The Cartan Fixed Point Theorems

Lawrence C. Paulson

December 14, 2021

Abstract

The Cartan fixed point theorems concern the group of holomorphic automorphisms on a connected open set of $\mathbb{C}^n$. Ciolli et al. [1] have formalised the one-dimensional case of these theorems in HOL Light. This entry contains their proofs, ported to Isabelle/HOL. Thus it addresses the authors remark that "it would be important to write a formal proof in a language that can be read by both humans and machines."

Contents

theory *Cartan*
imports *HOL−Complex-Analysis.Complex-Analysis*

begin

0.1 First Cartan Theorem

Ported from HOL Light. See Gianni Ciolli, Graziano Gentili, Marco Maggesi. A Certified Proof of the Cartan Fixed Point Theorems. J Automated Reasoning (2011) 47:319–336 DOI 10.1007/s10817-010-9198-6

lemma *deriv-left-inverse*:
  assumes *f holomorphic-on S* **and** *g holomorphic-on T*
    and *open S* **and** *open T*
    and f ‘ $S \subseteq T$
    and [*simp*]: $\bigwedge z.\ z \in S \Longrightarrow g\ (f\ z) = z$
    and $w \in S$
   shows *deriv f w* ∗ *deriv g* (*f w*) = *1*
proof −
  have *deriv f w* ∗ *deriv g* (*f w*) = *deriv g* (*f w*) ∗ *deriv f w*
   by (*simp add*: *algebra-simps*)
  also have ... = *deriv* (*g o f*) *w*
   using *assms*
    by (*metis analytic-on-imp-differentiable-at analytic-on-open deriv-chain image-subset-iff*)
  also have ... = *deriv id w*
   apply (*rule complex-derivative-transform-within-open* [**where** *s*=*S*])
   apply (*rule assms holomorphic-on-compose-gen holomorphic-intros*)+
   apply *simp*
   done
  also have ... = *1*
   using *higher-deriv-id* [*of 1*] **by** *simp*
  finally show *?thesis* **.**
qed

lemma *Cauchy-higher-deriv-bound*:
   assumes *holf*: *f holomorphic-on* (*ball z r*)

and *contf*: *continuous-on* (*cball z r*) *f*
and *0* < *r* **and** *0* < *n*
and *fin* : ⋀*w*. *w* ∈ *ball z r* ⟹ *f w* ∈ *ball y B0*
shows *norm* ((*deriv* ^^ *n*) *f z*) ≤ (*fact n*) ∗ *B0* / *r*^*n*
proof −
have *0* < *B0* **using** ‹*0* < *r*› *fin* [*of z*]
by (*metis ball-eq-empty ex-in-conv fin not-less*)
have *le-B0*: ⋀*w*. *cmod* (*w* − *z*) ≤ *r* ⟹ *cmod* (*f w* − *y*) ≤ *B0*
apply (*rule continuous-on-closure-norm-le* [*of ball z r* λ*w*. *f w* − *y*])
apply (*auto simp*: ‹*0* < *r*› *dist-norm norm-minus-commute*)
apply (*rule continuous-intros contf*)+
using *fin* **apply** (*simp add*: *dist-commute dist-norm less-eq-real-def*)
done
have (*deriv* ^^ *n*) *f z* = (*deriv* ^^ *n*) (λ*w*. *f w*) *z* − (*deriv* ^^ *n*) (λ*w*. *y*) *z*
using ‹*0* < *n*› **by** *simp*
also have ... = (*deriv* ^^ *n*) (λ*w*. *f w* − *y*) *z*
by (*rule higher-deriv-diff* [*OF holf*, *symmetric*]) (*auto simp*: ‹*0* < *r*› *holomorphic-on-const*)
finally have (*deriv* ^^ *n*) *f z* = (*deriv* ^^ *n*) (λ*w*. *f w* − *y*) *z* **.**
have *contf'*: *continuous-on* (*cball z r*) (λ*u*. *f u* − *y*)
by (*rule contf continuous-intros*)+
have *holf'*: (λ*u*. (*f u* − *y*)) *holomorphic-on* (*ball z r*)
by (*simp add*: *holf holomorphic-on-diff holomorphic-on-const*)
define *a* **where** *a* = (*2* ∗ *pi*)/(*fact n*)
have *0* < *a* **by** (*simp add*: *a-def*)
have *B0*/*r*^(*Suc n*)∗*2* ∗ *pi* ∗ *r* = *a*∗((*fact n*)∗*B0*/*r*^*n*)
using ‹*0* < *r*› **by** (*simp add*: *a-def divide-simps*)
have *der-dif*: (*deriv* ^^ *n*) (λ*w*. *f w* − *y*) *z* = (*deriv* ^^ *n*) *f z*
using ‹*0* < *r*› ‹*0* < *n*›
by (*auto simp*: *higher-deriv-diff* [*OF holf holomorphic-on-const*])
have *norm* ((*2* ∗ *of-real pi* ∗ i)/(*fact n*) ∗ (*deriv* ^^ *n*) (λ*w*. *f w* − *y*) *z*)
≤ (*B0*/*r*^(*Suc n*)) ∗ (*2* ∗ *pi* ∗ *r*)
apply (*rule has-contour-integral-bound-circlepath* [*of* (λ*u*. (*f u* − *y*)/(*u* − *z*)^(*Suc n*)) - *z*])
using *Cauchy-has-contour-integral-higher-derivative-circlepath* [*OF contf' holf'*]
using ‹*0* < *B0*› ‹*0* < *r*›
apply (*auto simp*: *norm-divide norm-mult norm-power divide-simps le-B0*)
done
then show *?thesis*
using ‹*0* < *r*›
by (*auto simp*: *norm-divide norm-mult norm-power field-simps der-dif le-B0*)
qed

