Wetzel's Problem and the Continuum Hypothesis

Lawrence C. Paulson

February 21, 2022

Abstract

Let F be a set of analytic functions on the complex plane such that, for each $z \in \mathbb{C}$, the set $\{f(z) \mid f \in F\}$ is countable; must then F itself be countable? The answer is yes if the Continuum Hypothesis is false, i.e., if the cardinality of $\mathbb{R}$ exceeds $\aleph_1$. But if CH is true then such an F, of cardinality $\aleph_1$, can be constructed by transfinite recursion.

The formal proof illustrates reasoning about complex analysis (analytic and homomorphic functions) and set theory (transfinite cardinalities) in a single setting. The mathematical text comes from *Proofs from THE BOOK* [1, pp. 137–8], by Aigner and Ziegler.

Contents

Acknowledgements The author was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178) funded by the European Research Council. Thanks also to Manuel Eberl for advice on proving a function to be holomorphic.

1 Wetzel's Problem, Solved by Erdös

Martin Aigner and Günter M. Ziegler. Proofs from THE BOOK. (Springer, 2018). Chapter 19: Sets, functions, and the continuum hypothesis Theorem 5 (pages 137–8)

theory *Wetzels-Problem* **imports**
HOL−Complex-Analysis.Complex-Analysis ZFC-in-HOL.ZFC-Typeclasses

begin

1.1 Added to the developer libraries

lemma *inj-on-restrict-iff*: $A \subseteq B \Longrightarrow$ *inj-on* (*restrict f B*) $A \longleftrightarrow$ *inj-on f A*
by (*metis inj-on-cong restrict-def subset-iff*)

lemma *Rats-closure-real*: *closure* $\mathbb{Q}$ = (*UNIV*::*real set*)
proof −
have $\bigwedge x$::*real*. *x* ∈ *closure* $\mathbb{Q}$
by (*metis closure-approachable dist-real-def rational-approximation*)
then show *?thesis* **by** *auto*
qed

lemma *fsigma-UNIV* [*iff*]: *fsigma* (*UNIV* :: *'a*::*real-inner set*)
proof −
have (*UNIV* ::*'a set*) = ($\bigcup i$. *cball 0* (*of-nat i*))
by (*auto simp*: *real-arch-simple*)
then show *?thesis*
by (*metis closed-cball fsigma.intros*)
qed

theorem *complex-non-denum*: $\nexists f$ :: *nat* ⇒ *complex*. *surj f*
by (*metis* (*full-types*) *Re-complex-of-real comp-surj real-non-denum surj-def*)

lemma *uncountable-UNIV-complex*: *uncountable* (*UNIV* :: *complex set*)
using *complex-non-denum* **unfolding** *uncountable-def* **by** *auto*

lemma *analytic-on-prod* [*analytic-intros*]:
($\bigwedge i$. *i* ∈ *I* ⟹ (*f i*) *analytic-on S*) ⟹ (λ*x*. *prod* (λ*i*. *f i x*) *I*) *analytic-on S*
by (*induct I rule*: *infinite-finite-induct*) (*auto simp*: *analytic-on-mult*)

lemma *holomorphic-countable-zeros*:
assumes *S*: *f holomorphic-on S open S connected S* **and** *fsigma S*
and ¬ *f constant-on S*
shows *countable* {*z*∈*S*. *f z* = *0*}
proof −
obtain *F*::*nat* ⇒ *complex set*
where *F*: *range F* ⊆ *Collect compact* **and** *Seq*: *S* = ($\bigcup i$. *F i*)
using ‹*fsigma S*› **by** (*meson fsigma-Union-compact*)
have *fin*: *finite* {*z* ∈ *F i*. *f z* = *0*} **for** *i*

using *holomorphic-compact-finite-zeros assms F Seq Union-iff* **by** *blast*
have $\{z \in S.\ f\ z = 0\} = (\bigcup i.\ \{z \in F\ i.\ f\ z = 0\})$
using *Seq* **by** *auto*
with *fin* **show** *?thesis*
by (*simp add*: *countable-finite*)
qed

lemma *holomorphic-countable-equal*:
assumes *f holomorphic-on S g holomorphic-on S open S connected S* **and** *fsigma S*
and *eq*: *uncountable* $\{z{\in}S.\ f\ z = g\ z\}$
shows $S \subseteq \{z{\in}S.\ f\ z = g\ z\}$
proof −
obtain *z* **where** *z*: $z{\in}S\ f\ z = g\ z$
using *eq not-finite-existsD uncountable-infinite* **by** *blast*
have $(\lambda x.\ f\ x - g\ x)$ *holomorphic-on S*
by (*simp add*: *assms holomorphic-on-diff*)
then have $(\lambda x.\ f\ x - g\ x)$ *constant-on S*
using *holomorphic-countable-zeros assms* **by** *force*
with *z* **have** $\bigwedge x.\ x{\in}S \Longrightarrow f\ x - g\ x = 0$
unfolding *constant-on-def* **by** *force*
then show *?thesis*
by *auto*
qed

lemma *holomorphic-countable-equal-UNIV*:
assumes *fg*: *f holomorphic-on UNIV g holomorphic-on UNIV*
and *eq*: *uncountable* $\{z.\ f\ z = g\ z\}$
shows $f{=}g$
using *holomorphic-countable-equal* [*OF fg*] *eq* **by** *fastforce*

lemma *finite-iff-less-Aleph0*: *finite* (*elts x*) $\longleftrightarrow$ *vcard x* $< \omega$
proof
show *finite* (*elts x*) $\Longrightarrow$ *vcard x* $< \omega$
by (*metis Card-ω Card-def finite-lesspoll-infinite infinite-ω lesspoll-imp-Card-less*)
show *vcard x* $< \omega \Longrightarrow$ *finite* (*elts x*)
by (*meson Ord-cardinal cardinal-eqpoll eqpoll-finite-iff infinite-Ord-omega less-le-not-le*)
qed

lemma *cadd-left-commute*: $j \oplus (i \oplus k) = i \oplus (j \oplus k)$
using *cadd-assoc cadd-commute* **by** *force*

lemmas *cadd-ac* = *cadd-assoc cadd-commute cadd-left-commute*

lemma *csucc-lt-csucc-iff*: $[\![Card\ \kappa'; Card\ \kappa]\!] \Longrightarrow (csucc\ \kappa' < csucc\ \kappa) = (\kappa' < \kappa)$
by (*metis Card-csucc Card-is-Ord Card-lt-csucc-iff Ord-not-less*)

lemma *csucc-le-csucc-iff*: $[\![Card\ \kappa'; Card\ \kappa]\!] \Longrightarrow (csucc\ \kappa' \leq csucc\ \kappa) = (\kappa' \leq \kappa)$
by (*metis Card-csucc Card-is-Ord Card-lt-csucc-iff Ord-not-less*)

lemma *Card-Un* [*simp*,*intro*]:
assumes *Card*(x) *Card*(y) **shows** *Card*($x \sqcup y$)
by (*metis Card-is-Ord Ord-linear-le assms sup.absorb2 sup.orderE*)

lemma *csucc-0* [*simp*]: *csucc 0 = 1*
by (*simp add*: *finite-csucc one-V-def*)

