

Cofinality and the Delta System Lemma

Pedro Sánchez Terraf^{*†}

February 6, 2026

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

1	Introduction	2
2	Library of basic ZF results	2
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
3	Cofinality	24
3.1	Basic results and definitions	24
3.2	The factorization lemma	32
3.3	Classical results on cofinalities	41
4	Cardinal Arithmetic under Choice	49
4.1	Results on cardinal exponentiation	49
4.2	Miscellaneous	51
4.3	Countable and uncountable sets	54

^{*}Universidad Nacional de Córdoba. Facultad de Matemática, Astronomía, Física y Computación.

[†]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
5	The Delta System Lemma	66
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* ($\langle \omega \rangle$)
notation *csucc* ($\langle _+ \rangle$ [90])
no_notation *Aleph* ($\langle \aleph _ \rangle$ [90] 90)
notation *Aleph* ($\langle \aleph _ \rangle$)
syntax *_ge* :: $[i, i] \Rightarrow o$ (**infixl** $\langle \geq \rangle$ 50)
translations $x \geq y \rightarrow y \leq x$

2.1 Some minimal arithmetic/ordinal stuff

lemma *Un_leD1* : $i \cup j \leq k \Longrightarrow \text{Ord}(i) \Longrightarrow \text{Ord}(j) \Longrightarrow \text{Ord}(k) \Longrightarrow i \leq k$
by (rule *Un_least_lt_iff*[*THEN iffD1*[*THEN conjunct1*]],*simp_all*)

lemma *Un_leD2* : $i \cup j \leq k \Longrightarrow \text{Ord}(i) \Longrightarrow \text{Ord}(j) \Longrightarrow \text{Ord}(k) \Longrightarrow j \leq k$
by (rule *Un_least_lt_iff*[*THEN iffD1*[*THEN conjunct2*]],*simp_all*)

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

lemma *Un_memD2* : $i \cup j \in k \Longrightarrow \text{Ord}(i) \Longrightarrow \text{Ord}(j) \Longrightarrow \text{Ord}(k) \Longrightarrow j \leq k$
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 $\text{Ord}(\gamma)$ $\beta \in \gamma$ $\forall \alpha \in \gamma. \alpha \leq \beta$
shows $\gamma = \text{succ}(\beta)$
using *assms Ord_trans*[*of* $_ \beta \gamma$]
unfolding *lt_def*
by (*intro equalityI subsetI*) *auto*

lemma *Least_antitone*:
assumes
 $\text{Ord}(j)$ $P(j) \wedge i. P(i) \Longrightarrow Q(i)$
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*:
 $\text{Ord}(j) \Longrightarrow j \in A \Longrightarrow A \subseteq B \Longrightarrow (\mu i. i \in B) \leq (\mu i. i \in A)$
using *subset_iff* **by** (*auto intro:Least_antitone*)

lemma *le_neq_imp_lt*:
 $x < y \Longrightarrow x \neq y \Longrightarrow x < y$

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

Strict upper bound of a set of ordinals.

definition

str_bound :: $i \Rightarrow i$ **where**
str_bound(*A*) $\equiv \bigcup_{a \in A}. \text{succ}(a)$

lemma *str_bound_type* [*TC*]: $\forall a \in A. \text{Ord}(a) \Longrightarrow \text{Ord}(\text{str_bound}(A))$

unfolding *str_bound_def* **by** *auto*

lemma *str_bound_lt*: $\forall a \in A. \text{Ord}(a) \Longrightarrow \forall a \in A. a < \text{str_bound}(A)$

unfolding *str_bound_def* **using** *str_bound_type*
by (*blast intro:ltI*)

lemma *naturals_lt_nat*[*intro*]: $n \in \omega \Longrightarrow n < \omega$

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

$\bigwedge x y. x \in D \Longrightarrow y \in D \Longrightarrow x \neq y \Longrightarrow A(x) \cap A(y) = 0$
 $\bigwedge x. x \in D \Longrightarrow A(x) \neq 0$ $a \in D$ $b \in D$ $a \neq b$

shows

$A(a) \neq A(b)$

using *assms* **by** *fastforce*

lemma *lt_neq_symmetry*:

assumes

$\bigwedge \alpha \beta. \alpha \in \gamma \Longrightarrow \beta \in \gamma \Longrightarrow \alpha < \beta \Longrightarrow Q(\alpha, \beta)$
 $\bigwedge \alpha \beta. Q(\alpha, \beta) \Longrightarrow Q(\beta, \alpha)$
 $\alpha \in \gamma$ $\beta \in \gamma$ $\alpha \neq \beta$
 $\text{Ord}(\gamma)$

shows

$Q(\alpha, \beta)$

proof -

from *assms*

consider $\alpha < \beta$ | $\beta < \alpha$

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*: $|A| = \text{succ}(n) \Longrightarrow A \neq 0$

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 $f: X \rightarrow Y \wedge y. y \in Y \implies \text{Finite}(\{x \in X . f'x = y\})$
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:  $f \in Pi(A,B) \implies range(f) = \{f'x \mid x \in A\}$ 
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
   $Ord(\gamma) \ \gamma \neq 0 \ f: A \rightarrow \gamma$ 
shows
   $f'x \in \gamma$ 
proof (cases  $x \in A$ )
case True
from assms  $\langle x \in A \rangle$ 
show ?thesis
  using domain_of_fun apply_rangeI by simp
next
case False
from assms  $\langle x \notin A \rangle$ 
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 lambda. 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 _ _ lambda. B] by force

lemma inj_to_Image:

```

```

assumes
   $f:A \rightarrow B$   $f \in \text{inj}(A,B)$ 
shows
   $f \in \text{inj}(A, f^{-1}A)$ 
using assms inj_inj_range range_eq_image by force

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

lemma fun_Pi_disjoint_Un:
assumes  $f \in \text{Pi}(A,B)$   $g \in \text{Pi}(C,D)$   $A \cap C = 0$ 
shows  $f \cup g \in \text{Pi}(A \cup C, \lambda x. B(x) \cup D(x))$ 
using assms
by (simp add: Pi_iff_extension Un_rls) (unfold function_def, blast)

lemma Un_restrict_decomposition:
assumes  $f \in \text{Pi}(A,B)$ 
shows  $f = \text{restrict}(f, A \cap C) \cup \text{restrict}(f, A - C)$ 
using assms
proof (rule fun_extension)
from assms
have  $\text{restrict}(f, A \cap C) \cup \text{restrict}(f, A - C) \in \text{Pi}(A \cap C \cup (A - C), \lambda x. B(x) \cup B(x))$ 
  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  $\text{restrict}(f, A \cap C) \cup \text{restrict}(f, A - C) \in \text{Pi}(A, B)$  by simp
next

```

```

fix x
assume  $x \in A$ 
with assms
show  $f \upharpoonright x = (\text{restrict}(f, A \cap C) \cup \text{restrict}(f, A - C)) \upharpoonright x$ 
  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  $x \in C$ ) simp_all
qed

```

```

lemma restrict_eq_imp_Un_into_Pi:
  assumes  $f \in \text{Pi}(A,B)$   $g \in \text{Pi}(C,D)$   $\text{restrict}(f, A \cap C) = \text{restrict}(g, A \cap C)$ 
  shows  $f \cup g \in \text{Pi}(A \cup C, \lambda x. B(x) \cup D(x))$ 
proof -
  note assms
  moreover from this
  have  $x \notin g \implies x \notin \text{restrict}(g, A \cap C)$  for  $x$ 
    using restrict_subset[of  $g$   $A \cap C$ ] by auto
  moreover from calculation
  have  $x \in f \implies x \in \text{restrict}(f, A - C) \vee x \in \text{restrict}(g, A \cap C)$  for  $x$ 
    by (subst (asm) Un_restrict_decomposition[of  $f$   $A$   $B$   $C$ ]) auto
  ultimately
  have  $f \cup g = \text{restrict}(f, A - C) \cup g$ 
    using restrict_subset[of  $g$   $A \cap C$ ]
    by (subst Un_restrict_decomposition[of  $f$   $A$   $B$   $C$ ]) auto
  moreover
  have  $A - C \cup C = A \cup C$  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  $f \in \text{Pi}(A,B)$   $g \in \text{Pi}(C,D)$ 
     $\text{restrict}(f, \text{domain}(f) \cap \text{domain}(g)) = \text{restrict}(g, \text{domain}(f) \cap \text{domain}(g))$ 
  shows  $f \cup g \in \text{Pi}(A \cup C, \lambda x. B(x) \cup D(x))$ 
  using assms domain_of_fun restrict_eq_imp_Un_into_Pi by simp

```

```

lemma restrict_subset_Sigma:  $f \subseteq \text{Sigma}(C,B) \implies \text{restrict}(f,A) \subseteq \text{Sigma}(A \cap C, B)$ 
  by (auto simp add: restrict_def)

```

2.3 Finite sets

```

lemma Replace_sing1:

```

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

— Not really necessary

lemma *Replace_sing2*:
assumes $\forall a. \neg P(d,a)$
shows $\{y . x \in \{d\}, P(x,y)\} = 0$
using *assms* **by** *auto*

lemma *Replace_sing3*:
assumes $\exists c e. c \neq e \wedge P(d,c) \wedge P(d,e)$
shows $\{y . x \in \{d\}, P(x,y)\} = 0$
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) $(\exists a. P(d,a)) \wedge (\forall y y'. P(d,y) \longrightarrow P(d,y') \longrightarrow y=y') \mid$
 (2) $\forall a. \neg P(d,a) \mid$ (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*: $Finite(A) \implies Finite(Replace(A, Q))$

proof (*induct rule:Finite_induct*)

case *0*

then

show *?case* **by** *simp*

next

case (*cons x B*)

moreover

have $\{b . a \in cons(x, B), Q(a, b)\} =$

$\{b . a \in B, Q(a, b)\} \cup \{b . a \in \{x\}, Q(a, b)\}$

using *Replace_Un unfolding cons_def* **by** *auto*

moreover

obtain *d* **where** $\{b . a \in \{x\}, Q(a, b)\} \subseteq \{d\}$

using *Replace_subset_sing*[of *_ Q*] **by** *blast*

moreover from *this*

have $Finite(\{b . a \in \{x\}, Q(a, b)\})$

using *subset_Finite* **by** *simp*

ultimately

show *?case* **using** *subset_Finite* **by** *simp*

qed

lemma *Finite_domain*: $Finite(A) \implies Finite(domain(A))$

using *Finite_Replace unfolding domain_def*

by *auto*

lemma *Finite_converse*: $Finite(A) \implies Finite(converse(A))$

using *Finite_Replace unfolding converse_def*

by *auto*

lemma *Finite_range*: $Finite(A) \implies Finite(range(A))$

using *Finite_domain Finite_converse unfolding range_def*

by *blast*

lemma *Finite_Sigma*: $Finite(A) \implies \forall x. Finite(B(x)) \implies Finite(Sigma(A, B))$

unfolding *Sigma_def* **using** *Finite_RepFun Finite_Union*

by *simp*

lemma *Finite_Pi*: $Finite(A) \implies \forall x. Finite(B(x)) \implies Finite(Pi(A, B))$

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 *lepoll_def* .

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

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

declare *bij_imp_eqpoll[intro]*

lemma *range_of_subset_eqpoll*:
assumes $f \in \text{inj}(X, Y) \ S \subseteq X$
shows $S \approx f \text{ `` } S$
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_bijective, 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 ( $\lambda x \in A \rightarrow B. g \circ x \circ f$ )  $\circ$  ( $\lambda x \in A' \rightarrow B'. \text{converse}(g) \circ x \circ \text{converse}(f)$ )
  = ( $\lambda x \in A' \rightarrow B'. (g \circ \text{converse}(g)) \circ x \circ (\text{converse}(f) \circ f)$ )
  using lam_cong comp_assoc comp_lam[of A' → B'] by auto
also
have ... = ( $\lambda x \in A' \rightarrow B'. \text{id}(B') \circ x \circ (\text{id}(A'))$ )
  using left_comp_inverse[OF bij_is_inj][OF ⟨f ∈ _⟩]
  right_comp_inverse[OF bij_is_surj][OF ⟨g ∈ _⟩]
  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' → B') unfolding id_def by simp
finally
show ( $\lambda x \in A \rightarrow B. g \circ x \circ f$ )  $\circ$  ( $\lambda x \in A' \rightarrow B'. \text{converse}(g) \circ x \circ \text{converse}(f)$ )
= id(A' → B') .
next
from ⟨f ∈ _⟩ ⟨g ∈ _⟩
have ( $\bigwedge x. x \in A \rightarrow B \implies g \circ x \circ f \in A' \rightarrow B'$ )
  using bij_is_fun comp_fun by blast
then
have ( $\lambda x \in A' \rightarrow B'. \text{converse}(g) \circ x \circ \text{converse}(f)$ )  $\circ$  ( $\lambda x \in A \rightarrow B. g \circ x \circ f$ )
  = ( $\lambda x \in A \rightarrow B. (\text{converse}(g) \circ g) \circ x \circ (f \circ \text{converse}(f))$ )
  using comp_lam comp_assoc by auto
also
have ... = ( $\lambda x \in A \rightarrow B. \text{id}(B) \circ x \circ (\text{id}(A))$ )
  using
  right_comp_inverse[OF bij_is_surj][OF ⟨f ∈ _⟩]
  left_comp_inverse[OF bij_is_inj][OF ⟨g ∈ _⟩] 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 → B) unfolding id_def by simp
finally
show ( $\lambda x \in A' \rightarrow B'. \text{converse}(g) \circ x \circ \text{converse}(f)$ )  $\circ$  ( $\lambda x \in A \rightarrow B. g \circ x \circ f$ )
= id(A → B) .
qed
qed

