Algebraic Invariants of the Circle, Torus, and Punctured Plane

Arthur Freitas Ramos David Barros Hulak Ruy J. G. B. de Queiroz

November 15, 2026

Abstract

We formalize real path lifts of based loops in the circle and prove that their integer endpoints classify these loops up to path homotopy. For the torus, the two coordinate lifts give a classification by pairs of integers and an explicit fundamental-group isomorphism with $\mathbb{Z} \times \mathbb{Z}$. Every continuous circle self-map acts on first singular homology by multiplication by its winding number. We construct a radial deformation retraction of the punctured complex plane onto the circle and compute the reduced homology of both spaces: it is isomorphic to the additive group of integers in degree one and trivial in every other degree. Neither space is contractible. The development also provides a reusable interface for path lifting in covering spaces.

Contents

1 Introduction **2**

2 Covering-space lifts for the circle **2**

3 Path lifting **4**

4 Generic pointed path lifts **18**

5 Pointed covering spaces **21**

6 Product paths and torus loops **21**

7 The fundamental group of the torus **26**

8 The punctured plane retracts onto the circle **30**

9 Seifert–van Kampen interface **33**

10 Homology of the circle and the punctured plane **33**

11 Brouwer degree **36**

12 Loops as singular cycles **37**

13 Concatenation and addition of cycles **39**

14 Winding number annihilates singular boundaries **43**

15 Winding number and the induced homology map **46**

16 Circle, torus, and punctured-plane invariants: collected results **49**

1 Introduction

Path lifting assigns an integer to each based loop in the circle. This integer classifies loops up to path homotopy and provides two coordinates for the corresponding classification on the torus. Radial retraction connects the circle with the punctured complex plane and transfers the homology computation between them. These constructions follow standard treatments of algebraic topology, such as Hatcher [1]. We build on the AFP circle entry [3] and use the classical Seifert–van Kampen entry [2].

The principal contributions are the real lift-endpoint classification of circle loops, an explicit torus fundamental-group isomorphism with $\mathbb{Z} \times \mathbb{Z}$, and the transfer of reduced homology to the concrete circle and punctured plane. We also prove that the actual map induced on first singular homology by every continuous circle self-map is multiplication by its winding number, in any integer coordinates. The proof represents loops as singular cycles, relates concatenation to addition using a singular triangle, and proves that winding number annihilates singular boundaries. A homotopy through rotations removes the basepoint restriction on the self-map. The generic lifting interface packages the analysis library's results. The Seifert–van Kampen statements are re-exports of the cited AFP entry; the degree laws specialize the homology library to the abstract one-sphere. The concrete induced-map theorem is separate from the library constant `Brouwer_degree2`; no additional identification with that constant, general Hurewicz theorem, or torus homology computation is asserted here.

The covering interface is independent of the circle development. The product, radial-retract, and homology theories may be imported separately; the final theory combines their interfaces.

theory *Circle-Covering*
  imports
    Fundamental-Group-Circle.Fundamental-Group-Circle
    HOL−Analysis.Further-Topology
begin

2 Covering-space lifts for the circle

This theory complements the winding-number proof in the circle entry with the real-lift description of based circle loops. The covering-space API used below is the one from the standard Isabelle analysis library; in particular, the exponential covering of the punctured plane supplies the analytic lifting argument.

definition *circle-cover* :: *real* ⇒ *complex* **where**
  circle-cover t = *circlepath 0 1 t*

lemma *circle-cover-exp*:
  circle-cover t = *exp* (*2* ∗ *of-real pi* ∗ i ∗ *of-real t*)
  by (*simp add*: *circle-cover-def circlepath*)

lemma *path-circle-cover* [*simp*]: *path circle-cover*
  unfolding *circle-cover-def*
  by (*simp only*: *path-circlepath*)

lemma *circle-cover-into-circle*: *circle-cover* ‘ *UNIV* ⊆ *circle*
  unfolding *circle-cover-exp circle-def*
  by (*auto simp*: *dist-norm norm-exp norm-mult*)

lemma *circle-cover-periodic-int*:
  circle-cover (*t* + *of-int n*) = *circle-cover t*
  unfolding *circle-cover-exp*
  by (*simp add*: *algebra-simps exp-add*)

```
lemma circle-cover-fiber:
  circle-cover s = circle-cover t ⟷ (∃ n::int. s = t + of-int n)
  unfolding circle-cover-exp
proof −
  have hexp: exp (2 ∗ of-real pi ∗ i ∗ of-real s) =
      exp (2 ∗ of-real pi ∗ i ∗ of-real t) ⟷
      (∃ n::int. 2 ∗ of-real pi ∗ i ∗ of-real s =
        2 ∗ of-real pi ∗ i ∗ of-real t + (of-int (2 ∗ n) ∗ pi) ∗ i)
    by (rule exp-eq)
  show exp (2 ∗ of-real pi ∗ i ∗ of-real s) =
      exp (2 ∗ of-real pi ∗ i ∗ of-real t) ⟷
      (∃ n::int. s = t + of-int n)
  proof
    assume h0: exp (2 ∗ of-real pi ∗ i ∗ of-real s) =
        exp (2 ∗ of-real pi ∗ i ∗ of-real t)
    then obtain n h where h:
        2 ∗ of-real pi ∗ i ∗ of-real s =
          2 ∗ of-real pi ∗ i ∗ of-real t + (of-int (2 ∗ n) ∗ pi) ∗ i
      using hexp by blast
    have h′: s = t + of-int n
    proof −
      have h′′:
          Im (2 ∗ of-real pi ∗ i ∗ of-real s) =
            Im (2 ∗ of-real pi ∗ i ∗ of-real t +
              (of-int (2 ∗ n) ∗ pi) ∗ i)
        using arg-cong [OF h, of Im] by blast
      have h′′′:
          2 ∗ pi ∗ s = 2 ∗ pi ∗ t + of-int (2 ∗ n) ∗ pi
        using h′′ by (simp add: algebra-simps)
      have hcancel:
          2 ∗ pi ∗ s = 2 ∗ pi ∗ (t + of-int n)
        using h′′′ by (simp add: algebra-simps)
      have hpi: 2 ∗ pi ≠ 0 by (simp add: pi-gt-zero)
      show ?thesis
        using hcancel hpi by (simp add: mult-left-cancel)
    qed
    then show ∃ n::int. s = t + of-int n by blast
  next
    assume h0: ∃ n::int. s = t + of-int n
    then obtain n h where h: s = t + of-int n by blast
    have h′: 2 ∗ of-real pi ∗ i ∗ of-real s =
        2 ∗ of-real pi ∗ i ∗ of-real t + (of-int (2 ∗ n) ∗ pi) ∗ i
      using h by (simp add: algebra-simps)
    then show exp (2 ∗ of-real pi ∗ i ∗ of-real s) =
        exp (2 ∗ of-real pi ∗ i ∗ of-real t)
      using hexp by blast
  qed
qed

lemma circle-cover-eq-iff:
  circle-cover s = circle-cover t ⟷ s − t ∈ ℤ
proof
  show circle-cover s = circle-cover t ⟹ s − t ∈ ℤ
  proof −
    assume h: circle-cover s = circle-cover t
    obtain n where n: s = t + of-int n
      using h circle-cover-fiber [of s t] by blast
    have s − t = of-int n using n by linarith
```

```
    then show s − t ∈ ℤ by (simp add: Ints-def)
  qed
next
  show s − t ∈ ℤ ⟹ circle-cover s = circle-cover t
  proof −
    assume h: s − t ∈ ℤ
    obtain n::int where n: s − t = of-int n
      using h by (auto simp: Ints-def)
    have s = t + of-int n using n by linarith
    then show circle-cover s = circle-cover t
      using circle-cover-fiber [of s t] by (blast dest: iffD2)
  qed
qed

lemma circle-cover-one [simp]: circle-cover 0 = (1::complex)
  by (simp add: circle-cover-exp)

lemma circle-cover-Arg:
  z ∈ circle ⟹ circle-cover (Arg z / (2 * pi)) = z
proof −
  assume z: z ∈ circle
  have ze: exp (i * Arg z) = z
    using z by (simp add: circle-def complex-norm-eq-1-exp-eq)
  have harg:
     2 * of-real pi * i * of-real (Arg z / (2 * pi)) = i * Arg z
    by (simp add: algebra-simps)
  have hcover:
     circle-cover (Arg z / (2 * pi)) = exp (i * Arg z)
    unfolding circle-cover-exp
    using harg by simp
  show circle-cover (Arg z / (2 * pi)) = z
    using hcover ze by (rule trans)
qed

lemma circle-cover-mem: z ∈ circle ⟹ z ∈ circle-cover ‘ UNIV
proof −
  assume hz: z ∈ circle
  have hc: circle-cover (Arg z / (2 * pi)) = z
    by (rule circle-cover-Arg [OF hz])
  show z ∈ circle-cover ‘ UNIV
    by (rule image-eqI[where x=Arg z / (2 * pi)])
       (auto simp add: hc)
qed

lemma circle-cover-range: circle-cover ‘ UNIV = circle
proof (rule subset-antisym)
  show circle-cover ‘ UNIV ⊆ circle
    by (rule circle-cover-into-circle)
  show circle ⊆ circle-cover ‘ UNIV
    by (rule subsetI) (rule circle-cover-mem)
qed

lemma circle-cover-surjective: circle-cover ‘ UNIV = circle
  by (rule circle-cover-range)
```