lemma *InfCard-Aleph* [*simp*, *intro*]:
assumes *Ord* α
shows *InfCard*(*Aleph* α)
unfolding *InfCard-def*
by (*metis Aleph-0 Aleph-increasing Card-Aleph antisym-conv1 assms in-succ-iff nless-le zero-in-succ*)

corollary *Aleph-csquare-eq* [*simp*]: *Ord* $\alpha \Longrightarrow \aleph\alpha \otimes \aleph\alpha = \aleph\alpha$
using *InfCard-csquare-eq* **by** *auto*

lemma *small-Times-iff*: *small* $(X \times Y) \longleftrightarrow$ *small* $X \wedge$ *small* $Y \vee X{=}\{\} \vee Y{=}\{\}$
(**is** - = *?rhs*)
proof
assume $*$: *small* $(X \times Y)$
{ have *small* $X \wedge$ *small* Y **if** $x \in X$ $y \in Y$ **for** x y
proof −
have $X \subseteq$ *fst* ‘ $(X \times Y)$ $Y \subseteq$ *snd* ‘ $(X \times Y)$
using *that* **by** *auto*
with *that* **show** *?thesis*
by (*metis* $*$ *replacement smaller-than-small*)
qed **}**
then show *?rhs*
by (*metis equals0I*)
next
assume *?rhs*
then show *small* $(X \times Y)$
by *auto*
qed

lemma *lepoll-small*:
assumes $A \lesssim B$ *small B*
shows *small A*
by (*meson assms lepoll-iff replacement smaller-than-small*)

lemma *countable-iff-vcard-less1*: *countable* (*elts x*) $\longleftrightarrow$ *vcard* $x < \aleph 1$
by (*simp add*: *countable-iff-le-Aleph0 lt-csucc-iff one-V-def*)

lemma *countable-infinite-vcard*: *countable* (*elts x*) $\wedge$ *infinite* (*elts x*) $\longleftrightarrow$ *vcard x* $= \aleph 0$
by (*metis Aleph-0 countable-iff-le-Aleph0 dual-order.refl finite-iff-less-Aleph0 less-V-def*)

lemma *vcard-set-image*: *inj-on f* (*elts x*) $\Longrightarrow$ *vcard* (*ZFC-in-HOL.set* (*f* ‘ *elts x*))
= *vcard x*
by (*simp add*: *cardinal-cong*)

definition *transrec* :: ((*V* ⇒ *'a*) ⇒ *V* ⇒ *'a*) ⇒ *V* ⇒ *'a*
where *transrec H a* ≡ *wfrec* {(*x*,*y*). *x* ∈ *elts y*} *H a*

lemma *transrec*: *transrec H a* = *H* (λ*x* ∈ *elts a*. *transrec H x*) *a*
proof −
have (*cut* (*wfrec* {(*x*, *y*). *x* ∈ *elts y*} *H*) {(*x*, *y*). *x* ∈ *elts y*} *a*)
= (λ*x*∈*elts a*. *wfrec* {(*x*, *y*). *x* ∈ *elts y*} *H x*)
by (*force simp*: *cut-def*)
then show *?thesis*
unfolding *transrec-def*
by (*simp add*: *foundation wfrec*)
qed

lemma *less-succ-self*: *x* < *succ x*
by (*simp add*: *less-eq-V-def order-neq-le-trans subset-insertI*)

lemma *subset-smaller-vcard*:
assumes κ ≤ *vcard x Card* κ
obtains *y* **where** *y* ≤ *x vcard y* = κ
proof −
obtain φ **where** φ: *bij-betw* φ (*elts* (*vcard x*)) (*elts x*)
using *cardinal-eqpoll eqpoll-def* **by** *blast*
show *thesis*
proof
let *?y* = *ZFC-in-HOL.set* (φ ‘ *elts* κ)
show *?y* ≤ *x*
by (*meson* φ *assms bij-betwE set-image-le-iff small-elts vsubsetD*)
show *vcard ?y* = κ
by (*metis vcard-set-image Card-def assms bij-betw-def bij-betw-subset* φ
less-eq-V-def)
qed
qed

lemma *vcard-sup*: *vcard* (*x* ⊔ *y*) ≤ *vcard x* ⊕ *vcard y*
proof −
have *elts* (*x* ⊔ *y*) ≲ *elts* (*x* ⨄ *y*)
unfolding *lepoll-def*
proof (*intro exI conjI*)
let *?f* = λ*z*. *if z* ∈ *elts x then Inl z else Inr z*
show *inj-on ?f* (*elts* (*x* ⊔ *y*))
by (*simp add*: *inj-on-def*)
show *?f* ‘ *elts* (*x* ⊔ *y*) ⊆ *elts* (*x* ⨄ *y*)
by *force*

qed
then show *?thesis*
using *cadd-ac*
by (*metis cadd-def cardinal-cong cardinal-idem lepoll-imp-Card-le vsum-0-eqpoll*)
qed

lemma *elts-cmult*: *elts* $(\kappa' \otimes \kappa) \approx$ *elts* $\kappa' \times$ *elts* κ
by (*simp add*: *cmult-def elts-vcard-VSigma-eqpoll*)

lemma *vcard-Sup-le-cmult*:
assumes *small* U **and** κ: $\bigwedge x.\ x \in U \Longrightarrow$ *vcard* $x \leq \kappa$
shows *vcard* $(\bigsqcup U) \leq$ *vcard* (*set* U) $\otimes \kappa$
proof −
have $\exists f.\ f \in$ *elts* $x \to$ *elts* $\kappa \wedge$ *inj-on* f (*elts* x) **if** $x \in U$ **for** x
using κ [*OF that*] **by** (*metis cardinal-le-lepoll image-subset-iff-funcset lepoll-def*)
then obtain φ **where** φ: $\bigwedge x.\ x \in U \Longrightarrow (\varphi\ x) \in$ *elts* $x \to$ *elts* $\kappa \wedge$ *inj-on* $(\varphi\ x)$ (*elts* x)
by *metis*
define u **where** $u \equiv \lambda y.\ @x.\ x \in U \wedge y \in$ *elts* x
have u: $u\ y \in U \wedge y \in$ *elts* $(u\ y)$ **if** $y \in \bigcup$(*elts* ‘ U) **for** y
unfolding *u-def* **by** (*metis* (*mono-tags*, *lifting*)*that someI2-ex UN-iff*)
define ψ **where** $\psi \equiv \lambda y.\ (u\ y, \varphi\ (u\ y)\ y)$
have U: *elts* (*vcard* (*set* U)) $\approx U$
by (*metis* ‹*small* U› *cardinal-eqpoll elts-of-set*)
have *elts* $(\bigsqcup U) = \bigcup$(*elts* ‘ U)
using ‹*small* U› **by** *blast*
also have $\ldots \lesssim U \times$ *elts* κ
unfolding *lepoll-def*
proof (*intro exI conjI*)
show *inj-on* ψ $(\bigcup$ (*elts* ‘ U))
using φ u **by** (*smt* (*verit*) ψ*-def inj-on-def prod.inject*)
show ψ ‘ $\bigcup$ (*elts* ‘ U) $\subseteq U \times$ *elts* κ
using φ u **by** (*auto simp*: ψ*-def*)
qed
also have $\ldots \approx$ *elts* (*vcard* (*set* U) $\otimes \kappa$)
using U *elts-cmult eqpoll-sym eqpoll-trans times-eqpoll-cong* **by** *blast*
finally have *elts* $(\bigsqcup U) \lesssim$ *elts* (*vcard* (*set* U) $\otimes \kappa$) **.**
then show *?thesis*
by (*simp add*: *cmult-def lepoll-cardinal-le*)
qed

