Cofinality and the Delta System Lemma

Pedro Sánchez Terraf[*][†]

May 26, 2024

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

[*]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/

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* (‹ω›)
notation *csucc* (‹_$^+$› [*90*])
no_notation *Aleph* (‹$\aleph$_› [*90*] *90*)
notation *Aleph* (‹$\aleph$_›)
syntax *_ge* :: [*i*,*i*] $\Rightarrow$ *o* (**infixl** ‹$\geq$› *50*)
translations $x \geq y \rightharpoonup y \leq x$

2.1 Some minimal arithmetic/ordinal stuff

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

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

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

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

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

lemma *le_neq_imp_lt*:
$x{\leq}y \Longrightarrow x{\neq}y \Longrightarrow x{<}y$

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**
$str\_bound(A) \equiv \bigcup a \in A.\ succ(a)$

lemma *str_bound_type* [*TC*]: $\forall a \in A.\ Ord(a) \Longrightarrow Ord(str\_bound(A))$
unfolding *str_bound_def* **by** *auto*

lemma *str_bound_lt*: $\forall a \in A.\ Ord(a) \Longrightarrow \forall a \in A.\ a < str\_bound(A)$
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
$\bigwedge \alpha\ \beta.\ \alpha \in \gamma \Longrightarrow \beta \in \gamma \Longrightarrow \alpha < \beta \Longrightarrow Q(\alpha,\beta)$
$\bigwedge \alpha\ \beta.\ Q(\alpha,\beta) \Longrightarrow Q(\beta,\alpha)$
$\alpha \in \gamma$ $\beta \in \gamma$ $\alpha \neq \beta$
$Ord(\gamma)$
shows
$Q(\alpha,\beta)$
proof -
from *assms*
consider $\alpha<\beta \mid \beta<\alpha$
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*: $|A| = succ(n) \Longrightarrow A \neq 0$
by *auto*

lemma *Ord_eq_Collect_lt*: $i<\alpha \Longrightarrow \{j\in\alpha.\ j<i\} = i$
— 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*] ⇒ *i* **where**
Finite_to_one$(X,Y) \equiv \{f{:}X\to Y.\ \forall y\in Y.\ Finite(\{x\in X\ .\ f`x = y\})\}$

lemma *Finite_to_oneI*[*intro*]:
assumes $f{:}X\to Y\ \bigwedge y.\ y\in Y \Longrightarrow Finite(\{x\in X\ .\ f`x = y\})$
shows $f \in$ *Finite_to_one*(X,Y)
using *assms* **unfolding** *Finite_to_one_def* **by** *simp*

lemma *Finite_to_oneD*[*dest*]:
$f \in$ *Finite_to_one*$(X,Y) \Longrightarrow f{:}X\to Y$
$f \in$ *Finite_to_one*$(X,Y) \Longrightarrow y\in Y \Longrightarrow Finite(\{x\in X\ .\ f`x = y\})$
unfolding *Finite_to_one_def* **by** *simp_all*

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

lemma *Diff_bij*:
assumes $\forall A\in F.\ X \subseteq A$ **shows** $(\lambda A\in F.\ A\text{-}X) \in bij(F,\ \{A\text{-}X.\ A\in F\})$
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 $(\lambda x\in A.\ b) : A \to B$
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 $h{:}B \to C$
shows $range(h) \subseteq C$
unfolding *range_def domain_def converse_def* **using** *range_type*[*OF _ assms*]
by *auto*

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

```
  shows ∃ a∈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 ∈ Pi(A,B) ⟹ range(f) = {f ` x . x ∈ A}
  using Pi_rangeD[of f A B] apply_rangeI[of f A B]
  by blast

lemma Pi_vimage_subset : f ∈ Pi(A,B) ⟹ f-``C ⊆ A
  unfolding Pi_def by auto

lemma apply_in_codomain_Ord:
  assumes
    Ord(γ) γ≠0 f: A → γ
  shows
    f`x∈γ
proof (cases x∈A)
  case True
  from assms ‹x∈A›
  show ?thesis
    using  domain_of_fun apply_rangeI by simp
next
  case False
  from assms ‹x∉A›
  show ?thesis
    using apply_0 Ord_0_lt ltD domain_of_fun by auto
qed

lemma range_eq_image:
  assumes f:A→B
  shows range(f) = f``A
proof
  show f `` A ⊆ range(f)
    unfolding image_def by blast
  {
    fix x
    assume x∈range(f)
    with assms
    have x∈f``A
      using domain_of_fun[of f A λ_. B] by auto
  }
  then
  show range(f) ⊆ f `` A ..
qed

lemma Image_sub_codomain: f:A→B ⟹ f``C ⊆ B
  using image_subset fun_is_rel[of _ _ λ_ . B] by force

lemma inj_to_Image:
```

assumes
f:*A*→*B* *f* ∈ *inj*(*A*,*B*)
shows
f ∈ *inj*(*A*,*f*``*A*)
using *assms inj_inj_range range_eq_image* **by** *force*

lemma *inj_imp_surj*:
fixes *f b*
notes *inj_is_fun*[*dest*]
defines [*simp*]: *ifx*(*x*) ≡ *if x*∈*range*(*f*) *then converse*(*f*)`*x else b*
assumes *f* ∈ *inj*(*B*,*A*) *b*∈*B*
shows (λ*x*∈*A*. *ifx*(*x*)) ∈ *surj*(*A*,*B*)
proof -
from *assms*
have *converse*(*f*) ∈ *surj*(*range*(*f*),*B*) *range*(*f*) ⊆ *A*
converse(*f*) : *range*(*f*) → *B*
using *inj_converse_surj range_fun_subset_codomain surj_is_fun* **by** *blast*+
with ‹*b*∈*B*›
show (λ*x*∈*A*. *ifx*(*x*)) ∈ *surj*(*A*,*B*)
unfolding *surj_def*
proof (*intro CollectI lam_type ballI*; *elim CollectE*)
fix *y*
assume *y* ∈ *B* ∀ *y*∈*B*. ∃ *x*∈*range*(*f*). *converse*(*f*) ` *x* = *y*
with ‹*range*(*f*) ⊆ *A*›
show ∃ *x*∈*A*. (λ*x*∈*A*. *ifx*(*x*)) ` *x* = *y*
by (*drule_tac bspec*, *auto*)
qed *simp*
qed

lemma *fun_Pi_disjoint_Un*:
assumes *f* ∈ *Pi*(*A*,*B*) *g* ∈ *Pi*(*C*,*D*) *A* ∩ *C* = *0*
shows *f* ∪ *g* ∈ *Pi*(*A* ∪ *C*, λ*x*. *B*(*x*) ∪ *D*(*x*))
using *assms*
by (*simp add*: *Pi_iff extension Un_rls*) (*unfold function_def*, *blast*)

lemma *Un_restrict_decomposition*:
assumes *f* ∈ *Pi*(*A*,*B*)
shows *f* = *restrict*(*f*, *A* ∩ *C*) ∪ *restrict*(*f*, *A* - *C*)
using *assms*
proof (*rule fun_extension*)
from *assms*
have *restrict*(*f*,*A*∩*C*) ∪ *restrict*(*f*,*A*-*C*) ∈ *Pi*(*A*∩*C* ∪ (*A*-*C*), λ*x*. *B*(*x*)∪*B*(*x*))
using *restrict_type2*[*of f A B*]
by (*rule_tac fun_Pi_disjoint_Un*) *force*+
moreover
have (*A* ∩ *C*) ∪ (*A* - *C*) = *A* **by** *auto*
ultimately
show *restrict*(*f*, *A* ∩ *C*) ∪ *restrict*(*f*, *A* - *C*) ∈ *Pi*(*A*, *B*) **by** *simp*
next

```
  fix x
  assume x ∈ A
  with assms
  show f ` x = (restrict(f, A ∩ C) ∪ restrict(f, A - C)) ` x
    using restrict fun_disjoint_apply1[of _ restrict(f,_)]
      fun_disjoint_apply2[of _ restrict(f,_)]
      domain_restrict[of f] apply_0 domain_of_fun
    by (cases x∈C) simp_all
qed

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

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

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

2.3 Finite sets

