Evaluate Winding Numbers through Cauchy Indices

Wenda Li

September 18, 2020

Abstract

In complex analysis, the winding number measures the number of times a path (counterclockwise) winds around a point, while the Cauchy index can approximate how the path winds. This entry provides a formalisation of the Cauchy index, which is then shown to be related to the winding number. In addition, this entry also offers a tactic that enables users to evaluate the winding number by calculating Cauchy indices. The connection between the winding number and the Cauchy index can be found in the literature [1] [2, Chapter 11].

1 Some useful lemmas in topology

theory *Missing-Topology* **imports** *HOL−Analysis.Multivariate-Analysis*
begin

1.1 Misc

lemma *open-times-image*:
  fixes *S*::*'a*::*real-normed-field set*
  assumes *open S c≠0*
  shows *open* (((∗) *c*) ‘ *S*)
⟨*proof*⟩

lemma *image-linear-greaterThan*:
  fixes *x*::*'a*::*linordered-field*
  assumes *c≠0*
  shows ((λ*x*. *c*∗*x*+*b*) ‘ {*x*<..}) = (*if c*>*0 then* {*c*∗*x*+*b* <..} *else* {..< *c*∗*x*+*b*})
⟨*proof*⟩

lemma *image-linear-lessThan*:
  fixes *x*::*'a*::*linordered-field*
  assumes *c≠0*
  shows ((λ*x*. *c*∗*x*+*b*) ‘ {..<*x*}) = (*if c*>*0 then* {..<*c*∗*x*+*b*} *else* {*c*∗*x*+*b*<..})
⟨*proof*⟩

lemma *continuous-on-neq-split*:
  fixes *f* :: *'a*::*linear-continuum-topology* ⇒ *'b*::*linorder-topology*

assumes $\forall x{\in}s.\ f\ x{\neq}y$ *continuous-on* $s\ f$ *connected* s
shows $(\forall x{\in}s.\ f\ x{>}y) \vee (\forall x{\in}s.\ f\ x{<}y)$
⟨*proof*⟩

lemma
fixes $f{::}'a{::}$*linorder-topology* $\Rightarrow$ $'b{::}$*topological-space*
assumes *continuous-on* $\{a..b\}\ f\ a{<}b$
shows *continuous-on-at-left*:*continuous* $(at\text{-}left\ b)\ f$
and *continuous-on-at-right*:*continuous* $(at\text{-}right\ a)\ f$
⟨*proof*⟩

1.2 More about *eventually*

lemma *eventually-comp-filtermap*:
$eventually\ (P\ o\ f)\ F \longleftrightarrow eventually\ P\ (filtermap\ f\ F)$
⟨*proof*⟩

lemma *eventually-uminus-at-top-at-bot*:
fixes $P{::}'a{::}\{$*ordered-ab-group-add*,*linorder*$\} \Rightarrow$ *bool*
shows $eventually\ (P\ o\ uminus)\ at\text{-}bot \longleftrightarrow eventually\ P\ at\text{-}top$
$eventually\ (P\ o\ uminus)\ at\text{-}top \longleftrightarrow eventually\ P\ at\text{-}bot$
⟨*proof*⟩

lemma *eventually-at-infinityI*:
fixes $P{::}'a{::}$*real-normed-vector* $\Rightarrow$ *bool*
assumes $\bigwedge x.\ c \leq norm\ x \Longrightarrow P\ x$
shows $eventually\ P\ at\text{-}infinity$
⟨*proof*⟩

lemma *eventually-at-bot-linorderI*:
fixes $c{::}'a{::}$*linorder*
assumes $\bigwedge x.\ x \leq c \Longrightarrow P\ x$
shows $eventually\ P\ at\text{-}bot$
⟨*proof*⟩

lemma *eventually-times-inverse-1*:
fixes $f{::}'a \Rightarrow 'b{::}\{$*field*,*t2-space*$\}$
assumes $(f \longrightarrow c)\ F\ c{\neq}0$
shows $\forall_F\ x\ in\ F.\ inverse\ (f\ x) * f\ x = 1$
⟨*proof*⟩

1.3 More about *filtermap*

lemma *filtermap-linear-at-within*:
assumes *bij* f **and** *cont*: *isCont* $f\ a$ **and** *open-map*: $\bigwedge S.\ open\ S \Longrightarrow open\ (f`S)$
shows $filtermap\ f\ (at\ a\ within\ S) = at\ (f\ a)\ within\ f`S$
⟨*proof*⟩

lemma *filtermap-at-bot-linear-eq*:
fixes $c{::}'a{::}$*linordered-field*

assumes $c{\neq}0$
shows *filtermap* $(\lambda x.\ x * c + b)$ *at-bot* = (*if* $c{>}0$ *then at-bot else at-top*)
⟨*proof*⟩

lemma *filtermap-linear-at-left*:
fixes $c{::}'a{::}\{$*linordered-field*,*linorder-topology*,*real-normed-field*$\}$
assumes $c{\neq}0$
shows *filtermap* $(\lambda x.\ c{*}x{+}b)$ (*at-left* x) = (*if* $c{>}0$ *then at-left* $(c{*}x{+}b)$ *else at-right* $(c{*}x{+}b)$)
⟨*proof*⟩

lemma *filtermap-linear-at-right*:
fixes $c{::}'a{::}\{$*linordered-field*,*linorder-topology*,*real-normed-field*$\}$
assumes $c{\neq}0$
shows *filtermap* $(\lambda x.\ c{*}x{+}b)$ (*at-right* x) = (*if* $c{>}0$ *then at-right* $(c{*}x{+}b)$ *else at-left* $(c{*}x{+}b)$)
⟨*proof*⟩

lemma *filtermap-at-top-linear-eq*:
fixes $c{::}'a{::}$*linordered-field*
assumes $c{\neq}0$
shows *filtermap* $(\lambda x.\ x * c + b)$ *at-top* = (*if* $c{>}0$ *then at-top else at-bot*)
⟨*proof*⟩

lemma *filtermap-nhds-open-map*:
assumes *cont*: *isCont f a*
and *open-map*: $\bigwedge S.$ *open* $S \Longrightarrow$ *open* (*f*`S)
shows *filtermap f* (*nhds a*) = *nhds* (*f a*)
⟨*proof*⟩

1.4 More about *filterlim*

lemma *filterlim-at-infinity-times*:
fixes $f :: 'a \Rightarrow 'b{::}$*real-normed-field*
assumes *filterlim f at-infinity F filterlim g at-infinity F*
shows *filterlim* $(\lambda x.\ f\ x * g\ x)$ *at-infinity F*
⟨*proof*⟩

lemma *filterlim-at-top-at-bot*[*elim*]:
fixes $f{::}'a \Rightarrow 'b{::}$*unbounded-dense-linorder* **and** $F{::}'a$ *filter*
assumes *top*:*filterlim f at-top F* **and** *bot*: *filterlim f at-bot F* **and** $F{\neq}$*bot*
shows *False*
⟨*proof*⟩

lemma *filterlim-at-top-nhds*[*elim*]:
fixes $f{::}'a \Rightarrow 'b{::}\{$*unbounded-dense-linorder*,*order-topology*$\}$ **and** $F{::}'a$ *filter*
assumes *top*:*filterlim f at-top F* **and** *tendsto*: $(f \longrightarrow c)\ F$ **and** $F{\neq}$*bot*
shows *False*

⟨*proof*⟩

lemma *filterlim-at-bot-nhds*[*elim*]:
fixes f::$'a \Rightarrow 'b$::{*unbounded-dense-linorder*,*order-topology*} **and** F::$'a$ *filter*
assumes *top*:*filterlim f at-bot F* **and** *tendsto*: $(f \longrightarrow c)\ F$ **and** $F \neq bot$
shows *False*
⟨*proof*⟩

lemma *filterlim-at-top-linear-iff*:
fixes f::$'a$::*linordered-field* $\Rightarrow$ $'b$
assumes $c \neq 0$
shows (*LIM x at-top. f* $(x * c + b)$:> *F2*) $\longleftrightarrow$ (*if c>0 then* (*LIM x at-top. f x* :> *F2*)
else (*LIM x at-bot. f x* :> *F2*))
⟨*proof*⟩

lemma *filterlim-at-bot-linear-iff*:
fixes f::$'a$::*linordered-field* $\Rightarrow$ $'b$
assumes $c \neq 0$
shows (*LIM x at-bot. f* $(x * c + b)$:> *F2*) $\longleftrightarrow$ (*if c>0 then* (*LIM x at-bot. f x* :> *F2*)
else (*LIM x at-top. f x* :> *F2*))
⟨*proof*⟩

lemma *filterlim-tendsto-add-at-top-iff*:
assumes *f*: $(f \longrightarrow c)\ F$
shows (*LIM x F.* $(f\ x + g\ x :: real)$:> *at-top*) $\longleftrightarrow$ (*LIM x F. g x* :> *at-top*)
⟨*proof*⟩

lemma *filterlim-tendsto-add-at-bot-iff*:
fixes *c*::*real*
assumes *f*: $(f \longrightarrow c)\ F$
shows (*LIM x F. f x* $+$ *g x* :> *at-bot*) $\longleftrightarrow$ (*LIM x F. g x* :> *at-bot*)
⟨*proof*⟩

lemma *tendsto-inverse-0-at-infinity*:
LIM x F. f x :> *at-infinity* $\Longrightarrow$ $((\lambda x.\ inverse\ (f\ x) :: real) \longrightarrow 0)\ F$
⟨*proof*⟩

lemma *filterlim-at-infinity-divide-iff*:
fixes f::$'a \Rightarrow 'b$::*real-normed-field*
assumes $(f \longrightarrow c)\ F\ c \neq 0$
shows (*LIM x F. f x / g x* :> *at-infinity*) $\longleftrightarrow$ (*LIM x F. g x* :> *at 0*)
⟨*proof*⟩

lemma *filterlim-tendsto-pos-mult-at-top-iff*:
fixes f::$'a \Rightarrow$ *real*

assumes $(f \longrightarrow c)\ F$ **and** $0 < c$
shows $(LIM\ x\ F.\ (f\ x * g\ x) :> at\text{-}top) \longleftrightarrow (LIM\ x\ F.\ g\ x :> at\text{-}top)$
⟨*proof*⟩

lemma *filterlim-tendsto-pos-mult-at-bot-iff*:
fixes $c :: real$
assumes $(f \longrightarrow c)\ F\ 0 < c$
shows $(LIM\ x\ F.\ f\ x * g\ x :> at\text{-}bot) \longleftrightarrow filterlim\ g\ at\text{-}bot\ F$
⟨*proof*⟩

lemma *filterlim-tendsto-neg-mult-at-top-iff*:
fixes $f :: 'a \Rightarrow real$
assumes $(f \longrightarrow c)\ F$ **and** $c < 0$
shows $(LIM\ x\ F.\ (f\ x * g\ x) :> at\text{-}top) \longleftrightarrow (LIM\ x\ F.\ g\ x :> at\text{-}bot)$
⟨*proof*⟩

lemma *filterlim-tendsto-neg-mult-at-bot-iff*:
fixes $c :: real$
assumes $(f \longrightarrow c)\ F\ 0 > c$
shows $(LIM\ x\ F.\ f\ x * g\ x :> at\text{-}bot) \longleftrightarrow filterlim\ g\ at\text{-}top\ F$
⟨*proof*⟩

lemma *Lim-add*:
fixes $f\ g :: - \Rightarrow 'a :: \{t2\text{-}space, topological\text{-}monoid\text{-}add\}$
assumes $\exists\, y.\ (f \longrightarrow y)\ F$ **and** $\exists\, y.\ (g \longrightarrow y)\ F$ **and** $F \neq bot$
shows $Lim\ F\ f + Lim\ F\ g = Lim\ F\ (\lambda x.\ f\ x + g\ x)$
⟨*proof*⟩

