

# Cofinality and the Delta System Lemma

Pedro Sánchez Terraf<sup>\*†</sup>

March 17, 2025

## Abstract

We formalize the basic results on cofinality of linearly ordered sets and ordinals and Šanin's Lemma for uncountable families of finite sets. We work in the set theory framework of Isabelle/ZF, using the Axiom of Choice as needed.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Library of basic ZF results</b>	<b>2</b>
2.1	Some minimal arithmetic/ordinal stuff . . . . .	3
2.2	Manipulation of function spaces . . . . .	5
2.3	Finite sets . . . . .	8
2.4	Basic results on equipollence, cardinality and related concepts	11
2.5	Morphisms of binary relations . . . . .	16
2.6	Alephs are infinite cardinals . . . . .	22
2.7	Transfinite recursive constructions . . . . .	24
<b>3</b>	<b>Cofinality</b>	<b>24</b>
3.1	Basic results and definitions . . . . .	24
3.2	The factorization lemma . . . . .	32
3.3	Classical results on cofinalities . . . . .	41
<b>4</b>	<b>Cardinal Arithmetic under Choice</b>	<b>49</b>
4.1	Results on cardinal exponentiation . . . . .	49
4.2	Miscellaneous . . . . .	51
4.3	Countable and uncountable sets . . . . .	54

<sup>\*</sup>Universidad Nacional de Córdoba. Facultad de Matemática, Astronomía, Física y Computación.

<sup>†</sup>Centro de Investigación y Estudios de Matemática (CIEM-FaMAF), Conicet. Córdoba. Argentina. Supported by Secyt-UNC project 33620180100465CB.  
<https://cs.famaf.unc.edu.ar/~pedro/>

4.4	Results on Alephs . . . . .	55
4.5	Applications of transfinite recursive constructions . . . . .	58
4.6	König’s Theorem . . . . .	64
<b>5</b>	<b>The Delta System Lemma</b>	<b>66</b>
5.1	Application to Cohen posets . . . . .	71

## 1 Introduction

The session we present gathers very basic results built on the set theory formalization of Isabelle/ZF [7]. In a sense, some of the material formalized here corresponds to a natural continuation of that work. This is even clearer after perusing Section 2, where notions like cardinal exponentiation are first defined, together with various lemmas that do not depend on the Axiom of Choice (*AC*); the same holds for the basic theory of cofinality of ordinals, which is developed in Section 3. In Section 4, (un)countability is defined and several results proved, now using *AC* freely; the latter is also needed to prove König’s Theorem on cofinality of cardinal exponentiation. The simplest infinitary version of the Delta System Lemma (DSL, also known as the “Sunflower Lemma”) due to Šanin is proved in Section 5, and it is applied to prove that Cohen posets satisfy the *countable chain condition*.

A greater part of this development was motivated by a joint project on the formalization of the ctm approach to forcing [1] by Gunther, Pagano, Steinberg, and the author. Indeed, most of the results presented here are required for the development of forcing. As it turns out, the material as formalized presently is not imported as a whole by the forcing formalization [3, 2], since the latter requires relativized versions of both the concepts and the proofs.

## 2 Library of basic *ZF* results

```
theory ZF_Library
imports
ZF_Constructible.Normal

begin
```

This theory gathers basic “combinatorial” results that can be proved in *ZF* (that is, without using the Axiom of Choice *AC*).

We begin by setting up math-friendly notation.

```
no_notation oadd (infixl <++> 65)
no_notation sum (infixr <+> 65)
notation oadd (infixl <+> 65)
```

```

notation nat (< $\omega$ >)
notation csucc (< $\_+$ > [90])
no_notation Aleph (< $\aleph$ > [90] 90)
notation Aleph (< $\aleph$ >)
syntax _ge :: [i,i]  $\Rightarrow$  o (infixl < $\geq$ > 50)
translations  $x \geq y \rightarrow y \leq x$ 

2.1 Some minimal arithmetic/ordinal stuff

lemma Un_leD1 :  $i \cup j \leq k \Rightarrow Ord(i) \Rightarrow Ord(j) \Rightarrow Ord(k) \Rightarrow i \leq k$ 
  by (rule Un_least_lt_iff[THEN iffD1[THEN conjunct1]],simp_all)

lemma Un_leD2 :  $i \cup j \leq k \Rightarrow Ord(i) \Rightarrow Ord(j) \Rightarrow Ord(k) \Rightarrow j \leq k$ 
  by (rule Un_least_lt_iff[THEN iffD1[THEN conjunct2]],simp_all)

lemma Un_memD1:  $i \cup j \in k \Rightarrow Ord(i) \Rightarrow Ord(j) \Rightarrow Ord(k) \Rightarrow i \leq k$ 
  by (drule ltI, assumption, drule leI, rule Un_least_lt_iff[THEN iffD1[THEN conjunct1]],simp_all)

lemma Un_memD2 :  $i \cup j \in k \Rightarrow Ord(i) \Rightarrow Ord(j) \Rightarrow Ord(k) \Rightarrow j \leq k$ 
  by (drule ltI, assumption, drule leI, rule Un_least_lt_iff[THEN iffD1[THEN conjunct2]],simp_all)

This lemma allows to apply arithmetic simprocs to ordinal addition

lemma nat_oadd_add[simp]:
  assumes  $m \in \omega$   $n \in \omega$  shows  $n + m = n \#+ m$ 
  using assms
  by induct simp_all

lemma Ord_has_max_imp_succ:
  assumes  $Ord(\gamma)$   $\beta \in \gamma$   $\forall \alpha \in \gamma. \alpha \leq \beta$ 
  shows  $\gamma = succ(\beta)$ 
  using assms Ord_trans[of  $\beta \gamma$ ]
  unfolding lt_def
  by (intro equalityI subsetI) auto

lemma Least_antitone:
  assumes
     $Ord(j) P(j) \wedge i. P(i) \Rightarrow Q(i)$ 
  shows
     $(\mu i. Q(i)) \leq (\mu i. P(i))$ 
  using assms LeastI2[of  $P j Q$ ] Least_le by simp

lemma Least_set_antitone:
   $Ord(j) \Rightarrow j \in A \Rightarrow A \subseteq B \Rightarrow (\mu i. i \in B) \leq (\mu i. i \in A)$ 
  using subset_iff by (auto intro:Least_antitone)

lemma le_neq_imp_lt:
   $x \leq y \Rightarrow x \neq y \Rightarrow x < y$ 

```

```

using ltD ltI[of x y] le_Ord2
unfolding succ_def by auto

```

Strict upper bound of a set of ordinals.

**definition**

```

str_bound :: i⇒i where
str_bound(A) ≡ ∪ a∈A. succ(a)

```

```

lemma str_bound_type [TC]: ∀ a∈A. Ord(a) ⇒ Ord(str_bound(A))
unfolding str_bound_def by auto

```

```

lemma str_bound_lt: ∀ a∈A. Ord(a) ⇒ ∀ a∈A. a < str_bound(A)
unfolding str_bound_def using str_bound_type
by (blast intro:ltI)

```

```

lemma naturals_lt_nat[intro]: n ∈ ω ⇒ n < ω
unfolding lt_def by simp

```

The next two lemmas are handy when one is constructing some object recursively. The first handles injectivity (of recursively constructed sequences of sets), while the second is helpful for establishing a symmetry argument.

```

lemma Int_eq_zero_imp_not_eq:
assumes
    ∃x y. x∈D ⇒ y ∈ D ⇒ x ≠ y ⇒ A(x) ∩ A(y) = 0
    ∃x. x∈D ⇒ A(x) ≠ ∅ a∈D b∈D a≠b
shows
    A(a) ≠ A(b)
using assms by fastforce

```

```

lemma lt_neq_symmetry:
assumes
    ∃α β. α ∈ γ ⇒ β ∈ γ ⇒ α < β ⇒ Q(α,β)
    ∃α β. Q(α,β) ⇒ Q(β,α)
    α ∈ γ β ∈ γ α ≠ β
    Ord(γ)
shows
    Q(α,β)

```

```

proof -
from assms
consider α<β | β<α
using Ord_linear_lt[of α β thesis] Ord_in_Ord[of γ]
by auto
then
show ?thesis by cases (auto simp add:assms)

```

qed

```

lemma cardinal_succ_not_0: |A| = succ(n) ⇒ A ≠ ∅
by auto

```

```

lemma Ord_eq_Collect_lt:  $i < \alpha \implies \{j \in \alpha. j < i\} = i$ 
  — almost the same proof as nat_eq_Collect_lt
  apply (rule equalityI)
  apply (blast dest: ltD)
  apply (auto simp add: Ord_mem_iff_lt)
  apply (rule Ord_trans ltI[OF _ lt_Ord]; auto simp add: lt_def dest: ltD) +
  done

```

## 2.2 Manipulation of function spaces

**definition**

```

Finite_to_one ::  $[i,i] \Rightarrow i$  where
Finite_to_one( $X, Y$ )  $\equiv \{f: X \rightarrow Y. \forall y \in Y. \text{Finite}(\{x \in X. f'x = y\})\}$ 

```

```

lemma Finite_to_oneI[intro]:
assumes  $f: X \rightarrow Y \wedge y \in Y \implies \text{Finite}(\{x \in X. f'x = y\})$ 
shows  $f \in \text{Finite\_to\_one}(X, Y)$ 
using assms unfolding Finite_to_one_def by simp

```

```

lemma Finite_to_oneD[dest]:
 $f \in \text{Finite\_to\_one}(X, Y) \implies f: X \rightarrow Y$ 
 $f \in \text{Finite\_to\_one}(X, Y) \implies y \in Y \implies \text{Finite}(\{x \in X. f'x = y\})$ 
unfolding Finite_to_one_def by simp_all

```

```

lemma subset_Diff_Un:  $X \subseteq A \implies A = (A - X) \cup X$  by auto

```

```

lemma Diff_bij:
assumes  $\forall A \in F. X \subseteq A$  shows  $(\lambda A \in F. A - X) \in \text{bij}(F, \{A - X. A \in F\})$ 
using assms unfolding bij_def inj_def surj_def
by (auto intro:lam_type, subst subset_Diff_Un[of X]) auto

```

```

lemma function_space_nonempty:
assumes  $b \in B$ 
shows  $(\lambda x \in A. b) : A \rightarrow B$ 
using assms lam_type by force

```

```

lemma vimage_lam:  $(\lambda x \in A. f(x)) - `` B = \{x \in A. f(x) \in B\}$ 
using lam_funtype[of A f, THEN [2] domain_type]
lam_funtype[of A f, THEN [2] apply_equality] lamI[of _ A f]
by auto blast

```

```

lemma range_fun_subset_codomain:
assumes  $h: B \rightarrow C$ 
shows  $\text{range}(h) \subseteq C$ 
unfolding range_def domain_def converse_def using range_type[OF _ assms]
by auto

```

```

lemma Pi_rangeD:
assumes  $f \in \text{Pi}(A, B)$   $b \in \text{range}(f)$ 

```

```

shows  $\exists a \in A. f'a = b$ 
using assms apply_equality[OF _ assms(1), of _ b]
      domain_type[OF _ assms(1)] by auto

lemma Pi_range_eq:  $f \in Pi(A, B) \implies range(f) = \{f'x . x \in A\}$ 
  using Pi_rangeD[of f A B] apply_rangeI[of f A B]
  by blast

lemma Pi_vimage_subset :  $f \in Pi(A, B) \implies f``C \subseteq A$ 
  unfolding Pi_def by auto

lemma apply_in_codomain_Ord:
  assumes
     $Ord(\gamma) \neq 0$ 
     $f: A \rightarrow \gamma$ 
  shows
     $f'x \in \gamma$ 
  proof (cases x:A)
    case True
    from assms {x:A}
    show ?thesis
      using domain_of_fun apply_rangeI by simp
  next
    case False
    from assms {x  $\notin A$ }
    show ?thesis
      using apply_0 Ord_0_lt ltD domain_of_fun by auto
  qed

lemma range_eq_image:
  assumes  $f: A \rightarrow B$ 
  shows  $range(f) = f``A$ 
  proof
    show  $f``A \subseteq range(f)$ 
      unfolding image_def by blast
    {
      fix x
      assume  $x \in range(f)$ 
      with assms
      have  $x \in f``A$ 
        using domain_of_fun[of f A _ B] by auto
    }
    then
    show  $range(f) \subseteq f``A ..$ 
  qed

lemma Image_sub_codomain:  $f: A \rightarrow B \implies f``C \subseteq B$ 
  using image_subset_fun_is_rel[of _ _ _ B] by force

lemma inj_to_Image:

```

```

assumes
   $f:A \rightarrow B$   $f \in inj(A,B)$ 
shows
   $f \in inj(A,f``A)$ 
using assms inj_inj_range range_eq_image by force

lemma inj_imp_surj:
fixes  $f b$ 
notes inj_is_fun[dest]
defines [simp]:  $ifx(x) \equiv if x \in range(f) then converse(f)`x else b$ 
assumes  $f \in inj(B,A)$   $b \in B$ 
shows  $(\lambda x \in A. ifx(x)) \in surj(A,B)$ 
proof -
from assms
have  $converse(f) \in surj(range(f),B)$   $range(f) \subseteq A$ 
   $converse(f) : range(f) \rightarrow B$ 
using inj_converse_surj range_fun_subset_codomain surj_is_fun by blast+
with  $\langle b \in B \rangle$ 
show  $(\lambda x \in A. ifx(x)) \in surj(A,B)$ 
unfolding surj_def
proof (intro CollectI lam_type ballI; elim CollectE)
fix  $y$ 
assume  $y \in B \forall y \in B. \exists x \in range(f). converse(f)`x = y$ 
with  $\langle range(f) \subseteq A \rangle$ 
show  $\exists x \in A. (\lambda x \in A. ifx(x))`x = y$ 
  by (drule_tac bspec, auto)
qed simp
qed

lemma fun_Pi_disjoint_Un:
assumes  $f \in Pi(A,B)$   $g \in Pi(C,D)$   $A \cap C = \emptyset$ 
shows  $f \cup g \in Pi(A \cup C, \lambda x. B(x) \cup D(x))$ 
using assms
by (simp add: Pi_iff extension Un_rls) (unfold function_def, blast)

lemma Un_restrict_decomposition:
assumes  $f \in Pi(A,B)$ 
shows  $f = restrict(f, A \cap C) \cup restrict(f, A - C)$ 
using assms
proof (rule fun_extension)
from assms
have  $restrict(f, A \cap C) \cup restrict(f, A - C) \in Pi(A \cap C \cup (A - C), \lambda x. B(x) \cup D(x))$ 
using restrict_type2[of f A B]
by (rule_tac fun_Pi_disjoint_Un) force+
moreover
have  $(A \cap C) \cup (A - C) = A$  by auto
ultimately
show  $restrict(f, A \cap C) \cup restrict(f, A - C) \in Pi(A, B)$  by simp
next

```

```

fix x
assume x ∈ A
with assms
show f ` x = (restrict(f, A ∩ C) ∪ restrict(f, A - C)) ` x
  using restrict_fun_disjoint_apply1[of _ restrict(f,_)]
    fun_disjoint_apply2[of _ restrict(f,_)]
      domain_restrict[of f] apply_0 domain_of_fun
    by (cases x∈C) simp_all
qed

lemma restrict_eq_imp_Un_into_Pi:
assumes f ∈ Pi(A,B) g ∈ Pi(C,D) restrict(f, A ∩ C) = restrict(g, A ∩ C)
shows f ∪ g ∈ Pi(A ∪ C, λx. B(x) ∪ D(x))
proof -
  note assms
  moreover from this
  have x ∉ g ⟹ x ∉ restrict(g, A ∩ C) for x
    using restrict_subset[of g A ∩ C] by auto
  moreover from calculation
  have x ∈ f ⟹ x ∈ restrict(f, A - C) ∨ x ∈ restrict(g, A ∩ C) for x
    by (subst (asm) Un_restrict_decomposition[of f A B C]) auto
  ultimately
  have f ∪ g = restrict(f, A - C) ∪ g
    using restrict_subset[of g A ∩ C]
      by (subst Un_restrict_decomposition[of f A B C]) auto
  moreover
  have A - C ∪ C = A ∪ C by auto
  moreover
  note assms
  ultimately
  show ?thesis
    using fun_Pi_disjoint_Un[OF
      restrict_type2[of f A B A-C], of g C D]
    by auto
qed

lemma restrict_eq_imp_Un_into_Pi':
assumes f ∈ Pi(A,B) g ∈ Pi(C,D)
  restrict(f, domain(f) ∩ domain(g)) = restrict(g, domain(f) ∩ domain(g))
shows f ∪ g ∈ Pi(A ∪ C, λx. B(x) ∪ D(x))
using assms domain_of_fun restrict_eq_imp_Un_into_Pi by simp

lemma restrict_subset_Sigma: f ⊆ Sigma(C,B) ⟹ restrict(f,A) ⊆ Sigma(A ∩ C,
B)
  by (auto simp add: restrict_def)

```

## 2.3 Finite sets

lemma Replace\_sing1:

$\llbracket (\exists a. P(d,a)) \wedge (\forall y y'. P(d,y) \longrightarrow P(d,y') \longrightarrow y=y') \rrbracket \implies \exists a. \{y . x \in \{d\}, P(x,y)\} = \{a\}$   
**by blast**

— Not really necessary

**lemma** *Replace\_sing2*:  
**assumes**  $\forall a. \neg P(d,a)$   
**shows**  $\{y . x \in \{d\}, P(x,y)\} = \emptyset$   
**using assms by auto**

**lemma** *Replace\_sing3*:  
**assumes**  $\exists c e. c \neq e \wedge P(d,c) \wedge P(d,e)$   
**shows**  $\{y . x \in \{d\}, P(x,y)\} = \emptyset$   
**proof -**  
{  
  **fix**  $z$   
  {  
    **assume**  $\forall y. P(d,y) \longrightarrow y = z$   
    **with assms**  
    **have** *False* **by auto**  
  }  
  **then**  
  **have**  $z \notin \{y . x \in \{d\}, P(x,y)\}$   
  **using Replace\_iff by auto**  
}  
**then**  
**show** ?thesis  
  **by** (*intro equalityI subsetI*) *simp\_all*  
**qed**

**lemma** *Replace\_Un*:  $\{b . a \in A \cup B, Q(a,b)\} = \{b . a \in A, Q(a,b)\} \cup \{b . a \in B, Q(a,b)\}$   
**by** (*intro equalityI subsetI*) (*auto simp add:Replace\_iff*)

**lemma** *Replace\_subset\_sing*:  $\exists z. \{y . x \in \{d\}, P(x,y)\} \subseteq \{z\}$   
**proof -**  
**consider**  
  (1)  $(\exists a. P(d,a)) \wedge (\forall y y'. P(d,y) \longrightarrow P(d,y') \longrightarrow y=y')$  |  
  (2)  $\forall a. \neg P(d,a)$  | (3)  $\exists c e. c \neq e \wedge P(d,c) \wedge P(d,e)$  **by auto**  
**then**  
**show**  $\exists z. \{y . x \in \{d\}, P(x,y)\} \subseteq \{z\}$   
**proof** (*cases*)  
  **case** 1  
  **then show** ?thesis **using Replace\_sing1[of P d]** **by auto**  
**next**  
  **case** 2  
  **then show** ?thesis **by auto**  
**next**  
  **case** 3

```

    then show ?thesis using Replace_sing3[of P d] by auto
qed
qed

lemma Finite_Replace: Finite(A) ==> Finite(Replace(A,Q))
proof (induct rule:Finite_induct)
  case 0
  then
  show ?case by simp
next
  case (cons x B)
  moreover
  have {b . a ∈ cons(x, B), Q(a, b)} =
    {b . a ∈ B, Q(a, b)} ∪ {b . a ∈ {x}, Q(a, b)}
    using Replace_Un unfolding cons_def by auto
  moreover
  obtain d where {b . a ∈ {x}, Q(a, b)} ⊆ {d}
    using Replace_subset_sing[of _ Q] by blast
  moreover from this
  have Finite({b . a ∈ {x}, Q(a, b)})
    using subset_Finite by simp
  ultimately
  show ?case using subset_Finite by simp
qed

lemma Finite_domain: Finite(A) ==> Finite(domain(A))
  using Finite_Replace unfolding domain_def
  by auto

lemma Finite_converse: Finite(A) ==> Finite(converse(A))
  using Finite_Replace unfolding converse_def
  by auto

lemma Finite_range: Finite(A) ==> Finite(range(A))
  using Finite_domain Finite_converse unfolding range_def
  by blast

lemma Finite_Sigma: Finite(A) ==> ∀ x. Finite(B(x)) ==> Finite(Sigma(A,B))
  unfolding Sigma_def using Finite_RepFun Finite_Union
  by simp

lemma Finite_Pi: Finite(A) ==> ∀ x. Finite(B(x)) ==> Finite(Pi(A,B))
  using Finite_Sigma
  Finite_Pow subset_Finite[of Pi(A,B) Pow(Sigma(A,B))]
  unfolding Pi_def
  by auto

```

## 2.4 Basic results on equipollence, cardinality and related concepts

**lemma** *lepolld[dest]*:  $A \lesssim B \implies \exists f. f \in \text{inj}(A, B)$   
**unfolding** *lepolld\_def*.

**lemma** *lepolli[intro]*:  $f \in \text{inj}(A, B) \implies A \lesssim B$   
**unfolding** *lepolli\_def* **by** *blast*

**lemma** *eqpolld[dest]*:  $A \approx B \implies \exists f. f \in \text{bij}(A, B)$   
**unfolding** *eqpolld\_def*.

**declare** *bij\_imp\_eqpoll[intro]*

**lemma** *range\_of\_subset\_eqpoll*:  
**assumes**  $f \in \text{inj}(X, Y)$   $S \subseteq X$   
**shows**  $S \approx f `` S$   
**using** *assms restrict\_bij* **by** *blast*

I thank Miguel Pagano for this proof.