lemma *higher-deriv-comp-lemma*:
assumes *s*: *open s* **and** *holf*: *f holomorphic-on s*
and *z* ∈ *s*
and *t*: *open t* **and** *holg*: *g holomorphic-on t*
and *fst*: *f* ‘ *s* ⊆ *t*
and *n*: *i* ≤ *n*

```
    and dfz: deriv f z = 1 and zero: ⋀i. ⟦1 < i; i ≤ n⟧ ⟹ (deriv ^^ i) f z = 0
   shows (deriv ^^ i) (g o f) z = (deriv ^^ i) g (f z)
using n holg
proof (induction i arbitrary: g)
  case 0 then show ?case by simp
next
  case (Suc i)
  have g ∘ f holomorphic-on s using Suc.prems holf
    using fst  by (simp add: holomorphic-on-compose-gen image-subset-iff)
  then have 1: deriv (g ∘ f) holomorphic-on s
    by (simp add: holomorphic-deriv s)
  have dg: deriv g holomorphic-on t
    using Suc.prems by (simp add: Suc.prems(2) holomorphic-deriv t)
  then have deriv g holomorphic-on f ‘ s
    using fst  by (simp add: holomorphic-on-subset image-subset-iff)
  then have dgf: (deriv g o f) holomorphic-on s
    by (simp add: holf holomorphic-on-compose)
  then have 2: (λw. (deriv g o f) w * deriv f w) holomorphic-on s
    by (blast intro: holomorphic-intros holomorphic-on-compose holf s)
  have (deriv ^^ i) (deriv (g o f)) z = (deriv ^^ i) (λw. deriv g (f w) * deriv f w)
z
    apply (rule higher-deriv-transform-within-open [OF 1 2 [unfolded o-def] s ‹z ∈
s›])
    apply (rule deriv-chain)
   using holf Suc.prems fst apply (auto simp: holomorphic-on-imp-differentiable-at
s t)
    done
  also have ... = (∑j=0..i. of-nat(i choose j) * (deriv ^^ j) (λw. deriv g (f w))
z * (deriv ^^ (i − j)) (deriv f) z)
    apply (rule higher-deriv-mult [OF dgf [unfolded o-def] - s ‹z ∈ s›])
    by (simp add: holf holomorphic-deriv s)
  also have ... = (∑j=i..i. of-nat(i choose j) * (deriv ^^ j) (λw. deriv g (f w)) z
* (deriv ^^ Suc (i − j)) f z)
  proof −
    have *: (deriv ^^ j) (λw. deriv g (f w)) z = 0  if j < i and nz: (deriv ^^ (i −
j)) (deriv f) z ≠ 0 for j
    proof −
      have 1 < Suc (i − j) Suc (i − j) ≤ n
        using ‹j < i› ‹Suc i ≤ n› by auto
      then show ?thesis  by (metis comp-def funpow.simps(2) funpow-swap1 zero
nz)
    qed
    then show ?thesis
      apply (simp only: funpow-Suc-right o-def)
      apply (rule comm-monoid-add-class.sum.mono-neutral-right, auto)
      done
  qed
  also have ... = (deriv ^^ i) (deriv g) (f z)
    using Suc.IH [OF - dg] Suc.prems by (simp add: dfz)
```