1.5 Isolate and discrete

definition (**in** *topological-space*) *isolate* :: $'a \Rightarrow 'a\ set \Rightarrow bool$ (**infixr** *isolate 60*)
where $x\ isolate\ S \longleftrightarrow (x \in S \wedge (\exists\, T.\ open\ T \wedge T \cap S = \{x\}))$

definition (**in** *topological-space*) *discrete* :: $'a\ set \Rightarrow bool$
where $discrete\ S \longleftrightarrow (\forall\, x \in S.\ x\ isolate\ S)$

definition (**in** *metric-space*) *uniform-discrete* :: $'a\ set \Rightarrow bool$ **where**
$uniform\text{-}discrete\ S \longleftrightarrow (\exists\, e > 0.\ \forall\, x \in S.\ \forall\, y \in S.\ dist\ x\ y < e \longrightarrow x = y)$

lemma *uniformI1*:
assumes $e > 0\ \bigwedge x\ y.\ [\![x \in S; y \in S; dist\ x\ y < e]\!] \Longrightarrow x = y$
shows *uniform-discrete S*
⟨*proof*⟩

lemma *uniformI2*:
assumes $e > 0\ \bigwedge x\ y.\ [\![x \in S; y \in S; x \neq y]\!] \Longrightarrow dist\ x\ y \geq e$
shows *uniform-discrete S*
⟨*proof*⟩

lemma *isolate-islimpt-iff*:$(x\ isolate\ S) \longleftrightarrow (\neg\ (x\ islimpt\ S) \wedge x{\in}S)$
⟨*proof*⟩

lemma *isolate-dist-Ex-iff*:
fixes $x{::}'a{::}metric\text{-}space$
shows $x\ isolate\ S \longleftrightarrow (x{\in}S \wedge (\exists\, e{>}0.\ \forall\, y{\in}S.\ dist\ x\ y < e \longrightarrow y{=}x))$
⟨*proof*⟩

lemma *discrete-empty*[*simp*]: *discrete* {}
⟨*proof*⟩

lemma *uniform-discrete-empty*[*simp*]: *uniform-discrete* {}
⟨*proof*⟩

lemma *isolate-insert*:
fixes $x :: 'a{::}t1\text{-}space$
shows $x\ isolate\ (insert\ a\ S) \longleftrightarrow x\ isolate\ S \vee (x{=}a \wedge \neg\ (x\ islimpt\ S))$
⟨*proof*⟩

lemma *uniform-discrete-imp-closed*:
uniform-discrete S $\Longrightarrow$ *closed S*
⟨*proof*⟩

lemma *uniform-discrete-imp-discrete*:
uniform-discrete S $\Longrightarrow$ *discrete S*
⟨*proof*⟩

lemma *isolate-subset*:$x\ isolate\ S \Longrightarrow T \subseteq S \Longrightarrow x{\in}T \Longrightarrow x\ isolate\ T$
⟨*proof*⟩

lemma *discrete-subset*[*elim*]: *discrete S* $\Longrightarrow$ $T \subseteq S$ $\Longrightarrow$ *discrete T*
⟨*proof*⟩

lemma *uniform-discrete-subset*[*elim*]: *uniform-discrete S* $\Longrightarrow$ $T \subseteq S$ $\Longrightarrow$ *uniform-discrete T*
⟨*proof*⟩

lemma *continuous-on-discrete*: *discrete S* $\Longrightarrow$ *continuous-on S f*
⟨*proof*⟩

lemma *uniform-discrete-insert*:
fixes $S :: 'a{::}euclidean\text{-}space\ set$
shows *uniform-discrete* $(insert\ a\ S) \longleftrightarrow$ *uniform-discrete S*
⟨*proof*⟩

lemma *discrete-compact-finite-iff*:

fixes S :: $'a$::*t1-space set*
shows *discrete* $S \wedge$ *compact* $S \longleftrightarrow$ *finite* S
⟨*proof*⟩

lemma *uniform-discrete-finite-iff*:
fixes S :: $'a$::*heine-borel set*
shows *uniform-discrete* $S \wedge$ *bounded* $S \longleftrightarrow$ *finite* S
⟨*proof*⟩

lemma *uniform-discrete-image-scale*:
fixes f :: $'a$::*euclidean-space* $\Rightarrow$ $'b$::*euclidean-space*
assumes *uniform-discrete* S **and** *dist*:$\forall\, x{\in}S.\ \forall\, y{\in}S.$ *dist* $x\ y = c * dist\ (f\ x)\ (f\ y)$
shows *uniform-discrete* $(f\ `\ S)$
⟨*proof*⟩

end

2 Some useful lemmas in algebra

theory *Missing-Algebraic* **imports**
HOL−Computational-Algebra.Polynomial-Factorial
HOL−Computational-Algebra.Fundamental-Theorem-Algebra
HOL−Complex-Analysis.Complex-Analysis
Missing-Topology
Budan-Fourier.BF-Misc
begin

2.1 Misc

lemma *poly-holomorphic-on*[*simp*]:
(*poly p*) *holomorphic-on s*
⟨*proof*⟩

lemma *order-zorder*:
fixes p::*complex poly* **and** z::*complex*
assumes $p{\neq}0$
shows *order z p* = *nat* (*zorder* (*poly p*) *z*)
⟨*proof*⟩

lemma *pcompose-pCons-0*:*pcompose p* [:*a*:] = [:*poly p a*:]
⟨*proof*⟩

lemma *pcompose-coeff-0*:
coeff (*pcompose p q*) *0* = *poly p* (*coeff q 0*)
⟨*proof*⟩

lemma *poly-field-differentiable-at*[*simp*]:
poly p field-differentiable (*at x within s*)
⟨*proof*⟩

lemma *deriv-pderiv*:
deriv (*poly p*) = *poly* (*pderiv p*)
⟨*proof*⟩

lemma *lead-coeff-map-poly-nz*:
assumes *f* (*lead-coeff p*) ≠*0 f 0*=*0*
shows *lead-coeff* (*map-poly f p*) = *f* (*lead-coeff p*)
⟨*proof*⟩

lemma *filterlim-poly-at-infinity*:
fixes *p*::′*a*::*real-normed-field poly*
assumes *degree p*>*0*
shows *filterlim* (*poly p*) *at-infinity at-infinity*
⟨*proof*⟩

lemma *poly-divide-tendsto-aux*:
fixes *p*::′*a*::*real-normed-field poly*
shows ((λ*x*. *poly p x*/*x*^(*degree p*)) ⟶ *lead-coeff p*) *at-infinity*
⟨*proof*⟩

lemma *filterlim-power-at-infinity*:
assumes *n*≠*0*
shows *filterlim* (λ*x*::′*a*::*real-normed-field*. *x*^*n*) *at-infinity at-infinity*
⟨*proof*⟩

lemma *poly-divide-tendsto-0-at-infinity*:
fixes *p*::′*a*::*real-normed-field poly*
assumes *degree p* > *degree q*
shows ((λ*x*. *poly q x* / *poly p x*) ⟶ *0*) *at-infinity*
⟨*proof*⟩

lemma *lead-coeff-list-def*:
lead-coeff p= (*if coeffs p*=[] *then 0 else last* (*coeffs p*))
⟨*proof*⟩

lemma *poly-linepath-comp*:
fixes *a*::′*a*::{*real-normed-vector*,*comm-semiring-0*,*real-algebra-1*}
shows *poly p o* (*linepath a b*) = *poly* (*p* ∘$_p$ [:*a*, *b*−*a*:]) *o of-real*
⟨*proof*⟩

lemma *poly-eventually-not-zero*:
fixes *p*::*real poly*
assumes *p*≠*0*
shows *eventually* (λ*x*. *poly p x*≠*0*) *at-infinity*
⟨*proof*⟩

2.2 More about *degree*

lemma *degree-div-less*:
  fixes *x y*::*'a*::*field poly*
  assumes *degree x*≠*0 degree y*≠*0*
  shows *degree* (*x div y*) < *degree x*
⟨*proof*⟩

lemma *map-poly-degree-eq*:
  assumes *f* (*lead-coeff p*) ≠*0*
  shows *degree* (*map-poly f p*) = *degree p*
  ⟨*proof*⟩

lemma *map-poly-degree-less*:
  assumes *f* (*lead-coeff p*) =*0 degree p*≠*0*
  shows *degree* (*map-poly f p*) < *degree p*
⟨*proof*⟩

lemma *map-poly-degree-leq*[*simp*]:
  shows *degree* (*map-poly f p*) ≤ *degree p*
  ⟨*proof*⟩

2.3 roots / zeros of a univariate function

definition *roots-within*::(*'a* ⇒ *'b*::*zero*) ⇒ *'a set* ⇒ *'a set* **where**
  roots-within f s = {*x*∈*s*. *f x* = *0*}

abbreviation *roots*::(*'a* ⇒ *'b*::*zero*) ⇒ *'a set* **where**
  roots f ≡ *roots-within f UNIV*

2.4 The argument principle specialised to polynomials.

lemma *argument-principle-poly*:
  assumes *p*≠*0* **and** *valid*:*valid-path g* **and** *loop*: *pathfinish g* = *pathstart g*
    and *no-proots*:*path-image g* ⊆ − *proots p*
  shows *contour-integral g* (λ*x*. *deriv* (*poly p*) *x* / *poly p x*) = *2* ∗ *of-real pi* ∗ i ∗
          (∑ *x*∈*proots p*. *winding-number g x* ∗ *of-nat* (*order x p*))
⟨*proof*⟩

end

3 Some useful lemmas about transcendental functions

theory *Missing-Transcendental* **imports**
  Missing-Topology
  Missing-Algebraic
begin