```
lemma Replace_sing1:
```

⟦ (∃ *a*. *P*(*d*,*a*)) ∧ (∀ *y* *y'*. *P*(*d*,*y*) ⟶ *P*(*d*,*y'*) ⟶ *y*=*y'*) ⟧ ⟹ ∃ *a*. {*y* . *x* ∈ {*d*}, *P*(*x*,*y*)} = {*a*}
by *blast*

— Not really necessary
lemma *Replace_sing2*:
assumes ∀ *a*. ¬ *P*(*d*,*a*)
shows {*y* . *x* ∈ {*d*}, *P*(*x*,*y*)} = *0*
using *assms* **by** *auto*

lemma *Replace_sing3*:
assumes ∃ *c* *e*. *c* ≠ *e* ∧ *P*(*d*,*c*) ∧ *P*(*d*,*e*)
shows {*y* . *x* ∈ {*d*}, *P*(*x*,*y*)} = *0*
proof -
{
fix *z*
{
assume ∀ *y*. *P*(*d*, *y*) ⟶ *y* = *z*
with *assms*
have *False* **by** *auto*
}
then
have *z* ∉ {*y* . *x* ∈ {*d*}, *P*(*x*,*y*)}
using *Replace_iff* **by** *auto*
}
then
show *?thesis*
by (*intro equalityI subsetI*) *simp_all*
qed

lemma *Replace_Un*: {*b* . *a* ∈ *A* ∪ *B*, *Q*(*a*, *b*)} =
{*b* . *a* ∈ *A*, *Q*(*a*, *b*)} ∪ {*b* . *a* ∈ *B*, *Q*(*a*, *b*)}
by (*intro equalityI subsetI*) (*auto simp add*:*Replace_iff*)

lemma *Replace_subset_sing*: ∃ *z*. {*y* . *x* ∈ {*d*}, *P*(*x*,*y*)} ⊆ {*z*}
proof -
consider
(*1*) (∃ *a*. *P*(*d*,*a*)) ∧ (∀ *y* *y'*. *P*(*d*,*y*) ⟶ *P*(*d*,*y'*) ⟶ *y*=*y'*) |
(*2*) ∀ *a*. ¬ *P*(*d*,*a*) | (*3*) ∃ *c* *e*. *c* ≠ *e* ∧ *P*(*d*,*c*) ∧ *P*(*d*,*e*) **by** *auto*
then
show ∃ *z*. {*y* . *x* ∈ {*d*}, *P*(*x*,*y*)} ⊆ {*z*}
proof (*cases*)
case *1*
then show *?thesis* **using** *Replace_sing1*[*of P d*] **by** *auto*
next
case *2*
then show *?thesis* **by** *auto*
next
case *3*

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

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

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

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

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

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

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

2.4 Basic results on equipollence, cardinality and related concepts

lemma *lepollD*[*dest*]: $A \lesssim B \Longrightarrow \exists f. f \in inj(A, B)$
unfolding *lepoll_def* **.**

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

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

declare *bij_imp_eqpoll*[*intro*]

lemma *range_of_subset_eqpoll*:
assumes $f \in inj(X,Y)$ $S \subseteq X$
shows $S \approx f$ `` S
using *assms restrict_bij* **by** *blast*

I thank Miguel Pagano for this proof.

lemma *function_space_eqpoll_cong*:
assumes
$A \approx A'$ $B \approx B'$
shows
$A \to B \approx A' \to B'$
proof -
from *assms*(*1*)[*THEN eqpoll_sym*] *assms*(*2*)
obtain *f g* **where** $f \in bij(A',A)$ $g \in bij(B,B')$
by *blast*
then
have *converse*(g) : $B' \to B$ *converse*(f): $A \to A'$
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: $A \to B$
with ‹$f \in bij(A',A)$› ‹$g \in bij(B,B')$›
show g *O* F *O* f : $A' \to B'$
using *bij_is_fun comp_fun* **by** *blast*
next
fix F
assume F: $A' \to B'$
with ‹*converse*(g) : $B' \to B$› ‹*converse*(f): $A \to A'$›
show *converse*(g) *O* F *O* *converse*(f) : $A \to B$
using *comp_fun* **by** *blast*
next
from ‹f∈_› ‹g∈_› ‹*converse*(f)∈_› ‹*converse*(g)∈_›
have ($\bigwedge x.\ x \in A' \to B' \Longrightarrow$ *converse*(g) *O* x *O* *converse*(f) $\in A \to B$)
using *bij_is_fun comp_fun* **by** *blast*

then
have $(\lambda x{\in}A \to B.\ g\ O\ x\ O\ f)\ O\ (\lambda x{\in}A' \to B'.\ converse(g)\ O\ x\ O\ converse(f))$
$= (\lambda x{\in}A' \to B'.\ (g\ O\ converse(g))\ O\ x\ O\ (converse(f)\ O\ f))$
using *lam_cong comp_assoc comp_lam*[*of* $A' \to B'$] **by** *auto*
also
have ... $= (\lambda x{\in}A' \to B'.\ id(B')\ O\ x\ O\ (id(A')))$
using *left_comp_inverse*[*OF bij_is_inj*[*OF* ‹*f*∈_›]]
right_comp_inverse[*OF bij_is_surj*[*OF* ‹*g*∈_›]]
by *auto*
also
have ... $= (\lambda x{\in}A' \to B'.\ x)$
using *left_comp_id*[*OF fun_is_rel*] *right_comp_id*[*OF fun_is_rel*] *lam_cong* **by** *auto*
also
have ... $= id(A'{\to}B')$ **unfolding** *id_def* **by** *simp*
finally
show $(\lambda x{\in}A$ -> $B.\ g\ O\ x\ O\ f)\ O\ (\lambda x{\in}A'$ -> $B'.\ converse(g)\ O\ x\ O\ converse(f)) = id(A'$ -> $B')$ **.**
next
from ‹*f*∈_› ‹*g*∈_›
have $(\bigwedge x.\ x \in A \to B \Longrightarrow g\ O\ x\ O\ f \in A' \to B')$
using *bij_is_fun comp_fun* **by** *blast*
then
have $(\lambda x{\in}A'$ -> $B'.\ converse(g)\ O\ x\ O\ converse(f))\ O\ (\lambda x{\in}A$ -> $B.\ g\ O\ x\ O\ f)$
$= (\lambda x{\in}A \to B.\ (converse(g)\ O\ g)\ O\ x\ O\ (f\ O\ converse(f)))$
using *comp_lam comp_assoc* **by** *auto*
also
have ... $= (\lambda x{\in}A \to B.\ id(B)\ O\ x\ O\ (id(A)))$
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 ... $= (\lambda x{\in}A \to B.\ x)$
using *left_comp_id*[*OF fun_is_rel*] *right_comp_id*[*OF fun_is_rel*] *lam_cong* **by** *auto*
also
have ... $= id(A{\to}B)$ **unfolding** *id_def* **by** *simp*
finally
show $(\lambda x{\in}A' \to B'.\ converse(g)\ O\ x\ O\ converse(f))\ O\ (\lambda x{\in}A$ -> $B.\ g\ O\ x\ O\ f) = id(A$ -> $B)$ **.**
qed
qed

lemma *curry_eqpoll*:
fixes $d\ \nu 1\ \nu 2\ \kappa$
shows $\nu 1 \to \nu 2 \to \kappa \approx \nu 1 \times \nu 2 \to \kappa$
unfolding *eqpoll_def*

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

lemma *Pow_eqpoll_function_space*:
 fixes *d X*
 notes *bool_of_o_def* [*simp*]
 defines [*simp*]:$d(A) \equiv (\lambda x\in X.\ bool\_of\_o(x\in A))$
  — the witnessing map for the thesis:
 shows $Pow(X) \approx X \to 2$
 unfolding *eqpoll_def*
proof (*intro exI*, *rule lam_bijective*)
 — We give explicit mutual inverses
 fix *A*
 assume $A\in Pow(X)$
 then
 show $d(A) : X \to 2$
  using *lam_type*[*of* _ $\lambda x.\ bool\_of\_o(x\in A)$ λ_. *2*]
  by *force*
 from ‹$A\in Pow(X)$›
 show $\{y\in X.\ d(A)`y = 1\} = A$
  by (*auto*)
next
 fix *f*
 assume f: $X\to 2$
 then
 show $d(\{y \in X\ .\ f ` y = 1\}) = f$
  using *apply_type*[*OF* ‹f: $X\to 2$›]
  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 :: $[i,i] \Rightarrow i$ ($\_^{\uparrow}\_$ [76,1] 75) **where**
  $\kappa^{\uparrow\nu} \equiv |\nu \to \kappa|$

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* ‹$Ord(\kappa)$›
  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 ‹$Ord(\kappa)$›
  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 ‹$d \lesssim \kappa$› ‹$Ord(\kappa)$›

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

abbreviation
Infinite :: *i*⇒*o* **where**
Infinite(*X*) ≡ ¬ *Finite*(*X*)

lemma *Infinite_not_empty*: *Infinite*(*X*) ⟹ *X* ≠ *0*
using *empty_lepollI* **by** *auto*

lemma *Infinite_imp_nats_lepoll*:
assumes *Infinite*(*X*) $n \in \omega$
shows $n \lesssim X$
using ‹$n \in \omega$›
proof (*induct*)
case *0*
then
show *?case* **using** *empty_lepollI* **by** *simp*
next
case (*succ x*)
show *?case*
proof -
from ‹*Infinite*(*X*)› **and** ‹$x \in \omega$›
have ¬ ($x \approx X$)
using *eqpoll_sym* **unfolding** *Finite_def* **by** *auto*
with ‹$x \lesssim X$›
obtain *f* **where** *f* ∈ *inj*(*x*,*X*) *f* ∉ *surj*(*x*,*X*)
unfolding *bij_def eqpoll_def* **by** *auto*
moreover from *this*
obtain *b* **where** *b* ∈ *X* ∀ *a*∈*x*. *f*‘*a* ≠ *b*
using *inj_is_fun* **unfolding** *surj_def* **by** *auto*
ultimately
have *f* ∈ *inj*(*x*,*X*-{*b*})
unfolding *inj_def* **by** (*auto intro*:*Pi_type*)
then
have *cons*(⟨*x*, *b*⟩, *f*) ∈ *inj*(*succ*(*x*), *cons*(*b*, *X* - {*b*}))
using *inj_extend*[*of f x X*-{*b*} *x b*] **unfolding** *succ_def*
by (*auto dest*:*mem_irrefl*)
moreover from ‹*b*∈*X*›
have *cons*(*b*, *X* - {*b*}) = *X* **by** *auto*
ultimately

```
    show succ(x) ≲ X by auto
  qed
qed

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

lemma lepoll_nat_imp_Infinite: ω ≲ X ⟹ Infinite(X)
proof (rule ccontr, simp)
  assume ω ≲ X Finite(X)
  moreover from this
  obtain n where X ≈ n n ∈ ω
    unfolding Finite_def by auto
  moreover from calculation
  have ω ≲ n
    using lepoll_eq_trans by simp
  ultimately
  show False
    using lepoll_nat_imp_Finite nat_not_Finite by simp
qed

lemma InfCard_imp_Infinite: InfCard(κ) ⟹ Infinite(κ)
  using le_imp_lepoll[THEN lepoll_nat_imp_Infinite, of κ]
  unfolding InfCard_def by simp

lemma lt_surj_empty_imp_Card:
  assumes Ord(κ) ⋀α. α < κ ⟹ surj(α,κ) = 0
  shows Card(κ)
proof -
  {
    assume |κ| < κ
    with assms
    have False
      using LeastI[of λi. i ≈ κ κ, OF eqpoll_refl]
        Least_le[of λi. i ≈ κ |κ|, OF Ord_cardinal_eqpoll]
      unfolding Card_def cardinal_def eqpoll_def bij_def
      by simp
  }
  with assms
  show ?thesis
    using Ord_cardinal_le[of κ] not_lt_imp_le[of |κ| κ] 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
    $f \in mono\_map(A,r,B,s)$ $B \subseteq C$
  shows
    $f \in mono\_map(A,r,C,s)$
  unfolding *mono_map_def*
proof (*intro CollectI ballI impI*)
  from ‹$f \in mono\_map(A,\_,B,\_)$›
  have f: $A \to B$
    using *mono_map_is_fun* **by** *simp*
  with ‹$B \subseteq C$›
  show f: $A \to C$
    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** ‹f: $A \to B$›
  have $f`x \in B$ $f`y \in B$
    using *apply_type* **by** *simp_all*
  moreover
  note ‹$f \in mono\_map(\_,r,\_,s)$›
  ultimately
  show $\langle f ` x, f ` y \rangle \in s$
    unfolding *mono_map_def* **by** *blast*
qed

lemma *ordertype_zero_imp_zero*: $ordertype(A,r) = 0 \Longrightarrow A = 0$
  using *ordermap_type*[*of A r*]
  by (*cases* $A=0$) *auto*

lemma *mono_map_increasing*:
  $j \in mono\_map(A,r,B,s) \Longrightarrow a \in A \Longrightarrow c \in A \Longrightarrow \langle a,c \rangle \in r \Longrightarrow \langle j`a,j`c \rangle \in s$
  unfolding *mono_map_def* **by** *simp*

lemma *linear_mono_map_reflects*:
  assumes
    $linear(\alpha,r)$ $trans[\beta](s)$ $irrefl(\beta,s)$ $f \in mono\_map(\alpha,r,\beta,s)$
    $x \in \alpha$ $y \in \alpha$ $\langle f`x,f`y \rangle \in s$
  shows
    $\langle x,y \rangle \in r$
proof -
  from ‹$f \in mono\_map(\_,\_,\_,\_)$›
  have *preserves*:$x \in \alpha \Longrightarrow y \in \alpha \Longrightarrow \langle x,y \rangle \in r \Longrightarrow \langle f`x,f`y \rangle \in s$ **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** ‹$linear(\alpha,r)$›
    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 ‹$f\in mono\_map(\_,\_,\beta,\_)$› ‹$x\in\alpha$› ‹$y\in\alpha$›
have $f`x\in\beta$ $f`y\in\beta$
using *apply_type*[*OF mono_map_is_fun*] **by** *simp_all*
moreover
note ‹$\langle f`x,f`y\rangle\in s$› ‹*trans*[$\beta$]($s$)› ‹*irrefl*($\beta$,$s$)›
ultimately
have *False*
using *trans_onD*[*of* β *s f`x f`y f`x*] *irreflE* **by** *blast*
}
with *assms*
show $\langle x,y\rangle\in r$ **by** *blast*
qed

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

lemmas *Memrel_mono_map_reflects* = *linear_mono_map_reflects*
[*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'*:
⟦ *linear*(A,r); *irrefl*(B,s); $f \in$ *mono_map*(A,r,B,s) ⟧ $\Longrightarrow f \in$ *inj*(A,B)
unfolding *irrefl_def mono_map_def inj_def* **using** *linearE*
by (*clarify*, *rename_tac x w*)
(*erule_tac x=w* **and** *y=x* **in** *linearE*, *assumption+*, (*force intro*: *apply_type*)+)

lemma *mono_map_imp_ord_iso_image*:
assumes
linear(α,r) *trans*[β](s) *irrefl*(β,s) $f\in$*mono_map*(α,r,β,s)
shows
$f \in$ *ord_iso*(α,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 $f \in$ *inj*(α,β)
using *mono_map_is_inj'* **by** *blast*
moreover from ‹$f \in$ *mono_map*(_,_,_,_)›
have $f \in$ *surj*(α, $f``\alpha$)
unfolding *mono_map_def* **using** *surj_image* **by** *auto*
ultimately
show $f \in$ *bij*(α, $f``\alpha$)
unfolding *bij_def* **using** *inj_is_fun inj_to_Image* **by** *simp*

from ‹*f*∈*mono_map*(_,_,_,_)›
show *x*∈α ⟹ *y*∈α ⟹ ⟨*x*,*y*⟩∈*r* ⟹ ⟨*f*‘*x*,*f*‘*y*⟩∈*s* **for** *x y*
unfolding *mono_map_def* **by** *blast*
with *assms*
show ⟨*f*‘*x*,*f*‘*y*⟩∈*s* ⟹ *x*∈α ⟹ *y*∈α ⟹ ⟨*x*,*y*⟩∈*r* **for** *x y*
using *linear_mono_map_reflects*
by *blast*
qed