**lemma** *function\_space\_eqpoll\_cong*:  
**assumes**  
 $A \approx A'$   $B \approx B'$   
**shows**  
 $A \rightarrow B \approx A' \rightarrow B'$   
**proof** -  
**from** *assms(1)[THEN eqpoll\_sym]* *assms(2)*  
**obtain**  $f g$  **where**  $f \in \text{bij}(A', A)$   $g \in \text{bij}(B, B')$   
**by** *blast*  
**then**  
**have**  $\text{converse}(g) : B' \rightarrow B$   $\text{converse}(f) : A \rightarrow A'$   
**using** *bij\_converse\_bij bij\_is\_fun* **by** *auto*  
**show** *?thesis*  
**unfolding** *eqpoll\_def*  
**proof** (*intro exI fg\_imp\_bijequiv, rule\_tac [1-2] lam\_type*)  
**fix**  $F$   
**assume**  $F : A \rightarrow B$   
**with**  $\langle f \in \text{bij}(A', A) \rangle \langle g \in \text{bij}(B, B') \rangle$   
**show**  $g \circ F \circ f : A' \rightarrow B'$   
**using** *bij\_is\_fun comp\_fun* **by** *blast*  
**next**  
**fix**  $F$   
**assume**  $F : A' \rightarrow B'$   
**with**  $\langle \text{converse}(g) : B' \rightarrow B \rangle \langle \text{converse}(f) : A \rightarrow A' \rangle$   
**show**  $\text{converse}(g) \circ F \circ \text{converse}(f) : A \rightarrow B$   
**using** *comp\_fun* **by** *blast*  
**next**  
**from**  $\langle f \in \_ \rangle \langle g \in \_ \rangle \langle \text{converse}(f) \in \_ \rangle \langle \text{converse}(g) \in \_ \rangle$   
**have**  $(\bigwedge x. x \in A' \rightarrow B' \implies \text{converse}(g) \circ x \circ \text{converse}(f) \in A \rightarrow B)$   
**using** *bij\_is\_fun comp\_fun* **by** *blast*

```

then
have  $(\lambda x \in A \rightarrow B. g O x O f) O (\lambda x \in A' \rightarrow B'. converse(g) O x O converse(f))$ 
       $= (\lambda x \in A' \rightarrow B'. (g O converse(g)) O x O (converse(f) O f))$ 
using lam_cong comp_assoc comp_lam[of  $A' \rightarrow B'$ ] by auto
also
have ... =  $(\lambda x \in A' \rightarrow B'. id(B') O x O (id(A')))$ 
using left_comp_inverse[OF bij_is_inj[OF f_inj]] right_comp_inverse[OF bij_is_surj[OF g_inj]]
by auto
also
have ... =  $(\lambda x \in A' \rightarrow B'. x)$ 
using left_comp_id[OF fun_is_rel] right_comp_id[OF fun_is_rel] lam_cong
by auto
also
have ... =  $id(A' \rightarrow B')$  unfolding id_def by simp
finally
show  $(\lambda x \in A \rightarrow B. g O x O f) O (\lambda x \in A' \rightarrow B'. converse(g) O x O converse(f))$ 
       $= id(A' \rightarrow B')$ .
next
from  $\langle f \in \_, g \in \_ \rangle$ 
have  $(\bigwedge x. x \in A \rightarrow B \implies g O x O f \in A' \rightarrow B')$ 
using bij_is_fun comp_fun by blast
then
have  $(\lambda x \in A' \rightarrow B'. converse(g) O x O converse(f)) O (\lambda x \in A \rightarrow B. g O x O f)$ 
       $= (\lambda x \in A \rightarrow B. (converse(g) O g) O x O (f O converse(f)))$ 
using comp_lam comp_assoc by auto
also
have ... =  $(\lambda x \in A \rightarrow B. id(B) O x O (id(A)))$ 
using right_comp_inverse[OF bij_is_surj[OF f_surj]] left_comp_inverse[OF bij_is_inj[OF g_inj]] lam_cong
by auto
also
have ... =  $(\lambda x \in A \rightarrow B. x)$ 
using left_comp_id[OF fun_is_rel] right_comp_id[OF fun_is_rel] lam_cong
by auto
also
have ... =  $id(A \rightarrow B)$  unfolding id_def by simp
finally
show  $(\lambda x \in A' \rightarrow B'. converse(g) O x O converse(f)) O (\lambda x \in A \rightarrow B. g O x O f) = id(A \rightarrow B)$ .
qed
qed

lemma curry_eqpoll:
fixes d ν1 ν2 κ
shows ν1 → ν2 → κ ≈ ν1 × ν2 → κ
unfolding eqpoll_def

```

```

proof (intro exI, rule lam_bijection,
    rule_tac [1-2] lam_type, rule_tac [2] lam_type)
fix f z
assume f :  $\nu 1 \rightarrow \nu 2 \rightarrow \kappa$   $z \in \nu 1 \times \nu 2$ 
then
show f‘fst(z) ‘snd(z) ∈  $\kappa$ 
    by simp
next
fix f x y
assume f :  $\nu 1 \times \nu 2 \rightarrow \kappa$   $x \in \nu 1$   $y \in \nu 2$ 
then
show f‘⟨x,y⟩ ∈  $\kappa$  by simp
next — one composition is the identity:
fix f
assume f :  $\nu 1 \times \nu 2 \rightarrow \kappa$ 
then
show ( $\lambda x \in \nu 1 \times \nu 2. (\lambda x \in \nu 1. \lambda xa \in \nu 2. f ‘ \langle x, xa \rangle) ‘ fst(x) ‘ snd(x)) = f$ 
    by (auto intro:fun_extension)
qed simp — the other composition follows automatically

```

```

lemma Pow_eqpoll_function_space:
fixes d X
notes bool_of_o_def [simp]
defines [simp]:d(A) ≡ ( $\lambda x \in X. \text{bool\_of\_o}(x \in A)$ )
    — the witnessing map for the thesis:
shows Pow(X) ≈ X → 2
unfolding eqpoll_def
proof (intro exI, rule lam_bijection)
    — We give explicit mutual inverses
fix A
assume A ∈ Pow(X)
then
show d(A) : X → 2
    using lam_type[of _  $\lambda x. \text{bool\_of\_o}(x \in A)$   $\lambda_. 2$ ]
    by force
from ⟨A ∈ Pow(X)⟩
show {y ∈ X. d(A)‘y = 1} = A
    by (auto)
next
fix f
assume f : X → 2
then
show d({y ∈ X . f ‘ y = 1}) = f
    using apply_type[OF ⟨f: X → 2⟩]
    by (force intro:fun_extension)
qed blast

```

```

lemma cantor_inj: f ∉ inj(Pow(A), A)
using inj_imp_surj[OF _ Pow_bottom] cantor_surj by blast

```

```

definition
  cexp ::  $[i,i] \Rightarrow i (\hookrightarrow [76,1] 75)$  where
     $\kappa^{\uparrow\nu} \equiv |\nu \rightarrow \kappa|$ 

lemma Card_cexp:  $\text{Card}(\kappa^{\uparrow\nu})$ 
  unfolding cexp_def Card_cardinal by simp

lemma eq_csucc_ord:
   $\text{Ord}(i) \implies i^+ = |i|^+$ 
  using Card_lt_iff Least_cong unfolding csucc_def by auto

I thank Miguel Pagano for this proof.

lemma lesspoll_csucc:
  assumes  $\text{Ord}(\kappa)$ 
  shows  $d \prec \kappa^+ \longleftrightarrow d \lesssim \kappa$ 
proof
  assume  $d \prec \kappa^+$ 
  moreover
  note Card_is_Ord Ord( $\kappa$ )
  moreover from calculation
  have  $\kappa < \kappa^+ \text{ Card}(\kappa^+)$ 
    using Ord_cardinal_eqpoll csucc_basic by simp_all
  moreover from calculation
  have  $d \prec |\kappa|^+ \text{ Card}(|\kappa|) d \approx |d|$ 
    using eq_csucc_ord[of  $\kappa$ ] lesspoll_imp_eqpoll_eqpoll_sym by simp_all
  moreover from calculation
  have  $|d| < |\kappa|^+$ 
    using lesspoll_cardinal_lt csucc_basic by simp
  moreover from calculation
  have  $|d| \lesssim |\kappa|$ 
    using Card_lt_csucc_iff le_imp_lepoll by simp
  moreover from calculation
  have  $|d| \lesssim \kappa$ 
    using lepoll_eq_trans Ord_cardinal_eqpoll by simp
  ultimately
  show  $d \lesssim \kappa$ 
    using eq_lepoll_trans by simp
next
  from Ord( $\kappa$ )
  have  $\kappa < \kappa^+ \text{ Card}(\kappa^+)$ 
    using csucc_basic by simp_all
  moreover
  assume  $d \lesssim \kappa$ 
  ultimately
  have  $d \lesssim \kappa^+$ 
    using le_imp_lepoll leI lepoll_trans by simp
  moreover
  from d_lesssim_kappa Ord( $\kappa$ )

```

```

have  $\kappa^+ \lesssim \kappa$  if  $d \approx \kappa^+$ 
  using eqpoll_sym[OF that] eq_lepoll_trans[OF _ < d ≈ κ] by simp
moreover from calculation ⟨Card(_)⟩
have  $\neg d \approx \kappa^+$ 
  using lesspoll_irrefl lesspoll_trans1 lt_Card_imp_lesspoll[OF _ < κ < _]
  by auto
ultimately
show  $d \prec \kappa^+$ 
  unfolding lesspoll_def by simp
qed

abbreviation
Infinite ::  $i \Rightarrow o$  where
Infinite( $X$ )  $\equiv \neg \text{Finite}(X)$ 

lemma Infinite_not_empty:  $\text{Infinite}(X) \implies X \neq 0$ 
using empty_lepollI by auto

lemma Infinite_imp_nats_lepoll:
assumes Infinite( $X$ )  $n \in \omega$ 
shows  $n \lesssim X$ 
using ⟨ $n \in \omegaempty_lepollI by simp
next
case (succ  $x$ )
show ?case
proof -
from ⟨Infinite( $X$ )⟩ and ⟨ $x \in \omega\neg(x \approx X)$ 
using eqpoll_sym unfolding Finite_def by auto
with ⟨ $x \lesssim Xf$  where  $f \in \text{inj}(x, X)$   $f \notin \text{surj}(x, X)$ 
  unfolding bij_def eqpoll_def by auto
moreover from this
obtain  $b$  where  $b \in X \ \forall a \in x. f'a \neq b$ 
  using inj_is_fun unfolding surj_def by auto
ultimately
have  $f \in \text{inj}(x, X - \{b\})$ 
  unfolding inj_def by (auto intro:Pi_type)
then
have  $\text{cons}(\langle x, b \rangle, f) \in \text{inj}(\text{succ}(x), \text{cons}(b, X - \{b\}))$ 
  using inj_extend[of  $f x X - \{b\}$   $x b$ ] unfolding succ_def
  by (auto dest:mem_irrefl)
moreover from ⟨ $b \in X\text{cons}(b, X - \{b\}) = X$  by auto
ultimately$ 
```

```

show succ(x)  $\lesssim$  X by auto
qed
qed

lemma zero_lesspoll: assumes  $0 < \kappa$  shows  $0 \prec \kappa$ 
using assms eqpoll_0_iff[THEN iffD1, of  $\kappa$ ] eqpoll_sym
unfolding lesspoll_def lepoll_def
by (auto simp add: inj_def)

lemma lepoll_nat_imp_Infinite:  $\omega \lesssim X \implies \text{Infinite}(X)$ 
proof (rule ccontr, simp)
assume  $\omega \lesssim X \text{ Finite}(X)$ 
moreover from this
obtain n where  $X \approx n \ n \in \omega$ 
unfolding Finite_def by auto
moreover from calculation
have  $\omega \lesssim n$ 
using lepoll_eq_trans by simp
ultimately
show False
using lepoll_nat_imp_Finite nat_not_Finite by simp
qed

lemma InfCard_imp_Infinite: InfCard( $\kappa$ )  $\implies \text{Infinite}(\kappa)$ 
using le_imp_lepoll[THEN lepoll_nat_imp_Infinite, of  $\kappa$ ]
unfolding InfCard_def by simp

lemma lt_surj_empty_imp_Card:
assumes Ord( $\kappa$ )  $\wedge \alpha. \alpha < \kappa \implies \text{surj}(\alpha, \kappa) = 0$ 
shows Card( $\kappa$ )
proof -
{
assume  $|\kappa| < \kappa$ 
with assms
have False
using LeastI[of  $\lambda i. i \approx \kappa \ \kappa$ , OF eqpoll_refl]
Least_le[of  $\lambda i. i \approx \kappa \ |\kappa|$ , OF Ord_cardinal_eqpoll]
unfolding Card_def cardinal_def eqpoll_def bij_def
by simp
}
with assms
show ?thesis
using Ord_cardinal_le[of  $\kappa$ ] not_lt_imp_le[of  $|\kappa| \ \kappa$ ] le_anti_sym
unfolding Card_def by auto
qed

```

## 2.5 Morphisms of binary relations

The main case of interest is in the case of partial orders.

```

lemma mono_map_mono:
  assumes
     $f \in \text{mono\_map}(A, r, B, s) \quad B \subseteq C$ 
  shows
     $f \in \text{mono\_map}(A, r, C, s)$ 
  unfolding mono_map_def
  proof (intro CollectI ballI impI)
    from  $\langle f \in \text{mono\_map}(A, \_, B, \_) \rangle$ 
    have  $f: A \rightarrow B$ 
      using mono_map_is_fun by simp
    with  $\langle B \subseteq C \rangle$ 
    show  $f: A \rightarrow C$ 
      using fun_weaken_type by simp
    fix  $x y$ 
    assume  $x \in A \quad y \in A \quad \langle x, y \rangle \in r$ 
    moreover from this and  $\langle f: A \rightarrow B \rangle$ 
    have  $f'x \in B \quad f'y \in B$ 
      using apply_type by simp_all
    moreover
    note  $\langle f \in \text{mono\_map}(\_, r, \_, s) \rangle$ 
    ultimately
    show  $\langle f'x, f'y \rangle \in s$ 
    unfolding mono_map_def by blast
  qed

lemma ordertype_zero_imp_zero:  $\text{ordertype}(A, r) = 0 \implies A = 0$ 
  using ordermap_type[of A r]
  by (cases A=0) auto

lemma mono_map_increasing:
   $j \in \text{mono\_map}(A, r, B, s) \implies a \in A \implies c \in A \implies \langle a, c \rangle \in r \implies \langle j'a, j'c \rangle \in s$ 
  unfolding mono_map_def by simp

lemma linear_mono_map_reflects:
  assumes
     $\text{linear}(\alpha, r) \quad \text{trans}[\beta](s) \quad \text{irrefl}(\beta, s) \quad f \in \text{mono\_map}(\alpha, r, \beta, s)$ 
     $x \in \alpha \quad y \in \alpha \quad \langle f'x, f'y \rangle \in s$ 
  shows
     $\langle x, y \rangle \in r$ 
  proof -
    from  $\langle f \in \text{mono\_map}(\_, \_, \_, \_) \rangle$ 
    have preserves: $x \in \alpha \implies y \in \alpha \implies \langle x, y \rangle \in r \implies \langle f'x, f'y \rangle \in s$  for  $x y$ 
    unfolding mono_map_def by blast
    {
      assume  $\langle x, y \rangle \notin r \quad x \in \alpha \quad y \in \alpha$ 
      moreover
      note  $\langle f'x, f'y \rangle \in s$  and  $\langle \text{linear}(\alpha, r) \rangle$ 
      moreover from calculation
      have  $y = x \vee \langle y, x \rangle \in r$ 
    }
  
```

```

unfolding linear_def by blast
moreover
note preserves [of y x]
ultimately
have  $y = x \vee \langle f'y, f'x \rangle \in s$  by blast
moreover from  $\langle f \in \text{mono\_map}(\_, \_, \beta, \_) \rangle \langle x \in \alpha \rangle \langle y \in \alpha \rangle$ 
have  $f'x \in \beta \ f'y \in \beta$ 
using apply_type[ $\text{OF } \text{mono\_map\_is\_fun}$ ] by simp_all
moreover
note  $\langle \langle f'x, f'y \rangle \in s \rangle \langle \text{trans}[\beta](s) \rangle \langle \text{irrefl}(\beta, s) \rangle$ 
ultimately
have False
using trans_onD[of  $\beta \ s \ f'x \ f'y \ f'x$ ] irreflE by blast
}
with assms
show  $\langle x, y \rangle \in r$  by blast
qed

```

```

lemma irrefl_Memrel: irrefl(x, Memrel(x))
  unfolding irrefl_def using mem_irrefl by auto

```

```

lemmas Memrel_mono_map_reflects = linear_mono_map_reflects
[ $\text{OF well\_ord\_is\_linear}[\text{OF well\_ord\_Memrel}] \text{well\_ord\_is\_trans\_on}[\text{OF well\_ord\_Memrel}]$ 
irrefl_Memrel]

```

— Same proof as Paulson's mono\_map\_is\_inj'

```

lemma mono_map_is_inj':
  [ $\text{linear}(A, r); \text{irrefl}(B, s); f \in \text{mono\_map}(A, r, B, s)$ ]  $\implies f \in \text{inj}(A, B)$ 
  unfolding irrefl_def mono_map_def inj_def using linearE
  by (clarify, rename_tac x w)
  (erule_tac x=w and y=x in linearE, assumption+, (force intro: apply_type)+)

```

```

lemma mono_map_imp_ord_iso_image:
  assumes
    linear( $\alpha, r$ ) trans[ $\beta$ ](s) irrefl( $\beta, s$ )  $f \in \text{mono\_map}(\alpha, r, \beta, s)$ 
  shows
     $f \in \text{ord\_iso}(\alpha, r, f``\alpha, s)$ 
  unfolding ord_iso_def
  proof (intro CollectI ballI iffI)
    — Enough to show it's bijective and preserves both ways
    from assms
    have  $f \in \text{inj}(\alpha, \beta)$ 
      using mono_map_is_inj' by blast
    moreover from  $\langle f \in \text{mono\_map}(\_, \_, \_, \_) \rangle$ 
    have  $f \in \text{surj}(\alpha, f``\alpha)$ 
      unfolding mono_map_def using surj_image by auto
    ultimately
    show  $f \in \text{bij}(\alpha, f``\alpha)$ 
      unfolding bij_def using inj_is_fun inj_to_Image by simp
  
```

```

from ⟨ $f \in \text{mono\_map}(\_, \_, \_, \_)$ ⟩
show  $x \in \alpha \implies y \in \alpha \implies \langle x, y \rangle \in r \implies \langle f'x, f'y \rangle \in s$  for  $x y$ 
  unfolding  $\text{mono\_map\_def}$  by  $\text{blast}$ 
  with  $\text{assms}$ 
show  $\langle f'x, f'y \rangle \in s \implies x \in \alpha \implies y \in \alpha \implies \langle x, y \rangle \in r$  for  $x y$ 
  using  $\text{linear\_mono\_map\_reflects}$ 
  by  $\text{blast}$ 
qed

```

We introduce the following notation for strictly increasing maps between ordinals.

#### **abbreviation**

```

mono_map_Memrel ::  $[i, i] \Rightarrow i$  (infixr  $\hookrightarrow_{<} 60$ ) where
 $\alpha \rightarrow_{<} \beta \equiv \text{mono\_map}(\alpha, \text{Memrel}(\alpha), \beta, \text{Memrel}(\beta))$ 

```

**lemma**  $\text{mono\_map\_imp\_ord\_iso\_Memrel}$ :

**assumes**

$\text{Ord}(\alpha) \text{ Ord}(\beta) f : \alpha \rightarrow_{<} \beta$

**shows**

$f \in \text{ord\_iso}(\alpha, \text{Memrel}(\alpha), f''\alpha, \text{Memrel}(\beta))$

**using**  $\text{assms } \text{mono\_map\_imp\_ord\_iso\_image}[\text{OF well\_ord\_is\_linear}[\text{OF well\_ord\_Memrel}]$   
 $\text{well\_ord\_is\_trans\_on}[\text{OF well\_ord\_Memrel}] \text{ irrefl\_Memrel}]$  **by**  $\text{blast}$

**lemma**  $\text{mono\_map\_ordertype\_image}'$ :

**assumes**

$X \subseteq \alpha \text{ Ord}(\alpha) \text{ Ord}(\beta) f \in \text{mono\_map}(X, \text{Memrel}(\alpha), \beta, \text{Memrel}(\beta))$

**shows**

$\text{ordertype}(f''X, \text{Memrel}(\beta)) = \text{ordertype}(X, \text{Memrel}(\alpha))$

**using**  $\text{assms } \text{mono\_map\_is\_fun}[\text{of } f X \_\beta] \text{ ordertype\_eq}$

$\text{mono\_map\_imp\_ord\_iso\_image}[\text{OF well\_ord\_is\_linear}[\text{OF well\_ord\_Memrel}],$   
 $\text{THEN linear\_subset}]$

$\text{well\_ord\_is\_trans\_on}[\text{OF well\_ord\_Memrel}] \text{ irrefl\_Memrel}, \text{ of } \alpha X \beta f]$

$\text{well\_ord\_subset}[\text{OF well\_ord\_Memrel}] \text{ Image\_sub\_codomain}[\text{off } X \beta X]$  **by**  
 $\text{auto}$

**lemma**  $\text{mono\_map\_ordertype\_image}$ :

**assumes**

$\text{Ord}(\alpha) \text{ Ord}(\beta) f : \alpha \rightarrow_{<} \beta$

**shows**

$\text{ordertype}(f''\alpha, \text{Memrel}(\beta)) = \alpha$

**using**  $\text{assms } \text{mono\_map\_is\_fun} \text{ ordertype\_Memrel} \text{ ordertype\_eq}[\text{of } f \alpha \text{ Memrel}(\alpha)]$

$\text{mono\_map\_imp\_ord\_iso\_Memrel} \text{ well\_ord\_subset}[\text{OF well\_ord\_Memrel}] \text{ Im-}$   
 $\text{age\_sub\_codomain}[\text{of } \alpha]$