3.1 Misc

lemma *Im-Ln-eq-pi-half*:
$z \neq 0 \Longrightarrow (Im(Ln\ z) = pi/2 \longleftrightarrow 0 < Im(z) \wedge Re(z) = 0)$
$z \neq 0 \Longrightarrow (Im(Ln\ z) = -pi/2 \longleftrightarrow Im(z) < 0 \wedge Re(z) = 0)$
⟨*proof*⟩

lemma *Im-Ln-eq*:
assumes *z≠0*
shows *Im (Ln z) = (if Re z≠0 then*
if Re z>0 then
arctan (Im z/Re z)
else if Im z≥0 then
arctan (Im z/Re z) + pi
else
arctan (Im z/Re z) − pi
else
if Im z>0 then pi/2 else −pi/2)
⟨*proof*⟩

lemma *exp-Arg2pi2pi-multivalue*:
assumes *exp* (i∗ *of-real x*) = *z*
shows ∃ *k*::*int*. *x* = *Arg2pi z* + *2*∗*k*∗*pi*
⟨*proof*⟩

lemma *cos-eq-neg-periodic-intro*:
assumes *x* − *y*=*2*∗(*of-int k*)∗*pi* + *pi* ∨ *x* + *y* = *2*∗(*of-int k*)∗*pi* + *pi*
shows *cos x* = − *cos y* ⟨*proof*⟩

lemma *cos-eq-periodic-intro*:
assumes *x* − *y*=*2*∗(*of-int k*)∗*pi* ∨ *x* + *y* = *2*∗(*of-int k*)∗*pi*
shows *cos x* = *cos y* ⟨*proof*⟩

lemma *sin-tan-half*: *sin* (*2*∗*x*) = *2* ∗ *tan x* / (*1* + (*tan x*) ^ *2*)
⟨*proof*⟩

lemma *cos-tan-half*: *cos x* ≠*0* ⟹ *cos* (*2*∗*x*) = (*1* − (*tan x*) ^ *2*) / (*1*+ (*tan x*) ^ *2*)
⟨*proof*⟩

lemma *tan-eq-arctan-Ex*:
shows *tan x* = *y* ⟷ (∃ *k*::*int*. *x* = *arctan y* + *k*∗*pi* ∨ (*x* = *pi*/*2* + *k*∗*pi* ∧ *y*=*0*))
⟨*proof*⟩

lemma *arccos-unique*:
assumes *0* ≤ *x*
and *x* ≤ *pi*
and *cos x* = *y*
shows *arccos y* = *x*

⟨*proof*⟩

lemma *cos-eq-arccos-Ex*:
cos x = y ⟷ −1≤y ∧ y≤1 ∧ (∃ k::int. x = arccos y + 2∗k∗pi ∨ x = − arccos y + 2∗k∗pi)
⟨*proof*⟩

lemma *uniform-discrete-tan-eq*:
uniform-discrete {x::real. tan x = y}
⟨*proof*⟩

lemma *get-norm-value*:
fixes *a::'a::{floor-ceiling}*
assumes *pp>0*
obtains *k::int* **and** *a1* **where** *a=(of-int k)∗pp+a1 a0≤a1 a1<a0+pp*
⟨*proof*⟩

lemma *filtermap-tan-at-right*:
fixes *a::real*
assumes *cos a≠0*
shows *filtermap tan (at-right a) = at-right (tan a)*
⟨*proof*⟩

lemma *filtermap-tan-at-left*:
fixes *a::real*
assumes *cos a≠0*
shows *filtermap tan (at-left a) = at-left (tan a)*
⟨*proof*⟩

lemma *cos-zero-iff-int2*:
fixes *x::real*
shows *cos x = 0 ⟷ (∃ n::int. x = n ∗ pi + pi/2)*
⟨*proof*⟩

lemma *filtermap-tan-at-right-inf*:
fixes *a::real*
assumes *cos a=0*
shows *filtermap tan (at-right a) = at-bot*
⟨*proof*⟩

lemma *filtermap-tan-at-left-inf*:
fixes *a::real*
assumes *cos a=0*
shows *filtermap tan (at-left a) = at-top*
⟨*proof*⟩

3.2 Periodic set

definition *periodic-set*:: *real set* ⇒ *real* ⇒ *bool* **where**
periodic-set S δ ⟷ (∃ *B*. *finite B* ∧ (∀ *x*∈*S*. ∃ *b*∈*B*. ∃ *k*::*int*. *x* = *b* + *k* ∗ *δ*))

lemma *periodic-set-multiple*:
assumes *k*≠*0*
shows *periodic-set S δ* ⟷ *periodic-set S* (*of-int k*∗*δ*)
⟨*proof*⟩

lemma *periodic-set-empty*[*simp*]: *periodic-set* {} *δ*
⟨*proof*⟩

lemma *periodic-set-finite*:
assumes *finite S*
shows *periodic-set S δ*
⟨*proof*⟩

lemma *periodic-set-subset*[*elim*]:
assumes *periodic-set S δ T* ⊆ *S*
shows *periodic-set T δ*
⟨*proof*⟩

lemma *periodic-set-union*:
assumes *periodic-set S δ periodic-set T δ*
shows *periodic-set* (*S* ∪ *T*) *δ*
⟨*proof*⟩

lemma *periodic-imp-uniform-discrete*:
assumes *periodic-set S δ*
shows *uniform-discrete S*
⟨*proof*⟩

lemma *periodic-set-tan-linear*:
assumes *a*≠*0 c*≠*0*
shows *periodic-set* (*roots* (λ*x*. *a*∗*tan* (*x*/*c*) + *b*)) (*c*∗*pi*)
⟨*proof*⟩

lemma *periodic-set-cos-linear*:
assumes *a*≠*0 c*≠*0*
shows *periodic-set* (*roots* (λ*x*. *a*∗*cos* (*x*/*c*) + *b*)) (*2*∗*c*∗*pi*)
⟨*proof*⟩

lemma *periodic-set-tan-poly*:
assumes *p*≠*0 c*≠*0*
shows *periodic-set* (*roots* (λ*x*. *poly p* (*tan* (*x*/*c*)))) (*c*∗*pi*)
⟨*proof*⟩

lemma *periodic-set-sin-cos-linear*:
fixes *a b c* ::*real*

assumes $a{\neq}0 \vee b{\neq}0 \vee c{\neq}0$
shows *periodic-set* (*roots* ($\lambda x.\ a * cos\ x + b * sin\ x + c$)) ($4{*}pi$)
⟨*proof*⟩

end

4 Some useful lemmas in analysis

theory *Missing-Analysis*
imports *HOL−Complex-Analysis.Complex-Analysis*
begin

4.1 More about paths

lemma *pathfinish-offset*[*simp*]:
pathfinish ($\lambda t.\ g\ t - z$) = *pathfinish* $g - z$
⟨*proof*⟩

lemma *pathstart-offset*[*simp*]:
pathstart ($\lambda t.\ g\ t - z$) = *pathstart* $g - z$
⟨*proof*⟩

lemma *pathimage-offset*[*simp*]:
fixes g :: - ⇒ $'b$::*topological-group-add*
shows $p \in$ *path-image* ($\lambda t.\ g\ t - z$) ⟷ $p{+}z \in$ *path-image* g
⟨*proof*⟩

lemma *path-offset*[*simp*]:
fixes g :: - ⇒ $'b$::*topological-group-add*
shows *path* ($\lambda t.\ g\ t - z$) ⟷ *path* g
⟨*proof*⟩

lemma *not-on-circlepathI*:
assumes *cmod* ($z{-}z0$) ≠ $|r|$
shows $z \notin$ *path-image* (*part-circlepath z0 r st tt*)
⟨*proof*⟩

lemma *circlepath-inj-on*:
assumes $r{>}0$
shows *inj-on* (*circlepath z r*) $\{0..{<}1\}$
⟨*proof*⟩

4.2 More lemmas related to *winding-number*

lemma *winding-number-comp*:
assumes *open s f holomorphic-on s path-image* $\gamma \subseteq s$
valid-path γ $z \notin$ *path-image* ($f \circ \gamma$)
shows *winding-number* ($f \circ \gamma$) z = $1/(2{*}pi{*}\mathrm{i}){*}$ *contour-integral* γ ($\lambda w.$ *deriv f* w / ($f\ w - z$))

⟨*proof*⟩

lemma *winding-number-uminus-comp*:
assumes *valid-path* γ $-$ $z \notin$ *path-image* γ
shows *winding-number* (*uminus* $\circ$ γ) z = *winding-number* γ ($-z$)
⟨*proof*⟩

lemma *winding-number-comp-linear*:
assumes $c{\neq}0$ *valid-path* γ **and** *not-image*: $(z-b)/c \notin$ *path-image* γ
shows *winding-number* ((λx. $c{*}x{+}b$) $\circ$ γ) z = *winding-number* γ (($z-b$)/c) (**is** *?L* = *?R*)
⟨*proof*⟩

end

5 Cauchy's index theorem

theory *Cauchy-Index-Theorem* **imports**
HOL−Complex-Analysis.Complex-Analysis
Sturm-Tarski.Sturm-Tarski
HOL−Computational-Algebra.Fundamental-Theorem-Algebra
Missing-Transcendental
Missing-Algebraic
Missing-Analysis
begin

This theory formalises Cauchy indices on the complex plane and relate them to winding numbers