finally show *?case*
  by (*simp only*: *funpow-Suc-right o-def*)
qed

lemma *higher-deriv-comp-iter-lemma*:
  assumes *s*: *open s* **and** *holf*: *f holomorphic-on s*
    and *fss*: *f ‘ s ⊆ s*
    and *z ∈ s* **and** [*simp*]: *f z = z*
    and *n*: *i ≤ n*
    and *dfz*: *deriv f z = 1* **and** *zero*: ⋀*i*. ⟦*1 < i*; *i ≤ n*⟧ ⟹ (*deriv* ^^ *i*) *f z = 0*
   shows (*deriv* ^^ *i*) (*f*^^*m*) *z* = (*deriv* ^^ *i*) *f z*
proof −
 have *holfm*: (*f*^^*m*) *holomorphic-on s* **for** *m*
  apply (*induction m*, *simp add*: *holomorphic-on-ident*)
  apply (*simp only*: *funpow-Suc-right holomorphic-on-compose-gen* [*OF holf* - *fss*])
  done
 show *?thesis* **using** *n*
 proof (*induction m*)
  case *0* **with** *dfz* **show** *?case*
   by (*auto simp*: *zero*)
 next
  case (*Suc m*)
  have (*deriv* ^^ *i*) (*f* ^^ *m* ∘ *f*) *z* = (*deriv* ^^ *i*) (*f* ^^ *m*) (*f z*)
   using *Suc.prems holfm* ‹*z ∈ s*› *dfz fss higher-deriv-comp-lemma holf s zero* **by** *blast*
  also have ... = (*deriv* ^^ *i*) *f z*
   by (*simp add*: *Suc*)
  finally show *?case*
   by (*simp only*: *funpow-Suc-right*)
 qed
qed

lemma *higher-deriv-iter-top-lemma*:
  assumes *s*: *open s* **and** *holf*: *f holomorphic-on s*
    and *fss*: *f ‘ s ⊆ s*
    and *z ∈ s* **and** [*simp*]: *f z = z*
    and *dfz* [*simp*]: *deriv f z = 1*
    and *n*: *1 < n* ⋀*i*. ⟦*1 < i*; *i < n*⟧ ⟹ (*deriv* ^^ *i*) *f z = 0*
   shows (*deriv* ^^ *n*) (*f* ^^ *m*) *z* = *m* ∗ (*deriv* ^^ *n*) *f z*
using *n*
proof (*induction n arbitrary*: *m*)
 case *0* **then show** *?case* **by** *simp*
next
 case (*Suc n*)
 have [*simp*]: (*f*^^*m*) *z* = *z* **for** *m*
  by (*induction m*) *auto*
 have *fms-sb*: (*f*^^*m*) ‘ *s ⊆ s* **for** *m*