lemma *csucc-le-Card-iff*: $[\![$*Card* κ'; *Card* $\kappa]\!] \Longrightarrow$ *csucc* $\kappa' \leq \kappa \longleftrightarrow \kappa' < \kappa$
by (*metis Card-csucc Card-is-Ord Card-lt-csucc-iff Ord-not-le*)

lemma *cadd-InfCard-le*:
assumes $\alpha \leq \kappa$ $\beta \leq \kappa$ *InfCard* κ
shows $\alpha \oplus \beta \leq \kappa$
using *assms* **by** (*metis InfCard-cdouble-eq cadd-le-mono*)

lemma *cmult-InfCard-le*:
assumes $\alpha \leq \kappa$ $\beta \leq \kappa$ *InfCard* κ
shows $\alpha \otimes \beta \leq \kappa$
using *assms*
by (*metis InfCard-csquare-eq cmult-le-mono*)

lemma *vcard-Aleph* [*simp*]: *Ord* $\alpha \Longrightarrow$ *vcard* $(\aleph\alpha) = \aleph\alpha$
by (*simp add*: *Card-cardinal-eq*)

lemma *omega-le-Aleph* [*simp*]: *Ord* $\alpha \Longrightarrow \omega \leq \aleph\alpha$
using *InfCard-def* **by** *auto*

1.2 Making the embedding explicit

definition *V-of* :: $'a$::*embeddable* $\Rightarrow$ *V*
where *V-of* $\equiv$ *SOME f*. *inj f*

lemma *inj-V-of*: *inj V-of*
unfolding *V-of-def* **by** (*metis embeddable-class.ex-inj some-eq-imp*)

declare *inv-f-f* [*OF inj-V-of*, *simp*]

lemma *inv-V-of-image-eq* [*simp*]: *inv V-of* ‘ (*V-of* ‘ *X*) = *X*
by (*auto simp*: *image-comp*)

lemma *infinite-V-of*: *infinite* (*UNIV*::$'a$ *set*) $\Longrightarrow$ *infinite* (*range* (*V-of*::$'a$::*embeddable*$\Rightarrow$ *V*))
using *finite-imageD inj-V-of* **by** *blast*

lemma *countable-V-of*: *countable* (*range* (*V-of*::$'a$::*countable*$\Rightarrow$ *V*))
by *blast*

lemma *elts-set-V-of*: *small X* $\Longrightarrow$ *elts* (*ZFC-in-HOL.set* (*V-of* ‘ *X*)) $\approx$ *X*
by (*metis inj-V-of inj-eq inj-on-def inj-on-image-eqpoll-self replacement set-of-elts small-iff*)

lemma *V-of-image-times*: *V-of* ‘ (*X* × *Y*) $\approx$ (*V-of* ‘ *X*) × (*V-of* ‘ *Y*)
proof −
have *V-of* ‘ (*X* × *Y*) $\approx$ *X* × *Y*
by (*meson inj-V-of inj-eq inj-onI inj-on-image-eqpoll-self*)
also have ... $\approx$ (*V-of* ‘ *X*) × (*V-of* ‘ *Y*)
by (*metis eqpoll-sym inj-V-of inj-eq inj-onI inj-on-image-eqpoll-self times-eqpoll-cong*)
finally show *?thesis* **.**
qed

1.3 The cardinality of the continuum

definition *Real-set* $\equiv$ *ZFC-in-HOL.set* (*range* (*V-of*::*real*$\Rightarrow$ *V*))
definition *Complex-set* $\equiv$ *ZFC-in-HOL.set* (*range* (*V-of*::*complex*$\Rightarrow$ *V*))
definition *C-continuum* $\equiv$ *vcard Real-set*

lemma *V-of-Real-set*: *bij-betw V-of* (*UNIV*::*real set*) (*elts Real-set*)
by (*simp add*: *Real-set-def bij-betw-def inj-V-of*)

lemma *uncountable-Real-set*: *uncountable* (*elts Real-set*)
using *V-of-Real-set countable-iff-bij uncountable-UNIV-real* **by** *blast*

lemma *Card C-continuum*
by (*simp add*: *C-continuum-def Card-def*)

lemma *C-continuum-ge*: *C-continuum* $\geq$ $\aleph$*1*
proof −
have ¬ *C-continuum* < $\aleph$*1*
proof −
have ¬ *vcard Real-set* ≤ $\aleph$*0*
using *countable-iff-le-Aleph0 uncountable-Real-set* **by** *presburger*
then show *?thesis*
by (*simp add*: *C-continuum-def lt-csucc-iff one-V-def*)
qed
then show *?thesis*
by (*simp add*: *C-continuum-def Ord-not-less*)
qed

lemma *V-of-Complex-set*: *bij-betw V-of* (*UNIV*::*complex set*) (*elts Complex-set*)
by (*simp add*: *Complex-set-def bij-betw-def inj-V-of*)

lemma *uncountable-Complex-set*: *uncountable* (*elts Complex-set*)
using *V-of-Complex-set countableI-bij2 uncountable-UNIV-complex* **by** *blast*

lemma *Complex-vcard*: *vcard Complex-set* = *C-continuum*
proof −
define *emb1* **where** *emb1* ≡ *V-of o complex-of-real o inv V-of*
have *elts Real-set* ≈ *elts Complex-set*
proof (*rule lepoll-antisym*)
show *elts Real-set* $\lesssim$ *elts Complex-set*
unfolding *lepoll-def*
proof (*intro conjI exI*)
show *inj-on emb1* (*elts Real-set*)
unfolding *emb1-def Real-set-def*
by (*simp add*: *inj-V-of inj-compose inj-of-real inj-on-imageI*)
show *emb1* ‘ *elts Real-set* ⊆ *elts Complex-set*
by (*simp add*: *emb1-def Real-set-def Complex-set-def image-subset-iff*)
qed
define *emb2* **where** *emb2* ≡ (λ*z*. (*V-of* (*Re z*), *V-of* (*Im z*))) *o inv V-of*
have *elts Complex-set* $\lesssim$ *elts Real-set* × *elts Real-set*
unfolding *lepoll-def*
proof (*intro conjI exI*)
show *inj-on emb2* (*elts Complex-set*)

unfolding *emb2-def Complex-set-def inj-on-def*
by (*simp add*: *complex.expand inj-V-of inj-eq*)
show *emb2* ‘ *elts Complex-set* ⊆ *elts Real-set* × *elts Real-set*
by (*simp add*: *emb2-def Real-set-def Complex-set-def image-subset-iff*)
qed
also have . . . ≈ *elts Real-set*
by (*simp add*: *infinite-times-eqpoll-self uncountable-Real-set uncountable-infinite*)
finally show *elts Complex-set* ≲ *elts Real-set* **.**
qed
then show *?thesis*
by (*simp add*: *C-continuum-def cardinal-cong*)
qed