3 Path lifting

```
definition circle-path-lift :: (real ⇒ complex) ⇒ (real ⇒ real) ⇒ bool where
```

```
  circle-path-lift p q ⟷
    path q ∧ pathstart q = 0 ∧
    (∀ t ∈ {0..1}. circle-cover (q t) = p t)

lemma circle-path-lift-exists:
  assumes ppath: path p
      and pimage: path-image p ⊆ circle
      and pstart: pathstart p = 1
  shows ∃ q. circle-path-lift p q
proof −
  have pnonzero: 0 ∉ path-image p
    using pimage zero-notin-circle by blast
  obtain h where hpath: path h and himage: path-image h ⊆ UNIV
      and hstart: pathstart h = 0
      and hexp: ⋀t. t ∈ {0..1} ⟹ exp (h t) = p t
    using covering-space-lift-path-strong
      [OF covering-space-exp-punctured-plane, of 0 p] ppath pimage pstart pnonzero
    by (auto simp: circle-subset-punctured)
  define r where r ≡ λt. Im (h t) / (2 * pi)
  have rpath: path r
    unfolding r-def path-def
    using hpath by (intro continuous-intros) (auto simp add: path-def pi-gt-zero)
  have rstart: pathstart r = 0
    unfolding r-def pathstart-def
    using hstart by (simp add: pathstart-def)
  have rcover: circle-cover (r t) = p t if t ∈ {0..1} for t
  proof −
    have pt: p t ∈ circle
      using pimage that by (auto simp: path-image-def)
    have nexp: norm (exp (h t)) = 1
      using hexp that by (simp add: norm-of-circle [OF pt])
    have hre: Re (h t) = 0
      by (rule norm-exp-imaginary [OF nexp])
    have heq: 2 * of-real pi * i * of-real (Im (h t) / (2 * pi)) = h t
      apply (rule complex-eqI)
       apply (simp add: hre)
      apply (simp add: algebra-simps pi-gt-zero)
    done
    show ?thesis
      unfolding circle-cover-exp r-def
      using heq hexp that by simp
  qed
  have circle-path-lift p r
    unfolding circle-path-lift-def
    using rpath rstart rcover by blast
  then show ?thesis by blast
qed

definition circle-lift-exp :: (real ⇒ real) ⇒ (real ⇒ complex) where
  circle-lift-exp q t = i * of-real (2 * pi * q t)

lemma path-circle-lift-exp:
  assumes path q
  shows path (circle-lift-exp q)
  unfolding circle-lift-exp-def path-def
  using assms by (intro continuous-intros) (auto simp: path-def)

lemma pathstart-circle-lift-exp:
```

```
  assumes pathstart q = 0
  shows pathstart (circle-lift-exp q) = 0
  using assms by (simp add: circle-lift-exp-def pathstart-def)

lemma circle-lift-exp-map:
  exp (circle-lift-exp q t) = circle-cover (q t)
  unfolding circle-lift-exp-def circle-cover-exp
  by (simp add: algebra-simps)

lemma circle-path-lift-unique:
  assumes l1: circle-path-lift p q1
      and l2: circle-path-lift p q2
      and ppath: path p
      and pimage: path-image p ⊆ circle
  shows ∀ t ∈ {0..1}. q1 t = q2 t
proof −
  have q1path: path q1
    using l1 by (simp add: circle-path-lift-def)
  have q1start: pathstart q1 = 0
    using l1 by (simp add: circle-path-lift-def)
  have q2path: path q2
    using l2 by (simp add: circle-path-lift-def)
  have q2start: pathstart q2 = 0
    using l2 by (simp add: circle-path-lift-def)
  have pnonzero: path-image p ⊆ − {0}
    using pimage circle-subset-punctured by blast
  have qeq: q1 t = q2 t if t ∈ {0..1} for t
  proof −
    have heq:
       circle-lift-exp q1 t = circle-lift-exp q2 t
      proof (rule covering-space-lift-unique
         [OF covering-space-exp-punctured-plane, of
           circle-lift-exp q1 0 circle-lift-exp q2 {0..1} p t])
        show circle-lift-exp q1 0 = circle-lift-exp q2 0
          using q1start q2start by (simp add: circle-lift-exp-def pathstart-def)
        show continuous-on {0..1} p
          using ppath by (simp add: path-def)
        show p ∈ {0..1} → − {0}
          using pnonzero by (auto simp: path-image-def)
        show continuous-on {0..1} (circle-lift-exp q1)
          unfolding circle-lift-exp-def
          using q1path unfolding path-def
          by (intro continuous-intros) (auto simp: path-def)
        show circle-lift-exp q1 ∈ {0..1} → UNIV
          by simp
        show ⋀x. x ∈ {0..1} ⟹
            p x = exp (circle-lift-exp q1 x)
          using l1 by (auto simp: circle-path-lift-def circle-lift-exp-map)
        show continuous-on {0..1} (circle-lift-exp q2)
          unfolding circle-lift-exp-def
          using q2path unfolding path-def
          by (intro continuous-intros) (auto simp: path-def)
        show circle-lift-exp q2 ∈ {0..1} → UNIV
          by simp
        show ⋀x. x ∈ {0..1} ⟹
            p x = exp (circle-lift-exp q2 x)
          using l2 by (auto simp: circle-path-lift-def circle-lift-exp-map)
        show connected ({0..1}::real set)
```

```
      by (simp add: connected-Icc)
    show 0 ∈ ({0..1}::real set)
      by simp
    show t ∈ ({0..1}::real set)
      using that .
  qed
  have him: 2 * pi * q1 t = 2 * pi * q2 t
    using arg-cong [OF heq, of Im] by (simp add: circle-lift-exp-def)
  have hpi: 2 * pi ≠ 0
    by (simp add: pi-gt-zero)
  then show ?thesis
    using him by (simp add: mult-left-cancel)
 qed
 show ?thesis
  using qeq by blast
qed

definition circle-lift :: (real ⇒ complex) ⇒ (real ⇒ real) where
 circle-lift p = (SOME q. circle-path-lift p q)

lemma circle-lift-spec:
 assumes path p path-image p ⊆ circle pathstart p = 1
 shows circle-path-lift p (circle-lift p)
proof −
 have ∃ q. circle-path-lift p q
  using assms by (rule circle-path-lift-exists)
 then show ?thesis
  unfolding circle-lift-def
  by (rule someI-ex)
qed

lemma circle-lift-unique:
 assumes l1: circle-path-lift p q1
    and l2: circle-path-lift p q2
    and ppath: path p and pimage: path-image p ⊆ circle
 shows ∀ t ∈ {0..1}. q1 t = q2 t
 using circle-path-lift-unique [OF l1 l2 ppath pimage] .

definition circle-lift-endpoint :: (real ⇒ complex) ⇒ int where
 circle-lift-endpoint p =
   (SOME n::int. circle-lift p 1 = of-int n)

lemma circle-lift-endpoint-spec:
 assumes loop: is-loop p
 shows circle-lift p 1 = of-int (circle-lift-endpoint p)
proof −
 have lp: circle-path-lift p (circle-lift p)
  using loop by (intro circle-lift-spec) (simp-all add: is-loop-def)
 have hp1: circle-cover (circle-lift p 1) = p 1
  using lp by (simp add: circle-path-lift-def)
 have hpend: p 1 = 1
  using loop by (simp add: is-loop-def pathfinish-def)
 have hcover: circle-cover (circle-lift p 1) = 1
  using hp1 hpend by simp
 have hzero: circle-cover 0 = (1::complex)
  by (rule circle-cover-one)
 have heqcover: circle-cover (circle-lift p 1) = circle-cover 0
  using hcover hzero by simp
```

```
  have hfiber: ∃ n::int. circle-lift p 1 = 0 + of-int n
    using circle-cover-fiber [of circle-lift p 1 0] heqcover by blast
  obtain n::int where n: circle-lift p 1 = 0 + of-int n
    using hfiber by blast
  have ex: ∃ n::int. circle-lift p 1 = of-int n
    using n by (intro exI[of - n]) simp
  show ?thesis
    unfolding circle-lift-endpoint-def
    by (rule someI-ex [OF ex])
qed

lemma circle-lift-endpoint-integer:
  assumes is-loop p
  shows circle-lift p 1 ∈ ℤ
  using circle-lift-endpoint-spec [OF assms]
  by (simp add: Ints-def)

lemma path-add-const:
  fixes q :: real ⇒ real and a :: real
  assumes path q
  shows path (λt. q t + a)
proof −
  have hpath: path ((λx. a + x) ∘ q)
    using path-translation-eq [of a q] assms by simp
  then show ?thesis
    by (simp add: comp-def add.commute)
qed

lemma pathstart-add-const:
  fixes q :: real ⇒ real and a :: real
  assumes pathstart q = 0
  shows pathstart (λt. q t + a) = a
proof −
  have q0: q 0 = 0
    using assms by (simp add: pathstart-def)
  show ?thesis
    unfolding pathstart-def
    using q0 by simp
qed

lemma circle-path-lift-join:
  assumes lp: circle-path-lift p P
      and lq: circle-path-lift q Q
      and endpoint: P 1 = of-int n
  shows circle-path-lift (p +++ q) (P +++ (λt. Q t + P 1))
proof −
  have Ppath: path P
    using lp by (simp add: circle-path-lift-def)
  have Pstart: pathstart P = 0
    using lp by (simp add: circle-path-lift-def)
  have Qpath: path Q
    using lq by (simp add: circle-path-lift-def)
  have Qstart: pathstart Q = 0
    using lq by (simp add: circle-path-lift-def)
  have Qshiftpath: path (λt. Q t + P 1)
    by (rule path-add-const [OF Qpath])
  have Qshiftstart: pathstart (λt. Q t + P 1) = P 1
    by (rule pathstart-add-const [OF Qstart])
```

```
  have Pfinish: pathfinish P = P 1
    by (simp add: pathfinish-def)
  have connect: pathfinish P = pathstart (λt. Q t + P 1)
    using Pfinish Qshiftstart by simp
  have joinedpath: path (P +++ (λt. Q t + P 1))
    by (rule path-join-imp [OF Ppath Qshiftpath connect])
  have P0: P 0 = 0
    using Pstart by (simp add: pathstart-def)
  have joinedstart: pathstart (P +++ (λt. Q t + P 1)) = 0
    by (simp add: pathstart-join pathstart-def joinpaths-def P0)
  have cover-join:
      circle-cover ((P +++ (λt. Q t + P 1)) t) = (p +++ q) t
    if t ∈ {0..1} for t
  proof (cases t ≤ 1/2)
    case True
    have ti: 2 * t ∈ {0..1}
      using that True by auto
    have hP: circle-cover (P (2 * t)) = p (2 * t)
      using lp ti by (simp add: circle-path-lift-def)
    then show ?thesis
      by (simp only: joinpaths-def if-P True)
  next
    case False
    have ti: 2 * t - 1 ∈ {0..1}
      using that False by auto
    have hQ: circle-cover (Q (2 * t - 1)) = q (2 * t - 1)
      using lq ti by (simp add: circle-path-lift-def)
    have hperiod:
        circle-cover (Q (2 * t - 1) + P 1) =
          circle-cover (Q (2 * t - 1))
      using endpoint by (simp add: circle-cover-periodic-int)
    have notnorm: ¬ t * 2 ≤ 1
      using False by linarith
    show ?thesis
      using hQ hperiod notnorm by (simp add: joinpaths-def)
  qed
  show ?thesis
    unfolding circle-path-lift-def
    using joinedpath joinedstart cover-join by blast
qed

lemma circle-lift-endpoint-join:
  assumes p: is-loop p and q: is-loop q
  shows circle-lift-endpoint (p +++ q) =
    circle-lift-endpoint p + circle-lift-endpoint q
proof −
  have lp: circle-path-lift p (circle-lift p)
    using p by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have lq: circle-path-lift q (circle-lift q)
    using q by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have ep: circle-lift p 1 = of-int (circle-lift-endpoint p)
    by (rule circle-lift-endpoint-spec [OF p])
  have eq: circle-lift q 1 = of-int (circle-lift-endpoint q)
    by (rule circle-lift-endpoint-spec [OF q])
  have lj:
      circle-path-lift (p +++ q)
        (circle-lift p +++ (λt. circle-lift q t + circle-lift p 1))
    by (rule circle-path-lift-join [OF lp lq ep])
```

```
  have pj: is-loop (p +++ q)
    by (rule is-loop-join [OF p q])
  have lcj: circle-path-lift (p +++ q) (circle-lift (p +++ q))
    using pj by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have ppath: path (p +++ q)
    using pj by (rule is-loop-path)
  have pimage: path-image (p +++ q) ⊆ circle
    using pj by (rule is-loop-image)
  have leq:
      circle-lift (p +++ q) 1 =
        (circle-lift p +++ (λt. circle-lift q t + circle-lift p 1)) 1
    using circle-path-lift-unique [OF lcj lj ppath pimage]
      by simp
  have cend:
      (circle-lift p +++ (λt. circle-lift q t + circle-lift p 1)) 1 =
        circle-lift q 1 + circle-lift p 1
    by (simp add: joinpaths-def)
  have ej: circle-lift (p +++ q) 1 =
      of-int (circle-lift-endpoint (p +++ q))
    by (rule circle-lift-endpoint-spec [OF pj])
  have hreal:
      real-of-int (circle-lift-endpoint (p +++ q)) =
        real-of-int (circle-lift-endpoint q) +
        real-of-int (circle-lift-endpoint p)
  proof −
    have of-int (circle-lift-endpoint (p +++ q)) =
        circle-lift (p +++ q) 1
      using ej by simp
    also have ... =
        (circle-lift p +++ (λt. circle-lift q t + circle-lift p 1)) 1
      using leq .
    also have ... = circle-lift q 1 + circle-lift p 1
      using cend .
    also have ... =
        of-int (circle-lift-endpoint q) + of-int (circle-lift-endpoint p)
      using ep eq by simp
    finally show ?thesis .
  qed
  have hint:
      real-of-int (circle-lift-endpoint (p +++ q)) =
        real-of-int (circle-lift-endpoint p + circle-lift-endpoint q)
    using hreal by (simp add: add.commute)
  then show ?thesis
    by (metis of-int-eq-iff)
qed

lemma circle-path-lift-reverse:
  assumes lp: circle-path-lift p P
      and endpoint: P 1 = of-int n
  shows circle-path-lift (reversepath p) (λt. P (1 − t) − P 1)
proof −
  have Ppath: path P
    using lp by (simp add: circle-path-lift-def)
  have Rpath: path (reversepath P)
    using Ppath by (simp add: path-reversepath)
  have Cpath: path (λt. reversepath P t + (− P 1))
    by (rule path-add-const [OF Rpath])
  have Cpath': path (λt. P (1 − t) − P 1)
```

```
    by (rule path-eq [OF Cpath])
      (simp only: reversepath-def diff-conv-add-uminus)
  have Cstart: pathstart (λt. P (1 − t) − P 1) = 0
    by (simp add: pathstart-def)
  have cover-reverse:
      circle-cover ((λt. P (1 − t) − P 1) t) = (reversepath p) t
    if t ∈ {0..1} for t
  proof −
    have ti: 1 − t ∈ {0..1}
      using that by auto
    have hP: circle-cover (P (1 − t)) = p (1 − t)
      using lp ti by (simp add: circle-path-lift-def)
    have hshift:
        P (1 − t) − P 1 = P (1 − t) + of-int (− n)
      using endpoint by (simp only: endpoint diff-conv-add-uminus of-int-minus)
    have hperiod′:
        circle-cover (P (1 − t) + of-int (− n)) = circle-cover (P (1 − t))
      by (rule circle-cover-periodic-int [of P (1 − t) − n])
    have hperiod:
        circle-cover (P (1 − t) − P 1) = circle-cover (P (1 − t))
      using hshift hperiod′ by (simp only: hshift)
    show ?thesis
      using hP hperiod by (simp add: reversepath-def)
  qed
  show ?thesis
    unfolding circle-path-lift-def
    using Cpath′ Cstart cover-reverse by blast
qed

lemma circle-lift-endpoint-reverse:
  assumes loop: is-loop p
  shows circle-lift-endpoint (reversepath p) =
    − circle-lift-endpoint p
proof −
  have lp: circle-path-lift p (circle-lift p)
    using loop by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have ep: circle-lift p 1 = of-int (circle-lift-endpoint p)
    by (rule circle-lift-endpoint-spec [OF loop])
  have lr:
      circle-path-lift (reversepath p)
        (λt. circle-lift p (1 − t) − circle-lift p 1)
    by (rule circle-path-lift-reverse [OF lp ep])
  have rloop: is-loop (reversepath p)
    by (rule is-loop-reversepath [OF loop])
  have lrc: circle-path-lift (reversepath p) (circle-lift (reversepath p))
    using rloop by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have rpath: path (reversepath p)
    using rloop by (rule is-loop-path)
  have rimage: path-image (reversepath p) ⊆ circle
    using rloop by (rule is-loop-image)
  have er:
      circle-lift (reversepath p) 1 =
        of-int (circle-lift-endpoint (reversepath p))
    by (rule circle-lift-endpoint-spec [OF rloop])
  have p0: circle-lift p 0 = 0
    using lp by (simp add: circle-path-lift-def pathstart-def)
  have eq:
      circle-lift (reversepath p) 1 =
```

```
      (λt. circle-lift p (1 − t) − circle-lift p 1) 1
    using circle-path-lift-unique [OF lrc lr rpath rimage] by simp
  have hreal:
      real-of-int (circle-lift-endpoint (reversepath p)) =
        − real-of-int (circle-lift-endpoint p)
    using er eq ep p0 by simp
  have hint:
      real-of-int (circle-lift-endpoint (reversepath p)) =
        real-of-int (− circle-lift-endpoint p)
    using hreal by simp
  then show ?thesis
    by (metis of-int-eq-iff)
qed

lemma continuous-on-circle-cover: continuous-on UNIV circle-cover
  unfolding circle-cover-exp
  by (intro continuous-intros)

lemma circle-lift-endpoint-homotopic:
  assumes p: is-loop p and q: is-loop q
      and hom: homotopic-paths circle p q
  shows circle-lift-endpoint p = circle-lift-endpoint q
proof −
  have lp: circle-path-lift p (circle-lift p)
    using p by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have lq: circle-path-lift q (circle-lift q)
    using q by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have ppath: path p
    by (rule is-loop-path [OF p])
  have qpath: path q
    by (rule is-loop-path [OF q])
  have Ppath: path (circle-lift p)
    using lp by (simp add: circle-path-lift-def)
  have Qpath: path (circle-lift q)
    using lq by (simp add: circle-path-lift-def)
  have pimage: path-image p ⊆ − {0}
    using is-loop-image [OF p] circle-subset-punctured by blast
  have qimage: path-image q ⊆ − {0}
    using is-loop-image [OF q] circle-subset-punctured by blast
  have hcomplex-finish:
      pathfinish (circle-lift-exp (circle-lift p)) =
        pathfinish (circle-lift-exp (circle-lift q))
  proof (rule covering-space-monodromy
      [OF covering-space-exp-punctured-plane, of p q
        circle-lift-exp (circle-lift p)
        circle-lift-exp (circle-lift q)])
    show path p
      by (rule ppath)
    show path-image p ⊆ − {0}
      by (rule pimage)
    show path q
      by (rule qpath)
    show path-image q ⊆ − {0}
      by (rule qimage)
    show homotopic-paths (− {0}) p q
      using hom circle-subset-punctured by (rule homotopic-paths-subset)
    show path (circle-lift-exp (circle-lift p))
      by (rule path-circle-lift-exp [OF Ppath])
```

```
    show path-image (circle-lift-exp (circle-lift p)) ⊆ UNIV
      by simp
    show ⋀t. t ∈ {0..1} ⟹
        exp (circle-lift-exp (circle-lift p) t) = p t
      using lp by (auto simp: circle-path-lift-def circle-lift-exp-map)
    show path (circle-lift-exp (circle-lift q))
      by (rule path-circle-lift-exp [OF Qpath])
    show path-image (circle-lift-exp (circle-lift q)) ⊆ UNIV
      by simp
    show ⋀t. t ∈ {0..1} ⟹
        exp (circle-lift-exp (circle-lift q) t) = q t
      using lq by (auto simp: circle-path-lift-def circle-lift-exp-map)
    show pathstart (circle-lift-exp (circle-lift p)) =
        pathstart (circle-lift-exp (circle-lift q))
      using lp lq by (simp add: circle-path-lift-def
        pathstart-circle-lift-exp)
  qed
  have hcomplex:
      circle-lift-exp (circle-lift p) 1 =
        circle-lift-exp (circle-lift q) 1
    using hcomplex-finish by (simp add: pathfinish-def)
  have hreal:
      circle-lift p 1 = circle-lift q 1
    using arg-cong [OF hcomplex, of Im]
    by (simp add: circle-lift-exp-def)
  have ep: circle-lift p 1 = of-int (circle-lift-endpoint p)
    by (rule circle-lift-endpoint-spec [OF p])
  have eq: circle-lift q 1 = of-int (circle-lift-endpoint q)
    by (rule circle-lift-endpoint-spec [OF q])
  have hreal-int:
      real-of-int (circle-lift-endpoint p) =
        real-of-int (circle-lift-endpoint q)
    using hreal ep eq by simp
  then show ?thesis
    by (metis of-int-eq-iff)
qed

lemma circle-lift-endpoint-eq-homotopic:
  assumes p: is-loop p and q: is-loop q
      and eq: circle-lift-endpoint p = circle-lift-endpoint q
  shows homotopic-paths circle p q
proof −
  have lp: circle-path-lift p (circle-lift p)
    using p by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have lq: circle-path-lift q (circle-lift q)
    using q by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have Ppath: path (circle-lift p)
    using lp by (simp add: circle-path-lift-def)
  have Qpath: path (circle-lift q)
    using lq by (simp add: circle-path-lift-def)
  have Pstart: pathstart (circle-lift p) = 0
    using lp by (simp add: circle-path-lift-def)
  have Qstart: pathstart (circle-lift q) = 0
    using lq by (simp add: circle-path-lift-def)
  have ep: circle-lift p 1 = of-int (circle-lift-endpoint p)
    by (rule circle-lift-endpoint-spec [OF p])
  have eq': circle-lift q 1 = of-int (circle-lift-endpoint q)
    by (rule circle-lift-endpoint-spec [OF q])
```

```
  have Pfinish: pathfinish (circle-lift p) = pathfinish (circle-lift q)
    using ep eq' eq by (simp add: pathfinish-def)
  have hlin:
      homotopic-paths (UNIV::real set) (circle-lift p) (circle-lift q)
  proof (rule homotopic-paths-linear [OF Ppath Qpath])
    show pathstart (circle-lift q) = pathstart (circle-lift p)
      using Pstart Qstart by simp
    show pathfinish (circle-lift q) = pathfinish (circle-lift p)
      using Pfinish by simp
    show ⋀t. t ∈ {0..1} ⟹
        closed-segment (circle-lift p t) (circle-lift q t) ⊆ (UNIV::real set)
      by simp
  qed
  have cover-fun: circle-cover ∈ (UNIV::real set) → circle
    using circle-cover-into-circle
    by (auto simp: image-subset-iff-funcset)
  have himg:
      homotopic-paths circle
        (circle-cover ∘ circle-lift p)
        (circle-cover ∘ circle-lift q)
    by (rule homotopic-paths-continuous-image
      [OF hlin continuous-on-circle-cover cover-fun])
  have hpcomp:
      homotopic-paths circle p (circle-cover ∘ circle-lift p)
  proof (rule homotopic-paths-eq)
    show path p
      by (rule is-loop-path [OF p])
    show path-image p ⊆ circle
      by (rule is-loop-image [OF p])
    show ⋀t. t ∈ {0..1} ⟹
        p t = (circle-cover ∘ circle-lift p) t
      using lp by (auto simp: circle-path-lift-def)
  qed
  have hqcomp:
      homotopic-paths circle q (circle-cover ∘ circle-lift q)
  proof (rule homotopic-paths-eq)
    show path q
      by (rule is-loop-path [OF q])
    show path-image q ⊆ circle
      by (rule is-loop-image [OF q])
    show ⋀t. t ∈ {0..1} ⟹
        q t = (circle-cover ∘ circle-lift q) t
      using lq by (auto simp: circle-path-lift-def)
  qed
  show ?thesis
    by (meson hpcomp himg hqcomp homotopic-paths-sym homotopic-paths-trans)
qed

theorem circle-lift-endpoint-eq-iff:
  assumes p: is-loop p and q: is-loop q
  shows circle-lift-endpoint p = circle-lift-endpoint q ⟷
    homotopic-paths circle p q
proof
  assume circle-lift-endpoint p = circle-lift-endpoint q
  then show homotopic-paths circle p q
    by (rule circle-lift-endpoint-eq-homotopic [OF p q])
next
  assume homotopic-paths circle p q
```

```
  then show circle-lift-endpoint p = circle-lift-endpoint q
    by (rule circle-lift-endpoint-homotopic [OF p q])
qed

lemma circle-path-lift-circlepath:
  circle-path-lift (circlepath 0 1) id
  unfolding circle-path-lift-def circle-cover-def
  by (simp add: path-def pathstart-def)

lemma circle-lift-endpoint-circlepath:
  circle-lift-endpoint (circlepath 0 1) = 1
proof −
  have loop: is-loop (circlepath 0 1)
    by (rule is-loop-circlepath)
  have chosen: circle-path-lift (circlepath 0 1)
      (circle-lift (circlepath 0 1))
    using loop by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have ppath: path (circlepath 0 1)
    by (rule is-loop-path [OF loop])
  have pimage: path-image (circlepath 0 1) ⊆ circle
    by (rule is-loop-image [OF loop])
  have heq:
      circle-lift (circlepath 0 1) 1 = id 1
    using circle-path-lift-unique
      [OF chosen circle-path-lift-circlepath ppath pimage] by simp
  have ep:
      circle-lift (circlepath 0 1) 1 =
        of-int (circle-lift-endpoint (circlepath 0 1))
    by (rule circle-lift-endpoint-spec [OF loop])
  have hreal:
      real-of-int (circle-lift-endpoint (circlepath 0 1)) = 1
    using ep heq by simp
  have hint:
      real-of-int (circle-lift-endpoint (circlepath 0 1)) =
        real-of-int 1
    using hreal by simp
  then show ?thesis
    by (metis of-int-eq-iff)
qed

lemma circle-path-lift-linepath:
  circle-path-lift (linepath (1::complex) 1) (λt. 0)
  unfolding circle-path-lift-def
  by (simp add: circle-cover-one path-def pathstart-def)

lemma circle-lift-endpoint-linepath:
  circle-lift-endpoint (linepath (1::complex) 1) = 0
proof −
  have loop: is-loop (linepath (1::complex) 1)
    by (rule is-loop-linepath-1)
  have chosen:
      circle-path-lift (linepath (1::complex) 1)
        (circle-lift (linepath 1 1))
    using loop by (intro circle-lift-spec) (simp-all add: is-loop-def)
  have ppath: path (linepath (1::complex) 1)
    by (rule is-loop-path [OF loop])
  have pimage: path-image (linepath (1::complex) 1) ⊆ circle
    by (rule is-loop-image [OF loop])
```

```
  have heq:
      circle-lift (linepath (1::complex) 1) 1 = (λt. 0) 1
    using circle-path-lift-unique
      [OF chosen circle-path-lift-linepath ppath pimage] by simp
  have ep:
      circle-lift (linepath (1::complex) 1) 1 =
        of-int (circle-lift-endpoint (linepath 1 1))
    by (rule circle-lift-endpoint-spec [OF loop])
  have hreal:
      real-of-int (circle-lift-endpoint (linepath 1 1)) = 0
    using ep heq by simp
  have hint:
      real-of-int (circle-lift-endpoint (linepath 1 1)) =
        real-of-int 0
    using hreal by simp
  then show ?thesis
    by (metis of-int-eq-iff)
qed

lemma circle-lift-endpoint-cpow:
  circle-lift-endpoint (cpow n) = int n
proof (induction n)
  case 0
  have h: circle-lift-endpoint (linepath (1::complex) 1) = 0
    by (rule circle-lift-endpoint-linepath)
  then show ?case by (simp add: cpow.simps(1) linepath-refl)
next
  case (Suc n)
  have h:
      circle-lift-endpoint (cpow (Suc n)) =
        circle-lift-endpoint (circlepath 0 1) +
        circle-lift-endpoint (cpow n)
    unfolding cpow.simps(2)
    by (rule circle-lift-endpoint-join)
      (rule is-loop-circlepath, rule is-loop-cpow)
  then show ?case
    by (simp add: circle-lift-endpoint-circlepath Suc.IH)
qed

lemma circle-lift-endpoint-eq-wnd:
  assumes loop: is-loop p
  shows circle-lift-endpoint p = wnd p
proof −
  have class-of:
      loop-class p ∈ carrier pi1-circle
    using loop by (simp add: carrier-pi1)
  show ?thesis
  proof (cases wnd p ≥ 0)
    case True
    define r where r = cpow (nat (wnd p))
    have rloop: is-loop r
      unfolding r-def by (rule is-loop-cpow)
    have deg-eq: deg (loop-class p) = deg (loop-class r)
      using loop rloop by (simp add: deg-class r-def wnd-cpow True)
    have rcarrier: loop-class r ∈ carrier pi1-circle
      using rloop by (simp add: carrier-pi1)
    have rclass: loop-class p = loop-class r
    proof (rule inj-onD [OF deg-inj])
```

```
    show loop-class p ∈ carrier pi1-circle
      by (rule class-of)
    show loop-class r ∈ carrier pi1-circle
      by (rule rcarrier)
    show deg (loop-class p) = deg (loop-class r)
      by (rule deg-eq)
  qed
  have hom: homotopic-paths circle p r
    using loop-class-eq-iff [OF loop rloop] rclass by blast
  have endpoint-eq:
      circle-lift-endpoint p = circle-lift-endpoint r
    by (rule circle-lift-endpoint-homotopic [OF loop rloop hom])
  then show ?thesis
    using endpoint-eq r-def circle-lift-endpoint-cpow True by simp
next
  case False
  define r where r = reversepath (cpow (nat (− wnd p)))
  have rloop: is-loop r
    unfolding r-def by (simp add: is-loop-cpow)
  have wr: wnd r = wnd p
  proof −
    have h:
        wnd (reversepath (cpow (nat (− wnd p)))) =
          − int (nat (− wnd p))
      by (simp add: wnd-reversepath is-loop-cpow wnd-cpow)
    then show ?thesis
      unfolding r-def
      using False by simp
  qed
  have deg-eq: deg (loop-class p) = deg (loop-class r)
    using loop rloop
    by (simp add: deg-class wr)
  have rcarrier: loop-class r ∈ carrier pi1-circle
    using rloop by (simp add: carrier-pi1)
  have rclass: loop-class p = loop-class r
  proof (rule inj-onD [OF deg-inj])
    show loop-class p ∈ carrier pi1-circle
      by (rule class-of)
    show loop-class r ∈ carrier pi1-circle
      by (rule rcarrier)
    show deg (loop-class p) = deg (loop-class r)
      by (rule deg-eq)
  qed
  have hom: homotopic-paths circle p r
    using loop-class-eq-iff [OF loop rloop] rclass by blast
  have endpoint-eq:
      circle-lift-endpoint p = circle-lift-endpoint r
    by (rule circle-lift-endpoint-homotopic [OF loop rloop hom])
  have er:
      circle-lift-endpoint r =
        − circle-lift-endpoint (cpow (nat (− wnd p)))
    unfolding r-def
    by (rule circle-lift-endpoint-reverse [OF is-loop-cpow])
  then show ?thesis
    using endpoint-eq er circle-lift-endpoint-cpow False by simp
qed
qed
```