**by**  $\text{auto}$

**lemma**  $\text{apply\_in\_image}$ :  $f : A \rightarrow B \implies a \in A \implies f'a \in f''A$

**using**  $\text{range\_eq\_image} \text{ apply\_rangeI}[\text{of } f]$  **by**  $\text{simp}$

```

lemma Image_subset_Ord_imp_lt:
  assumes
     $Ord(\alpha) \ h `` A \subseteq \alpha \ x \in domain(h) \ x \in A \ function(h)$ 
  shows
     $h ` x < \alpha$ 
  using assms
  unfolding domain_def using imageI ltI function_apply_equality by auto

lemma ordermap_le_arg:
  assumes
     $X \subseteq \beta \ x \in X \ Ord(\beta)$ 
  shows
     $x \in X \implies ordermap(X, Memrel(\beta)) ` x \leq x$ 
proof (induct rule: Ord_induct[OF subsetD, OF assms])
  case (1 x)
  have wf[X](Memrel(\beta))
  using wf_imp_wf_on[OF wf_Memrel] .
  with 1
  have ordermap(X, Memrel(\beta)) ` x = {ordermap(X, Memrel(\beta)) ` y . y \in {y \in X . y \in x} \wedge y \in \beta} }
  using ordermap_unfold Ord_trans[of _ x \beta] by auto
  also from assms
  have ... = {ordermap(X, Memrel(\beta)) ` y . y \in {y \in X . y \in x} }
  using Ord_trans[of _ x \beta] Ord_in_Ord by blast
  finally
  have ordm:ordermap(X, Memrel(\beta)) ` x = {ordermap(X, Memrel(\beta)) ` y . y \in {y \in X . y \in x}} .
  from 1
  have y \in x  $\implies y \in X \implies ordermap(X, Memrel(\beta)) ` y \leq y$  for y by simp
  with {x \in \beta} and {Ord(\beta)}
  have y \in x  $\implies y \in X \implies ordermap(X, Memrel(\beta)) ` y \in x$  for y
  using ltI[OF _ Ord_in_Ord[of \beta x]] lt_trans1 ltD by blast
  with ordm
  have ordermap(X, Memrel(\beta)) ` x \subseteq x by auto
  with {x \in X} assms
  show ?case
  using subset_imp_le Ord_in_Ord[of \beta x] Ord_ordermap
  well_ord_subset[OF well_ord_Memrel, of \beta] by force
qed

lemma subset_imp_ordertype_le:
  assumes
     $X \subseteq \beta \ Ord(\beta)$ 
  shows
     $ordertype(X, Memrel(\beta)) \leq \beta$ 
proof -
  {
    fix x
    assume x \in X

```

```

with assms
have ordermap(X,Memrel( $\beta$ )) ‘ $x \leq x$ 
  using ordermap_le_arg by simp
with ‘ $x \in X$ ’ and assms
have ordermap(X,Memrel( $\beta$ )) ‘ $x \in \beta$  (is ? $y \in \_\_$ )
  using ltD[of ? $y$  succ( $x$ )] Ord_trans[of ? $y$   $x \beta$ ] by auto
}
then
have ordertype(X, Memrel( $\beta$ ))  $\subseteq \beta$ 
  using ordertype_unfold[of X] by auto
with assms
show ?thesis
  using subset_imp_le Ord_ordertype[OF well_ord_subset, OF well_ord_Memrel]
by simp
qed

lemma mono_map_imp_le:
assumes
   $f \in \text{mono\_map}(\alpha, \text{Memrel}(\alpha), \beta, \text{Memrel}(\beta)) \quad \text{Ord}(\alpha) \quad \text{Ord}(\beta)$ 
shows
   $\alpha \leq \beta$ 
proof -
  from assms
  have  $f \in \langle \alpha, \text{Memrel}(\alpha) \rangle \cong \langle f``\alpha, \text{Memrel}(\beta) \rangle$ 
    using mono_map_imp_ord_iso_Memrel by simp
  then
  have converse( $f$ )  $\in \langle f``\alpha, \text{Memrel}(\beta) \rangle \cong \langle \alpha, \text{Memrel}(\alpha) \rangle$ 
    using ord_iso_sym by simp
  with ‘Ord( $\alpha$ )’
  have  $\alpha = \text{ordertype}(f``\alpha, \text{Memrel}(\beta))$ 
    using ordertype_eq well_ord_Memrel ordertype_Memrel by auto
  also from assms
  have ordertype( $f``\alpha, \text{Memrel}(\beta)$ )  $\leq \beta$ 
    using subset_imp_ordertype_le mono_map_is_fun[off] Image_sub_codomain[of  $f$ ] by force
  finally
  show ?thesis .
qed

—  $\llbracket \text{Ord}(A); f \in \text{mono\_map}(A, \text{Memrel}(A), B, \text{Memrel}(Aa)) \rrbracket \implies f \in \text{inj}(A, B)$ 
lemmas Memrel_mono_map_is_inj = mono_map_is_inj
  [OF well_ord_is_linear[OF well_ord_Memrel]
   wf_imp_wf_on[OF wf_Memrel]]
```

**lemma** *mono\_mapI*:
**assumes**  $f: A \rightarrow B \wedge \forall x y. x \in A \implies y \in A \implies \langle x, y \rangle \in r \implies \langle f`x, f`y \rangle \in s$ 
**shows**  $f \in \text{mono\_map}(A, r, B, s)$ 
**unfolding** *mono\_map\_def* **using** *assms* **by** *simp*

```

lemmas mono_mapD = mono_map_is_fun mono_map_increasing

bundle mono_map_rules = mono_mapI[intro!] mono_map_is_fun[dest] mono_mapD[dest]

lemma nats_le_InfCard:
  assumes n ∈ ω InfCard(κ)
  shows n ≤ κ
  using assms Ord_is_Transset
    le_trans[of n ω κ, OF le_subset_iff[THEN iffD2]]
  unfolding InfCard_def Transset_def by simp

lemma nat_into_InfCard:
  assumes n ∈ ω InfCard(κ)
  shows n ∈ κ
  using assms le_imp_subset[of ω κ]
  unfolding InfCard_def by auto

```

## 2.6 Alephs are infinite cardinals

```

lemma Aleph_zero_eq_nat: ℙ₀ = ω
  unfolding Aleph_def by simp

lemma InfCard_Aleph:
  notes Aleph_zero_eq_nat[simp]
  assumes Ord(α)
  shows InfCard(ℙ_α)
proof -
  have ¬ (ℙ_α ∈ ω)
  proof (cases α=0)
    case True
    then show ?thesis using mem_irrefl by auto
  next
    case False
    with ⟨Ord(α)⟩
    have ω ∈ ℙ_α using Ord_0_lt[of α] ltD by (auto dest:Aleph_increasing)
    then show ?thesis using foundation by blast
  qed
  with ⟨Ord(α)⟩
  have ¬ (|ℙ_α| ∈ ω)
    using Card_cardinal_eq by auto
  then
    have ¬ Finite(ℙ_α) by auto
    with ⟨Ord(α)⟩
    show ?thesis
      using Inf_Card_is_InfCard by simp
qed

```

Most properties of cardinals depend on *AC*, even for the countable. Here we just state the definition of this concept, and most proofs will appear after assuming Choice.

```
definition
countable ::  $i \Rightarrow o$  where
countable( $X$ )  $\equiv X \lesssim \omega$ 
```

```
lemma countableI[intro]:  $X \lesssim \omega \implies \text{countable}(X)$ 
unfolding countable_def by simp
```

```
lemma countableD[dest]:  $\text{countable}(X) \implies X \lesssim \omega$ 
unfolding countable_def by simp
```

A *delta system* is family of sets with a common pairwise intersection. We will work with this notion in Section 5, but we state the definition here in order to have it available in a choiceless context.

**definition**

```
delta_system ::  $i \Rightarrow o$  where
delta_system( $D$ )  $\equiv \exists r. \forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = r$ 
```

```
lemma delta_systemI[intro]:
assumes  $\forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = r$ 
shows delta_system( $D$ )
using assms unfolding delta_system_def by simp
```

```
lemma delta_systemD[dest]:
delta_system( $D$ )  $\implies \exists r. \forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = r$ 
unfolding delta_system_def by simp
```

Hence, pairwise intersections equal the intersection of the whole family.

```
lemma delta_system_root_eq_Inter:
assumes delta_system( $D$ )
shows  $\forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = \bigcap D$ 
proof (clarify, intro equalityI, auto)
fix  $A' B' x C$ 
assume hyp:  $A' \in D$   $B' \in D$   $A' \neq B'$   $x \in A'$   $x \in B'$   $C \in D$ 
with assms
obtain  $r$  where delta:  $\forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = r$ 
by auto
show  $x \in C$ 
proof (cases  $C = A'$ )
case True
with hyp and assms
show ?thesis by simp
next
case False
moreover
note hyp
moreover from calculation and delta
have  $r = C \cap A'$   $A' \cap B' = r$   $x \in r$  by auto
ultimately
show ?thesis by simp
```

```

qed
qed

lemmas Limit_Aleph = InfCard_Aleph[THEN InfCard_is_Limit]

lemmas Aleph_cont = Normal_imp_cont[OF Normal_Aleph]
lemmas Aleph_sup = Normal_Union[OF __ Normal_Aleph]

bundle Ord_dests = Limit_is_Ord[dest] Card_is_Ord[dest]
bundle Aleph_dests = Aleph_cont[dest] Aleph_sup[dest]
bundle Aleph_intros = Aleph_increasing[intro!]
bundle Aleph_mem_dests = Aleph_increasing[OF ltI, THEN ltD, dest]

```

## 2.7 Transfinite recursive constructions

**definition**

```

rec_constr :: [i,i] => i where
rec_constr(f,α) ≡ transrec(α,λa g. f'(g``a))

```

The function *rec\_constr* allows to perform *recursive constructions*: given a choice function on the powerset of some set, a transfinite sequence is created by successively choosing some new element.

The next result explains its use.

```

lemma rec_constr_unfold: rec_constr(f,α) = f'({rec_constr(f,β). β∈α})
  using def_transrec[OF rec_constr_def, of f α] image_lam by simp

```

```

lemma rec_constr_type: assumes f:Pow(G)→ G Ord(α)
  shows rec_constr(f,α) ∈ G
  using assms(2,1)
  by (induct rule:trans_induct)
    (subst rec_constr_unfold, rule apply_type[of f Pow(G) λ_. G], auto)

```

end

## 3 Cofinality

```

theory Cofinality
imports
  ZF_Library
begin

```

### 3.1 Basic results and definitions

A set  $X$  is *cofinal* in  $A$  (with respect to the relation  $r$ ) if every element of  $A$  is “bounded above” by some element of  $X$ . Note that  $X$  does not need to be a subset of  $A$ .

**definition**

```
cofinal :: [i,i,i] ⇒ o where
cofinal(X,A,r) ≡ ∀ a∈A. ∃ x∈X. ⟨a,x⟩∈r ∨ a = x
```

A function is cofinal if it range is.

**definition**

```
cofinal_fun :: [i,i,i] ⇒ o where
cofinal_fun(f,A,r) ≡ ∀ a∈A. ∃ x∈domain(f). ⟨a,f‘x⟩∈r ∨ a = f‘x
```

**lemma** cofinal\_funI:

```
assumes ∀ a. a∈A ⇒ ∃ x∈domain(f). ⟨a,f‘x⟩∈r ∨ a = f‘x
shows cofinal_fun(f,A,r)
using assms unfolding cofinal_fun_def by simp
```

**lemma** cofinal\_funD:

```
assumes cofinal_fun(f,A,r) a∈A
shows ∃ x∈domain(f). ⟨a,f‘x⟩∈r ∨ a = f‘x
using assms unfolding cofinal_fun_def by simp
```

**lemma** cofinal\_in\_cofinal:

```
assumes
  trans(r) cofinal(Y,X,r) cofinal(X,A,r)
shows
  cofinal(Y,A,r)
  unfolding cofinal_def
proof
  fix a
  assume a∈A
  moreover from ⟨cofinal(X,A,r)⟩
  have b∈A⇒∃ x∈X. ⟨b,x⟩∈r ∨ b=x for b
    unfolding cofinal_def by simp
  ultimately
  obtain y where y∈X ⟨a,y⟩∈r ∨ a=y by auto
  moreover from ⟨cofinal(Y,X,r)⟩
  have c∈X⇒∃ y∈Y. ⟨c,y⟩∈r ∨ c=y for c
    unfolding cofinal_def by simp
  ultimately
  obtain x where x∈Y ⟨y,x⟩∈r ∨ y=x by auto
  with ⟨a∈A⟩ ⟨y∈X⟩ ⟨⟨a,y⟩∈r ∨ a=y⟩ ⟨trans(r)⟩
  show ∃ x∈Y. ⟨a,x⟩∈r ∨ a=x unfolding trans_def by auto
qed
```

**lemma** codomain\_is\_cofinal:

```
assumes cofinal_fun(f,A,r) f:C → D
shows cofinal(D,A,r)
unfolding cofinal_def
proof
  fix b
  assume b ∈ A
```

```

moreover from assms
have  $a \in A \implies \exists x \in \text{domain}(f). \langle a, f^x \rangle \in r \vee a = f^x$  for a
  unfolding cofinal_fun_def by simp
ultimately
obtain x where  $x \in \text{domain}(f) \langle b, f^x \rangle \in r \vee b = f^x$ 
  by blast
moreover from  $\langle f: C \rightarrow D \rangle \langle x \in \text{domain}(f) \rangle$ 
have  $f^x \in D$ 
  using domain_of_fun apply_rangeI by simp
ultimately
show  $\exists y \in D. \langle b, y \rangle \in r \vee b = y$  by auto
qed

lemma cofinal_range_iff_cofinal_fun:
assumes function(f)
shows cofinal(range(f), A, r)  $\longleftrightarrow$  cofinal_fun(f, A, r)
unfolding cofinal_fun_def
proof (intro iffI ballI)
fix a
assume  $a \in A \langle \text{cofinal}(\text{range}(f), A, r) \rangle$ 
then
obtain y where  $y \in \text{range}(f) \langle a, y \rangle \in r \vee a = y$ 
  unfolding cofinal_def by blast
moreover from this
obtain x where  $\langle x, y \rangle \in f$ 
  unfolding range_def domain_def converse_def by blast
moreover
note  $\langle \text{function}(f) \rangle$ 
ultimately
have  $\langle a, f^x \rangle \in r \vee a = f^x$ 
  using function_apply_equality by blast
with  $\langle \langle x, y \rangle \in f \rangle$ 
show  $\exists x \in \text{domain}(f). \langle a, f^x \rangle \in r \vee a = f^x$  by blast
next
assume  $\forall a \in A. \exists x \in \text{domain}(f). \langle a, f^x \rangle \in r \vee a = f^x$ 
with assms
show cofinal(range(f), A, r)
  using function_apply_Pair[of f] unfolding cofinal_def by fast
qed

lemma cofinal_comp:
assumes
 $f \in \text{mono\_map}(C, s, D, r)$  cofinal_fun(f, D, r)  $h: B \rightarrow C$  cofinal_fun(h, C, s)
trans(r)
shows cofinal_fun(f O h, D, r)
unfolding cofinal_fun_def
proof
fix a
from  $\langle f \in \text{mono\_map}(C, s, D, r) \rangle$ 

```

```

have  $f:C \rightarrow D$ 
  using mono_map_is_fun by simp
  with  $\langle h:B \rightarrow C \rangle$ 
have  $\text{domain}(f) = C$   $\text{domain}(h) = B$ 
  using domain_of_fun by simp_all
moreover
assume  $a \in D$ 
moreover
note  $\langle \text{cofinal\_fun}(f,D,r) \rangle$ 
ultimately
obtain  $c$  where  $c \in C$   $\langle a, f^c c \rangle \in r \vee a = f^c c$ 
  unfolding cofinal_fun_def by blast
with  $\langle \text{cofinal\_fun}(h,C,s) \rangle$   $\langle \text{domain}(h) = B \rangle$ 
obtain  $b$  where  $b \in B$   $\langle c, h^b b \rangle \in s \vee c = h^b b$ 
  unfolding cofinal_fun_def by blast
moreover from this and  $\langle h:B \rightarrow C \rangle$ 
have  $h^b b \in C$  by simp
moreover
note  $\langle f \in \text{mono\_map}(C,s,D,r) \rangle$   $\langle c \in C \rangle$ 
ultimately
have  $\langle f^c, f^c(h^b b) \rangle \in r \vee f^c c = f^c(h^b b)$ 
  unfolding mono_map_def by blast
with  $\langle \langle a, f^c c \rangle \in r \vee a = f^c c \rangle$   $\langle \text{trans}(r) \rangle$   $\langle h:B \rightarrow C \rangle$   $\langle b \in B \rangle$ 
have  $\langle a, (f \circ h)^c b \rangle \in r \vee a = (f \circ h)^c b$ 
  using transD by auto
moreover from  $\langle h:B \rightarrow C \rangle$   $\langle \text{domain}(f) = C \rangle$   $\langle \text{domain}(h) = B \rangle$ 
have  $\text{domain}(f \circ h) = B$ 
  using range_fun_subset_codomain by blast
moreover
note  $\langle b \in B \rangle$ 
ultimately
show  $\exists x \in \text{domain}(f \circ h). \langle a, (f \circ h)^c x \rangle \in r \vee a = (f \circ h)^c x$  by blast
qed

definition
 $cf\_fun :: [i,i] \Rightarrow o$  where
 $cf\_fun(f,\alpha) \equiv \text{cofinal\_fun}(f,\alpha, \text{Memrel}(\alpha))$ 

lemma cf_funI[intro!]:  $\text{cofinal\_fun}(f,\alpha, \text{Memrel}(\alpha)) \implies cf\_fun(f,\alpha)$ 
  unfolding cf_fun_def by simp

lemma cf_funD[dest!]:  $cf\_fun(f,\alpha) \implies \text{cofinal\_fun}(f,\alpha, \text{Memrel}(\alpha))$ 
  unfolding cf_fun_def by simp

lemma cf_fun_comp:
assumes
   $\text{Ord}(\alpha) f \in \text{mono\_map}(C,s,\alpha, \text{Memrel}(\alpha))$   $cf\_fun(f,\alpha)$ 
   $h:B \rightarrow C$   $\text{cofinal\_fun}(h,C,s)$ 
shows  $cf\_fun(f \circ h,\alpha)$ 

```

```

using assms cofinal_comp[OF _ _ _ _ trans_Memrel] by auto

definition
  cf :: i⇒i where
  cf(γ) ≡ μ β. ∃ A. A ⊆ γ ∧ cofinal(A, γ, Memrel(γ)) ∧ β = ordertype(A, Memrel(γ))

lemma Ord_cf [TC]: Ord(cf(β))
  unfolding cf_def using Ord_Least by simp

lemma gamma_cofinal_gamma:
  assumes Ord(γ)
  shows cofinal(γ, γ, Memrel(γ))
  unfolding cofinal_def by auto

lemma cf_is_ordertype:
  assumes Ord(γ)
  shows ∃ A. A ⊆ γ ∧ cofinal(A, γ, Memrel(γ)) ∧ cf(γ) = ordertype(A, Memrel(γ))
    (is ?P(cf(γ)))
  using gamma_cofinal_gamma LeastI[of ?P γ] ordertype_Memrel[symmetric]
assms
  unfolding cf_def by blast

lemma cf_fun_succ':
  assumes Ord(β) Ord(α) f:α→succ(β)
  shows (∃ x∈α. f‘x=β) ↔ cf_fun(f, succ(β))
proof (intro iffI)
  assume (∃ x∈α. f‘x=β)
  with assms
  show cf_fun(f, succ(β))
    using domain_of_fun[OF ⟨f:α→succ(β)⟩]
    unfolding cf_fun_def cofinal_fun_def by auto
next
  assume cf_fun(f, succ(β))
  with assms
  obtain x where x∈α ⟨β, f‘x⟩ ∈ Memrel(succ(β)) ∨ β = f ‘ x
    using domain_of_fun[OF ⟨f:α→succ(β)⟩]
    unfolding cf_fun_def cofinal_fun_def by auto
  moreover from ⟨Ord(β)⟩
  have ⟨β, y⟩ ∉ Memrel(succ(β)) for y
    using foundation unfolding Memrel_def by blast
  ultimately
  show ∃ x∈α. f ‘ x = β by blast
qed

lemma cf_fun_succ:
  Ord(β) ⟹ f:1→succ(β) ⟹ f‘0=β ⟹ cf_fun(f, succ(β))
  using cf_fun_succ' by blast

lemma ordertype_0_not_cofinal_succ:

```

```

assumes ordertype(A,Memrel(succ(i))) = 0 A⊆succ(i) Ord(i)
shows ¬cofinal(A,succ(i),Memrel(succ(i)))
proof
have 1:ordertype(A,Memrel(succ(i))) = ordertype(0,Memrel(0))
  using ⟨ordertype(A,Memrel(succ(i))) = 0⟩ ordertype_0 by simp
from ⟨A⊆succ(i)⟩ ⟨Ord(i)⟩
have ∃f. f ∈ ⟨A, Memrel(succ(i))⟩ ≈ ⟨0, Memrel(0)⟩
  using well_ord_Memrel well_ord_subset
  ordertype_eq_imp_ord_iso[OF 1] Ord_0 by blast
then
have A=0
  using ord_iso_is_bij bij_imp_eqpoll_eqpoll_0_is_0 by blast
moreover
assume cofinal(A, succ(i), Memrel(succ(i)))
moreover
note ⟨Ord(i)⟩
ultimately
show False
  using not_mem_empty unfolding cofinal_def by auto
qed

```

I thank Edwin Pacheco Rodríguez for the following lemma.

```

lemma cf_succ:
  assumes Ord(α)
  shows cf(succ(α)) = 1
proof -
define f where f ≡ {⟨0,α⟩}
then
have f : 1 → succ(α) f‘0 = α
  using fun_extend3[of 0 0 succ(α) 0 α] singleton_0 by auto
with assms
have cf_fun(f,succ(α))
  using cf_fun_succ unfolding cofinal_fun_def by simp
from ⟨f:1→succ(α)⟩
have 0∈domain(f) using domain_of_fun by simp
define A where A={f‘0}
with ⟨cf_fun(f,succ(α))⟩ ⟨0∈domain(f)⟩ ⟨f‘0=α⟩
have cofinal(A,succ(α),Memrel(succ(α)))
  unfolding cofinal_def cofinal_fun_def by simp
moreover from ⟨f‘0=α⟩ ⟨A={f‘0}⟩
have A ⊆ succ(α) unfolding succ_def by auto
moreover from ⟨Ord(α)⟩ ⟨A⊆ succ(α)⟩
have well_ord(A,Memrel(succ(α)))
  using Ord_succ well_ord_Memrel well_ord_subset relation_Memrel by blast
moreover from ⟨Ord(α)⟩
have ¬(∃A. A ⊆ succ(α) ∧ cofinal(A, succ(α), Memrel(succ(α))) ∧ 0 = ordertype(A, Memrel(succ(α))))
  (is ¬?P(0))
  using ordertype_0_not_cofinal_succ unfolding cf_def by auto

```

```

moreover
have 1 = ordertype(A,Memrel(succ(α)))
proof -
  from ⟨A={f'0}⟩
  have A≈1 using singleton_eqpoll_1 by simp
  with ⟨well_ord(A,Memrel(succ(α)))⟩
  show ?thesis using nat_1I ordertype_eq_n by simp
qed
ultimately
show cf(succ(α)) = 1 using Ord_1 Least_equality[of ?P 1]
  unfolding cf_def by blast
qed

lemma cf_zero [simp]:
cf(0) = 0
unfolding cf_def cofinal_def using
ordertype_0 subset_empty_iff Least_le[of _ 0] by auto

lemma surj_is_cofinal: f ∈ surj(δ,γ) ⇒ cf_fun(f,γ)
  unfolding surj_def cofinal_fun_def cf_fun_def
  using domain_of_fun by force

lemma cf_zero_iff: Ord(α) ⇒ cf(α) = 0 ↔ α = 0
proof (intro iffI)
  assume α = 0 Ord(α)
  then
    show cf(α) = 0 using cf_zero by simp
next
  assume cf(α) = 0 Ord(α)
  moreover from this
  obtain A where A⊆α cf(α) = ordertype(A,Memrel(α))
    cofinal(A,α,Memrel(α))
    using cf_is_ordertype by blast
  ultimately
  have cofinal(0,α,Memrel(α))
    using ordertype_zero_imp_zero[of A Memrel(α)] by simp
  then
    show α=0
      unfolding cofinal_def by blast
qed

