : {n <= m} + {S m <= n}.
  Definition le_ge_dec n m : {n <= m} + {n >= m}.
  Definition le_gt_dec n m : {n <= m} + {n > m}.
  Definition le_lt_eq_dec n m : n <= m -> {n < m} + {n = m}.
```

```
Definition heavyside n := match (zerop n) with left _ => 0 | right _ => 1 end.

  Theorem dec_le : forall n m, decidable (n <= m).
  Theorem dec_lt : forall n m, decidable (n < m).
  Theorem dec_gt : forall n m, decidable (n > m).
  Theorem dec_ge : forall n m, decidable (n >= m).
  Theorem not_eq : forall n m, n <> m -> n < m \/ m < n.
  Theorem not_le : forall n m, ~ n <= m -> n > m.
  Theorem not_gt : forall n m, ~ n > m -> n <= m.
  Theorem not_ge : forall n m, ~ n >= m -> n < m.
  Theorem not_lt : forall n m, ~ n < m -> n >= m.
*)
```

Here we show how and addition and multiplicaton behave with respect to ordering.

```
(*
  Lemma plus_reg_l : forall n m p, p + n = p + m -> n = m.
  Lemma plus_le_reg_l : forall n m p, p + n <= p + m -> n <= m.
  Lemma plus_lt_reg_l : forall n m p, p + n < p + m -> n < m.

  Lemma plus_le_compat_l : forall n m p, n <= m -> p + n <= p + m.
  Lemma plus_le_compat_r : forall n m p, n <= m -> n + p <= m + p.
  Lemma le_plus_l : forall n m, n <= n + m.
  Lemma le_plus_r : forall n m, m <= n + m.
  Theorem le_plus_trans : forall n m p, n <= m -> n <= m + p.
  Theorem lt_plus_trans : forall n m p, n < m -> n < m + p.
  Lemma plus_lt_compat_l : forall n m p, n < m -> p + n < p + m.
  Lemma plus_lt_compat_r : forall n m p, n < m -> n + p < m + p.
  Lemma plus_le_compat : forall n m p q, n <= m -> p <= q -> n + p <= m + q.
  Lemma plus_le_lt_compat : forall n m p q, n <= m -> p < q -> n + p < m + q.
  Lemma plus_lt_le_compat : forall n m p q, n < m -> p <= q -> n + p < m + q.
  Lemma plus_lt_compat : forall n m p q, n < m -> p < q -> n + p < m + q.

  Lemma mult_O_le : forall n m, m = 0 \/ n <= m * n.
  Lemma mult_le_compat_l : forall n m p, n <= m -> p * n <= p * m.
  Lemma mult_le_compat_r : forall n m p, n <= m -> n * p <= m * p.
  Lemma mult_le_compat :
    forall n m p (q:nat), n <= m -> p <= q -> n * p <= m * q.
  Lemma mult_S_lt_compat_l : forall n m p, m < p -> S n * m < S n * p.
  Lemma mult_lt_compat_r : forall n m p, n < m -> 0 < p -> n * p < m * p.
  Lemma mult_S_le_reg_l : forall n m p, S n * m <= S n * p -> m <= p.
  Lemma plus_le_reg_l : forall n m p, p + n <= p + m -> n <= m.
  Lemma plus_lt_reg_l : forall n m p, p + n < p + m -> n < m.
  Lemma mult_S_le_reg_l : forall n m p, S n * m <= S n * p -> m <= p.
*)
```

Here we give some properties of subtraction.

```
(*
   Lemma minus_n_n : forall n, 0 = n - n.
   Lemma minus_n_O : forall n, n = n - 0.
   Lemma le_plus_minus : forall n m, n <= m -> m = n + (m - n).
   Lemma plus_minus : forall n m p, n = m + p -> p = n - m.
   Lemma minus_plus : forall n m, n + m - n = m.
   Theorem le_minus : forall n m, n - m <= n.
```

```
   Lemma minus_plus_simpl_l_reverse : forall n m p, n - m = p + n - (p + m).
   Lemma minus_Sn_m : forall n m, m <= n -> S (n - m) = S n - m.
   Lemma le_plus_minus : forall n m, n <= m -> m = n + (m - n).
   Lemma le_plus_minus_r : forall n m, n <= m -> n + (m - n) = m.
   Lemma lt_minus : forall n m, m <= n -> 0 < m -> n - m < n.
   Lemma lt_O_minus_lt : forall n m, 0 < n - m -> m < n.
   Theorem not_le_minus_0 : forall n m, ~ m <= n -> n - m = 0.
*)
```

Additional theorems about integers.

```
Theorem lt_to_plus: forall a b:nat, a<b = exists c:nat, 0<c & c+a=b.
Lemma mult_lt_le_compat : forall n m p q,
  0<q -> n < m -> p <= q -> n * p < m * q.
Lemma mult_le_lt_compat : forall n m p q,
  0< m -> n <= m -> p < q -> n * p < m * q.
Lemma zero_lt_oneN: 0 < 1.
Lemma lt_n_succ_leN: forall a b,  a < b -> S a <= b.
Lemma power_x_0N: forall a, a ^ 0 =  1.
Lemma power_0_0N:  0 ^ 0 = 1.
Lemma power_x_1N: forall a,  a ^ 1 = a.

Lemma plus_simplifiable_leftN: forall a b b':nat,
  a + b = a + b' -> b = b'.
Lemma plus_simplifiable_rightN: forall a b b':nat,
  b + a = b' + a -> b = b'.
Lemma Sn_is_1plus: forall n, S n = 1 + n.
Lemma Sn_is_plus1: forall n, S n = n +1.
Lemma lt_i_n : forall i n, i< n  -> 1 <= n-i.
Lemma double_subN: forall n p, p <= n -> n - (n- p) = p.
Lemma nonzero_suc: forall n, 0<>n -> exists m, n = S m.
Lemma mult_S_lt_reg_l : forall n m p, S n * m < S n * p -> m < p.
Lemma mult_lt_reg_l : forall n m p, 0<>n -> n * m < n * p -> m < p.
Lemma mult_lt_reg_r : forall n m p, 0<>n -> m * n <  p * n -> m < p.
Lemma minus_wrong: forall n m, n<=m -> n-m = 0.
Lemma pred_minus: forall n m, m<n -> n-m =  S(n - S m).

Lemma plus_n_Sm_subSn: forall n m, n + S m - n - 1 = m.
Lemma plus_n_Sm_subSm: forall n m, n + S m - m - 1 = n.

Lemma minus_SnSi: forall i n, i < S n -> S n - i - 1 = n - i.
Lemma double_compl_nat:forall i n, i< n ->
  i = n - (n - i - 1) -1.
Lemma double_compl_ex:forall i n, i< n -> (n - i - 1) < n.
Lemma plus_reg_r : forall n m p, n + p = m + p -> n = m.
```

12.3 Theorems removed from Chapter 6

We study here the function *pow* on the type *nat*.