1.4 Cardinality of an arbitrary HOL set

definition *gcard* :: *'a*::*embeddable set* ⇒ *V*
where *gcard X* ≡ *vcard* (*ZFC-in-HOL.set* (*V-of* ‘ *X*))

lemma *gcard-big-0*: ¬ *small X* ⟹ *gcard X* = *0*
by (*metis elts-eq-empty-iff elts-of-set gcard-def inv-V-of-image-eq replacement vcard-0*)

lemma *gcard-empty-0* [*simp*]: *gcard* {} = *0*
by (*metis gcard-def image-is-empty vcard-0 zero-V-def*)

lemma *gcard-single-1* [*simp*]: *gcard* {*x*} = *1*
by (*simp add*: *gcard-def*)

lemma *gcard-finite-set*: ⟦*finite X*; *a* ∉ *X*⟧ ⟹ *gcard* (*insert a X*) = *succ* (*gcard X*)
by (*simp add*: *gcard-def inj-V-of inj-image-mem-iff finite-csucc vcard-finite-set*)

lemma *gcard-eq-card*: *finite X* ⟹ *gcard X* = *ord-of-nat* (*card X*)
by (*induction X rule*: *finite-induct*) (*auto simp add*: *gcard-finite-set*)

lemma *Card-gcard* [*iff*]: *Card* (*gcard X*)
by (*simp add*: *Card-def gcard-def*)

lemma *gcard-eq-vcard* [*simp*]: *gcard* (*elts x*) = *vcard x*
by (*metis cardinal-cong elts-set-V-of gcard-def small-elts*)

lemma *gcard-eqpoll*: *small X* ⟹ *elts* (*gcard X*) ≈ *X*
by (*metis cardinal-eqpoll elts-set-V-of eqpoll-trans gcard-def*)

lemma *gcard-image-le*:
assumes *small A*
shows *gcard* (*f* ‘ *A*) ≤ *gcard A*
proof −
have (*V-of* ‘ *f* ‘ *A*) ≲ (*V-of* ‘ *A*)

by (*metis image-lepoll inv-V-of-image-eq lepoll-trans*)
then show *?thesis*
by (*simp add*: *assms gcard-def lepoll-imp-Card-le*)
qed

lemma *gcard-image*: *inj-on f A* $\Longrightarrow$ *gcard* (*f* ‘ *A*) = *gcard A*
by (*metis dual-order.antisym gcard-big-0 gcard-image-le small-image-iff the-inv-into-onto*)

lemma *lepoll-imp-gcard-le*:
assumes $A \lesssim B$ *small B*
shows *gcard A* $\leq$ *gcard B*
proof −
have *elts* (*ZFC-in-HOL.set* (*V-of* ‘ *A*)) $\approx$ *A elts* (*ZFC-in-HOL.set* (*V-of* ‘ *B*)) $\approx$ *B*
by (*meson assms elts-set-V-of lepoll-small*)+
with ‹$A \lesssim B$› **show** *?thesis*
unfolding *gcard-def*
by (*meson lepoll-imp-Card-le eqpoll-sym lepoll-iff-leqpoll lepoll-trans*)
qed

lemma *subset-imp-gcard-le*:
assumes $A \subseteq B$ *small B*
shows *gcard A* $\leq$ *gcard B*
by (*simp add*: *assms lepoll-imp-gcard-le subset-imp-lepoll*)

lemma *gcard-le-lepoll*: ⟦*gcard A* $\leq \alpha$; *small A*⟧ $\Longrightarrow$ $A \lesssim$ *elts* α
by (*meson eqpoll-sym gcard-eqpoll lepoll-trans1 less-eq-V-def subset-imp-lepoll*)

lemma *gcard-Union-le-cmult*:
assumes *small U* **and** κ: $\bigwedge x.\ x \in U \Longrightarrow$ *gcard x* $\leq \kappa$ **and** *sm*: $\bigwedge x.\ x \in U \Longrightarrow$ *small x*
shows *gcard* ($\bigcup U$) $\leq$ *gcard U* $\otimes \kappa$
proof −
have $\exists f.\ f \in x \to$ *elts* $\kappa \wedge$ *inj-on f x* **if** $x \in U$ **for** *x*
using κ [*OF that*] *gcard-le-lepoll* **by** (*smt* (*verit*) *Pi-iff TC-small imageI lepoll-def subset-eq*)
then obtain φ **where** φ: $\bigwedge x.\ x \in U \Longrightarrow (\varphi\ x) \in x \to$ *elts* $\kappa \wedge$ *inj-on* ($\varphi\ x$) *x*
by *metis*
define *u* **where** $u \equiv \lambda y.\ @x.\ x \in U \wedge y \in x$
have *u*: *u y* $\in U \wedge y \in$ (*u y*) **if** $y \in \bigcup(\ U)$ **for** *y*
unfolding *u-def* **using** *that* **by** (*fast intro*!: *someI2*)
define ψ **where** $\psi \equiv \lambda y.\ (u\ y, \varphi\ (u\ y)\ y)$
have *U*: *elts* (*gcard U*) $\approx$ *U*
by (*simp add*: *gcard-eqpoll*)
have $\bigcup U \lesssim U \times$ *elts* κ
unfolding *lepoll-def*
proof (*intro exI conjI*)
show *inj-on* ψ ($\bigcup\ U$)
using φ *u* **by** (*smt* (*verit*) ψ*-def inj-on-def prod.inject*)

show ψ '$\bigcup$ $U \subseteq U \times$ *elts* κ
using φ u **by** (*auto simp*: ψ-*def*)
qed
also have ... $\approx$ *elts* (*gcard* $U \otimes \kappa$)
using *U elts-cmult eqpoll-sym eqpoll-trans times-eqpoll-cong* **by** *blast*
finally have $(\bigcup U) \lesssim$ *elts* (*gcard* $U \otimes \kappa$) **.**
then show *?thesis*
by (*metis cardinal-idem cmult-def gcard-eq-vcard lepoll-imp-gcard-le small-elts*)
qed

lemma *countable-iff-g-le-Aleph0*: *small* $X \Longrightarrow$ *countable* $X \longleftrightarrow$ *gcard* $X \leq \aleph 0$
unfolding *gcard-def*
by (*metis countable-iff-le-Aleph0 countable-image elts-of-set inv-V-of-image-eq replacement*)

lemma *countable-imp-g-le-Aleph0*: *countable* $X \Longrightarrow$ *gcard* $X \leq \aleph 0$
by (*meson countable countable-iff-g-le-Aleph0*)

lemma *finite-iff-g-le-Aleph0*: *small* $X \Longrightarrow$ *finite* $X \longleftrightarrow$ *gcard* $X < \aleph 0$
by (*metis Aleph-0 elts-set-V-of eqpoll-finite-iff finite-iff-less-Aleph0 gcard-def*)

lemma *finite-imp-g-le-Aleph0*: *finite* $X \Longrightarrow$ *gcard* $X < \aleph 0$
by (*meson finite-iff-g-le-Aleph0 finite-imp-small*)