```

lemma *curry_eqpoll*:

fixes *d* *ν1* *ν2* *κ*

shows $\nu1 \rightarrow \nu2 \rightarrow \kappa \approx \nu1 \times \nu2 \rightarrow \kappa$

unfolding *eqpoll_def*

```

proof (intro exI, rule lam_bijjective,
  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)  $\in \kappa$ 
    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⟩  $\in \kappa$  by simp
next — one composition is the identity:
  fix f
  assume f :  $\nu 1 \times \nu 2 \rightarrow \kappa$ 
  then
  show ( $\lambda x \in \nu 1 \times \nu 2. (\lambda x \in \nu 1. \lambda xa \in \nu 2. f \langle x, xa \rangle) \langle fst(x), snd(x) \rangle = f$ )
    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)  $\equiv (\lambda x \in X. bool\_of\_o(x \in A))$ 
    — the witnessing map for the thesis:
  shows Pow(X)  $\approx X \rightarrow 2$ 
  unfolding eqpoll_def
proof (intro exI, rule lam_bijjective)
  — We give explicit mutual inverses
  fix A
  assume A  $\in Pow(X)$ 
  then
  show d(A) :  $X \rightarrow 2$ 
    using lam_type[of  $\lambda x. bool\_of\_o(x \in A)$   $\lambda \_ . 2$ ]
    by force
  from ⟨A  $\in Pow(X)$ ⟩
  show {y  $\in X. d(A) \langle y, 1 \rangle = A$ } = A
    by (auto)
next
  fix f
  assume f :  $X \rightarrow 2$ 
  then
  show d({y  $\in X . f \langle y, 1 \rangle = 1$ }) = f
    using apply_type[OF ⟨f :  $X \rightarrow 2$ ⟩]
    by (force intro:fun_extension)
qed blast

```

```

lemma cantor_inj: f  $\notin inj(Pow(A), A)$ 
  using inj_imp_surj[OF Pow_bottom] cantor_surj by blast

```

definition

$cexp :: [i, i] \Rightarrow i \langle _ \uparrow _ \rangle [76, 1] 75$ **where**
 $\kappa^{\uparrow \nu} \equiv |\nu \rightarrow \kappa|$

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

lemma eq_csucc_ord :
 $Ord(i) \Longrightarrow i^+ = |i|^+$
using $Card_lt_iff$ $Least_cong$ **unfolding** $csucc_def$ **by** $auto$

I thank Miguel Pagano for this proof.

lemma $lesspoll_csucc$:
assumes $Ord(\kappa)$
shows $d \prec \kappa^+ \longleftrightarrow d \lesssim \kappa$
proof
assume $d \prec \kappa^+$
moreover
note $Card_is_Ord \langle Ord(\kappa) \rangle$
moreover from $calculation$
have $\kappa < \kappa^+ \ Card(\kappa^+)$
using $Ord_cardinal_eqpoll$ $csucc_basic$ **by** $simp_all$
moreover from $calculation$
have $d \prec |\kappa|^+ \ Card(|\kappa|) \ d \approx |d|$
using eq_csucc_ord [of κ] $lesspoll_imp_eqpoll$ $eqpoll_sym$ **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 $\langle Ord(\kappa) \rangle$
have $\kappa < \kappa^+ \ Card(\kappa^+)$
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 $\langle d \lesssim \kappa \rangle \langle Ord(\kappa) \rangle$

```

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

```

abbreviation

```

Infinite ::  $i \Rightarrow o$  where
Infinite( $X$ )  $\equiv \neg$  Finite( $X$ )

```

```

lemma Infinite_not_empty: 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  $\langle n \in \omega \rangle$ 

```

```

proof (induct)

```

```

  case 0

```

```

  then

```

```

    show ?case using empty_lepollI by simp

```

```

next

```

```

  case (succ  $x$ )

```

```

  show ?case

```

```

  proof -

```

```

    from  $\langle$ Infinite( $X$ ) $\rangle$  and  $\langle x \in \omega \rangle$ 

```

```

    have  $\neg (x \approx X)$ 

```

```

      using eqpoll_sym unfolding Finite_def by auto

```

```

    with  $\langle x \lesssim X \rangle$ 

```

```

    obtain  $f$  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  $\langle b \in X \rangle$ 

```

```

    have  $\text{cons}(b, X - \{b\}) = X$  by auto

```

```

    ultimately

```

```

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

```

```

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

```

```

lemma lt_surj_empty_imp_Card:
  assumes  $\text{Ord}(\kappa) \wedge \alpha < \kappa \implies \text{surj}(\alpha, \kappa) = 0$ 
  shows  $\text{Card}(\kappa)$ 
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
 $f \in \text{mono_map}(A,r,B,s) \ B \subseteq C$
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 $x \in A \ y \in A \ \langle x,y \rangle \in r$
moreover from *this* **and** $\langle f: A \rightarrow B \rangle$
have $f'x \in B \ f'y \in B$
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
 $\text{linear}(\alpha,r) \ \text{trans}[\beta](s) \ \text{irrefl}(\beta,s) \ f \in \text{mono_map}(\alpha,r,\beta,s)$
 $x \in \alpha \ y \in \alpha \ \langle f'x, f'y \rangle \in s$
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 $\langle x, y \rangle \notin r \ x \in \alpha \ y \in \alpha$
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[OF 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:  $\text{irrefl}(x, \text{Memrel}(x))$ 
  unfolding irrefl_def using mem_irrefl by auto

```

```

lemmas Memrel_mono_map_reflects = linear_mono_map_reflects
  [OF well_ord_is_linear[OF well_ord_Memrel] well_ord_is_trans_on[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
     $\text{linear}(\alpha, r) \ \text{trans}[\beta](s) \ \text{irrefl}(\beta, s) \ f \in \text{mono\_map}(\alpha, r, \beta, s)$ 
  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

```

 $\text{mono\_map\_Memrel} :: [i, i] \Rightarrow i$  (infixr  $\langle \rightarrow_{<} \rangle$  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 assms $\text{mono_map_imp_ord_iso_image}$ [$\text{OF well_ord_is_linear}$ [$\text{OF well_ord_Memrel}$]
 $\text{well_ord_is_trans_on}$ [$\text{OF well_ord_Memrel}$] irrefl_Memrel] **by** 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 assms mono_map_is_fun [$\text{of } f X _ \beta$] ordertype_eq

$\text{mono_map_imp_ord_iso_image}$ [$\text{OF well_ord_is_linear}$ [$\text{OF well_ord_Memrel}$],
 $\text{THEN linear_subset}$]

$\text{well_ord_is_trans_on}$ [$\text{OF well_ord_Memrel}$] irrefl_Memrel , $\text{of } \alpha X \beta f$]

well_ord_subset [$\text{OF well_ord_Memrel}$] $\text{Image_sub_codomain}$ [$\text{of } f X \beta X$] **by**
 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 assms mono_map_is_fun ordertype_Memrel ordertype_eq [$\text{of } f \alpha \text{Memrel}(\alpha)$]

$\text{mono_map_imp_ord_iso_Memrel}$ well_ord_subset [$\text{OF well_ord_Memrel}$] $\text{Image_sub_codomain}$ [$\text{of } _ \alpha$]

by auto

lemma apply_in_image : $f: A \rightarrow B \implies a \in A \implies f'a \in f''A$

using range_eq_image apply_rangeI [$\text{of } f$] **by** 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  $\dots = \{ordermap(X, Memrel(\beta))'y . y \in \{y \in X . y \in x\}\}$ 
    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  $\langle x \in \beta \rangle$  and  $\langle Ord(\beta) \rangle$ 
  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  $\langle x \in X \rangle$  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( $X, \text{Memrel}(\beta)$ ) ' $x \leq x$ '
  using ordermap_le_arg by simp
with  $\langle x \in X \rangle$  and assms
have ordermap( $X, \text{Memrel}(\beta)$ ) ' $x \in \beta$  (is  $?y \in \_$ )'
  using ltD[of  $?y \text{ succ}(x)$ ] Ord_trans[of  $?y x \beta$ ] by auto
}
then
have ordertype( $X, \text{Memrel}(\beta)$ )  $\subseteq \beta$ 
  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
   $f \in \text{mono\_map}(\alpha, \text{Memrel}(\alpha), \beta, \text{Memrel}(\beta))$  Ord( $\alpha$ ) 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  $\text{converse}(f) \in \langle f''\alpha, \text{Memrel}(\beta) \rangle \cong \langle \alpha, \text{Memrel}(\alpha) \rangle$ 
  using ord_iso_sym by simp
with  $\langle \text{Ord}(\alpha) \rangle$ 
have  $\alpha = \text{ordertype}(f''\alpha, \text{Memrel}(\beta))$ 
  using ordertype_eq_well_ord_Memrel ordertype_Memrel by auto
also from assms
have  $\text{ordertype}(f''\alpha, \text{Memrel}(\beta)) \leq \beta$ 
  using subset_imp_ordertype_le mono_map_is_fun[of  $f$ ] 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  $f: A \rightarrow B \wedge x y. x \in A \implies y \in A \implies \langle x, y \rangle \in r \implies \langle f'x, f'y \rangle \in s$ 
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 $n \in \omega$ *InfCard*(κ)

shows $n \leq \kappa$

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 $n \in \omega$ *InfCard*(κ)

shows $n \in \kappa$

using *assms le_imp_subset[of ω κ]*

unfolding *InfCard_def* **by** *auto*

2.6 Alephs are infinite cardinals

lemma *Aleph_zero_eq_nat:* $\aleph_0 = \omega$

unfolding *Aleph_def* **by** *simp*

lemma *InfCard_Aleph:*

notes *Aleph_zero_eq_nat[simp]*

assumes *Ord*(α)

shows *InfCard*(\aleph_α)

proof -

have $\neg (\aleph_\alpha \in \omega)$

proof (*cases* $\alpha=0$)

case *True*

then show *?thesis* **using** *mem_irrefl* **by** *auto*

next

case *False*

with $\langle \text{Ord}(\alpha) \rangle$

have $\omega \in \aleph_\alpha$ **using** *Ord_0_lt[of α] ltD* **by** (*auto dest:Aleph_increasing*)

then show *?thesis* **using** *foundation* **by** *blast*

qed

with $\langle \text{Ord}(\alpha) \rangle$

have $\neg (|\aleph_\alpha| \in \omega)$

using *Card_cardinal_eq* **by** *auto*

then

have $\neg \text{Finite}(\aleph_\alpha)$ **by** *auto*

with $\langle \text{Ord}(\alpha) \rangle$

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'*($\{rec_constr(f, \beta). \beta \in \alpha\}$)
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