```
Lemma power_of_sumN: forall a b c, a ^ (b+c) = (a ^ b) *(a ^ c).
Lemma nat_B_pow: forall n m,
  nat_to_B (n ^ m) = card_pow (nat_to_B n)(nat_to_B m).
Lemma power_of_prodN: forall a b c,
```

```
  (a * b)  ^ c =  (a ^ c) *  (b ^ c).
Lemma power_1_xN: forall a,  1 ^ a = 1.
Lemma nat_not_zero_pr: forall a,  a <> 0 -> nat_to_B a <> card_zero.
Lemma power_0_x: forall a, a <> 0 -> 0 ^ a = 0.
Lemma non_zero_apowbN: forall a b, 0 < a -> 0 < a ^ b.
Lemma finite_power_lt1N: forall a a' b, a < a' -> 0 <  b ->  a ^ b < a' ^ b.
Lemma finite_power_lt2N: forall a b b',
  b < b' -> 1 < a ->  a^ b < a ^ b'.
Lemma mult_simplifiable_leftN: forall a b b':nat,
  0 <>a -> a * b = a * b' -> b = b'.
Lemma mult_simplifiable_rightN: forall a b b',
  0 <>a -> b * a = b' * a -> b = b'.
```

Other removed theorems from chapter 6

```
Lemma nat_B_sub: forall a b,
  nat_to_B (a-b) = card_sub (nat_to_B a) (nat_to_B b).
Lemma card_sub_pr4N: forall a b a' b',
  a<=b -> a' <= b' ->  (b-a) + (b'-a') = (b+b')- (a+a').
Lemma card_sub_associativeN: forall a b c,
  (b +c) <= a -> (a-b) -c = a - (b+c).
Lemma nat_B_pred: forall a, 0 <> a -> nat_to_B (pred a) = predc (nat_to_B a).
Lemma prec_is_cardinal_prec: forall a, inc a Bnat ->
  a <> card_zero -> cardinal_nat (prec a) = pred (cardinal_nat a).
Lemma card_sub_associative1N: forall a b,
  (S b) <= a -> pred (a - b) = a  - S b.
```

12.3.1 Division

Euclidean division is curiously defined in Coq. The following two lemmas say that if $b > 0$, then for every a we have $\{x : \mathbb{N} | \exists y : \mathbb{N},\ Z\}$ where Z is the division property $a = bq + r$ and $r < b$, and (x, y) is (q, r) or (r, q). This expression is a type; from it one can extract q and the associated property (namely that there is r such that Z).

```
(*
  Lemma quotient :
    forall n,
      n > 0 ->
      forall m:nat, {q : nat | exists r : nat, m = q * n + r /\ n > r}.
  Lemma modulo :
    forall n,
      n > 0 ->
      forall m:nat, {r : nat | exists q : nat, m = q * n + r /\ n > r}.
*)
```

The ssreflect library has a clever definition of quotient and remainder (the first definition defines quotient and remainder, the second defines only the quotient). The two functions *divn* and *modn* are deduced from these recursive function. We give one theorem that showq how these quantities can be used.

```
Definition edivn_rec d := fix loop (m q : nat) {struct m} :=
  if m - d is m'.+1 then loop m' q.+1 else (q, m).
Definition modn_rec d := fix loop (m : nat) :=
  if m - d is m'.+1 then loop m' else m.
```

```
Lemma divn_eq : forall m d, m = m %/ d * d + m %% d.
```

In the previous version of this document, we explained another implementation of these operations. It is not used anymore. These lemmas show existence and uniqueness of division.

```
Lemma least_int_prop0: forall p:nat->Prop,
  ~(p 0) -> (exists x, p x) -> (exists x, p (S x) & ~ p x).

Lemma division_prop_nat: forall a b q r, 0 <>b ->
  (a=b*q+r & r<b) = (b*q <= a & a < b* (S q) & r = a - (b*q)).
Lemma Ndivision_unique: forall a b q q' r r', 0 <> b ->
  a = b* q + r -> r < b -> a = b* q' + r' -> r'< b ->
  (q = q' & r = r').
Lemma Ndivision_existence: forall a b, 0 <> b ->
  exists q, exists r, (a = b* q + r & r < b).
```

Hence we provide the following definition. For simplicity, in the case $b = 0$, we shall use $b = 1$ instead (quotient one, remainder zero). However, we say that b divides a only when b is non-zero.

```
Definition Nquo a b :=
  cardinal_nat (card_quo (Ro (nat_to_B a))
    (Yo (b = 0) card_one (Ro (nat_to_B b)))).
Definition Nrem a b :=
  cardinal_nat (card_rem (Ro (nat_to_B a))
    (Yo (b = 0) card_one (Ro (nat_to_B b)))).
Definition Ndivides b a:= 0 <> b & Nrem a b = 0.

Lemma Ndivision_exists: forall a b,  0 <> b ->
  (a = b* (Nquo a b) + (Nrem a b) & (Nrem a b < b)).
Lemma Ndivision_pr: forall a b q r, 0 <> b ->
  a = b* q + r -> r < b -> (q = Nquo a b & r = Nrem a b).

Lemma Ndivision_pr_q: forall a b q r, 0 <> b ->
  a = b* q + r -> r < b -> q = Nquo a b.
Lemma Ndivision_pr_r: forall a b q r, 0 <> b ->
  a = b* q + r -> r < b -> r = Nrem a b.
```

All properties true for *Nquo* and *Nrem* are true for *card_quo* and *card_rem*. For this reason, we shall only prove our theorems for the case of type *nat*.

```
Lemma nat_B_division: forall a b, 0 <> b ->
  (nat_to_B (Nquo a b) =  card_quo (nat_to_B a) (nat_to_B b) &
    nat_to_B (Nrem a b) =  card_rem (nat_to_B a) (nat_to_B b)).
Lemma nat_B_quo: forall a b, 0 <> b ->
  nat_to_B (Nquo a b) =  card_quo (nat_to_B a) (nat_to_B b).
Lemma nat_B_rem: forall a b, 0 <> b ->
  nat_to_B (Nrem a b) =  card_rem (nat_to_B a) (nat_to_B b).
```

Now some consequences when division is exact. Bourbaki says: every multiple a' of a multiple a of b is a multiple of b. One can restate this as: if b divides a, then b divides ac.

```
Lemma inc_quotient_bnat:forall a b, inc a Bnat-> inc b Bnat -> b <> card_zero ->
```

```
  inc (card_quo a b) Bnat.
Lemma inc_remainder_bnat:forall a b,
    inc a Bnat-> inc b Bnat -> b <> card_zero ->
    inc (card_rem a b) Bnat.

Lemma Ndivides_pr: forall a b,
  Ndivides b a -> a =  b * (Nquo a b).
Lemma Ndivides_pr1: forall a b,  0 <> b ->  Ndivides b (b *a).
Lemma Ndivides_pr2: forall a b q,  0 <> b ->
  a = b * q -> q = Nquo a b.
Lemma one_divides_all: forall a, Ndivides 1 a.
Lemma Ndivides_pr3: forall a b q,
  Ndivides b a -> q = Nquo a b -> a =  b * q.
Lemma Ndivides_pr4: forall b q, 0 <> b ->
  Nquo (b * q) b = q.
Lemma Ndivision_itself: forall a, 0 <> a ->
  (Ndivides a a & Nquo a a = 1).
Lemma Ndivides_itself: forall a, 0 <> a -> Ndivides a a.
Lemma Nquo: forall a, 0 <> a -> Nquo a a = 1.
Lemma Ndivision_of_zero: forall a, 0 <> a ->
  (Ndivides a 0 & Nquo 0 a = 0).
Lemma Ndivides_trans: forall a b a', Ndivides a a'-> Ndivides b  a
  -> Ndivides b a'.
Lemma Ndivides_trans1: forall a b a', Ndivides a a'-> Ndivides b a
  -> Nquo a' b =  (Nquo a' a) *(Nquo a b).
Lemma Ndivides_trans2: forall a b c,
  Ndivides b a-> Ndivides b (a *c).
Lemma non_zero_mult: forall a b, 0 <> a -> 0 <> b -> 0 <> (a*b).
Lemma Nquo_simplify:  forall a b c, 0 <> b -> 0 <> c ->
  Nquo (a * c) (b * c) = Nquo a b.
```

If b divides a and a', it divides the sum and the difference.

```
Lemma divides_and_sum:  forall a a' b,  Ndivides b a ->  Ndivides b a'
  -> (Ndivides b  (a + a') &
    Nquo  (a + a')  b =  (Nquo a b) + (Nquo a' b)).
Lemma distrib_prod2_subN: forall a b c, c<= b->
  a * (b-c) = (a*b) - (a*c).
Lemma divides_and_difference:  forall a a' b, a' <= a ->
  Ndivides b a ->  Ndivides b a'
  -> (Ndivides b  (a -a') &
    (Nquo a' b) <= (Nquo a b) &
    Nquo  (a - a')  b =  (Nquo a b) -  (Nquo a' b)).
```

The following lemma may have some intererest, but is currently unused. Assume $a = bq + r$ with $r < b$, where a and b are integers, b is non-zero. The last relation says that r is an integer. The quantity bq is also an integer, so that q is finite. If q is a cardinal, we deduce that q is an integer.