apply (*induction m*)
using *fss*
apply *force*+
done
have *holfm*: (f^^m) *holomorphic-on s* **for** *m*
apply (*induction m*, *simp add*: *holomorphic-on-ident*)
apply (*simp only*: *funpow-Suc-right holomorphic-on-compose-gen* [*OF holf* - *fss*])
done
then have *holdfm*: *deriv* (f ^^ m) *holomorphic-on s* **for** *m*
by (*simp add*: *holomorphic-deriv s*)
have *holdffm*: (λz. *deriv f* ((f ^^ m) z)) *holomorphic-on s* **for** *m*
apply (*rule holomorphic-on-compose-gen* [**where** *g*=*deriv f* **and** *t*=*s*, *unfolded o-def*])
using *s* ‹*z* ∈ *s*› *holfm holf fms-sb* **by** (*auto intro*: *holomorphic-intros*)
have *f-cd-w*: $\bigwedge w$. $w \in s \Longrightarrow$ *f field-differentiable at w*
using *holf holomorphic-on-imp-differentiable-at s* **by** *blast*
have *f-cd-mw*: $\bigwedge m\ w$. $w \in s \Longrightarrow$ (f^^m) *field-differentiable at w*
using *holfm holomorphic-on-imp-differentiable-at s* **by** *auto*
have *der-fm* [*simp*]: *deriv* (f ^^ m) z = *1* **for** *m*
apply (*induction m*, *simp add*: *deriv-ident*)
apply (*subst funpow-Suc-right*)
apply (*subst deriv-chain*)
using ‹*z* ∈ *s*› *holfm holomorphic-on-imp-differentiable-at s f-cd-w* **apply** *auto*
done
note *Suc*(*3*) [*simp*]
note *n-Suc* = *Suc*
show *?case*
proof (*induction m*)
case *0* **with** *n-Suc* **show** *?case*
by (*metis Zero-not-Suc funpow-simps-right*(*1*) *higher-deriv-id lambda-zero nat-neq-iff of-nat-0*)
next
case (*Suc m*)
have *deriv-nffm*: (*deriv* ^^ n) (*deriv f o* (f ^^ m)) z = (*deriv* ^^ n) (*deriv f*) ((f ^^ m) z)
apply (*rule higher-deriv-comp-lemma* [*OF s holfm* ‹*z* ∈ *s*› *s* - *fms-sb order-refl*])
using ‹*z* ∈ *s*› *fss higher-deriv-comp-iter-lemma holf holf holomorphic-deriv s*
apply *auto*
done
have *deriv* (f ^^ m ∘ f) *holomorphic-on s*
by (*metis funpow-Suc-right holdfm*)
moreover have (λw. *deriv f* ((f ^^ m) w) ∗ *deriv* (f ^^ m) w) *holomorphic-on s*
by (*rule holomorphic-on-mult* [*OF holdffm holdfm*])
ultimately have (*deriv* ^^ n) (*deriv* (f ^^ m ∘ f)) z = (*deriv* ^^ n) (λw. *deriv f* ((f ^^ m) w) ∗ *deriv* (f ^^ m) w) z
apply (*rule higher-deriv-transform-within-open* [*OF* - - *s* ‹*z* ∈ *s*›])
by (*metis comp-funpow deriv-chain f-cd-mw f-cd-w fms-sb funpow-swap1 im-*

age-subset-iff o-id)
also have ... =
($\sum$ *i=0..n. of-nat*(*n choose i*) ∗ (*deriv* ^^ *i*) (λ*w. deriv f* ((*f* ^^ *m*) *w*)) *z* ∗
(*deriv* ^^ (*n* − *i*)) (*deriv* (*f* ^^ *m*)) *z*)
by (*rule higher-deriv-mult* [*OF holdffm holdfm s* ‹*z* ∈ *s*›])
also have ... = ($\sum$ *i* ∈ {*0*,*n*}. *of-nat*(*n choose i*) ∗ (*deriv* ^^ *i*) (λ*w. deriv f* ((*f* ^^ *m*) *w*)) *z* ∗
(*deriv* ^^ (*n* − *i*)) (*deriv* (*f* ^^ *m*)) *z*)
proof −
have ∗: (*deriv* ^^ *i*) (λ*w. deriv f* ((*f* ^^ *m*) *w*)) *z* = *0* **if** *i* ≤ *n* *0* < *i* *i* ≠ *n* **and** *nz*: (*deriv* ^^ (*n* − *i*)) (*deriv* (*f* ^^ *m*)) *z* ≠ *0* **for** *i*
proof −
have *less*: *1* < *Suc* (*n*−*i*) **and** *le*: *Suc* (*n*−*i*) ≤ *n*
using *that* **by** *auto*
have (*deriv* ^^ (*Suc* (*n* − *i*))) (*f* ^^ *m*) *z* = (*deriv* ^^ (*Suc* (*n* − *i*))) *f z*
apply (*rule higher-deriv-comp-iter-lemma* [*OF s holf fss* ‹*z* ∈ *s*› ‹*f z* = *z*› *le dfz*])
by *simp*
also have ... = *0*
using *n-Suc*(*3*) *less le le-imp-less-Suc* **by** *blast*
finally have (*deriv* ^^ (*Suc* (*n* − *i*))) (*f* ^^ *m*) *z* = *0* **.**
then show *?thesis* **by** (*simp add*: *funpow-swap1 nz*)
qed
show *?thesis*
by (*rule comm-monoid-add-class.sum.mono-neutral-right*) (*auto simp*: ∗)
qed
also have ... = *of-nat* (*Suc m*) ∗ (*deriv* ^^ *n*) (*deriv f*) *z*
apply (*subst Groups-Big.comm-monoid-add-class.sum.insert*)
apply (*simp-all add*: *deriv-nffm* [*unfolded o-def*] *of-nat-Suc* [*of 0*] *del*: *of-nat-Suc*)
using *n-Suc*(*2*) *Suc*
apply (*auto simp del*: *funpow.simps simp*: *algebra-simps funpow-simps-right*)
done
finally have (*deriv* ^^ *n*) (*deriv* (*f* ^^ *m* ∘ *f*)) *z* = *of-nat* (*Suc m*) ∗ (*deriv* ^^ *n*) (*deriv f*) *z* **.**
then show *?case*
apply (*simp only*: *funpow-Suc-right*)
apply (*simp add*: *o-def del*: *of-nat-Suc*)
done
qed
qed