lemma *countable-infinite-gcard*: *countable* $X \wedge$ *infinite* $X \longleftrightarrow$ *gcard* $X = \aleph 0$
proof −
have *gcard* $X = \omega$
if *countable* X **and** *infinite* X
using *that countable countable-imp-g-le-Aleph0 finite-iff-g-le-Aleph0 less-V-def*
by *auto*
moreover have *countable* X **if** *gcard* $X = \omega$
by (*metis Aleph-0 countable-iff-g-le-Aleph0 dual-order.refl gcard-big-0 omega-nonzero that*)
moreover have *False* **if** *gcard* $X = \omega$ **and** *finite* X
by (*metis Aleph-0 dual-order.irrefl finite-iff-g-le-Aleph0 finite-imp-small that*)
ultimately show *?thesis*
by *auto*
qed

lemma *uncountable-gcard*: *small* $X \Longrightarrow$ *uncountable* $X \longleftrightarrow$ *gcard* $X > \aleph 0$
by (*simp add*: *Ord-not-le countable-iff-g-le-Aleph0 gcard-def*)

lemma *uncountable-gcard-ge*: *small* $X \Longrightarrow$ *uncountable* $X \longleftrightarrow$ *gcard* $X \geq \aleph 1$
by (*simp add*: *uncountable-gcard csucc-le-Card-iff one-V-def*)

lemma *subset-smaller-gcard*:
assumes κ: $\kappa \leq$ *gcard* X *Card* κ
obtains Y **where** $Y \subseteq X$ *gcard* $Y = \kappa$
proof (*cases small* X)

case *True*
with *subset-smaller-vcard* [*OF* κ [*unfolded gcard-def*]] **show** *?thesis*
by (*metis elts-of-set gcard-def less-eq-V-def replacement set-of-elts subset-imageE that*)
next
case *False*
with *assms* **show** *?thesis*
by (*metis antisym gcard-big-0 le-0 order-refl that*)
qed

lemma *Real-gcard*: *gcard* (*UNIV*::*real set*) = *C-continuum*
by (*metis C-continuum-def Real-set-def gcard-def*)

lemma *Complex-gcard*: *gcard* (*UNIV*::*complex set*) = *C-continuum*
by (*metis Complex-set-def Complex-vcard gcard-def*)

lemma *gcard-Times* [*simp*]: *gcard* (*X* × *Y*) = *gcard X* ⊗ *gcard Y*
proof (*cases small X* ∧ *small Y*)
case *True*
have *V-of* ‘ (*X* × *Y*) ≈ (*V-of* ‘ *X*) × (*V-of* ‘ *Y*)
by (*metis V-of-image-times*)
also have ... ≈ *elts* (*vcard* (*ZFC-in-HOL.set* (*V-of* ‘ *X*))) × *elts* (*vcard* (*ZFC-in-HOL.set* (*V-of* ‘ *Y*)))
by (*metis True cardinal-eqpoll eqpoll-sym replacement set-of-elts small-iff times-eqpoll-cong*)
also have ... ≈ *elts* (*vtimes* (*vcard* (*ZFC-in-HOL.set* (*V-of* ‘ *X*))) (*vcard* (*ZFC-in-HOL.set* (*V-of* ‘ *Y*))))
using *elts-VSigma* **by** *auto*
finally show *?thesis*
using *True cardinal-cong* **by** (*simp add*: *gcard-def cmult-def*)
next
case *False*
have *gcard* (*X* × *Y*) = *0*
by (*metis False Times-empty gcard-big-0 gcard-empty-0 small-Times-iff*)
then show *?thesis*
by (*metis False cmult-0 cmult-commute gcard-big-0*)
qed

1.5 Wetzel's problem

definition *Wetzel* :: (*complex* ⇒ *complex*) *set* ⇒ *bool*
where *Wetzel* ≡ λ*F*. (∀ *f*∈*F*. *f analytic-on UNIV*) ∧ (∀ *z*. *countable*((λ*f*. *f z*) ‘ *F*))

1.5.1 When the continuum hypothesis is false

proposition *Erdos-Wetzel-nonCH*:
assumes *W*: *Wetzel F* **and** *NCH*: *C-continuum* > ℵ*1* **and** *small F*
shows *countable F*
proof −

have $\exists z0.\ gcard\ ((\lambda f.\ f\ z0)\ `\ F) \geq \aleph 1$ **if** *uncountable F*
proof −
have *gcard F* $\geq \aleph 1$
using ‹*small F*› *that uncountable-gcard-ge* **by** *blast*
then obtain F' **where** $F' \subseteq F$ **and** F': *gcard* $F' = \aleph 1$
by (*meson Card-Aleph Ord-1 subset-smaller-gcard* ‹*small F*›)
then obtain φ **where** φ: *bij-betw* φ (*elts* ($\aleph 1$)) F'
by (*metis TC-small eqpoll-def gcard-eqpoll*)
define S **where** $S \equiv \lambda\alpha\ \beta.\ \{z.\ \varphi\ \alpha\ z = \varphi\ \beta\ z\}$
have *co-S*: *gcard* ($S\ \alpha\ \beta$) $\leq \aleph 0$ **if** $\alpha \in$ *elts* β $\beta \in$ *elts* ($\aleph 1$) **for** $\alpha\ \beta$
proof −
have $\varphi\ \alpha$ *holomorphic-on UNIV* $\varphi\ \beta$ *holomorphic-on UNIV*
using W ‹$F' \subseteq F$› **unfolding** *Wetzel-def*
by (*meson Ord-ω1 Ord-trans* φ *analytic-imp-holomorphic bij-betwE subsetD that*)+
moreover have $\varphi\ \alpha \neq \varphi\ \beta$
by (*metis Ord-ω1 Ord-in-Ord Ord-trans OrdmemD* φ *bij-betw-imp-inj-on inj-on-def less-V-def that*)
ultimately have *countable* ($S\ \alpha\ \beta$)
using *holomorphic-countable-equal-UNIV* **unfolding** *S-def* **by** *blast*
then show *?thesis*
using *countable-imp-g-le-Aleph0* **by** *blast*
qed
define *SS* **where** $SS \equiv \bigsqcup((\lambda\beta.\ \bigsqcup((\lambda\alpha.\ S\ \alpha\ \beta)\ `\ elts\ \beta))\ `\ elts(\aleph 1))$