$\text{cofinal} :: [i,i,i] \Rightarrow o$ **where**
 $\text{cofinal}(X,A,r) \equiv \forall a \in A. \exists x \in X. \langle a,x \rangle \in r \vee a = x$

A function is cofinal if its range is.

definition

$\text{cofinal_fun} :: [i,i,i] \Rightarrow o$ **where**
 $\text{cofinal_fun}(f,A,r) \equiv \forall a \in A. \exists x \in \text{domain}(f). \langle a,f'x \rangle \in r \vee a = f'x$

lemma *cofinal_funI*:

assumes $\bigwedge a. a \in A \implies \exists x \in \text{domain}(f). \langle a,f'x \rangle \in r \vee a = f'x$
shows $\text{cofinal_fun}(f,A,r)$
using *assms* **unfolding** *cofinal_fun_def* **by** *simp*

lemma *cofinal_funD*:

assumes $\text{cofinal_fun}(f,A,r)$ $a \in A$
shows $\exists x \in \text{domain}(f). \langle a,f'x \rangle \in r \vee a = f'x$
using *assms* **unfolding** *cofinal_fun_def* **by** *simp*

lemma *cofinal_in_cofinal*:

assumes
 $\text{trans}(r)$ $\text{cofinal}(Y,X,r)$ $\text{cofinal}(X,A,r)$
shows
 $\text{cofinal}(Y,A,r)$
unfolding *cofinal_def*

proof

fix a
assume $a \in A$
moreover from $\langle \text{cofinal}(X,A,r) \rangle$
have $b \in A \implies \exists x \in X. \langle b,x \rangle \in r \vee b = x$ **for** b
unfolding *cofinal_def* **by** *simp*
ultimately
obtain y **where** $y \in X$ $\langle a,y \rangle \in r \vee a = y$ **by** *auto*
moreover from $\langle \text{cofinal}(Y,X,r) \rangle$
have $c \in X \implies \exists y \in Y. \langle c,y \rangle \in r \vee c = y$ **for** c
unfolding *cofinal_def* **by** *simp*
ultimately
obtain x **where** $x \in Y$ $\langle y,x \rangle \in r \vee y = x$ **by** *auto*
with $\langle a \in A \rangle$ $\langle y \in X \rangle$ $\langle \langle a,y \rangle \in r \vee a = y \rangle$ $\langle \text{trans}(r) \rangle$
show $\exists x \in Y. \langle a,x \rangle \in r \vee a = x$ **unfolding** *trans_def* **by** *auto*
qed

lemma *codomain_is_cofinal*:

assumes $\text{cofinal_fun}(f,A,r)$ $f: C \rightarrow D$
shows $\text{cofinal}(D,A,r)$
unfolding *cofinal_def*

proof

fix b
assume $b \in A$

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 $\text{cofinal}(\text{range}(f), A, r) \longleftrightarrow \text{cofinal_fun}(f, A, r)$
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 $\langle x, y \rangle \in f$
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 $\text{cofinal}(\text{range}(f), A, r)$
using *function_apply_Pair[of f]* **unfolding** *cofinal_def* **by** *fast*
qed

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

```

have f:C → D
  using mono_map_is_fun by simp
with ⟨h:B → C⟩
have domain(f) = C domain(h) = B
  using domain_of_fun by simp_all
moreover
assume a ∈ D
moreover
note ⟨cofinal_fun(f,D,r)⟩
ultimately
obtain c where c∈C ⟨a, f ' c⟩ ∈ r ∨ a = f ' c
  unfolding cofinal_fun_def by blast
with ⟨cofinal_fun(h,C,s)⟩ ⟨domain(h) = B⟩
obtain b where b ∈ B ⟨c, h ' b⟩ ∈ s ∨ c = h ' b
  unfolding cofinal_fun_def by blast
moreover from this and ⟨h:B → C⟩
have h'b ∈ C by simp
moreover
note ⟨f ∈ mono_map(C,s,D,r)⟩ ⟨c∈C⟩
ultimately
have ⟨f'c, f' (h ' b)⟩ ∈ r ∨ f'c = f' (h ' b)
  unfolding mono_map_def by blast
with ⟨⟨a, f ' c⟩ ∈ r ∨ a = f ' c⟩ ⟨trans(r)⟩ ⟨h:B → C⟩ ⟨b∈B⟩
have ⟨a, (f O h) ' b⟩ ∈ r ∨ a = (f O h) ' b
  using transD by auto
moreover from ⟨h:B → C⟩ ⟨domain(f) = C⟩ ⟨domain(h) = B⟩
have domain(f O h) = B
  using range_fun_subset_codomain by blast
moreover
note ⟨b∈B⟩
ultimately
show ∃ x∈domain(f O h). ⟨a, (f O h) ' x⟩ ∈ r ∨ a = (f O h) ' x by blast
qed

```

definition

```

cf_fun :: [i,i] ⇒ o where
cf_fun(f,α) ≡ cofinal_fun(f,α,Memrel(α))

```

lemma cf_funI[intro!]: cofinal_fun(f,α,Memrel(α)) ⇒ cf_fun(f,α)
 unfolding cf_fun_def by simp

lemma cf_funD[dest!]: cf_fun(f,α) ⇒ cofinal_fun(f,α,Memrel(α))
 unfolding cf_fun_def by simp

lemma cf_fun_comp:

```

assumes
  Ord(α) f ∈ mono_map(C,s,α,Memrel(α)) cf_fun(f,α)
  h:B → C cofinal_fun(h,C,s)
shows cf_fun(f O h,α)

```

using *assms* *cofinal_comp*[*OF* _____ *trans_Memrel*] **by** *auto*

definition

cf :: $i \Rightarrow i$ **where**

$cf(\gamma) \equiv \mu \beta. \exists A. A \subseteq \gamma \wedge cofinal(A, \gamma, Memrel(\gamma)) \wedge \beta = ordertype(A, Memrel(\gamma))$

lemma *Ord_cf* [*TC*]: $Ord(cf(\beta))$

unfolding *cf_def* **using** *Ord_Least* **by** *simp*

lemma *gamma_cofinal_gamma*:

assumes $Ord(\gamma)$

shows $cofinal(\gamma, \gamma, Memrel(\gamma))$

unfolding *cofinal_def* **by** *auto*

lemma *cf_is_ordertype*:

assumes $Ord(\gamma)$

shows $\exists A. A \subseteq \gamma \wedge cofinal(A, \gamma, Memrel(\gamma)) \wedge cf(\gamma) = ordertype(A, Memrel(\gamma))$

(**is** $?P(cf(\gamma))$)

using *gamma_cofinal_gamma* *LeastI*[*of* $?P \ \gamma$] *ordertype_Memrel*[*symmetric*]

assms

unfolding *cf_def* **by** *blast*

lemma *cf_fun_succ'*:

assumes $Ord(\beta) \ Ord(\alpha) \ f: \alpha \rightarrow succ(\beta)$

shows $(\exists x \in \alpha. f'x = \beta) \longleftrightarrow cf_fun(f, succ(\beta))$

proof (*intro iffI*)

assume $(\exists x \in \alpha. f'x = \beta)$

with *assms*

show $cf_fun(f, succ(\beta))$

using *domain_of_fun*[*OF* $\langle f: \alpha \rightarrow succ(\beta) \rangle$]

unfolding *cf_fun_def* *cofinal_fun_def* **by** *auto*

next

assume $cf_fun(f, succ(\beta))$

with *assms*

obtain x **where** $x \in \alpha \ \langle \beta, f'x \rangle \in Memrel(succ(\beta)) \vee \beta = f'x$

using *domain_of_fun*[*OF* $\langle f: \alpha \rightarrow succ(\beta) \rangle$]

unfolding *cf_fun_def* *cofinal_fun_def* **by** *auto*

moreover from $\langle Ord(\beta) \rangle$

have $\langle \beta, y \rangle \notin Memrel(succ(\beta))$ **for** y

using *foundation* **unfolding** *Memrel_def* **by** *blast*

ultimately

show $\exists x \in \alpha. f'x = \beta$ **by** *blast*

qed

lemma *cf_fun_succ*:

$Ord(\beta) \implies f: 1 \rightarrow succ(\beta) \implies f'0 = \beta \implies cf_fun(f, succ(\beta))$

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( $\alpha$ )))
proof -
  from  $\langle A = \{f'0\} \rangle$ 
  have  $A \approx 1$  using singleton_eqpoll_1 by simp
  with  $\langle \text{well\_ord}(A, \text{Memrel}(\text{succ}(\alpha))) \rangle$ 
  show ?thesis using nat_1I ordertype_eq_n by simp
qed
ultimately
show cf(succ( $\alpha$ )) = 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:  $f \in \text{surj}(\delta, \gamma) \implies \text{cf\_fun}(f, \gamma)$ 
  unfolding surj_def cofinal_fun_def cf_fun_def
  using domain_of_fun by force

```

```

lemma cf_zero_iff:  $\text{Ord}(\alpha) \implies \text{cf}(\alpha) = 0 \longleftrightarrow \alpha = 0$ 
proof (intro iffI)
  assume  $\alpha = 0$  Ord( $\alpha$ )
  then
  show cf( $\alpha$ ) = 0 using cf_zero by simp
next
  assume cf( $\alpha$ ) = 0 Ord( $\alpha$ )
  moreover from this
  obtain A where  $A \subseteq \alpha$  cf( $\alpha$ ) = ordertype(A, Memrel( $\alpha$ ))
    cofinal(A,  $\alpha$ , Memrel( $\alpha$ ))
  using cf_is_ordertype by blast
  ultimately
  have cofinal(0,  $\alpha$ , Memrel( $\alpha$ ))
    using ordertype_zero_imp_zero[of A Memrel( $\alpha$ )] by simp
  then
  show  $\alpha = 0$ 
    unfolding cofinal_def by blast
qed

```

— TODO: define Succ (predicate for successor ordinals)

```

lemma cf_eq_one_iff:
  assumes Ord( $\gamma$ )
  shows cf( $\gamma$ ) = 1  $\longleftrightarrow (\exists \alpha. \text{Ord}(\alpha) \wedge \gamma = \text{succ}(\alpha))$ 
proof (intro iffI)
  assume  $\exists \alpha. \text{Ord}(\alpha) \wedge \gamma = \text{succ}(\alpha)$ 
  then
  show cf( $\gamma$ ) = 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  $f \equiv converse(ordermap(A, Memrel(\gamma)))$ 
  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 ordermap_bij,
      THEN bij_converse_bij, of  $\gamma$   $A$ ] bij_is_surj
    by simp
  with  $\langle cofinal(A, \gamma, Memrel(\gamma)) \rangle$ 
  have  $\forall a \in \gamma. \langle a, f'0 \rangle \in Memrel(\gamma) \vee a = f'0$ 
    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  $\gamma$   $f'0$ ]
      surj_is_fun[of  $f$   $1$   $A$ ] apply_type[of  $f$   $1$   $\lambda_. A$   $0$ ]
    unfolding lt_def
    by (auto intro: Ord_in_Ord)
qed

```

lemma *ordertype_in_cf_imp_not_cofinal*:

```

  assumes
     $ordertype(A, Memrel(\gamma)) \in cf(\gamma)$ 
     $A \subseteq \gamma$ 
  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(\gamma))]
    by auto

```

qed

lemma *cofinal_mono_map_cf*:

assumes $Ord(\gamma)$

shows $\exists j \in mono_map(cf(\gamma), Memrel(cf(\gamma)), \gamma, Memrel(\gamma)) . cf_fun(j, \gamma)$

proof -

note *assms*

moreover from *this*

obtain A **where** $A \subseteq \gamma$ $cf(\gamma) = ordertype(A, Memrel(\gamma))$

cofinal(A, γ , Memrel(γ))

using *cf_is_ordertype* **by** *blast*

moreover

define j **where** $j \equiv converse(ordertype(A, Memrel(\gamma)))$

moreover from *calculation*

have $j : cf(\gamma) \rightarrow_{<} \gamma$

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 $j \in surj(cf(\gamma), A)$

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 $\langle cofinal(A, \gamma, Memrel(\gamma)) \rangle$

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 $\langle j \in mono_map(_, _, _, _) \rangle$

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  $j \delta \xi \gamma f$ 
  assumes  $j\_mono: j : \xi \rightarrow < \gamma$ 
    and  $ords: Ord(\delta) Ord(\xi) Limit(\gamma)$ 
    and  $f\_type: f: \delta \rightarrow \gamma$ 
begin

```

Here, f is cofinal function from δ to γ , and the ordinal ξ is meant to be the cofinality of γ . Hence, there exists an increasing map j from ξ to γ 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

```

 $factor\_body :: [i, i, i] \Rightarrow o$  where
 $factor\_body(\beta, h, x) \equiv (x \in \delta \wedge j'\beta \leq f'x \wedge (\forall \alpha < \beta . f'(h'\alpha) < f'x)) \vee x = \delta$ 

```

definition

```

 $factor\_rec :: [i, i] \Rightarrow i$  where
 $factor\_rec(\beta, h) \equiv \mu x. factor\_body(\beta, h, x)$ 

```

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

lemma $factor_body_mono$:

```

assumes
   $\beta \in \xi \ \alpha < \beta$ 
   $factor\_body(\beta, \lambda x \in \beta. G(x), x)$ 
shows
   $factor\_body(\alpha, \lambda x \in \alpha. G(x), x)$ 

```

proof -

```

from  $\langle \alpha < \beta \rangle$ 
have  $\alpha \in \beta$  using  $ltD$  by  $simp$ 
moreover
note  $\langle \beta \in \xi \rangle$ 
moreover from  $calculation$ 
have  $\alpha \in \xi$  using  $ords \ ltD \ Ord\_cf \ Ord\_trans$  by  $blast$ 
ultimately
have  $j'\alpha \in j'\beta$  using  $j\_mono \ mono\_map\_increasing$  by  $blast$ 
moreover from  $\langle \beta \in \xi \rangle$ 
have  $j'\beta \in \gamma$ 
  using  $j\_mono \ domain\_of\_fun \ apply\_rangeI \ mono\_map\_is\_fun$  by  $force$ 
moreover from  $this$ 
have  $Ord(j'\beta)$ 

```

```

    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  $\dots \leq \text{factor\_rec}(\beta, \lambda x \in \beta. \text{factor}(x))$ 
    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 δ 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  $f' \text{factor}(\alpha) < f' \text{factor}(\beta)$ 
proof -
  from assms
  have  $f' ((\lambda x \in \beta. \text{factor}(x)) ' \alpha) < f' \text{factor}(\beta)$ 
    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 ξ .

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  $\in$  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 $f \circ \text{fun_factor}$ respects membership

```

fix  $\alpha \beta$ 
assume  $\langle \alpha, \beta \rangle \in \text{Memrel}(\xi)$ 
then
have  $\alpha < \beta$ 
  using  $\text{Ord\_in\_Ord}$  [of  $\xi$ ] ltI ords by blast
assume  $\alpha \in \xi \ \beta \in \xi$ 
moreover from this and assms
have  $\text{factor}(\alpha) \neq \delta \ \text{factor}(\beta) \neq \delta$  by auto
moreover
have  $\text{Ord}(\gamma) \ \gamma \neq 0$  using  $\text{ords\_Limit\_is\_Ord}$  by auto
moreover
note  $\langle \alpha < \beta \rangle \ \langle \text{fun\_factor} : \xi \rightarrow \delta \rangle$ 
ultimately
show  $\langle (f \circ \text{fun\_factor}) \ \alpha, (f \circ \text{fun\_factor}) \ \beta \rangle \in \text{Memrel}(\gamma)$ 
  using  $\text{ltD}$  [of  $f \ \text{factor}(\alpha) \ f \ \text{factor}(\beta)$ ]
   $\text{f\_factor\_increasing\_apply\_in\_codomain\_Ord\_f\_type}$ 
  unfolding  $\text{fun\_factor\_def}$  by auto
qed

end — cofinal_factor

```

We state next the factorization lemma.

```

lemma cofinal_fun_factorization:
notes  $\text{le\_imp\_subset}$  [dest]  $\text{lt\_trans2}$  [trans]
assumes
   $\text{Ord}(\delta) \ \text{Limit}(\gamma) \ f : \delta \rightarrow \gamma \ \text{cf\_fun}(f, \gamma)$ 
shows
   $\exists g \in \text{cf}(\gamma) \rightarrow < \delta. \ f \circ g : \text{cf}(\gamma) \rightarrow < \gamma \wedge$ 
   $\text{cofinal\_fun}(f \circ g, \gamma, \text{Memrel}(\gamma))$ 

```

```

proof -
from  $\langle \text{Limit}(\gamma) \rangle$ 
have  $\text{Ord}(\gamma)$  using  $\text{Limit\_is\_Ord}$  by simp
then
obtain  $j$  where  $j : \text{cf}(\gamma) \rightarrow < \gamma \ \text{cf\_fun}(j, \gamma)$ 
  using  $\text{cofinal\_mono\_map\_cf}$  by blast
then
have  $\text{domain}(j) = \text{cf}(\gamma)$ 
  using  $\text{domain\_of\_fun\_mono\_map\_is\_fun}$  by force
from  $\langle j \in \_ \rangle$  assms
interpret  $\text{cofinal\_factor } j \ \delta \ \text{cf}(\gamma)$ 
  by ( $\text{unfold\_locales}$ ) ( $\text{simp\_all}$ )

```

The core of the argument is to show that the factor function indeed maps into δ , 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 $\text{cf}(\gamma)$ can be cofinal in γ by definition of cofinality. Hence there must be a witness that satisfies the first disjunct.

have *factor_not_delta*: $\text{factor}(\beta) \neq \delta$ **if** $\beta \in \text{cf}(\gamma)$ **for** β

For this, we induct on β ranging over $\text{cf}(\gamma)$.

```

proof (induct  $\beta$  rule:Ord_induct[OF _ Ord_cf[of  $\gamma$ ]])
  case 1 with that show ?case .
next
  case (2  $\beta$ )
  then
  have IH:  $z \in \beta \implies \text{factor}(z) \neq \delta$  for  $z$  by simp
  define h where  $h \equiv \lambda x \in \beta. f' \text{factor}(x)$ 
  from IH
  have  $z \in \beta \implies \text{factor}(z) \in \delta$  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  $\text{factor}(y) \neq \delta$  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  $\langle \beta \in \text{cf}(\gamma) \rangle$  and  $\langle y \in \beta \rangle$ 
    have  $y \in \text{cf}(\gamma)$ 
      using Ord_trans Ord_cf by blast
    moreover from this
    have Ord( $y$ )
      using Ord_cf Ord_in_Ord by blast
    moreover
    assume  $\langle x, y \rangle \in \text{Memrel}(\beta)$ 
    moreover from calculation
    have  $x < y$  by (blast intro:ltI)
    ultimately
    show  $\langle h'x, h'y \rangle \in \text{Memrel}(\gamma)$ 
      unfolding h_def using f_factor_increasing ltD by (auto)
  qed
  with  $\langle \beta \in \text{cf}(\gamma) \rangle\ \langle \text{Ord}(\gamma) \rangle$ 
  have  $\text{ordertype}(h''\beta, \text{Memrel}(\gamma)) = \beta$ 
    using mono_map_ordertype_image[of  $\beta$ ] Ord_cf Ord_in_Ord by blast
  also
  note  $\langle \beta \in \text{cf}(\gamma) \rangle$ 
  finally
  have  $\text{ordertype}(h''\beta, \text{Memrel}(\gamma)) \in \text{cf}(\gamma)$  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  $\alpha\_0 \in \gamma \ \forall x \in h \ \text{''} \beta. \neg \langle \alpha\_0, x \rangle \in \text{Memrel}(\gamma) \wedge \alpha\_0 \neq x$ 
  unfolding cofinal_def by auto
with  $\langle \text{Ord}(\gamma) \rangle \ \langle h''\beta \subseteq \gamma \rangle$ 
have  $\forall x \in h \ \text{''} \beta. x \in \alpha\_0$ 
  using well_ord_Memrel[of  $\gamma$ ] well_ord_is_linear[of  $\gamma \ \text{Memrel}(\gamma)$ ]
  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  $\langle \text{cf\_fun}(f, \gamma) \rangle$ 
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  $\langle \forall x \in h \ \text{''} \beta. x \in \alpha\_0 \rangle \ \langle \text{Ord}(\alpha\_0) \rangle \ \langle h: \beta \rightarrow \gamma \rangle$ 
have  $x \in \beta \implies h'x < \alpha\_0$  for  $x$ 
  using fun_is_function[of  $h \ \beta \ \lambda_. \ \gamma$ ]
  Image_subset_Ord_imp_lt domain_of_fun[of  $h \ \beta \ \lambda_. \ \gamma$ ]
  by blast
moreover
have  $x \in \beta \implies h'x < f'\vartheta$  for  $x$ 
proof -
  fix  $x$ 
  assume  $x \in \beta$ 
  with  $\langle \forall x \in h \ \text{''} \beta. x \in \alpha\_0 \rangle \ \langle \text{Ord}(\alpha\_0) \rangle \ \langle h: \beta \rightarrow \gamma \rangle$ 
  have  $h'x < \alpha\_0$ 
    using fun_is_function[of  $h \ \beta \ \lambda_. \ \gamma$ ]
    Image_subset_Ord_imp_lt domain_of_fun[of  $h \ \beta \ \lambda_. \ \gamma$ ]
    by blast
  also from  $\langle \langle \alpha\_0 \cup \_, f' \vartheta \rangle \in \text{Memrel}(\gamma) \vee \alpha\_0 \cup \_ = f' \vartheta \rangle$ 
   $\langle \text{Ord}(f'\vartheta) \rangle \ \langle \text{Ord}(\alpha\_0) \rangle \ \langle \text{Ord}(j'\beta) \rangle$ 
  have  $\alpha\_0 \leq f'\vartheta$ 
    using Un_leD1[OF leI [OF ltI]] Un_leD1[OF le_eqI] by blast

```