5.1 Misc

lemma *atMostAtLeast-subset-convex*:
fixes C :: *real set*
assumes *convex* C
and $x \in C$ $y \in C$
shows {x .. y} $\subseteq$ C
⟨*proof*⟩

lemma *arg-elim*:
f x ⟹ $x{=}$ y ⟹ *f y*
⟨*proof*⟩

lemma *arg-elim2*:
f x1 x2 ⟹ *x1*= *y1* ⟹*x2*=*y2* ⟹ *f y1 y2*
⟨*proof*⟩

lemma *arg-elim3*:
⟦*f x1 x2 x3*;*x1*= *y1*;*x2*=*y2*;*x3*=*y3* ⟧ ⟹ *f y1 y2 y3*
⟨*proof*⟩

lemma *IVT-strict*:
fixes $f :: 'a::linear\text{-}continuum\text{-}topology \Rightarrow 'b::linorder\text{-}topology$
assumes $(f\ a > y \wedge y > f\ b) \vee (f\ a < y \wedge y < f\ b)$ $a<b$ *continuous-on* $\{a\ ..\ b\}$ *f*
shows $\exists x.\ a < x \wedge x < b \wedge f\ x = y$
⟨*proof*⟩

lemma (**in** *dense-linorder*) *atLeastAtMost-subseteq-greaterThanLessThan-iff*:
$\{a\ ..\ b\} \subseteq \{\ c\ <..<\ d\ \} \longleftrightarrow (a \leq b \longrightarrow c < a \wedge b < d)$
⟨*proof*⟩

lemma *Re-winding-number-half-right*:
assumes $\forall\, p{\in}$*path-image* γ. *Re* $p \geq$ *Re* z **and** *valid-path* γ **and** $z{\notin}$*path-image* γ
shows *Re*(*winding-number* γ z) = (*Im* (*Ln* (*pathfinish* γ − z)) − *Im* (*Ln* (*pathstart* γ − z)))/(2∗*pi*)
⟨*proof*⟩

lemma *Re-winding-number-half-upper*:
assumes *pimage*:$\forall\, p{\in}$*path-image* γ. *Im* $p \geq$ *Im* z **and** *valid-path* γ **and** $z{\notin}$*path-image* γ
shows *Re*(*winding-number* γ z) =
(*Im* (*Ln* (i∗z − i∗*pathfinish* γ)) − *Im* (*Ln* (i∗z − i∗*pathstart* γ)))/(2∗*pi*)
⟨*proof*⟩

lemma *Re-winding-number-half-lower*:
assumes *pimage*:$\forall\, p{\in}$*path-image* γ. *Im* $p \leq$ *Im* z **and** *valid-path* γ **and** $z{\notin}$*path-image* γ
shows *Re*(*winding-number* γ z) =
(*Im* (*Ln* (i∗*pathfinish* γ − i∗z)) − *Im* (*Ln* (i∗*pathstart* γ − i∗z)))/(2∗*pi*)
⟨*proof*⟩

lemma *Re-winding-number-half-left*:
assumes *neg-img*:$\forall\, p{\in}$*path-image* γ. *Re* $p \leq$ *Re* z **and** *valid-path* γ **and** $z{\notin}$*path-image* γ
shows *Re*(*winding-number* γ z) = (*Im* (*Ln* (z − *pathfinish* γ)) − *Im* (*Ln* (z − *pathstart* γ)))/(2∗*pi*)
⟨*proof*⟩

lemma *continuous-on-open-Collect-neq*:
fixes $f\ g :: 'a::topological\text{-}space \Rightarrow 'b::t2\text{-}space$
assumes f: *continuous-on* S f **and** g: *continuous-on* S g **and** *open* S
shows *open* $\{x{\in}S.\ f\ x \neq g\ x\}$
⟨*proof*⟩

5.2 Sign at a filter

definition *has-sgnx*::(*real* ⇒ *real*) ⇒ *real* ⇒ *real filter* ⇒ *bool*
(**infixr** *has′-sgnx 55*) **where**
(*f has-sgnx c*) *F*= (*eventually* (λ*x*. *sgn*(*f x*) = *c*) *F*)

definition *sgnx-able* (**infixr** *sgnx′-able 55*) **where**
(*f sgnx-able F*) = (∃ *c*. (*f has-sgnx c*) *F*)

definition *sgnx* **where**
sgnx f F = (*SOME c*. (*f has-sgnx c*) *F*)

lemma *has-sgnx-eq-rhs*: (*f has-sgnx x*) *F* ⟹ *x* = *y* ⟹ (*f has-sgnx y*) *F*
⟨*proof*⟩

named-theorems *sgnx-intros introduction rules for has-sgnx*
⟨*ML*⟩

lemma *sgnx-able-sgnx*:*f sgnx-able F* ⟹ (*f has-sgnx* (*sgnx f F*)) *F*
⟨*proof*⟩

lemma *has-sgnx-imp-sgnx-able*[*elim*]:
(*f has-sgnx c*) *F* ⟹ *f sgnx-able F*
⟨*proof*⟩

lemma *has-sgnx-unique*:
assumes *F*≠*bot* (*f has-sgnx c1*) *F* (*f has-sgnx c2*) *F*
shows *c1*=*c2*
⟨*proof*⟩

lemma *has-sgnx-imp-sgnx*[*elim*]:
(*f has-sgnx c*) *F* ⟹*F*≠*bot* ⟹ *sgnx f F* = *c*
⟨*proof*⟩

lemma *has-sgnx-const*[*simp*,*sgnx-intros*]:
((λ-. *c*) *has-sgnx sgn c*) *F*
⟨*proof*⟩

lemma *finite-sgnx-at-left-at-right*:
assumes *finite* {*t*. *f t*=*0* ∧ *a*<*t* ∧ *t*<*b*} *continuous-on* ({*a*<..<*b*} − *s*) *f finite s*
and *x*:*x*∈{*a*<..<*b*}
shows *f sgnx-able* (*at-left x*) *sgnx f* (*at-left x*)≠*0*
f sgnx-able (*at-right x*) *sgnx f* (*at-right x*)≠*0*
⟨*proof*⟩

lemma *sgnx-able-poly*[*simp*]:
(*poly p*) *sgnx-able* (*at-right a*)
(*poly p*) *sgnx-able* (*at-left a*)
(*poly p*) *sgnx-able at-top*

(*poly p*) *sgnx-able at-bot*
⟨*proof*⟩

lemma *has-sgnx-identity*[*intro*,*sgnx-intros*]:
shows *x*≥*0* ⟹((λ*x*. *x*) *has-sgnx 1*) (*at-right x*)
x≤*0* ⟹ ((λ*x*. *x*) *has-sgnx* −*1*) (*at-left x*)
⟨*proof*⟩

lemma *has-sgnx-divide*[*sgnx-intros*]:
assumes (*f has-sgnx c1*) *F* (*g has-sgnx c2*) *F*
shows ((λ*x*. *f x* / *g x*) *has-sgnx c1* / *c2*) *F*
⟨*proof*⟩

lemma *sgnx-able-divide*[*sgnx-intros*]:
assumes *f sgnx-able F g sgnx-able F*
shows (λ*x*. *f x* / *g x*) *sgnx-able F*
⟨*proof*⟩

lemma *sgnx-divide*:
assumes *F*≠*bot f sgnx-able F g sgnx-able F*
shows *sgnx* (λ*x*. *f x* / *g x*) *F* =*sgnx f F* / *sgnx g F*
⟨*proof*⟩

lemma *has-sgnx-times*[*sgnx-intros*]:
assumes (*f has-sgnx c1*) *F* (*g has-sgnx c2*) *F*
shows ((λ*x*. *f x*∗ *g x*) *has-sgnx c1* ∗ *c2*) *F*
⟨*proof*⟩

lemma *sgnx-able-times*[*sgnx-intros*]:
assumes *f sgnx-able F g sgnx-able F*
shows (λ*x*. *f x* ∗ *g x*) *sgnx-able F*
⟨*proof*⟩

lemma *sgnx-times*:
assumes *F*≠*bot f sgnx-able F g sgnx-able F*
shows *sgnx* (λ*x*. *f x* ∗ *g x*) *F* =*sgnx f F* ∗ *sgnx g F*
⟨*proof*⟩

lemma *tendsto-nonzero-has-sgnx*:
assumes (*f* ⟶ *c*) *F c*≠*0*
shows (*f has-sgnx sgn c*) *F*
⟨*proof*⟩

lemma *tendsto-nonzero-sgnx*:
assumes (*f* ⟶ *c*) *F F*≠*bot c*≠*0*
shows *sgnx f F* = *sgn c*
⟨*proof*⟩

lemma *filterlim-divide-at-bot-at-top-iff*:
assumes $(f \longrightarrow c)\ F\ c{\neq}0$
shows
$(LIM\ x\ F.\ f\ x\ /\ g\ x :>\ at\text{-}bot) \longleftrightarrow (g \longrightarrow 0)\ F$
$\wedge\ ((\lambda x.\ g\ x)\ has\text{-}sgnx\ -\ sgn\ c)\ F$
$(LIM\ x\ F.\ f\ x\ /\ g\ x :>\ at\text{-}top) \longleftrightarrow (g \longrightarrow 0)\ F$
$\wedge\ ((\lambda x.\ g\ x)\ has\text{-}sgnx\ sgn\ c)\ F$
⟨*proof*⟩

lemma *poly-sgnx-left-right*:
fixes $c\ a{::}real$ **and** $p{::}real\ poly$
assumes $p{\neq}0$
shows $sgnx\ (poly\ p)\ (at\text{-}left\ a) = (if\ even\ (order\ a\ p)$
$then\ sgnx\ (poly\ p)\ (at\text{-}right\ a)$
$else\ -sgnx\ (poly\ p)\ (at\text{-}right\ a))$
⟨*proof*⟩

lemma *poly-has-sgnx-left-right*:
fixes $c\ a{::}real$ **and** $p{::}real\ poly$
assumes $p{\neq}0$
shows $(poly\ p\ has\text{-}sgnx\ c)\ (at\text{-}left\ a) \longleftrightarrow (if\ even\ (order\ a\ p)$
$then\ (poly\ p\ has\text{-}sgnx\ c)\ (at\text{-}right\ a)$
$else\ (poly\ p\ has\text{-}sgnx\ -c)\ (at\text{-}right\ a))$
⟨*proof*⟩

lemma *sign-r-pos-sgnx-iff*:
$sign\text{-}r\text{-}pos\ p\ a \longleftrightarrow sgnx\ (poly\ p)\ (at\text{-}right\ a) > 0$
⟨*proof*⟩

lemma *sgnx-values*:
assumes $f\ sgnx\text{-}able\ F\ F \neq bot$
shows $sgnx\ f\ F = -1 \vee sgnx\ f\ F = 0 \vee sgnx\ f\ F = 1$
⟨*proof*⟩

lemma *has-sgnx-poly-at-top*:
$(poly\ p\ has\text{-}sgnx\ \ sgn\text{-}pos\text{-}inf\ p)\ at\text{-}top$
⟨*proof*⟩

lemma *has-sgnx-poly-at-bot*:
$(poly\ p\ has\text{-}sgnx\ \ sgn\text{-}neg\text{-}inf\ p)\ at\text{-}bot$
⟨*proof*⟩

lemma *sgnx-poly-at-top*:
$sgnx\ (poly\ p)\ at\text{-}top = sgn\text{-}pos\text{-}inf\ p$
⟨*proof*⟩

lemma *sgnx-poly-at-bot*:

sgnx (poly p) at-bot = sgn-neg-inf p
⟨*proof*⟩

lemma *poly-has-sgnx-values*:
assumes *p≠0*
shows
(poly p has-sgnx 1) (at-left a) ∨ (poly p has-sgnx − 1) (at-left a)
(poly p has-sgnx 1) (at-right a) ∨ (poly p has-sgnx − 1) (at-right a)
(poly p has-sgnx 1) at-top ∨ (poly p has-sgnx − 1) at-top
(poly p has-sgnx 1) at-bot ∨ (poly p has-sgnx − 1) at-bot
⟨*proof*⟩