```
Lemma division_result_integer: forall a b q r,
  inc a Bnat-> inc b Bnat ->
  b <> card_zero -> division_prop a b q r -> is_cardinal q ->
  (inc q Bnat & inc r Bnat).

Lemma lt_a_power_b_aN: forall a b,  1< b -> a < pow b a.
```

12.4 Finite sequences and lists

If P⦃i⦄ is equivalent to $i \in \mathrm{I}$, where I is a finite set of integers, then $(x_i)_{i\in\mathrm{I}}$ may be written as $(x_i)_{\mathrm{P}⦃i⦄}$. In fact, such a notation can be used whatever I. As an example one can see $(t_i)_{a\le i\le b}$. The sum of such a family may be denoted by $\sum_{i=a}^{b} t_i$.

Lists are defined in Coq by

```
Inductive list (A : Type) : Type :=
    nil : list A
 | cons : A -> list A -> list A
```

A list can be either empty (this is *nil*), or of the form *cons A a b* where a is of type A and b is a list of type A. The parameter A is often implicit. The expression *cons A a b* is denoted by $a::b$. There are many functions in the standard library that deal with lists. For instance, *seq* can produce the list containing 1, 2, 3. Given a list containing x_1, x_2 and x_3 (of type A) it is possible to create the list containing $(1,x_1)$, $(2,x_2)$, and $(3,x_3)$ (of type $\mathbb{N}\times\mathrm{A}$) then the set of all these values. This is a finite sequence (i.e., a functional graph, with domain $\{1,2,3\}$). In this section, we explain how to convert operations defined by Bourbaki for finite sequences (like sum and product) into operations on Coq lists.

12.4.1 Lists as functions

Given a function g, we define here the list L containing $g(0)$, $g(1)$, $g(2)$ up to $g(n-1)$. The list has length n; it is stored in natural order[1]: On the diagram below, the mapping $g \mapsto \mathrm{L}$ is denoted by *fl*. Conversely given a L of length n, we define a function g that returns the k-th element of the list, and 0 if $k \ge n$. It will be denoted by *lf* on the diagram below.

We consider a variant of *lf* where L is a list of sets (the default value is then $\emptyset$) and, later on, a variant of *fl*, where g is a function in the Bourbaki sense

```
Fixpoint fct_to_list_rev (A:Type) (f: nat->A)(n:nat): list A :=
  match n with O => nil
    | S m => (f m) ::(fct_to_list_rev f m) end.

Definition fct_to_list A f n := rev (fct_to_list_rev (A:=A) f n).

Definition list_to_fct (a: list nat) :=
  fun n => nth n a 0.

Definition list_to_fctB (a: list Set) :=
  fun n => nth n a emptyset.


Lemma card_interval_c0_pr: forall n,
  cardinal_nat (interval_co_0a (nat_to_B n)) = n.
Lemma list_extens: forall (A:Type) (l1 l2 : list A) (u:A),
   length l1 = length l2 ->
   (forall i, i< length l1 -> nth i l1 u = nth i l2 u) -> l1 = l2.

Lemma fct_to_list_length :   forall A (f:nat->A) n,
  length (fct_to_list f n) = n.
```

[1] In the previous version, we used the other order: $g(n-1)$ was the head of the list

```
Lemma list_to_fct_pr0: forall a l1 l2,
  list_to_fct (l2 ++ a :: l1) (length l2) = a.
Lemma list_to_fct_pr0B: forall a l1 l2,
  list_to_fctB (l2 ++ a :: l1)  (length l2) = a.
Lemma list_to_fct_pr: forall (A:Type) (f:nat->A) (u:A) n i,
  i<n -> nth i (fct_to_list f n) u = f i.
Lemma list_to_fct_pr1: forall f n i,
  i<n -> list_to_fct (fct_to_list f n) i = f i.
Lemma list_to_fct_pr1B: forall f n i,
  i<n -> list_to_fctB (fct_to_list f n) i = f i.
Lemma list_to_fct_pr3: forall l2 l1,
  fct_to_list (list_to_fct (l2++l1)) (length l2) = l2.
Lemma list_to_fct_pr4: forall l,
  fct_to_list (list_to_fct l) (length l) = l.
Lemma list_to_fct_pr3B: forall l2 l1,
  fct_to_list (list_to_fctB (l2++l1)) (length l2) = l2.
Lemma list_to_fct_pr4B: forall l,
  fct_to_list (list_to_fctB l) (length l) = l.
```

Note that if g and g' agree on $[0, n-1]$ then $fl(g) = fl(g')$. On the other hand, if L is a list of size n and $L' = a$::L, if the associated functions are g and g', then $g'(n) = a$, and g and g' agree on $[0, n-1]$.

```
Lemma fct_to_list_unique: forall (A:Type) (f g: nat-> A) n,
  (forall i, i< n -> f i = g i) -> fct_to_list f n = fct_to_list g n.

Lemma app_nth3 : forall A (a:A),
  forall l' d n, n >= 1 -> nth n (a::l') d = nth (n-1) l' d.
```

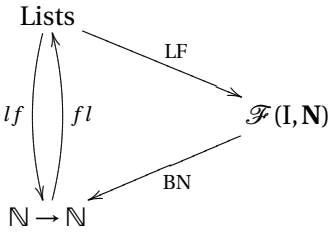
Given a list L of elements of $\mathbb{N}$, of length n, if $g = lf(L)$, we construct a function $G : [0, n[\to \mathbf{N}$ via $g(\mathrm{card}(i)) = \mathrm{card}(G(i))$. The mapping $L \mapsto G$ will be denoted by LF on the diagram below. Similarly, given a list L of sets, we construct $G : [0, n[\to E$ via $g(\mathrm{card}(i)) = G(i)$. This is well-defined if all elements of the list belong to the set E, see later.

```
Definition list_to_f (l: list nat):=
  BL (fun n => nat_to_B (list_to_fct l (cardinal_nat n)))
  (interval_co_0a (nat_to_B (length l))) Bnat.
Definition list_to_fB (l: list Set) E:=
  BL (fun n => list_to_fctB l (cardinal_nat n))
  (interval_co_0a  (nat_to_B (length l))) E.

Lemma list_to_f_axioms: forall (l: list nat),
  transf_axioms (fun n =>(Ro (nat_to_B (list_to_fct l (cardinal_nat n)))))
  (interval_co_0a (nat_to_B (length l))) Bnat.
Lemma list_to_f_function: forall (l: list nat),
  is_function (list_to_f l).
Lemma list_to_f_source: forall (l: list nat),
  source (list_to_f l) = (interval_co_0a (nat_to_B (length l))).
Lemma list_to_f_target: forall (l: list nat),
  target (list_to_f l) = Bnat.
Lemma list_to_f_W: forall (l: list nat) n,
  inc n (interval_co_0a (nat_to_B (length l))) ->
  W n (list_to_f l) = nat_to_B (list_to_fct l (cardinal_nat n)).
```

```
Lemma list_to_f_W1: forall (l: list nat) n,
  n < length l ->
  W (nat_to_B n) (list_to_f l) = nat_to_B (list_to_fct l n).
Lemma list_to_f_W2: forall (l: list nat) n,
  n < length l ->
  cardinal_nat(W (nat_to_B n) (list_to_f l)) = list_to_fct l n.
```

(Finite Lists)



Given a function $[0,n[\to \mathbf{N}$, we can construct a function $\mathbb{N}\to\mathbb{N}$, by extending the function with zero, and using the natural isomorphism between $\mathbb{N}$ and $\mathbf{N}$. It will be denoted by *BN* on the diagram. The composition $fl \circ BN$ is the inverse of *LF*.

```
Definition back_to_nat f n:=
  cardinal_nat (Yo (inc (nat_to_B n) (source f))
    (W (nat_to_B n) f) card_zero).