```

    finally
      show  $h'x < f'\vartheta$  .
    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( $f \ O \ \text{fun\_factor}, \gamma, \text{Memrel}(\gamma)$ )
    proof (intro cofinal_funI)
      fix a
      assume  $a \in \gamma$ 
      with  $\langle \text{cf\_fun}(j, \gamma) \rangle \langle \text{domain}(j) = \text{cf}(\gamma) \rangle$ 
      obtain x where  $x \in \text{cf}(\gamma) \ a \in j'x \vee a = j'x$ 
        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  $a \in (f \ O \ \text{fun\_factor}) \ 'x \vee a = (f \ O \ \text{fun\_factor}) \ 'x$ 
      proof -
        from  $\langle x \in \text{cf}(\gamma) \rangle$  factor_not_delta
        have  $j \ 'x \leq f \ ' \text{factor}(x)$ 
          using mem_not_refl factor_body_factor factor_in_delta
          unfolding factor_body_def by auto
        with  $\langle a \in j'x \vee a = j'x \rangle$ 
        have  $a \in f \ ' \text{factor}(x) \vee a = f \ ' \text{factor}(x)$ 
          using ltD by blast
        with  $\langle x \in \text{cf}(\gamma) \rangle$ 
        show ?thesis using lam_funtype[of  $\text{cf}(\gamma)$  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  $(f \ O \ \text{fun\_factor}) \ 'x \in \gamma$ 
      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  $\exists x \in \text{domain}(f \ O \ \text{fun\_factor}). \langle a, (f \ O \ \text{fun\_factor}) \ ' \ x \rangle \in \text{Memrel}(\gamma)$ 
       $\vee a = (f \ O \ \text{fun\_factor}) \ ' \ x$ 
  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( $\gamma$ )
   $f : \delta \rightarrow_{<} \gamma \ f \ O \ g \in \text{mono\_map}(\alpha, r, \gamma, \text{Memrel}(\gamma))$ 
   $\text{cofinal\_fun}(f \ O \ g, \gamma, \text{Memrel}(\gamma)) \ g : \alpha \rightarrow \delta$ 
shows
   $\text{cf\_fun}(g, \delta)$ 
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  $\langle \text{cofinal\_fun}(f \ O \ g, \gamma, \_) \rangle$ 
  obtain y where  $y \in \alpha \ \langle f \ ' \ a, (f \ O \ g) \ ' \ y \rangle \in \text{Memrel}(\gamma) \vee f \ ' \ a = (f \ O \ g) \ ' \ y$ 
    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  $\langle f \ ' \ a, f \ ' \ (g \ ' \ y) \rangle \in \text{Memrel}(\gamma) \vee f \ ' \ a = f \ ' \ (g \ ' \ y) \ g \ ' \ y \in \delta$ 
    using comp_fun_apply[of  $g \ \alpha \ \delta \ y \ f$ ] by auto
  with  $\text{assms}(1-3)$  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  $\langle \text{Ord}(\delta) \rangle$ 
  show ?thesis using Ord_0_le by simp
next
  case (succ  $\gamma$ )
  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  $\langle f'x=\gamma \rangle$ 
  have  $?f:1\rightarrow\text{succ}(\gamma)$ 
  using singleton_0 singleton_fun[of 0  $\gamma$ ] singleton_subsetI fun_weaken_type
by simp
  with  $\langle \text{Ord}(\gamma) \rangle$   $\langle f'x=\gamma \rangle$ 
  have  $\text{cf}(\text{succ}(\gamma)) = 1$  using cf_succ by simp
  with  $\langle \delta\neq 0 \rangle$  succ
  show ?thesis using Ord_0_lt_iff_succ_leI  $\langle \text{Ord}(\delta) \rangle$  by simp
next
  case (limit)
  with assms
  obtain  $g$  where  $g:\text{cf}(\gamma) \rightarrow_{<} \delta$ 
  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( $A,\gamma,\text{Memrel}(\gamma)$ )
shows
  cf( $\gamma$ ) = cf(ordertype( $A,\text{Memrel}(\gamma)$ ))
proof (intro le_anti_sym)

```

We show the result by proving the two inequalities.

```

from  $\langle \text{Limit}(\gamma) \rangle$ 
have Ord( $\gamma$ )
  using Limit_is_Ord by simp
with  $\langle A\subseteq\gamma \rangle$ 
have well_ord( $A,\text{Memrel}(\gamma)$ )
  using well_ord_Memrel well_ord_subset by blast
then
obtain  $f$   $\alpha$  where  $f:\langle \alpha, \text{Memrel}(\alpha) \rangle \cong \langle A,\text{Memrel}(\gamma) \rangle$  Ord( $\alpha$ )  $\alpha = \text{order-}$ 

```

```

type(A, Memrel( $\gamma$ ))
  using ordertype_ord_iso Ord_ordertype ord_iso_sym by blast
  moreover from this
  have f:  $\alpha \rightarrow A$ 
    using ord_iso_is_mono_map mono_map_is_fun[of f Memrel( $\alpha$ )] by blast
  moreover from this
  have function(f)
    using fun_is_function by simp
  moreover from  $\langle f: \langle \alpha, \text{Memrel}(\alpha) \rangle \cong \langle A, \text{Memrel}(\gamma) \rangle \rangle$ 
  have range(f) = A
    using ord_iso_is_bij bij_is_surj surj_range by blast
  moreover note  $\langle \text{cofinal}(A, \gamma, \_) \rangle$ 
  ultimately
  have cf_fun(f,  $\gamma$ )
    using cofinal_range_iff_cofinal_fun by blast
  moreover from  $\langle \text{Ord}(\alpha) \rangle$ 
  obtain h where h : cf( $\alpha$ )  $\rightarrow_{<} \alpha$  cf_fun(h,  $\alpha$ )
    using cofinal_mono_map_cf by blast
  moreover from  $\langle \text{Ord}(\gamma) \rangle$ 
  have trans(Memrel( $\gamma$ ))
    using trans_Memrel by simp
  moreover
  note  $\langle A \subseteq \gamma \rangle$ 
  ultimately
  have cofinal_fun(f O h,  $\gamma$ , Memrel( $\gamma$ ))
    using cofinal_comp_ord_iso_is_mono_map[OF  $\langle f: \langle \alpha, \_ \rangle \cong \langle A, \_ \rangle \rangle$ ] mono_map_is_fun
      mono_map_mono by blast
  moreover from  $\langle f: \alpha \rightarrow A \rangle \langle A \subseteq \gamma \rangle \langle h \in \text{mono\_map}(cf(\alpha), \_, \alpha, \_) \rangle$ 
  have f O h : cf( $\alpha$ )  $\rightarrow \gamma$ 
    using Pi_mono[of A  $\gamma$ ] comp_fun mono_map_is_fun by blast
  moreover
  note  $\langle \text{Ord}(\gamma) \rangle \langle \text{Ord}(\alpha) \rangle \langle \alpha = \text{ordertype}(A, \text{Memrel}(\gamma)) \rangle$ 
  ultimately
  show cf( $\gamma$ )  $\leq$  cf(ordertype(A, Memrel( $\gamma$ )))
    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  $\langle f: \langle \alpha, \_ \rangle \cong \langle A, \_ \rangle \rangle \langle A \subseteq \gamma \rangle$ 
have f :  $\alpha \rightarrow_{<} \gamma$ 
  using mono_map_mono[OF ord_iso_is_mono_map] by simp
then
have f:  $\alpha \rightarrow \gamma$ 
  using mono_map_is_fun by simp
with  $\langle \text{cf\_fun}(f, \gamma) \rangle \langle \text{Limit}(\gamma) \rangle \langle \text{Ord}(\alpha) \rangle$ 
obtain g where g : cf( $\gamma$ )  $\rightarrow_{<} \alpha$ 
  f O g : cf( $\gamma$ )  $\rightarrow_{<} \gamma$ 
  cofinal_fun(f O g,  $\gamma$ , Memrel( $\gamma$ ))
  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*(γ)
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 $\dots = cf(cf(\gamma))$
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*(γ)
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 γ] *surj_is_fun* **by** *blast*
qed

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

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*(κ) **shows** *Limit*($cf(\kappa)$)
using *Ord_cf[of κ , 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 (2 α)
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*(κ)
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** $g \in \text{inj}(cf(\kappa), \nu)$
using `le_imp_lepoll` **by** `blast`
from `this` **and** `calculation(2,3,5)`
obtain f **where** $f \in \text{surj}(\nu, cf(\kappa))$ $f: \nu \rightarrow cf(\kappa)$
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(κ))`
using `surj_is_cofinal` **by** `simp`
moreover
note `h_cofinal_mono` $\langle \text{Limit}(\kappa) \rangle$
moreover from `calculation`
have `cf_fun(h O f,κ)`
using `cf_fun_comp` **by** `blast`
moreover from `calculation`
have $h \circ f \in \nu \rightarrow \kappa$
using `comp_fun` **by** `simp`
ultimately
show `?thesis` **by** `blast`
qed

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

```

note  $\langle \text{Limit}(\kappa) \rangle \langle f: \nu \rightarrow \_ \rangle \langle \alpha \in \nu \rangle$ 
moreover from this
have  $\text{Ord}(f \text{ ' } \alpha)$ 
  using  $\text{apply\_type}[of f \nu \lambda\_ . \kappa, \text{THEN } [2] \text{Ord\_in\_Ord}]$ 
  by blast
ultimately
show ?thesis
  using  $\text{Ord\_trans}[of n \text{succ}(n) f \text{ ' } \alpha] \text{ltI}$  by blast
next
case 2
have  $n \in f \text{ ' } \alpha$  by (simp add:2[symmetric])
with  $\langle \text{Limit}(\kappa) \rangle \langle f: \nu \rightarrow \_ \rangle \langle \alpha \in \nu \rangle$ 
show ?thesis
  using ltI
   $\text{apply\_type}[of f \nu \lambda\_ . \kappa, \text{THEN } [2] \text{Ord\_in\_Ord}]$ 
  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  $\langle \text{Limit}(\gamma) \rangle$ 
  moreover from this
  have  $(\lambda x \in \gamma. \aleph_x) : \gamma \rightarrow \aleph_\gamma$  (is  $?f : \_ \rightarrow \_$ )
    using lam_funtype[of  $\_ \text{Aleph}$ ] fun_weaken_type[of  $\_ \_ \_ \aleph_\gamma$ ] by blast
  moreover from  $\langle \text{Limit}(\gamma) \rangle$ 
  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  $?f \in \text{mono\_map}(\gamma, \text{Memrel}(\gamma), \aleph_\gamma, \text{Memrel}(\aleph_\gamma))$ 