lemma *poly-sgnx-values*:
assumes *p≠0*
shows *sgnx (poly p) (at-left a) = 1 ∨ sgnx (poly p) (at-left a) = −1*
sgnx (poly p) (at-right a) = 1 ∨ sgnx (poly p) (at-right a) = −1
⟨*proof*⟩

lemma *has-sgnx-inverse*: *(f has-sgnx c) F ⟷ ((inverse o f) has-sgnx (inverse c)) F*
⟨*proof*⟩

lemma *has-sgnx-derivative-at-left*:
assumes *g-deriv*:*(g has-field-derivative c) (at x)* **and** *g x=0* **and** *c≠0*
shows *(g has-sgnx − sgn c) (at-left x)*
⟨*proof*⟩

lemma *has-sgnx-derivative-at-right*:
assumes *g-deriv*:*(g has-field-derivative c) (at x)* **and** *g x=0* **and** *c≠0*
shows *(g has-sgnx sgn c) (at-right x)*
⟨*proof*⟩

lemma *has-sgnx-split*:
(f has-sgnx c) (at x) ⟷ (f has-sgnx c) (at-left x) ∧ (f has-sgnx c) (at-right x)
⟨*proof*⟩

lemma *sgnx-at-top-IVT*:
assumes *sgnx (poly p) (at-right a) ≠ sgnx (poly p) at-top*
shows *∃ x>a. poly p x=0*
⟨*proof*⟩

lemma *sgnx-at-left-at-right-IVT*:
assumes *sgnx (poly p) (at-right a) ≠ sgnx (poly p) (at-left b) a<b*
shows *∃ x. a<x ∧ x<b ∧ poly p x=0*
⟨*proof*⟩

lemma *sgnx-at-bot-IVT*:
assumes *sgnx (poly p) (at-left a) ≠ sgnx (poly p) at-bot*

shows $\exists x{<}a.\ poly\ p\ x{=}0$
⟨*proof*⟩

lemma *sgnx-poly-nz*:
assumes *poly p x*≠*0*
shows *sgnx* (*poly p*) (*at-left x*) = *sgn* (*poly p x*)
sgnx (*poly p*) (*at-right x*) = *sgn* (*poly p x*)
⟨*proof*⟩

5.3 Finite predicate segments over an interval

inductive *finite-Psegments*::(*real* ⇒ *bool*) ⇒ *real* ⇒ *real* ⇒ *bool* **for** *P* **where**
emptyI: *a*≥*b* ⟹ *finite-Psegments P a b*|
insertI-1: ⟦*s*∈{*a*..<*b*};*s*=*a*∨*P s*;∀ *t*∈{*s*<..<*b*}. *P t*; *finite-Psegments P a s*⟧
⟹ *finite-Psegments P a b*|
insertI-2: ⟦*s*∈{*a*..<*b*};*s*=*a*∨*P s*;(∀ *t*∈{*s*<..<*b*}. ¬*P t*);*finite-Psegments P a s*⟧
⟹ *finite-Psegments P a b*

lemma *finite-Psegments-pos-linear*:
assumes *finite-Psegments P* (*b*∗*lb*+*c*) (*b*∗*ub*+*c*) **and** *b*>*0*
shows *finite-Psegments* (*P o* (λ*t*. *b*∗*t*+*c*)) *lb ub*
⟨*proof*⟩

lemma *finite-Psegments-congE*:
assumes *finite-Psegments Q lb ub*
⋀*t*. ⟦*lb*<*t*;*t*<*ub*⟧ ⟹ *Q t* ⟷ *P t*
shows *finite-Psegments P lb ub* ⟨*proof*⟩

lemma *finite-Psegments-constI*:
assumes ⋀*t*. ⟦*a*<*t*;*t*<*b*⟧ ⟹ *P t* = *c*
shows *finite-Psegments P a b*
⟨*proof*⟩

context
begin

private lemma *finite-Psegments-less-eq1*:
assumes *finite-Psegments P a c b*≤*c*
shows *finite-Psegments P a b* ⟨*proof*⟩ **lemma** *finite-Psegments-less-eq2*:
assumes *finite-Psegments P a c a*≤*b*
shows *finite-Psegments P b c* ⟨*proof*⟩

lemma *finite-Psegments-included*:
assumes *finite-Psegments P a d a*≤*b c*≤*d*
shows *finite-Psegments P b c*
⟨*proof*⟩
end

lemma *finite-Psegments-combine*:
assumes *finite-Psegments P a b finite-Psegments P b c* $b \in \{a..c\}$ *closed* $(\{x.\ P\ x\} \cap \{a..c\})$
shows *finite-Psegments P a c* ⟨*proof*⟩

5.4 Finite segment intersection of a path with the imaginary axis

definition *finite-ReZ-segments*::(*real* ⇒ *complex*) ⇒ *complex* ⇒ *bool* **where**
finite-ReZ-segments g z = *finite-Psegments* $(\lambda t.\ Re\ (g\ t - z) = 0)\ 0\ 1$

lemma *finite-ReZ-segments-joinpaths*:
assumes *g1*:*finite-ReZ-segments g1 z* **and** *g2*: *finite-ReZ-segments g2 z* **and**
path g1 path g2 pathfinish g1=*pathstart g2*
shows *finite-ReZ-segments* $(g1+{+}+g2)\ z$
⟨*proof*⟩

lemma *finite-ReZ-segments-congE*:
assumes *finite-ReZ-segments p1 z1*
$\bigwedge t.\ [\![0<t;t<1]\!] \Longrightarrow\ Re(p1\ t-\ z1) = Re(p2\ t\ -\ z2)$
shows *finite-ReZ-segments p2 z2*
⟨*proof*⟩

lemma *finite-ReZ-segments-constI*:
assumes $\forall\, t.\ 0<t \wedge t<1 \longrightarrow g\ t = c$
shows *finite-ReZ-segments g z*
⟨*proof*⟩

lemma *finite-ReZ-segment-cases* [*consumes 1*, *case-names subEq subNEq*,*cases pred*:*finite-ReZ-segments*]:
assumes *finite-ReZ-segments g z*
and *subEq*:$(\bigwedge s.\ [\![s \in \{0..<1\};s=0 \vee Re\ (g\ s) = Re\ z;$
$\forall\, t \in \{s<..<1\}.\ Re\ (g\ t) = Re\ z;$*finite-ReZ-segments* $(subpath\ 0\ s\ g)\ z]\!] \Longrightarrow P)$
and *subNEq*:$(\bigwedge s.\ [\![s \in \{0..<1\};s=0 \vee Re\ (g\ s) = Re\ z;$
$\forall\, t \in \{s<..<1\}.\ Re\ (g\ t) \neq Re\ z;$*finite-ReZ-segments* $(subpath\ 0\ s\ g)\ z]\!] \Longrightarrow P)$
shows *P*
⟨*proof*⟩

lemma *finite-ReZ-segments-induct* [*case-names sub0 subEq subNEq*, *induct pred*:*finite-ReZ-segments*]:
assumes *finite-ReZ-segments g z*
assumes *sub0*:$\bigwedge g\ z.\ (P\ (subpath\ 0\ 0\ g)\ z)$
and *subEq*:$(\bigwedge s\ g\ z.\ [\![s \in \{0..<1\};s=0 \vee Re\ (g\ s) = Re\ z;$
$\forall\, t \in \{s<..<1\}.\ Re\ (g\ t) = Re\ z;$*finite-ReZ-segments* $(subpath\ 0\ s\ g)\ z;$
$P\ (subpath\ 0\ s\ g)\ z]\!] \Longrightarrow P\ g\ z)$
and *subNEq*:$(\bigwedge s\ g\ z.\ [\![s \in \{0..<1\};s=0 \vee Re\ (g\ s) = Re\ z;$
$\forall\, t \in \{s<..<1\}.\ Re\ (g\ t) \neq Re\ z;$*finite-ReZ-segments* $(subpath\ 0\ s\ g)\ z;$
$P\ (subpath\ 0\ s\ g)\ z]\!] \Longrightarrow P\ g\ z)$

shows $P\ g\ z$
⟨*proof*⟩

lemma *finite-ReZ-segments-shiftpah*:
assumes *finite-ReZ-segments g z s*∈{*0..1*} *path g* **and** *loop*:*pathfinish g* = *path-start g*
shows *finite-ReZ-segments* (*shiftpath s g*) *z*
⟨*proof*⟩

lemma *finite-imp-finite-ReZ-segments*:
assumes *finite* {*t*. *Re* (*g t* − *z*) = *0* ∧ *0* ≤ *t* ∧ *t*≤*1*}
shows *finite-ReZ-segments g z*
⟨*proof*⟩

lemma *finite-ReZ-segments-poly-linepath*:
shows *finite-ReZ-segments* (*poly p o linepath a b*) *z*
⟨*proof*⟩

lemma *part-circlepath-half-finite-inter*:
assumes *st*≠*tt r*≠*0 c*≠*0*
shows *finite* {*t*. *part-circlepath z0 r st tt t* · *c* = *d* ∧ *0*≤ *t* ∧ *t*≤*1*} (**is** *finite ?T*)
⟨*proof*⟩

lemma *linepath-half-finite-inter*:
assumes *a* · *c* ≠ *d* ∨ *b* · *c* ≠ *d*
shows *finite* {*t*. *linepath a b t* · *c* = *d* ∧ *0*≤ *t* ∧ *t*≤*1*} (**is** *finite ?S*)
⟨*proof*⟩

lemma *finite-half-joinpaths-inter*:
assumes *finite* {*t*. *l1 t* · *c* = *d* ∧ *0*≤ *t* ∧ *t*≤*1*} *finite* {*t*. *l2 t* · *c* = *d* ∧ *0*≤ *t* ∧ *t*≤*1*}
shows *finite* {*t*. (*l1*+++*l2*) *t* · *c* = *d* ∧ *0*≤ *t* ∧ *t*≤*1*}
⟨*proof*⟩

lemma *finite-ReZ-segments-linepath*:
finite-ReZ-segments (*linepath a b*) *z*
⟨*proof*⟩

lemma *finite-ReZ-segments-part-circlepath*:
finite-ReZ-segments (*part-circlepath z0 r st tt*) *z*
⟨*proof*⟩

lemma *finite-ReZ-segments-poly-of-real*:
shows *finite-ReZ-segments* (*poly p o of-real*) *z*
⟨*proof*⟩

lemma *finite-ReZ-segments-subpath*:
assumes *finite-ReZ-segments g z*
0≤*u u*≤*v v*≤*1*