Should be proved for n-dimensional vectors of complex numbers

theorem *first-Cartan-dim-1*:
assumes *holf*: *f holomorphic-on s*
and *open s connected s bounded s*
and *fss*: *f* ‘ *s* ⊆ *s*
and *z* ∈ *s* **and** [*simp*]: *f z* = *z*
and *dfz* [*simp*]: *deriv f z* = *1*
and *w* ∈ *s*

shows *f w* = *w*
proof −
obtain *c* **where** *0* < *c* **and** *c*: *s* ⊆ *ball z c*
using ‹*bounded s*› *bounded-subset-ballD* **by** *blast*
obtain *r* **where** *0* < *r* **and** *r*: *cball z r* ⊆ *s*
using ‹*z* ∈ *s*› *open-contains-cball* ‹*open s*› **by** *blast*
then have *bzr*: *ball z r* ⊆ *s* **using** *ball-subset-cball* **by** *blast*
have *fms-sb*: (*f*^^*m*) ‘ *s* ⊆ *s* **for** *m*
apply (*induction m*)
using *fss* **apply** *force*+
done
have *holfm*: (*f*^^*m*) *holomorphic-on s* **for** *m*
apply (*induction m*, *simp add*: *holomorphic-on-ident*)
apply (*simp only*: *funpow-Suc-right holomorphic-on-compose-gen* [*OF holf* - *fss*])
done
have ∗: (*deriv* ^^ *n*) *f z* = (*deriv* ^^ *n*) *id z* **for** *n*
proof −
consider *n* = *0* | *n* = *1* | *1* < *n* **by** *arith*
then show *?thesis*
proof *cases*
assume *n* = *0* **then show** *?thesis* **by** *force*
next
assume *n* = *1* **then show** *?thesis* **by** *force*
next
assume *n1*: *n* > *1*
then have (*deriv* ^^ *n*) *f z* = *0*
proof (*induction n rule*: *less-induct*)
case (*less n*)
have *le*: *real m* ∗ *cmod* ((*deriv* ^^ *n*) *f z*) ≤ *fact n* ∗ *c* / *r* ^ *n* **if** *m*≠*0* **for** *m*
proof −
have *holfm'*: (*f* ^^ *m*) *holomorphic-on ball z r*
using *holfm bzr holomorphic-on-subset* **by** *blast*
then have *contfm'*: *continuous-on* (*cball z r*) (*f* ^^ *m*)
using ‹*cball z r* ⊆ *s*› *holfm holomorphic-on-imp-continuous-on holomorphic-on-subset* **by** *blast*
have *real m* ∗ *cmod* ((*deriv* ^^ *n*) *f z*) = *cmod* (*real m* ∗ (*deriv* ^^ *n*) *f z*)
by (*simp add*: *norm-mult*)
also have ... = *cmod* ((*deriv* ^^ *n*) (*f* ^^ *m*) *z*)
apply (*subst higher-deriv-iter-top-lemma* [*OF* ‹*open s*› *holf fss* ‹*z* ∈ *s*› ‹*f z* = *z*› *dfz*])
using *less* **apply** *auto*
done
also have ... ≤ *fact n* ∗ *c* / *r* ^ *n*
apply (*rule Cauchy-higher-deriv-bound* [*OF holfm' contfm'* ‹*0* < *r*›, **where** *y*=*z*])
using *less.prems* **apply** *linarith*
using *fms-sb c r ball-subset-cball*

```
        apply blast
        done
      finally show ?thesis .
    qed
    have cmod ((deriv ^^ n) f z) = 0
      apply (rule real-archimedian-rdiv-eq-0 [where c = (fact n) * c / r ^ n])
      apply simp
      using ‹0 < r› ‹0 < c›
      apply (simp add: divide-simps)
      apply (blast intro: le)
      done
    then show ?case by simp
  qed
  with n1 show ?thesis by simp
 qed
qed
have f w = id w
  by (rule holomorphic-fun-eq-on-connected
        [OF holf holomorphic-on-id ‹open s› ‹connected s› * ‹z ∈ s› ‹w ∈ s›])
also have ... = w by simp
finally show ?thesis .
qed
```