```

— TODO: define Succ (predicate for successor ordinals)

```

lemma cf_eq_one_iff:
assumes Ord(γ)
shows cf(γ) = 1 ↔ (∃α. Ord(α) ∧ γ = succ(α))
proof (intro iffI)
  assume ∃α. Ord(α) ∧ γ = succ(α)
  then
  show cf(γ) = 1 using cf_succ by auto

```

```

next
assume  $cf(\gamma) = 1$ 
moreover from assms
obtain  $A$  where  $A \subseteq \gamma$   $cf(\gamma) = ordertype(A, Memrel(\gamma))$ 
cofinal( $A, \gamma, Memrel(\gamma)$ )
using  $cf\_is\_ordertype$  by blast
ultimately
have  $ordertype(A, Memrel(\gamma)) = 1$  by simp
moreover
define  $f$  where  $f \equiv converse(ordertype(A, Memrel(\gamma)))$ 
moreover from this  $\langle ordertype(A, Memrel(\gamma)) = 1 \rangle \langle A \subseteq \gamma \rangle$  assms
have  $f \in surj(1, A)$ 
using well_ord_subset[OF well_ord_Memrel, THEN ordemap_bij,
THEN bij_converse_bij, of  $\gamma$   $A$ ] bij_is_surj
by simp
with  $\langle cofinal(A, \gamma, Memrel(\gamma)) \rangle$ 
have  $\forall a \in \gamma. \langle a, f^{\prime}0 \rangle \in Memrel(\gamma) \vee a = f^{\prime}0$ 
unfolding cofinal_def surj_def
by auto
with assms  $\langle A \subseteq \gamma \rangle \langle f \in surj(1, A) \rangle$ 
show  $\exists \alpha. Ord(\alpha) \wedge \gamma = succ(\alpha)$ 
using Ord_has_max_imp_succ[of  $\gamma f^{\prime}0$ ]
surj_is_fun[of f 1 A] apply_type[of f 1 λ_.A 0]
unfolding lt_def
by (auto intro: Ord_in_Ord)
qed

lemma ordertype_in_cf_imp_not_cofinal:
assumes
ordertype(A, Memrel(γ)) ∈ cf(γ)
 $A \subseteq \gamma$ 
shows
 $\neg cofinal(A, \gamma, Memrel(\gamma))$ 
proof
note  $\langle A \subseteq \gamma \rangle$ 
moreover
assume  $cofinal(A, \gamma, Memrel(\gamma))$ 
ultimately
have  $\exists B. B \subseteq \gamma \wedge cofinal(B, \gamma, Memrel(\gamma)) \wedge ordertype(A, Memrel(\gamma)) = ordertype(B, Memrel(\gamma))$ 
(is ?P(ordertype(A, _)))
by blast
moreover from assms
have  $ordertype(A, Memrel(\gamma)) < cf(\gamma)$ 
using Ord_cf_ltI by blast
ultimately
show False
unfolding cf_def using less_LeastE[of ?P ordertype(A, Memrel(γ))]
by auto

```

qed

```
lemma cofinal_mono_map_cf:
  assumes Ord(γ)
  shows ∃j ∈ mono_map(cf(γ), Memrel(cf(γ)), γ, Memrel(γ)) . cf_fun(j,γ)
proof -
  note assms
  moreover from this
  obtain A where A ⊆ γ cf(γ) = ordertype(A,Memrel(γ))
    cofinal(A,γ,Memrel(γ))
    using cf_is_ordertype by blast
  moreover
  define j where j ≡ converse(ordermap(A,Memrel(γ)))
  moreover from calculation
  have j :cf(γ) → < γ
    using ordertype_ord_iso[THEN ord_iso_sym,
      THEN ord_iso_is_mono_map, THEN mono_map_mono,
      of A Memrel(γ) γ] well_ord_Memrel[THEN well_ord_subset]
    by simp
  moreover from calculation
  have j ∈ surj(cf(γ),A)
    using well_ord_Memrel[THEN well_ord_subset, THEN ordertype_ord_iso,
      THEN ord_iso_sym, of γ A, THEN ord_iso_is_bij,
      THEN bij_is_surj]
    by simp
  with ⟨cofinal(A,γ,Memrel(γ))⟩
  have cf_fun(j,γ)
    using cofinal_range_iff_cofinal_fun[of j γ Memrel(γ)]
    surj_range[of j cf(γ) A] surj_is_fun fun_is_function
    by fastforce
  with ⟨j ∈ mono_map(_,_,_,_)}⟩
  show ?thesis by auto
qed
```

### 3.2 The factorization lemma

In this subsection we prove a factorization lemma for cofinal functions into ordinals, which shows that any cofinal function between ordinals can be “decomposed” in such a way that a commutative triangle of strictly increasing maps arises.

The factorization lemma has a kind of fundamental character, in that the rest of the basic results on cofinality (for, instance, idempotence) follow easily from it, in a more algebraic way.

This is a consequence that the proof encapsulates uses of transfinite recursion in the basic theory of cofinality; indeed, only one use is needed. In the setting of Isabelle/ZF, this is convenient since the machinery of recursion is pretty clumsy. On the downside, this way of presenting things results in a longer

proof of the factorization lemma. This approach was taken by the author in the notes [8] for an introductory course in Set Theory.

To organize the use of the hypotheses of the factorization lemma, we set up a locale containing all the relevant ingredients.

```
locale cofinal_factor =
  fixes j δ ξ γ f
  assumes j_mono:  $j : \xi \rightarrow_{\text{c}} \gamma$ 
    and ords:  $\text{Ord}(\delta) \text{ Ord}(\xi) \text{ Limit}(\gamma)$ 
    and f_type:  $f : \delta \rightarrow \gamma$ 
begin
```

Here,  $f$  is cofinal function from  $\delta$  to  $\gamma$ , and the ordinal  $\xi$  is meant to be the cofinality of  $\gamma$ . Hence, there exists an increasing map  $j$  from  $\xi$  to  $\gamma$  by the last lemma.

The main goal is to construct an increasing function  $g \in \xi \rightarrow \delta$  such that the composition  $f \circ g$  is still cofinal but also increasing.

#### definition

```
factor_body :: [i,i,i] ⇒ o where
  factor_body(β,h,x) ≡ (x ∈ δ ∧ j'β ≤ f'x ∧ (∀ α < β . f'(h'α) < f'x)) ∨ x = δ
```

#### definition

```
factor_rec :: [i,i] ⇒ i where
  factor_rec(β,h) ≡ μ x. factor_body(β,h,x)
```

$\text{factor\_rec}$  is the inductive step for the definition by transfinite recursion of the  $\text{factor}$  function (called  $g$  above), which in turn is obtained by minimizing the predicate  $\text{factor\_body}$ . Next we show that this predicate is monotonous.

#### lemma factor\_body\_mono:

assumes

```
β ∈ ξ α < β
  factor_body(β, λx ∈ β. G(x), x)
```

shows

```
factor_body(α, λx ∈ α. G(x), x)
```

#### proof -

from ⟨α < β⟩

have α ∈ β using ltD by simp

moreover

note ⟨β ∈ ξ⟩

moreover from calculation

have α ∈ ξ using ords ltD Ord\_cf Ord\_trans by blast

ultimately

have j'α ∈ j'β using j\_mono mono\_map\_increasing by blast

moreover from ⟨β ∈ ξ⟩

have j'β ∈ γ

using j\_mono domain\_of\_fun apply\_rangeI mono\_map\_is\_fun by force

moreover from this

have Ord(j'β)

```

using Ord_in_Ord ords Limit_is_Ord by auto
ultimately
have  $j'\alpha \leq j'\beta$  unfolding lt_def by blast
then
have  $j'\beta \leq f'\vartheta \implies j'\alpha \leq f'\vartheta$  for  $\vartheta$  using le_trans by blast
moreover
have  $f'((\lambda w \in \alpha. G(w)) 'y) < f'z$  if  $z \in \delta \ \forall x < \beta. f'((\lambda w \in \beta. G(w)) 'x) < f'z$   $y < \alpha$  for
 $y z$ 
proof -
  note  $\langle y < \alpha \rangle$ 
  also
  note  $\langle \alpha < \beta \rangle$ 
  finally
  have  $y < \beta$  by simp
  with  $\langle \forall x < \beta. f'((\lambda w \in \beta. G(w)) 'x) < f'z \rangle$ 
  have  $f'((\lambda w \in \beta. G(w)) 'y) < f'z$  by simp
  moreover from  $\langle y < \alpha \rangle \langle y < \beta \rangle$ 
  have  $(\lambda w \in \beta. G(w)) 'y = (\lambda w \in \alpha. G(w)) 'y$ 
    using beta_if by (auto dest: ltD)
  ultimately show ?thesis by simp
qed
moreover
note  $\langle \text{factor\_body}(\beta, \lambda x \in \beta. G(x), x) \rangle$ 
ultimately
show ?thesis
  unfolding factor_body_def by blast
qed

```

```

lemma factor_body_simp[simp]: factor_body( $\alpha, g, \delta$ )
  unfolding factor_body_def by simp

```

```

lemma factor_rec_mono:
  assumes
     $\beta \in \xi \ \alpha < \beta$ 
  shows
     $\text{factor\_rec}(\alpha, \lambda x \in \alpha. G(x)) \leq \text{factor\_rec}(\beta, \lambda x \in \beta. G(x))$ 
  unfolding factor_rec_def
  using assms ords factor_body_mono Least_antitone by simp

```

We now define the factor as higher-order function. Later it will be restricted to a set to obtain a bona fide function of type  $i$ .

#### definition

```

factor ::  $i \Rightarrow i$  where
factor( $\beta$ )  $\equiv$  transrec( $\beta, \text{factor\_rec}$ )

```

```

lemma factor_unfold:
  factor( $\alpha$ )  $=$  factor_rec( $\alpha, \lambda x \in \alpha. \text{factor}(x)$ )
  using def_transrec[OF factor_def] .

```

```

lemma factor_mono:
  assumes  $\beta \in \xi \alpha < \beta \text{ factor}(\alpha) \neq \delta \text{ factor}(\beta) \neq \delta$ 
  shows  $\text{factor}(\alpha) \leq \text{factor}(\beta)$ 
proof -
  have  $\text{factor}(\alpha) = \text{factor\_rec}(\alpha, \lambda x \in \alpha. \text{factor}(x))$ 
    using factor_unfold .
  also from assms and factor_rec_mono
  have ...  $\leq \text{factor\_rec}(\beta, \lambda x \in \beta. \text{factor}(x))$ 
    by simp
  also
    have  $\text{factor\_rec}(\beta, \lambda x \in \beta. \text{factor}(x)) = \text{factor}(\beta)$ 
      using def_transrec[OF factor_def, symmetric] .
  finally show ?thesis .
qed

```

The factor satisfies the predicate body of the minimization.

```

lemma factor_body_factor:
   $\text{factor\_body}(\alpha, \lambda x \in \alpha. \text{factor}(x), \text{factor}(\alpha))$ 
  using ords factor_unfold[of  $\alpha$ ]
    LeastI[of factor_body(., .)  $\delta$ ]
  unfolding factor_rec_def by simp

```

```

lemma factor_type [TC]:  $\text{Ord}(\text{factor}(\alpha))$ 
  using ords factor_unfold[of  $\alpha$ ]
    unfolding factor_rec_def by simp

```

The value  $\delta$  in *factor\_body* (and therefore, in *factor*) is meant to be a “default value”. Whenever it is not attained, the factor function behaves as expected: It is increasing and its composition with  $f$  also is.

```

lemma f_factor_increasing:
  assumes  $\beta \in \xi \alpha < \beta \text{ factor}(\beta) \neq \delta$ 
  shows  $f \cdot \text{factor}(\alpha) < f \cdot \text{factor}(\beta)$ 
proof -
  from assms
  have  $f \cdot ((\lambda x \in \beta. \text{factor}(x)) \cdot \alpha) < f \cdot \text{factor}(\beta)$ 
    using factor_unfold[of  $\beta$ ] ords LeastI[of factor_body( $\beta, \lambda x \in \beta. \text{factor}(x)$ )]
    unfolding factor_rec_def factor_body_def
    by (auto simp del:beta_if)
  with  $\langle \alpha < \beta \rangle$ 
  show ?thesis using ltD by auto
qed

```

```

lemma factor_increasing:
  assumes  $\beta \in \xi \alpha < \beta \text{ factor}(\alpha) \neq \delta \text{ factor}(\beta) \neq \delta$ 
  shows  $\text{factor}(\alpha) < \text{factor}(\beta)$ 
  using assms f_factor_increasing factor_mono by (force intro:le_neq_imp_lt)

```

```

lemma factor_in_delta:
  assumes  $\text{factor}(\beta) \neq \delta$ 

```

```

shows factor( $\beta$ )  $\in \delta$ 
using assms factor_body_factor ords
unfolding factor_body_def by auto

```

Finally, we define the (set) factor function as the restriction of factor to the ordinal  $\xi$ .

**definition**

```

fun_factor ::  $i$  where
fun_factor  $\equiv \lambda\beta\in\xi. \text{factor}(\beta)$ 

```

```

lemma fun_factor_is_mono_map:
assumes  $\bigwedge \beta. \beta \in \xi \implies \text{factor}(\beta) \neq \delta$ 
shows fun_factor  $\in \text{mono\_map}(\xi, \text{Memrel}(\xi), \delta, \text{Memrel}(\delta))$ 
unfolding mono_map_def
proof (intro CollectI ballI impI)

```

Proof that *fun\_factor* respects membership:

```

fix  $\alpha \beta$ 
assume  $\alpha \in \xi \beta \in \xi$ 
moreover
note assms
moreover from calculation
have  $\text{factor}(\alpha) \neq \delta \text{ factor}(\beta) \neq \delta \text{ Ord}(\beta)$ 
using factor_in_delta Ord_in_Ord ords by auto
moreover
assume  $\langle \alpha, \beta \rangle \in \text{Memrel}(\xi)$ 
ultimately
show  $\langle \text{fun\_factor } ' \alpha, \text{fun\_factor } ' \beta \rangle \in \text{Memrel}(\delta)$ 
unfolding fun_factor_def
using ltI factor_increasing[THEN ltD] factor_in_delta
by simp
next

```

Proof that it has the appropriate type:

```

from assms
show fun_factor :  $\xi \rightarrow \delta$ 
unfolding fun_factor_def
using ltI lam_type factor_in_delta by simp
qed

```

```

lemma f_fun_factor_is_mono_map:
assumes  $\bigwedge \beta. \beta \in \xi \implies \text{factor}(\beta) \neq \delta$ 
shows  $f \circ \text{fun\_factor} \in \text{mono\_map}(\xi, \text{Memrel}(\xi), \gamma, \text{Memrel}(\gamma))$ 
unfolding mono_map_def
using f_type
proof (intro CollectI ballI impI comp_fun[of __ __  $\delta$ ])
from assms
show fun_factor :  $\xi \rightarrow \delta$ 
using fun_factor_is_mono_map mono_map_is_fun by simp

```

Proof that  $f \circ fun\_factor$  respects membership

```

fix  $\alpha \beta$ 
assume  $\langle \alpha, \beta \rangle \in Memrel(\xi)$ 
then
have  $\alpha < \beta$ 
  using Ord_in_Ord[of  $\xi$ ] ltI ords by blast
assume  $\alpha \in \xi \beta \in \xi$ 
moreover from this and assms
have factor( $\alpha$ )  $\neq \delta$  factor( $\beta$ )  $\neq \delta$  by auto
moreover
have Ord( $\gamma$ )  $\gamma \neq 0$  using ords Limit_is_Ord by auto
moreover
note  $\langle \alpha < \beta \rangle \langle fun\_factor : \xi \rightarrow \delta \rangle$ 
ultimately
show  $\langle (f \circ fun\_factor) \alpha, (f \circ fun\_factor) \beta \rangle \in Memrel(\gamma)$ 
  using ltD[of  $f$  'factor( $\alpha$ ) f 'factor( $\beta$ )]
    f_factor_increasing apply_in_codomain_Ord f_type
    unfolding fun_factor_def by auto
qed

end — cofinal_factor

```

We state next the factorization lemma.

```

lemma cofinal_fun_factorization:
  notes le_imp_subset [dest] lt_trans2 [trans]
  assumes
    Ord( $\delta$ ) Limit( $\gamma$ )  $f : \delta \rightarrow \gamma$  cf_fun( $f, \gamma$ )
  shows
     $\exists g \in cf(\gamma) \rightarrow_{<} \delta. f \circ g : cf(\gamma) \rightarrow_{<} \gamma \wedge$ 
      cofinal_fun( $f \circ g, \gamma, Memrel(\gamma)$ )
proof -
  from ⟨Limit( $\gamma$ )⟩
  have Ord( $\gamma$ ) using Limit_is_Ord by simp
  then
  obtain  $j$  where  $j : cf(\gamma) \rightarrow_{<} \gamma$  cf_fun( $j, \gamma$ )
    using cofinal_mono_map_cf by blast
  then
  have domain( $j$ ) = cf( $\gamma$ )
    using domain_of_fun mono_map_is_fun by force
  from  $\langle j \in \_\rangle$  assms
  interpret cofinal_factor  $j \delta$  cf( $\gamma$ )
    by (unfold_locales) (simp_all)

```

The core of the argument is to show that the factor function indeed maps into  $\delta$ , therefore its values satisfy the first disjunct of *factor\_body*. This holds in turn because no restriction of the factor composed with  $f$  to a proper initial segment of  $cf(\gamma)$  can be cofinal in  $\gamma$  by definition of cofinality. Hence there must be a witness that satisfies the first disjunct.

```
have factor_not_delta: factor( $\beta$ ) ≠  $\delta$  if  $\beta \in cf(\gamma)$  for  $\beta$ 
```

For this, we induct on  $\beta$  ranging over  $cf(\gamma)$ .

```
proof (induct  $\beta$  rule:Ord_induct[OF _ Ord_cf[of  $\gamma$ ]])
  case 1 with that show ?case .
  next
    case (2  $\beta$ )
    then
      have IH:  $z \in \beta \implies factor(z) \neq \delta$  for  $z$  by simp
      define h where  $h \equiv \lambda x \in \beta. f'factor(x)$ 
      from IH
      have  $z \in \beta \implies factor(z) \in \delta$  for  $z$ 
        using factor_in_delta by blast
      with  $\langle f : \delta \rightarrow \gamma \rangle$ 
      have  $h : \beta \rightarrow \gamma$  unfolding h_def using apply_funtype lam_type by auto
      then
        have  $h : \beta \rightarrow_{<} \gamma$ 
          unfolding mono_map_def
      proof (intro CollectI ballI impI)
        fix  $x y$ 
        assume  $x \in \beta$   $y \in \beta$ 
        moreover from this and IH
        have  $factor(y) \neq \delta$  by simp
        moreover from calculation and  $\langle h \in \beta \rightarrow \gamma \rangle$ 
        have  $h'x \in \gamma$   $h'y \in \gamma$  by simp_all
        moreover from  $\langle \beta \in cf(\gamma) \rangle$  and  $\langle y \in \beta \rangle$ 
        have  $y \in cf(\gamma)$ 
          using Ord_trans Ord_cf by blast
        moreover from this
        have  $Ord(y)$ 
          using Ord_cf Ord_in_Ord by blast
        moreover
        assume  $\langle x, y \rangle \in Memrel(\beta)$ 
        moreover from calculation
        have  $x < y$  by (blast intro: ltI)
        ultimately
        show  $\langle h'x, h'y \rangle \in Memrel(\gamma)$ 
          unfolding h_def using f_factor_increasing ltD by (auto)
      qed
      with  $\langle \beta \in cf(\gamma) \rangle$   $\langle Ord(\gamma) \rangle$ 
      have ordertype( $h``\beta, Memrel(\gamma)$ ) =  $\beta$ 
        using mono_map_ordertype_image[of  $\beta$ ] Ord_cf Ord_in_Ord by blast
      also
      note  $\langle \beta \in cf(\gamma) \rangle$ 
      finally
      have ordertype( $h``\beta, Memrel(\gamma)$ ) ∈  $cf(\gamma)$  by simp
      moreover from  $\langle h \in \beta \rightarrow \gamma \rangle$ 
      have  $h``\beta \subseteq \gamma$ 
        using mono_map_is_fun Image_sub_codomain by blast
```

```

ultimately
have  $\neg \text{cofinal}(h``\beta, \gamma, \text{Memrel}(\gamma))$ 
  using ordertype_in_cf_imp_not_cofinal by simp
then
obtain  $\alpha_0$  where  $\alpha_0 \in \gamma \forall x \in h `` \beta. \neg \langle \alpha_0, x \rangle \in \text{Memrel}(\gamma) \wedge \alpha_0 \neq x$ 
  unfolding cofinal_def by auto
with  $\langle \text{Ord}(\gamma) \rangle \langle h `` \beta \subseteq \gamma \rangle$ 
have  $\forall x \in h `` \beta. x \in \alpha_0$ 
  using well_ord_Memrel[ $\gamma$ ] well_ord_is_linear[ $\gamma$  Memrel( $\gamma$ )]
  unfolding linear_def by blast
from  $\langle \alpha_0 \in \gamma \rangle \langle j \in \text{mono_map}(\_, \_, \gamma, \_) \rangle \langle \text{Ord}(\gamma) \rangle$ 
have  $j``\beta \in \gamma$ 
  using mono_map_is_fun_apply_in_codomain_Ord by force
with  $\langle \alpha_0 \in \gamma \rangle \langle \text{Ord}(\gamma) \rangle$ 
have  $\alpha_0 \cup j``\beta \in \gamma$ 
  using Un_least_mem_iff_Ord_in_Ord by auto
with  $\langle \text{cf}_\text{fun}(f, \gamma) \rangle$ 
obtain  $\vartheta$  where  $\vartheta \in \text{domain}(f) \langle \alpha_0 \cup j``\beta, f `` \vartheta \rangle \in \text{Memrel}(\gamma) \vee \alpha_0 \cup j``\beta = f `` \vartheta$ 
  by (auto simp add:cofinal_fun_def) blast
moreover from this and  $\langle f: \delta \rightarrow \gamma \rangle$ 
have  $\vartheta \in \delta$  using domain_of_fun by auto
moreover
note  $\langle \text{Ord}(\gamma) \rangle$ 
moreover from this and  $\langle f: \delta \rightarrow \gamma \rangle \langle \alpha_0 \in \gamma \rangle$ 
have  $\text{Ord}(f``\vartheta)$ 
  using apply_in_codomain_Ord_Ord_in_Ord by blast
moreover from calculation and  $\langle \alpha_0 \in \gamma \rangle$  and  $\langle \text{Ord}(\delta) \rangle$  and  $\langle j``\beta \in \gamma \rangle$ 
have  $\text{Ord}(\alpha_0) \text{Ord}(j``\beta) \text{Ord}(\vartheta)$ 
  using Ord_in_Ord by auto
moreover from  $\forall x \in h `` \beta. x \in \alpha_0 \langle \text{Ord}(\alpha_0) \rangle \langle h: \beta \rightarrow \gamma \rangle$ 
have  $x \in \beta \implies h``x < \alpha_0$  for  $x$ 
  using fun_is_function[ $h \beta \lambda_. \gamma$ ]
    Image_subset_Ord_imp_lt_domain_of_fun[ $h \beta \lambda_. \gamma$ ]
  by blast
moreover
have  $x \in \beta \implies h``x < f``\vartheta$  for  $x$ 
proof -
fix  $x$ 
assume  $x \in \beta$ 
with  $\forall x \in h `` \beta. x \in \alpha_0 \langle \text{Ord}(\alpha_0) \rangle \langle h: \beta \rightarrow \gamma \rangle$ 
have  $h``x < \alpha_0$ 
  using fun_is_function[ $h \beta \lambda_. \gamma$ ]
    Image_subset_Ord_imp_lt_domain_of_fun[ $h \beta \lambda_. \gamma$ ]
  by blast
also from  $\langle \alpha_0 \cup \_, f `` \vartheta \rangle \in \text{Memrel}(\gamma) \vee \alpha_0 \cup \_ = f `` \vartheta \rangle$ 
   $\langle \text{Ord}(f``\vartheta) \rangle \langle \text{Ord}(\alpha_0) \rangle \langle \text{Ord}(j``\beta) \rangle$ 
have  $\alpha_0 \leq f``\vartheta$ 
  using Un_leD1[OF leI [OF ltI]] Un_leD1[OF le_eqI] by blast