have *F'-eq*: $F' =\ \varphi\ `\ elts\ \omega 1$
using φ *bij-betw-imp-surj-on* **by** *auto*
have §: $\bigwedge x\ xa.\ xa \in elts\ \omega 1 \Longrightarrow gcard\ (\bigcup \alpha \in elts\ xa.\ S\ \alpha\ xa) \leq \omega$
by (*metis Aleph-0 TC-small co-S countable-UN countable-iff-g-le-Aleph0 less-ω1-imp-countable*)
have *gcard SS* $\leq$ *gcard* $((\lambda\beta.\ \bigcup \alpha \in elts\ \beta.\ S\ \alpha\ \beta)\ `\ elts\ \omega 1) \otimes \aleph 0$
apply (*simp add*: *SS-def*)
by (*metis* (*no-types*, *lifting*) § *TC-small gcard-Union-le-cmult imageE*)
also have $\ldots\ \leq \aleph 1$
proof (*rule cmult-InfCard-le*)
show *gcard* $((\lambda\beta.\ \bigcup \alpha \in elts\ \beta.\ S\ \alpha\ \beta)\ `\ elts\ \omega 1) \leq \omega 1$
using *gcard-image-le* **by** *fastforce*
qed *auto*
finally have *gcard SS* $\leq \aleph 1$ **.**
with *NCH* **obtain** *z0* **where** $z0 \notin SS$
by (*metis Complex-gcard UNIV-eq-I less-le-not-le*)
then have *inj-on* ($\lambda x.\ \varphi\ x\ z0$) (*elts* $\omega 1$)
apply (*simp add*: *SS-def S-def inj-on-def*)
by (*metis Ord-ω1 Ord-in-Ord Ord-linear*)
then have *gcard* $((\lambda f.\ f\ z0)\ `\ F') = \aleph 1$
by (*smt* (*verit*) F' *F'-eq gcard-image imageE inj-on-def*)
then show *?thesis*
by (*metis TC-small* ‹$F' \subseteq F$› *image-mono subset-imp-gcard-le*)
qed

with *W* **show** *?thesis*
unfolding *Wetzel-def* **by** (*meson countable uncountable-gcard-ge*)
qed

1.5.2 When the continuum hypothesis is true

lemma *Rats-closure-real2*: *closure* ($\mathbb{Q}$×$\mathbb{Q}$) = (*UNIV*::*real set*)×(*UNIV*::*real set*)
by (*simp add*: *Rats-closure-real closure-Times*)

proposition *Erdos-Wetzel-CH*:
assumes *CH*: *C-continuum* = $\aleph$*1*
obtains *F* **where** *Wetzel F* **and** *uncountable F*
proof −
define *D* **where** *D* ≡ {*z*. *Re z* ∈ $\mathbb{Q}$ ∧ *Im z* ∈ $\mathbb{Q}$}
have *Deq*: *D* = ($\bigcup$ *x*∈$\mathbb{Q}$. $\bigcup$ *y*∈$\mathbb{Q}$. {*Complex x y*})
using *complex.collapse* **by** (*force simp*: *D-def*)
with *countable-rat* **have** *countable D*
by *blast*
have *infinite D*
unfolding *Deq*
by (*intro infinite-disjoint-family-imp-infinite-UNION Rats-infinite*) (*auto simp*: *disjoint-family-on-def*)
have ∃ *w*. *Re w* ∈ $\mathbb{Q}$ ∧ *Im w* ∈ $\mathbb{Q}$ ∧ *norm* (*w* − *z*) < *e* **if** *e* > *0* **for** *z* **and** *e*::*real*
proof −
obtain *x y* **where** *x*∈$\mathbb{Q}$ *y*∈$\mathbb{Q}$ **and** *xy*: *dist* (*x*,*y*) (*Re z*, *Im z*) < *e*
using ‹*e* > *0*› *Rats-closure-real2* **by** (*force simp*: *closure-approachable*)
moreover have *dist* (*x*,*y*) (*Re z*, *Im z*) = *norm* (*Complex x y* − *z*)
by (*simp add*: *norm-complex-def norm-prod-def dist-norm*)
ultimately show ∃ *w*. *Re w* ∈ $\mathbb{Q}$ ∧ *Im w* ∈ $\mathbb{Q}$ ∧ *norm* (*w* − *z*) < *e*
by (*metis complex.sel*)
qed
then have *cloD*: *closure D* = *UNIV*
by (*auto simp*: *D-def closure-approachable dist-complex-def*)
obtain ζ **where** ζ: *bij-betw* ζ (*elts* ($\aleph$*1*)) (*UNIV*::*complex set*)
by (*metis Complex-gcard TC-small assms eqpoll-def gcard-eqpoll*)
define *inD* **where** *inD* ≡ λβ *f*. (∀ α ∈ *elts* β. *f* (ζ α) ∈ *D*)
define Φ **where** Φ ≡ λβ *f*. *f* β *analytic-on UNIV* ∧ *inD* β (*f* β) ∧ *inj-on f* (*elts* (*succ* β))
have ∗: ∃ *h*. Φ γ ((*restrict f* (*elts* γ))(γ:=*h*))
if γ: γ ∈ *elts* ($\aleph$*1*) **and** ∀ β ∈ *elts* γ. Φ β *f* **for** γ *f*
proof −
have *f*: ∀ β ∈ *elts* γ. *f* β *analytic-on UNIV* ∧ *inD* β (*f* β)
using *that* **by** (*auto simp*: Φ-*def*)
have *inj*: *inj-on f* (*elts* γ)
using *that* **by** (*simp add*: Φ-*def inj-on-def*) (*meson Ord-ω1 Ord-in-Ord Ord-linear*)
obtain *h* **where** *h analytic-on UNIV inD* γ *h* (∀ β ∈ *elts* γ. *h* ≠ *f* β)
proof (*cases finite* (*elts* γ))
case *True*

```
    then obtain η where η: bij-betw η {..<card (elts γ)} (elts γ)
      using bij-betw-from-nat-into-finite by blast
    define g where g ≡ f o η
    define w where w ≡ ζ o η
    have gf: ∀ i<card (elts γ). h ≠ g i ⟹ ∀ β∈elts γ. h ≠ f β for h
      using η by (auto simp: bij-betw-iff-bijections g-def)
    have ∗: ∃ h. h analytic-on UNIV ∧ (∀ i<n. h (w i) ∈ D ∧ h (w i) ≠ g i (w
i))
      if n ≤ card (elts γ) for n
      using that
    proof (induction n)
      case 0
      then show ?case
        using analytic-on-const by blast
    next
      case (Suc n)
      then obtain h where h analytic-on UNIV and hg: ∀ i<n. h (w i) ∈ D ∧
h (w i) ≠ g i (w i)
        using Suc-leD by blast
      define p where p ≡ λz. ∏ i<n. z − w i
      have p0: p z = 0 ⟷ (∃ i<n. z = w i) for z
        unfolding p-def by force
      obtain d where d: d ∈ D − {g n (w n)}
        using ‹infinite D› by (metis ex-in-conv finite.emptyI infinite-remove)
      define h′ where h′ ≡ λz. h z + p z ∗ (d − h (w n)) / p (w n)
      have h′-eq: h′ (w i) = h (w i) if i<n for i
        using that by (force simp: h′-def p0)
      show ?case
      proof (intro exI strip conjI)
        have nless: n < card (elts γ)
          using Suc.prems Suc-le-eq by blast
        with η have η n ≠ η i if i<n for i
          using that unfolding bij-betw-iff-bijections
          by (metis lessThan-iff less-not-refl order-less-trans)
        with ζ η γ have pwn-nonzero: p (w n) ≠ 0
          apply (clarsimp simp: p0 w-def bij-betw-iff-bijections)
          by (metis Ord-ω1 Ord-trans nless lessThan-iff order-less-trans)
        then show h′ analytic-on UNIV
          unfolding h′-def p-def by (intro analytic-intros ‹h analytic-on UNIV›)
        fix i
        assume i < Suc n
        then have §: i < n ∨ i = n
          by linarith
        then show h′ (w i) ∈ D
          using h′-eq hg d h′-def pwn-nonzero by force
        show h′ (w i) ≠ g i (w i)
          using § h′-eq hg h′-def d pwn-nonzero by fastforce
      qed
    qed
```