Lemma back_to_nat_pr: forall f n, inc  (nat_to_B n) (source f) ->
  back_to_nat f n =  cardinal_nat (W  (nat_to_B n) f).
Lemma back_to_nat_pr1: forall f n k,
  source f = (interval_co_0a (nat_to_B k)) ->
  n < k -> back_to_nat f n =  cardinal_nat (W (nat_to_B n) f).
Lemma back_to_nat_pr2: forall (l: list nat) n,
  n < (length l) -> back_to_nat (list_to_f l) n = list_to_fct l n.

Lemma list_to_f_pr1: forall f n, is_function f -> target f = Bnat ->
  source f = (interval_co_0a (nat_to_B n)) ->
  f = list_to_f (fct_to_list (back_to_nat f) n).
Lemma list_to_f_pr2: forall l,
  fct_to_list (back_to_nat (list_to_f l)) (length l) = l.
```

Given a list L and a predicate P, we define P(L) to be true if every element of the list satisfies P. Given a predicate with two arguments, we say that the list satisfies the predicate whenever $\mathrm{P}(a,b)$ is true if a comes before b. This means that, if f is the function associated to the list, then $i<j$ implies $\mathrm{P}(f(i),f(j))$.

```
(*
  Fixpoint single_list_prop (A:Type) (L: list A) (q: A->Prop) :=
    match L with nil => True | a :: b => q a /\ single_list_prop b q end.
  Fixpoint double_list_prop (A:Type) (L: list A) (q: A->A->Prop) :=
    match L with nil => True
        | a :: b => single_list_prop b (q a) /\ double_list_prop b q
  end.
*)
```

These definitions changed in Version 2. The two-arguments case was unused, and removed; the single-argument case has been replaced by an inductive object.

```
Inductive list_prop (A:Type)  (q: A->Prop) : list A -> Prop :=
  | list_prop_nil: list_prop  q nil
  | list_prop_cons: forall (a:A)(l:list A),
    q a -> list_prop q l -> list_prop q (a::l).

Lemma list_prop1: forall A (q: A->Prop),  list_prop q nil.
Lemma list_prop2: forall A a b (q: A->Prop),
  q a -> (list_prop q b) = (list_prop q (a::b)).
Lemma list_prop3: forall A a b (q: A->Prop),
  ~ (q a) -> ~(list_prop q (a::b)).
Lemma list_prop_app: forall A a b (q: A->Prop),
  (list_prop q a) -> (list_prop q b)
  -> (list_prop q (a++b)).
Lemma list_prop_refine: forall A L (p q: A->Prop),
  (forall a, p a -> q a) ->  list_prop p L ->  list_prop q L.
Lemma list_prop_nth: forall  A (q: A->Prop) L u n,
  list_prop q L -> n < length L ->
  q (nth n L u).
```

The contraction $C_{fv}(L)$ of a list L is inductively defined by $C_{fv}(a::L) = f(a, C_{fv}(L))$, the value of the empty list being v. If $f(a,b) = b \cup \{a\}$, we call this the *range* of the list and denote it by $r(L)$. We write $L \subset E$ if $P_E(L)$ holds, where $P_E(x)$ is $x \in E$; this means that every element of the list belongs to E.

```
Fixpoint contraction (A B: Type) (L: list A) (f: A-> B->B) (v: B):B :=
  match L with | nil => v
    | a :: b => f a (contraction b f v) end.

Definition list_range l := contraction l (fun a b => tack_on b a) emptyset.
Definition list_subset L E := list_prop (fun x => inc x E) L.
```

The range of a list is the smallest set E such that $L \subset E$. We show $L \subset r(L)$ by induction. We have $r(L) \subset r(a::L)$. We then use the fact that if P implies Q, then P(L) implies Q(L), where P is $x \in r(L)$ and Q is $x \in r(a::L)$. We can now state: if $L \subset E$ is a list of length n, there is an associated function $[0,n[\rightarrow E$.

```
Lemma list_range_pr: forall L, list_subset L (list_range L).
Lemma list_range_pr1: forall L E, list_subset L E -> sub (list_range L) E.

Lemma list_to_fB_axioms: forall l E, list_subset l E ->
  transf_axioms (fun n => (list_to_fctB l (cardinal_nat n)))
  (interval_co_0a  (nat_to_B (length l))) E.
Lemma list_to_fB_function:  forall l E, list_subset l E ->
  is_function (list_to_fB l E).
Lemma list_to_fB_W: forall l E n, list_subset l E ->
  inc n (interval_co_0a (nat_to_B (length l))) ->
  W n (list_to_fB l E) = list_to_fctB l (cardinal_nat n).
Lemma list_to_fB_W1: forall  l E n, list_subset l E ->
  n < length l ->
  W (nat_to_B n) (list_to_fB l E) = list_to_fctB l n.
Lemma fct_to_rev: forall (A:Type) (f:nat->A) n,
  rev (fct_to_list f n) =  fct_to_list(fun i=> f(n-i-1)) n.
```

More properties of intervals.

```
Lemma partition_tack_on_intco: forall a, inc a Bnat ->
  partition_fam (variantLc (interval_co_0a a)
    (singleton a)) (interval_co_0a (succ a)).
Lemma interval_co_0a_restr: forall a f, inc a Bnat ->
  (L (interval_co Bnat_order card_zero a) f
    = (restr (L (interval_co Bnat_order card_zero (succ a)) f)
      (interval_co_0a a))).
```

Let L_1 and L_2 be two lists, $L = L_1 ++ a::L_2$, Let G_1 and G be the functions associated to L_1 and L. If L_1 is a list of size n, then $G(n) = a$, and G and G_1 agree on $[0, n-1]$. In fact, G_1 is the restriction of G to the interval $[0, n[$, and if L_2 is empty, then G is the function obtained from G_1 by adding the relation $G(n) = a$.

```
Lemma length_app1: forall (A:Type) (a:A) l l',
  length l < length (l ++ a :: l').
Lemma length_app2:  forall (A:Type) (a:A) l ,
  nat_to_B (length (l++a::nil)) = succ (nat_to_B (length l)).

Lemma list_to_f_cons0: forall a l l',
  W (nat_to_B  (length l)) (list_to_f (l++ a:: l')) = nat_to_B a.
Lemma list_to_f_cons1: forall a l l' n, n < length l ->
  W (nat_to_B n) (list_to_f (l ++ a :: l')) =  W (nat_to_B n) (list_to_f l).
Lemma list_to_f_cons2: forall a l l',
  list_to_f l = restriction (list_to_f (l++ a :: l'))
              (interval_co_0a (nat_to_B (length l))).
Lemma list_to_f_cons3: forall a l,
  list_to_f (l++a::nil) = tack_on_f (list_to_f l)
  (nat_to_B (length l))  (nat_to_B a).

Lemma list_subset_cons: forall a l l' E,
  list_subset l E -> inc a E -> list_subset l' E ->
  list_subset (l'++a::l) E.
Lemma list_to_f_consB0: forall a l l' E,
  list_subset l E -> inc a E -> list_subset l' E ->
  W (nat_to_B  (length l)) (list_to_fB (l++ a:: l') E) = a.
Lemma list_to_f_consB1: forall a l l' n E, n< length l ->
  list_subset l E -> inc a E ->  list_subset l' E ->
  W (nat_to_B n) (list_to_fB (l++ a :: l') E) = W (nat_to_B n) (list_to_fB l E).