```

```

    by auto
  with ‹Limit(γ)›
  have ?f ∈ ‹γ, Memrel(γ)› ≅ ‹?f“γ, Memrel(ℵγ)›
    using mono_map_imp_ord_iso_Memrel[of γ ℵγ ?f]
    Card_Aleph
  by blast
  then
  have converse(?f) ∈ ‹?f“γ, Memrel(ℵγ)› ≅ ‹γ, Memrel(γ)›
    using ord_iso_sym by simp
  with ‹Limit(γ)›
  have ordertype(?f“γ, Memrel(ℵγ)) = γ
    using ordertype_eq[OF _ well_ord_Memrel]
    ordertype_Memrel by auto
  moreover from ‹Limit(γ)›
  have cofinal(?f“γ, ℵγ, Memrel(ℵγ))
    unfolding cofinal_def
  proof (standard, intro ballI)
    fix a
    assume a ∈ ℵγ ℵγ = (⋃ i < γ. ℵi)
    moreover from this
    obtain i where i < γ a ∈ ℵi
      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 ℵi < ℵsucc(i)
      by (auto)
    ultimately
    have ‹a, ℵi› ∈ Memrel(ℵγ)
      using ltD by (auto dest: Aleph_increasing)
    moreover from ‹i < γ›
    have ℵi ∈ ?f“γ
      using ltD apply_in_image[OF ‹?f : _ → _›] by auto
    ultimately
    show ∃ x ∈ ?f “ γ. ‹a, x› ∈ Memrel(ℵγ) ∨ a = x by blast
  qed
  moreover
  note ‹?f : γ → ℵγ› ‹Limit(γ)›
  ultimately
  show cf(ℵγ) = cf(γ)
    using cf_ordertype_cofinal[OF Limit_Aleph Image_sub_codomain, of γ ?f γ
  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
     $A^{\uparrow B} = A'^{\uparrow B'}$ 
  unfolding cexp_def using cardinal_eqpoll_iff
    function_space_eqpoll_cong assms
  by simp
```

lemma *cexp_cexp_cmult*: $(\kappa^{\uparrow \nu 1})^{\uparrow \nu 2} = \kappa^{\uparrow \nu 2} \otimes \nu 1$

proof -

```
  have  $(\kappa^{\uparrow \nu 1})^{\uparrow \nu 2} = (\nu 1 \rightarrow \kappa)^{\uparrow \nu 2}$ 
    using cardinal_eqpoll
    by (intro cexp_eqpoll_cong) (simp_all add: cexp_def)
  also
  have  $\dots = \kappa^{\uparrow \nu 2} \times \nu 1$ 
    unfolding cexp_def using curry_eqpoll cardinal_cong by blast
  also
  have  $\dots = \kappa^{\uparrow \nu 2} \otimes \nu 1$ 
    using cardinal_eqpoll[THEN eqpoll_sym]
    unfolding cmult_def by (intro cexp_eqpoll_cong) (simp)
  finally
  show ?thesis .
```

qed

```
lemma cardinal_Pow:  $|Pow(X)| = 2^{\uparrow X}$  — 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*]
— $\llbracket \text{InfCard}(\kappa); \text{InfCard}(\nu) \rrbracket \implies \text{InfCard}(\kappa^{\uparrow\nu})$

4.2 Miscellaneous

lemma *cardinal_RepFun_le*: $|\{f(a) \cdot a \in A\}| \leq |A|$

proof -

have $(\lambda x \in A. f(x)) \in \text{surj}(A, \{f(a) \cdot a \in A\})$
unfolding *surj_def* **using** *lam_funtype* **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_succ_iff*: $\text{Card}(K) \implies |K'| < K^+ \iff |K'| \leq K$

by (*simp add: Card_lt_succ_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) $J \lesssim K \wedge i. i \in J \implies |X(i)| \leq K$

shows $|\bigcup i \in J. X(i)| \leq K$

proof -

from $\langle J \lesssim K \rangle$

obtain f **where** $f \in \text{inj}(J, K)$ **by** *blast*

define Y **where** $Y(k) \equiv \text{if } k \in \text{range}(f) \text{ then } X(\text{converse}(f) \text{ `} k \text{) else } 0$ **for** k

have $i \in J \implies f^i \in K$ **for** i

using *inj_is_fun*[*OF* $\langle f \in \text{inj}(J, K) \rangle$] **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 $\langle f \in \text{inj}(J, K) \rangle \ \langle i \in J \implies f^i \in K \rangle$

have $x \in Y(f^i) \ f^i \in K$

unfolding *Y_def*

using *inj_is_fun*[*OF* $\langle f \in \text{inj}(J, K) \rangle$]

right_inverse apply_rangeI **by** *auto*

then

show $x \in (\bigcup i \in K. Y(i))$ **by** *auto*

qed

then

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 $\text{inj_converse_fun}[OF \langle f \in \text{inj}(J, K) \rangle]$ **unfolding** Y_def
by $(\text{rule_tac } \text{le_trans}[OF _ \text{cardinal_UN_le}]) (\text{auto intro: Ord_0_le})+$
qed

— For backwards compatibility

lemmas $\text{lepoll_imp_cardinal_UN_le} = \text{lepoll_imp_cardinal_UN_le}$

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

lemma $\text{lepoll_imp_subset_bij}: X \lesssim Y \longleftrightarrow (\exists Z. Z \subseteq Y \wedge Z \approx X)$

proof

assume $X \lesssim Y$

then

obtain j **where** $j \in \text{inj}(X, Y)$

by blast

then

have $\text{range}(j) \subseteq Y \ j \in \text{bij}(X, \text{range}(j))$

using $\text{inj_bij_range } \text{inj_is_fun } \text{range_fun_subset_codomain}$

by $\text{blast}+$

then

show $\exists Z. Z \subseteq Y \wedge Z \approx X$

using $\text{eqpoll_sym } \text{unfolding } \text{eqpoll_def}$

by force

next

assume $\exists Z. Z \subseteq Y \wedge Z \approx X$

then

obtain $Z \ f$ **where** $f \in \text{bij}(Z, X) \ Z \subseteq Y$

unfolding eqpoll_def **by** force

then

have $\text{converse}(f) \in \text{inj}(X, Y)$

using $\text{bij_is_inj } \text{inj_weaken_type } \text{bij_converse_bij}$ **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 $\text{cardinal_Card_eqpoll_iff}: \text{Card}(\kappa) \implies |X| = \kappa \longleftrightarrow X \approx \kappa$
using $\text{Card_cardinal_eq}[of \ \kappa] \ \text{cardinal_eqpoll_iff}[of \ X \ \kappa]$ **by** auto
 — Compare le_Card_iff

lemma $\text{lepoll_imp_lepoll_cardinal}: \text{assumes } X \lesssim Y \text{ shows } X \lesssim |Y|$

using *assms cardinal_Card_eqpoll_iff*[of $|Y|$ Y]
lepoll_eq_trans[of $_ _ |Y|$] **by** *simp*

lemma *lepoll_Un*:

assumes *InfCard*(κ) $A \lesssim \kappa$ $B \lesssim \kappa$

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 κ B κ] *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 $\text{sum}(A,B)$ κ]

lepoll_trans[of $A \cup B$]

by *auto*

qed

lemma *cardinal_Un_le*:

assumes *InfCard*(κ) $|A| \leq \kappa$ $|B| \leq \kappa$

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'*: $\text{Finite}(|i|) \longleftrightarrow \text{Finite}(i)$

using *cardinal_eqpoll_iff eqpoll_imp_Finite_iff* **by** *fastforce*

lemma *cardinal_subset_of_Card*:

assumes *Card*(γ) $a \subseteq \gamma$

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*(γ) $\implies |X| < \gamma \longleftrightarrow \neg |X| \geq \gamma$

using *not_le_iff_lt*

by *auto*

4.3 Countable and uncountable sets

lemma *countable_iff_cardinal_le_nat*: $\text{countable}(X) \longleftrightarrow |X| \leq \omega$
using *le_Card_iff*[of ω 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*: $\text{countable}(X) \implies f \in \text{surj}(X, Y) \implies \text{countable}(Y)$
using *surj_implies_cardinal_le*[of f X Y , *THEN le_trans*]
countable_iff_cardinal_le_nat **by** *simp*

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

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

lemma *countable_union_countable*:
assumes $\bigwedge x. x \in C \implies \text{countable}(x)$ $\text{countable}(C)$
shows $\text{countable}(\bigcup C)$
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*:
 $\text{uncountable}(X) \longleftrightarrow \omega < |X|$
using *countable_iff_cardinal_le_nat not_le_iff_lt* **by** *simp*

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

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

lemma *uncountable_not_subset_countable*:
assumes $\text{countable}(X)$ $\text{uncountable}(Y)$
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*(α)
 shows $d < \aleph_{\text{succ}(\alpha)} \iff d \lesssim \aleph_\alpha$
 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 $a < b \implies \aleph_a < \aleph_b$
 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 $\text{uncountable}(X) \iff (\exists S. S \subseteq X \wedge S \approx \aleph_1)$

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 ω] *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

$\forall n \in domain(G). |G'n| < \aleph_1 \implies |\bigcup n \in domain(G). G'n| \leq \omega$

proof -

```

assume function( $G$ )
let  $?N = domain(G)$  and  $?R = \bigcup n \in domain(G). G'n$ 
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  $?G = \lambda n \in range(f). f^{-1}\{n\}$ 
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  $\langle \text{range}(f) \subseteq \omega \rangle$ 
have  $\aleph_1 = (\bigcup n \in \text{range}(f). f^{-1}\{n\})$ 
proof (intro equalityI, intro subsetI)
  fix  $x$ 
  assume  $x \in \aleph_1$ 
  with  $\langle f: \aleph_1 \rightarrow \omega \rangle$   $\langle \text{function}(f) \rangle$   $\langle \text{domain}(f) = \aleph_1 \rangle$ 
  have  $x \in f^{-1}\{f'x\}$   $f'x \in \text{range}(f)$ 
    using function_apply_Pair vimage_iff apply_rangeI by simp_all
  then
  show  $x \in (\bigcup n \in \text{range}(f). f^{-1}\{n\})$  by auto
qed auto
{
  assume  $\forall n \in \text{range}(f). |f^{-1}\{n\}| < \aleph_1$ 
  then
  have  $\forall n \in \text{domain}(?G). |?G'n| < \aleph_1$ 
    using zero_lt_Aleph1 by (auto)
  with  $\langle \text{function}(?G) \rangle$   $\langle \text{domain}(?G) \lesssim \omega \rangle$ 
  have  $|\bigcup n \in \text{domain}(?G). ?G'n| \leq \omega$ 
    using lt_Aleph_imp_cardinal_UN_le_nat by blast
  then
  have  $|\bigcup n \in \text{range}(f). f^{-1}\{n\}| \leq \omega$  by simp
  with  $\langle \aleph_1 = \_ \rangle$ 
  have  $|\aleph_1| \leq \omega$  by simp
  then
  have  $\aleph_1 \leq \omega$ 
    using Card_Aleph Card_cardinal_eq
    by simp
  then
  have False
    using nat_lt_Aleph1 by (blast dest:lt_trans2)
}
with  $\langle \text{range}(f) \subseteq \omega \rangle$ 
obtain  $n$  where  $n \in \omega \wedge \neg(|f^{-1}\{n\}| < \aleph_1)$ 
  by blast
moreover from this
have  $\aleph_1 \leq |f^{-1}\{n\}|$ 
  using not_lt_iff_le Card_is_Ord by auto
moreover
note  $\langle n \in \omega \implies f^{-1}\{n\} \subseteq \aleph_1 \rangle$ 
ultimately
show ?thesis
  using subset_imp_le_cardinal[THEN le_anti_sym, of _  $\aleph_1$ ]
    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 $f \circ g : \aleph_1 \rightarrow \omega$ $\text{converse}(g) \in \text{bij}(X, \aleph_1)$
using *bij_is_fun comp_fun bij_converse_bij* **by** *blast+*
then
obtain n **where** $n \in \omega$ $|(f \circ g)^{-1}\{n\}| = \aleph_1$
using *Aleph1_eq_cardinal_vimage* **by** *auto*
then
have $\aleph_1 = |\text{converse}(g)^{-1}(f^{-1}\{n\})|$
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 $\dots = |f^{-1}\{n\}|$
using *range_of_subset_eqpoll* [of $\text{converse}(g)$ X $_$ $f^{-1}\{n\}$]
bij_is_inj cardinal_cong bij_is_fun eqpoll_sym Pi_vimage_subset
by *fastforce*
finally
show *?thesis* **using** $\langle n \in \omega \rangle$ **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

$\bigwedge X. |X| < \gamma \implies X \subseteq G \implies \exists a \in G. \forall s \in X. Q(s, a) \ b \in G \ \text{Card}(\gamma)$

shows

$\exists S. S : \gamma \rightarrow G \wedge (\forall \alpha \in \gamma. \forall \beta \in \gamma. \alpha < \beta \implies Q(S^\alpha, S^\beta))$

proof -

let $?cdlt\gamma = \{X \in \text{Pow}(G) . |X| < \gamma\}$ — “cardinal less than γ ”