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

abbreviation
mono_map_Memrel :: [*i*,*i*] ⇒ *i* (**infixr** ‹→<› *60*) **where**
α →< β ≡ *mono_map*(α,*Memrel*(α),β,*Memrel*(β))

lemma *mono_map_imp_ord_iso_Memrel*:
assumes
Ord(α) *Ord*(β) *f*:α →< β
shows
f ∈ *ord_iso*(α,*Memrel*(α),*f*‘‘α,*Memrel*(β))
using *assms mono_map_imp_ord_iso_image*[*OF well_ord_is_linear*[*OF well_ord_Memrel*]
well_ord_is_trans_on[*OF well_ord_Memrel*] *irrefl_Memrel*] **by** *blast*

lemma *mono_map_ordertype_image′*:
assumes
X⊆α *Ord*(α) *Ord*(β) *f* ∈ *mono_map*(*X*,*Memrel*(α),β,*Memrel*(β))
shows
ordertype(*f*‘‘*X*,*Memrel*(β)) = *ordertype*(*X*,*Memrel*(α))
using *assms mono_map_is_fun*[*of f X* _ β] *ordertype_eq*
mono_map_imp_ord_iso_image[*OF well_ord_is_linear*[*OF well_ord_Memrel*, *THEN linear_subset*]
well_ord_is_trans_on[*OF well_ord_Memrel*] *irrefl_Memrel*, *of* α *X* β *f*]
well_ord_subset[*OF well_ord_Memrel*] *Image_sub_codomain*[*of f X* β *X*] **by** *auto*

lemma *mono_map_ordertype_image*:
assumes
Ord(α) *Ord*(β) *f*:α →< β
shows
ordertype(*f*‘‘α,*Memrel*(β)) = α
using *assms mono_map_is_fun ordertype_Memrel ordertype_eq*[*of f* α *Memrel*(α)]
mono_map_imp_ord_iso_Memrel well_ord_subset[*OF well_ord_Memrel*] *Image_sub_codomain*[*of* _ α]
by *auto*

lemma *apply_in_image*: *f*:*A*→*B* ⟹ *a*∈*A* ⟹ *f*‘*a* ∈ *f*‘‘*A*
using *range_eq_image apply_rangeI*[*of f*] **by** *simp*

lemma *Image_subset_Ord_imp_lt*:
  assumes
    Ord(α) *h*``*A* ⊆ α *x*∈*domain*(*h*) *x*∈*A* *function*(*h*)
  shows
    h`*x* < α
  using *assms*
  unfolding *domain_def* **using** *imageI ltI function_apply_equality* **by** *auto*

lemma *ordermap_le_arg*:
  assumes
    X⊆β *x*∈*X* *Ord*(β)
  shows
    x∈*X* ⟹ *ordermap*(*X*,*Memrel*(β))`*x*≤*x*
proof (*induct rule*:*Ord_induct*[*OF subsetD*, *OF assms*])
  case (*1 x*)
  have *wf*[*X*](*Memrel*(β))
    using *wf_imp_wf_on*[*OF wf_Memrel*] **.**
  with *1*
  have *ordermap*(*X*,*Memrel*(β))`*x* = {*ordermap*(*X*,*Memrel*(β))`*y* . *y*∈{*y*∈*X* . *y*∈*x*
∧ *y*∈β}}
    using *ordermap_unfold Ord_trans*[*of _ x* β] **by** *auto*
  also from *assms*
  have ... = {*ordermap*(*X*,*Memrel*(β))`*y* . *y*∈{*y*∈*X* . *y*∈*x*}}
    using *Ord_trans*[*of _ x* β] *Ord_in_Ord* **by** *blast*
  finally
  have *ordm*:*ordermap*(*X*,*Memrel*(β))`*x* = {*ordermap*(*X*,*Memrel*(β))`*y* . *y*∈{*y*∈*X*
. *y*∈*x*}} **.**
  from *1*
  have *y*∈*x* ⟹ *y*∈*X* ⟹ *ordermap*(*X*,*Memrel*(β))`*y* ≤ *y* **for** *y* **by** *simp*
  with ‹*x*∈β› **and** ‹*Ord*(β)›
  have *y*∈*x* ⟹ *y*∈*X* ⟹ *ordermap*(*X*,*Memrel*(β))`*y* ∈ *x* **for** *y*
    using *ltI*[*OF _ Ord_in_Ord*[*of* β *x*]] *lt_trans1 ltD* **by** *blast*
  with *ordm*
  have *ordermap*(*X*,*Memrel*(β))`*x* ⊆ *x* **by** *auto*
  with ‹*x*∈*X*› *assms*
  show *?case*
    using *subset_imp_le Ord_in_Ord*[*of* β *x*] *Ord_ordermap*
      well_ord_subset[*OF well_ord_Memrel*, *of* β] **by** *force*
qed

lemma *subset_imp_ordertype_le*:
  assumes
    X⊆β *Ord*(β)
  shows
    ordertype(*X*,*Memrel*(β))≤β
proof -
  {
    fix *x*
    assume *x*∈*X*

with *assms*
have *ordermap*(X,*Memrel*(β))‘$x \le x$
using *ordermap_le_arg* **by** *simp*
with ‹$x \in X$› **and** *assms*
have *ordermap*(X,*Memrel*(β))‘$x \in \beta$ (**is** ?$y \in$ _)
using *ltD*[*of* ?*y* *succ*(x)] *Ord_trans*[*of* ?*y* x β] **by** *auto*
}
then
have *ordertype*(X, *Memrel*(β)) $\subseteq \beta$
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
$f \in$*mono_map*(α, *Memrel*(α),β, *Memrel*(β)) *Ord*(α) *Ord*(β)
shows
$\alpha \le \beta$
proof -
from *assms*
have $f \in \langle\alpha$, *Memrel*(α)$\rangle \cong \langle f``\alpha$, *Memrel*($\beta$)$\rangle$
using *mono_map_imp_ord_iso_Memrel* **by** *simp*
then
have *converse*(f) $\in \langle f``\alpha$, *Memrel*(β)$\rangle \cong \langle\alpha$, *Memrel*($\alpha$)$\rangle$
using *ord_iso_sym* **by** *simp*
with ‹*Ord*(α)›
have α = *ordertype*($f``\alpha$,*Memrel*(β))
using *ordertype_eq well_ord_Memrel ordertype_Memrel* **by** *auto*
also from *assms*
have *ordertype*($f``\alpha$,*Memrel*(β)) $\le \beta$
using *subset_imp_ordertype_le mono_map_is_fun*[*of f*] *Image_sub_codomain*[*of f*] **by** *force*
finally
show ?*thesis* **.**
qed

— ⟦*Ord*(A); $f \in$ *mono_map*(A, *Memrel*(A), B, *Memrel*(Aa))⟧ $\Longrightarrow f \in$ *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: $A \to B$ $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow \langle x,y\rangle \in r \Longrightarrow \langle f`x,f`y\rangle \in s$
shows $f \in$ *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 $n \le \kappa$
  using *assms Ord_is_Transset*
    le_trans[*of* $n\ \omega\ \kappa$, *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* $\omega\ \kappa$]
  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*($\aleph_\alpha$)
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 ‹*Ord*(α)›
    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 ‹*Ord*(α)›
  have $\neg (|\aleph_\alpha| \in \omega)$
    using *Card_cardinal_eq* **by** *auto*
  then
  have $\neg$ *Finite*($\aleph_\alpha$) **by** *auto*
  with ‹*Ord*(α)›
  show *?thesis*
    using *Inf_Card_is_InfCard* **by** *simp*
qed

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

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

lemma *countableI*[*intro*]: $X \lesssim \omega \Longrightarrow countable(X)$
  unfolding *countable_def* **by** *simp*

lemma *countableD*[*dest*]: $countable(X) \Longrightarrow X \lesssim \omega$
  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*]:
  $delta\_system(D) \Longrightarrow \exists r.\ \forall A{\in}D.\ \forall B{\in}D.\ A \neq B \longrightarrow A \cap B = r$
  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,λ*a g*. *f*‘(*g*‘‘*a*))

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

The next result explains its use.

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

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

end

3 Cofinality

theory *Cofinality*
  imports
    ZF_Library
begin

3.1 Basic results and definitions

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

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

A function is cofinal if it range is.

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

lemma *cofinal_funI*:
  assumes $\bigwedge a.\ a{\in}A \Longrightarrow \exists x{\in}domain(f).\ \langle a,f`x\rangle{\in}r \vee a = f`x$
  shows $cofinal\_fun(f,A,r)$
  using *assms* **unfolding** *cofinal_fun_def* **by** *simp*

lemma *cofinal_funD*:
  assumes $cofinal\_fun(f,A,r)$ $a{\in}A$
  shows $\exists x{\in}domain(f).\ \langle a,f`x\rangle{\in}r \vee a = f`x$
  using *assms* **unfolding** *cofinal_fun_def* **by** *simp*

lemma *cofinal_in_cofinal*:
  assumes
    $trans(r)$ $cofinal(Y,X,r)$ $cofinal(X,A,r)$
  shows
    $cofinal(Y,A,r)$
  unfolding *cofinal_def*
proof
  fix a
  assume $a{\in}A$
  moreover from ‹$cofinal(X,A,r)$›
  have $b{\in}A{\Longrightarrow}\exists x{\in}X.\ \langle b,x\rangle{\in}r \vee b{=}x$ **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 ‹$cofinal(Y,X,r)$›
  have $c{\in}X{\Longrightarrow}\exists y{\in}Y.\ \langle c,y\rangle{\in}r \vee c{=}y$ **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 ‹$a{\in}A$› ‹$y{\in}X$› ‹$\langle a,y\rangle{\in}r \vee a{=}y$› ‹$trans(r)$›
  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 $cofinal\_fun(f,A,r)$ $f{:}C \to D$
  shows $cofinal(D,A,r)$
  unfolding *cofinal_def*
proof
  fix b
  assume $b \in A$

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

lemma *cofinal_range_iff_cofinal_fun*:
assumes *function*(f)
shows *cofinal*(*range*(f),A,r) $\longleftrightarrow$ *cofinal_fun*(f,A,r)
unfolding *cofinal_fun_def*
proof (*intro iffI ballI*)
fix *a*
assume $a{\in}A$ ‹*cofinal*(*range*(f),A,r)›
then
obtain *y* **where** $y{\in}range(f)$ $\langle a,y\rangle \in r \vee a = y$
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 ‹*function*(f)›
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 $\exists x{\in}domain(f).\ \langle a,\ f\ `\ x\rangle \in r \vee a = f\ `\ x$ **by** *blast*
next
assume $\forall a{\in}A.\ \exists x{\in}domain(f).\ \langle a,\ f\ `\ x\rangle \in r \vee a = f\ `\ x$
with *assms*
show *cofinal*(*range*(f), A, r)
using *function_apply_Pair*[*of f*] **unfolding** *cofinal_def* **by** *fast*
qed

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

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

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

lemma *cf_funI*[*intro*!]: $cofinal\_fun(f,\alpha,Memrel(\alpha)) \Longrightarrow cf\_fun(f,\alpha)$
  unfolding *cf_fun_def* **by** *simp*

lemma *cf_funD*[*dest*!]: $cf\_fun(f,\alpha) \Longrightarrow cofinal\_fun(f,\alpha,Memrel(\alpha))$
  unfolding *cf_fun_def* **by** *simp*