Lemma list_to_f_consB0: forall a l E, list_subset l E -> inc a E ->
  W (nat_to_B  (length l)) (list_to_fB (a :: l) E) = a.
Lemma list_to_f_consB1: forall a l n E, n< length l ->
  list_subset l E -> inc a E ->
  W (nat_to_B n) (list_to_fB (a :: l) E) =  W (nat_to_B n) (list_to_fB l E).
Lemma list_to_f_consB2: forall a l l' E,
  list_subset l E -> inc a E ->  list_subset l' E ->
  list_to_fB l E = restriction (list_to_fB (l++ a :: l') E)
              (interval_co_0a (nat_to_B (length l))).
Lemma list_to_f_consB3: forall a l E,
  list_subset l E -> inc a E ->
  list_to_fB (l++a::nil) E
  = tack_on_f  (list_to_fB l E) (nat_to_B (length l))  a.
```

We denote by *LFB* the variant of *lf* that converts a list $L \subset E$ into a function $I \to E$. The source of this function is an interval $[0, n[$; we shalll call this an *iid* function. We denote by

FLB the variant of *fl* which is the inverse of *LFB*, i.e. $LFB_{\mathrm{E}}(FLB(f)) = f$ and $FLB(LFB_{\mathrm{E}}(\mathrm{L})) = \mathrm{L}$, whenever f is a function whose source is $[0, n[$ and is target is E, and whenever $\mathrm{L} \subset \mathrm{E}$.

```
Definition fct_to_listB1 f n:=
  fct_to_list (fun n => W (nat_to_B n) f) n.
Definition fct_to_listB f := fct_to_listB1 f (cardinal_nat (source f)).
Definition iid_function f :=
  is_function f & exists n, source f = interval_co_0a (nat_to_B n).

Lemma list_to_fB_pr:  forall l E, list_subset l E ->
  iid_function (list_to_fB l E).
Lemma fct_to_list_lengthB :  forall f,  iid_function f ->
  nat_to_B (length (fct_to_listB f)) = cardinal (source f).
Lemma fct_to_listB_pr0: forall f i,
  iid_function f -> i< cardinal_nat (source f) ->
  list_to_fctB (fct_to_listB f) i = W (nat_to_B i) f.

Lemma fct_to_listB_pr1: forall l E, list_subset l E ->
  fct_to_listB(list_to_fB l E) = l.
Lemma fct_to_listB_pr2: forall f, iid_function f ->
   list_subset (fct_to_listB f) (target f).
Lemma fct_to_listB_pr3: forall f, iid_function f ->
  list_to_fB  (fct_to_listB f) (target f) = f.
```

12.4.2 Contracting lists

We show here the following. Assume that $f(n)$ is a cardinal for all n. Let $\mathrm{F}(n)$ be the cardinal sum of the family $i \mapsto f(i)$ on $[0, n-1]$. Then $\mathrm{F}(n+1) = f(n) + \mathrm{F}(n)$, and there is a similar relation for the product. The same formula holds if $\mathrm{F}(n+1)$ is the cardinal sum of the graph of the function f and $\mathrm{F}(n)$ is cardinal sum of the graph of the restriction of f to $[0, n-1]$. We apply this to the case where f is *LF(L++a::nil)*; its restriction is *LF(L)*, and $f(n) = a$.

```
Lemma induction_on_sum: forall a f,  inc a Bnat ->
  (forall a, inc a Bnat -> is_cardinal (f a)) ->
  let iter := fun n=> cardinal_sum (L (interval_co_0a n)f)
    in card_plus (iter a) (f a) = (iter (succ a)).
Lemma induction_on_prod: forall a f,  inc a Bnat ->
  (forall a, inc a Bnat -> is_cardinal (f a)) ->
  let iter := fun n=> cardinal_prod (L (interval_co_0a n ) f)
    in card_mult (iter a) (f a) = (iter (succ a)).
Lemma induction_on_sum1: forall f n,
  is_function f -> source f = interval_co_0a (succ n) -> inc n Bnat ->
  (forall a, inc a (source f) -> is_cardinal (W a f)) ->
  card_plus (cardinal_sum (graph (restriction f (interval_co_0a n))))
  (W n f) = cardinal_sum (graph f).
Lemma induction_on_prod1: forall f n,
  is_function f -> source f = interval_co_0a (succ n) -> inc n Bnat ->
  (forall a, inc a (source f) -> is_cardinal (W a f)) ->
  card_mult (cardinal_prod (graph (restriction f (interval_co_0a n))))
  (W n f) = cardinal_prod (graph f).
```

Denote by S(L) the cardinal sum of the family *LF(L)*. The induction principle says $\mathrm{S}(\mathrm{L}{+}{+}\, a{::}nil) = \mathrm{S}(\mathrm{L}) + a$. By associativity we get $\mathrm{S}(\mathrm{L}' + +\mathrm{L}) = \mathrm{S}(\mathrm{L}') + \mathrm{S}(\mathrm{L})$. We have $\mathrm{S}(nil) = 0$ and $\mathrm{S}(a{::}nil) = a$. If we take $\mathrm{L}' = a{::}nil$, the associativity formula gives $\mathrm{S}(a{::}\mathrm{L}) = a + \mathrm{S}(\mathrm{L})$. **Note**: in

version 2, we changed the ordering of the elements of the list. This changes the properties of S; we proved the previous formula by using commutativity (rather than associativity).

```
Lemma induction_on_sum2: forall a l,
  card_plus (cardinal_sum (graph (list_to_f l))) (nat_to_B a)
  = cardinal_sum (graph (list_to_f (l++a::nil))).
Lemma induction_on_prod2: forall a l,
  card_mult (cardinal_prod (graph (list_to_f l))) (nat_to_B a)
  = cardinal_prod (graph (list_to_f (l++a::nil))).

Lemma induction_on_sum0:
  cardinal_sum (graph (list_to_f nil)) = card_zero.
Lemma induction_on_sum5: forall a,
  cardinal_sum (graph (list_to_f (a::nil))) = nat_to_B a.
Lemma induction_on_prod0:
  cardinal_prod (graph (list_to_f nil)) = card_one.
Lemma induction_on_prod5: forall a,
  cardinal_prod (graph (list_to_f (a::nil))) = nat_to_B a.

Lemma induction_on_sum4: forall l l',
  card_plus (cardinal_sum (graph (list_to_f l)))
  (cardinal_sum (graph (list_to_f l')))
  = cardinal_sum (graph (list_to_f (l++l'))).
Lemma induction_on_prod4: forall l l',
  card_mult (cardinal_prod (graph (list_to_f l)))
  (cardinal_prod (graph (list_to_f l')))
  = cardinal_prod (graph (list_to_f (l++l'))).
```

We define here the sum and product of a list of integers as a contraction. We shall denote this by $\Sigma(\mathrm{L})$ and $\Pi(\mathrm{L})$. This operation is related to the previous one by $\mathscr{N}(\Sigma(\mathrm{L})) = \sum LF(\mathrm{L})$ and $\mathscr{N}(\Pi(\mathrm{L})) = \prod LF(\mathrm{L})$.

```
Definition list_sum l := contraction (rev l) plus 0.
Definition list_prod l := contraction (rev l) mult 1.

Lemma list_sum_pr: forall l,
  nat_to_B (list_sum l) = cardinal_sum (graph (list_to_f l)).
Lemma list_prod_pr: forall l,
  nat_to_B (list_prod l) = cardinal_prod (graph (list_to_f l)).
```

If we denote by $a{+}{+}b$ the concatenation of two lists, then $\mathrm{C}_{fv}(a{+}{+}b) = f(\mathrm{C}_{fv}(a), \mathrm{C}_{fv}(b))$ provided that the result is true for the empty list, i.e., $f(v,b) = b$ for all b, and if f is associative. As a consequence $\Sigma(a{+}{+}b) = \Sigma(a) + \Sigma(b)$ and $\Pi(a{+}{+}b) = \Pi(a)\cdot\Pi(b)$. This is a general property of contractions of an associative function f

Denote by $\Sigma'_n(f)$ the expression $\Sigma(fl(f,n))$. This is the sum of the list of the values $f(i)$ for $i < n$. If we denote by $f_1 + f_2$ the function $i \mapsto f_1(i) + f_2(i)$ then we have $\Sigma'_n f + \Sigma'_n g = \Sigma'_n(f+g)$. There are similar formulas for the product. We have two induction formulas; the trivial one is $\Sigma'_{n+1}(f) = f(n) + \Sigma'_n(f)$; the non-trivial one is $\Sigma'_{n+1}(f) = f(0) + \Sigma'_n(f \circ \mathrm{S})$, where $(f \circ \mathrm{S})(i) = f(i+1)$.