lemma *circle-lift-endpoint-eq-deg*:
assumes *is-loop p*
shows *circle-lift-endpoint p = deg (loop-class p)*
using *circle-lift-endpoint-eq-wnd* [*OF assms*]
deg-class [*OF assms*] **by** *simp*

end
theory *Covering-Spaces*
imports *HOL−Analysis.Homeomorphism*
begin

4 Generic pointed path lifts

The standard analysis library contains the point-set definition of a covering space and its lifting theorems. This theory packages those results in a small interface that can be reused without exposing the implementation of a particular covering map. The domain and codomain of the covering are real-normed vector spaces, exactly as required by the analytic covering-space API.

definition *covering-path-lift* ::
'a::*real-normed-vector set* ⇒
(*'a* ⇒ *'b*::*real-normed-vector*) ⇒
(*real* ⇒ *'b*) ⇒ *'a* ⇒ (*real* ⇒ *'a*) ⇒ *bool*
where
covering-path-lift C p g a h ⟷
path h ∧ *path-image h* ⊆ *C* ∧ *pathstart h = a* ∧
(∀ *t* ∈ {*0..1*}. *p* (*h t*) = *g t*)

lemma *covering-path-liftI*:
assumes *path h path-image h* ⊆ *C pathstart h = a*
and ⋀*t*. *t* ∈ {*0..1*} ⟹ *p* (*h t*) = *g t*
shows *covering-path-lift C p g a h*
using *assms* **unfolding** *covering-path-lift-def* **by** *blast*

lemma *covering-path-liftD*:
assumes *covering-path-lift C p g a h*
shows *path h path-image h* ⊆ *C pathstart h = a*
and ⋀*t*. *t* ∈ {*0..1*} ⟹ *p* (*h t*) = *g t*
using *assms* **unfolding** *covering-path-lift-def* **by** *blast+*

lemma *covering-path-lift-exists*:
assumes *cov*: *covering-space C p S* **and** *ac*: *a* ∈ *C*
and *pg*: *path g* **and** *ig*: *path-image g* ⊆ *S*
and *start*: *pathstart g = p a*
shows ∃ *h*. *covering-path-lift C p g a h*
proof −
obtain *h* **where** *path h path-image h* ⊆ *C pathstart h = a*
and *eq*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *p* (*h t*) = *g t*
using *covering-space-lift-path-strong* [*OF cov ac pg ig start*]
by *blast*
then show *?thesis* **by** (*intro exI*) (*rule covering-path-liftI*)
qed

lemma *covering-path-lift-unique*:
assumes *cov*: *covering-space C p S*
and *pg*: *path g* **and** *ig*: *path-image g* ⊆ *S*
and *h1*: *covering-path-lift C p g a h1*
and *h2*: *covering-path-lift C p g a h2*
shows ⋀*t*. *t* ∈ {*0..1*} ⟹ *h1 t = h2 t*