show *?thesis*
using ∗ [*OF order-refl*] *η that gf*
by (*simp add*: *w-def bij-betw-iff-bijections inD-def*) *metis*
next
case *False*
then obtain *η* **where** *η*: *bij-betw η* (*UNIV*::*nat set*) (*elts γ*)
by (*meson γ countable-infiniteE′ less-ω1-imp-countable*)
then have *η-inject* [*simp*]: *η i* = *η j* ⟷ *i*=*j* **for** *i j*
by (*simp add*: *bij-betw-imp-inj-on inj-eq*)
define *g* **where** *g* ≡ *f o η*
define *w* **where** *w* ≡ *ζ o η*
then have *w-inject* [*simp*]: *w i* = *w j* ⟷ *i*=*j* **for** *i j*
by (*smt* (*verit*) *Ord-ω1 Ord-trans UNIV-I η γ ζ bij-betw-iff-bijections comp-apply*)
define *p* **where** *p* ≡ *λn z*. ∏ *i*<*n*. *z* − *w i*
define *q* **where** *q* ≡ *λn*. ∏ *i*<*n*. *1* + *norm* (*w i*)
define *h* **where** *h* ≡ *λn ε z*. ∑ *i*<*n*. *ε i* ∗ *p i z*
define *BALL* **where** *BALL* ≡ *λn ε*. *ball* (*h n ε* (*w n*)) (*norm* (*p n* (*w n*)) / (*fact n* ∗ *q n*))
— The demonimator above is the key to keeping the epsilons small
define *DD* **where** *DD* ≡ *λn ε*. *D* ∩ *BALL n ε* − {*g n* (*w n*)}
define *dd* **where** *dd* ≡ *λn ε*. *SOME x*. *x* ∈ *DD n ε*
have *p0*: *p n z* = *0* ⟷ (∃ *i*<*n*. *z* = *w i*) **for** *z n*
unfolding *p-def* **by** *force*
have [*simp*]: *p n* (*w i*) = *0* **if** *i*<*n* **for** *i n*
using *that* **by** (*simp add*: *p0*)
have *q-gt0*: *0* < *q n* **for** *n*
unfolding *q-def* **by** (*smt* (*verit*) *norm-not-less-zero prod-pos*)
have *DD n ε* ≠ {} **for** *n ε*
proof −
have *r* > *0* ⟹ *infinite* (*D* ∩ *ball z r*) **for** *z r*
by (*metis islimpt-UNIV limpt-of-closure islimpt-eq-infinite-ball cloD*)
then have *infinite* (*D* ∩ *BALL n ε*) **for** *n ε*
by (*simp add*: *BALL-def p0 q-gt0*)
then show *?thesis*
by (*metis DD-def finite.emptyI infinite-remove*)
qed
then have *dd-in-DD*: *dd n ε* ∈ *DD n ε* **for** *n ε*
by (*simp add*: *dd-def some-in-eq*)

have *h-cong*: *h n ε* = *h n ε′* **if** ⋀*i*. *i*<*n* ⟹ *ε i* = *ε′ i* **for** *n ε ε′*
using *that* **by** (*simp add*: *h-def*)
have *dd-cong*: *dd n ε* = *dd n ε′* **if** ⋀*i*. *i*<*n* ⟹ *ε i* = *ε′ i* **for** *n ε ε′*
using *that* **by** (*metis dd-def DD-def BALL-def h-cong*)

have [*simp*]: *h n* (*cut ε less-than n*) = *h n ε* **for** *n ε*
by (*meson cut-apply h-cong less-than-iff*)
have [*simp*]: *dd n* (*cut ε less-than n*) = *dd n ε* **for** *n ε*
by (*meson cut-apply dd-cong less-than-iff*)

define *coeff* **where** *coeff* ≡ *wfrec less-than* (λε *n*. (*dd n* ε − *h n* ε (*w n*)) / *p n* (*w n*))
have *coeff-eq*: *coeff n* = (*dd n coeff* − *h n coeff* (*w n*)) / *p n* (*w n*) **for** *n*
by (*simp add*: *def-wfrec* [*OF coeff-def*])

have *norm-coeff*: *norm* (*coeff n*) < *1* / (*fact n* ∗ *q n*) **for** *n*
using *dd-in-DD* [*of n coeff*]
by (*simp add*: *q-gt0 coeff-eq DD-def BALL-def dist-norm norm-minus-commute norm-divide divide-simps*)
have *h-truncated*: *h n coeff* (*w k*) = *h* (*Suc k*) *coeff* (*w k*) **if** *k* < *n* **for** *n k*
proof −
have (∑ *i*<*n*. *coeff i* ∗ *p i* (*w k*)) = (∑ *i*<*Suc k*. *coeff i* ∗ *p i* (*w k*)) + (∑ *i*=*Suc k*..<*n*. *coeff i* ∗ *p i* (*w k*))
by (*smt* (*verit*) *Suc-le-eq atLeast0LessThan le0 sum.atLeastLessThan-concat that*)
also have ... = (∑ *i*<*Suc k*. *coeff i* ∗ *p i* (*w k*))
by *simp*
finally show *?thesis*
by (*simp add*: *h-def*)
qed
have *norm-p-bound*: *norm* (*p n z'*) ≤ *q n* ∗ (*1* + *norm z*) ^ *n*
if *dist z z'* ≤ *1* **for** *n z z'*
proof (*induction n*)
case *0*
then show *?case*
by (*auto simp*: *p-def q-def*)
next
case (*Suc n*)
have *norm z'* − *norm z* ≤ *1*
by (*smt* (*verit*) *dist-norm norm-triangle-ineq3 that*)
then have §: *norm* (*z'* − *w n*) ≤ (*1* + *norm* (*w n*)) ∗ (*1* + *norm z*)
by (*simp add*: *mult.commute add-mono distrib-left norm-triangle-le-diff*)
have *norm* (*p n z'*) ∗ *norm* (*z'* − *w n*) ≤ (*q n* ∗ (*1* + *norm z*) ^ *n*) ∗ *norm* (*z'* − *w n*)
by (*metis Suc mult.commute mult-left-mono norm-ge-zero*)
also have ... ≤ (*q n* ∗ (*1* + *norm z*) ^ *n*) ∗ (*1* + *norm* (*w n*)) ∗ ((*1* + *norm z*))
by (*smt* (*verit*) § *Suc mult.assoc mult-left-mono norm-ge-zero*)
also have ... ≤ *q n* ∗ (*1* + *norm* (*w n*)) ∗ ((*1* + *norm z*) ∗ (*1* + *norm z*) ^ *n*)
by (*simp add*: *mult-ac*)
finally have *norm* (*p n z'*) ∗ *norm* (*z'* − *w n*) ≤ *q n* ∗ (*1* + *norm* (*w n*)) ∗ ((*1* + *norm z*) ∗ (*1* + *norm z*) ^ *n*) **.**
with *that* **show** *?case*
by (*auto simp*: *p-def q-def norm-mult simp del*: *fact-Suc*)
qed