shows *finite-ReZ-segments* (*subpath u v g*) *z*
⟨*proof*⟩

5.5 jump and jumpF

definition *jump*::(*real* ⇒ *real*) ⇒ *real* ⇒ *int* **where**
jump f a = (*if*
(*LIM x* (*at-left a*). *f x* :> *at-bot*) ∧ (*LIM x* (*at-right a*). *f x* :> *at-top*)
then 1 else if
(*LIM x* (*at-left a*). *f x* :> *at-top*) ∧ (*LIM x* (*at-right a*). *f x* :> *at-bot*)
then −*1 else 0*)

definition *jumpF*::(*real* ⇒ *real*) ⇒ *real filter* ⇒ *real* **where**
jumpF f F ≡ (*if filterlim f at-top F then 1*/*2 else*
if filterlim f at-bot F then −*1*/*2 else* (*0*::*real*))

lemma *jumpF-const*[*simp*]:
assumes *F*≠*bot*
shows *jumpF* (λ-. *c*) *F* = *0*
⟨*proof*⟩

lemma *jumpF-not-infinity*:
assumes *continuous F g F*≠*bot*
shows *jumpF g F* = *0*
⟨*proof*⟩

lemma *jumpF-linear-comp*:
assumes *c*≠*0*
shows
jumpF (*f o* (λ*x*. *c*∗*x*+*b*)) (*at-left x*) =
(*if c*>*0 then jumpF f* (*at-left* (*c*∗*x*+*b*)) *else jumpF f* (*at-right* (*c*∗*x*+*b*)))
(**is** *?case1*)
jumpF (*f o* (λ*x*. *c*∗*x*+*b*)) (*at-right x*) =
(*if c*>*0 then jumpF f* (*at-right* (*c*∗*x*+*b*)) *else jumpF f* (*at-left* (*c*∗*x*+*b*)))
(**is** *?case2*)
⟨*proof*⟩

lemma *jump-const*[*simp*]:*jump* (λ-. *c*) *a* = *0*
⟨*proof*⟩

lemma *jump-not-infinity*:
isCont f a ⟹ *jump f a* =*0*
⟨*proof*⟩

lemma *jump-jump-poly-aux*:
assumes *p*≠*0 coprime p q*
shows *jump* (λ*x*. *poly q x* / *poly p x*) *a* = *jump-poly q p a*
⟨*proof*⟩

lemma *jump-jumpF*:
assumes *cont*:*isCont* (*inverse o f*) *a* **and**
sgnxl:(*f has-sgnx l*) (*at-left a*) **and** *sgnxr*:(*f has-sgnx r*) (*at-right a*) **and**
l≠*0* *r*≠*0*
shows *jump f a* = *jumpF f* (*at-right a*) − *jumpF f* (*at-left a*)
⟨*proof*⟩

lemma *jump-linear-comp*:
assumes *c*≠*0*
shows *jump* (*f o* (λ*x*. *c*∗*x*+*b*)) *x* = (*if c*>*0 then jump f* (*c*∗*x*+*b*) *else* −*jump f* (*c*∗*x*+*b*))
⟨*proof*⟩

lemma *jump-divide-derivative*:
assumes *isCont f x g x* = *0 f x*≠*0*
and *g-deriv*:(*g has-field-derivative c*) (*at x*) **and** *c*≠*0*
shows *jump* (λ*t*. *f t*/*g t*) *x* = (*if sgn c* = *sgn* (*f x*) *then 1 else* −*1*)
⟨*proof*⟩

lemma *jump-jump-poly*: *jump* (λ*x*. *poly q x* / *poly p x*) *a* = *jump-poly q p a*
⟨*proof*⟩

lemma *jump-Im-divide-Re-0*:
assumes *path g Re* (*g x*)≠*0* *0*<*x x*<*1*
shows *jump* (λ*t*. *Im* (*g t*) / *Re* (*g t*)) *x* = *0*
⟨*proof*⟩

lemma *jumpF-im-divide-Re-0*:
assumes *path g Re* (*g x*)≠*0*
shows ⟦*0*≤*x*;*x*<*1*⟧ ⟹ *jumpF* (λ*t*. *Im* (*g t*) / *Re* (*g t*)) (*at-right x*) = *0*
⟦*0*<*x*;*x*≤*1*⟧ ⟹ *jumpF* (λ*t*. *Im* (*g t*) / *Re* (*g t*)) (*at-left x*) = *0*
⟨*proof*⟩

lemma *jump-cong*:
assumes *x*=*y* **and** *eventually* (λ*x*. *f x*=*g x*) (*at x*)
shows *jump f x* = *jump g y*
⟨*proof*⟩

lemma *jumpF-cong*:
assumes *F*=*G* **and** *eventually* (λ*x*. *f x*=*g x*) *F*
shows *jumpF f F* = *jumpF g G*
⟨*proof*⟩

lemma *jump-at-left-at-right-eq*:
assumes *isCont f x* **and** *f x* ≠ *0* **and** *sgnx-eq*:*sgnx g* (*at-left x*) = *sgnx g* (*at-right x*)
shows *jump* (λ*t*. *f t*/*g t*) *x* = *0*
⟨*proof*⟩

lemma *jumpF-pos-has-sgnx*:
assumes *jumpF f F > 0*
shows *(f has-sgnx 1) F*
⟨*proof*⟩

lemma *jumpF-neg-has-sgnx*:
assumes *jumpF f F < 0*
shows *(f has-sgnx −1) F*
⟨*proof*⟩

lemma *jumpF-IVT*:
fixes *f*::*real* ⇒ *real* **and** *a b*::*real*
defines *right*≡(λ(*R*::*real* ⇒ *real* ⇒ *bool*). *R* (*jumpF f* (*at-right a*)) *0*
∨ (*continuous* (*at-right a*) *f* ∧ *R* (*f a*) *0*))
and
left≡(λ(*R*::*real* ⇒ *real* ⇒ *bool*). *R* (*jumpF f* (*at-left b*)) *0*
∨ (*continuous* (*at-left b*) *f* ∧ *R* (*f b*) *0*))
assumes *a*<*b* **and** *cont*:*continuous-on* {*a*<..<*b*} *f* **and**
right-left:*right greater* ∧ *left less* ∨ *right less* ∧ *left greater*
shows ∃ *x*. *a*<*x* ∧ *x*<*b* ∧ *f x* =*0*
⟨*proof*⟩

lemma *jumpF-eventually-const*:
assumes *eventually* (λ*x*. *f x*=*c*) *F F*≠*bot*
shows *jumpF f F* = *0*
⟨*proof*⟩

lemma *jumpF-tan-comp*:
jumpF (*f o tan*) (*at-right x*) = (*if cos x* = *0*
then jumpF f at-bot else jumpF f (*at-right* (*tan x*)))
jumpF (*f o tan*) (*at-left x*) = (*if cos x* =*0*
then jumpF f at-top else jumpF f (*at-left* (*tan x*)))
⟨*proof*⟩

5.6 Finite jumpFs over an interval

definition *finite-jumpFs*::(*real* ⇒ *real*) ⇒ *real* ⇒ *real* ⇒ *bool* **where**
finite-jumpFs f a b = *finite* {*x*. (*jumpF f* (*at-left x*) ≠*0* ∨ *jumpF f* (*at-right x*)
≠*0*) ∧ *a*≤*x* ∧ *x*≤*b*}

lemma *finite-jumpFs-linear-pos*:
assumes *c*>*0*
shows *finite-jumpFs* (*f o* (λ*x*. *c* ∗ *x* + *b*)) *lb ub* ⟷ *finite-jumpFs f* (*c* ∗ *lb* +*b*)
(*c* ∗ *ub* + *b*)
⟨*proof*⟩

lemma *finite-jumpFs-consts*:

finite-jumpFs (λ- . *c*) *lb ub*
⟨*proof*⟩

lemma *finite-jumpFs-combine*:
assumes *finite-jumpFs f a b finite-jumpFs f b c*
shows *finite-jumpFs f a c*
⟨*proof*⟩

lemma *finite-jumpFs-subE*:
assumes *finite-jumpFs f a b a*≤*a′ b′*≤*b*
shows *finite-jumpFs f a′ b′*
⟨*proof*⟩

lemma *finite-Psegments-Re-imp-jumpFs*:
assumes *finite-Psegments* (λ*t*. *Re* (*g t* − *z*) = *0*) *a b continuous-on* {*a*..*b*} *g*
shows *finite-jumpFs* (λ*t*. *Im* (*g t* − *z*)/*Re* (*g t* − *z*)) *a b*
⟨*proof*⟩

lemma *finite-ReZ-segments-imp-jumpFs*:
assumes *finite-ReZ-segments g z path g*
shows *finite-jumpFs* (λ*t*. *Im* (*g t* − *z*)/*Re* (*g t* − *z*)) *0 1*
⟨*proof*⟩

5.7 *jumpF* at path ends

definition *jumpF-pathstart*::(*real* ⇒ *complex*) ⇒ *complex* ⇒ *real* **where**
jumpF-pathstart g z= *jumpF* (λ*t*. *Im*(*g t*− *z*)/*Re*(*g t* − *z*)) (*at-right 0*)

definition *jumpF-pathfinish*::(*real* ⇒ *complex*) ⇒ *complex* ⇒ *real* **where**
jumpF-pathfinish g z= *jumpF* (λ*t*. *Im*(*g t* − *z*)/*Re*(*g t* −*z*)) (*at-left 1*)

lemma *jumpF-pathstart-eq-0*:
assumes *path g Re*(*pathstart g*)≠*Re z*
shows *jumpF-pathstart g z* = *0*
⟨*proof*⟩

lemma *jumpF-pathfinish-eq-0*:
assumes *path g Re*(*pathfinish g*)≠*Re z*
shows *jumpF-pathfinish g z* = *0*
⟨*proof*⟩

lemma
shows *jumpF-pathfinish-reversepath*: *jumpF-pathfinish* (*reversepath g*) *z* = *jumpF-pathstart g z*
and *jumpF-pathstart-reversepath*: *jumpF-pathstart* (*reversepath g*) *z* = *jumpF-pathfinish g z*
⟨*proof*⟩

lemma *jumpF-pathstart-joinpaths*[*simp*]:

jumpF-pathstart (g1+++g2) z = jumpF-pathstart g1 z
⟨*proof*⟩

lemma *jumpF-pathfinish-joinpaths*[*simp*]:
jumpF-pathfinish (g1+++g2) z = jumpF-pathfinish g2 z
⟨*proof*⟩