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

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

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

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

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

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

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

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

lemma ordertype_0_not_cofinal_succ:
```

assumes *ordertype*(*A*,*Memrel*(*succc*(*i*))) = *0* *A*⊆*succ*(*i*) *Ord*(*i*)
shows ¬*cofinal*(*A*,*succ*(*i*),*Memrel*(*succ*(*i*)))
proof
have *1*:*ordertype*(*A*,*Memrel*(*succ*(*i*))) = *ordertype*(*0*,*Memrel*(*0*))
using ‹*ordertype*(*A*,*Memrel*(*succ*(*i*))) = *0*› *ordertype_0* **by** *simp*
from ‹*A*⊆*succ*(*i*)› ‹*Ord*(*i*)›
have ∃*f*. *f* ∈ ⟨*A*, *Memrel*(*succ*(*i*))⟩ ≅ ⟨*0*, *Memrel*(*0*)⟩
using *well_ord_Memrel well_ord_subset*
ordertype_eq_imp_ord_iso[*OF 1*] *Ord_0* **by** *blast*
then
have *A*=*0*
using *ord_iso_is_bij bij_imp_eqpoll eqpoll_0_is_0* **by** *blast*
moreover
assume *cofinal*(*A*, *succ*(*i*), *Memrel*(*succ*(*i*)))
moreover
note ‹*Ord*(*i*)›
ultimately
show *False*
using *not_mem_empty* **unfolding** *cofinal_def* **by** *auto*
qed

I thank Edwin Pacheco Rodríguez for the following lemma.

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

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

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

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

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

— TODO: define Succ (predicate for successor ordinals)
lemma *cf_eq_one_iff*:
assumes *Ord*(γ)
shows *cf*(γ) = *1* ⟷ (∃α. *Ord*(α) ∧ γ = *succ*(α))
proof (*intro iffI*)
assume ∃α. *Ord*(α) ∧ γ = *succ*(α)
then
show *cf*(γ) = *1* **using** *cf_succ* **by** *auto*

next
  assume *cf*(γ) = *1*
  moreover from *assms*
  obtain *A* **where** *A* ⊆ γ *cf*(γ) = *ordertype*(*A*,*Memrel*(γ))
    cofinal(*A*,γ,*Memrel*(γ))
    using *cf_is_ordertype* **by** *blast*
  ultimately
  have *ordertype*(*A*,*Memrel*(γ)) = *1* **by** *simp*
  moreover
  define *f* **where** *f*≡*converse*(*ordermap*(*A*,*Memrel*(γ)))
  moreover from *this* ‹*ordertype*(*A*,*Memrel*(γ)) = *1*› ‹*A* ⊆ γ› *assms*
  have *f* ∈ *surj*(*1*,*A*)
    using *well_ord_subset*[*OF well_ord_Memrel*, *THEN ordermap_bij*,
        THEN bij_converse_bij, *of* γ *A*] *bij_is_surj*
    by *simp*
  with ‹*cofinal*(*A*,γ,*Memrel*(γ))›
  have ∀ *a*∈γ. ⟨*a*, *f‘0*⟩ ∈ *Memrel*(γ) ∨ *a* = *f‘0*
    unfolding *cofinal_def surj_def*
    by *auto*
  with *assms* ‹*A* ⊆ γ› ‹*f* ∈ *surj*(*1*,*A*)›
  show ∃ α. *Ord*(α) ∧ γ = *succ*(α)
    using *Ord_has_max_imp_succ*[*of* γ *f‘0*]
      surj_is_fun[*of f 1 A*] *apply_type*[*of f 1* λ_.*A 0*]
    unfolding *lt_def*
    by (*auto intro*:*Ord_in_Ord*)
qed

lemma *ordertype_in_cf_imp_not_cofinal*:
  assumes
    ordertype(*A*,*Memrel*(γ)) ∈ *cf*(γ)
    A ⊆ γ
  shows
    ¬ *cofinal*(*A*,γ,*Memrel*(γ))
proof
  note ‹*A* ⊆ γ›
  moreover
  assume *cofinal*(*A*,γ,*Memrel*(γ))
  ultimately
  have ∃ *B*. *B* ⊆ γ ∧ *cofinal*(*B*, γ, *Memrel*(γ)) ∧ *ordertype*(*A*,*Memrel*(γ)) = *ordertype*(*B*, *Memrel*(γ))
    (**is** *?P*(*ordertype*(*A*,_)))
    by *blast*
  moreover from *assms*
  have *ordertype*(*A*,*Memrel*(γ)) < *cf*(γ)
    using *Ord_cf ltI* **by** *blast*
  ultimately
  show *False*
    unfolding *cf_def* **using** *less_LeastE*[*of ?P ordertype*(*A*,*Memrel*(γ))]
    by *auto*

qed

lemma *cofinal_mono_map_cf*:
  assumes *Ord*(γ)
  shows $\exists j \in$ *mono_map*(*cf*(γ), *Memrel*(*cf*(γ)), γ, *Memrel*(γ)) . *cf_fun*(*j*,γ)
proof -
  note *assms*
  moreover from *this*
  obtain *A* **where** $A \subseteq \gamma$ *cf*(γ) = *ordertype*(*A*,*Memrel*(γ))
    cofinal(*A*,γ,*Memrel*(γ))
    using *cf_is_ordertype* **by** *blast*
  moreover
  define *j* **where** $j \equiv$ *converse*(*ordermap*(*A*,*Memrel*(γ)))
  moreover from *calculation*
  have *j* :*cf*(γ) $\to_{<}$ γ
    using *ordertype_ord_iso*[*THEN ord_iso_sym*,
        THEN ord_iso_is_mono_map, *THEN mono_map_mono*,
        of A Memrel(γ) γ] *well_ord_Memrel*[*THEN well_ord_subset*]
    by *simp*
  moreover from *calculation*
  have $j \in$ *surj*(*cf*(γ),*A*)
    using *well_ord_Memrel*[*THEN well_ord_subset*, *THEN ordertype_ord_iso*,
        THEN ord_iso_sym, *of* γ *A*, *THEN ord_iso_is_bij*,
        THEN bij_is_surj]
    by *simp*
  with ‹*cofinal*(*A*,γ,*Memrel*(γ))›
  have *cf_fun*(*j*,γ)
    using *cofinal_range_iff_cofinal_fun*[*of j* γ *Memrel*(γ)]
      surj_range[*of j cf*(γ) *A*] *surj_is_fun fun_is_function*
    by *fastforce*
  with ‹$j \in$ *mono_map*(_,_,_,_)›
  show *?thesis* **by** *auto*
qed

3.2 The factorization lemma

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

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

This is a consequence that the proof encapsulates uses of transfinite recursion in the basic theory of cofinality; indeed, only one use is needed. In the setting of Isabelle/ZF, this is convenient since the machinery of recursion is pretty clumsy. On the downside, this way of presenting things results in a longer proof of the factorization lemma. This approach was taken by the author in the notes [8] for an introductory course in Set Theory.

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

locale *cofinal_factor* =
  fixes $j\ \delta\ \xi\ \gamma\ f$
  assumes *j_mono*: $j :\xi \to_< \gamma$
    and *ords*: $Ord(\delta)\ Ord(\xi)\ Limit(\gamma)$
    and *f_type*: $f: \delta \to \gamma$
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 $g \in \xi \to \delta$ such that the composition $f\ O\ g$ is still cofinal but also increasing.

definition
  factor_body :: $[i,i,i] \Rightarrow o$ **where**
  $factor\_body(\beta,h,x) \equiv (x{\in}\delta \wedge j`\beta \le f`x \wedge (\forall\,\alpha{<}\beta\ .\ f`(h`\alpha) < f`x)) \vee x{=}\delta$

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 ‹$\alpha{<}\beta$›
  have $\alpha{\in}\beta$ **using** *ltD* **by** *simp*
  moreover
  note ‹$\beta{\in}\xi$›
  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 ‹$\beta{\in}\xi$›
  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 $j`\beta \leq f`\vartheta \Longrightarrow j`\alpha \leq f`\vartheta$ **for** ϑ **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 ‹$y{<}\alpha$›
also
note ‹$\alpha{<}\beta$›
finally
have $y{<}\beta$ **by** *simp*
with ‹$\forall\, x{<}\beta.\ f`((\lambda w{\in}\beta.\ G(w))`x) < f`z$›
have $f\ `\ ((\lambda w{\in}\beta.\ G(w))\ `\ y) < f\ `\ z$ **by** *simp*
moreover from ‹$y{<}\alpha$› ‹$y{<}\beta$›
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 ‹*factor_body*(β,$\lambda x{\in}\beta.\ G(x)$,$x$)›
ultimately
show *?thesis*
unfolding *factor_body_def* **by** *blast*
qed

lemma *factor_body_simp*[*simp*]: *factor_body*(α,g,δ)
unfolding *factor_body_def* **by** *simp*

lemma *factor_rec_mono*:
assumes
$\beta{\in}\xi\ \alpha{<}\beta$
shows
factor_rec(α,$\lambda x{\in}\alpha.\ G(x)$) $\leq$ *factor_rec*(β,$\lambda x{\in}\beta.\ G(x)$)
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(β) $\equiv$ *transrec*(β,*factor_rec*)

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

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

The factor satisfies the predicate body of the minimization.

lemma *factor_body_factor*:
$factor\_body(\alpha,\lambda x\in\alpha.\ factor(x),factor(\alpha))$
using *ords factor_unfold*[*of* α]
LeastI[*of factor_body*(_,_) δ]
unfolding *factor_rec_def* **by** *simp*

lemma *factor_type* [*TC*]: $Ord(factor(\alpha))$
using *ords factor_unfold*[*of* α]
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 $\beta\in\xi$ $\alpha<\beta$ $factor(\beta)\neq\delta$
shows $f`factor(\alpha) < f`factor(\beta)$
proof -
from *assms*
have $f ` ((\lambda x\in\beta.\ factor(x)) ` \alpha) < f ` factor(\beta)$
using *factor_unfold*[*of* β] *ords LeastI*[*of factor_body*$(\beta,\lambda x\in\beta.\ factor(x))$]
unfolding *factor_rec_def factor_body_def*
by (*auto simp del*:*beta_if*)
with ‹$\alpha<\beta$›
show *?thesis* **using** *ltD* **by** *auto*
qed

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

lemma *factor_in_delta*:
assumes $factor(\beta) \neq \delta$

shows *factor*(β) $\in$ δ
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 $\equiv$ $\lambda\beta\in\xi$. *factor*(β)

lemma *fun_factor_is_mono_map*:
assumes $\bigwedge\beta$. $\beta \in \xi \Longrightarrow$ *factor*(β) $\neq \delta$
shows *fun_factor* $\in$ *mono_map*(ξ, *Memrel*(ξ), δ, *Memrel*(δ))
unfolding *mono_map_def*
proof (*intro CollectI ballI impI*)

Proof that *fun_factor* respects membership:

fix α β
assume $\alpha\in\xi$ $\beta\in\xi$
moreover
note *assms*
moreover from *calculation*
have *factor*(α)$\neq\delta$ *factor*(β)$\neq\delta$ *Ord*(β)
using *factor_in_delta Ord_in_Ord ords* **by** *auto*
moreover
assume $\langle\alpha, \beta\rangle \in$ *Memrel*(ξ)
ultimately
show $\langle$*fun_factor* ‘ α, *fun_factor* ‘ $\beta\rangle \in$ *Memrel*(δ)
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* : $\xi \to \delta$
unfolding *fun_factor_def*
using *ltI lam_type factor_in_delta* **by** *simp*
qed

lemma *f_fun_factor_is_mono_map*:
assumes $\bigwedge\beta$. $\beta \in \xi \Longrightarrow$ *factor*(β) $\neq \delta$
shows *f O fun_factor* $\in$ *mono_map*(ξ, *Memrel*(ξ), γ, *Memrel*(γ))
unfolding *mono_map_def*
using *f_type*
proof (*intro CollectI ballI impI comp_fun*[*of* _ _ δ])
from *assms*
show *fun_factor* : $\xi \to \delta$
using *fun_factor_is_mono_map mono_map_is_fun* **by** *simp*

Proof that *f O fun_factor* respects membership

fix α β
assume $\langle\alpha, \beta\rangle \in$ *Memrel*(ξ)
then
have $\alpha<\beta$
  using *Ord_in_Ord*[*of* ξ] *ltI ords* **by** *blast*
assume $\alpha\in\xi$ $\beta\in\xi$
moreover from *this* **and** *assms*
have *factor*(α)$\neq\delta$ *factor*(β)$\neq\delta$ **by** *auto*
moreover
have *Ord*(γ) $\gamma\neq 0$ **using** *ords Limit_is_Ord* **by** *auto*
moreover
note ‹$\alpha<\beta$› ‹*fun_factor* : $\xi \to \delta$›
ultimately
show $\langle$(*f O fun_factor*) ‘ α, (*f O fun_factor*) ‘ $\beta\rangle \in$ *Memrel*(γ)
  using *ltD*[*of f* ‘ *factor*(α) *f* ‘ *factor*(β)]
    f_factor_increasing apply_in_codomain_Ord f_type
  unfolding *fun_factor_def* **by** *auto*
qed

end — *cofinal_factor*

We state next the factorization lemma.

lemma *cofinal_fun_factorization*:
notes *le_imp_subset* [*dest*] *lt_trans2* [*trans*]
assumes
  Ord(δ) *Limit*(γ) *f*: $\delta \to \gamma$ *cf_fun*(*f*,γ)
shows
  $\exists\, g \in cf(\gamma) \to_{<} \delta.\ f\ O\ g : cf(\gamma) \to_{<} \gamma\ \wedge$
    cofinal_fun(*f O g*,γ,*Memrel*(γ))
proof -
from ‹*Limit*(γ)›
have *Ord*(γ) **using** *Limit_is_Ord* **by** *simp*
then
obtain *j* **where** *j* :$cf(\gamma) \to_{<} \gamma$ *cf_fun*(*j*,γ)
  using *cofinal_mono_map_cf* **by** *blast*
then
have *domain*(*j*) = *cf*(γ)
  using *domain_of_fun mono_map_is_fun* **by** *force*
from ‹$j \in$ _› *assms*
interpret *cofinal_factor j* δ *cf*(γ)
  by (*unfold_locales*) (*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 $cf(\gamma)$ can be cofinal in γ by definition of cofinality. Hence there must be a witness that satisfies the first disjunct.

have *factor_not_delta*: *factor*(β)≠δ **if** β ∈ *cf*(γ) **for** β

For this, we induct on β ranging over *cf*(γ).