Second Cartan Theorem.

```
lemma Cartan-is-linear:
  assumes holf: f holomorphic-on s
    and open s and connected s
    and 0 ∈ s
    and ins: ⋀u z. ⟦norm u = 1; z ∈ s⟧ ⟹ u * z ∈ s
    and feq: ⋀u z. ⟦norm u = 1; z ∈ s⟧ ⟹ f (u * z) = u * f z
  shows ∃ c. ∀ z ∈ s. f z = c * z
proof –
  have [simp]: f 0 = 0
    using feq [of −1 0] assms by simp
  have uneq: u^n * (deriv ^^ n) f (u * z) = u * (deriv ^^ n) f z
     if norm u = 1 z ∈ s for n u z
  proof –
    have holfuw: (λw. f (u * w)) holomorphic-on s
      apply (rule holomorphic-on-compose-gen [OF - holf, unfolded o-def])
      using that ins by (auto simp: holomorphic-on-linear)
    have hol-d-fuw: (deriv ^^ n) (λw. u * f w) holomorphic-on s for n
      by (rule holomorphic-higher-deriv holomorphic-intros holf assms)+
    have *: (deriv ^^ n) (λw. u * f w) z = u * (deriv ^^ n) f z if z ∈ s for z
    using that
    proof (induction n arbitrary: z)
      case 0 then show ?case by simp
    next
      case (Suc n)
      have deriv ((deriv ^^ n) (λw. u * f w)) z = deriv (λw. u * (deriv ^^ n) f w)
```

z
apply (*rule complex-derivative-transform-within-open* [*OF hol-d-fuw*])
apply (*auto intro*!: *holomorphic-higher-deriv holomorphic-intros assms Suc*)
done
also have ... = *u* ∗ *deriv* ((*deriv* ^^ *n*) *f*) *z*
apply (*rule deriv-cmult*)
using *Suc* ‹*open s*› *holf holomorphic-higher-deriv holomorphic-on-imp-differentiable-at*
by *blast*
finally show *?case* **by** *simp*
qed
have (*deriv* ^^ *n*) (λ*w*. *f* (*u* ∗ *w*)) *z* = *u* ^ *n* ∗ (*deriv* ^^ *n*) *f* (*u* ∗ *z*)
apply (*rule higher-deriv-compose-linear* [*OF holf* ‹*open s*› ‹*open s*›])
apply (*simp add*: *that*)
apply (*simp add*: *ins that*)
done
moreover have (*deriv* ^^ *n*) (λ*w*. *f* (*u* ∗ *w*)) *z* = *u* ∗ (*deriv* ^^ *n*) *f z*
apply (*subst higher-deriv-transform-within-open* [*OF holfuw*, *of* λ*w*. *u* ∗ *f w*])
apply (*rule holomorphic-intros holf assms that*)+
apply *blast*
using ∗ ‹*z* ∈ *s*› **apply** *blast*
done
ultimately show *?thesis* **by** *metis*
qed
have *dnf0*: (*deriv* ^^ *n*) *f 0* = *0* **if** *len*: *2* ≤ *n* **for** *n*
proof −
have ∗∗: *z* = *0* **if** ⋀*u*::*complex*. *norm u* = *1* ⟹ *u* ^ *n* ∗ *z* = *u* ∗ *z* **for** *z*
proof −
have ∃ *u*::*complex*. *norm u* = *1* ∧ *u* ^ *n* ≠ *u*
using *complex-not-root-unity* [*of n*−*1*] *len*
apply (*simp add*: *algebra-simps le-diff-conv2*, *clarify*)
apply (*rule-tac x*=*u* **in** *exI*)
apply (*subst* (*asm*) *power-diff*)
apply *auto*
done
with *that* **show** *?thesis*
by *auto*
qed
show *?thesis*
apply (*rule* ∗∗)
using *uneq* [*OF* - ‹*0* ∈ *s*›]
by *force*
qed
show *?thesis*
apply (*rule-tac x* = *deriv f 0* **in** *exI*, *clarify*)
apply (*rule holomorphic-fun-eq-on-connected* [*OF holf* - ‹*open s*› ‹*connected s*›
- ‹*0* ∈ *s*›])
using *dnf0* **apply** (*auto simp*: *holomorphic-on-linear*)
done
qed