```

```

finally
show  $h^{\dot{x}} < f^{\dot{\vartheta}}$  .
qed
ultimately
have factor_body( $\beta, \lambda x \in \beta. \text{factor}(x), \vartheta$ )
  unfolding h_def factor_body_def using ltD
  by (auto dest: Un_memD2 Un_leD2[OF le_eqI])
with  $\langle \text{Ord}(\vartheta) \rangle$ 
have factor( $\beta$ )  $\leq \vartheta$ 
  using factor_unfold[of  $\beta$ ] Least_le unfolding factor_rec_def by auto
with  $\langle \vartheta \in \delta \rangle \langle \text{Ord}(\delta) \rangle$ 
have factor( $\beta$ )  $\in \delta$ 
  using leI[of  $\vartheta$ ] ltI[of  $\vartheta$ ] by (auto dest: ltD)
then
show ?case by (auto elim: mem_irrefl)
qed
moreover
have cofinal_fun(f O fun_factor,  $\gamma$ , Memrel( $\gamma$ ))
proof (intro cofinal_funI)
fix a
assume  $a \in \gamma$ 
with  $\langle cf\_fun(j, \gamma) \rangle \langle \text{domain}(j) = cf(\gamma) \rangle$ 
obtain x where  $x \in cf(\gamma)$   $a \in j^{\dot{x}} \vee a = j^{\dot{x}}$ 
  by (auto simp add: cofinal_fun_def) blast
with factor_not_delta
have  $x \in \text{domain}(f O \text{fun\_factor})$ 
  using f_fun_factor_is_mono_map mono_map_is_fun domain_of_fun by
force
moreover
have  $a \in (f O \text{fun\_factor})^{\dot{x}} \vee a = (f O \text{fun\_factor})^{\dot{x}}$ 
proof -
from  $\langle x \in cf(\gamma) \rangle$  factor_not_delta
have  $j^{\dot{x}} \leq f^{\dot{x}} \text{factor}(x)$ 
  using mem_not_refl factor_body_factor_factor_in_delta
  unfolding factor_body_def by auto
with  $\langle a \in j^{\dot{x}} \vee a = j^{\dot{x}} \rangle$ 
have  $a \in f^{\dot{x}} \text{factor}(x) \vee a = f^{\dot{x}} \text{factor}(x)$ 
  using ltD by blast
with  $\langle x \in cf(\gamma) \rangle$ 
show ?thesis using lam_funtype[of cf( $\gamma$ ) factor]
  unfolding fun_factor_def by auto
qed
moreover
note  $\langle a \in \gamma \rangle$ 
moreover from calculation and  $\langle \text{Ord}(\gamma) \rangle$  and factor_not_delta
have  $(f O \text{fun\_factor})^{\dot{x}} \in \gamma$ 
  using Limit_nonzero apply_in_codomain_Ord mono_map_is_fun[of f O
fun_factor]
  f_fun_factor_is_mono_map by blast

```

```

ultimately
show  $\exists x \in \text{domain}(f \circ \text{fun\_factor})$ .  $\langle a, (f \circ \text{fun\_factor})`x \rangle \in \text{Memrel}(\gamma)$ 
 $\quad \vee a = (f \circ \text{fun\_factor})`x$ 
by blast
qed
ultimately
show ?thesis
using fun_factor_is_mono_map f_fun_factor_is_mono_map by blast
qed

```

As a final observation in this part, we note that if the original cofinal map was increasing, then the factor function is also cofinal.

```

lemma factor_is_cofinal:
assumes
 $Ord(\delta) \ Ord(\gamma)$ 
 $f : \delta \rightarrow_{\leq} \gamma \ f \circ g \in \text{mono\_map}(\alpha, r, \gamma, \text{Memrel}(\gamma))$ 
 $\text{cofinal\_fun}(f \circ g, \gamma, \text{Memrel}(\gamma)) \ g : \alpha \rightarrow \delta$ 
shows
 $cf\_fun(g, \delta)$ 
unfolding cf_fun_def cofinal_fun_def
proof
fix a
assume  $a \in \delta$ 
with  $\langle f \in \text{mono\_map}(\delta, \_, \gamma, \_) \rangle$ 
have  $f`a \in \gamma$ 
using mono_map_is_fun by force
with  $\langle \text{cofinal\_fun}(f \circ g, \gamma, \_) \rangle$ 
obtain y where  $y \in \alpha \ \langle f`a, (f \circ g)`y \rangle \in \text{Memrel}(\gamma) \ \vee f`a = (f \circ g)`y$ 
unfolding cofinal_fun_def using domain_of_fun[ $OF \langle g : \alpha \rightarrow \delta \rangle$ ] by blast
with  $\langle g : \alpha \rightarrow \delta \rangle$ 
have  $\langle f`a, f`g`y \rangle \in \text{Memrel}(\gamma) \ \vee f`a = f`g`y$ 
using comp_fun_apply[of g alpha delta y f] by auto
with assms(1-3) and  $\langle a \in \delta \rangle$ 
have  $\langle a, g`y \rangle \in \text{Memrel}(\delta) \ \vee a = g`y$ 
using Memrel_mono_map_reflects Memrel_mono_map_is_inj[of delta f gamma gamma]
inj_apply_equality[of f delta gamma] by blast
with  $\langle y \in \alpha \rangle$ 
show  $\exists x \in \text{domain}(g)$ .  $\langle a, g`x \rangle \in \text{Memrel}(\delta) \ \vee a = g`x$ 
using domain_of_fun[ $OF \langle g : \alpha \rightarrow \delta \rangle$ ] by blast
qed

```

### 3.3 Classical results on cofinalities

Now the rest of the results follow in a more algebraic way. The next proof one invokes a case analysis on whether the argument is zero, a successor ordinal or a limit one; the last case being the most relevant one and is immediate from the factorization lemma.

```
lemma cf_le_domain_cofinal_fun:
```

```

assumes
   $Ord(\gamma) \ Ord(\delta) \ f:\delta \rightarrow \gamma \ cf\_fun(f,\gamma)$ 
shows
   $cf(\gamma) \leq \delta$ 
using assms
proof (cases rule:Ord_cases)
  case 0
  with ⟨Ord(δ)⟩
  show ?thesis using Ord_0_le by simp
next
  case (succ γ)
  with assms
  obtain x where  $x \in \delta \ f'x = \gamma$  using cf_fun_succ' by blast
  then
    have  $\delta \neq 0$  by blast
    let ?f = {⟨0, f'x⟩}
    from ⟨f'x = γ⟩
    have ?f :  $1 \rightarrow succ(\gamma)$ 
      using singleton_0 singleton_fun[of 0 γ] singleton_subsetI fun_weaken_type
    by simp
    with ⟨Ord(γ)⟩ ⟨f'x = γ⟩
    have  $cf(succ(\gamma)) = 1$  using cf_succ by simp
    with ⟨ $\delta \neq 0$ ⟩ succ
    show ?thesis using Ord_0_lt_iff_succ_leI ⟨Ord(δ)⟩ by simp
  next
    case (limit)
    with assms
    obtain g where  $g : cf(\gamma) \rightarrow_{<} \delta$ 
      using cofinal_fun_factorization by blast
    with assms
    show ?thesis using mono_map_imp_le by simp
qed

lemma cf_ordertype_cofinal:
assumes
   $Limit(\gamma) \ A \subseteq \gamma \ cofinal(A, \gamma, Memrel(\gamma))$ 
shows
   $cf(\gamma) = cf(ordertype(A, Memrel(\gamma)))$ 
proof (intro le_anti_sym)

```

We show the result by proving the two inequalities.

```

from ⟨Limit(γ)⟩
have Ord(γ)
  using Limit_is_Ord by simp
with ⟨A ⊆ γ⟩
have well_ord(A, Memrel(γ))
  using well_ord_Memrel well_ord_subset by blast
then
  obtain f α where  $f : \langle \alpha, Memrel(\alpha) \rangle \cong \langle A, Memrel(\gamma) \rangle \ Ord(\alpha) \ \alpha = order-$ 

```

```

type(A,Memrel(γ))
  using ordertype_ord_iso Ord_ordertype ord_iso_sym by blast
  moreover from this
  have f: α → A
    using ord_iso_is_mono_map mono_map_is_fun[OF f Memrel(α)] by blast
  moreover from this
  have function(f)
    using fun_is_function by simp
  moreover from ⟨f:⟨α, Memrel(α)⟩ ≈ ⟨A,Memrel(γ)⟩⟩
  have range(f) = A
    using ord_iso_is_bij bij_is_surj surj_range by blast
  moreover note ⟨cofinal(A,γ,_)⟩
  ultimately
  have cf_fun(f,γ)
    using cofinal_range_iff_cofinal_fun by blast
  moreover from ⟨Ord(α)⟩
  obtain h where h :cf(α) →< α cf_fun(h,α)
    using cofinal_mono_map_cf by blast
  moreover from ⟨Ord(γ)⟩
  have trans(Memrel(γ))
    using trans_Memrel by simp
  moreover
  note ⟨A ⊆ γ⟩
  ultimately
  have cofinal_fun(f O h,γ,Memrel(γ))
    using cofinal_comp_ord_iso_is_mono_map[OF ⟨f:⟨α,_)⟩ ≈ ⟨A,_)⟩] mono_map_is_fun
      mono_map_mono by blast
  moreover from ⟨f:α→A⟩ ⟨A ⊆ γ⟩ ⟨h∈mono_map(cf(α),_,α,_)⟩
  have f O h : cf(α) → γ
    using Pi_mono[of A γ] comp_fun mono_map_is_fun by blast
  moreover
  note ⟨Ord(γ)⟩ ⟨Ord(α)⟩ ⟨α = ordertype(A,Memrel(γ))⟩
  ultimately
  show cf(γ) ≤ cf(ordertype(A,Memrel(γ)))
    using cf_le_domain_cofinal_fun[OF _ _ f O h]
      by (auto simp add:cf_fun_def)

```

That finishes the first inequality. Now we go the other side.

```

from ⟨f:⟨α, _)⟩ ≈ ⟨A,_)⟩ ⟨A ⊆ γ⟩
have f :α →< γ
  using mono_map_mono[OF ord_iso_is_mono_map] by simp
then
have f: α → γ
  using mono_map_is_fun by simp
with ⟨cf_fun(f,γ)⟩ ⟨Limit(γ)⟩ ⟨Ord(α)⟩
obtain g where g :cf(γ) →< α
  f O g :cf(γ) →< γ
  cofinal_fun(f O g,γ,Memrel(γ))
  using cofinal_fun_factorization by blast

```

```

moreover from this
have  $g:cf(\gamma) \rightarrow \alpha$ 
  using mono_map_is_fun by simp
moreover
note  $\langle Ord(\alpha) \rangle$ 
moreover from calculation and  $\langle f : \alpha \rightarrow_{<} \gamma \rangle \langle Ord(\gamma) \rangle$ 
have cf_fun(g,  $\alpha$ )
  using factor_is_cofinal by blast
moreover
note  $\langle \alpha = ordertype(A, Memrel(\gamma)) \rangle$ 
ultimately
show  $cf(ordertype(A, Memrel(\gamma))) \leq cf(\gamma)$ 
  using cf_le_domain_cofinal_fun[OF _ Ord_cf mono_map_is_fun] by simp
qed

lemma cf_idemp:
assumes Limit( $\gamma$ )
shows  $cf(\gamma) = cf(cf(\gamma))$ 
proof -
  from assms
  obtain A where  $A \subseteq \gamma$  cofinal( $A, \gamma, Memrel(\gamma)$ )  $cf(\gamma) = ordertype(A, Memrel(\gamma))$ 
    using Limit_is_Ord cf_is_ordertype by blast
  with assms
  have  $cf(\gamma) = cf(ordertype(A, Memrel(\gamma)))$  using cf_ordertype_cofinal by simp
  also
  have ... =  $cf(cf(\gamma))$ 
    using  $\langle cf(\gamma) = ordertype(A, Memrel(\gamma)) \rangle$  by simp
  finally
  show  $cf(\gamma) = cf(cf(\gamma))$  .
qed

lemma cf_le_cardinal:
assumes Limit( $\gamma$ )
shows  $cf(\gamma) \leq |\gamma|$ 
proof -
  from assms
  have  $\langle Ord(\gamma) \rangle$  using Limit_is_Ord by simp
  then
  obtain f where  $f \in surj(|\gamma|, \gamma)$ 
    using Ord_cardinal_eqpoll unfolding eqpoll_def bij_def by blast
  with  $\langle Ord(\gamma) \rangle$ 
  show ?thesis
    using Card_is_Ord[OF Card_cardinal] surj_is_cofinal
      cf_le_domain_cofinal_fun[of  $\gamma$ ] surj_is_fun by blast
qed

lemma regular_is_Card:
notes le_imp_subset [dest]
assumes Limit( $\gamma$ )  $\gamma = cf(\gamma)$ 

```

```

shows Card( $\gamma$ )
proof -
  from assms
  have  $|\gamma| \subseteq \gamma$ 
    using Limit_is_Ord Ord_cardinal_le by blast
  also from  $\langle \gamma = cf(\gamma) \rangle$ 
  have  $\gamma \subseteq cf(\gamma)$  by simp
  finally
  have  $|\gamma| \subseteq cf(\gamma)$  .
  with assms
  show ?thesis unfolding Card_def using cf_le_cardinal by force
qed

lemma Limit_cf: assumes Limit( $\kappa$ ) shows Limit( $cf(\kappa)$ )
  using Ord_cf[of  $\kappa$ , THEN Ord_cases]
  —  $cf(\kappa)$  being 0 or successor leads to contradiction
proof (cases)
  case 1
  with  $\langle Limit(\kappa) \rangle$ 
  show ?thesis using cf_zero_iff Limit_is_Ord by simp
next
  case (?  $\alpha$ )
  moreover
  note  $\langle Limit(\kappa) \rangle$ 
  moreover from calculation
  have  $cf(\kappa) = 1$ 
    using cf_idemp cf_succ by fastforce
  ultimately
  show ?thesis
    using succ_LimitE cf_eq_one_iff Limit_is_Ord
    by auto
qed

lemma InfCard_cf: Limit( $\kappa$ )  $\implies$  InfCard( $cf(\kappa)$ )
  using regular_is_Card cf_idemp Limit_cf nat_le_Limit Limit_cf
  unfolding InfCard_def by simp

lemma cf_le_cf_fun:
  notes [dest] = Limit_is_Ord
  assumes  $cf(\kappa) \leq \nu$  Limit( $\kappa$ )
  shows  $\exists f. f:\nu \rightarrow \kappa \wedge cf\_fun(f, \kappa)$ 
proof -
  note assms
  moreover from this
  obtain  $h$  where  $h\_cofinal\_mono: cf\_fun(h, \kappa)$ 
     $h : cf(\kappa) \rightarrow_{<} \kappa$ 
     $h : cf(\kappa) \rightarrow \kappa$ 
  using cofinal_mono_map_cf mono_map_is_fun by force
  moreover from calculation

```

```

obtain g where  $g \in inj(cf(\kappa), \nu)$ 
  using le_imp_lepoll by blast
from this and calculation(2,3,5)
obtain f where  $f \in surj(\nu, cf(\kappa))$   $f: \nu \rightarrow cf(\kappa)$ 
  using inj_imp_surj[OF _ Limit_has_0[THEN ltD]]
    surj_is_fun Limit_cf by blast
moreover from this
have cf_fun(f,cf( $\kappa$ ))
  using surj_is_cofinal by simp
moreover
note h_cofinal_mono `Limit( $\kappa$ )'
moreover from calculation
have cf_fun(h O f, $\kappa$ )
  using cf_fun_comp by blast
moreover from calculation
have  $h O f \in \nu \rightarrow \kappa$ 
  using comp_fun by simp
ultimately
show ?thesis by blast
qed

```

```

lemma Limit_cofinal_fun_lt:
notes [dest] = Limit_is_Ord
assumes Limit( $\kappa$ )  $f: \nu \rightarrow \kappa$  cf_fun(f, $\kappa$ )  $n \in \kappa$ 
shows  $\exists \alpha \in \nu. n < f'\alpha$ 
proof -
  from `Limit( $\kappa$ )`  $\langle n \in \kappa \rangle$ 
  have succ(n)  $\in \kappa$ 
    using Limit_has_succ[OF _ ltI, THEN ltD] by auto
  moreover
  note `f: \nu \rightarrow \_\_` 
  moreover from this
  have domain(f) =  $\nu$ 
    using domain_of_fun by simp
  moreover
  note `cf_fun(f,\kappa)` 
  ultimately
  obtain  $\alpha$  where  $\alpha \in \nu$  succ(n)  $\in f'\alpha \vee succ(n) = f' \alpha$ 
    using cf_funD[THEN cofinal_funD] by blast
  moreover from this
  consider (1) succ(n)  $\in f'\alpha$  | (2) succ(n) =  $f' \alpha$ 
    by blast
  then
  have  $n < f'\alpha$ 
  proof (cases)
    case 1
    moreover
    have  $n \in succ(n)$  by simp
    moreover

```

```

note <Limit( $\kappa$ )> < $f: \nu \rightarrow \_\_$ > < $\alpha \in \nu$ >
moreover from this
have  $Ord(f` \alpha)$ 
    using apply_type[of  $f$   $\nu$   $\lambda\_. \kappa$ , THEN [2] Ord_in_Ord]
    by blast
ultimately
show ?thesis
    using Ord_trans[of  $n$  succ( $n$ )  $f` \alpha$ ] ltI by blast
next
    case 2
    have  $n \in f` \alpha$  by (simp add:2[symmetric])
    with <Limit( $\kappa$ )> < $f: \nu \rightarrow \_\_$ > < $\alpha \in \nu$ >
    show ?thesis
        using ltI
        apply type[of  $f$   $\nu$   $\lambda\_. \kappa$ , THEN [2] Ord_in_Ord]
        by blast
    qed
    ultimately
    show ?thesis by blast
qed

context
includes Ord_dests and Aleph_dests and Aleph_intros and Aleph_mem_dests
and mono_map_rules
begin

```

We end this section by calculating the cofinality of Alephs, for the zero and limit case. The successor case depends on *AC*.

```

lemma cf_nat:  $cf(\omega) = \omega$ 
    using Limit_nat[THEN InfCard_cf] cf_le_cardinal[of  $\omega$ ]
    Card_nat[THEN Card_cardinal_eq] le_anti_sym
    unfolding InfCard_def by auto

lemma cf_Aleph_zero:  $cf(\aleph_0) = \aleph_0$ 
    using cf_nat unfolding Aleph_def by simp

lemma cf_Aleph_Limit:
    assumes Limit( $\gamma$ )
    shows  $cf(\aleph_\gamma) = cf(\gamma)$ 
proof -
    note <Limit( $\gamma$ )>
    moreover from this
    have  $(\lambda x \in \gamma. \aleph_x) : \gamma \rightarrow \aleph_\gamma$  (is ? $f : \_\_ \rightarrow \_\_$ )
        using lam_funtype[of _ Aleph] fun_weaken_type[of _ _ _  $\aleph_\gamma$ ] by blast
    moreover from <Limit( $\gamma$ )>
    have  $x \in y \implies \aleph_x \in \aleph_y$  (if  $x \in \gamma$   $y \in \gamma$  for  $x$   $y$ )
        using that Ord_in_Ord[of  $\gamma$ ] Ord_trans[of _ _  $\gamma$ ] by blast
    ultimately
    have ? $f \in$  mono_map( $\gamma$ , Memrel( $\gamma$ ),  $\aleph_\gamma$ , Memrel( $\aleph_\gamma$ ))