5.8 Cauchy index

— Deprecated, use "cindexE" if possible
definition *cindex*::*real* ⇒ *real* ⇒ (*real* ⇒ *real*) ⇒ *int* **where**
cindex a b f = (∑ *x*∈{*x*. *jump f x*≠*0* ∧ *a*<*x* ∧ *x*<*b*}. *jump f x*)

definition *cindexE*::*real* ⇒ *real* ⇒ (*real* ⇒ *real*) ⇒ *real* **where**
cindexE a b f = (∑ *x*∈{*x*. *jumpF f* (*at-right x*) ≠*0* ∧ *a*≤*x* ∧ *x*<*b*}. *jumpF f* (*at-right x*))
− (∑ *x*∈{*x*. *jumpF f* (*at-left x*) ≠*0* ∧ *a*<*x* ∧ *x*≤*b*}. *jumpF f* (*at-left x*))

definition *cindexE-ubd*::(*real* ⇒ *real*) ⇒ *real* **where**
cindexE-ubd f = (∑ *x*∈{*x*. *jumpF f* (*at-right x*) ≠*0* }. *jumpF f* (*at-right x*))
− (∑ *x*∈{*x*. *jumpF f* (*at-left x*) ≠*0*}. *jumpF f* (*at-left x*))

lemma *cindexE-empty*:
cindexE a a f = *0*
⟨*proof*⟩

lemma *cindex-const*: *cindex a b* (λ-. *c*) = *0*
⟨*proof*⟩

lemma *cindex-eq-cindex-poly*: *cindex a b* (λ*x*. *poly q x*/*poly p x*) = *cindex-poly a b q p*
⟨*proof*⟩

lemma *cindex-combine*:
assumes *finite*:*finite* {*x*. *jump f x*≠*0* ∧ *a*<*x* ∧ *x*<*c*} **and** *a*<*b b*<*c*
shows *cindex a c f* = *cindex a b f* + *jump f b* + *cindex b c f*
⟨*proof*⟩

lemma *cindexE-combine*:
assumes *finite*:*finite-jumpFs f a c* **and** *a*≤*b b*≤*c*
shows *cindexE a c f* = *cindexE a b f* + *cindexE b c f*
⟨*proof*⟩

lemma *cindex-linear-comp*:
assumes *c*≠*0*
shows *cindex lb ub* (*f o* (λ*x*. *c*∗*x*+*b*)) = (*if c*>*0*
then cindex (*c*∗*lb*+*b*) (*c*∗*ub*+*b*) *f*

else − *cindex* (*c*∗*ub*+*b*) (*c*∗*lb*+*b*) *f*)
⟨*proof*⟩

lemma *cindexE-linear-comp*:
assumes *c*≠*0*
shows *cindexE lb ub* (*f o* (λ*x*. *c*∗*x*+*b*)) = (*if c*>*0*
then cindexE (*c*∗*lb*+*b*) (*c*∗*ub*+*b*) *f*
else − *cindexE* (*c*∗*ub*+*b*) (*c*∗*lb*+*b*) *f*)
⟨*proof*⟩

lemma *cindexE-cong*:
assumes *finite s* **and** *fg-eq*:⋀*x*. ⟦*a*<*x*;*x*<*b*;*x*∉*s*⟧ ⟹ *f x* = *g x*
shows *cindexE a b f* = *cindexE a b g*
⟨*proof*⟩

lemma *cindexE-constI*:
assumes ⋀*t*. ⟦*a*<*t*;*t*<*b*⟧ ⟹ *f t*=*c*
shows *cindexE a b f* = *0*
⟨*proof*⟩

lemma *cindex-eq-cindexE-divide*:
fixes *f g*::*real* ⇒ *real*
defines *h* ≡ (λ*x*. *f x*/*g x*)
assumes *a*<*b* **and**
finite-fg: *finite* {*x*. (*f x*=*0*∨*g x*=*0*) ∧ *a*≤*x*∧*x*≤*b*} **and**
g-imp-f:∀ *x*∈{*a*..*b*}. *g x*=*0* ⟶ *f x*≠*0* **and**
f-cont:*continuous-on* {*a*..*b*} *f* **and**
g-cont:*continuous-on* {*a*..*b*} *g*
shows *cindexE a b h* = *jumpF h* (*at-right a*) + *cindex a b h* − *jumpF h* (*at-left b*)
⟨*proof*⟩

5.9 Cauchy index along a path

— Deprecated, use "cindex_pathE" if possible
definition *cindex-path*::(*real* ⇒ *complex*) ⇒ *complex* ⇒ *int* **where**
cindex-path g z = *cindex 0 1* (λ*t*. *Im* (*g t* − *z*) / *Re* (*g t* − *z*))

definition *cindex-pathE*::(*real* ⇒ *complex*) ⇒ *complex* ⇒ *real* **where**
cindex-pathE g z = *cindexE 0 1* (λ*t*. *Im* (*g t* − *z*) / *Re* (*g t* − *z*))

lemma *cindex-pathE-point*: *cindex-pathE* (*linepath a a*) *b* = *0*
⟨*proof*⟩

lemma *cindex-path-reversepath*:
cindex-path (*reversepath g*) *z* = − *cindex-path g z*
⟨*proof*⟩

lemma *cindex-pathE-reversepath*: *cindex-pathE* (*reversepath g*) *z* = −*cindex-pathE*

g z
⟨*proof*⟩

lemma *cindex-pathE-reversepath'*: *cindex-pathE g z* = −*cindex-pathE* (*reversepath g*) *z*
⟨*proof*⟩

lemma *cindex-pathE-joinpaths*:
assumes *g1*:*finite-ReZ-segments g1 z* **and** *g2*: *finite-ReZ-segments g2 z* **and**
path g1 path g2 pathfinish g1 = *pathstart g2*
shows *cindex-pathE* (*g1+++g2*) *z* = *cindex-pathE g1 z* + *cindex-pathE g2 z*
⟨*proof*⟩

lemma *cindex-pathE-constI*:
assumes ⋀*t*. ⟦*0*<*t*;*t*<*1*⟧ ⟹ *g t*=*c*
shows *cindex-pathE g z* = *0*
⟨*proof*⟩

lemma *cindex-pathE-subpath-combine*:
assumes *g*:*finite-ReZ-segments g z***and** *path g* **and**
0≤*a a*≤*b b*≤*c c*≤*1*
shows *cindex-pathE* (*subpath a b g*) *z* + *cindex-pathE* (*subpath b c g*) *z*
= *cindex-pathE* (*subpath a c g*) *z*
⟨*proof*⟩

lemma *cindex-pathE-shiftpath*:
assumes *finite-ReZ-segments g z s*∈{*0..1*} *path g* **and** *loop*:*pathfinish g* = *pathstart g*
shows *cindex-pathE* (*shiftpath s g*) *z* = *cindex-pathE g z*
⟨*proof*⟩

5.10 Cauchy's Index Theorem

theorem *winding-number-cindex-pathE-aux*:
fixes *g*::*real* ⇒ *complex*
assumes *finite-ReZ-segments g z* **and** *valid-path g z* ∉ *path-image g* **and**
Re-ends:*Re* (*g 1*) = *Re z Re* (*g 0*) = *Re z*
shows *2* ∗ *Re*(*winding-number g z*) = − *cindex-pathE g z*
⟨*proof*⟩

theorem *winding-number-cindex-pathE*:
fixes *g*::*real* ⇒ *complex*
assumes *finite-ReZ-segments g z* **and** *valid-path g z* ∉ *path-image g* **and**
loop: *pathfinish g* = *pathstart g*
shows *winding-number g z* = − *cindex-pathE g z* / *2*
⟨*proof*⟩

REMARK: The usual statement of Cauchy's Index theorem (i.e. Analytic Theory of Polynomials (2002): Theorem 11.1.3) is about the equality between the number of polynomial roots and the Cauchy index, which

is the joint application of ⟦*finite-ReZ-segments ?g ?z*; *valid-path ?g*; *?z* ∉ *path-image ?g*; *pathfinish ?g* = *pathstart ?g*⟧ ⟹ *winding-number ?g ?z* = *complex-of-real* (− *cindex-pathE ?g ?z* / *2*) and ⟦*open ?s*; *connected ?s*; *?f holomorphic-on ?s* − *?poles*; *?h holomorphic-on ?s*; *valid-path ?g*; *pathfinish ?g* = *pathstart ?g*; *path-image ?g* ⊆ *?s* − {*w*. *?f w* = *0* ∨ *w* ∈ *?poles*}; ∀ *z*. *z* ∉ *?s* ⟶ *winding-number ?g z* = *0*; *finite* {*w*. *?f w* = *0* ∨ *w* ∈ *?poles*}; ∀ *p*∈*?poles*. *is-pole ?f p*⟧ ⟹ *contour-integral ?g* (λ*x*. *deriv ?f x* ∗ *?h x* / *?f x*) = *complex-of-real* (*2* ∗ *pi*) ∗ i ∗ (∑ *p*∈{*w*. *?f w* = *0* ∨ *w* ∈ *?poles*}. *winding-number ?g p* ∗ *?h p* ∗ *of-int* (*zorder ?f p*)).

end

6 Evaluate winding numbers by calculating Cauchy indices

theory *Winding-Number-Eval* **imports**
Cauchy-Index-Theorem
HOL−Eisbach.Eisbach-Tools
begin

6.1 Misc

lemma *not-on-closed-segmentI*:
fixes *z*::*'a*::*euclidean-space*
assumes *norm* (*z* − *a*) ∗$_R$ (*b* − *z*) ≠ *norm* (*b* − *z*) ∗$_R$ (*z* − *a*)
shows *z* ∉ *closed-segment a b*
⟨*proof*⟩

lemma *not-on-closed-segmentI-complex*:
fixes *z*::*complex*
assumes (*Re b* − *Re z*) ∗ (*Im z* − *Im a*) ≠ (*Im b* − *Im z*) ∗ (*Re z* − *Re a*)
shows *z* ∉ *closed-segment a b*
⟨*proof*⟩

6.2 finite intersection with the two axes

definition *finite-axes-cross*::(*real* ⇒ *complex*) ⇒ *complex* ⇒ *bool* **where**
finite-axes-cross g z = *finite* {*t*. (*Re* (*g t*−*z*) = *0* ∨ *Im* (*g t*−*z*) = *0*) ∧ *0* ≤ *t* ∧ *t* ≤ *1*}

lemma *finite-cross-intros*:
⟦*Re a*≠*Re z* ∨ *Re b* ≠*Re z*; *Im a*≠*Im z* ∨ *Im b*≠*Im z*⟧⟹*finite-axes-cross* (*linepath a b*) *z*
⟦*st* ≠ *tt*; *r* ≠ *0*⟧ ⟹ *finite-axes-cross* (*part-circlepath z0 r st tt*) *z*
⟦*finite-axes-cross g1 z*;*finite-axes-cross g2 z*⟧ ⟹ *finite-axes-cross* (*g1*+++*g2*) *z*
⟨*proof*⟩

lemma *cindex-path-joinpaths*:

assumes *finite-axes-cross g1 z finite-axes-cross g2 z*
and *path g1 path g2 pathfinish g1 = pathstart g2 pathfinish g1≠z*
shows *cindex-path (g1+++g2) z = cindex-path g1 z + jumpF-pathstart g2 z − jumpF-pathfinish g1 z + cindex-path g2 z*
⟨*proof*⟩