```
proof (induct β rule:Ord_induct[OF _ Ord_cf[of γ]])
  case 1 with that show ?case .
next
  case (2 β)
  then
  have IH: z∈β ⟹ factor(z)≠δ for z by simp
  define h where h ≡ λx∈β. f‘factor(x)
  from IH
  have z∈β ⟹ factor(z) ∈ δ for z
    using factor_in_delta by blast
  with ‹f:δ→γ›
  have h : β → γ unfolding h_def using apply_funtype lam_type by auto
  then
  have h : β →< γ
    unfolding mono_map_def
  proof (intro CollectI ballI impI)
    fix x y
    assume x∈β y∈β
    moreover from this and IH
    have factor(y) ≠ δ by simp
    moreover from calculation and ‹h ∈ β → γ›
    have h‘x ∈ γ h‘y ∈ γ by simp_all
    moreover from ‹β∈cf(γ)› and ‹y∈β›
    have y ∈ cf(γ)
      using Ord_trans Ord_cf by blast
    moreover from this
    have Ord(y)
      using Ord_cf Ord_in_Ord by blast
    moreover
    assume ⟨x,y⟩ ∈ Memrel(β)
    moreover from calculation
    have x<y by (blast intro:ltI)
    ultimately
    show ⟨h ‘ x, h ‘ y⟩ ∈ Memrel(γ)
      unfolding h_def using f_factor_increasing ltD by (auto)
  qed
  with ‹β∈cf(γ)› ‹Ord(γ)›
  have ordertype(h‘‘β,Memrel(γ)) = β
    using mono_map_ordertype_image[of β] Ord_cf Ord_in_Ord by blast
  also
  note ‹β ∈cf(γ)›
  finally
  have ordertype(h‘‘β,Memrel(γ)) ∈ cf(γ) by simp
  moreover from ‹h ∈ β → γ›
  have h‘‘β ⊆ γ
    using mono_map_is_fun Image_sub_codomain by blast
```

ultimately
have ¬ *cofinal*(*h*‘‘β,γ,*Memrel*(γ))
using *ordertype_in_cf_imp_not_cofinal* **by** *simp*
then
obtain *α_0* **where** *α_0*∈γ ∀ *x*∈*h* ‘‘ β. ¬ ⟨*α_0*, *x*⟩ ∈ *Memrel*(γ) ∧ *α_0* ≠ *x*
unfolding *cofinal_def* **by** *auto*
with ‹*Ord*(γ)› ‹*h*‘‘β ⊆ γ›
have ∀ *x*∈*h* ‘‘ β. *x* ∈ *α_0*
using *well_ord_Memrel*[*of* γ] *well_ord_is_linear*[*of* γ *Memrel*(γ)]
unfolding *linear_def* **by** *blast*
from ‹*α_0* ∈ γ› ‹*j* ∈ *mono_map*(_,_,γ,_)› ‹*Ord*(γ)›
have *j*‘β ∈ γ
using *mono_map_is_fun apply_in_codomain_Ord* **by** *force*
with ‹*α_0* ∈ γ› ‹*Ord*(γ)›
have *α_0* ∪ *j*‘β ∈ γ
using *Un_least_mem_iff Ord_in_Ord* **by** *auto*
with ‹*cf_fun*(*f*,γ)›
obtain ϑ **where** ϑ∈*domain*(*f*) ⟨*α_0* ∪ *j*‘β, *f* ‘ ϑ⟩ ∈ *Memrel*(γ) ∨ *α_0* ∪ *j*‘β = *f* ‘ ϑ
by (*auto simp add*:*cofinal_fun_def*) *blast*
moreover from *this* **and** ‹*f*:δ→γ›
have ϑ ∈ δ **using** *domain_of_fun* **by** *auto*
moreover
note ‹*Ord*(γ)›
moreover from *this* **and** ‹*f*:δ→γ› ‹*α_0* ∈ γ›
have *Ord*(*f*‘ϑ)
using *apply_in_codomain_Ord Ord_in_Ord* **by** *blast*
moreover from *calculation* **and** ‹*α_0* ∈ γ› **and** ‹*Ord*(δ)› **and** ‹*j*‘β ∈ γ›
have *Ord*(*α_0*) *Ord*(*j*‘β) *Ord*(ϑ)
using *Ord_in_Ord* **by** *auto*
moreover from ‹∀ *x*∈*h* ‘‘ β. *x* ∈ *α_0*› ‹*Ord*(*α_0*)› ‹*h*:β→γ›
have *x*∈β ⟹ *h*‘*x* < *α_0* **for** *x*
using *fun_is_function*[*of h* β λ_. γ]
Image_subset_Ord_imp_lt domain_of_fun[*of h* β λ_. γ]
by *blast*
moreover
have *x*∈β ⟹ *h*‘*x* < *f*‘ϑ **for** *x*
proof -
fix *x*
assume *x*∈β
with ‹∀ *x*∈*h* ‘‘ β. *x* ∈ *α_0*› ‹*Ord*(*α_0*)› ‹*h*:β→γ›
have *h*‘*x* < *α_0*
using *fun_is_function*[*of h* β λ_. γ]
Image_subset_Ord_imp_lt domain_of_fun[*of h* β λ_. γ]
by *blast*
also from ‹⟨*α_0* ∪ _, *f* ‘ ϑ⟩ ∈ *Memrel*(γ) ∨ *α_0* ∪ _= *f* ‘ ϑ›
‹*Ord*(*f*‘ϑ)› ‹*Ord*(*α_0*)› ‹*Ord*(*j*‘β)›
have *α_0* ≤ *f*‘ϑ
using *Un_leD1*[*OF leI* [*OF ltI*]] *Un_leD1*[*OF le_eqI*] **by** *blast*

finally
show *h'x < f'ϑ* **.**
qed
ultimately
have *factor_body(β,λx∈β. factor(x),ϑ)*
unfolding *h_def factor_body_def* **using** *ltD*
by (*auto dest:Un_memD2 Un_leD2*[*OF le_eqI*])
with ‹*Ord(ϑ)*›
have *factor(β) ≤ ϑ*
using *factor_unfold*[*of β*] *Least_le* **unfolding** *factor_rec_def* **by** *auto*
with ‹*ϑ∈δ*› ‹*Ord(δ)*›
have *factor(β) ∈ δ*
using *leI*[*of ϑ*] *ltI*[*of ϑ*] **by** (*auto dest:ltD*)
then
show *?case* **by** (*auto elim:mem_irrefl*)
qed
moreover
have *cofinal_fun(f O fun_factor, γ, Memrel(γ))*
proof (*intro cofinal_funI*)
fix *a*
assume *a ∈ γ*
with ‹*cf_fun(j,γ)*› ‹*domain(j) = cf(γ)*›
obtain *x* **where** *x∈cf(γ) a ∈ j'x ∨ a = j'x*
by (*auto simp add:cofinal_fun_def*) *blast*
with *factor_not_delta*
have *x ∈ domain(f O fun_factor)*
using *f_fun_factor_is_mono_map mono_map_is_fun domain_of_fun* **by**
force
moreover
have *a ∈ (f O fun_factor) 'x ∨ a = (f O fun_factor) 'x*
proof -
from ‹*x∈cf(γ)*› *factor_not_delta*
have *j ' x ≤ f ' factor(x)*
using *mem_not_refl factor_body_factor factor_in_delta*
unfolding *factor_body_def* **by** *auto*
with ‹*a ∈ j'x ∨ a = j'x*›
have *a ∈ f ' factor(x) ∨ a = f ' factor(x)*
using *ltD* **by** *blast*
with ‹*x∈cf(γ)*›
show *?thesis* **using** *lam_funtype*[*of cf(γ) factor*]
unfolding *fun_factor_def* **by** *auto*
qed
moreover
note ‹*a ∈ γ*›
moreover from *calculation* **and** ‹*Ord(γ)*› **and** *factor_not_delta*
have *(f O fun_factor) 'x ∈ γ*
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 $\exists x \in$ *domain*(*f O fun_factor*). $\langle a$, (*f O fun_factor*) ‘ $x\rangle \in$ *Memrel*(γ)
$\vee$ a = (*f O 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(δ) *Ord*(γ)
f :$\delta \to_{<} \gamma$ f *O* $g \in$ *mono_map*(α,r,γ,*Memrel*(γ))
cofinal_fun(f *O* g,γ,*Memrel*(γ)) g: $\alpha \to \delta$
shows
cf_fun(g,δ)
unfolding *cf_fun_def cofinal_fun_def*
proof
fix a
assume $a \in \delta$
with ‹$f \in$ *mono_map*(δ,_,γ,_)›
have f‘$a \in \gamma$
using *mono_map_is_fun* **by** *force*
with ‹*cofinal_fun*(f *O* g,γ,_)›
obtain y **where** $y \in \alpha$ $\langle f$‘a, (f *O* g) ‘ $y\rangle \in$ *Memrel*(γ) $\vee$ f‘a = (f *O* g) ‘ y
unfolding *cofinal_fun_def* **using** *domain_of_fun*[*OF* ‹g:$\alpha \to \delta$›] **by** *blast*
with ‹g:$\alpha \to \delta$›
have $\langle f$‘a, f ‘ (g ‘ y)$\rangle \in$ *Memrel*(γ) $\vee$ f‘a = f ‘ (g ‘ y) g‘$y \in \delta$
using *comp_fun_apply*[*of g* α δ *y f*] **by** *auto*
with *assms*(*1-3*) **and** ‹$a \in \delta$›
have $\langle a$, g ‘ $y\rangle \in$ *Memrel*(δ) $\vee$ a = g ‘ y
using *Memrel_mono_map_reflects Memrel_mono_map_is_inj*[*of* δ *f* γ γ]
inj_apply_equality[*of f* δ γ] **by** *blast*
with ‹$y \in \alpha$›
show $\exists x \in$ *domain*(g). $\langle a$, g ‘ $x\rangle \in$ *Memrel*(δ) $\vee$ a = g ‘ x
using *domain_of_fun*[*OF* ‹g:$\alpha \to \delta$›] **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(γ) *Ord*(δ) *f*:δ → γ *cf_fun*(*f*,γ)
shows
cf(γ)≤δ
using *assms*
proof (*cases rule*:*Ord_cases*)
case *0*
with ‹*Ord*(δ)›
show *?thesis* **using** *Ord_0_le* **by** *simp*
next
case (*succ* γ)
with *assms*
obtain *x* **where** *x*∈δ *f*'*x*=γ **using** *cf_fun_succ'* **by** *blast*
then
have δ≠*0* **by** *blast*
let *?f*={⟨*0*,*f*'*x*⟩}
from ‹*f*'*x*=γ›
have *?f*:*1*→*succ*(γ)
using *singleton_0 singleton_fun*[*of 0* γ] *singleton_subsetI fun_weaken_type* **by** *simp*
with ‹*Ord*(γ)› ‹*f*'*x*=γ›
have *cf*(*succ*(γ)) = *1* **using** *cf_succ* **by** *simp*
with ‹δ≠*0*› *succ*
show *?thesis* **using** *Ord_0_lt_iff succ_leI* ‹*Ord*(δ)› **by** *simp*
next
case (*limit*)
with *assms*
obtain *g* **where** *g* :*cf*(γ) $\to_<$ δ
using *cofinal_fun_factorization* **by** *blast*
with *assms*
show *?thesis* **using** *mono_map_imp_le* **by** *simp*
qed

lemma *cf_ordertype_cofinal*:
assumes
Limit(γ) *A*⊆γ *cofinal*(*A*,γ,*Memrel*(γ))
shows
cf(γ) = *cf*(*ordertype*(*A*,*Memrel*(γ)))
proof (*intro le_anti_sym*)

We show the result by proving the two inequalities.

from ‹*Limit*(γ)›
have *Ord*(γ)
using *Limit_is_Ord* **by** *simp*
with ‹*A* ⊆ γ›
have *well_ord*(*A*,*Memrel*(γ))
using *well_ord_Memrel well_ord_subset* **by** *blast*
then
obtain *f* α **where** *f*:⟨α, *Memrel*(α)⟩ ≅ ⟨*A*,*Memrel*(γ)⟩ *Ord*(α) α = *order-*

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

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

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

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

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

lemma *cf_le_cardinal*:
assumes *Limit*(γ)
shows *cf*(γ) $\leq$ $|\gamma|$
proof -
from *assms*
have ‹*Ord*(γ)› **using** *Limit_is_Ord* **by** *simp*
then
obtain f **where** $f \in$ *surj*($|\gamma|$,γ)
using *Ord_cardinal_eqpoll* **unfolding** *eqpoll_def bij_def* **by** *blast*
with ‹*Ord*(γ)›
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) γ = *cf*(γ)

shows *Card*(γ)
proof -
from *assms*
have |γ| ⊆ γ
using *Limit_is_Ord Ord_cardinal_le* **by** *blast*
also from ‹γ = *cf*(γ)›
have γ ⊆ *cf*(γ) **by** *simp*
finally
have |γ| ⊆ *cf*(γ) **.**
with *assms*
show *?thesis* **unfolding** *Card_def* **using** *cf_le_cardinal* **by** *force*
qed

lemma *Limit_cf*: **assumes** *Limit*(κ) **shows** *Limit*(*cf*(κ))
using *Ord_cf*[*of* κ, *THEN Ord_cases*]
— *cf*(κ) being 0 or successor leads to contradiction
proof (*cases*)
case *1*
with ‹*Limit*(κ)›
show *?thesis* **using** *cf_zero_iff Limit_is_Ord* **by** *simp*
next
case (*2* α)
moreover
note ‹*Limit*(κ)›
moreover from *calculation*
have *cf*(κ) = *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*(κ) ⟹ *InfCard*(*cf*(κ))
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*(κ) ≤ ν *Limit*(κ)
shows ∃*f*. *f*:ν → κ ∧ *cf_fun*(*f*, κ)
proof -
note *assms*
moreover from *this*
obtain *h* **where** *h_cofinal_mono*: *cf_fun*(*h*,κ)
h :*cf*(κ) $\to_<$ κ
h : *cf*(κ) → κ
using *cofinal_mono_map_cf mono_map_is_fun* **by** *force*
moreover from *calculation*