proof −
have *cg*: *continuous-on* $\{0..1\}$ *g*
using *pg* **unfolding** *path-def* **by** *blast*
have *ig′*: $g \in \{0..1\} \to S$
using *ig* **unfolding** *path-image-def* **by** *auto*
have *h1d*: *path h1 path-image h1* $\subseteq$ *C pathstart h1* $=$ *a*
and *h1eq*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow p\ (h1\ t) = g\ t$
using *covering-path-liftD* [*OF h1*] **by** *blast+*
have *h2d*: *path h2 path-image h2* $\subseteq$ *C pathstart h2* $=$ *a*
and *h2eq*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow p\ (h2\ t) = g\ t$
using *covering-path-liftD* [*OF h2*] **by** *blast+*
have *ch1*: *continuous-on* $\{0..1\}$ *h1*
using *h1d*(*1*) **unfolding** *path-def* **by** *blast*
have *ch2*: *continuous-on* $\{0..1\}$ *h2*
using *h2d*(*1*) **unfolding** *path-def* **by** *blast*
have *ih1*: $h1 \in \{0..1\} \to C$
using *h1d*(*2*) **unfolding** *path-image-def* **by** *auto*
have *ih2*: $h2 \in \{0..1\} \to C$
using *h2d*(*2*) **unfolding** *path-image-def* **by** *auto*
have *connected01*: *connected* ($\{0..1\}$::*real set*)
by *simp*
have *eq0*: *h1 0* $=$ *h2 0*
using *h1d*(*3*) *h2d*(*3*) **unfolding** *pathstart-def* **by** *simp*
have *h1eq′*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow g\ t = p\ (h1\ t)$
using *h1eq* **by** (*metis*)
have *h2eq′*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow g\ t = p\ (h2\ t)$
using *h2eq* **by** (*metis*)
show *h1 t* $=$ *h2 t* **if** $t \in \{0..1\}$ **for** *t*
apply (*rule covering-space-lift-unique*
[*of C p S h1 0 h2* $\{0..1\}$ *g t*])
using *cov eq0 cg ig′ ch1 ih1 h1eq′ ch2 ih2 h2eq′ connected01 that*
by *simp-all*
qed

definition *covering-path-lift-choice* ::
′a::*real-normed-vector set* ⇒
(*′a* ⇒ *′b*::*real-normed-vector*) ⇒
(*real* ⇒ *′b*) ⇒ *′a* ⇒ (*real* ⇒ *′a*)
where
covering-path-lift-choice C p g a $=$ (*SOME h. covering-path-lift C p g a h*)

lemma *covering-path-lift-choice-spec*:
assumes *covering-space C p S a* $\in$ *C path g*
path-image g $\subseteq$ *S pathstart g* $=$ *p a*
shows *covering-path-lift C p g a* (*covering-path-lift-choice C p g a*)
unfolding *covering-path-lift-choice-def*
using *someI-ex* [*OF covering-path-lift-exists* [*OF assms*(*1*−*5*)]] **.**

lemma *covering-path-lift-choice-unique*:
assumes *cov*: *covering-space C p S* **and** *ac*: $a \in C$
and *pg*: *path g* **and** *ig*: *path-image g* $\subseteq S$
and *start*: *pathstart g* $=$ *p a*
and *h*: *covering-path-lift C p g a h*
shows $\bigwedge t.\ t \in \{0..1\} \Longrightarrow$
covering-path-lift-choice C p g a t $=$ *h t*
proof −
have *choice*: *covering-path-lift C p g a*
(*covering-path-lift-choice C p g a*)

```
    by (rule covering-path-lift-choice-spec [OF cov ac pg ig start])
  show covering-path-lift-choice C p g a t = h t
    if t ∈ {0..1} for t
    using covering-path-lift-unique [OF cov pg ig choice h that] .
qed

definition covering-lift-endpoint ::
  'a::real-normed-vector set ⇒
   ('a ⇒ 'b::real-normed-vector) ⇒
   (real ⇒ 'b) ⇒ 'a ⇒ 'a
where
  covering-lift-endpoint C p g a = covering-path-lift-choice C p g a 1

lemma covering-lift-endpoint-eq:
  assumes cov: covering-space C p S and ac: a ∈ C
    and pg: path g and ig: path-image g ⊆ S
    and start: pathstart g = p a
    and h: covering-path-lift C p g a h
  shows covering-lift-endpoint C p g a = h 1
  unfolding covering-lift-endpoint-def
  by (rule covering-path-lift-choice-unique [OF cov ac pg ig start h]) simp

lemma covering-lift-endpoint-fiber:
  assumes cov: covering-space C p S and ac: a ∈ C
    and pg: path g and ig: path-image g ⊆ S
    and start: pathstart g = p a
  shows covering-lift-endpoint C p g a ∈ C
    and p (covering-lift-endpoint C p g a) = pathfinish g
proof −
  have h: covering-path-lift C p g a
      (covering-path-lift-choice C p g a)
    by (rule covering-path-lift-choice-spec [OF cov ac pg ig start])
  have hd: path (covering-path-lift-choice C p g a)
      path-image (covering-path-lift-choice C p g a) ⊆ C
      pathstart (covering-path-lift-choice C p g a) = a
    and heq: ⋀t. t ∈ {0..1} ⟹
      p (covering-path-lift-choice C p g a t) = g t
    using covering-path-liftD [OF h] by blast+
  show covering-lift-endpoint C p g a ∈ C
    using hd(2) unfolding covering-lift-endpoint-def path-image-def by auto
  show p (covering-lift-endpoint C p g a) = pathfinish g
    using heq [of 1] unfolding covering-lift-endpoint-def pathfinish-def by simp
qed

lemma covering-lift-endpoint-homotopic:
  assumes cov: covering-space C p S
    and pg1: path g1 and ig1: path-image g1 ⊆ S
    and pg2: path g2 and ig2: path-image g2 ⊆ S
    and hom: homotopic-paths S g1 g2
    and ac: a ∈ C
    and start1: pathstart g1 = p a and start2: pathstart g2 = p a
  shows covering-lift-endpoint C p g1 a =
      covering-lift-endpoint C p g2 a
proof −
  have h1: covering-path-lift C p g1 a
      (covering-path-lift-choice C p g1 a)
    by (rule covering-path-lift-choice-spec [OF cov ac pg1 ig1 start1])
  have h2: covering-path-lift C p g2 a
```

```
    (covering-path-lift-choice C p g2 a)
  by (rule covering-path-lift-choice-spec [OF cov ac pg2 ig2 start2])
have f1: pathfinish (covering-path-lift-choice C p g1 a) =
    pathfinish (covering-path-lift-choice C p g2 a)
  by (rule covering-space-monodromy [OF cov pg1 ig1 pg2 ig2 hom])
    (use covering-path-liftD [OF h1] covering-path-liftD [OF h2] in auto)
show ?thesis
  using f1 unfolding covering-lift-endpoint-def pathfinish-def by simp
qed
```

5 Pointed covering spaces

A pointed covering packages the sheet chosen above the base point. The locale is deliberately lightweight: later developments can instantiate it with the exponential circle cover, universal covers, or covering graphs.

```
locale pointed-covering-space =
  fixes C :: 'a::real-normed-vector set
    and p :: 'a ⇒ 'b::real-normed-vector
    and S :: 'b set
    and a0 :: 'a
  assumes covering: covering-space C p S
    and basepoint: a0 ∈ C
begin

lemma continuous:
  continuous-on C p
  using covering-space-imp-continuous [OF covering] .

lemma surjective:
  p ‘ C = S
  using covering-space-imp-surjective [OF covering] .

abbreviation lift-endpoint where
  lift-endpoint g ≡ covering-lift-endpoint C p g a0

lemma lift-endpoint-homotopic:
  assumes path g1 path-image g1 ⊆ S
    path g2 path-image g2 ⊆ S
    homotopic-paths S g1 g2
    pathstart g1 = p a0 pathstart g2 = p a0
  shows lift-endpoint g1 = lift-endpoint g2
  using covering-lift-endpoint-homotopic [OF covering assms(1−5) basepoint assms(6−7)] .

end

end
theory Circle-Product
  imports Circle-Covering Covering-Spaces
begin
```

6 Product paths and torus loops

The product construction is the first genuinely functorial extension of the circle calculation. We keep the path-level presentation explicit: this makes the two integer coordinates available independently of the quotient presentation constructed in the torus fundamental-group theory.

```
definition product-path ::
```

```
  (real ⇒ 'a::topological-space) ⇒
   (real ⇒ 'b::topological-space) ⇒
   (real ⇒ 'a × 'b)
where
  product-path p q = (λt. (p t, q t))

lemma path-product-path:
  assumes path p path q
  shows path (product-path p q)
  using assms unfolding product-path-def path-def
  by (intro continuous-on-Pair)

lemma path-image-product-path:
  path-image (product-path p q) ⊆ path-image p × path-image q
  by (auto simp: product-path-def path-image-def)

lemma pathstart-product-path [simp]:
  pathstart (product-path p q) = (pathstart p, pathstart q)
  by (simp add: product-path-def pathstart-def)

lemma pathfinish-product-path [simp]:
  pathfinish (product-path p q) = (pathfinish p, pathfinish q)
  by (simp add: product-path-def pathfinish-def)

definition torus-loop :: (real ⇒ complex × complex) ⇒ bool
where
  torus-loop p ⟷
     path p ∧ path-image p ⊆ circle × circle ∧
     pathstart p = (1, 1) ∧ pathfinish p = (1, 1)

lemma torus-loopD:
  assumes torus-loop p
  shows path p path-image p ⊆ circle × circle
    pathstart p = (1, 1) pathfinish p = (1, 1)
  using assms unfolding torus-loop-def by blast+

lemma torus-loop-fst:
  assumes torus-loop p
  shows is-loop (fst ∘ p)
  using torus-loopD [OF assms]
  by (auto simp: is-loop-def path-def path-image-def pathstart-def
      pathfinish-def continuous-on-fst)

lemma torus-loop-snd:
  assumes torus-loop p
  shows is-loop (snd ∘ p)
  using torus-loopD [OF assms]
  by (auto simp: is-loop-def path-def path-image-def pathstart-def
      pathfinish-def continuous-on-snd)

lemma torus-loop-product:
  assumes is-loop p is-loop q
  shows torus-loop (product-path p q)
proof −
  have im: path-image (product-path p q) ⊆
      path-image p × path-image q
    by (rule path-image-product-path)
  show ?thesis
```

```
    using assms im unfolding torus-loop-def
    by (auto simp: is-loop-def path-product-path)
qed

definition torus-lift-endpoint ::
  (real ⇒ complex × complex) ⇒ int × int
where
  torus-lift-endpoint p =
    (circle-lift-endpoint (fst ∘ p),
     circle-lift-endpoint (snd ∘ p))

lemma torus-lift-endpoint-fst [simp]:
  fst (torus-lift-endpoint p) = circle-lift-endpoint (fst ∘ p)
  by (simp add: torus-lift-endpoint-def)

lemma torus-lift-endpoint-snd [simp]:
  snd (torus-lift-endpoint p) = circle-lift-endpoint (snd ∘ p)
  by (simp add: torus-lift-endpoint-def)

lemma torus-loop-join:
  assumes torus-loop p torus-loop q
  shows torus-loop (p +++ q)
proof −
  have tp: path p path-image p ⊆ circle × circle
      pathstart p = (1, 1) pathfinish p = (1, 1)
    using torus-loopD [OF assms(1)] by blast+
  have tq: path q path-image q ⊆ circle × circle
      pathstart q = (1, 1) pathfinish q = (1, 1)
    using torus-loopD [OF assms(2)] by blast+
  have conn: pathfinish p = pathstart q
    using tp(4) tq(3) by simp
  have pp: path (p +++ q)
    by (rule path-join-imp [OF tp(1) tq(1) conn])
  have pi: path-image (p +++ q) ⊆ circle × circle
    by (rule subset-path-image-join)
       (use tp(2) tq(2) in auto)
  show ?thesis
    using pp pi tp tq
    unfolding torus-loop-def by simp
qed

lemma torus-loop-reverse:
  assumes torus-loop p
  shows torus-loop (reversepath p)
  using torus-loopD [OF assms]
  by (simp add: torus-loop-def)

lemma product-path-join:
  product-path (p +++ r) (q +++ s) =
    product-path p q +++ product-path r s
  by (rule ext) (simp add: product-path-def joinpaths-def)

lemma product-path-reverse:
  product-path (reversepath p) (reversepath q) =
    reversepath (product-path p q)
  by (rule ext) (simp add: product-path-def reversepath-def)

lemma torus-lift-endpoint-join:
```

```
  assumes torus-loop p torus-loop q
  shows torus-lift-endpoint (p +++ q) =
    torus-lift-endpoint p + torus-lift-endpoint q
proof −
  have hp1: is-loop (fst ∘ p) by (rule torus-loop-fst [OF assms(1)])
  have hq1: is-loop (fst ∘ q) by (rule torus-loop-fst [OF assms(2)])
  have hp2: is-loop (snd ∘ p) by (rule torus-loop-snd [OF assms(1)])
  have hq2: is-loop (snd ∘ q) by (rule torus-loop-snd [OF assms(2)])
  have fcomp: fst ∘ (p +++ q) = (fst ∘ p) +++ (fst ∘ q)
    by (rule path-compose-join)
  have scomp: snd ∘ (p +++ q) = (snd ∘ p) +++ (snd ∘ q)
    by (rule path-compose-join)
  show ?thesis
    unfolding torus-lift-endpoint-def
    by (simp add: fcomp scomp circle-lift-endpoint-join [OF hp1 hq1]
      circle-lift-endpoint-join [OF hp2 hq2])
qed

lemma torus-lift-endpoint-reverse:
  assumes torus-loop p
  shows torus-lift-endpoint (reversepath p) =
    − torus-lift-endpoint p
proof −
  have hp1: is-loop (fst ∘ p) by (rule torus-loop-fst [OF assms])
  have hp2: is-loop (snd ∘ p) by (rule torus-loop-snd [OF assms])
  have fcomp:
      fst ∘ reversepath p = reversepath (fst ∘ p)
    by (rule path-compose-reversepath)
  have scomp:
      snd ∘ reversepath p = reversepath (snd ∘ p)
    by (rule path-compose-reversepath)
  show ?thesis
    unfolding torus-lift-endpoint-def
    by (simp add: fcomp scomp circle-lift-endpoint-reverse [OF hp1]
      circle-lift-endpoint-reverse [OF hp2])
qed

lemma homotopic-paths-product:
  fixes S :: 'a::topological-space set and T :: 'b::topological-space set
    and p q :: real ⇒ 'a and r s :: real ⇒ 'b
  assumes h1: homotopic-paths S p q
    and h2: homotopic-paths T r s
  shows homotopic-paths (S × T)
    (product-path p r) (product-path q s)
proof −
  obtain h where conth: continuous-on ({0..1} × {0..1}) h
    and him: h ∈ ({0..1} × {0..1}) → S
    and h0: ∀ x ∈ {0..1}. h (0,x) = p x
    and h1x: ∀ x ∈ {0..1}. h (1,x) = q x
    and hendraw: ∀ t ∈ {0..1::real}.
      pathstart (h ∘ Pair t) = pathstart p ∧
      pathfinish (h ∘ Pair t) = pathfinish p
    using homotopic-paths [THEN iffD1, OF h1] by blast
  have hend: ⋀t. t ∈ {0..1} ⟹
      h (t,0) = p 0 ∧ h (t,1) = p 1
    using hendraw by (auto simp: pathstart-def pathfinish-def)
  obtain k where contk: continuous-on ({0..1} × {0..1}) k
    and kim: k ∈ ({0..1} × {0..1}) → T
```

```
    and k0: ∀ x ∈ {0..1}. k (0,x) = r x
    and k1: ∀ x ∈ {0..1}. k (1,x) = s x
    and kendraw: ∀ t ∈ {0..1::real}.
      pathstart (k ∘ Pair t) = pathstart r ∧
      pathfinish (k ∘ Pair t) = pathfinish r
    using homotopic-paths [THEN iffD1, OF h2] by blast
  have kend: ⋀t. t ∈ {0..1} ⟹
      k (t,0) = r 0 ∧ k (t,1) = r 1
    using kendraw by (auto simp: pathstart-def pathfinish-def)
  define H where H = (λz. (h z, k z))
  have Hc: continuous-on ({0..1} × {0..1}) H
    unfolding H-def
    by (intro continuous-on-Pair conth contk)
  have Him: H ∈ ({0..1} × {0..1}) → S × T
    using him kim unfolding H-def by auto
  have H0: ∀ x ∈ {0..1}.
      H (0, x) = product-path p r x
    using h0 k0 unfolding H-def product-path-def by auto
  have Htop: ∀ x ∈ {0..1}.
      H (1, x) = product-path q s x
    using h1x k1 unfolding H-def product-path-def by auto
  have Hends: ∀ t ∈ {0..1::real}.
      pathstart (H ∘ Pair t) =
        pathstart (product-path p r) ∧
      pathfinish (H ∘ Pair t) =
        pathfinish (product-path p r)
    using hend kend
    by (auto simp: H-def product-path-def pathstart-def pathfinish-def o-def)
  show ?thesis
    apply (rule homotopic-paths [THEN iffD2])
    apply (rule exI[of - H])
    using Hc Him H0 Htop Hends by blast
qed

lemma torus-loop-homotopic-iff:
  assumes p: torus-loop p and q: torus-loop q
  shows homotopic-paths (circle × circle) p q ⟷
    torus-lift-endpoint p = torus-lift-endpoint q
proof
  assume h: homotopic-paths (circle × circle) p q
  have hf: homotopic-paths circle (fst ∘ p) (fst ∘ q)
    apply (rule homotopic-paths-continuous-image [OF h])
    apply (intro continuous-intros)
    by auto
  have hs: homotopic-paths circle (snd ∘ p) (snd ∘ q)
    apply (rule homotopic-paths-continuous-image [OF h])
    apply (intro continuous-intros)
    by auto
  have ef: circle-lift-endpoint (fst ∘ p) =
      circle-lift-endpoint (fst ∘ q)
    by (rule circle-lift-endpoint-homotopic
      [OF torus-loop-fst [OF p] torus-loop-fst [OF q] hf])
  have es: circle-lift-endpoint (snd ∘ p) =
      circle-lift-endpoint (snd ∘ q)
    by (rule circle-lift-endpoint-homotopic
      [OF torus-loop-snd [OF p] torus-loop-snd [OF q] hs])
  show torus-lift-endpoint p = torus-lift-endpoint q
    using ef es unfolding torus-lift-endpoint-def by simp
```

```
next
  assume e: torus-lift-endpoint p = torus-lift-endpoint q
  have ef: circle-lift-endpoint (fst ∘ p) =
      circle-lift-endpoint (fst ∘ q)
    using e unfolding torus-lift-endpoint-def by simp
  have es: circle-lift-endpoint (snd ∘ p) =
      circle-lift-endpoint (snd ∘ q)
    using e unfolding torus-lift-endpoint-def by simp
  have hf: homotopic-paths circle (fst ∘ p) (fst ∘ q)
    by (rule circle-lift-endpoint-eq-homotopic
      [OF torus-loop-fst [OF p] torus-loop-fst [OF q] ef])
  have hs: homotopic-paths circle (snd ∘ p) (snd ∘ q)
    by (rule circle-lift-endpoint-eq-homotopic
      [OF torus-loop-snd [OF p] torus-loop-snd [OF q] es])
  have hp: homotopic-paths (circle × circle)
      (product-path (fst ∘ p) (snd ∘ p))
      (product-path (fst ∘ q) (snd ∘ q))
    by (rule homotopic-paths-product [OF hf hs])
  show homotopic-paths (circle × circle) p q
    using hp by (simp add: product-path-def)
qed

end
theory Torus-Fundamental-Group
  imports Circle-Torus-Algebraic-Topology.Circle-Product
begin
```