```

```

    by auto
  with ‹Limit(γ)›
  have ?f ∈ ⟨γ, Memrel(γ)⟩ ≈ ⟨?f“γ, Memrel(ℵ_γ)⟩
    using mono_map_imp_ord_iso_Memrel[of γ ℵ_γ ?f]
      Card_Aleph
    by blast
  then
    have converse(?f) ∈ ⟨?f“γ, Memrel(ℵ_γ)⟩ ≈ ⟨γ, Memrel(γ)⟩
      using ord_iso_sym by simp
    with ‹Limit(γ)›
    have ordertype(?f“γ, Memrel(ℵ_γ)) = γ
      using ordertype_eq[OF _ well_ord_Memrel]
        ordertype_Memrel by auto
    moreover from ‹Limit(γ)›
    have cofinal(?f“γ, ℵ_γ, Memrel(ℵ_γ))
      unfolding cofinal_def
      proof (standard, intro ballI)
        fix a
        assume a ∈ ℵ_γ ℵ_γ = (⋃ i < γ. ℵ_i)
        moreover from this
        obtain i where i < γ a ∈ ℵ_i
          by auto
        moreover from this and ‹Limit(γ)›
        have Ord(i) using ltD Ord_in_Ord by blast
        moreover from ‹Limit(γ)› and calculation
        have succ(i) ∈ γ using ltD by auto
        moreover from this and ‹Ord(i)›
        have ℵ_i < ℵ_succ(i)
          by (auto)
        ultimately
        have ⟨a, ℵ_i⟩ ∈ Memrel(ℵ_γ)
          using ltD by (auto dest:Aleph_increasing)
        moreover from ‹i < γ›
        have ℵ_i ∈ ?f“γ
          using ltD apply_in_image[OF _ _ → _] by auto
        ultimately
        show ∃ x ∈ ?f “ γ. ⟨a, x⟩ ∈ Memrel(ℵ_γ) ∨ a = x by blast
      qed
    moreover
    note ‹?f: γ → ℵ_γ› ‹Limit(γ)›
    ultimately
    show cf(ℵ_γ) = cf(γ)
      using cf_ordertype_cofinal[OF Limit_Aleph_Image_sub_codomain, of γ ?f γ
      ]
      Limit_is_Ord by simp
  qed
end — includes

```

```
end
```

## 4 Cardinal Arithmetic under Choice

```
theory Cardinal_Library
imports
  ZF_Library
  ZF.Cardinal_AC
```

```
begin
```

This theory includes results on cardinalities that depend on  $AC$

### 4.1 Results on cardinal exponentiation

Non trivial instances of cardinal exponentiation require that the relevant function spaces are well-ordered, hence this implies a strong use of choice.

```
lemma cexp_eqpoll_cong:
  assumes
     $A \approx A' B \approx B'$ 
  shows
     $A^B = A'^{B'}$ 
  unfolding cexp_def using cardinal_eqpoll_iff
    function_space_eqpoll_cong assms
  by simp

lemma cexp_cexp_cmult:  $(\kappa^{\nu_1})^{\nu_2} = \kappa^{\nu_2 \times \nu_1}$ 
proof -
  have  $(\kappa^{\nu_1})^{\nu_2} = (\nu_1 \rightarrow \kappa)^{\nu_2}$ 
  using cardinal_eqpoll
  by (intro cexp_eqpoll_cong) (simp_all add:cexp_def)
  also
  have ... =  $\kappa^{\nu_2 \times \nu_1}$ 
  unfolding cexp_def using curry_eqpoll cardinal_cong by blast
  also
  have ... =  $\kappa^{\nu_2} \otimes \nu_1$ 
  using cardinal_eqpoll[THEN eqpoll_sym]
  unfolding cmult_def by (intro cexp_eqpoll_cong) (simp)
  finally
  show ?thesis .
qed

lemma cardinal_Pow:  $|Pow(X)| = 2^{|X|}$  — Perhaps it's better with  $|X|$ 
using cardinal_eqpoll_iff[THEN iffD2, OF Pow_eqpoll_function_space]
unfolding cexp_def by simp

lemma cantor_cexp:
  assumes Card( $\nu$ )
```

```

shows  $\nu < 2^{\uparrow\nu}$ 
using assms Card_is_Ord Card_cexp
proof (intro not_le_iff_lt[THEN iffD1] notI)
assume  $2^{\uparrow\nu} \leq \nu$ 
then
have  $|Pow(\nu)| \leq \nu$ 
using cardinal_Pow by simp
with assms
have  $Pow(\nu) \lesssim \nu$ 
using cardinal_eqpoll_iff Card_le_imp_lepoll Card_cardinal_eq
by auto
then
obtain g where  $g \in inj(Pow(\nu), \nu)$ 
by blast
then
show False
using cantor_inj by simp
qed simp

lemma cexp_left_mono:
assumes  $\kappa_1 \leq \kappa_2$ 
shows  $\kappa_1^{\uparrow\nu} \leq \kappa_2^{\uparrow\nu}$ 

proof -
from assms
have  $\kappa_1 \subseteq \kappa_2$ 
using le_subset_iff by simp
then
have  $\nu \rightarrow \kappa_1 \subseteq \nu \rightarrow \kappa_2$ 
using Pi_weaken_type by auto
then
show ?thesis unfolding cexp_def
using lepoll_imp_cardinal_le_subset_imp_lepoll by simp
qed

lemma cantor_cexp':
assumes  $2 \leq \kappa$  Card( $\nu$ )
shows  $\nu < \kappa^{\uparrow\nu}$ 
using cexp_left_mono assms cantor_cexp_lt_trans2 by blast

lemma InfCard_cexp:
assumes  $2 \leq \kappa$  InfCard( $\nu$ )
shows InfCard( $\kappa^{\uparrow\nu}$ )
using assms cantor_cexp'[THEN leI] le_trans Card_cexp
unfolding InfCard_def by auto

lemmas InfCard_cexp' = InfCard_cexp[OF nats_le_InfCard, simplified]
—  $\llbracket InfCard(\kappa); InfCard(\nu) \rrbracket \implies InfCard(\kappa^{\uparrow\nu})$ 

```

## 4.2 Miscellaneous

```

lemma cardinal_RepFun_le:  $|\{f(a) . a \in A\}| \leq |A|$ 
proof -
  have  $(\lambda x \in A. f(x)) \in surj(A, \{f(a) . a \in A\})$ 
    unfolding surj_def using lam_funtypes by auto
  then
  show ?thesis
    using surj_implies_cardinal_le by blast
qed

lemma subset_imp_le_cardinal:  $A \subseteq B \implies |A| \leq |B|$ 
  using subset_imp_lepoll[THEN lepoll_imp_cardinal_le] .

lemma lt_cardinal_imp_not_subset:  $|A| < |B| \implies \neg B \subseteq A$ 
  using subset_imp_le_cardinal_le_imp_not_lt by blast

lemma cardinal_lt_csucc_iff:  $Card(K) \implies |K'| < K^+ \longleftrightarrow |K'| \leq K$ 
  by (simp add: Card_lt_csucc_iff)

lemma cardinal_UN_le_nat:
   $(\bigwedge i. i \in \omega \implies |X(i)| \leq \omega) \implies |\bigcup_{i \in \omega} X(i)| \leq \omega$ 
  by (simp add: cardinal_UN_le InfCard_nat)

lemma lepoll_imp_cardinal_UN_le:
  notes [dest] = InfCard_is_Card Card_is_Ord
  assumes InfCard(K)  $J \lesssim K \wedge \bigwedge i. i \in J \implies |X(i)| \leq K$ 
  shows  $|\bigcup_{i \in J} X(i)| \leq K$ 
proof -
  from  $\langle J \lesssim K \rangle$ 
  obtain f where  $f \in inj(J, K)$  by blast
  define Y where  $Y(k) \equiv \text{if } k \in \text{range}(f) \text{ then } X(\text{converse}(f)^k) \text{ else } 0$  for k
  have  $i \in J \implies f'i \in K$  for i
    using inj_is_fun[OF f ∈ inj(J, K)] by auto
  have  $(\bigcup_{i \in J} X(i)) \subseteq (\bigcup_{i \in K} Y(i))$ 
  proof (standard, elim UN_E)
    fix x i
    assume  $i \in J$   $x \in X(i)$ 
    with  $\langle f \in inj(J, K) \rangle$   $\langle i \in J \implies f'i \in K \rangle$ 
    have  $x \in Y(f'i)$   $f'i \in K$ 
      unfolding Y_def
      using inj_is_fun[OF f ∈ inj(J, K)]
        right_inverse apply_rangeI by auto
    then
    show  $x \in (\bigcup_{i \in K} Y(i))$  by auto
  qed
  have  $|\bigcup_{i \in J} X(i)| \leq |\bigcup_{i \in K} Y(i)|$ 
    unfolding Y_def using subset_imp_le_cardinal by simp
    with assms  $\langle \bigwedge i. i \in J \implies f'i \in K \rangle$ 

```

```

show  $\bigcup_{i \in J} X(i) \leq K$ 
  using inj_converse_fun[ $f \in inj(J, K)$ ] unfolding Y_def
  by (rule_tac le_trans[OF _ cardinal_UN_le]) (auto intro:Ord_0_le)+

qed

— For backwards compatibility
lemmas lepoll_imp_cardinal_UN_le = lepoll_imp_cardinal_UN_le

lemma cardinal_lt_csucc_iff':
  includes Ord_dests
  assumes Card( $\kappa$ )
  shows  $\kappa < |X| \longleftrightarrow \kappa^+ \leq |X|$ 
  using assms cardinal_lt_csucc_iff[of  $\kappa$  X] Card_csucc[of  $\kappa$ ]
  not_le_iff_lt[of  $\kappa^+ |X|$ ] not_le_iff_lt[of  $|X| \kappa$ ]
  by blast

lemma lepoll_imp_subset_bij:  $X \lesssim Y \longleftrightarrow (\exists Z. Z \subseteq Y \wedge Z \approx X)$ 
proof
  assume  $X \lesssim Y$ 
  then
  obtain j where  $j \in inj(X, Y)$ 
    by blast
  then
  have range(j)  $\subseteq Y$   $j \in bij(X, range(j))$ 
    using inj_bij_range inj_is_fun range_fun_subset_codomain
    by blast+
  then
  show  $\exists Z. Z \subseteq Y \wedge Z \approx X$ 
    using eqpoll_sym unfolding eqpoll_def
    by force
next
  assume  $\exists Z. Z \subseteq Y \wedge Z \approx X$ 
  then
  obtain f where  $f \in bij(Z, X)$   $Z \subseteq Y$ 
    unfolding eqpoll_def by force
  then
  have converse(f)  $\in inj(X, Y)$ 
    using bij_is_inj inj_weaken_type bij_converse_bij by blast
  then
  show  $X \lesssim Y$  by blast
qed

```

The following result proves to be very useful when combining *cardinal* and  $(\approx)$  in a calculation.

```

lemma cardinal_Card_eqpoll_iff:  $Card(\kappa) \implies |X| = \kappa \longleftrightarrow X \approx \kappa$ 
  using Card_cardinal_eq[of  $\kappa$ ] cardinal_eqpoll_iff[of X  $\kappa$ ] by auto
  — Compare le_Card_iff

```

```

lemma lepoll_imp_lepoll_cardinal: assumes  $X \lesssim Y$  shows  $X \lesssim |Y|$ 

```

```

using assms cardinal_Card_eqpoll_iff[of |Y| Y]
  lepoll_eq_trans[of __ |Y|] by simp

lemma lepoll_Un:
  assumes InfCard( $\kappa$ )  $A \lesssim \kappa$   $B \lesssim \kappa$ 
  shows  $A \cup B \lesssim \kappa$ 
proof -
  have  $A \cup B \lesssim \text{sum}(A, B)$ 
    using Un_lepoll_sum .
  moreover
  note assms
  moreover from this
  have  $|\text{sum}(A, B)| \leq \kappa \oplus \kappa$ 
    using sum_lepoll_mono[of A  $\kappa$  B  $\kappa$ ] lepoll_imp_cardinal_le
    unfolding cadd_def by auto
  ultimately
  show ?thesis
    using InfCard_cdouble_eq Card_cardinal_eq
      InfCard_is_Card Card_le_imp_lepoll[of sum(A, B)  $\kappa$ ]
        lepoll_trans[of A  $\cup$  B]
    by auto
qed

```

```

lemma cardinal_Un_le:
  assumes InfCard( $\kappa$ )  $|A| \leq \kappa$   $|B| \leq \kappa$ 
  shows  $|A \cup B| \leq \kappa$ 
  using assms lepoll_Un_le_Card_iff InfCard_is_Card by auto

```

This is the unconditional version under choice of *Cardinal.Finite\_cardinal\_iff*.

```

lemma Finite_cardinal_iff': Finite(|i|)  $\longleftrightarrow$  Finite(i)
  using cardinal_eqpoll_iff eqpoll_imp_Finite_iff by fastforce

```

```

lemma cardinal_subset_of_Card:
  assumes Card( $\gamma$ )  $a \subseteq \gamma$ 
  shows  $|a| < \gamma \vee |a| = \gamma$ 
proof -
  from assms
  have  $|a| < |\gamma| \vee |a| = |\gamma|$ 
    using subset_imp_le_cardinal_le_iff by simp
  with assms
  show ?thesis
    using Card_cardinal_eq by simp
qed

```

```

lemma cardinal_cases:
  includes Ord_dests
  shows Card( $\gamma$ )  $\implies |X| < \gamma \longleftrightarrow \neg |X| \geq \gamma$ 
  using not_le_iff_lt
  by auto

```

### 4.3 Countable and uncountable sets

```

lemma countable_iff_cardinal_le_nat: countable(X)  $\longleftrightarrow |X| \leq \omega$ 
  using le_Card_iff[of  $\omega$  X] Card_nat
  unfolding countable_def by simp

lemma lepoll_countable:  $X \lesssim Y \implies \text{countable}(Y) \implies \text{countable}(X)$ 
  using lepoll_trans[of X Y] by blast

— Next lemma can be proved without using AC
lemma surj_countable: countable(X)  $\implies f \in \text{surj}(X, Y) \implies \text{countable}(Y)$ 
  using surj_implies_cardinal_le[of f X Y, THEN le_trans]
  countable_iff_cardinal_le_nat by simp

lemma Finite_imp_countable: Finite(X)  $\implies \text{countable}(X)$ 
  unfolding Finite_def
  by (auto intro:InfCard_nat nats_le_InfCard[of _  $\omega$ ,
    THEN le_imp_lepoll] dest!:eq_lepoll_trans[of X _  $\omega$ ])

lemma countable_imp_countable_UN:
  assumes countable(J)  $\wedge \forall i \in J. \text{countable}(X(i))$ 
  shows countable( $\bigcup_{i \in J} X(i)$ )
  using assms lepoll_imp_cardinal_UN_le[of  $\omega$  J X] InfCard_nat
  countable_iff_cardinal_le_nat
  by auto

lemma countable_union_countable:
  assumes  $\forall x. x \in C \implies \text{countable}(x)$  countable(C)
  shows countable( $\bigcup C$ )
  using assms countable_imp_countable_UN[of C  $\lambda x. x$ ] by simp

abbreviation
  uncountable ::  $i \Rightarrow o$  where
  uncountable(X)  $\equiv \neg \text{countable}(X)$ 

lemma uncountable_iff_nat_lt_cardinal:
  uncountable(X)  $\longleftrightarrow \omega < |X|$ 
  using countable_iff_cardinal_le_nat not_le_iff_lt by simp

lemma uncountable_not_empty: uncountable(X)  $\implies X \neq 0$ 
  using empty_lepollI by auto

lemma uncountable_imp_Infinite: uncountable(X)  $\implies \text{Infinite}(X)$ 
  using uncountable_iff_nat_lt_cardinal[of X] lepoll_nat_imp_Infinite[of X]
  cardinal_le_imp_lepoll[of  $\omega$  X] leI
  by simp

lemma uncountable_not_subset_countable:
  assumes countable(X) uncountable(Y)
  shows  $\neg (Y \subseteq X)$ 

```

```

using assms lepoll_trans subset_imp_lepoll[of Y X]
by blast

```

#### 4.4 Results on Alephs

```

lemma nat_lt_Aleph1:  $\omega < \aleph_1$ 
  by (simp add: Aleph_def lt_csucc)

```

```

lemma zero_lt_Aleph1:  $0 < \aleph_1$ 
  by (rule lt_trans[of _  $\omega$ ], auto simp add: ltI nat_lt_Aleph1)

```

```

lemma le_aleph1_nat:  $\text{Card}(k) \implies k < \aleph_1 \implies k \leq \omega$ 
  by (simp add: Aleph_def Card_lt_csucc_iff Card_nat)

```

```

lemma Aleph_succ:  $\aleph_{\text{succ}(\alpha)} = \aleph_\alpha^+$ 
  unfolding Aleph_def by simp

```

```

lemma lesspoll_aleph_plus_one:
  assumes Ord( $\alpha$ )
  shows  $d \prec \aleph_{\text{succ}(\alpha)} \longleftrightarrow d \lesssim \aleph_\alpha$ 
  using assms lesspoll_csucc Aleph_succ Card_is_Ord by simp

```

```

lemma cardinal_Aleph [simp]:  $\text{Ord}(\alpha) \implies |\aleph_\alpha| = \aleph_\alpha$ 
  using Card_cardinal_eq by simp

```

— Could be proved without using AC

```

lemma Aleph_lesspoll_increasing:
  includes Aleph_intros
  shows  $a < b \implies \aleph_a \prec \aleph_b$ 
  using cardinal_lt_iff_lesspoll[of  $\aleph_a \aleph_b$ ] Card_cardinal_eq[of  $\aleph_b$ ]
    lt_Ord lt_Ord2 Card_Aleph[THEN Card_is_Ord] by auto

```

```

lemma uncountable_iff_subset_eqpoll_Aleph1:
  includes Ord_dests
  notes Aleph_zero_eq_nat[simp] Card_nat[simp] Aleph_succ[simp]
  shows uncountable( $X$ )  $\longleftrightarrow (\exists S. S \subseteq X \wedge S \approx \aleph_1)$ 
proof
  assume uncountable( $X$ )
  then
  have  $\aleph_1 \lesssim X$ 
    using uncountable_iff_nat_lt_cardinal_cardinal_lt_csucc_iff'
      cardinal_le_imp_lepoll by force
  then
  obtain  $S$  where  $S \subseteq X$   $S \approx \aleph_1$ 
    using lepoll_imp_subset_bij by auto
  then
  show  $\exists S. S \subseteq X \wedge S \approx \aleph_1$ 
    using cardinal_cong Card_csucc[of  $\omega$ ] Card_cardinal_eq by auto
next

```

```

assume  $\exists S. S \subseteq X \wedge S \approx \aleph_1$ 
then
have  $\aleph_1 \lesssim X$ 
using subset_imp_lepoll[THEN [2] eq_lepoll_trans, of  $\aleph_1$  _  $X$ ,
    OF eqpoll_sym] by auto
then
show uncountable( $X$ )
using Aleph_lesspoll_increasing[of 0 1, THEN [2] lesspoll_trans1,
    of  $\aleph_1$ ] lepoll_trans[of  $\aleph_1$   $X$   $\omega$ ]
by auto
qed

lemma lt_Aleph_imp_cardinal_UN_le_nat: function( $G$ )  $\implies$  domain( $G$ )  $\lesssim \omega$ 
 $\implies$ 
 $\forall n \in \text{domain}(G). |G^n| < \aleph_1 \implies |\bigcup_{n \in \text{domain}(G)} G^n| \leq \omega$ 
proof -
assume function( $G$ )
let ?N = domain( $G$ ) and ?R =  $\bigcup_{n \in \text{domain}(G)} G^n$ 
assume ?N  $\lesssim \omega$ 
assume Eq1:  $\forall n \in ?N. |G^n| < \aleph_1$ 
{
  fix  $n$ 
  assume  $n \in ?N$ 
  with Eq1 have  $|G^n| \leq \omega$ 
  using le_aleph1_nat by simp
}
then
have  $n \in ?N \implies |G^n| \leq \omega$  for  $n$  .
with  $\langle ?N \lesssim \omega \rangle$ 
show ?thesis
using InfCard_nat lepoll_imp_cardinal_UN_le by simp
qed

lemma Aleph1_eq_cardinal_vimage:  $f: \aleph_1 \rightarrow \omega \implies \exists n \in \omega. |f^{-1}\{n\}| = \aleph_1$ 
proof -
assume  $f: \aleph_1 \rightarrow \omega$ 
then
have function( $f$ ) domain( $f$ ) =  $\aleph_1$  range( $f$ )  $\subseteq \omega$ 
by (simp_all add: domain_of_fun_fun_is_function range_fun_subset_codomain)
let ?G =  $\lambda n \in \text{range}(f). f^{-1}\{n\}$ 
from  $\langle f: \aleph_1 \rightarrow \omega \rangle$ 
have range( $f$ )  $\subseteq \omega$  by (simp add: range_fun_subset_codomain)
then
have domain(?G)  $\lesssim \omega$ 
using subset_imp_lepoll by simp
have function(?G) by (simp add: function_lam)
from  $\langle f: \aleph_1 \rightarrow \omega \rangle$ 
have  $n \in \omega \implies f^{-1}\{n\} \subseteq \aleph_1$  for  $n$ 
using Pi_vimage_subset by simp

```

```

with ⟨range(f) ⊆ ω⟩
have ℙ₁ = (⋃ n ∈ range(f). f⁻“{n})
proof (intro equalityI, intro subsetI)
fix x
assume x ∈ ℙ₁
with ⟨f: ℙ₁ → ω⟩ ⟨function(f)⟩ ⟨domain(f) = ℙ₁⟩
have x ∈ f⁻“{f‘x} f‘x ∈ range(f)
using function_apply_Pair vimage_iff apply_rangeI by simp_all
then
show x ∈ (⋃ n ∈ range(f). f⁻“{n}) by auto
qed auto
{
assume ∀ n ∈ range(f). |f⁻“{n}| < ℙ₁
then
have ∀ n ∈ domain(?G). |?G‘n| < ℙ₁
using zero_lt_Aleph1 by (auto)
with ⟨function(?G)⟩ ⟨domain(?G) ≤ ω⟩
have |⋃ n ∈ domain(?G). ?G‘n| ≤ ω
using lt_Aleph_imp_cardinal_UN_le_nat by blast
then
have |⋃ n ∈ range(f). f⁻“{n}| ≤ ω by simp
with ℙ₁ = _
have |ℙ₁| ≤ ω by simp
then
have ℙ₁ ≤ ω
using Card_Aleph_Card_cardinal_eq
by simp
then
have False
using nat_lt_Aleph1 by (blast dest: lt_trans2)
}
with ⟨range(f) ⊆ ω⟩
obtain n where n ∈ ω ¬(|f⁻“{n}| < ℙ₁)
by blast
moreover from this
have ℙ₁ ≤ |f⁻“{n}|
using not_lt_iff_le Card_is_Ord by auto
moreover
note ⟨n ∈ ω ⟹ f⁻“{n} ⊆ ℙ₁⟩
ultimately
show ?thesis
using subset_imp_le_cardinal[THEN le_anti_sym, of _ ℙ₁]
Card_Aleph_Card_cardinal_eq by auto
qed