Should be proved for n-dimensional vectors of complex numbers

```
theorem second-Cartan-dim-1:
  assumes holf: f holomorphic-on ball 0 r
      and holg: g holomorphic-on ball 0 r
      and [simp]: f 0 = 0 and [simp]: g 0 = 0
      and ballf: ⋀z. z ∈ ball 0 r ⟹ f z ∈ ball 0 r
      and ballg: ⋀z. z ∈ ball 0 r ⟹ g z ∈ ball 0 r
      and fg: ⋀z. z ∈ ball 0 r ⟹ f (g z) = z
      and gf: ⋀z. z ∈ ball 0 r ⟹ g (f z) = z
      and 0 < r
    shows ∃ t. ∀ z ∈ ball 0 r. g z = exp(i ∗ of-real t) ∗ z
proof −
  have c-le-1: c ≤ 1
    if 0 ≤ c ⋀x. 0 ≤ x ⟹ x < r ⟹ c ∗ x < r for c
  proof −
    have rst: ⋀r s t::real. 0 = r ∨ s/r < t ∨ r < 0 ∨ ¬ s < r ∗ t
         by (metis (no-types) mult-less-cancel-left-disj nonzero-mult-div-cancel-left times-divide-eq-right)
    { assume ¬ r < c ∧ c ∗ (c ∗ (c ∗ (c ∗ r))) < 1
     then have 1 ≤ c ⟹ (∃ r. ¬ 1 < r ∧ ¬ r < c)
           using ‹0 ≤ c› by (metis (full-types) less-eq-real-def mult.right-neutral mult-left-mono not-less)
      then have ¬ 1 < c ∨ ¬ 1 ≤ c
        by linarith }
    moreover
    { have ¬ 0 ≤ r / c ⟹ ¬ 1 ≤ c
          using ‹0 < r› by force
      then have 1 < c ⟹ ¬ 1 ≤ c
        using rst ‹0 < r› that
          by (metis div-by-1 frac-less2 less-le-trans mult.commute not-le order-refl pos-divide-le-eq zero-less-one) }
    ultimately show ?thesis
      by (metis (no-types) linear not-less)
  qed
  have ugeq: u ∗ g z = g (u ∗ z) if nou: norm u = 1 and z: z ∈ ball 0 r for u z
  proof −
    have [simp]: u ≠ 0 using that by auto
    have hol1: (λa. f (u ∗ g a) / u) holomorphic-on ball 0 r
      apply (rule holomorphic-intros)
      apply (rule holomorphic-on-compose-gen [OF - holf, unfolded o-def])
      apply (rule holomorphic-intros holg)+
      using nou ballg
      apply (auto simp: dist-norm norm-mult holomorphic-on-const)
      done
    have cdf: f field-differentiable at 0
      using ‹0 < r› holf holomorphic-on-imp-differentiable-at by auto
    have cdg: g field-differentiable at 0
      using ‹0 < r› holg holomorphic-on-imp-differentiable-at by auto
    have cd-fug: (λa. f (u ∗ g a)) field-differentiable at 0
```