7 The fundamental group of the torus

Elements are homotopy classes of loops based at (*1*,*1*). Multiplication is induced by path concatenation. The two lift endpoints give an explicit isomorphism to the product of the additive integer groups.

```
definition torus-class where
  torus-class p = {q. torus-loop q ∧ homotopic-paths (circle × circle) p q}

definition torus-classes where
  torus-classes = {torus-class p |p. torus-loop p}

definition torus-rep where
  torus-rep A = (SOME p. torus-loop p ∧ torus-class p = A)

lemma torus-class-mem:
  torus-loop p ⟹ torus-class p ∈ torus-classes
  by (auto simp: torus-classes-def)

lemma torus-class-self:
  torus-loop p ⟹ p ∈ torus-class p
  by (simp add: torus-class-def torus-loop-def)

lemma torus-class-eq-iff:
  assumes torus-loop p torus-loop q
  shows torus-class p = torus-class q ⟷
    torus-lift-endpoint p = torus-lift-endpoint q
proof
  assume torus-class p = torus-class q
  then have q ∈ torus-class p
    using torus-class-self[OF assms(2)] by simp
```

then show *torus-lift-endpoint p = torus-lift-endpoint q*
using *assms* **by** (*simp add*: *torus-class-def torus-loop-homotopic-iff*)
next
assume *torus-lift-endpoint p = torus-lift-endpoint q*
with *assms* **show** *torus-class p = torus-class q*
by (*auto simp*: *torus-class-def torus-loop-homotopic-iff*)
qed

lemma *torus-rep-props*:
assumes *A* ∈ *torus-classes*
shows *torus-loop* (*torus-rep A*) ∧ *torus-class* (*torus-rep A*) = *A*
proof −
have ∃ *p*. *torus-loop p* ∧ *torus-class p* = *A*
using *assms* **by** (*auto simp*: *torus-classes-def*)
then show *?thesis* **unfolding** *torus-rep-def* **by** (*rule someI-ex*)
qed

definition *torus-degree* **where**
torus-degree A = *torus-lift-endpoint* (*torus-rep A*)

lemma *torus-degree-class*:
assumes *torus-loop p*
shows *torus-degree* (*torus-class p*) = *torus-lift-endpoint p*
using *torus-rep-props*[*OF torus-class-mem*[*OF assms*]] *assms*
by (*auto simp*: *torus-degree-def torus-class-eq-iff*)

lemma *torus-degree-inj*: *inj-on torus-degree torus-classes*
proof (*rule inj-onI*)
fix *A B*
assume *A*: *A* ∈ *torus-classes* **and** *B*: *B* ∈ *torus-classes*
and *eq*: *torus-degree A* = *torus-degree B*
have *torus-class* (*torus-rep A*) = *torus-class* (*torus-rep B*)
apply (*rule iffD2*[*OF torus-class-eq-iff*])
using *torus-rep-props*[*OF A*] *torus-rep-props*[*OF B*] *eq*
by (*auto simp*: *torus-degree-def*)
then show *A* = *B* **using** *torus-rep-props A B* **by** *metis*
qed

lemma *torus-endpoint-product*:
torus-lift-endpoint (*product-path p q*) =
(*circle-lift-endpoint p*, *circle-lift-endpoint q*)
by (*simp add*: *torus-lift-endpoint-def product-path-def o-def*)

lemma *circle-endpoint-surj*:
obtains *p* **where** *is-loop p circle-lift-endpoint p* = *n*
proof −
have *n* ∈ *deg* ‘ *carrier pi1-circle* **by** (*subst deg-surj*) *simp*
then obtain *A* **where** *A*: *A* ∈ *carrier pi1-circle deg A* = *n*
by *blast*
have *L*: *is-loop* (*rep A*) **and** *C*: *loop-class* (*rep A*) = *A*
using *A rep-props* **by** *auto*
have *circle-lift-endpoint* (*rep A*) = *n*
using *A C circle-lift-endpoint-eq-wnd*[*OF L*] *deg-class*[*OF L*] **by** *simp*
then show *thesis* **using** *that L* **by** *blast*
qed

lemma *torus-degree-surj*: *torus-degree* ‘ *torus-classes* = *UNIV*
proof (*rule UNIV-eq-I*[*THEN sym*])

```
  fix z :: int × int
  obtain p where p: is-loop p circle-lift-endpoint p = fst z
    by (rule circle-endpoint-surj)
  obtain q where q: is-loop q circle-lift-endpoint q = snd z
    by (rule circle-endpoint-surj)
  have L: torus-loop (product-path p q)
    by (rule torus-loop-product[OF p(1) q(1)])
  have torus-degree (torus-class (product-path p q)) = z
    by (simp add: torus-degree-class[OF L] torus-endpoint-product p q)
  then show z ∈ torus-degree ‘ torus-classes
    using torus-class-mem[OF L] by blast
qed

definition pi1-torus :: (real ⇒ complex × complex) set monoid where
  pi1-torus = (|carrier = torus-classes,
    mult = (λA B. torus-class (torus-rep A +++ torus-rep B)),
    one = torus-class (product-path (linepath 1 1) (linepath 1 1))|)

lemma torus-mult-class:
  assumes torus-loop p torus-loop q
  shows torus-class p ⊗pi1-torus torus-class q = torus-class (p +++ q)
proof −
  have p: torus-loop (torus-rep (torus-class p))
    torus-lift-endpoint (torus-rep (torus-class p)) = torus-lift-endpoint p
    using torus-rep-props[OF torus-class-mem[OF assms(1)]] assms(1)
    by (auto simp: torus-class-eq-iff)
  have q: torus-loop (torus-rep (torus-class q))
    torus-lift-endpoint (torus-rep (torus-class q)) = torus-lift-endpoint q
    using torus-rep-props[OF torus-class-mem[OF assms(2)]] assms(2)
    by (auto simp: torus-class-eq-iff)
  show ?thesis
    unfolding pi1-torus-def
    using assms p q
    by (simp add: torus-class-eq-iff torus-loop-join torus-lift-endpoint-join)
qed

lemma torus-mult-closed:
  assumes A ∈ torus-classes B ∈ torus-classes
  shows A ⊗pi1-torus B ∈ torus-classes
  using torus-rep-props assms
  by (auto simp: pi1-torus-def intro: torus-class-mem torus-loop-join)

lemma torus-degree-mult:
  assumes A ∈ torus-classes B ∈ torus-classes
  shows torus-degree (A ⊗pi1-torus B) = torus-degree A + torus-degree B
proof −
  have A: torus-loop (torus-rep A) and B: torus-loop (torus-rep B)
    using torus-rep-props assms by blast+
  have torus-degree (A ⊗pi1-torus B) =
      torus-lift-endpoint (torus-rep A +++ torus-rep B)
    by (simp only: pi1-torus-def monoid.select-convs
        torus-degree-class[OF torus-loop-join[OF A B]])
  then show ?thesis
    by (simp add: torus-lift-endpoint-join[OF A B] torus-degree-def)
qed

lemma torus-one-closed: 1pi1-torus ∈ torus-classes
  by (simp add: pi1-torus-def torus-class-mem torus-loop-product)
```

lemma *torus-degree-one*: *torus-degree* $\mathbf{1}_{pi1\text{-}torus} = 0$
by (*simp add*: *pi1-torus-def torus-degree-class torus-loop-product*
torus-endpoint-product circle-lift-endpoint-linepath zero-prod-def)

theorem *group-pi1-torus*: *group pi1-torus*
proof (*rule groupI*)
show $x \otimes_{pi1\text{-}torus} y \in$ *carrier pi1-torus*
if $x \in$ *carrier pi1-torus* $y \in$ *carrier pi1-torus* **for** x y
using *that torus-mult-closed* **by** (*simp add*: *pi1-torus-def*)
show $\mathbf{1}_{pi1\text{-}torus} \in$ *carrier pi1-torus*
using *torus-one-closed* **by** (*simp add*: *pi1-torus-def*)
show $(x \otimes_{pi1\text{-}torus} y) \otimes_{pi1\text{-}torus} z =$
$x \otimes_{pi1\text{-}torus} (y \otimes_{pi1\text{-}torus} z)$
if $x \in$ *carrier pi1-torus* $y \in$ *carrier pi1-torus* $z \in$ *carrier pi1-torus* **for** x y z
proof −
have *C*: $x \in$ *torus-classes* $y \in$ *torus-classes* $z \in$ *torus-classes*
using *that* **by** (*simp-all add*: *pi1-torus-def*)
show *?thesis*
by (*rule inj-onD*[*OF torus-degree-inj*])
(*simp-all add*: *torus-degree-mult torus-mult-closed C add.assoc*)
qed
show $\mathbf{1}_{pi1\text{-}torus} \otimes_{pi1\text{-}torus} x = x$
if $x \in$ *carrier pi1-torus* **for** x
proof −
have *C*: $x \in$ *torus-classes* **using** *that* **by** (*simp add*: *pi1-torus-def*)
show *?thesis*
by (*rule inj-onD*[*OF torus-degree-inj*])
(*simp-all add*: *torus-degree-mult torus-mult-closed torus-one-closed torus-degree-one C*)
qed
show $\exists\, y{\in}$*carrier pi1-torus*. $y \otimes_{pi1\text{-}torus} x = \mathbf{1}_{pi1\text{-}torus}$
if $x \in$ *carrier pi1-torus* **for** x
proof −
have *C*: $x \in$ *torus-classes* **using** *that* **by** (*simp add*: *pi1-torus-def*)
have − *torus-degree* $x \in$ *torus-degree* ‘ *torus-classes*
by (*simp add*: *torus-degree-surj*)
then obtain y **where** *Y*: $y \in$ *torus-classes torus-degree* $y = -$ *torus-degree* x
by (*auto simp*: *image-iff*)
have $y \otimes_{pi1\text{-}torus} x = \mathbf{1}_{pi1\text{-}torus}$
by (*rule inj-onD*[*OF torus-degree-inj*])
(*simp-all add*: *torus-degree-mult torus-mult-closed torus-one-closed torus-degree-one C Y*)
with *Y* **show** *?thesis* **by** (*auto simp*: *pi1-torus-def*)
qed
qed