```
Lemma contraction_assoc: forall  (A :Type) (L1 L2: list A)
  (f: A-> A->A) (v: A),
  (forall a b c, f a (f b c) = f (f a b) c) ->
  (forall a, f v a = a) ->
```

```
  (contraction (L1++L2) f v) = f (contraction L1 f v)(contraction L2 f v).


Lemma list_sum_single: forall a, list_sum (a::nil) = a.
Lemma list_prod_single: forall a, list_prod (a::nil) = a.
Lemma list_sum_app: forall a b, list_sum (a++b) = (list_sum a)+ (list_sum b).
Lemma list_sum_cons: forall a b, list_sum (a::b) = a + (list_sum b).
Lemma list_sum_consr: forall a b, list_sum (a++(b::nil)) = (list_sum a)+ b.
Lemma list_prod_app: forall a b,
  list_prod (a++b) = (list_prod a)* (list_prod b) .
Lemma list_prod_cons: forall a b, list_prod (a::b) = a*(list_prod b).
Lemma list_prod_consr: forall a b, list_prod (a++(b::nil)) = (list_prod a)* b.


Definition fct_sum f n:= list_sum (fct_to_list f n).
Definition fct_prod f n:= list_prod(fct_to_list f n).

Lemma fct_sum0: forall f, fct_sum f 0 = 0.
Lemma fct_prod0: forall f, fct_prod f 0 = 1.
Lemma fct_sum_rec: forall f n, fct_sum f (S n) = (fct_sum f n) + (f n).
Lemma fct_prod_rec: forall f n, fct_prod f (S n) = (fct_prod f n) * (f n).
Lemma fct_sum_rec1: forall f n,
  fct_sum f (S n) = (f 0) + (fct_sum (fun i=> f (S i)) n).
Lemma fct_prod_rec1: forall f n,
  fct_prod f (S n) = (f 0) * (fct_prod (fun i=> f (S i)) n).
Lemma fct_sum_plus: forall f g n,
  (fct_sum f n) +  (fct_sum g n) = fct_sum (fun i=> (f i) + (g i)) n.
Lemma fct_prod_mult: forall f g n,
  (fct_prod f n) *  (fct_prod  g n) =fct_prod (fun i=> (f i) * (g i)) n.
```

We show here some trivial results. The sum of a constant function is the product, and the sum is unchanged if we replace the list by its reverse. A bit more complicated: the reverse of the list associated to a function f is the list associated to $i \mapsto f(n-i-1)$.

```
Lemma fct_sum_const: forall n m, fct_sum (fun _ => m) n = n *m.
Lemma fct_prod_const: forall n m, fct_prod (fun _ => m) n = pow m n.
Lemma list_sum_rev: forall l, list_sum l = list_sum (rev l).
Lemma list_prod_rev: forall l, list_prod l = list_prod (rev l).
Lemma fct_sum_rev: forall f n,
  fct_sum f n =  fct_sum (fun i=> f(n-i-1)) n.
Lemma fct_prod_rev: forall f n,
  fct_prod f n =  fct_prod (fun i=> f(n-i-1)) n.
Lemma fct_to_rev: forall (A:Type) (f:nat->A) n,
  rev (fct_to_list f n) =  fct_to_list(fun i=> f(n-i-1)) n.
```

We consider here the inverse of *BN*: if f is a function of type *nat*→*nat*, we construct a function $[0,n[\to \mathbf{N}$. It is the composition of *fl* and *LF*. We shall denote it by *NB*. The first theorem is a statement about *NB*, the two others are statements about the graph of *NB*. The third theorem is deduced from the second by applying $[0,n+1[= [0,n]$.

```
Lemma l_to_fct: forall f n,
  BL (fun p => nat_to_B(f (cardinal_nat p))) (interval_co_0a (nat_to_B n))
  Bnat = list_to_f (fct_to_list f n).
Lemma l_to_fct1: forall f n,
  L (interval_co_0a (nat_to_B n)) (fun p => nat_to_B(f (cardinal_nat p)))
```

```
  = graph (list_to_f (fct_to_list f n)).
Lemma l_to_fct2: forall f n,
  L (interval_Bnat card_zero (nat_to_B n))
  (fun p => nat_to_B(f (cardinal_nat p)))
  = graph (list_to_f (fct_to_list f (S n))).
```

12.4.3 Iterated functions

Note: all useful results of this section have been moved to section 12.4.2. The remaining trivial results are given without comment.

```
Definition function_on_nat f :=
  fun m =>nat_to_B (f (cardinal_nat m)).

Lemma inc_function_on_nat_Bnat : forall f n,
  inc (function_on_nat f n) Bnat.
Lemma function_on_nat_pr : forall f n,
  cardinal_nat(function_on_nat f n) = f (cardinal_nat n).
Lemma function_on_nat_pr1 : forall f n,
  function_on_nat f (nat_to_B n) = nat_to_B (f n).
```

12.4.4 Factorial

In a previous version, we defined the factorial function as shown below. This definition is equivalrnt to the one provided by ssrnat.

```
Fixpoint factorial (n:nat) : nat :=
  match n with
  | 0 => 1
  | S p => (factorial p) *  S p
  end.

Lemma factorial0: factorial 0 = 1.
Lemma factorial1: factorial 1 = 1.
Lemma factorial2: factorial 2 = 2.

Lemma factorial_succ: forall n,  factorial (S n) = (factorial n) * (S n).
Lemma factorial_nonzero: forall n, 0 <> factorial n.

Lemma factorial_prop: forall f, f 0 = 1 ->
  (forall n, f (S n) =  (f n) * (S n)) ->
   forall x, f x = factorial x.
Lemma factorial_prop1: forall n, factorial n = fct_prod S n.
```

We prove here: if J $\subset$ I, the product $\prod f_i$ restricted to J divides the product $\prod f_i$ restricted to I. We used this result to show that $n!$ divides $m!$. This requires 44 lines of proof, but proving by induction on c that $b!$ divides $(b+c)!$ requires only 3 lines (for that vasr of *nat*, ten lines in the case of *Bnat*).

```
Lemma divides_restriction_product: forall f x, fgraph f ->
     (forall i, inc i (domain f) -> is_finite_c (V i f)) ->
     (forall i, inc i (domain f) -> (V i f) <> card_zero) ->
     is_finite_set (domain f) -> sub x (domain f) ->
```

```
     BNdivides (cardinal_prod (restr f x)) (cardinal_prod f).

Lemma quotient_of_factorials: forall a b, b <= a ->
  Ndivides (factorial b) (factorial a).
Lemma quotient_of_factorials1: forall a b, b <= a ->
  Ndivides (factorial (a - b)) (factorial a).
Lemma tack_on_nat: forall a b, is_finite_set (tack_on a b) ->
  ~ (inc b a) -> cardinal_nat (tack_on a b) = S (cardinal_nat a).
```

12.4.5 The binomial coefficient

In the previous version, the binomial function was defined by induced as follows.

```
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.
```

12.5 Removed theorems

The lemmas and definition shown here existed in previous version, but have been withdrawn.

A correspondence $\Gamma = (G, E, E)$, whose graph G is an order on E, is also called an order by Bourbaki (this definition is in fact never used).

```
Definition order_c r :=
  is_correspondence r & source r = target r & source r = substrate (graph r)
  & order (graph r).
Theorem order_cor_pr: forall f,
  is_correspondence f ->
  order_c f =
  (source f = target f & source f = (domain (graph f)) &
    compose_graph (graph f)(graph f) = graph f &
    intersection2 (graph f) (opposite_order (graph f))
    = diagonal (substrate (graph f))).
```

Here is is the original definition of a product order. One can notice that f is uniquely defined by g. This iargument has beeen removed in the new version.

```
Definition product_order_r (f g:Set): EEP :=
  fun x x' =>
   inc x (productb f) & inc x' (productb f) &
   forall i, inc i (domain f) -> gle (V i g) (V i x)(V i x').