```
  obtain g where g ∈ inj(cf(κ), ν)
    using le_imp_lepoll by blast
  from this and calculation(2,3,5)
  obtain f where f ∈ surj(ν, cf(κ)) f: ν → cf(κ)
    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 ‹Limit(κ)›
  moreover from calculation
  have cf_fun(h O f,κ)
    using cf_fun_comp by blast
  moreover from calculation
  have h O f ∈ ν -> κ
    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 ∃α∈ν. n < f`α
proof -
  from ‹Limit(κ)› ‹n∈κ›
  have succ(n) ∈ κ
    using Limit_has_succ[OF _ ltI, THEN ltD] by auto
  moreover
  note ‹f: ν → _›
  moreover from this
  have domain(f) = ν
    using domain_of_fun by simp
  moreover
  note ‹cf_fun(f,κ)›
  ultimately
  obtain α where α ∈ ν succ(n) ∈ f`α ∨ succ(n) = f `α
    using cf_funD[THEN cofinal_funD] by blast
  moreover from this
  consider (1) succ(n) ∈ f`α | (2) succ(n) = f `α
    by blast
  then
  have n < f`α
  proof (cases)
    case 1
    moreover
    have n ∈ succ(n) by simp
    moreover
```

note ‹*Limit*(κ)› ‹*f*: ν → _› ‹α ∈ ν›
moreover from *this*
have *Ord*(*f* ‘ α)
  using *apply_type*[*of f* ν λ_. κ, *THEN* [*2*] *Ord_in_Ord*]
  by *blast*
ultimately
show *?thesis*
  using *Ord_trans*[*of n succ*(*n*) *f* ‘ α] *ltI* **by** *blast*
next
case *2*
have *n* ∈ *f* ‘ α **by** (*simp add*:*2*[*symmetric*])
with ‹*Limit*(κ)› ‹*f*: ν → _› ‹α ∈ ν›
show *?thesis*
  using *ltI*
    apply_type[*of f* ν λ_. κ, *THEN* [*2*] *Ord_in_Ord*]
  by *blast*
qed
ultimately
show *?thesis* **by** *blast*
qed

context
includes *Ord_dests Aleph_dests Aleph_intros Aleph_mem_dests 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*(ω) = ω
using *Limit_nat*[*THEN InfCard_cf*] *cf_le_cardinal*[*of* ω]
  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*(γ)
shows *cf*($\aleph_\gamma$) = *cf*(γ)
proof -
note ‹*Limit*(γ)›
moreover from *this*
have (λ*x*∈γ. $\aleph_x$) : γ → $\aleph_\gamma$ (**is** *?f* : _ → _)
  using *lam_funtype*[*of* _ *Aleph*] *fun_weaken_type*[*of* _ _ _ $\aleph_\gamma$] **by** *blast*
moreover from ‹*Limit*(γ)›
have *x* ∈ *y* ⟹ $\aleph_x$ ∈ $\aleph_y$ **if** *x* ∈ γ *y* ∈ γ **for** *x y*
  using *that Ord_in_Ord*[*of* γ] *Ord_trans*[*of* _ _ γ] **by** *blast*
ultimately
have *?f* ∈ *mono_map*(γ,*Memrel*(γ),$\aleph_\gamma$, *Memrel*($\aleph_\gamma$))
  by *auto*

with ‹*Limit*(γ)›
have *?f* ∈ ⟨γ, *Memrel*(γ)⟩ ≅ ⟨*?f*‘‘γ, *Memrel*(ℵ_γ)⟩
using *mono_map_imp_ord_iso_Memrel*[*of* γ ℵ_γ *?f*]
Card_Aleph
by *blast*
then
have *converse*(*?f*) ∈ ⟨*?f*‘‘γ, *Memrel*(ℵ_γ)⟩ ≅ ⟨γ, *Memrel*(γ)⟩
using *ord_iso_sym* **by** *simp*
with ‹*Limit*(γ)›
have *ordertype*(*?f*‘‘γ, *Memrel*(ℵ_γ)) = γ
using *ordertype_eq*[*OF* _ *well_ord_Memrel*]
ordertype_Memrel **by** *auto*
moreover from ‹*Limit*(γ)›
have *cofinal*(*?f*‘‘γ, ℵ_γ, *Memrel*(ℵ_γ))
unfolding *cofinal_def*
proof (*standard*, *intro ballI*)
fix *a*
assume *a*∈ℵ_γ ℵ_γ = (⋃*i*<γ. ℵ_*i*)
moreover from *this*
obtain *i* **where** *i*<γ *a*∈ℵ_*i*
by *auto*
moreover from *this* **and** ‹*Limit*(γ)›
have *Ord*(*i*) **using** *ltD Ord_in_Ord* **by** *blast*
moreover from ‹*Limit*(γ)› **and** *calculation*
have *succ*(*i*) ∈ γ **using** *ltD* **by** *auto*
moreover from *this* **and** ‹*Ord*(*i*)›
have ℵ_*i* < ℵ_*succ*(*i*)
by (*auto*)
ultimately
have ⟨*a*,ℵ_*i*⟩ ∈ *Memrel*(ℵ_γ)
using *ltD* **by** (*auto dest*:*Aleph_increasing*)
moreover from ‹*i*<γ›
have ℵ_*i* ∈ *?f*‘‘γ
using *ltD apply_in_image*[*OF* ‹*?f* : _ → _›] **by** *auto*
ultimately
show ∃ *x*∈*?f* ‘‘ γ. ⟨*a*, *x*⟩ ∈ *Memrel*(ℵ_γ) ∨ *a* = *x* **by** *blast*
qed
moreover
note ‹*?f*: γ → ℵ_γ› ‹*Limit*(γ)›
ultimately
show *cf*(ℵ_γ) = *cf*(γ)
using *cf_ordertype_cofinal*[*OF Limit_Aleph Image_sub_codomain*, *of* γ *?f* γ
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 $(\kappa^{\uparrow \nu 1})^{\uparrow \nu 2} = (\nu 1 \to \kappa)^{\uparrow \nu 2}$
    using *cardinal_eqpoll*
    by (*intro cexp_eqpoll_cong*) (*simp_all add*:*cexp_def*)
  also
  have $\ldots = \kappa^{\uparrow \nu 2 \times \nu 1}$
    unfolding *cexp_def* **using** *curry_eqpoll cardinal_cong* **by** *blast*
  also
  have $\ldots = \kappa^{\uparrow \nu 2 \otimes \nu 1}$
    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*(ν)
  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 $\nu \to \kappa 1 \subseteq \nu \to \kappa 2$
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]*
— $[\![InfCard(\kappa); InfCard(\nu)]\!] \Longrightarrow InfCard(\kappa^{\uparrow\nu})$

4.2 Miscellaneous

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

proof -
  have $(\lambda x\in A.\ f(x)) \in surj(A, \{f(a) .\ a\in A\})$
    unfolding *surj_def* **using** *lam_funtype* **by** *auto*
  then
  show *?thesis*
    using *surj_implies_cardinal_le* **by** *blast*
qed

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

lemma *lt_cardinal_imp_not_subset*: $|A| < |B| \Longrightarrow \neg\ B \subseteq A$
  using *subset_imp_le_cardinal le_imp_not_lt* **by** *blast*

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

lemma *cardinal_UN_le_nat*:
  $(\bigwedge i.\ i\in\omega \Longrightarrow |X(i)| \leq \omega) \Longrightarrow |\bigcup i\in\omega.\ X(i)| \leq \omega$
  by (*simp add*: *cardinal_UN_le InfCard_nat*)

lemma *lepoll_imp_cardinal_UN_le*:
  notes [*dest*] = *InfCard_is_Card Card_is_Ord*
  assumes $InfCard(K)\ J \lesssim K\ \bigwedge i.\ i\in J \Longrightarrow |X(i)| \leq K$
  shows $|\bigcup i\in J.\ X(i)| \leq K$
proof -
  from ‹$J \lesssim K$›
  obtain f **where** $f \in inj(J,K)$ **by** *blast*
  define Y **where** $Y(k) \equiv$ *if* $k\in range(f)$ *then* $X(converse(f)\text{‘}k)$ *else* 0 **for** k
  have $i\in J \Longrightarrow f\text{‘}i \in K$ **for** i
    using *inj_is_fun*[*OF* ‹$f \in inj(J,K)$›] **by** *auto*
  have $(\bigcup i\in J.\ X(i)) \subseteq (\bigcup i\in K.\ Y(i))$
  proof (*standard*, *elim UN_E*)
    fix $x\ i$
    assume $i\in J\ x\in X(i)$
    with ‹$f \in inj(J,K)$› ‹$i\in J \Longrightarrow f\text{‘}i \in K$›
    have $x \in Y(f\text{‘}i)\ f\text{‘}i \in K$
      unfolding *Y_def*
      using *inj_is_fun*[*OF* ‹$f \in inj(J,K)$›]
        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* ‹$\bigwedge i.\ i\in J \Longrightarrow f\text{‘}i \in K$›
  show $|\bigcup i\in J.\ X(i)| \leq K$
    using *inj_converse_fun*[*OF* ‹$f \in inj(J,K)$›] **unfolding** *Y_def*

by (*rule_tac le_trans*[*OF _ cardinal_UN_le*]) (*auto intro*:*Ord_0_le*)+
qed

— For backwards compatibility
lemmas *leqpoll_imp_cardinal_UN_le* = *lepoll_imp_cardinal_UN_le*

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

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

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

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

lemma *lepoll_imp_lepoll_cardinal*: **assumes** $X \lesssim Y$ **shows** $X \lesssim |Y|$
using *assms cardinal_Card_eqpoll_iff*[*of* $|Y|$ *Y*]
lepoll_eq_trans[*of _ _* $|Y|$] **by** *simp*

lemma *lepoll_Un*:
  assumes *InfCard*(κ) $A \lesssim \kappa$ $B \lesssim \kappa$
  shows $A \cup B \lesssim \kappa$
proof -
  have $A \cup B \lesssim$ *sum*(*A*,*B*)
    using *Un_lepoll_sum* **.**
  moreover
  note *assms*
  moreover from *this*
  have |*sum*(*A*,*B*)| ≤ κ ⊕ κ
    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 sum*(*A*,*B*) κ]
      lepoll_trans[*of A*∪*B*]
    by *auto*
qed

lemma *cardinal_Un_le*:
  assumes *InfCard*(κ) |*A*| ≤ κ |*B*| ≤ κ
  shows |*A* ∪ *B*| ≤ κ
  using *assms lepoll_Un le_Card_iff InfCard_is_Card* **by** *auto*

This is the unconditional version under choice of *Cardinal.Finite_cardinal_iff*.

lemma *Finite_cardinal_iff'*: *Finite*(|*i*|) ⟷ *Finite*(*i*)
  using *cardinal_eqpoll_iff eqpoll_imp_Finite_iff* **by** *fastforce*

lemma *cardinal_subset_of_Card*:
  assumes *Card*(γ) *a* ⊆ γ
  shows |*a*| < γ ∨ |*a*| = γ
proof -
  from *assms*
  have |*a*| < |γ| ∨ |*a*| = |γ|
    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) ⟹ |*X*| < γ ⟷ ¬ |*X*| ≥ γ
  using *not_le_iff_lt*
  by *auto*

4.3 Countable and uncountable sets

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

lemma *lepoll_countable*: $X \lesssim Y \Longrightarrow$ *countable*(Y) $\Longrightarrow$ *countable*(X)
using *lepoll_trans*[*of* X Y] **by** *blast*

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

lemma *Finite_imp_countable*: *Finite*(X) $\Longrightarrow$ *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 *countable*(J) $\bigwedge i.\ i\in J \Longrightarrow$ *countable*($X(i)$)
shows *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 $\bigwedge x.\ x \in C \Longrightarrow$ *countable*(x) *countable*(C)
shows *countable*($\bigcup C$)
using *assms countable_imp_countable_UN*[*of* C $\lambda x.\ x$] **by** *simp*

abbreviation
uncountable :: $i \Rightarrow o$ **where**
uncountable(X) $\equiv \neg$ *countable*(X)

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

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

lemma *uncountable_imp_Infinite*: *uncountable*(X) $\Longrightarrow$ *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 *countable*(X) *uncountable*(Y)
shows $\neg$ ($Y \subseteq X$)

using *assms lepoll_trans subset_imp_lepoll*[*of Y X*]
by *blast*

4.4 Results on Alephs

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

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

lemma *le_aleph1_nat*: $Card(k) \Longrightarrow k<\aleph_1 \Longrightarrow k \leq \omega$
by (*simp add*: *Aleph_def Card_lt_csucc_iff Card_nat*)

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

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

lemma *cardinal_Aleph* [*simp*]: $Ord(\alpha) \Longrightarrow |\aleph_\alpha| = \aleph_\alpha$
using *Card_cardinal_eq* **by** *simp*

— Could be proved without using AC
lemma *Aleph_lesspoll_increasing*:
includes *Aleph_intros*
shows $a < b \Longrightarrow \aleph_a \prec \aleph_b$
using *cardinal_lt_iff_lesspoll*[*of* $\aleph_a$ $\aleph_b$] *Card_cardinal_eq*[*of* $\aleph_b$]
lt_Ord lt_Ord2 Card_Aleph[*THEN Card_is_Ord*] **by** *auto*

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

lemma *lt_Aleph_imp_cardinal_UN_le_nat*: *function*(*G*) $\Longrightarrow$ *domain*(*G*) $\lesssim \omega$ $\Longrightarrow$
$\forall n\in domain(G).\ |G`n|<\aleph_1 \Longrightarrow |\bigcup n\in domain(G).\ G`n|\le\omega$
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** $|G`n| \le \omega$
using *le_aleph1_nat* **by** *simp*
}
then
have $n\in ?N \Longrightarrow |G`n| \le \omega$ **for** *n* **.**
with ‹*?N* $\lesssim \omega$›
show *?thesis*
using *InfCard_nat lepoll_imp_cardinal_UN_le* **by** *simp*
qed

lemma *Aleph1_eq_cardinal_vimage*: $f:\aleph_1\to\omega \Longrightarrow \exists n\in\omega.\ |f\text{-}``\{n\}| = \aleph_1$
proof -
assume $f:\aleph_1\to\omega$
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*=$\lambda n\in range(f).\ f\text{-}``\{n\}$
from ‹$f:\aleph_1\to\omega$›
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 ‹$f:\aleph_1\to\omega$›
have $n\in\omega \Longrightarrow f\text{-}``\{n\} \subseteq \aleph_1$ **for** *n*
using *Pi_vimage_subset* **by** *simp*

with ‹*range*(*f*) ⊆ *ω*›
have $\aleph_1$ = (⋃*n*∈*range*(*f*). *f*-‘‘{*n*})
proof (*intro equalityI*, *intro subsetI*)
fix *x*
assume *x* ∈ $\aleph_1$
with ‹*f*:$\aleph_1$→*ω*› ‹*function*(*f*)› ‹*domain*(*f*) = $\aleph_1$›
have *x* ∈ *f*-‘‘{*f*‘*x*} *f*‘*x* ∈ *range*(*f*)
using *function_apply_Pair vimage_iff apply_rangeI* **by** *simp_all*
then
show *x* ∈ (⋃*n*∈*range*(*f*). *f*-‘‘{*n*}) **by** *auto*
qed *auto*
{
assume ∀ *n*∈*range*(*f*). |*f*-‘‘{*n*}| < $\aleph_1$
then
have ∀ *n*∈*domain*(*?G*). |*?G*‘*n*| < $\aleph_1$
using *zero_lt_Aleph1* **by** (*auto*)
with ‹*function*(*?G*)› ‹*domain*(*?G*) ≲ *ω*›
have |⋃*n*∈*domain*(*?G*). *?G*‘*n*|≤*ω*
using *lt_Aleph_imp_cardinal_UN_le_nat* **by** *blast*
then
have |⋃*n*∈*range*(*f*). *f*-‘‘{*n*}|≤*ω* **by** *simp*
with ‹$\aleph_1$ = _›
have |$\aleph_1$| ≤ *ω* **by** *simp*
then
have $\aleph_1$ ≤ *ω*
using *Card_Aleph Card_cardinal_eq*
by *simp*
then
have *False*
using *nat_lt_Aleph1* **by** (*blast dest*:*lt_trans2*)
}
with ‹*range*(*f*)⊆*ω*›
obtain *n* **where** *n*∈*ω* ¬(|*f* -‘‘ {*n*}| < $\aleph_1$)
by *blast*
moreover from *this*
have $\aleph_1$ ≤ |*f*-‘‘{*n*}|
using *not_lt_iff_le Card_is_Ord* **by** *auto*
moreover
note ‹*n*∈*ω* ⟹ *f*-‘‘{*n*} ⊆ $\aleph_1$›
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 ((≈) versus *cardinal*)
lemma *eqpoll_Aleph1_cardinal_vimage*:
assumes *X* ≈ $\aleph_1$ *f* : *X* → *ω*

shows $\exists\, n{\in}\omega.\ |f\text{-``}\{n\}| = \aleph_1$
proof -
from *assms*
obtain *g* **where** $g{\in}bij(\aleph_1,X)$
using *eqpoll_sym* **by** *blast*
with ‹$f : X \to \omega$›
have $f\ O\ g : \aleph_1 \to \omega\ \ converse(g) \in bij(X, \aleph_1)$
using *bij_is_fun comp_fun bij_converse_bij* **by** *blast+*
then
obtain *n* **where** $n{\in}\omega\ |(f\ O\ g)\text{-``}\{n\}| = \aleph_1$
using *Aleph1_eq_cardinal_vimage* **by** *auto*
then
have $\aleph_1 = |converse(g)\ \text{``}\ (f\ \text{-``}\{n\})|$
using *image_comp converse_comp*
unfolding *vimage_def* **by** *simp*
also from ‹$converse(g) \in bij(X, \aleph_1)$› ‹$f\colon X\to \omega$›
have $\ldots = |f\ \text{-``}\{n\}|$
using *range_of_subset_eqpoll*[*of converse*(*g*) *X* _ *f* -‘‘{*n*}]
bij_is_inj cardinal_cong bij_is_fun eqpoll_sym Pi_vimage_subset
by *fastforce*
finally
show *?thesis* **using** ‹$n{\in}\omega$› **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
$\bigwedge X.\ |X| < \gamma \Longrightarrow X \subseteq G \Longrightarrow \exists\, a{\in}G.\ \forall\, s{\in}X.\ Q(s,a)\ \ b{\in}G\ \ Card(\gamma)$
shows
$\exists\, S.\ S : \gamma \to G \wedge (\forall\, \alpha \in \gamma.\ \forall\, \beta \in \gamma.\ \ \alpha{<}\beta \longrightarrow Q(S\text{‘}\alpha,S\text{‘}\beta))$
proof -
let $?cdlt\gamma{=}\{X{\in}Pow(G)\ .\ |X|{<}\gamma\}$ — “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 $\exists f.\ f \in Pi(?cdlt\gamma,?inQ)$
using *AC_ball_Pi*[*of ?cdlt*γ *?inQ*] **by** *simp*
then
obtain *f* **where** $f\_type{:}f \in Pi(?cdlt\gamma,?inQ)$
by *auto*
moreover
define *Cb* **where** $Cb \equiv \lambda\_{\in}Pow(G)\text{-}?cdlt\gamma.\ b$