lemma *torus-inverse-class*:
assumes *torus-loop p*
shows $inv_{pi1\text{-}torus}$ (*torus-class p*) = *torus-class* (*reversepath p*)
proof (*rule group.inv-equality*[*OF group-pi1-torus*])
have *C*: *torus-class p* ∈ *torus-classes torus-class* (*reversepath p*) ∈ *torus-classes*
using *assms* **by** (*auto intro*: *torus-class-mem torus-loop-reverse*)
show *torus-class* (*reversepath p*) $\otimes_{pi1\text{-}torus}$ *torus-class p* = $\mathbf{1}_{pi1\text{-}torus}$
by (*rule inj-onD*[*OF torus-degree-inj*])
(*simp-all add*: *torus-degree-mult C torus-degree-class torus-loop-reverse assms*
torus-lift-endpoint-reverse torus-degree-one torus-mult-closed torus-one-closed)
show *torus-class p* ∈ *carrier pi1-torus torus-class* (*reversepath p*) ∈ *carrier pi1-torus*
using *C* **by** (*simp-all add*: *pi1-torus-def*)
qed

theorem *torus-degree-iso*:
torus-degree ∈ *iso pi1-torus*
(*Fundamental-Group-Circle.integer-group* ×× *Fundamental-Group-Circle.integer-group*)
proof (*rule isoI*)
show *torus-degree* ∈ *hom pi1-torus*
(*Fundamental-Group-Circle.integer-group* ×× *Fundamental-Group-Circle.integer-group*)
using *torus-degree-mult*
by (*auto simp*: *hom-def pi1-torus-def DirProd-def plus-prod-def split*: *prod.splits*)
show *bij-betw torus-degree* (*carrier pi1-torus*)
(*carrier* (*Fundamental-Group-Circle.integer-group* ×× *Fundamental-Group-Circle.integer-group*))
using *torus-degree-inj torus-degree-surj*
by (*simp add*: *bij-betw-def pi1-torus-def DirProd-def*)
qed

theorem *pi1-torus-iso-integer-product*:
pi1-torus ≅
(*Fundamental-Group-Circle.integer-group* ×× *Fundamental-Group-Circle.integer-group*)
by (*rule is-isoI*[*OF torus-degree-iso*])

end
theory *Circle-Radial-Retract*
imports *Circle-Covering*
begin

8 The punctured plane retracts onto the circle

The preceding circle entry already contains the analytic radial map *retr*. We expose it here with the geometric name used by the later homotopy and homology developments, together with its subspace-topology interface.

definition *punctured-plane* :: *complex set* **where**
punctured-plane = − {*0*}

definition *radial-retraction* :: *complex* ⇒ *complex* **where**
radial-retraction z = *retr z*

lemma *punctured-plane-eq*:
punctured-plane = − {*0*}
by (*simp add*: *punctured-plane-def*)

lemma *radial-retraction-continuous*:
continuous-on punctured-plane radial-retraction
using *continuous-on-retr*
by (*simp add*: *punctured-plane-def radial-retraction-def*)

lemma *radial-retraction-into-circle*:
radial-retraction ∈ *punctured-plane* → *circle*
unfolding *radial-retraction-def punctured-plane-def*
by (*rule retr-into-circle*)

lemma *radial-retraction-fixes*:
assumes *z* ∈ *circle*
shows *radial-retraction z* = *z*
using *assms* **by** (*simp add*: *radial-retraction-def retr-fixes*)

lemma *radial-retraction-maps*:
retraction-maps (*top-of-set punctured-plane*) (*top-of-set circle*)

```
    radial-retraction id
  unfolding retraction-maps-def
proof (intro conjI)
  show continuous-map (top-of-set punctured-plane)
    (top-of-set circle) radial-retraction
    using radial-retraction-continuous radial-retraction-into-circle
    by (simp add: continuous-map-in-subtopology)
next
  show continuous-map (top-of-set circle)
    (top-of-set punctured-plane) id
    using zero-notin-circle
    by (auto simp: continuous-map-in-subtopology punctured-plane-def)
next
  show ∀ x ∈ topspace (top-of-set circle). radial-retraction (id x) = x
    using radial-retraction-fixes by auto
qed

definition radial-homotopy :: real × complex ⇒ complex where
  radial-homotopy z =
    ((1 − fst z) ∗ (1 / norm (snd z)) + fst z) ∗R snd z

lemma radial-homotopy-coeff-pos:
  assumes t ∈ {0..1} z ∈ punctured-plane
  shows 0 < (1 − t) ∗ (1 / norm z) + t
proof −
  have z0: z ≠ 0
    using assms(2) by (simp add: punctured-plane-def)
  have nz: 0 < norm z
    using z0 by simp
  have t0: 0 ≤ t and t1: t ≤ 1
    using assms(1) by auto
  have ni: 0 < 1 / norm z
    using nz by (simp add: nz)
  have prod: 0 ≤ (1 − t) ∗ (1 / norm z)
    using t1 ni by (intro mult-nonneg-nonneg) (linarith, linarith)
  show ?thesis
  proof (cases t = 0)
    case True
    then show ?thesis using ni by simp
  next
    case False
    then have 0 < t using t0 by linarith
    then show ?thesis using prod by linarith
  qed
qed

lemma radial-homotopy-into-punctured-plane:
  radial-homotopy ∈ ({0..1} × punctured-plane) → punctured-plane
proof
  fix x :: real × complex
  assume x: x ∈ ({0..1} × punctured-plane)
  obtain t :: real and z :: complex where xeq: x = (t,z)
    by (cases x) simp
  have tz: (t,z) ∈ ({0..1} × punctured-plane)
    using x xeq by simp
  have tI: t ∈ {0..1} and zP: z ∈ punctured-plane
    using tz by auto
  have coeff: 0 < (1 − t) ∗ (1 / norm z) + t
```

```
    by (rule radial-homotopy-coeff-pos [OF tI zP])
  have nz: 0 < norm z
    using zP by (simp add: punctured-plane-def)
  have hn: 0 < norm (radial-homotopy (t,z))
    unfolding radial-homotopy-def
    using coeff nz
    by (simp add: norm-scaleR abs-of-pos coeff)
  have hne: radial-homotopy (t,z) ≠ 0
    using hn by auto
  show radial-homotopy x ∈ punctured-plane
    using hne xeq by (simp add: punctured-plane-def)
qed

lemma radial-homotopy-continuous:
  continuous-on ({0..1} × punctured-plane) radial-homotopy
  unfolding radial-homotopy-def
  apply (intro continuous-intros)
  by (auto simp: punctured-plane-def)

lemma radial-homotopy-continuous-map:
  continuous-map (prod-topology (top-of-set {0..1}) (top-of-set punctured-plane))
      (top-of-set punctured-plane) radial-homotopy
  using radial-homotopy-continuous radial-homotopy-into-punctured-plane
  by (simp add: continuous-map-in-subtopology prod-topology-subtopology-eu)

lemma radial-homotopy-zero:
  radial-homotopy (0,z) = radial-retraction z
  by (simp add: radial-homotopy-def radial-retraction-def retr-def
      scaleR-conv-of-real divide-inverse mult.commute)

lemma radial-homotopy-one:
  radial-homotopy (1,z) = id z
  by (simp add: radial-homotopy-def)

lemma radial-deformation:
  homotopic-with (λf. True) (top-of-set punctured-plane) (top-of-set punctured-plane)
      radial-retraction id
  unfolding homotopic-with-def
proof (intro exI conjI)
  show continuous-map (prod-topology (top-of-set {0..1}) (top-of-set punctured-plane))
      (top-of-set punctured-plane) radial-homotopy
    by (rule radial-homotopy-continuous-map)
  show ∀x. radial-homotopy (0,x) = radial-retraction x
    using radial-homotopy-zero by simp
  show ∀x. radial-homotopy (1,x) = id x
    using radial-homotopy-one by simp
  show ∀t ∈ {0..1::real}. True
    by simp
qed

lemma punctured-plane-homotopy-equivalent-circle:
  top-of-set punctured-plane homotopy-equivalent-space top-of-set circle
  by (rule deformation-retract-imp-homotopy-equivalent-space
      [OF radial-deformation radial-retraction-maps])

end
theory Circle-SVK
  imports Circle-Radial-Retract Seifert−Van−Kampen.Seifert-Van-Kampen
```

begin

9 Seifert–van Kampen interface

The substantial open-cover construction lives in the AFP *Seifert−Van−Kampen* entry. This theory makes that result a named layer of this development, so later circle and punctured-plane applications can depend on the established carrier-level fundamental-group interface.

context *classical-svk-setup*
begin

theorem *svk-classical-interface*:
bij-betw classical-svk-quotient-map (*fundamental-group-space W x0*)
(*carrier-full-amalgamated-free-product-space G1 G2 H i1 i2 mult1 one1 mult2 one2*)
by (*rule classical-seifert-van-kampen-bij-betw*)

end

context *topological-svk*
begin

interpretation *svk-carrier-interface*:
carrier-svk-encode-decode-interface
G1 G2 H i1 i2 mult1 one1 mult2 one2 encode svk-decode
by (*rule svk-locale*)

theorem *svk-topological-interface*:
bij-betw svk-carrier-interface.carrier-svk-quotient-map UNIV
(*carrier-full-amalgamated-free-product-space G1 G2 H i1 i2 mult1 one1 mult2 one2*)
by (*rule svk-carrier-interface.carrier-seifert-van-kampen-bij-betw*)

end

end
theory *Circle-Homology-Degree*
imports *Circle-Radial-Retract HOL−Homology.Homology*
begin

10 Homology of the circle and the punctured plane

The standard Isabelle homology library computes the reduced homology of an abstract sphere. The geometric circle used by the preceding theories is identified with the one-sphere using the library's Euclidean-coordinate homeomorphism, and the radial deformation retract then transfers the result to the punctured plane.

lemma *reduced-homology-group-nsphere-one*:
reduced-homology-group (*int* (*1*::*nat*)) (*nsphere* (*1*::*nat*)) ≅ *integer-group*
by (*metis reduced-homology-group-nsphere*(*1*))

lemma *reduced-homology-group-nsphere-one-other*:
p ≠ *int* (*1*::*nat*) ⟹
trivial-group (*reduced-homology-group p* (*nsphere* (*1*::*nat*)))
by (*rule reduced-homology-group-nsphere*(*2*))

lemma *circle-homeomorphic-nsphere-one*:
top-of-set circle homeomorphic-space nsphere 1
proof −

```
  have independent-basis: independent (Basis :: complex set)
    by (rule independent-Basis)
  have orthogonal-basis: pairwise orthogonal (Basis :: complex set)
    by (rule orthogonal-Basis)
  have card-basis: card (Basis :: complex set) = DIM(complex)
    using dim-eq-card-independent [OF independent-basis]
    by (simp add: span-Basis)
  have dim-complex: DIM(complex) = 2
    by simp
  have nonzero-dim: DIM(complex) ≠ 0
    by (simp add: dim-complex)
  have norm-basis: ⋀u. u ∈ (Basis :: complex set) ⟹ norm u = 1
    by (rule norm-Basis)
  obtain f g where
    hm: homeomorphic-maps (nsphere (DIM(complex) − 1))
      (top-of-set (sphere 0 1 ∩ span (Basis :: complex set))) f g
  proof (rule homeomorphic-maps-nsphere-euclidean-sphere)
    show independent (Basis :: complex set)
      by (rule independent-basis)
    show pairwise orthogonal (Basis :: complex set)
      by (rule orthogonal-basis)
    show card (Basis :: complex set) = DIM(complex)
      by (rule card-basis)
    show DIM(complex) ≠ 0
      by (rule nonzero-dim)
    show ⋀u. u ∈ (Basis :: complex set) ⟹ norm u = 1
      by (rule norm-basis)
  qed
  have hm': homeomorphic-maps (nsphere 1)
     (top-of-set (sphere (0::complex) 1)) f g
    using hm by (simp add: dim-complex)
  have hm'': homeomorphic-maps (nsphere 1) (top-of-set circle) f g
    using hm' by (simp add: circle-def)
  have nsphere 1 homeomorphic-space top-of-set circle
    by (rule homeomorphic-maps-imp-homeomorphic-space [OF hm''])
  then show ?thesis
    by (rule iffD1 [OF homeomorphic-space-sym])
qed

lemma reduced-homology-group-circle-one:
  reduced-homology-group (int (1::nat)) (top-of-set circle) ≅ integer-group
proof −
  have h:
      reduced-homology-group (int (1::nat)) (top-of-set circle) ≅
       reduced-homology-group (int (1::nat)) (nsphere (1::nat))
    by (rule homeomorphic-space-imp-isomorphic-reduced-homology-groups
        [OF circle-homeomorphic-nsphere-one])
  then show ?thesis
    using reduced-homology-group-nsphere-one by (rule iso-trans)
qed

lemma reduced-homology-group-circle-transfer:
  reduced-homology-group p (top-of-set circle) ≅
   reduced-homology-group p (nsphere (1::nat))
  by (rule homeomorphic-space-imp-isomorphic-reduced-homology-groups
      [OF circle-homeomorphic-nsphere-one])

lemma reduced-homology-group-circle-other:
```

```
  p ≠ int (1::nat) ⟹
   trivial-group (reduced-homology-group p (top-of-set circle))
proof −
  assume p: p ≠ int (1::nat)
  have h:
      reduced-homology-group p (top-of-set circle) ≅
       reduced-homology-group p (nsphere (1::nat))
    by (rule reduced-homology-group-circle-transfer)
  have hn: trivial-group (reduced-homology-group p (nsphere (1::nat)))
    using p by (metis reduced-homology-group-nsphere(2))
  have gc: group (reduced-homology-group p (top-of-set circle))
    by simp
  have gs: group (reduced-homology-group p (nsphere (1::nat)))
    by simp
  have hi:
      trivial-group (reduced-homology-group p (top-of-set circle)) ⟷
       trivial-group (reduced-homology-group p (nsphere (1::nat)))
    by (rule isomorphic-group-triviality [OF h gc gs])
  then show ?thesis
    using hn by blast
qed

lemma reduced-homology-group-punctured-plane-transfer:
  reduced-homology-group p (top-of-set punctured-plane) ≅
   reduced-homology-group p (top-of-set circle)
  using punctured-plane-homotopy-equivalent-circle
  by (rule homotopy-equivalent-space-imp-isomorphic-reduced-homology-groups)

lemma reduced-homology-group-punctured-plane-one:
  reduced-homology-group (int (1::nat)) (top-of-set punctured-plane) ≅ integer-group
proof −
  have h:
      reduced-homology-group (int (1::nat)) (top-of-set punctured-plane) ≅
       reduced-homology-group (int (1::nat)) (top-of-set circle)
    using punctured-plane-homotopy-equivalent-circle
    by (rule homotopy-equivalent-space-imp-isomorphic-reduced-homology-groups)
  then show ?thesis
    using reduced-homology-group-circle-one by (rule iso-trans)
qed

lemma reduced-homology-group-punctured-plane-other:
  p ≠ int (1::nat) ⟹
   trivial-group (reduced-homology-group p (top-of-set punctured-plane))
proof −
  assume p: p ≠ int (1::nat)
  have h:
      reduced-homology-group p (top-of-set punctured-plane) ≅
       reduced-homology-group p (top-of-set circle)
    by (rule reduced-homology-group-punctured-plane-transfer)
  have hc: trivial-group (reduced-homology-group p (top-of-set circle))
    by (rule reduced-homology-group-circle-other [OF p])
  have gp: group (reduced-homology-group p (top-of-set punctured-plane))
    by simp
  have gc: group (reduced-homology-group p (top-of-set circle))
    by simp
  have hi:
      trivial-group (reduced-homology-group p (top-of-set punctured-plane)) ⟷
       trivial-group (reduced-homology-group p (top-of-set circle))
```