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  $\langle b \in G \rangle$ 
have  $Cb \in Pow(G) \text{-?cdlt} \gamma \rightarrow G$ 
  unfolding  $Cb\_def$  by simp
moreover
note  $\langle Card(\gamma) \rangle$ 
ultimately
have  $f \cup Cb : (\prod x \in Pow(G). ?inQ(x) \cup G)$  using
   $fun\_Pi\_disjoint\_Un[ of f ?cdlt \gamma ?inQ Cb Pow(G) \text{-?cdlt} \gamma \lambda \_. G]$ 
   $Diff\_partition[ of \{X \in Pow(G). |X| < \gamma\} Pow(G), OF Collect\_subset]$ 
  by auto
moreover
have  $?inQ(x) \cup G = G$  for  $x$  by auto
ultimately
have  $f \cup Cb : Pow(G) \rightarrow G$  by simp
define  $S$  where  $S \equiv \lambda \alpha \in \gamma. rec\_constr(f \cup Cb, \alpha)$ 
from  $\langle f \cup Cb : Pow(G) \rightarrow G \rangle \langle Card(\gamma) \rangle$ 
have  $S : \gamma \rightarrow G$ 
  using  $Ord\_in\_Ord$  unfolding  $S\_def$ 
  by (intro lam\_type rec\_constr\_type) auto
moreover
have  $\forall \alpha \in \gamma. \forall \beta \in \gamma. \alpha < \beta \rightarrow Q(S \text{ ' } \alpha, S \text{ ' } \beta)$ 
proof (intro ballI impI)
  fix  $\alpha \beta$ 
  assume  $\beta \in \gamma$ 
  with  $\langle Card(\gamma) \rangle$ 
have  $\{rec\_constr(f \cup Cb, x) . x \in \beta\} = \{S'x . x \in \beta\}$ 
  using  $Ord\_trans[OF \_ \_ Card\_is\_Ord, of \_ \beta \gamma]$ 
  unfolding  $S\_def$ 
  by auto
moreover from  $\langle \beta \in \gamma \rangle \langle S : \gamma \rightarrow G \rangle \langle Card(\gamma) \rangle$ 
have  $\{S'x . x \in \beta\} \subseteq G$ 
  using  $Ord\_trans[OF \_ \_ Card\_is\_Ord, of \_ \beta \gamma]$ 
   $apply\_type[ of S \gamma \lambda \_. G]$  by auto
moreover from  $\langle Card(\gamma) \rangle \langle \beta \in \gamma \rangle$ 
have  $|\{S'x . x \in \beta\}| < \gamma$ 
  using  $cardinal\_RepFun\_le[ of \beta ] Ord\_in\_Ord$ 
   $lt\_transI[ of |\{S'x . x \in \beta\}| |\beta| \gamma]$ 
   $Card\_lt\_iff[ THEN iffD2, of \beta \gamma, OF \_ \_ ltI]$ 
  by force
moreover
have  $\forall x \in \beta. Q(S'x, f \text{ ' } \{S'x . x \in \beta\})$ 
proof -
  from calculation and f\_type
  have  $f \text{ ' } \{S'x . x \in \beta\} \in \{a \in G. \forall x \in \beta. Q(S'x, a)\}$ 
  using  $apply\_type[ of f ?cdlt \gamma ?inQ \{S'x . x \in \beta\}]$ 
  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  $Q(S \text{' } \alpha, S \text{' } \beta)$ 
  using fun_disjoint_apply1[of  $\{S \text{' } x . x \in \beta\}$  Cb f]
    domain_of_fun[of Cb] ltD[of  $\alpha \beta$ ]
  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  $b \in X$ 
    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  $\langle Infinite(X) \rangle$ 
    have  $X \neq Y$  by auto
  }
  with  $\langle b \in X \rangle$ 
  obtain S where  $S : \omega \rightarrow X \ \forall \alpha \in \omega. \forall \beta \in \omega. \alpha < \beta \longrightarrow S \text{' } \alpha \neq S \text{' } \beta$ 
    using bounded_cardinal_selection[of  $\omega X \lambda x y. x \neq y$ ]
    Card_nat by blast
  moreover from this
  have  $\alpha \in \omega \implies \beta \in \omega \implies \alpha \neq \beta \implies S \text{' } \alpha \neq S \text{' } \beta$  for  $\alpha \beta$ 
    by (rule_tac lt_neq_symmetry[of  $\omega \lambda \alpha \beta. S \text{' } \alpha \neq S \text{' } \beta$ ])
      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  $F \in \text{Finite\_to\_one}(X, Y) \cap \text{surj}(X, Y)$  Infinite( $X$ )
  shows  $|Y| = |X|$ 
proof -
  from  $\langle F \in \text{Finite\_to\_one}(X, Y) \cap \text{surj}(X, Y) \rangle$ 
  have  $X = (\bigcup y \in Y. \{x \in X . F'x = y\})$ 
    using apply_type by fastforce
  show ?thesis
proof (cases Finite( $Y$ ))
  case True
  with  $\langle X = (\bigcup y \in Y. \{x \in X . F'x = y\}) \rangle$  and assms
  show ?thesis
    using Finite_RepFun[THEN [2] Finite_Union, of Y  $\lambda y. \{x \in X . F'x = y\}$ ]
    by auto
next
  case False
  moreover from this
  have  $Y \lesssim |Y|$ 
    using cardinal_eqpoll eqpoll_sym eqpoll_imp_lepoll by simp
  moreover
  note assms
  moreover from calculation
  have  $y \in Y \implies |\{x \in X . F'x = y\}| \leq |Y|$  for  $y$ 
    using Infinite_imp_nats_lepoll[THEN lepoll_imp_cardinal_le, of Y
       $|\{x \in X . F'x = y\}|$ ] cardinal_idem by auto
  ultimately
  have  $|\bigcup y \in Y. \{x \in X . F'x = y\}| \leq |Y|$ 
    using lepoll_imp_cardinal_UN_le[of |Y| Y]
      Infinite_InfCard_cardinal[of Y] by simp
  moreover from  $\langle F \in \text{Finite\_to\_one}(X, Y) \cap \text{surj}(X, Y) \rangle$ 
  have  $|Y| \leq |X|$ 
    using surj_implies_cardinal_le by auto
  moreover
  note  $\langle X = (\bigcup y \in Y. \{x \in X . F'x = y\}) \rangle$ 
  ultimately
  show ?thesis
    using le_anti_sym by auto
qed
qed

lemma cardinal_map_Un:
  assumes Infinite( $X$ ) Finite( $b$ )
  shows  $|\{a \cup b . a \in X\}| = |X|$ 
proof -
  have  $(\lambda a \in X. a \cup b) \in \text{Finite\_to\_one}(X, \{a \cup b . a \in X\})$ 
     $(\lambda a \in X. a \cup b) \in \text{surj}(X, \{a \cup b . a \in X\})$ 
    unfolding surj_def
proof
  fix  $d$ 

```

```

have Finite( $\{a \in X . a \cup b = d\}$ ) (is Finite(?Y(b,d)))
  using ‹Finite(b)›
proof (induct arbitrary:d)
  case 0
  have  $\{a \in X . a \cup 0 = d\} = (\text{if } d \in X \text{ then } \{d\} \text{ else } 0)$ 
    by auto
  then
  show ?case by simp
next
  case (cons c b)
  from ‹ $c \notin b$ ›
  have  $?Y(\text{cons}(c,b),d) \subseteq (\text{if } c \in d \text{ then } ?Y(b,d) \cup ?Y(b,d-\{c\}) \text{ else } 0)$ 
    by auto
  with cons
  show ?case
    using subset_Finite
    by simp
qed
moreover
assume  $d \in \{x \cup b . x \in X\}$ 
ultimately
show Finite( $\{a \in X . (\lambda x \in X. x \cup b) ' a = d\}$ )
  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_succ:
  assumes InfCard(z)
  shows  $cf(z^+) = z^+$ 
proof (rule ccontr)
  assume  $cf(z^+) \neq z^+$ 
  moreover from ‹InfCard(z)›
  have Ord( $z^+$ ) Ord(z) Limit(z) Limit( $z^+$ ) Card( $z^+$ ) Card(z)
    using InfCard_succ Card_is_Ord InfCard_is_Card InfCard_is_Limit
    by fastforce+
  moreover from calculation

```

have $cf(z^+) < z^+$
using $cf_le_cardinal[of\ z^+,\ THEN\ le_iff[THEN\ iffD1]]$
 $Card_cardinal_eq$
by *simp*
ultimately
obtain G **where** $G:cf(z^+) \rightarrow z^+ \forall \beta \in z^+. \exists y \in cf(z^+). \beta < G'y$
using $Limit_cofinal_fun_lt[of\ z^+ _ cf(z^+)]\ Ord_cf$
 $cf_le_cf_fun[of\ z^+ \ cf(z^+)]\ le_refl[of\ cf(z^+)]$
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 $apply_type[of\ G\ cf(z^+)\ \lambda_.\ z^+,\ THEN\ ltI]\ Card_lt_iff[THEN\ iffD2]$
 $Ord_in_Ord[OF\ Card_is_Ord,\ of\ z^+]\ cardinal_lt_csucc_iff[THEN\ iffD1]$
by *auto*
from $\langle cf(z^+) < z^+ \rangle \langle InfCard(z) \rangle \langle Ord(z) \rangle$
have $cf(z^+) \lesssim z$
using $cardinal_lt_csucc_iff[of\ z\ cf(z^+)]\ Card_csucc[of\ z]$
 $le_Card_iff[of\ z\ cf(z^+)]\ InfCard_is_Card$
 $Card_lt_iff[of\ cf(z^+)\ z^+]\ lt_Ord[of\ cf(z^+)\ z^+]$
by *simp*
with $\langle cf(z^+) < z^+ \rangle \langle \forall \beta \in cf(z^+). |G'\beta| \leq _ \rangle \langle InfCard(z) \rangle$
have $|\bigcup \beta \in cf(z^+). G'\beta| \leq z$
using $InfCard_csucc[of\ z]$
 $subset_imp_lepoll[THEN\ lepoll_imp_cardinal_le,\ of\ \bigcup \beta \in cf(z^+). G'\beta\ z]$
by $(rule_tac\ lepoll_imp_cardinal_UN_le)\ auto$
moreover
note $\langle Ord(z) \rangle$
moreover from $\langle \forall \beta \in z^+. \exists y \in cf(z^+). \beta < G'y \rangle$ **and this**
have $z^+ \subseteq (\bigcup \beta \in cf(z^+). G'\beta)$
by $(blast\ dest:ltD)$
ultimately
have $z^+ \leq z$
using $subset_imp_le_cardinal[of\ z^+ \ \bigcup \beta \in cf(z^+). G'\beta]\ le_trans$
 $InfCard_is_Card\ Card_csucc[of\ z]\ Card_cardinal_eq$
by *auto*
with $\langle Ord(z) \rangle$
show *False*
using $lt_csucc[of\ z]\ not_lt_iff_le[THEN\ iffD2,\ of\ z\ z^+]$
 $Card_csucc[THEN\ Card_is_Ord]$
by *auto*
qed

And this finishes the calculation of cofinality of Alephs.

lemma $cf_Aleph_succ: Ord(z) \implies cf(\aleph_{succ(z)}) = \aleph_{succ(z)}$
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( $\kappa$ ) InfCard( $\nu$ ) cf( $\kappa$ )  $\leq \nu$ 
  shows
     $\kappa < \kappa^{\uparrow \nu}$ 
  using assms(1,2) Card_cexp
proof (intro not_le_iff_lt[THEN iffD1] notI)
  assume  $\kappa^{\uparrow \nu} \leq \kappa$ 
  moreover
  note  $\langle \text{InfCard}(\kappa) \rangle$ 
  moreover from calculation
  have  $\nu \rightarrow \kappa \lesssim \kappa$ 
    using Card_cardinal_eq[OF InfCard_is_Card, symmetric]
      Card_le_imp_lepoll
    unfolding cexp_def by simp
  ultimately
  obtain G where  $G \in \text{surj}(\kappa, \nu \rightarrow \kappa)$ 
    using inj_imp_surj[OF _ function_space_nonempty,
      OF _ nat_into_InfCard] by blast
  from assms
  obtain f where  $f: \nu \rightarrow \kappa$  cf_fun(f, $\kappa$ )
    using cf_le_cf_fun[OF _ InfCard_is_Limit] by blast
  define H where  $H(\alpha) \equiv \mu x. x \in \kappa \wedge (\forall m < f'\alpha. G'm'\alpha \neq x)$ 
    (is  $\_ \equiv \mu x. ?P(\alpha, x)$ ) for  $\alpha$ 
  have H_satisfies:  $?P(\alpha, H(\alpha))$  if  $\alpha \in \nu$  for  $\alpha$ 
proof -
  obtain h where  $?P(\alpha, h)$ 
proof -
  from  $\langle \alpha \in \nu \rangle \langle f: \nu \rightarrow \kappa \rangle \langle \text{InfCard}(\kappa) \rangle$ 
  have  $f'\alpha < \kappa$ 
    using apply_type[of _ _  $\lambda \_ . \kappa$ ] by (auto intro: ltI)
  have  $|\{G'm'\alpha . m \in \{x \in \kappa . x < f'\alpha\}\}| \leq |\{x \in \kappa . x < f'\alpha\}|$ 
    using cardinal_RepFun_le by simp
  also from  $\langle f'\alpha < \kappa \rangle \langle \text{InfCard}(\kappa) \rangle$ 
  have  $|\{x \in \kappa . x < f'\alpha\}| < |\kappa|$ 
    using Card_lt_iff[OF lt_Ord, THEN iffD2, of f'\alpha  $\kappa$   $\kappa$ ]
      Ord_eq_Collect_lt[of f'\alpha  $\kappa$ ] Card_cardinal_eq
    by force
  finally
  have  $|\{G'm'\alpha . m \in \{x \in \kappa . x < f'\alpha\}\}| < |\kappa|$  .