apply (*rule field-differentiable-compose* [**where** *g=f* **and** *f* = λ*a*. (*u* ∗ *g a*), *unfolded o-def*])
apply (*rule derivative-intros*)+
using *cdf cdg*
apply *auto*
done
have *deriv g 0* = *deriv g* (*f 0*)
by *simp*
then have *deriv f 0* ∗ *deriv g 0* = *1*
by (*metis open-ball* ‹*0* < *r*› *ballf centre-in-ball deriv-left-inverse gf holf holg image-subsetI*)
then have *equ*: *deriv f 0* ∗ *deriv* (λ*a*. *u* ∗ *g a*) *0* = *u*
by (*simp add*: *cdg deriv-cmult*)
have *der1*: *deriv* (λ*a*. *f* (*u* ∗ *g a*) / *u*) *0* = *1*
apply (*simp add*: *field-class.field-divide-inverse deriv-cmult-right* [*OF cd-fug*])
apply (*subst deriv-chain* [**where** *g=f* **and** *f* = λ*a*. (*u* ∗ *g a*), *unfolded o-def*])
apply (*rule derivative-intros cdf cdg* | *simp add*: *equ*)+
done
have *fugeq*: ⋀*w*. *w* ∈ *ball 0 r* ⟹ *f* (*u* ∗ *g w*) / *u* = *w*
apply (*rule first-Cartan-dim-1* [*OF hol1*, **where** *z=0*])
apply (*simp-all add*: ‹*0* < *r*›)
apply (*auto simp*: *der1*)
using *nou ballf ballg*
apply (*simp add*: *dist-norm norm-mult norm-divide*)
done
have *f*(*u* ∗ *g z*) = *u* ∗ *z*
by (*metis* ‹*u* ≠ *0*› *fugeq nonzero-mult-div-cancel-left z times-divide-eq-right*)
also have ... = *f* (*g* (*u* ∗ *z*))
by (*metis* (*no-types*, *lifting*) *fg mem-ball-0 mult-cancel-right2 norm-mult nou z*)
finally have *f*(*u* ∗ *g z*) = *f* (*g* (*u* ∗ *z*)) **.**
then have *g* (*f* (*u* ∗ *g z*)) = *g* (*f* (*g* (*u* ∗ *z*)))
by *simp*
then show *?thesis*
apply (*subst* (*asm*) *gf*)
apply (*simp add*: *dist-norm norm-mult nou*)
using *ballg mem-ball-0 z* **apply** *blast*
apply (*subst* (*asm*) *gf*)
apply (*simp add*: *dist-norm norm-mult nou*)
apply (*metis ballg mem-ball-0 mult.left-neutral norm-mult nou z*, *simp*)
done
qed
obtain *c* **where** *c*: ⋀*z*. *z* ∈ *ball 0 r* ⟹ *g z* = *c* ∗ *z*
apply (*rule exE* [*OF Cartan-is-linear* [*OF holg*]])
apply (*simp-all add*: ‹*0* < *r*› *ugeq*)
apply (*auto simp*: *dist-norm norm-mult*)
done
have *gr2*: *g* (*f* (*r*/*2*)) = *c* ∗ *f*(*r*/*2*)
apply (*rule c*) **using** ‹*0* < *r*› *ballf mem-ball-0* **by** *force*

```
  then have norm c > 0
    using ‹0 < r›
      by simp (metis ‹f 0 = 0› c dist-commute fg mem-ball mult-zero-left perfect-choose-dist)
  then have [simp]: c ≠ 0 by auto
  have xless: x < r ∗ cmod c if 0 ≤ x x < r for x
  proof −
    have x = norm (g (f (of-real x)))
    proof −
      have r > cmod (of-real x)
        by (simp add: that)
      then have complex-of-real x ∈ ball 0 r
        using mem-ball-0 by blast
      then show ?thesis
        using gf ‹0 ≤ x› by force
    qed
    then show ?thesis
      apply (rule ssubst)
      apply (subst c)
      apply (rule ballf)
      using ballf [of x] that
      apply (auto simp: norm-mult dist-0-norm)
      done
  qed
  have 11: 1 / norm c ≤ 1
    apply (rule c-le-1)
    using xless apply (auto simp: divide-simps)
    done
  have ⟦0 ≤ x; x < r⟧ ⟹ cmod c ∗ x < r for x
    using c [of x] ballg [of x] by (auto simp: norm-mult dist-0-norm)
    then have norm c ≤ 1
    by (force intro: c-le-1)
  moreover have 1 ≤ norm c
    using 11 by simp
  ultimately have norm c = 1  by (rule antisym)
  with complex-norm-eq-1-exp c show ?thesis
    by metis
qed

end
```

Bibliography

[1] G. Ciolli, G. Gentili, and M. Maggesi. A certified proof of the Cartan fixed point theorems. *J. Autom. Reason.*, 47(3):319–336, Oct. 2011.