6.3 More lemmas related *cindex-pathE / jumpF-pathstart / jumpF-pathfinish*

lemma *cindex-pathE-linepath*:
assumes *z∉closed-segment a b*
shows *cindex-pathE (linepath a b) z = (*
let c1 = Re a − Re z;
c2 = Re b − Re z;
*c3 = Im a * Re b + Re z * Im b + Im z * Re a − Im z * Re b − Im b * Re a − Re z * Im a;*
d1 = Im a − Im z;
d2 = Im b − Im z
in if (c1>0 ∧ c2<0) ∨ (c1<0 ∧ c2>0) then
(if c3>0 then 1 else −1)
else
(if (c1=0 ⟷ c2≠0) ∧ (c1=0 ⟶ d1≠0) ∧ (c2=0 ⟶ d2≠0) then
if (c1=0 ∧ (c2 >0 ⟷ d1>0)) ∨ (c2=0 ∧ (c1 >0 ⟷ d2<0)) then 1/2 else −1/2
else 0))
⟨*proof*⟩

lemma *cindex-path-linepath*:
assumes *z∉path-image (linepath a b)*
shows *cindex-path (linepath a b) z = (*
let c1=Re(a)−Re(z) ; c2=Re(b)−Re(z) ;
*c3 = Im(a)*Re(b)+Re(z)*Im(b)+Im(z)*Re(a) −Im(z)*Re(b) − Im(b)*Re(a) − Re(z)*Im(a)*
in if (c1>0 ∧ c2<0) ∨ (c1<0 ∧ c2>0) then (if c3>0 then 1 else −1) else 0)
⟨*proof*⟩

lemma *cindex-pathE-part-circlepath*:
assumes *cmod (z−z0) ≠r* **and** *r>0 0≤st st<tt tt≤2*pi*
shows *cindex-pathE (part-circlepath z r st tt) z0 = (*
if |Re z − Re z0| < r then
(let
ϑ = arccos ((Re z0 − Re z)/r);
*β = 2*pi − ϑ*
in
jumpF-pathstart (part-circlepath z r st tt) z0
+
*(if st<ϑ ∧ ϑ<tt then if r * sin ϑ + Im z > Im z0 then −1 else 1 else 0)*
+
*(if st<β ∧ β < tt then if r * sin β + Im z > Im z0 then 1 else −1 else 0)*

```
        −
        jumpF-pathfinish (part-circlepath z r st tt) z0
    )
  else
    if |Re z − Re z0| = r then
      jumpF-pathstart (part-circlepath z r st tt) z0
      − jumpF-pathfinish (part-circlepath z r st tt) z0
    else 0
  )
⟨proof⟩

lemma jumpF-pathstart-part-circlepath:
  assumes st<tt r>0 cmod (z−z0) ≠r
  shows jumpF-pathstart (part-circlepath z r st tt) z0 = (
          if r ∗ cos st + Re z − Re z0 = 0 then
            (let
              Δ = r∗ sin st + Im z − Im z0
            in
              if (sin st > 0 ∨ cos st=1 ) ∧ Δ < 0
                  ∨ (sin st < 0 ∨  cos st=−1 ) ∧ Δ > 0  then
                1/2
              else
                − 1/2)
          else 0)
⟨proof⟩

lemma jumpF-pathfinish-part-circlepath:
  assumes st<tt r>0 cmod (z−z0) ≠r
  shows jumpF-pathfinish (part-circlepath z r st tt) z0 = (
          if r ∗ cos tt + Re z − Re z0 = 0 then
            (let
              Δ = r∗ sin tt + Im z − Im z0
            in
              if (sin tt > 0 ∨ cos tt=−1 ) ∧ Δ < 0
                  ∨ (sin tt < 0 ∨  cos tt=1 ) ∧ Δ > 0  then
                − 1/2
              else
                1/2)
          else 0)
⟨proof⟩

lemma
  fixes z0 z::complex and r::real
  defines upper ≡ cindex-pathE (part-circlepath z r 0 pi) z0
     and lower ≡ cindex-pathE (part-circlepath z r pi (2∗pi)) z0
  shows cindex-pathE-circlepath-upper:
     ⟦cmod (z0−z) < r⟧ ⟹ upper = −1
     ⟦Im (z0−z) > r; |Re (z0 − z)| < r⟧ ⟹ upper = 1
     ⟦Im (z0−z) < −r; |Re (z0 − z)| < r⟧ ⟹ upper = −1
```

$[\![|Re\ (z0\ -\ z)| > r;\ r{>}0]\!] \Longrightarrow upper = 0$
and *cindex-pathE-circlepath-lower*:
$[\![cmod\ (z0{-}z) < r]\!] \Longrightarrow lower = -1$
$[\![Im\ (z0{-}z) > r;\ |Re\ (z0\ -\ z)| < r]\!] \Longrightarrow lower = -1$
$[\![Im\ (z0{-}z) < -r;\ |Re\ (z0\ -\ z)| < r]\!] \Longrightarrow lower = 1$
$[\![|Re\ (z0\ -\ z)| > r;\ r{>}0]\!] \Longrightarrow lower = 0$
⟨*proof*⟩

lemma *jumpF-pathstart-linepath*:
jumpF-pathstart (*linepath a b*) *z* =
(*if Re a* = *Re z* ∧ *Im a*≠*Im z* ∧ *Re b* ≠ *Re a then*
if (*Im a*>*Im z* ∧ *Re b* > *Re a*) ∨ (*Im a*<*Im z* ∧ *Re b* < *Re a*) *then* 1/2
else −1/2
else 0)
⟨*proof*⟩

lemma *jumpF-pathfinish-linepath*:
jumpF-pathfinish (*linepath a b*) *z* =
(*if Re b* = *Re z* ∧ *Im b* ≠*Im z* ∧ *Re b* ≠ *Re a then*
if (*Im b*>*Im z* ∧ *Re a* > *Re b*) ∨ (*Im b*<*Im z* ∧ *Re a* < *Re b*) *then* 1/2
else −1/2
else 0)
⟨*proof*⟩

6.4 Setting up the method for evaluating winding numbers

lemma *pathfinish-pathstart-partcirclepath-simps*:
pathstart (*part-circlepath z0 r* (3∗*pi*/2) *tt*) = *z0* − *Complex 0 r*
pathstart (*part-circlepath z0 r* (2∗*pi*) *tt*) = *z0* + *r*
pathfinish (*part-circlepath z0 r st* (3∗*pi*/2)) = *z0* − *Complex 0 r*
pathfinish (*part-circlepath z0 r st* (2∗*pi*)) = *z0* + *r*
pathstart (*part-circlepath z0 r 0 tt*) = *z0* + *r*
pathstart (*part-circlepath z0 r* (*pi*/2) *tt*) = *z0* + *Complex 0 r*
pathstart (*part-circlepath z0 r* (*pi*) *tt*) = *z0* − *r*
pathfinish (*part-circlepath z0 r st 0*) = *z0*+*r*
pathfinish (*part-circlepath z0 r st* (*pi*/2)) = *z0* + *Complex 0 r*
pathfinish (*part-circlepath z0 r st* (*pi*)) = *z0* − *r*
⟨*proof*⟩

lemma *winding-eq-intro*:
finite-ReZ-segments g z ⟹
valid-path g ⟹
z ∉ *path-image g* ⟹
pathfinish g = *pathstart g* ⟹
− *of-real*(*cindex-pathE g z*) =2∗*n* ⟹
winding-number g z = (*n*::*complex*)
⟨*proof*⟩

named-theorems *winding-intros* **and** *winding-simps*

lemmas [*winding-intros*] =
finite-ReZ-segments-joinpaths
valid-path-join
path-join-imp
not-in-path-image-join

lemmas [*winding-simps*] =
finite-ReZ-segments-linepath
finite-ReZ-segments-part-circlepath
jumpF-pathfinish-joinpaths
jumpF-pathstart-joinpaths
pathfinish-linepath
pathstart-linepath
pathfinish-join
pathstart-join
valid-path-linepath
valid-path-part-circlepath
path-part-circlepath
Re-complex-of-real
Im-complex-of-real
of-real-linepath
pathfinish-pathstart-partcirclepath-simps

method *rep-subst* =
(*subst cindex-pathE-joinpaths*; *rep-subst*)*?*

The method "eval_winding" *1*::*'a* will try to simplify of the form *winding-number g z* = *n* where *n* is an integer and *g* is a closed path comprised of *linepath*, *part-circlepath* and (+++).

Suppose *g* = *l1* +++ *l2*, usually, the key behind the success of this framework is whether we can prove *z* ∉ *path-image l1*, *z* ∉ *path-image l2* and calculate *cindex-pathE l1 z* and *cindex-pathE l2 z*.

method *eval-winding* =
((*rule-tac winding-eq-intro*;
rep-subst
)
, *auto simp only*:*winding-simps del*:*notI intro*!:*winding-intros*
, *tactic* ‹*distinct-subgoals-tac*›)

end

7 Some examples of applying the method winding_eval

theory *Winding-Number-Eval-Examples* **imports** *Winding-Number-Eval*
begin

lemma *example1*:
assumes *R>1*
shows *winding-number* (*part-circlepath 0 R 0 pi* +++ *linepath* (−*R*) *R*) i = *1*
⟨*proof*⟩

lemma *example2*:
assumes *R>1*
shows *winding-number* (*part-circlepath 0 R 0 pi* +++ *linepath* (−*R*) *R*) (−i) = *0*
⟨*proof*⟩

lemma *example3*:
fixes *lb ub z* :: *complex*
defines *rec* ≡ *linepath lb* (*Complex* (*Re ub*) (*Im lb*)) +++ *linepath* (*Complex* (*Re ub*) (*Im lb*)) *ub*
+++ *linepath ub* (*Complex* (*Re lb*) (*Im ub*)) +++ *linepath* (*Complex* (*Re lb*) (*Im ub*)) *lb*
assumes *order-asms*:*Re lb* < *Re z Re z* < *Re ub Im lb* < *Im z Im z* < *Im ub*
shows *winding-number rec z* = *1*
⟨*proof*⟩

end

8 Acknowledgements

The work was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178), funded by the European Research Council and led by Professor Lawrence Paulson at the University of Cambridge, UK.

References

[1] M. Eisermann. The fundamental theorem of algebra made effective: An elementary real-algebraic proof via Sturm chains. *American Mathematical Monthly*, 119(9):715–752, 2012.

[2] Q. I. Rahman and G. Schmeisser. *Analytic theory of polynomials*. Number 26. Oxford University Press, 2002.