by (*rule isomorphic-group-triviality* [*OF h gp gc*])
then show *?thesis*
using *hc* **by** *blast*
qed

lemma *circle-not-contractible*:
¬ *contractible-space* (*top-of-set circle*)
proof
assume *c*: *contractible-space* (*top-of-set circle*)
have *hc*:
contractible-space (*top-of-set circle*) ⟷
contractible-space (*nsphere* (*1*::*nat*))
by (*rule homeomorphic-space-contractibility* [*OF circle-homeomorphic-nsphere-one*])
have *hns*: *contractible-space* (*nsphere* (*1*::*nat*))
using *hc c* **by** *blast*
then show *False*
using *non-contractible-space-nsphere* [*of 1*::*nat*] **by** *blast*
qed

lemma *punctured-plane-not-contractible*:
¬ *contractible-space* (*top-of-set punctured-plane*)
proof
assume *c*: *contractible-space* (*top-of-set punctured-plane*)
have *hc*:
contractible-space (*top-of-set punctured-plane*) ⟷
contractible-space (*top-of-set circle*)
by (*rule homotopy-equivalent-space-contractibility*
[*OF punctured-plane-homotopy-equivalent-circle*])
have *hcircle*: *contractible-space* (*top-of-set circle*)
using *hc c* **by** *blast*
then show *False*
using *circle-not-contractible* **by** *blast*
qed

11 Brouwer degree

The homology library defines the degree of a self-map of an abstract sphere through its action on the top reduced homology group. We expose the basic calculational laws needed by later degree arguments.

lemma *Brouwer-degree-circle-identity*:
Brouwer-degree2 1 id = 1
by (*rule Brouwer-degree2-id*)

lemma *Brouwer-degree-circle-reflection*:
Brouwer-degree2 1 (λ*x i*. *if i = 0 then* −*x i else x i*) = −*1*
by (*rule Brouwer-degree2-reflection*)

lemma *Brouwer-degree-circle-compose*:
assumes *continuous-map* (*nsphere 1*) (*nsphere 1*) *f*
and *continuous-map* (*nsphere 1*) (*nsphere 1*) *g*
shows *Brouwer-degree2 1* (*g* ∘ *f*) =
Brouwer-degree2 1 g ∗ *Brouwer-degree2 1 f*
by (*rule Brouwer-degree2-compose* [*OF assms*])

lemma *Brouwer-degree-circle-homotopy-invariant*:
assumes *homotopic-with* (λ*x*. *True*) (*nsphere 1*) (*nsphere 1*) *f g*
shows *Brouwer-degree2 1 f = Brouwer-degree2 1 g*

by (*rule Brouwer-degree2-homotopic* [*OF assms*])

end
theory *Circle-Loop-Homology*
imports *Circle-Torus-Algebraic-Topology.Circle-Covering*
Circle-Torus-Algebraic-Topology.Circle-Homology-Degree
begin

12 Loops as singular cycles

definition *edge-coordinates* :: *real* ⇒ *nat* ⇒ *real* **where**
edge-coordinates t i = (*if i* = *0 then* 1−*t else if i* = *1 then t else 0*)

lemma *edge-coordinates-mem*:
t ∈ {*0..1*} ⟹ *edge-coordinates t* ∈ *standard-simplex 1*
by (*auto simp*: *edge-coordinates-def standard-simplex-def sum.atMost-Suc*)

lemma *edge-coordinates-continuous*:
continuous-map euclideanreal (*powertop-real UNIV*) *edge-coordinates*
unfolding *continuous-map-componentwise edge-coordinates-def*
by (*auto intro*!: *continuous-intros simp*: *if-distrib split*: *if-splits*)

definition *loop-simplex* :: (*real* ⇒ *complex*) ⇒ (*nat* ⇒ *real*) ⇒ *complex* **where**
loop-simplex p = *restrict* (λ*x*. *p* (*x 1*)) (*standard-simplex 1*)

lemma *loop-simplex-singular*:
assumes *path p path-image p* ⊆ *S*
shows *singular-simplex 1* (*top-of-set S*) (*loop-simplex p*)
proof −
have *coord*: *continuous-map* (*subtopology* (*powertop-real UNIV*) (*standard-simplex 1*))
(*top-of-set* {*0..1*}) (λ*x*. *x 1*)
by (*auto simp*: *continuous-map-in-subtopology standard-simplex-def*
intro: *continuous-map-from-subtopology continuous-map-product-projection*)
have *p*: *continuous-map* (*top-of-set* {*0..1*}) (*top-of-set S*) *p*
using *assms* **by** (*auto simp*: *continuous-map-subtopology-eu path-def path-image-def*)
show *?thesis*
unfolding *loop-simplex-def singular-simplex-def*
using *continuous-map-compose*[*OF coord p*]
by (*simp add*: *o-def*)
qed

lemma *loop-simplex-edge*:
t ∈ {*0..1*} ⟹ *loop-simplex p* (*edge-coordinates t*) = *p t*
using *edge-coordinates-mem*[*of t*]
by (*auto simp*: *loop-simplex-def edge-coordinates-def*)

lemma *loop-simplex-boundary*:
chain-boundary 1 (*frag-of* (*loop-simplex p*)) =
frag-of (*restrict* (λ*x*. *p 1*) (*standard-simplex 0*)) −
frag-of (*restrict* (λ*x*. *p 0*) (*standard-simplex 0*))
proof −
have *strip*: *singular-face 1 i* (*loop-simplex p*) = *singular-face 1 i* (λ*x*. *p* (*x 1*))
if *i* ≤ *1* **for** *i*
unfolding *loop-simplex-def* **by** (*rule singular-face-restrict*) (*use that* **in** *auto*)
have *faces*: *singular-face 1 0* (*loop-simplex p*) = *restrict* (λ*x*. *p 1*) (*standard-simplex 0*)
singular-face 1 1 (*loop-simplex p*) = *restrict* (λ*x*. *p 0*) (*standard-simplex 0*)
by (*simp-all only*: *strip*[*OF zero-le*] *strip*[*OF order-refl*])
(*auto simp*: *singular-face-def simplical-face-def standard-simplex-0 fun-eq-iff restrict-def*)

show *?thesis*
by (*simp add*: *chain-boundary-of sum.atMost-Suc faces*[*unfolded One-nat-def*])
qed

lemma *loop-simplex-cycle*:
assumes *is-loop p*
shows *singular-relcycle 1* (*top-of-set circle*) {} (*frag-of* (*loop-simplex p*))
using *loop-simplex-singular*[*OF is-loop-path*[*OF assms*] *is-loop-image*[*OF assms*]] *assms*
by (*simp add*: *singular-cycle singular-chain-of loop-simplex-boundary*[*unfolded One-nat-def*]
is-loop-def pathstart-def pathfinish-def)

definition *circle-loop-homology* **where**
circle-loop-homology p = *homologous-rel-set 1* (*top-of-set circle*) {} (*frag-of* (*loop-simplex p*))

lemma *circle-loop-homology-carrier*:
is-loop p ⟹ *circle-loop-homology p* ∈ *carrier* (*homology-group 1* (*top-of-set circle*))
using *loop-simplex-cycle*
by (*auto simp*: *circle-loop-homology-def carrier-relative-homology-group*[**where** *p=1*, *simplified*])

lemma *loop-simplex-compose*:
simplex-map 1 f (*loop-simplex p*) = *loop-simplex* (*f* ∘ *p*)
by (*auto simp*: *simplex-map-def loop-simplex-def fun-eq-iff*)

lemma *circle-loop-homology-natural*:
assumes *continuous-map* (*top-of-set circle*) (*top-of-set circle*) *f is-loop p*
shows *hom-induced 1* (*top-of-set circle*) {} (*top-of-set circle*) {} *f*
(*circle-loop-homology p*) = *circle-loop-homology* (*f* ∘ *p*)
unfolding *circle-loop-homology-def*
using *hom-induced-chain-map*[*OF assms*(*1*) - *loop-simplex-cycle*[*OF assms*(*2*)]]
by (*simp add*: *loop-simplex-compose*[*unfolded One-nat-def*])

definition *circle-loop-map* **where**
circle-loop-map p = *p* ∘ (λ*z*. *Arg2pi z* / (*2*∗*pi*))

lemma *circle-loop-map-homotopic*:
assumes *homotopic-loops circle p q*
shows *homotopic-with* (λ*h*. *True*) (*top-of-set circle*) (*top-of-set circle*)
(*circle-loop-map p*) (*circle-loop-map q*)
using *homotopic-loops-imp-homotopic-circlemaps*[*OF assms*]
by (*simp add*: *circle-loop-map-def circle-def*)

lemma *circle-loop-map-continuous*:
assumes *is-loop p*
shows *continuous-map* (*top-of-set circle*) (*top-of-set circle*) (*circle-loop-map p*)
proof −
have *homotopic-loops circle p p*
using *assms* **by** (*simp add*: *homotopic-loops-refl is-loop-def*)
then show *?thesis*
using *circle-loop-map-homotopic homotopic-with-imp-continuous-maps* **by** *blast*
qed

lemma *circle-loop-map-parametrize*:
assumes *is-loop p t* ∈ {*0..1*}
shows *circle-loop-map p* (*circlepath 0 1 t*) = *p t*
proof (*cases t* = *1*)
case *True*
have *endpoint*: *circlepath 0 1 1* = (*1*::*complex*)
using *is-loop-circlepath* **by** (*simp add*: *is-loop-def pathfinish-def*)

have *arg1*: *Arg2pi 1 = 0* **by** (*simp add*: *Arg2pi-eq-0*)
show *?thesis*
using *assms(1) True endpoint*
by (*simp add*: *circle-loop-map-def is-loop-def pathstart-def pathfinish-def arg1*)
next
case *False*
have *bounds*: *0 ≤ 2∗pi∗t 2∗pi∗t < 2∗pi*
using *assms(2) False* **by** (*auto simp*: *mult-nonneg-nonneg*)
have *arg*: *Arg2pi* (*circlepath 0 1 t*) = *2∗pi∗t*
using *bounds*
by (*simp add*: *circlepath-def part-circlepath-def linepath-def*
scaleR-conv-of-real Arg2pi-exp mult-ac)
show *?thesis* **by** (*simp add*: *circle-loop-map-def arg*)
qed

lemma *circle-loop-homology-map*:
assumes *is-loop p*
shows *hom-induced 1* (*top-of-set circle*) {} (*top-of-set circle*) {} (*circle-loop-map p*)
(*circle-loop-homology* (*circlepath 0 1*)) = *circle-loop-homology p*
proof −
have *eq*: *loop-simplex* (*circle-loop-map p* ∘ *circlepath 0 1*) = *loop-simplex p*
unfolding *loop-simplex-def*
using *circle-loop-map-parametrize*[*OF assms*]
by (*auto simp*: *fun-eq-iff standard-simplex-def*)
show *?thesis*
using *circle-loop-homology-natural*[*OF circle-loop-map-continuous*[*OF assms*] *is-loop-circlepath*]
by (*simp add*: *circle-loop-homology-def eq*)
qed

lemma *circle-loop-homology-homotopic*:
assumes *is-loop p is-loop q homotopic-paths circle p q*
shows *circle-loop-homology p = circle-loop-homology q*
proof −
have *hom*: *homotopic-loops circle p q*
using *assms* **by** (*intro homotopic-paths-imp-homotopic-loops*) (*auto simp*: *is-loop-def*)
have *eq*: *hom-induced 1* (*top-of-set circle*) {} (*top-of-set circle*) {} (*circle-loop-map p*) =
hom-induced 1 (*top-of-set circle*) {} (*top-of-set circle*) {} (*circle-loop-map q*)
using *circle-loop-map-homotopic*[*OF hom*]
by (*intro homology-homotopy-axiom*) *simp*
show *?thesis*
using *eq circle-loop-homology-map*[*OF assms(1)*] *circle-loop-homology-map*[*OF assms(2)*]
by *simp*
qed