Definition product_order f g:=
  graph_on (product_order_r f g)(productb f).


Definition axioms_product_order f g:=
```

```
  fgraph f & fgraph g & domain f = domain g &
  (forall i, inc i (domain f) -> order (V i g)) &
  (forall i, inc i (domain f) -> substrate (V i g) = V i f).


Lemma order_product_order: forall f g,
  axioms_order_product f g -> order (product_order f g).
Lemma related_product_order: forall f g x x',
  axioms_product_order f g ->
  related(product_order f g) x x' =
  (inc x (productb f) & inc x' (productb f) &
  forall i, inc i (domain f) -> related (V i g) (V i x)(V i x')).
Lemma substrate_product_order: forall f g,
  axioms_product_order f g -> substrate(product_order f g) = productb f.
Lemma product_order_def: forall f g,  axioms_product_order f g ->
  image_by_fun (prod_of_products_canon f f)(product_order f g)
   = (productb g).  (* 36 *)
```

These are the original definitions of the lexicograph orced

```
Definition lexicographic_order_r (r f g:Set): EEP :=
  fun x x' =>
    inc x (productb f) & inc x' (productb f) &
    forall j, least_element (induced_order r (Zo (domain f)
      (fun i =>  V i x <> V i x'))) j -> glt (V j g) (V j x)(V j x').

Definition lexicographic_order_axioms r f g:=
  worder r & substrate r = domain f &
  fgraph f & fgraph g & domain f = domain g &
  (forall i, inc i (domain f) -> order (V i g)) &
  (forall i, inc i (domain f) -> substrate (V i g) = V i f).


Definition graph_order_r(x y g:Set): EEP :=
  fun z z' =>
    inc z (set_of_gfunctions x y) & inc z' (set_of_gfunctions x y) &
    forall i, inc i x-> related g (V i z)(V i z').
Definition graph_order x y g :=
  graph_on(graph_order_r x y g) (set_of_gfunctions x y).
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).
```

Given two sets A and B, two distinct elements α and β, if I is the set that contains α and β, there is a family $(X_\iota)_{\iota\in I}$ such that such that $A = X_\alpha$ and $B = X_\beta$. This family is *Lvariant*. We shall denote it by $X_{\alpha\beta}(A,B)$. We show here uniqueness of the family.

```
Lemma two_terms_bij1: forall a b x y f,
  y <> x -> fgraph f -> domain f = doubleton x y -> V x f = a -> V y f = b ->
  range f  =  doubleton a b -> f = Lvariant x y a b.
```

Here are some trivial lemmas.

```
Lemma source_pfs:forall y x,
  source (partition_fun_of_set y x) = y.
```

```
Lemma target_pfs: forall y x,
  target (partition_fun_of_set y x) = powerset x.
Lemma source_graph_of_function: forall x y,
  source (graph_of_function x y) = set_of_sub_functions x y.
Lemma target_graph_of_function: forall x y,
  target (graph_of_function x y) =   (set_of_graphs x y).
Lemma sup_interval_co_0a: forall n, inc n Bnat ->
  supremum Bnat_order (interval_co_0a (succ n)) = n.
```

12.5.1 Definition of a function by induction

We explain here the initial implementation of section 7.2, more precisely the case when a function f is defined by (IND0), i.e., $f(0) = a$ and $f(n+1) = h(n, f(n))$ for $n \in \mathbf{N}$, or variants of this formulation.

In Version 1 we had the following two definitions (compare with *induction_defined0_set* and *induction_defined1_set*). They are of the form *choose IND0* and *choose IND1'*. We have two theorems saying that these objects satisfy (IND0) and (IND1') respectively, and two others stating existence and uniqueness of (IND0), and existence of (IND1'). Together with these four theorems, we show a variant of *integer_induction_stable* and the Bourbaki variant of (IND0).

```
Definition induction_defined1 E h a:= choosef(fun f=>
  is_function f & source f = Bnat & target f = E & W card_zero f = a &
  forall n,  inc n Bnat -> W (succ n) f = h n (W n f)).
Definition induction_defined2 E h a p:= choosef(fun f=>
  is_function f & source f = Bnat & target f = E & W card_zero f = a &
  forall n,  cardinal_lt n p -> W (succ n) f = h n (W n f)).

Lemma integer_induction_stable: forall E g  a,
  inc a E -> is_function g -> source g = E -> target g = E ->
  sub (target (induction_defined g a)) E.
Lemma induction_with_var: forall E h a,
  is_function h -> source h = product Bnat E -> target h = E -> inc a E ->
  exists_unique (fun f=>  is_function f & source f = Bnat & target f= E &
    W card_zero f = a
    &  forall n, inc n Bnat -> W (succ n) f = W (J n (W n f)) h).


Lemma induction_with_var1: forall E h a,
  (forall n x, inc n Bnat -> inc x E -> inc (h n x) E) -> inc a E ->
  exists_unique (fun f=>  is_function f & source f = Bnat & target f= E &
    W card_zero f = a
    &  forall n, inc n Bnat -> W (succ n) f = h n (W n f)).

Lemma induction_with_var2: forall E h a p,
  (forall n x, inc n Bnat -> inc x E -> cardinal_lt n p -> inc (h n x) E)
  -> inc a E -> inc p Bnat ->
  exists f,  is_function f & source f = Bnat & target f= E &
    W card_zero f = a
    &  forall n, cardinal_lt n p -> W (succ n) f = h n (W n f).
Lemma induction_defined_pr2: forall E h a p,
  (forall n x, inc n Bnat -> inc x E -> cardinal_lt n p -> inc (h n x) E)
  -> inc a E -> inc p Bnat ->
  let f :=  induction_defined2 E h a p in is_function f &
```

```
    source f = Bnat &  target f = E & W card_zero f = a &
    forall n,  cardinal_lt n p -> W (succ n) f = h n (W n f).

Lemma induction_defined_pr1: forall E h a,
  (forall n x, inc n Bnat -> inc x E -> inc (h n x) E)
  -> inc a E ->
  let f :=  induction_defined1 E h a in is_function f &
    source f = Bnat &  target f = E & W card_zero f = a &
    forall n,  inc n Bnat -> W (succ n) f = h n (W n f).
```

12.5.2 Intervals

We give here the original proof that the intersection of two intervals is an interval.

Let's say that an interval is of type B if it is bounded, of type L' if it is left unbounded, of type R' if it is right unbounded, of type U if it is $]\leftarrow,\rightarrow[$. Let's say that an interval is of type L if it is of type L' or U, of type R if it is of type R' or U.

Let's write $L' \cap L' = L'$ as a short-hand for: the intersection of two intervals of type L' is an interval of type L', this is lemma *intersection_i3* and will be explained later. If we consider the reverse ordering, an interval remains an interval, but the lemmas shown here are more precise (they say for instance that the opposite of L' is R').