show *?thesis*

```
    proof
      define hh where hh ≡ λz. suminf (λi. coeff i * p i z)
      have hh holomorphic-on UNIV
      proof (rule holomorphic-uniform-sequence)
        fix n  — Many thanks to Manuel Eberl for suggesting these approach
        show h n coeff holomorphic-on UNIV
          unfolding h-def p-def by (intro holomorphic-intros)
      next
        fix z
        have uniform-limit (cball z 1) (λn. h n coeff) hh sequentially
          unfolding hh-def h-def
        proof (rule Weierstrass-m-test)
          let ?M = λn. (1 + norm z) ^ n / fact n
          have ∃ N. ∀ n≥N. B ≤ (1 + real n) / (1 + norm z) for B
          proof
            show ∀ n≥nat ⌈B * (1 + norm z)⌉. B ≤ (1 + real n) / (1 + norm z)
                 using norm-ge-zero [of z] by (auto simp: divide-simps simp del:
norm-ge-zero)
          qed
          then have L: liminf (λn. ereal ((1 + real n) / (1 + norm z))) = ∞
            by (simp add: Lim-PInfty flip: liminf-PInfty)
          have ∀ F n in sequentially. 0 < (1 + cmod z) ^ n / fact n
            using norm-ge-zero [of z] by (simp del: norm-ge-zero)
          then show summable ?M
            by (rule ratio-test-convergence) (auto simp: add-nonneg-eq-0-iff L)
          fix n z'
          assume z' ∈ cball z 1
          then have norm (coeff n * p n z') ≤ norm (coeff n) * q n * (1 + norm
z) ^ n
          by (metis norm-p-bound norm-mult mem-cball mult.assoc mult-left-mono
norm-ge-zero)
          also have ... ≤ (1 / fact n) * (1 + norm z) ^ n
          proof (rule mult-right-mono)
            show norm (coeff n) * q n ≤ 1 / fact n
             by (metis divide-divide-eq-left less-divide-eq less-eq-real-def norm-coeff
q-gt0)
          qed auto
          also have ... ≤ ?M n
            by (simp add: divide-simps)
          finally show norm (coeff n * p n z') ≤ ?M n .
        qed
        then show ∃ d>0. cball z d ⊆ UNIV ∧ uniform-limit (cball z d) (λn. h
n coeff) hh sequentially
          using zero-less-one by blast
      qed auto
      then show hh analytic-on UNIV
        by (simp add: analytic-on-open)
      have hh-eq-dd: hh (w n) = dd n coeff for n
      proof −
```

```
        have hh (w n) = h (Suc n) coeff (w n)
          unfolding hh-def h-def by (intro suminf-finite) auto
        also have ... = dd n coeff
         by (induction n) (auto simp add: p0 h-def p-def coeff-eq [of Suc -] coeff-eq
[of 0])
        finally show ?thesis .
      qed
      then have hh (w n) ∈ D for n
        using DD-def dd-in-DD by fastforce
      then show inD γ hh
        unfolding inD-def by (metis η bij-betw-iff-bijections comp-apply w-def)
      have hh (w n) ≠ f (η n) (w n) for n
        using DD-def dd-in-DD g-def hh-eq-dd by auto
      then show ∀ β∈elts γ. hh ≠ f β
        by (metis η bij-betw-imp-surj-on imageE)
    qed
  qed
  with f show ?thesis
    using inj by (rule-tac x=h in exI) (auto simp: Φ-def inj-on-def)
 qed
 define G where G ≡ λf γ. @h. Φ γ ((restrict f (elts γ))(γ:=h))
 have nxt: Φ γ ((restrict f (elts γ))(γ:= G f γ))
  if γ ∈ elts (ℵ1) ∀ β ∈ elts γ. Φ β f for f γ
  unfolding G-def using ∗ [OF that] by (metis someI)
 have G-restr: G (restrict f (elts γ)) γ = G f γ if γ ∈ elts (ℵ1) for f γ
  by (auto simp: G-def)
 define f where f ≡ transrec G
 have Φf: Φ β f if β ∈ elts (ℵ1) for β
  using that
 proof (induction β rule: eps-induct)
  case (step γ)
  then have ∗: ∀ β∈elts γ. Φ β f
    using Ord-ω1 Ord-trans by blast
  have [simp]: inj-on (λβ. G (restrict f (elts β)) β) (elts γ) ⟷ inj-on f (elts γ)
    by (metis (no-types, lifting) f-def transrec inj-on-cong)
  have f γ = G f γ
    by (metis G-restr transrec f-def step.prems)
  with nxt [OF step.prems] ∗ show ?case
      by (metis Φ-def elts-succ fun-upd-same fun-upd-triv inj-on-restrict-eq re-
strict-upd)
 qed
 then have anf: ⋀β. β ∈ elts (ℵ1) ⟹ f β analytic-on UNIV
  and inD: ⋀α β. ⟦β ∈ elts (ℵ1); α ∈ elts β⟧ ⟹ f β (ζ α) ∈ D
  using Φ-def inD-def by blast+
 have injf: inj-on f (elts (ℵ1))
  using Φf unfolding inj-on-def Φ-def by (metis Ord-ω1 Ord-in-Ord Ord-linear-le
in-succ-iff)
 show ?thesis
 proof
```

```
    let ?F = f ‘ elts (ℵ1)
    have countable ((λf. f z) ‘ f ‘ elts ω1) for z
    proof −
      obtain α where α: ζ α = z α ∈ elts (ℵ1) Ord α
        by (meson Ord-ω1 Ord-in-Ord UNIV-I ζ bij-betw-iff-bijections)
      let ?B = elts ω1 − elts (succ α)
      have eq: elts ω1 = elts (succ α) ∪ ?B
       using α by (metis Diff-partition Ord-ω1 OrdmemD less-eq-V-def succ-le-iff)
      have (λf. f z) ‘ f ‘ ?B ⊆ D
        using α inD by clarsimp (meson Ord-ω1 Ord-in-Ord Ord-linear)
      then have countable ((λf. f z) ‘ f ‘ ?B)
        by (meson ‹countable D› countable-subset)
      moreover have countable ((λf. f z) ‘ f ‘ elts (succ α))
        by (simp add: α less-ω1-imp-countable)
      ultimately show ?thesis
        using eq by (metis countable-Un-iff image-Un)
    qed
    then show Wetzel ?F
      unfolding Wetzel-def by (blast intro: anf)
    show uncountable ?F
      using Ord-ω1 countable-iff-less-ω1 countable-image-inj-eq injf by blast
  qed
qed

theorem Erdos-Wetzel: C-continuum = ℵ1 ⟷ (∃F. Wetzel F ∧ uncountable F)
  by (metis C-continuum-ge Erdos-Wetzel-CH Erdos-Wetzel-nonCH TC-small less-V-def)

end
```

References

[1] M. Aigner and G. M. Ziegler. *Proofs from THE BOOK*. Springer, 6th edition, 2018.