```

```

moreover from  $\langle f'\alpha < \kappa \rangle \langle \text{InfCard}(\kappa) \rangle$ 
have  $m < f'\alpha \implies m \in \kappa$  for  $m$ 
  using  $\text{Ord\_trans}[of\ m\ f'\alpha\ \kappa]$ 
  by  $(\text{auto}\ \text{dest:ltD})$ 
ultimately
have  $\exists h. ?P(\alpha, h)$ 
  using  $\text{lt\_cardinal\_imp\_not\_subset}$  by  $\text{blast}$ 
with  $\text{that}$ 
show  $?thesis$  by  $\text{blast}$ 
qed
with  $\text{assms}$ 
show  $?P(\alpha, H(\alpha))$ 
  using  $\text{LeastI}[of\ ?P(\alpha)\ h]\ \text{lt\_Ord}\ \text{Ord\_in\_Ord}$ 
  unfolding  $H\_def$  by  $\text{fastforce}$ 
qed
then
have  $(\lambda \alpha \in \nu. H(\alpha)): \nu \rightarrow \kappa$ 
  using  $\text{lam\_type}$  by  $\text{auto}$ 
with  $\langle G \in \text{surj}(\kappa, \nu \rightarrow \kappa) \rangle$ 
obtain  $n$  where  $n \in \kappa\ G'n = (\lambda \alpha \in \nu. H(\alpha))$ 
  unfolding  $\text{surj\_def}$  by  $\text{blast}$ 
moreover
note  $\langle \text{InfCard}(\kappa) \rangle \langle f: \nu \rightarrow \kappa \rangle \langle \text{cf\_fun}(f, \_) \rangle$ 
ultimately
obtain  $\alpha$  where  $n < f'\alpha\ \alpha \in \nu$ 
  using  $\text{Limit\_cofinal\_fun\_lt}[OF\ \text{InfCard\_is\_Limit}]$  by  $\text{blast}$ 
moreover from  $\text{calculation}$  and  $\langle G'n = (\lambda \alpha \in \nu. H(\alpha)) \rangle$ 
have  $G'n'\alpha = H(\alpha)$ 
  using  $\text{ltD}$  by  $\text{simp}$ 
moreover from  $\text{calculation}$  and  $H\_satisfies$ 
have  $\forall m < f'\alpha. G'm'\alpha \neq H(\alpha)$  by  $\text{simp}$ 
ultimately
show  $\text{False}$  by  $\text{blast}$ 
qed  $\text{blast+}$ 

lemma  $\text{cf\_cexp}$ :
assumes
   $\text{Card}(\kappa)\ \text{InfCard}(\nu)\ 2 \leq \kappa$ 
shows
   $\nu < \text{cf}(\kappa^{\uparrow \nu})$ 
proof  $(\text{rule}\ \text{ccontr})$ 
assume  $\neg \nu < \text{cf}(\kappa^{\uparrow \nu})$ 
with  $\langle \text{InfCard}(\nu) \rangle$ 
have  $\text{cf}(\kappa^{\uparrow \nu}) \leq \nu$ 
  using  $\text{not\_lt\_iff\_le}\ \text{Ord\_cf}\ \text{InfCard\_is\_Card}\ \text{Card\_is\_Ord}$  by  $\text{simp}$ 
moreover
note  $\text{assms}$ 
moreover from  $\text{calculation}$ 
have  $\text{InfCard}(\kappa^{\uparrow \nu})$  using  $\text{InfCard\_cexp}$  by  $\text{simp}$ 

```

```

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  $\forall A \in F. \text{Finite}(A) \ F \approx \aleph_1$ 
  shows  $\exists D. D \subseteq F \wedge \text{delta\_system}(D) \wedge D \approx \aleph_1$ 
proof -

```

Since all members are finite,

```

from  $\langle \forall A \in F. \text{Finite}(A) \rangle$ 
have  $(\lambda A \in F. |A|) : F \rightarrow \omega$  (is ?cards : _)
  by (rule_tac lam_type) simp
moreover from this
have  $a : ?\text{cards} - \{n\} = \{ A \in F . |A| = n \}$  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** $n \in \omega \mid |?cards - \{n\}| = \aleph_1$
using *eqpoll_Aleph1_cardinal_vimage*[of F $?cards$] **by** *auto*
moreover
define G **where** $G \equiv ?cards - \{n\}$
moreover from *calculation*
have $G \subseteq F$ **by** *auto*
ultimately

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

have $A \in G \implies |A| = n$ **and** $G \approx \aleph_1$ **for** A
using *cardinal_Card_eqpoll_iff* **by** *auto*
with $\langle n \in \omega \rangle$

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 $\exists D. D \subseteq G \wedge \text{delta_system}(D) \wedge D \approx \aleph_1$
proof (*induct arbitrary:G*)
case 0 — This case is impossible
then
have $G \subseteq \{0\}$
using *cardinal_0_iff_0* **by** *auto*
with $\langle G \approx \aleph_1 \rangle$
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 $\forall a \in G. \text{Finite}(a)$
using *Finite_cardinal_iff' nat_into_Finite*[of *succ(n)*]
by *fastforce*
show $\exists D. D \subseteq G \wedge \text{delta_system}(D) \wedge D \approx \aleph_1$
proof (*cases* $\exists p. \{A \in G . p \in A\} \approx \aleph_1$)
case *True* — the positive case, uncountably many sets with a common element
then
obtain p **where** $\{A \in G . p \in A\} \approx \aleph_1$ **by** *blast*
moreover from *this*
have $\{A - \{p\} . A \in \{X \in G. p \in X\}\} \approx \aleph_1$ (**is** $?F \approx _$)
using *Diff_bij*[of $\{A \in G . p \in A\}$ $\{p\}$]
comp_bij[OF bij_converse_bij, where C= \aleph_1] **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** $D \subseteq \{A - \{p\} . A \in \{X \in G. p \in X\}\}$ *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 *delta_system*($\{B \cup \{p\} . B \in D\}$) (**is** *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 {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  $\langle \neg (\exists p. \{A \in G . p \in A\} \approx \aleph_1) \rangle$  — 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  $\{A \in G . S \cap A \neq 0\} = (\bigcup_{p \in S} ?G(p))$  for S
  by auto
ultimately
have countable(S)  $\implies$  countable( $\{A \in G . S \cap A \neq 0\}$ ) 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  $\exists A \in G. \forall S \in X. S \cap A = 0$  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  $\langle \text{countable}(\_) \implies \text{countable}(\{A \in G . \_ \cap A \neq 0\}) \rangle$ 
  have countable( $\{A \in G . (\bigcup X) \cap A \neq 0\}$ ) .
  with  $\langle G \approx \aleph_1 \rangle$ 
  obtain B where  $B \in G$   $B \notin \{A \in G . (\bigcup X) \cap A \neq 0\}$ 
    using nat_lt_Aleph1 cardinal_Card_eqpoll_iff[of  $\aleph_1$  G]
    uncountable_not_subset_countable[of  $\{A \in G . (\bigcup X) \cap A \neq 0\}$  G]

```

```

    uncountable_iff_nat_lt_cardinal
  by auto
  then
  show  $\exists A \in G. \forall S \in X. S \cap A = 0$  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  $S : \aleph_1 \rightarrow G$   $\alpha \in \aleph_1 \implies \beta \in \aleph_1 \implies \alpha < \beta \implies S'\alpha \cap S'\beta = 0$ 
    for  $\alpha \beta$ 
    using bounded_cardinal_selection[of  $\aleph_1$   $G$   $\lambda s a. s \cap a = 0$   $b$ ]
    by force
  then
  have  $\alpha \in \aleph_1 \implies \beta \in \aleph_1 \implies \alpha \neq \beta \implies S'\alpha \cap S'\beta = 0$  for  $\alpha \beta$ 
    using lt_neq_symmetry[of  $\aleph_1$   $\lambda \alpha \beta. S'\alpha \cap S'\beta = 0$ ] 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  $x = S'\aleph_1$  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  $S \subseteq F$   $S \approx \aleph_1$ 
    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 $i \Rightarrow o$).

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( $P, \text{leq}, A$ )  $\equiv A \subseteq P \wedge (\forall p \in A. \forall q \in A. p \neq q \longrightarrow \neg \text{compat\_in}(P, \text{leq}, p, q))$ 

```

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( $P, \text{leq}$ )  $\equiv \forall A. \text{antichain}(P, \text{leq}, A) \longrightarrow \text{countable}(A)$ 

```

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** $\langle \supseteq \rangle$ 50) **where**
 $f \supseteq g \equiv g \subseteq f$

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*
 $restrict_eq_imp_Un_into_Pi[of f d1 \lambda_. J \ g d2 \lambda_. J]$
by *auto*

qed

We finally arrive to our application of DSL.

lemma *ccc_Fn_2*: $ccc(Fn(I, 2), (\supseteq))$ **proof** -

{
fix A
assume $\neg countable(A)$
assume $A \subseteq Fn(I, 2)$
moreover from *this*
have $countable(\{p \in A. domain(p) = d\})$ **for** d
proof (*cases* $Finite(d) \wedge d \subseteq I$)
case *True*
with $\langle A \subseteq Fn(I, 2) \rangle$
have $\{p \in A . domain(p) = d\} \subseteq d \rightarrow 2$
using *domain_of_fun* **by** *fastforce*
moreover from *True*
have $Finite(d \rightarrow 2)$

```

    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 ≈ ℵ1
  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 ≈ ℵ1⟩
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 ∧ do-
main(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.
- [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, 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, 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.
- [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.