```
Lemma opposite_interval_cc: forall r a b,
  order r -> interval_cc r a b = interval_cc (opposite_order r) b a.
Lemma opposite_interval_oo: forall r a b,
  order r -> interval_oo r a b = interval_oo (opposite_order r) b a.
Lemma opposite_interval_oc: forall r a b,
  order r -> interval_oc r a b = interval_co (opposite_order r) b a.
Lemma opposite_interval_co: forall r a b,
  order r -> interval_co r a b = interval_oc (opposite_order r) b a.
Lemma opposite_bounded_interval: forall r x, order r ->
  is_bounded_interval r x -> is_bounded_interval (opposite_order r) x.
Lemma opposite_interval_ou: forall r a,
  order r -> interval_ou r a = interval_uo (opposite_order r) a.
Lemma opposite_interval_cu: forall r a,
  order r -> interval_cu r a = interval_uc (opposite_order r) a.
Lemma opposite_interval_uu: forall r,
  order r -> interval_uu r = interval_uu (opposite_order r).
Lemma opposite_interval_uo: forall r a,
  order r -> interval_uo r a = interval_ou (opposite_order r) a.
Lemma opposite_interval_uc: forall r a,
  order r -> interval_uc r a = interval_cu (opposite_order r) a.
Lemma opposite_unbounded_interval: forall r x, order r ->
  is_unbounded_interval r x -> is_unbounded_interval (opposite_order r) x.
Lemma opposite_interval: forall r x, order r ->
  is_interval r x -> is_interval (opposite_order r) x.
```

There are 9 types of intervals, thus 81 cases to consider. The case of intervals of type U is trivial, so that the number of cases is really 64. The new proof replaces bounded intervals by unbounded intervals, so that there are only 16 cases to consider. Let's start with these ones.

Case $L' \cap R' = R' \cap L' = B$. Consider $X =]\leftarrow, x[\cap]y, \rightarrow[$. If the intersection is non-empty, there is a such that $y \leq a \leq x$, thus $y \leq x$, and $X =]y, x[$. Otherwise $X =]x, x[$. Similarly, if we consider intervals that contain the end-point x or y, the intersection is empty, or an interval that contains the end-point x or y.

Case $L' \cap L' = L'$. Consider $X(b) =]\leftarrow, b[$ and $Y(b) =]\leftarrow, b]$. Let $d = \inf(b, c)$. We have $X(d) \subset X(b) \cap X(c) \subset Y(d)$. If d is in the intersection, then the intersection is $Y(d)$, otherwise it is $X(d)$. Replacing one of $X(b)$ or $X(c)$ by $Y(b)$ or $Y(c)$ is similar. Note: the intersection of two closed intervals is empty or closed, and the intersection of two open intervals is open, only when the order is total.

Using the reverse order, it follows $R' \cap R' = R'$, and this covers all unbounded intervals. All remaining cases are similar. We must consider what happens on the left, and what happens on the right. The big part of the proof (300 lines) consists in showing that the intersection of two bounded intervals is a bounded interval. This does not follow directly from our new theorem, but is easy (for instance, if X is a subinterval of $[a, b]$, of the form $]\leftarrow, x[$, it is $[a, x[$).

```
Lemma intersection_interval1: forall r x y,
  lattice r ->is_closed_interval r x -> is_closed_interval r y  ->
    is_bounded_interval r (intersection2 x y).
Lemma intersection_interval2: forall r x y,
  lattice r ->is_open_interval r x -> is_open_interval r y  ->
  is_bounded_interval r (intersection2 x y).     (* 39 *)
Lemma intersection_interval3: forall r a b a' b',
  lattice r -> inc a (substrate r) -> inc a' (substrate r)  ->
  inc b (substrate r)  -> inc b' (substrate r) ->
  is_bounded_interval r
      (intersection2(interval_co r a b)(interval_co r a' b')) (* 19 *)
Lemma intersection_interval4: forall r a b a' b',
  lattice r -> inc a (substrate r) -> inc a' (substrate r)  ->
  inc b (substrate r)  -> inc b' (substrate r) ->
  is_bounded_interval r (intersection2(interval_oc r a b)(interval_oc r a' b')).
Lemma intersection_interval5: forall r a b a' b',
  lattice r -> inc a (substrate r) -> inc a' (substrate r)  ->
  inc b (substrate r)  -> inc b' (substrate r) ->
  is_bounded_interval r
     (intersection2(interval_co r a b)(interval_oc r a' b')). (* 30 *)
Lemma intersection_interval6: forall r x y,
  lattice r ->is_semi_open_interval r x -> is_semi_open_interval r y  ->
    is_bounded_interval r (intersection2 x y).
Lemma intersection_interval7: forall r a b a' b',
  lattice r -> inc a (substrate r) -> inc a' (substrate r)  ->
  inc b (substrate r)  -> inc b' (substrate r) ->
  is_bounded_interval r
      (intersection2(interval_cc r a b)(interval_oo r a' b')). (* 30 *)
Lemma intersection_interval8: forall r a b a' b',
  lattice r -> inc a (substrate r) -> inc a' (substrate r)  ->
  inc b (substrate r)  -> inc b' (substrate r) ->
  is_bounded_interval r
      (intersection2(interval_cc r a b)(interval_oc r a' b')). (* 18 *)
Lemma intersection_interval9: forall r a b a' b',
  lattice r -> inc a (substrate r) -> inc a' (substrate r)  ->
  inc b (substrate r)  -> inc b' (substrate r) ->
  is_bounded_interval r
      (intersection2(interval_oo r a b)(interval_oc r a' b')). (* 34 *)
Lemma intersection_interval10: forall r a b a' b',
  lattice r -> inc a (substrate r) -> inc a' (substrate r)  ->
  inc b (substrate r)  -> inc b' (substrate r) ->
  is_bounded_interval r (intersection2(interval_oo r a b)(interval_co r a' b')).
Lemma intersection_interval11: forall r a b a' b',
  lattice r -> inc a (substrate r) -> inc a' (substrate r)  ->
```

```
  inc b (substrate r)  -> inc b' (substrate r) ->
  is_bounded_interval r (intersection2(interval_cc r a b)(interval_co r a' b')).
Lemma intersection_interval12: forall r x y, lattice r ->
  is_bounded_interval r x -> is_bounded_interval r y ->
  is_bounded_interval r (intersection2 x y).
```

We consider now the case of unbounded intervals.

```
Lemma intersection_interval13: forall r x,
  is_interval r x -> intersection2 x (interval_uu r) = x.
Lemma intersection_interval14: forall r x y, lattice r ->
  is_left_unbounded_interval r x -> is_left_unbounded_interval r y ->
  is_left_unbounded_interval r (intersection2 x y).  (* 18 *)
Lemma intersection_interval15: forall r x y, lattice r ->
  is_right_unbounded_interval r x -> is_right_unbounded_interval r y ->
  is_right_unbounded_interval r (intersection2 x y).
Lemma intersection_interval16: forall r x y, lattice r ->
  is_left_unbounded_interval r x -> is_right_unbounded_interval r y ->
  is_bounded_interval r (intersection2 x y). (* 19 *)
Lemma intersection_interval17: forall r x y, lattice r ->
  is_unbounded_interval r x -> is_unbounded_interval r y ->
  is_interval r (intersection2 x y).
Lemma intersection_interval18: forall r x y, lattice r ->
  is_left_unbounded_interval r x -> is_bounded_interval r y ->
  is_bounded_interval r (intersection2 x y). (* 97 *)
Lemma intersection_interval19: forall r x y, lattice r ->
  is_right_unbounded_interval r x -> is_bounded_interval r y ->
  is_bounded_interval r (intersection2 x y).
Lemma intersection_interval20: forall r x y, lattice r ->
  is_unbounded_interval r x -> is_bounded_interval r y ->
  is_bounded_interval r (intersection2 x y).
```

The result is now obvious.

```
Theorem intersection_interval: forall r x y,
  lattice r -> is_interval r x -> is_interval r y ->
  is_interval r (intersection2 x y).
```

Index

Bibliography

[1] Jon Barwise and Lawrence Moss. *Vicious Circles.* Number 60. CLSI Publications, 1996.

[2] Yves Bertod and Pierre Castéran. *Interactive Theorem Proving and Program Development.* Springer, 2004.

[3] N. Bourbaki. *Elements of Mathematics, Theory of Sets.* Springer, 1968.

[4] N. Bourbaki. *Éléments de mathématiques, Théorie des ensembles.* Diffusion CCLS, 1970.

[5] José Grimm. Implementation of Bourbaki's Elements of Mathematics in Coq: Part One, Theory of Sets. Research Report RR-6999, INRIA, 2009.

[6] Douglas Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid.* Basic Books, 1979.

[7] Jean-Louis Krivine. *Théorie axiomatique des ensembles.* Presses Universitaires de France, 1972.

[8] The Coq Development Team. The Coq reference manual. http://coq.inria.fr.

[9] Jean von Heijenoort, editor. *From Frege to Gödel.* Harvard University Press, 1967.

Contents

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399