```
moreover from ‹b∈G›
have Cb ∈ Pow(G)-?cdltγ → G
  unfolding Cb_def by simp
moreover
note ‹Card(γ)›
ultimately
have f ∪ Cb : (∏ x∈Pow(G). ?inQ(x) ∪ G) using
    fun_Pi_disjoint_Un[ of f ?cdltγ  ?inQ Cb Pow(G)-?cdltγ λ_.G]
    Diff_partition[of {X∈Pow(G). |X|<γ} Pow(G), OF Collect_subset]
  by auto
moreover
have ?inQ(x) ∪ G = G for x by auto
ultimately
have f ∪ Cb : Pow(G) → G by simp
define S where S≡λα∈γ. rec_constr(f ∪ Cb, α)
from ‹f ∪ Cb: Pow(G) → G› ‹Card(γ)›
have S : γ → G
  using Ord_in_Ord unfolding S_def
  by (intro lam_type rec_constr_type) auto
moreover
have ∀α∈γ. ∀β∈γ. α < β ⟶ Q(S ‘ α, S ‘ β)
proof (intro ballI impI)
  fix α β
  assume β∈γ
  with ‹Card(γ)›
  have {rec_constr(f ∪ Cb, x) . x∈β} = {S‘x . x ∈ β}
    using Ord_trans[OF _ _ Card_is_Ord, of _ β γ]
    unfolding S_def
    by auto
  moreover from ‹β∈γ› ‹S : γ → G› ‹Card(γ)›
  have {S‘x . x ∈ β} ⊆ G
    using Ord_trans[OF _ _ Card_is_Ord, of _ β γ]
      apply_type[of S γ λ_. G] by auto
  moreover from ‹Card(γ)› ‹β∈γ›
  have |{S‘x . x ∈ β}| < γ
    using cardinal_RepFun_le[of β]  Ord_in_Ord
      lt_trans1[of |{S‘x . x ∈ β}| |β| γ]
      Card_lt_iff[THEN iffD2, of β γ, OF _ _ ltI]
    by force
  moreover
  have ∀x∈β. Q(S‘x, f ‘ {S‘x . x ∈ β})
  proof -
    from calculation and f_type
    have f ‘ {S‘x . x ∈ β} ∈ {a∈G. ∀x∈β. Q(S‘x,a)}
      using apply_type[of f ?cdltγ ?inQ {S‘x . x ∈ β}]
      by blast
    then
    show ?thesis by simp
  qed
```

```
    moreover
    assume α∈γ α < β
    moreover
    note ‹β∈γ› ‹Cb ∈ Pow(G)-?cdltγ → G›
    ultimately
    show Q(S ‘ α, S ‘ β)
      using fun_disjoint_apply1[of {S‘x . x ∈ β} Cb f]
        domain_of_fun[of Cb] ltD[of α β]
      by (subst (2) S_def, auto) (subst rec_constr_unfold, auto)
  qed
  ultimately
  show ?thesis by blast
qed
```

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

```
lemma Infinite_iff_lepoll_nat: Infinite(X) ⟷ ω ≲ X
proof
  assume Infinite(X)
  then
  obtain b where b∈X
    using Infinite_not_empty by auto
  {
    fix Y
    assume |Y| < ω
    then
    have Finite(Y)
      using Finite_cardinal_iff' ltD nat_into_Finite by blast
    with ‹Infinite(X)›
    have X ≠ Y by auto
  }
  with ‹b∈X›
  obtain S where S : ω → X  ∀α∈ω. ∀β∈ω. α < β ⟶ S‘α ≠ S‘β
    using bounded_cardinal_selection[of ω X λx y. x≠y]
      Card_nat by blast
  moreover from this
  have α ∈ ω ⟹ β ∈ ω ⟹ α≠β ⟹ S‘α ≠ S‘β for α β
    by (rule_tac lt_neq_symmetry[of ω λα β. S‘α ≠ S‘β])
      auto
  ultimately
  show ω ≲ X
    unfolding lepoll_def inj_def by blast
qed (intro lepoll_nat_imp_Infinite)

lemma Infinite_InfCard_cardinal: Infinite(X) ⟹ 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 $F \in$ *Finite_to_one*$(X,Y) \cap$ *surj*(X,Y) *Infinite*(X)
  shows $|Y| = |X|$
proof -
  from ‹$F \in$ *Finite_to_one*$(X,Y) \cap$ *surj*(X,Y)›
  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 ‹$X = (\bigcup y\in Y. \{x\in X . F`x = y\})$› **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 $y \in Y \Longrightarrow |\{x\in X . F`x = y\}| \leq |Y|$ **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 ‹$F \in$ *Finite_to_one*$(X,Y) \cap$ *surj*(X,Y)›
    have $|Y| \leq |X|$
      using *surj_implies_cardinal_le* **by** *auto*
    moreover
    note ‹$X = (\bigcup y\in Y. \{x\in X . F`x = y\})$›
    ultimately
    show *?thesis*
      using *le_anti_sym* **by** *auto*
  qed
qed