```

— There is some asymmetry between assumptions and conclusion (( $\approx$ ) versus *cardinal*)

**lemma** eqpoll\_Aleph1\_cardinal\_vimage:  
**assumes**  $X \approx \aleph_1$   $f : X \rightarrow \omega$

```

shows  $\exists n \in \omega. |f^{-1}\{n\}| = \aleph_1$ 
proof -
  from assms
  obtain g where  $g \in \text{bij}(\aleph_1, X)$ 
    using eqpoll_sym by blast
  with  $\langle f : X \rightarrow \omega \rangle$ 
  have  $f O g : \aleph_1 \rightarrow \omega$  converse( $g \in \text{bij}(X, \aleph_1)$ )
    using bij_is_fun comp_fun bij_converse_bij by blast+
  then
  obtain n where  $n \in \omega$   $|(f O g)^{-1}\{n\}| = \aleph_1$ 
    using Aleph1_eq_cardinal_vimage by auto
  then
  have  $\aleph_1 = |\text{converse}(g) \cap (f^{-1}\{n\})|$ 
    using image_comp converse_comp
    unfolding vimage_def by simp
  also from  $\langle \text{converse}(g) \in \text{bij}(X, \aleph_1) \rangle \langle f : X \rightarrow \omega \rangle$ 
  have ... =  $|f^{-1}\{n\}|$ 
    using range_of_subset_eqpoll[of converse(g) X _ f^-1{n}]
      bij_is_inj cardinal_cong bij_is_fun eqpoll_sym Pi_vimage_subset
      by fastforce
  finally
  show ?thesis using ⟨n ∈ ω⟩ by auto
qed

```

## 4.5 Applications of transfinite recursive constructions

The next lemma is an application of recursive constructions. It works under the assumption that whenever the already constructed subsequence is small enough, another element can be added.

```

lemma bounded_cardinal_selection:
  includes Ord_dests
  assumes
     $\bigwedge X. |X| < \gamma \implies X \subseteq G \implies \exists a \in G. \forall s \in X. Q(s, a) \ b \in G \ Card(\gamma)$ 
  shows
     $\exists S. S : \gamma \rightarrow G \wedge (\forall \alpha \in \gamma. \forall \beta \in \gamma. \alpha < \beta \longrightarrow Q(S^\alpha, S^\beta))$ 
proof -
  let ?cdlt $\gamma$ = $\{X \in \text{Pow}(G) . |X| < \gamma\}$  — “cardinal less than  $\gamma$ ”
  and ?inQ= $\lambda Y. \{a \in G. \forall s \in Y. Q(s, a)\}$ 
  from assms
  have  $\forall Y \in ?cdlt\gamma. \exists a. a \in ?inQ(Y)$ 
    by blast
  then
  have  $\exists f. f \in \text{Pi}(?cdlt\gamma, ?inQ)$ 
    using AC_ball_Pi[of ?cdlt $\gamma$  ?inQ] by simp
  then
  obtain f where f_type: $f \in \text{Pi}(?cdlt\gamma, ?inQ)$ 
    by auto
  moreover
  define Cb where  $Cb \equiv \lambda \in \text{Pow}(G) - ?cdlt\gamma. b$ 

```

```

moreover from ⟨b∈G⟩
have Cb ∈ Pow(G)-?cdlt $\gamma$  → G
  unfolding Cb_def by simp
moreover
note ⟨Card( $\gamma$ )⟩
ultimately
have f ∪ Cb : ( $\prod x \in Pow(G)$ . ?inQ(x) ∪ G) using
  fun_Pi_disjoint_Un[ of f ?cdlt $\gamma$  ?inQ Cb Pow(G)-?cdlt $\gamma$  λ_.G]
  Diff_partition[of {X ∈ Pow(G). |X| <  $\gamma$ } Pow(G), OF Collect_subset]
by auto
moreover
have ?inQ(x) ∪ G = G for x by auto
ultimately
have f ∪ Cb : Pow(G) → G by simp
define S where S ≡ $\lambda\alpha \in \gamma$ . rec_constr(f ∪ Cb,  $\alpha$ )
from ⟨f ∪ Cb: Pow(G) → G⟩ ⟨Card( $\gamma$ )⟩
have S :  $\gamma \rightarrow G$ 
  using Ord_in_Ord unfolding S_def
  by (intro lam_type rec_constr_type) auto
moreover
have  $\forall \alpha \in \gamma$ .  $\forall \beta \in \gamma$ .  $\alpha < \beta \longrightarrow Q(S` \alpha, S` \beta)$ 
proof (intro ballI impI)
  fix  $\alpha$   $\beta$ 
  assume  $\beta \in \gamma$ 
  with ⟨Card( $\gamma$ )⟩
  have {rec_constr(f ∪ Cb, x) . x ∈  $\beta\} = \{S`x . x \in \beta\}$ 
    using Ord_trans[OF __ Card_is_Ord, of _  $\beta$   $\gamma$ ]
    unfolding S_def
    by auto
  moreover from ⟨ $\beta \in \gamma$ ⟩ ⟨S :  $\gamma \rightarrow G$ , ⟨Card( $\gamma$ )⟩
  have {S`x . x ∈  $\beta\} \subseteq G$ 
    using Ord_trans[OF __ Card_is_Ord, of _  $\beta$   $\gamma$ ]
    apply_type[of S  $\gamma$  λ_. G] by auto
  moreover from ⟨Card( $\gamma$ )⟩ ⟨ $\beta \in \gamma$ ⟩
  have |{S`x . x ∈  $\beta\}| < \gamma$ 
    using cardinal_RepFun_le[of  $\beta$ ] Ord_in_Ord
    lt_trans1[of |{S`x . x ∈  $\beta\}| | $\beta\| \gamma]
    Card_lt_iff[THEN iffD2, of  $\beta$   $\gamma$ , OF __ ltI]
    by force
  moreover
  have  $\forall x \in \beta$ . Q(S`x, f ` {S`x . x ∈  $\beta\})$ 
proof -
  from calculation and f_type
  have f ` {S`x . x ∈  $\beta\} \in \{a \in G. \forall x \in \beta. Q(S`x, a)\}$ 
    using apply_type[of f ?cdlt $\gamma$  ?inQ {S`x . x ∈  $\beta\}]]
    by blast
  then
  show ?thesis by simp
qed$$$ 
```

```

moreover
assume  $\alpha \in \gamma$   $\alpha < \beta$ 
moreover
note  $\langle \beta \in \gamma \rangle \langle Cb \in Pow(G) \text{-?cdlt} \gamma \rightarrow G \rangle$ 
ultimately
show  $Q(S' \alpha, S' \beta)$ 
  using fun disjoint apply1[of {S'x . x ∈ β}] Cb f
    domain_of_fun[of Cb] ltD[of α β]
    by (subst (2) S_def, auto) (subst rec_constr_unfold, auto)
qed
ultimately
show ?thesis by blast
qed

```

The following basic result can, in turn, be proved by a bounded-cardinal selection.

```

lemma Infinite_iff_lepoll_nat:  $Infinite(X) \longleftrightarrow \omega \lesssim X$ 
proof
  assume Infinite(X)
  then
  obtain b where b ∈ X
    using Infinite_not_empty by auto
  {
    fix Y
    assume  $|Y| < \omega$ 
    then
    have Finite(Y)
      using Finite_cardinal_iff' ltD nat_into_Finite by blast
    with ⟨Infinite(X)⟩
    have  $X \neq Y$  by auto
  }
  with ⟨b ∈ X⟩
  obtain S where S : ω → X  $\forall \alpha \in \omega. \forall \beta \in \omega. \alpha < \beta \longrightarrow S' \alpha \neq S' \beta$ 
    using bounded_cardinal_selection[of ω X λx y. x ≠ y]
      Card_nat by blast
  moreover from this
  have  $\alpha \in \omega \implies \beta \in \omega \implies \alpha \neq \beta \implies S' \alpha \neq S' \beta$  for  $\alpha, \beta$ 
    by (rule_tac lt_neq_symmetry[of ω λα β. S'α ≠ S'β])
      auto
  ultimately
  show  $\omega \lesssim X$ 
    unfolding lepoll_def inj_def by blast
qed (intro lepoll_nat_imp_Infinite)

```

```

lemma Infinite_InfCard_cardinal:  $Infinite(X) \implies InfCard(|X|)$ 
using lepoll_eq_trans eqpoll_sym lepoll_nat_imp_Infinite
  Infinite_iff_lepoll_nat Inf_Card_is_InfCard cardinal_eqpoll
by simp

```

```

lemma Finite_to_one_surj_imp_cardinal_eq:
  assumes F ∈ Finite_to_one(X, Y) ∩ surj(X, Y) Infinite(X)
  shows |Y| = |X|
proof -
  from ⟨F ∈ Finite_to_one(X, Y) ∩ surj(X, Y)⟩
  have X = (⋃y∈Y. {x∈X . F‘x = y})
    using apply_type by fastforce
  show ?thesis
  proof (cases Finite(Y))
    case True
    with ⟨X = (⋃y∈Y. {x∈X . F‘x = y})⟩ and assms
    show ?thesis
      using Finite_RepFun[THEN [2] Finite_Union, of Y λy. {x∈X . F‘x = y}]
      by auto
    next
    case False
    moreover from this
    have Y ≤ |Y|
      using cardinal_eqpoll_eqpoll_sym_eqpoll_imp_lepoll by simp
    moreover
    note assms
    moreover from calculation
    have y ∈ Y ⟹ |{x∈X . F‘x = y}| ≤ |Y| for y
      using Infinite_imp_nats_lepoll[THEN lepoll_imp_cardinal_le, of Y
        |{x∈X . F‘x = y}|] cardinal_idem by auto
    ultimately
    have |⋃y∈Y. {x∈X . F‘x = y}| ≤ |Y|
      using lepoll_imp_cardinal_UN_le[of |Y| Y]
        Infinite_InfCard_cardinal[of Y] by simp
    moreover from ⟨F ∈ Finite_to_one(X, Y) ∩ surj(X, Y)⟩
    have |Y| ≤ |X|
      using surj_implies_cardinal_le by auto
    moreover
    note ⟨X = (⋃y∈Y. {x∈X . F‘x = y})⟩
    ultimately
    show ?thesis
      using le_anti_sym by auto
  qed
qed

```

```

lemma cardinal_map_Un:
  assumes Infinite(X) Finite(b)
  shows |{a ∪ b . a ∈ X}| = |X|
proof -
  have (λa∈X. a ∪ b) ∈ Finite_to_one(X, {a ∪ b . a ∈ X})
    (λa∈X. a ∪ b) ∈ surj(X, {a ∪ b . a ∈ X})
    unfolding surj_def
  proof
    fix d

```

```

have Finite({a ∈ X . a ∪ b = d}) (is Finite(?Y(b,d)))
  using ⟨Finite(b)⟩
proof (induct arbitrary:d)
  case 0
  have {a ∈ X . a ∪ 0 = d} = (if d∈X then {d} else 0)
    by auto
  then
    show ?case by simp
next
  case (cons c b)
  from ⟨c ∉ b⟩
  have ?Y(cons(c,b),d) ⊆ (if c∈d then ?Y(b,d) ∪ ?Y(b,d-{c}) else 0)
    by auto
  with cons
  show ?case
    using subset_Finite
    by simp
qed
moreover
assume d ∈ {x ∪ b . x ∈ X}
ultimately
show Finite({a ∈ X . (λx∈X. x ∪ b) ` a = d})
  by simp
qed (auto intro:lam_funtype)
with assms
show ?thesis
  using Finite_to_one_surj_imp_cardinal_eq by fast
qed

end
theory Konig
imports
  Cofinality
  Cardinal_Library
begin

```

Now, using the Axiom of choice, we can show that all successor cardinals are regular.

```

lemma cf_csucc:
  assumes InfCard(z)
  shows cf(z+) = z+
proof (rule ccontr)
  assume cf(z+) ≠ z+
  moreover from ⟨InfCard(z)⟩
  have Ord(z+) Ord(z) Limit(z) Limit(z+) Card(z+) Card(z)
    using InfCard_csucc Card_is_Ord InfCard_is_Card InfCard_is_Limit
    by fastforce+
  moreover from calculation

```

```

have  $cf(z^+) < z^+$ 
  using  $cf\_le\_cardinal[cf(z^+), \text{THEN } le\_iff[\text{THEN } iffD1]]$ 
     $Card\_cardinal\_eq$ 
  by simp
ultimately
obtain  $G$  where  $G: cf(z^+) \rightarrow z^+ \forall \beta \in z^+. \exists y \in cf(z^+). \beta < G'y$ 
  using  $Limit\_cofinal\_fun\_lt[cf(z^+)] Ord\_cf$ 
     $cf\_le\_cf\_fun[cf(z^+)] le\_refl[cf(z^+)]$ 
  by auto
with  $\langle Card(z) \rangle \langle Card(z^+) \rangle \langle Ord(z^+) \rangle$ 
have  $\forall \beta \in cf(z^+). |G'\beta| \leq z$ 
  using  $apply\_type[G cf(z^+) \lambda_. z^+, \text{THEN } ltI] Card\_lt\_iff[\text{THEN } iffD2]$ 
     $Ord\_in\_Ord[Card\_is\_Ord, of z^+] cardinal\_lt\_csucc\_iff[\text{THEN } iffD1]$ 
  by auto
from  $\langle cf(z^+) < z^+ \rangle \langle InfCard(z) \rangle \langle Ord(z) \rangle$ 
have  $cf(z^+) \lesssim z$ 
  using  $cardinal\_lt\_csucc\_iff[z cf(z^+)] Card\_csucc[z]$ 
     $le\_Card\_iff[z cf(z^+)] InfCard\_is\_Card$ 
     $Card\_lt\_iff[cf(z^+) z^+] lt\_Ord[cf(z^+) z^+]$ 
  by simp
with  $\langle cf(z^+) < z^+ \rangle \langle \forall \beta \in cf(z^+). |G'\beta| \leq z \rangle \langle InfCard(z) \rangle$ 
have  $|\bigcup \beta \in cf(z^+). G'\beta| \leq z$ 
  using  $InfCard\_csucc[z]$ 
     $subset\_imp\_lepoll[\text{THEN } lepoll\_imp\_cardinal\_le, of \bigcup \beta \in cf(z^+). G'\beta z]$ 
  by (rule_tac lepoll_imp_cardinal_UN_le) auto
moreover
note  $\langle Ord(z) \rangle$ 
moreover from  $\forall \beta \in z^+. \exists y \in cf(z^+). \beta < G'y$  and this
have  $z^+ \subseteq (\bigcup \beta \in cf(z^+). G'\beta)$ 
  by (blast dest: ltD)
ultimately
have  $z^+ \leq z$ 
  using  $subset\_imp\_le\_cardinal[z^+ \bigcup \beta \in cf(z^+). G'\beta] le\_trans$ 
     $InfCard\_is\_Card Card\_csucc[z] Card\_cardinal\_eq$ 
  by auto
with  $\langle Ord(z) \rangle$ 
show False
  using  $lt\_csucc[z] not\_lt\_iff\_le[\text{THEN } iffD2, of z z^+]$ 
     $Card\_csucc[Card\_is\_Ord]$ 
  by auto
qed

```

And this finishes the calculation of cofinality of Alephs.

```

lemma  $cf\_Aleph\_succ: Ord(z) \Rightarrow cf(\aleph_{succ(z)}) = \aleph_{succ(z)}$ 
  using Aleph_succ cf_csucc InfCard_Aleph by simp

```

## 4.6 König's Theorem

We end this section by proving König's Theorem on the cofinality of cardinal exponentiation. This is a strengthening of Cantor's theorem and it is essentially the only basic way to prove strict cardinal inequalities.

It is proved rather straightforwardly with the tools already developed.

```

lemma konigs_theorem:
  notes [dest] = InfCard_is_Card Card_is_Ord
  and [trans] = lt_trans1 lt_trans2
  assumes
    InfCard(κ) InfCard(ν) cf(κ) ≤ ν
  shows
    κ < κ↑ν
  using assms(1,2) Card_cexp
  proof (intro not_le_iff_lt[THEN iffD1] notI)
    assume κ↑ν ≤ κ
    moreover
    note ‹InfCard(κ)›
    moreover from calculation
    have ν → κ ≈ κ
    using Card_cardinal_eq[OF InfCard_is_Card, symmetric]
      Card_le_imp_lepoll
    unfolding cexp_def by simp
    ultimately
    obtain G where G ∈ surj(κ, ν → κ)
      using inj_imp_surj[OF _ function_space_nonempty,
        OF_nat_into_InfCard] by blast
    from assms
    obtain f where f:ν → κ cf_fun(f,κ)
      using cf_le_cf_fun[OF _ InfCard_is_Limit] by blast
    define H where H(α) ≡ μ x. x ∈ κ ∧ ( ∀ m < f'α. Gmα ≠ x )
      (is _ ≡ μ x. ?P(α,x)) for α
    have H_satisfies: ?P(α,H(α)) if α ∈ ν for α
    proof -
      obtain h where ?P(α,h)
      proof -
        from ‹α ∈ ν› ‹f:ν → κ› ‹InfCard(κ)›
        have f'α < κ
          using apply_type[of _ _ λ_.κ] by (auto intro:ltI)
        have |{Gmα . m ∈ {x ∈ κ . x < f'α}}| ≤ |{x ∈ κ . x < f'α}|
          using cardinal_RepFun_le by simp
        also from ‹f'α < κ› ‹InfCard(κ)›
        have |{x ∈ κ . x < f'α}| < |κ|
          using Card_lt_iff[OF lt_Ord, THEN iffD2, of f'α κ κ]
            Ord_eq_Collect_lt[of f'α κ] Card_cardinal_eq
          by force
        finally
        have |{Gmα . m ∈ {x ∈ κ . x < f'α}}| < |κ| .

```

```

moreover from ⟨f‘α < κ⟩ ⟨InfCard(κ)⟩
have m < f‘α ==> m ∈ κ for m
  using Ord_trans[of m f‘α κ]
  by (auto dest:ltD)
ultimately
have ∃ h. ?P(α,h)
  using lt_cardinal_imp_not_subset by blast
with that
show ?thesis by blast
qed
with assms
show ?P(α,H(α))
  using LeastI[of ?P(α) h] lt_Ord_Ord_in_Ord
  unfolding H_def by fastforce
qed
then
have (λα∈ν. H(α)): ν → κ
  using lam_type by auto
with ⟨G ∈ surj(κ, ν → κ)⟩
obtain n where n ∈ κ G‘n = (λα∈ν. H(α))
  unfolding surj_def by blast
moreover
note ⟨InfCard(κ)⟩ ⟨f: ν → κ⟩ ⟨cf_fun(f,_)⟩
ultimately
obtain α where n < f‘α α ∈ ν
  using Limit_cofinal_fun_lt[OF InfCard_is_Limit] by blast
moreover from calculation and ⟨G‘n = (λα∈ν. H(α))⟩
have G‘n‘α = H(α)
  using ltD by simp
moreover from calculation and H_satisfies
have ∀ m < f‘α. G‘m‘α ≠ H(α) by simp
ultimately
show False by blast
qed blast+

```

**lemma** cf\_cexp:

**assumes**

$$\text{Card}(\kappa) \text{ InfCard}(\nu) \ 2 \leq \kappa$$

**shows**

$$\nu < cf(\kappa^{\uparrow\nu})$$

**proof** (rule ccontr)

**assume**  $\neg \nu < cf(\kappa^{\uparrow\nu})$

**with** ⟨InfCard(ν)⟩

**have**  $cf(\kappa^{\uparrow\nu}) \leq \nu$

**using** not\_lt\_iff\_le Ord\_cf InfCard\_is\_Card Card\_is\_Ord by simp

**moreover**

**note** assms

**moreover from** calculation

**have**  $\text{InfCard}(\kappa^{\uparrow\nu})$  **using** InfCard\_cexp by simp

```

moreover from calculation
have  $\kappa^{\uparrow\nu} < (\kappa^{\uparrow\nu})^{\uparrow\nu}$ 
  using konigs_theorem by simp
ultimately
show False using cexp_cexp_cmult InfCard_csquare_eq by auto
qed

```

Finally, the next two corollaries illustrate the only possible exceptions to the value of the cardinality of the continuum: The limit cardinals of countable cofinality. That these are the only exceptions is a consequence of Easton's Theorem [4, Thm 15.18].

```

corollary cf_continuum:  $\aleph_0 < cf(2^{\aleph_0})$ 
  using cf_cexp InfCard_Aleph_nat_into_Card by simp

corollary continuum_not_eq_Aleph_nat:  $2^{\aleph_0} \neq \aleph_\omega$ 
  using cf_continuum cf_Aleph_Limit[OF Limit_nat] cf_nat
    Aleph_zero_eq_nat by auto

```

end

## 5 The Delta System Lemma

```

theory Delta_System
imports
  Cardinal_Library

```

begin

The *Delta System Lemma* (DSL) states that any uncountable family of finite sets includes an uncountable delta system. This is the simplest non trivial version; others, for cardinals greater than  $\aleph_1$  assume some weak versions of the generalized continuum hypothesis for the cardinals involved.

The proof is essentially the one in [6, III.2.6] for the case  $\aleph_1$ ; another similar presentation can be found in [5, Chap. 16].

```

lemma delta_system_Aleph1:
  assumes  $\forall A \in F. \text{Finite}(A) \wedge F \approx \aleph_1$ 
  shows  $\exists D. D \subseteq F \wedge \text{delta_system}(D) \wedge D \approx \aleph_1$ 
proof -

```

Since all members are finite,

```

from  $\forall A \in F. \text{Finite}(A)$ 
have  $(\lambda A \in F. |A|) : F \rightarrow \omega$  (is ?cards : _)
  by (rule_tac lam_type) simp
moreover from this
have  $a : ?cards - ``\{n\} = \{ A \in F . |A| = n \}$  for n
  using vimage_lam by auto
moreover

```

**note**  $\langle F \approx \aleph_1 \rangle$   
**moreover from** *calculation*

there are uncountably many have the same cardinal:

```
obtain n where n∈ω |?cards -“ {n}| = aleph1
  using eqpoll_Aleph1_cardinal_vimage[of F ?cards] by auto
moreover
define G where G ≡ ?cards -“ {n}
moreover from calculation
have G ⊆ F by auto
ultimately
```

Therefore, without loss of generality, we can assume that all elements of the family have cardinality  $n \in \omega$ .

```
have A∈G ==> |A| = n and G ≈ aleph1 for A
  using cardinal_Card_eqpoll_iff by auto
with ⟨n∈ω⟩
```

So we prove the result by induction on this  $n$  and generalizing  $G$ , since the argument requires changing the family in order to apply the inductive hypothesis.

```
have ∃ D. D ⊆ G ∧ delta_system(D) ∧ D ≈ aleph1
proof (induct arbitrary:G)
  case 0 — This case is impossible
  then
    have G ⊆ {0}
      using cardinal_0_iff_0 by auto
    with ⟨G ≈ aleph1⟩
    show ?case
      using nat_lt_Aleph1_subset_imp_le_cardinal[of G {0}]
        lt_trans2 cardinal_Card_eqpoll_iff by auto
  next
    case (succ n)
    then
      have ∀ a∈G. Finite(a)
        using Finite_cardinal_iff' nat_into_Finite[of succ(n)]
          by fastforce
      show ∃ D. D ⊆ G ∧ delta_system(D) ∧ D ≈ aleph1
    proof (cases ∃ p. {A∈G . p ∈ A} ≈ aleph1)
      case True — the positive case, uncountably many sets with a common element
      then
        obtain p where {A∈G . p ∈ A} ≈ aleph1 by blast
        moreover from this
        have {A-{p} . A∈{X∈G. p∈X}} ≈ aleph1 (is ?F ≈ _)
          using Diff_bij[of {A∈G . p ∈ A} {p}]
            comp_bij[OF bij_converse_bij, where C=aleph1] by fast
```

Now using the hypothesis of the successor case,

```

moreover from  $\langle \bigwedge A. A \in G \implies |A| = \text{succ}(n) \rangle$   $\langle \forall a \in G. \text{Finite}(a) \rangle$ 
and this
have  $p \in A \implies A \in G \implies |A - \{p\}| = n$  for  $A$ 
using Finite_imp_succ_cardinal_Diff[of _ p] by force
moreover from this and  $\langle n \in \omega \rangle$ 
have  $\forall a \in ?F. \text{Finite}(a)$ 
using Finite_cardinal_iff' nat_into_Finite by auto
moreover

```

we may apply the inductive hypothesis to the new family  $\{A - \{p\} . A \in \{X \in G . p \in X\}\}$ :

```

note  $\langle (\bigwedge A. A \in ?F \implies |A| = n) \implies ?F \approx \aleph_1 \implies$ 
 $\exists D. D \subseteq ?F \wedge \text{delta_system}(D) \wedge D \approx \aleph_1 \rangle$ 
ultimately
obtain  $D$  where  $D \subseteq \{A - \{p\} . A \in \{X \in G . p \in X\}\}$   $\text{delta_system}(D)$   $D \approx \aleph_1$ 
by auto
moreover from this
obtain  $r$  where  $\forall A \in D. \forall B \in D. A \neq B \longrightarrow A \cap B = r$ 
by fastforce
then
have  $\forall A \in D. \forall B \in D. A \cup \{p\} \neq B \cup \{p\} \longrightarrow (A \cup \{p\}) \cap (B \cup \{p\}) = r \cup \{p\}$ 
by blast
ultimately
have  $\text{delta_system}(\{B \cup \{p\} . B \in D\})$  (is  $\text{delta_system}(?D)$ )
by fastforce
moreover from  $\langle D \approx \aleph_1 \rangle$ 
have  $|D| = \aleph_1$  Infinite( $D$ )
using cardinal_eqpoll_iff
by (auto intro!: uncountable_iff_subset_eqpoll_Aleph1 [THEN iffD2]
uncountable_imp_Infinite) force
moreover from this
have  $?D \approx \aleph_1$ 
using cardinal_map_Un[of  $D \setminus \{p\}$ ] naturals_lt_nat
cardinal_eqpoll_iff[THEN iffD1] by simp
moreover
note  $\langle D \subseteq \{A - \{p\} . A \in \{X \in G . p \in X\}\} \rangle$ 
have  $?D \subseteq G$ 
proof -
{
fix  $A$ 
assume  $A \in G$   $p \in A$ 
moreover from this
have  $A = A - \{p\} \cup \{p\}$ 
by blast
ultimately
have  $A - \{p\} \cup \{p\} \in G$ 
by auto
}
with  $\langle D \subseteq \{A - \{p\} . A \in \{X \in G . p \in X\}\} \rangle$ 

```

```

show ?thesis
  by blast
qed
ultimately
show  $\exists D. D \subseteq G \wedge \text{delta\_system}(D) \wedge D \approx \aleph_1$  by auto
next
  case False
  note  $\neg (\exists p. \{A \in G . p \in A\} \approx \aleph_1)$  — the other case
  moreover from  $\langle G \approx \aleph_1 \rangle$ 
  have  $\{A \in G . p \in A\} \lesssim \aleph_1$  (is  $?G(p) \lesssim \_\_$ ) for p
    by (blast intro:lepoll_eq_trans[OF subset_imp_lepoll])
  ultimately
  have  $?G(p) \prec \aleph_1$  for p
    unfolding lesspoll_def by simp
  then
  have  $?G(p) \lesssim \omega$  for p
    using lesspoll_aleph_plus_one[of 0] Aleph_zero_eq_nat by auto
  moreover
  have  $\{A \in G . S \cap A \neq \emptyset\} = (\bigcup_{p \in S} ?G(p))$  for S
    by auto
  ultimately
  have countable(S)  $\implies$  countable( $\{A \in G . S \cap A \neq \emptyset\}$ ) for S
    using InfCard_nat.Card_nat
    le_Card_iff[THEN iffD2, THEN [3] lepoll_imp_cardinal_UN_le,
      THEN [2] le_Card_iff[THEN iffD1], of  $\omega$  S]
    unfolding countable_def by simp

```

For every countable subfamily of  $G$  there is another some element disjoint from all of them:

```

have  $\exists A \in G. \forall S \in X. S \cap A = \emptyset$  if  $|X| < \aleph_1$   $X \subseteq G$  for X
proof -
  from  $\langle n \in \omega \rangle \langle \bigwedge A. A \in G \implies |A| = \text{succ}(n) \rangle$ 
  have  $A \in G \implies \text{Finite}(A)$  for A
    using cardinal_Card_eqpoll_iff
    unfolding Finite_def by fastforce
  with  $\langle X \subseteq G \rangle$ 
  have  $A \in X \implies \text{countable}(A)$  for A
    using Finite_imp_countable by auto
  with  $\langle |X| < \aleph_1 \rangle$ 
  have countable( $\bigcup X$ )
    using Card_nat[THEN cardinal_lt_csucc_iff, of X]
    countable_union_countable_countable_iff_cardinal_le_nat
    unfolding Aleph_def by simp
  with  $\langle \text{countable}(\_) \implies \text{countable}(\{A \in G . \_\_ \cap A \neq \emptyset\}) \rangle$ 
  have countable( $\{A \in G . (\bigcup X) \cap A \neq \emptyset\}$ ).
  with  $\langle G \approx \aleph_1 \rangle$ 
  obtain B where  $B \in G$   $B \notin \{A \in G . (\bigcup X) \cap A \neq \emptyset\}$ 
    using nat_lt_Aleph1_cardinal_Card_eqpoll_iff[of  $\aleph_1$  G]
    uncountable_not_subset_countable[of  $\{A \in G . (\bigcup X) \cap A \neq \emptyset\}$  G]

```

```

uncountable_iff_nat_lt_cardinal
by auto
then
show  $\exists A \in G. \forall S \in X. S \cap A = \emptyset$  by auto
qed
moreover from  $\langle G \approx \aleph_1 \rangle$ 
obtain b where  $b \in G$ 
using uncountable_iff_subset_eqpoll_Aleph1
uncountable_not_empty by blast
ultimately

```

Hence, the hypotheses to perform a bounded-cardinal selection are satisfied,

```

obtain S where  $S : \aleph_1 \rightarrow G \alpha \in \aleph_1 \implies \beta \in \aleph_1 \implies \alpha < \beta \implies S^\alpha \cap S^\beta = \emptyset$ 
for  $\alpha \beta$ 
using bounded_cardinal_selection[of  $\aleph_1 G \lambda s a. s \cap a = \emptyset b$ ]
by force
then
have  $\alpha \in \aleph_1 \implies \beta \in \aleph_1 \implies \alpha \neq \beta \implies S^\alpha \cap S^\beta = \emptyset$  for  $\alpha \beta$ 
using lt_neq_symmetry[of  $\aleph_1 \lambda \alpha \beta. S^\alpha \cap S^\beta = \emptyset$ ] Card_is_Ord
by auto blast

```

and a symmetry argument shows that obtained  $S$  is an injective  $\aleph_1$ -sequence of disjoint elements of  $G$ .

```

moreover from this and  $\langle \bigwedge A. A \in G \implies |A| = \text{succ}(n) \rangle \langle S : \aleph_1 \rightarrow G \rangle$ 
have  $S \in \text{inj}(\aleph_1, G)$ 
using cardinal_succ_not_0 Int_eq_zero_imp_not_eq[of  $\aleph_1 \lambda x. S^x$ ]
unfolding inj_def by fastforce
moreover from calculation
have  $\text{range}(S) \approx \aleph_1$ 
using inj_bij_range_eqpoll_sym unfolding eqpoll_def by fast
moreover from calculation
have  $\text{range}(S) \subseteq G$ 
using inj_is_fun range_fun_subset_codomain by fast
ultimately
show  $\exists D. D \subseteq G \wedge \text{delta_system}(D) \wedge D \approx \aleph_1$ 
using inj_is_fun range_eq_image[of  $S \aleph_1 G$ ]
image_function[OF fun_is_function, OF inj_is_fun, of  $S \aleph_1 G$ ]
domain_of_fun[OF inj_is_fun, of  $S \aleph_1 G$ ]
by (rule_tac x= $S^\alpha \aleph_1$  in exI) auto

```

This finishes the successor case and hence the proof.

```

qed
qed
with  $\langle G \subseteq F \rangle$ 
show ?thesis by blast
qed

```

```

lemma delta_system_uncountable:
assumes  $\forall A \in F. \text{Finite}(A) \text{ uncountable}(F)$ 

```

```

shows  $\exists D. D \subseteq F \wedge \text{delta\_system}(D) \wedge D \approx \aleph_1$ 
proof -
  from assms
  obtain S where  $S \subseteq F \wedge S \approx \aleph_1$ 
    using uncountable_iff_subset_eqpoll_Aleph1[of F] by auto
  moreover from  $\langle \forall A \in F. \text{Finite}(A) \rangle$  and this
  have  $\forall A \in S. \text{Finite}(A)$  by auto
  ultimately
  show ?thesis using delta_system_Aleph1[of S]
    by auto
qed

end

```

## 5.1 Application to Cohen posets

```

theory Cohen_Posets
  imports
    Delta_System

```

```
begin
```

We end this session by applying DSL to the combinatorics of finite function posets. We first define some basic concepts; we take a different approach from [1], in that the order relation is presented as a predicate (of type  $i \Rightarrow i \Rightarrow o$ ).

Two elements of a poset are *compatible* if they have a common lower bound.

```

definition compat_in ::  $[i, [i, i] \Rightarrow o, i, i] \Rightarrow o$  where
  compat_in( $A, r, p, q$ )  $\equiv \exists d \in A . r(d, p) \wedge r(d, q)$ 

```

An *antichain* is a subset of pairwise incompatible members.

```

definition
  antichain ::  $[i, [i, i] \Rightarrow o, i] \Rightarrow o$  where
  antichain( $P, leq, A$ )  $\equiv A \subseteq P \wedge (\forall p \in A. \forall q \in A. p \neq q \longrightarrow \neg \text{compat\_in}(P, leq, p, q))$ 

```

A poset has the *countable chain condition* (ccc) if all of its antichains are countable.

```

definition
  ccc ::  $[i, [i, i] \Rightarrow o] \Rightarrow o$  where
  ccc( $P, leq$ )  $\equiv \forall A. \text{antichain}(P, leq, A) \longrightarrow \text{countable}(A)$ 

```

Finally, the *Cohen poset* is the set of finite partial functions between two sets with the order of reverse inclusion.

```

definition
  Fn ::  $[i, i] \Rightarrow i$  where
  Fn( $I, J$ )  $\equiv \bigcup \{(d \rightarrow J) . d \in \{x \in \text{Pow}(I). \text{Finite}(x)\}\}$ 

```

```

abbreviation
  Supset ::  $i \Rightarrow i \Rightarrow o$  (infixl  $\sqsupseteq$  50) where
     $f \sqsupseteq g \equiv g \subseteq f$ 

lemma FnI[intro]:
  assumes  $p : d \rightarrow J$   $d \subseteq I$  Finite(d)
  shows  $p \in Fn(I, J)$ 
  using assms unfolding Fn_def by auto

lemma FnD[dest]:
  assumes  $p \in Fn(I, J)$ 
  shows  $\exists d. p : d \rightarrow J \wedge d \subseteq I \wedge Finite(d)$ 
  using assms unfolding Fn_def by auto

lemma Fn_is_function:  $p \in Fn(I, J) \implies function(p)$ 
  unfolding Fn_def using fun_is_function by auto

lemma restrict_eq_imp_compat:
  assumes  $f \in Fn(I, J)$   $g \in Fn(I, J)$ 
   $restrict(f, domain(f) \cap domain(g)) = restrict(g, domain(f) \cap domain(g))$ 
  shows  $f \cup g \in Fn(I, J)$ 
proof -
  from assms
  obtain d1 d2 where  $f : d1 \rightarrow J$   $d1 \in Pow(I)$  Finite(d1)
     $g : d2 \rightarrow J$   $d2 \in Pow(I)$  Finite(d2)
    by blast
  with assms
  show ?thesis
    using domain_of_fun
    restrict_eq_imp_Un_into_Pi[of f d1 λ_. J g d2 λ_. J]
    by auto
qed

```

We finally arrive to our application of DSL.

```

lemma ccc_Fn_2: ccc(Fn(I, 2), (⊇))
proof -
  {
    fix A
    assume ¬ countable(A)
    assume A ⊆ Fn(I, 2)
    moreover from this
    have countable({p ∈ A. domain(p) = d}) for d
    proof (cases Finite(d) ∧ d ⊆ I)
      case True
      with ‹A ⊆ Fn(I, 2)›
      have {p ∈ A . domain(p) = d} ⊆ d → 2
        using domain_of_fun by fastforce
      moreover from True
      have Finite(d → 2)
    qed
  }

```

```

using Finite_Pi lesspoll_nat_is_Finite by auto
ultimately
show ?thesis using subset_Finite[of _ d→2 ] Finite_imp_countable
by auto
next
case False
with ⟨A ⊆ Fn(I, 2)⟩
have {p ∈ A . domain(p) = d} = 0
by (intro equalityI) (auto dest!: domain_of_fun)
then
show ?thesis using empty_lepollI by auto
qed
moreover
have uncountable({domain(p) . p ∈ A})
proof
from ⟨A ⊆ Fn(I, 2)⟩
have A = (⋃ d∈{domain(p) . p ∈ A}. {p ∈ A. domain(p) = d})
by auto
moreover
assume countable({domain(p) . p ∈ A})
moreover
note ⟨∀ d. countable({p ∈ A. domain(p) = d})⟩ ⊢ ¬countable(A)
ultimately
show False
using countable_imp_countable_UN[of {domain(p). p ∈ A}]
λd. {p ∈ A. domain(p) = d }
by auto
qed
moreover from ⟨A ⊆ Fn(I, 2)⟩
have p ∈ A ⟹ Finite(domain(p)) for p
using lesspoll_nat_is_Finite[of domain(p)]
domain_of_fun[of p _ λ_. 2] by auto
ultimately
obtain D where delta_system(D) D ⊆ {domain(p) . p ∈ A} D ≈ ℙ₁
using delta_system_uncountable[of {domain(p) . p ∈ A}] by auto
then
have delta: ∀ d1 ∈ D. ∀ d2 ∈ D. d1 ≠ d2 → d1 ∩ d2 = ∩ D
using delta_system_root_eq_Inter
by simp
moreover from ⟨D ≈ ℙ₁⟩
have uncountable(D)
using uncountable_iff_subset_eqpoll_Aleph1 by auto
moreover from this and ⟨D ⊆ {domain(p) . p ∈ A}⟩
obtain p1 where p1 ∈ A domain(p1) ∈ D
using uncountable_not_empty[of D] by blast
moreover from this and ⟨p1 ∈ A ⟹ Finite(domain(p1))⟩
have Finite(domain(p1)) using Finite_domain by simp
moreover
define r where r ≡ ∩ D

```

```

ultimately
have Finite(r) using subset_Finite[of r domain(p1)] by auto
have countable({restrict(p,r) . p ∈ A})
proof -
  have f ∈ Fn(I, 2) ==> restrict(f,r) ∈ Pow(r × 2) for f
    using restrict_subset_Sigma[of f _ λ_. 2 r]
    by (force simp: Pi_def)
  with ⟨A ⊆ Fn(I, 2)⟩
  have {restrict(f,r) . f ∈ A } ⊆ Pow(r × 2)
    by fast
  with ⟨Finite(r)⟩
  show ?thesis
    using Finite_Sigma[THEN Finite_Pow, of r λ_. 2]
    by (intro Finite_imp_countable) (auto intro:subset_Finite)
qed
moreover
have uncountable({p ∈ A. domain(p) ∈ D}) (is uncountable(?X))
proof
  from ⟨D ⊆ {domain(p) . p ∈ A}⟩
  have (λp ∈ ?X. domain(p)) ∈ surj(?X, D)
    using lam_type unfolding surj_def by auto
  moreover
  assume countable(?X)
  moreover
  note ⟨uncountable(D)⟩
  ultimately
  show False
    using surj_countable by auto
qed
moreover
have D = (⋃f ∈ Pow(r × 2) . {domain(p) . p ∈ {x ∈ A. restrict(x,r) = f ∧ domain(x) ∈ D}})
proof -
  {
    fix z
    assume z ∈ D
    with ⟨D ⊆ _⟩
    obtain p where domain(p) = z p ∈ A
      by auto
    moreover from ⟨A ⊆ Fn(I, 2)⟩ and this
    have p : z → 2
      using domain_of_fun by force
    moreover from this
    have restrict(p,r) ⊆ r × 2
      using function_restrictI[of p r] fun_is_function[of p z λ_. 2]
      restrict_subset_Sigma[of p z λ_. 2 r]
      by (auto simp:Pi_def)
    ultimately
    have ∃p ∈ A. restrict(p,r) ∈ Pow(r × 2) ∧ domain(p) = z by auto
  }

```

```

}
then
show ?thesis
  by (intro equalityI) (force) +
qed
obtain f where uncountable({domain(p) . p ∈ {x ∈ A. restrict(x,r) = f ∧ do-
main(x) ∈ D}})
  (is uncountable(?Y(f)))
proof -
{
  from ⟨Finite(r)⟩
  have countable(Pow(r × 2))
    using Finite_Sigma[THEN Finite_Pow, THEN Finite_imp_countable]
    by simp
  moreover
  assume countable(?Y(f)) for f
  moreover
  note ⟨D = (⋃f ∈ Pow(r × 2) . ?Y(f))⟩
  moreover
  note ⟨uncountable(D)⟩
  ultimately
  have False
    using countable_imp_countable_UN[of Pow(r × 2) ?Y] by auto
}
with that
show ?thesis by auto
qed
then
obtain j where j ∈ inj(nat, ?Y(f))
  using uncountable_iff_nat_lt_cardinal[THEN iffD1, THEN leI,
  THEN cardinal_le_imp_lepoll, THEN lepollD]
  by auto
then
have j'0 ≠ j'1 j'0 ∈ ?Y(f) j'1 ∈ ?Y(f)
  using inj_is_fun[THEN apply_type, of j nat ?Y(f)]
  unfolding inj_def by auto
then
obtain p q where domain(p) ≠ domain(q) p ∈ A q ∈ A
  domain(p) ∈ D domain(q) ∈ D
  restrict(p,r) = restrict(q,r) by auto
moreover from this and delta
have domain(p) ∩ domain(q) = r unfolding r_def by simp
moreover
note ⟨A ⊆ Fn(I, 2)⟩
moreover from calculation
have p ∪ q ∈ Fn(I, 2)
  by (rule_tac restrict_eq_imp_compat) auto
ultimately
have ∃p ∈ A. ∃q ∈ A. p ≠ q ∧ compat_in(Fn(I, 2), (⊇), p, q)

```

```

unfolding compat_in_def
  by (rule_tac bexI[of _ p], rule_tac bexI[of _ q]) blast
}
then
  show ?thesis unfolding ccc_def antichain_def by auto
qed

```

The fact that a poset  $P$  has the ccc has useful consequences for the theory of forcing, since it implies that cardinals from the original model are exactly the cardinals in any generic extension by  $P$  [6, Chap. IV].

**end**

## References

- [1] E. GUNTHER, M. PAGANO, P. SÁNCHEZ TERRAF, Formalization of Forcing in Isabelle/ZF, in: N. Peltier, V. Sofronie-Stokkermans (Eds.), Automated Reasoning. 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part II, Lecture Notes in Artificial Intelligence **12167**, Springer International Publishing: 221–235 (2020). doi:[10.1007/978-3-030-51054-1](https://doi.org/10.1007/978-3-030-51054-1).
- [2] E. GUNTHER, M. PAGANO, P. SÁNCHEZ TERRAF, M. STEINBERG, The independence of the continuum hypothesis in Isabelle/ZF, *Archive of Formal Proofs* (2022). [https://isa-afp.org/entries/Independence\\_CH.html](https://isa-afp.org/entries/Independence_CH.html), Formal proof development.
- [3] E. GUNTHER, M. PAGANO, P. SÁNCHEZ TERRAF, M. STEINBERG, Transitive models of fragments of ZFC, *Archive of Formal Proofs* (2022). [https://isa-afp.org/entries/Transitive\\_Models.html](https://isa-afp.org/entries/Transitive_Models.html), Formal proof development.
- [4] T. JECH, “Set Theory. The Millennium Edition”, Springer Monographs in Mathematics, Springer-Verlag (2002), third edition. Corrected fourth printing, 2006.
- [5] W. JUST, M. WEESE, “Discovering Modern Set Theory. II”, Grad. Studies in Mathematics **18**, American Mathematical Society (1997).
- [6] K. KUNEN, “Set Theory”, Studies in Logic, College Publications (2011), second edition. Revised edition, 2013.
- [7] L.C. PAULSON, K. GRABCZEWSKI, Mechanizing set theory, *J. Autom. Reasoning* **17**: 291–323 (1996). doi:[10.1007/BF00283132](https://doi.org/10.1007/BF00283132).
- [8] P. SÁNCHEZ TERRAF, Course notes on set theory, online, (2019). In Spanish, [https://cs.famaf.unc.edu.ar/~pedro/home\\_en.html#set\\_theory](https://cs.famaf.unc.edu.ar/~pedro/home_en.html#set_theory).