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

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

end
theory Konig
  imports
    Cofinality
    Cardinal_Library

begin
```

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

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

have $cf(z^+) < z^+$
  using *cf_le_cardinal[of* z^+*, THEN le_iff[THEN iffD1]]*
    Card_cardinal_eq
  by *simp*
ultimately
obtain G **where** $G{:}cf(z^+)\to z^+$ $\forall\beta\in z^+.\ \exists y\in cf(z^+).\ \beta < G`y$
  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 ‹$Card(z)$› ‹$Card(z^+)$› ‹$Ord(z^+)$›
have $\forall\beta\in cf(z^+).\ |G`\beta| \le z$
  using *apply_type[of G* $cf(z^+)$ λ*_.* 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 ‹$cf(z^+) < z^+$› ‹$InfCard(z)$› ‹$Ord(z)$›
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 ‹$cf(z^+) < z^+$› ‹$\forall\beta\in cf(z^+).\ |G`\beta| \le$ _› ‹$InfCard(z)$›
have $|\bigcup\beta\in cf(z^+).\ G`\beta| \le z$
  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 ‹$Ord(z)$›
moreover from ‹$\forall\beta\in z^+.\ \exists y\in cf(z^+).\ \beta < G`y$› **and** *this*
have $z^+ \subseteq (\bigcup\beta\in cf(z^+).\ G`\beta)$
  by (*blast dest:ltD*)
ultimately
have $z^+ \le z$
  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 ‹$Ord(z)$›
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*: $Ord(z) \Longrightarrow cf(\aleph_{succ(z)}) = \aleph_{succ(z)}$
  using *Aleph_succ cf_csucc InfCard_Aleph* **by** *simp*

4.6 König's Theorem

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

It is proved rather straightforwardly with the tools already developed.

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

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

lemma cf_cexp:
  assumes
    Card(κ) InfCard(ν) 2 ≤ κ
  shows
    ν < cf(κ↑ν)
proof (rule ccontr)
  assume ¬ ν < cf(κ↑ν)
  with ‹InfCard(ν)›
  have cf(κ↑ν) ≤ ν
    using not_lt_iff_le Ord_cf InfCard_is_Card Card_is_Ord by simp
  moreover
  note assms
  moreover from calculation
  have InfCard(κ↑ν) using InfCard_cexp by simp
```

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

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

corollary *cf_continuum*: $\aleph_0 < cf(2^{\uparrow\aleph_0})$
using *cf_cexp InfCard_Aleph nat_into_Card* **by** *simp*

corollary *continuum_not_eq_Aleph_nat*: $2^{\uparrow\aleph_0} \neq \aleph_\omega$
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 $\forall A\in F.\ Finite(A)$ $F \approx \aleph_1$
shows $\exists D.\ D \subseteq F \wedge delta\_system(D) \wedge D \approx \aleph_1$
proof -

Since all members are finite,

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

note ‹$F \approx \aleph_1$›
moreover from *calculation*

there are uncountably many have the same cardinal:

obtain n **where** $n\in\omega$ $|?cards$ -‘‘ $\{n\}| = \aleph_1$
  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 $A\in G \Longrightarrow |A| = n$ **and** $G \approx \aleph_1$ **for** A
  using *cardinal_Card_eqpoll_iff* **by** *auto*
with ‹$n\in\omega$›

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 $\exists D.\ D \subseteq G \wedge delta\_system(D) \wedge D \approx \aleph_1$
proof (*induct arbitrary*:G)
  case *0* — This case is impossible
  then
  have $G \subseteq \{0\}$
    using *cardinal_0_iff_0* **by** *auto*
  with ‹$G \approx \aleph_1$›
  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 $\forall a\in G.\ Finite(a)$
    using *Finite_cardinal_iff' nat_into_Finite*[*of succ*(n)]
    by *fastforce*
  show $\exists D.\ D \subseteq G \wedge delta\_system(D) \wedge D \approx \aleph_1$
  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 ‹⋀*A*. *A*∈*G* ⟹ |*A*|=*succ*(*n*)› ‹∀ *a*∈*G*. *Finite*(*a*)›
  and *this*
have *p*∈*A* ⟹ *A*∈*G* ⟹ |*A* - {*p*}| = *n* **for** *A*
  using *Finite_imp_succ_cardinal_Diff*[*of* _ *p*] **by** *force*
moreover from *this* **and** ‹*n*∈ω›
have ∀ *a*∈*?F*. *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* ∈ {*X* ∈ *G* . *p* ∈ *X*}}:

note ‹(⋀*A*. *A* ∈ *?F* ⟹ |*A*| = *n*) ⟹ *?F* ≈ ℵ*1* ⟹
    ∃ *D*. *D* ⊆ *?F* ∧ *delta_system*(*D*) ∧ *D* ≈ ℵ*1*›
ultimately
obtain *D* **where** *D*⊆{*A*-{*p*} . *A*∈{*X*∈*G*. *p*∈*X*}} *delta_system*(*D*) *D* ≈ ℵ*1*
  by *auto*
moreover from *this*
obtain *r* **where** ∀ *A*∈*D*. ∀ *B*∈*D*. *A* ≠ *B* ⟶ *A* ∩ *B* = *r*
  by *fastforce*
then
have ∀ *A*∈*D*.∀ *B*∈*D*. *A*∪{*p*} ≠ *B*∪{*p*}⟶(*A* ∪ {*p*}) ∩ (*B* ∪ {*p*}) = *r*∪{*p*}
  by *blast*
ultimately
have *delta_system*({*B* ∪ {*p*} . *B*∈*D*}) (**is** *delta_system*(*?D*))
  by *fastforce*
moreover from ‹*D* ≈ ℵ*1*›
have |*D*| = ℵ*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* ≈ ℵ*1*
  using *cardinal_map_Un*[*of D* {*p*}] *naturals_lt_nat*
   cardinal_eqpoll_iff[*THEN iffD1*] **by** *simp*
moreover
note ‹*D* ⊆ {*A*-{*p*} . *A*∈{*X*∈*G*. *p*∈*X*}}›
have *?D* ⊆ *G*
proof -
  {
   fix *A*
   assume *A*∈*G* *p*∈*A*
   moreover from *this*
   have *A* = *A* - {*p*} ∪ {*p*}
    by *blast*
   ultimately
   have *A* -{*p*} ∪ {*p*} ∈ *G*
    by *auto*
  }
  with ‹*D* ⊆ {*A*-{*p*} . *A*∈{*X*∈*G*. *p*∈*X*}}›

show *?thesis*
by *blast*
qed
ultimately
show $\exists D.\ D \subseteq G \wedge delta\_system(D) \wedge D \approx \aleph_1$ **by** *auto*
next
case *False*
note ‹¬ ($\exists p.\ \{A \in G\ .\ p \in A\} \approx \aleph_1$)› — the other case
moreover from ‹$G \approx \aleph_1$›
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 $countable(S) \Longrightarrow countable(\{A \in G\ .\ S \cap A \neq 0\})$ **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* ω *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 ‹$n{\in}\omega$› ‹$\bigwedge A.\ A{\in}G \Longrightarrow |A| = succ(n)$›
have $A{\in}G \Longrightarrow Finite(A)$ **for** A
using *cardinal_Card_eqpoll_iff*
unfolding *Finite_def* **by** *fastforce*
with ‹$X{\subseteq}G$›
have $A{\in}X \Longrightarrow countable(A)$ **for** A
using *Finite_imp_countable* **by** *auto*
with ‹$|X| < \aleph_1$›
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 ‹$countable(\_) \Longrightarrow countable(\{A \in G\ .\ \_\ \cap A \neq 0\})$›
have $countable(\{A \in G\ .\ (\bigcup X) \cap A \neq 0\})$ **.**
with ‹$G \approx \aleph_1$›
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 ‹$G \approx \aleph_1$›
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** $S{:}\aleph_1{\to}G\ \alpha{\in}\aleph_1 \Longrightarrow \beta{\in}\aleph_1 \Longrightarrow \alpha{<}\beta \Longrightarrow S`\alpha \cap S`\beta = 0$
for $\alpha\ \beta$
using *bounded_cardinal_selection*[*of* $\aleph_1$ *G* $\lambda s\ a.\ s \cap a = 0$ *b*]
by *force*
then
have $\alpha \in \aleph_1 \Longrightarrow \beta \in \aleph_1 \Longrightarrow \alpha{\neq}\beta \Longrightarrow S`\alpha \cap S`\beta = 0$ **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** ‹$\bigwedge A.\ A{\in}G \Longrightarrow |A| = succ(n)$› ‹$S : \aleph_1 \to G$›
have $S \in inj(\aleph_1,\ G)$
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 $range(S) \approx \aleph_1$
using *inj_bij_range eqpoll_sym* **unfolding** *eqpoll_def* **by** *fast*
moreover from *calculation*
have $range(S) \subseteq G$
using *inj_is_fun range_fun_subset_codomain* **by** *fast*
ultimately
show $\exists D.\ D \subseteq G \wedge delta\_system(D) \wedge D \approx \aleph_1$
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 ‹$G \subseteq F$›
show *?thesis* **by** *blast*
qed

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

shows $\exists D.\ D \subseteq F \wedge delta\_system(D) \wedge D \approx \aleph_1$
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 ‹$\forall A\in F.\ Finite(A)$› **and** *this*
have $\forall A\in S.\ Finite(A)$ **by** *auto*
ultimately
show *?thesis* **using** *delta_system_Aleph1*[*of S*]
by *auto*
qed

end

5.1 Application to Cohen posets

theory *Cohen_Posets*
imports
Delta_System

begin

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

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

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

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

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

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

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

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**
$Fn(I,J) \equiv \bigcup\{(d\rightarrow J)\ .\ d \in \{x \in Pow(I).\ \ Finite(x)\}\}$

abbreviation
  Supset :: $i \Rightarrow i \Rightarrow o$ (**infixl** ‹⊇› *50*) **where**
  $f \supseteq g \equiv g \subseteq f$

lemma *FnI*[*intro*]:
  assumes $p : d \to J$ $d \subseteq I$ *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 $\exists d.\ p : d \to J \wedge d \subseteq I \wedge$ *Finite*(d)
  using *assms* **unfolding** *Fn_def* **by** *auto*

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

We finally arrive to our application of DSL.

lemma *ccc_Fn_2*: *ccc*(*Fn*(I,2), (⊇))
proof -
  {
    fix A
    assume ¬ *countable*(A)
    assume $A \subseteq$ *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 ‹$A \subseteq$ *Fn*(I, 2)›
      have $\{p \in A$. *domain*(p) $= d\} \subseteq d \to 2$
        using *domain_of_fun* **by** *fastforce*
      moreover from *True*
      have *Finite*($d \to 2$)

```
    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 (λ*p*∈*?X*. *domain*(*p*)) ∈ *surj*(*?X*, *D*)
    using *lam_type* **unfolding** *surj_def* **by** *auto*
  moreover
  assume *countable*(*?X*)
  moreover
  note ‹*uncountable*(*D*)›
  ultimately
  show *False*
    using *surj_countable* **by** *auto*
qed
moreover
have *D* = (⋃*f*∈*Pow*(*r*×*2*) . {*domain*(*p*) . *p*∈{ *x*∈*A*. *restrict*(*x*,*r*) = *f* ∧ *domain*(*x*) ∈ *D*}})
proof -
  {
    fix *z*
    assume *z* ∈ *D*
    with ‹*D* ⊆ _›
    obtain *p* **where** *domain*(*p*) = *z p* ∈ *A*
      by *auto*
    moreover from ‹*A* ⊆ *Fn*(*I*, *2*)› **and** *this*
    have *p* : *z* → *2*
      using *domain_of_fun* **by** *force*
    moreover from *this*
    have *restrict*(*p*,*r*) ⊆ *r* × *2*
      using *function_restrictI*[*of p r*] *fun_is_function*[*of p z λ_. 2*]
        restrict_subset_Sigma[*of p z λ_. 2 r*]
      by (*auto simp*:*Pi_def*)
    ultimately
    have ∃ *p*∈*A*. *restrict*(*p*,*r*) ∈ *Pow*(*r*×*2*) ∧ *domain*(*p*) = *z* **by** *auto*

}
then
show *?thesis*
by (*intro equalityI*) (*force*)+
qed
obtain *f* **where** *uncountable*({*domain*(*p*) . *p*∈{*x*∈*A*. *restrict*(*x*,*r*) = *f* ∧ *domain*(*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.