end
theory *Circle-Loop-Concatenation*
imports *Circle-Loop-Homology*
begin

13 Concatenation and addition of cycles

lemma *homology-class-add*:
homologous-rel-set 1 X {} (*a*+*b*) =
homologous-rel-set 1 X {} *a* ⊗$_{homology\text{-}group\ 1\ X}$ *homologous-rel-set 1 X* {} *b*
proof −
have *split*: ∃ *u v*. *homologous-rel 1 X* {} *a u* ∧ *homologous-rel 1 X* {} *b v* ∧ *x*=*u*+*v*
if *homologous-rel 1 X* {} (*a*+*b*) *x* **for** *x*
using *that* **by** (*metis diff-add-cancel diff-diff-eq2 homologous-rel-def homologous-rel-refl*)

```
  show ?thesis
    using split by (auto simp: mult-relative-homology-group[where p=1, simplified]
        intro: homologous-rel-add)
qed

definition join-simplex where
  join-simplex p q = restrict (λx. (p +++ q) (x 1 / 2 + x 2)) (standard-simplex 2)

lemma join-simplex-singular:
  assumes is-loop p is-loop q
  shows singular-simplex 2 (top-of-set circle) (join-simplex p q)
proof −
  have coord: continuous-map (subtopology (powertop-real UNIV) (standard-simplex 2))
      (top-of-set {0..1}) (λx. x 1 / 2 + x 2)
  proof (subst continuous-map-in-subtopology, intro conjI)
    show continuous-map (subtopology (powertop-real UNIV) (standard-simplex 2)) euclideanreal
        (λx. x 1 / 2 + x 2)
      by (intro continuous-intros continuous-map-from-subtopology continuous-map-product-projection)
auto
    have bounds: (λx. x 1 / 2 + x 2) ‘ topspace (subtopology (powertop-real UNIV) (standard-simplex
2)) ⊆ {0..1}
    proof (rule image-subsetI)
      fix x assume x ∈ topspace (subtopology (powertop-real UNIV) (standard-simplex 2))
      then have x: 0 ≤ x 0 0 ≤ x 1 0 ≤ x 2 x 0 + x 1 + x 2 = 1
        by (auto simp: standard-simplex-def sum.atMost-Suc numeral-2-eq-2)
      show x 1 / 2 + x 2 ∈ {0..1} using x by auto
    qed
    show (λx. x 1 / 2 + x 2) ∈ topspace (subtopology (powertop-real UNIV) (standard-simplex 2)) →
{0..1}
      using bounds by auto
  qed
  have p: continuous-map (top-of-set {0..1}) (top-of-set circle) (p +++ q)
    using is-loop-join[OF assms]
    by (auto simp: continuous-map-subtopology-eu is-loop-def path-def path-image-def)
  show ?thesis unfolding join-simplex-def singular-simplex-def
    using continuous-map-compose[OF coord p] by (simp add: o-def)
qed

lemma join-simplex-faces:
  assumes is-loop p is-loop q
  shows singular-face 2 0 (join-simplex p q) = loop-simplex q
    singular-face 2 1 (join-simplex p q) = loop-simplex (p +++ q)
    singular-face 2 2 (join-simplex p q) = loop-simplex p
proof −
  have strip: singular-face 2 i (join-simplex p q) =
      singular-face 2 i (λx. (p +++ q) (x 1 / 2 + x 2))
    if i ≤ 2 for i
    unfolding join-simplex-def by (rule singular-face-restrict) (use that in auto)
  have ends: p 1 = q 0 using assms by (simp add: is-loop-def pathstart-def pathfinish-def)
  have glue: (p +++ q) (s/2+t) = q t if s+t=1 0 ≤ t for s t
  proof (cases t=0)
    case True then show ?thesis using that ends by (simp add: joinpaths-def)
  next
    case False
    have gt: s/2+t > 1/2 using that False by linarith
    have eq: s + 2∗t − 1 = t using that by linarith
    show ?thesis using gt by (simp add: joinpaths-def eq)
  qed
```

show *singular-face 2 0* (*join-simplex p q*) = *loop-simplex q*
apply (*subst strip*)
apply *simp*
apply (*simp only*: *singular-face-def loop-simplex-def*)
using *ends*
by (*auto simp*: *fun-eq-iff simplical-face-def standard-simplex-def sum.atMost-Suc numeral-2-eq-2 glue*)
show *singular-face 2 1* (*join-simplex p q*) = *loop-simplex* (*p +++ q*)
apply (*subst strip*)
apply *simp*
apply (*simp only*: *singular-face-def loop-simplex-def*)
by (*auto simp*: *simplical-face-def fun-eq-iff*)
show *singular-face 2 2* (*join-simplex p q*) = *loop-simplex p*
apply (*subst strip*)
apply *simp*
apply (*simp only*: *singular-face-def loop-simplex-def*)
by (*auto simp*: *fun-eq-iff simplical-face-def standard-simplex-def joinpaths-def numeral-2-eq-2*)
qed

lemma *circle-loop-homology-join*:
assumes *is-loop p is-loop q*
shows *circle-loop-homology* (*p +++ q*) =
circle-loop-homology p $\otimes_{\textit{homology-group 1 (top-of-set circle)}}$ *circle-loop-homology q*
proof −
have *boundary*: *chain-boundary 2* (*frag-of* (*join-simplex p q*)) =
frag-of (*loop-simplex q*) − *frag-of* (*loop-simplex* (*p +++ q*)) + *frag-of* (*loop-simplex p*)
using *join-simplex-faces*[*OF assms*]
by (*simp add*: *chain-boundary-of sum.atMost-Suc numeral-2-eq-2*)
have *homologous-rel 1* (*top-of-set circle*) {} (*frag-of* (*loop-simplex* (*p +++ q*)))
(*frag-of* (*loop-simplex p*) + *frag-of* (*loop-simplex q*))
unfolding *homologous-rel-def singular-boundary*
by (*rule exI*[**where** *x*=−*frag-of* (*join-simplex p q*)])
(*simp add*: *singular-chain-minus singular-chain-of*
join-simplex-singular[*OF assms*, *unfolded numeral-2-eq-2*]
chain-boundary-minus boundary[*unfolded numeral-2-eq-2*] *algebra-simps*)
then show *?thesis*
by (*simp add*: *circle-loop-homology-def*
homology-class-add[*unfolded One-nat-def*, *symmetric*] *homologous-rel-set-eq*)
qed

lemma *circle-loop-homology-constant*:
circle-loop-homology (*linepath 1 1*) = $\mathbf{1}_{\textit{homology-group 1 (top-of-set circle)}}$
proof −
interpret *H*: *group homology-group 1* (*top-of-set circle*) **by** *simp*
have *eq*: (*linepath* (*1*::*complex*) *1 +++ linepath 1 1*) = *linepath 1 1*
by (*auto simp*: *fun-eq-iff joinpaths-def linepath-refl*)
show *?thesis*
using *circle-loop-homology-join*[*OF is-loop-linepath-1 is-loop-linepath-1*]
by (*simp add*: *eq H.l-cancel-one' circle-loop-homology-carrier*)
qed

lemma *circle-loop-homology-reverse*:
assumes *is-loop p*
shows *circle-loop-homology* (*reversepath p*) =
$\textit{inv}_{\textit{homology-group 1 (top-of-set circle)}}$ (*circle-loop-homology p*)
proof −
interpret *H*: *group homology-group 1* (*top-of-set circle*) **by** *simp*
have *hom*: *homotopic-paths circle* (*reversepath p +++ p*) (*linepath 1 1*)

using *homotopic-paths-linv*[*OF is-loop-path*[*OF assms*] *is-loop-image*[*OF assms*]] *assms*
by (*simp add*: *is-loop-def*)
have *eq*: *circle-loop-homology* (*reversepath p*) $\otimes_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$
circle-loop-homology p = $\mathbf{1}_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$
using *circle-loop-homology-homotopic*[*OF is-loop-join*[*OF is-loop-reversepath*[*OF assms*] *assms*]
is-loop-linepath-1 hom]
by (*simp add*: *circle-loop-homology-join is-loop-reversepath assms circle-loop-homology-constant*)
show *?thesis*
by (*rule sym*, *rule H.inv-equality*) (*use eq assms* **in** ‹*auto intro*: *circle-loop-homology-carrier*›)
qed

lemma *circle-loop-homology-cpow*:
circle-loop-homology (*Fundamental-Group-Circle.cpow n*) =
circle-loop-homology (*circlepath 0 1*) $[\widehat{\ }]_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$ *n*
proof (*induction n*)
case *0* **then show** *?case* **by** (*simp add*: *circle-loop-homology-constant*[*simplified linepath-refl*])
next
case (*Suc n*)
have *step*: *circle-loop-homology* (*circlepath 0 1*) $[\widehat{\ }]_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$ *Suc n* =
circle-loop-homology (*circlepath 0 1*) $\otimes_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$
(*circle-loop-homology* (*circlepath 0 1*) $[\widehat{\ }]_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$ *n*)
by (*rule monoid.nat-pow-Suc2*)
(*simp-all add*: *group.is-monoid circle-loop-homology-carrier is-loop-circlepath*)
then show *?case*
using *Suc*
by (*simp only*: *Fundamental-Group-Circle.cpow.simps*
circle-loop-homology-join[*OF is-loop-circlepath is-loop-cpow*] *step*)
qed

theorem *circle-loop-homology-winding*:
assumes *is-loop p*
shows *circle-loop-homology p* =
circle-loop-homology (*circlepath 0 1*) $[\widehat{\ }]_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$ *wnd p*
proof (*cases 0* $\leq$ *wnd p*)
case *True*
have *eq*: *circle-lift-endpoint p* = *circle-lift-endpoint* (*Fundamental-Group-Circle.cpow* (*nat* (*wnd p*)))
by (*simp add*: *circle-lift-endpoint-eq-wnd*[*OF assms*] *circle-lift-endpoint-cpow True*)
have *hom*: *homotopic-paths circle p* (*Fundamental-Group-Circle.cpow* (*nat* (*wnd p*)))
by (*rule circle-lift-endpoint-eq-homotopic*[*OF assms is-loop-cpow eq*])
show *?thesis*
using *circle-loop-homology-homotopic*[*OF assms is-loop-cpow hom*]
by (*simp add*: *circle-loop-homology-cpow int-pow-int*[*symmetric*] *True*)
next
case *False*
have *nonpos*: *wnd p* $\leq$ *0* **using** *False* **by** *linarith*
have *eq*: *circle-lift-endpoint p* = *circle-lift-endpoint* (*reversepath* (*Fundamental-Group-Circle.cpow* (*nat* (−*wnd p*))))
by (*simp add*: *circle-lift-endpoint-eq-wnd*[*OF assms*] *circle-lift-endpoint-reverse is-loop-cpow*
circle-lift-endpoint-cpow nonpos)
have *hom*: *homotopic-paths circle p* (*reversepath* (*Fundamental-Group-Circle.cpow* (*nat* (−*wnd p*))))
by (*rule circle-lift-endpoint-eq-homotopic*[*OF assms is-loop-reversepath*[*OF is-loop-cpow*] *eq*])
have *class-eq*: *circle-loop-homology p* =
inv$_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$
(*circle-loop-homology* (*circlepath 0 1*) $[\widehat{\ }]_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$ *nat* (−*wnd p*))
using *circle-loop-homology-homotopic*[*OF assms is-loop-reversepath*[*OF is-loop-cpow*] *hom*]
by (*simp only*: *circle-loop-homology-reverse*[*OF is-loop-cpow*] *circle-loop-homology-cpow*)
have *neg*: *wnd p* < *0* **using** *False* **by** *linarith*

have *power*: *circle-loop-homology* (*circlepath 0 1*) [^]$_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$ *wnd p* =
inv$_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$
(*circle-loop-homology* (*circlepath 0 1*) [^]$_{homology\text{-}group\ 1\ (top\text{-}of\text{-}set\ circle)}$ *nat* (−*wnd p*))
by (*rule trans*[*OF int-pow-def2*]) (*simp only*: *if-P*[*OF neg*])
show *?thesis* **using** *class-eq power* **by** *simp*
qed

end
theory *Circle-Winding-Cocycle*
imports *Circle-Loop-Homology*
begin

14 Winding number annihilates singular boundaries

The winding functional is extended linearly to singular chains. The boundary of each singular triangle has winding zero, because its boundary loop contracts in the triangle. This detects a nonzero first homology class without assuming a Hurewicz theorem.

definition *winding-triangle* :: *complex set* **where**
winding-triangle = {*z*. *0* ≤ *Re z* ∧ *0* ≤ *Im z* ∧ *Re z* + *Im z* ≤ *1*}

lemma *convex-winding-triangle*: *convex winding-triangle*
proof −
have *eq*: *winding-triangle* = {*z*. *0* ≤ *Re z*} ∩ {*z*. *0* ≤ *Im z*} ∩
{*z*. *inner* (*1*+i) *z* ≤ *1*}
by (*auto simp*: *winding-triangle-def inner-complex-def*)
show *?thesis* **unfolding** *eq*
by (*intro convex-Int convex-halfspace-Re-ge convex-halfspace-Im-ge convex-halfspace-le*)
qed

definition *triangle-coordinates* :: *complex* ⇒ *nat* ⇒ *real* **where**
triangle-coordinates z i =
(*if i=0 then 1* − *Re z* − *Im z else if i=1 then Re z else if i=2 then Im z else 0*)

lemma *triangle-coordinates-mem*:
assumes *z* ∈ *winding-triangle*
shows *triangle-coordinates z* ∈ *standard-simplex 2*
using *assms*
by (*auto simp*: *winding-triangle-def triangle-coordinates-def standard-simplex-def*
sum.atMost-Suc numeral-2-eq-2 split: *if-splits*; *linarith*)

lemma *triangle-coordinates-continuous*:
continuous-map euclidean (*powertop-real UNIV*) *triangle-coordinates*
unfolding *continuous-map-componentwise triangle-coordinates-def*
by (*auto intro*!: *continuous-intros simp*: *if-distrib split*: *if-splits*)

lemma *singular-triangle-continuous*:
assumes *singular-simplex 2* (*top-of-set circle*) *f*
shows *continuous-on winding-triangle* (*f* ∘ *triangle-coordinates*)
(*f* ∘ *triangle-coordinates*) ‘ *winding-triangle* ⊆ *circle*
proof −
have *c*: *continuous-map* (*top-of-set winding-triangle*)
(*subtopology* (*powertop-real UNIV*) (*standard-simplex 2*)) *triangle-coordinates*
using *triangle-coordinates-continuous triangle-coordinates-mem*
by (*auto simp*: *continuous-map-in-subtopology intro*: *continuous-map-from-subtopology*)
have *continuous-map* (*top-of-set winding-triangle*) (*top-of-set circle*) (*f* ∘ *triangle-coordinates*)
using *assms c* **unfolding** *singular-simplex-def* **by** (*blast intro*: *continuous-map-compose*)
then show *continuous-on winding-triangle* (*f* ∘ *triangle-coordinates*)

```
    (f ∘ triangle-coordinates) ‘ winding-triangle ⊆ circle
    by (auto simp: continuous-map-subtopology-eu)
qed

lemma triangle-winding-add:
  assumes cont: continuous-on winding-triangle f and im: f ‘ winding-triangle ⊆ circle
  shows winding-number (f ∘ linepath 1 i) 0 −
      winding-number (f ∘ linepath 0 i) 0 +
      winding-number (f ∘ linepath 0 1) 0 = 0
proof −
  have verts: (0::complex) ∈ winding-triangle 1 ∈ winding-triangle i ∈ winding-triangle
    by (simp-all add: winding-triangle-def)
  have edges: path-image (linepath 0 1) ⊆ winding-triangle
    path-image (linepath 1 i) ⊆ winding-triangle
    path-image (linepath 0 i) ⊆ winding-triangle
    using convex-winding-triangle verts
    by (simp-all add: convex-contains-segment)
  have hom: homotopic-paths winding-triangle (linepath 0 1 +++ linepath 1 i) (linepath 0 i)
  proof (rule homotopic-paths-linear)
    show path (linepath 0 1 +++ linepath 1 i) path (linepath 0 i) by auto
    show pathstart (linepath 0 i) = pathstart (linepath 0 1 +++ linepath 1 i)
      pathfinish (linepath 0 i) = pathfinish (linepath 0 1 +++ linepath 1 i) by simp-all
    show closed-segment ((linepath 0 1 +++ linepath 1 i) t) (linepath 0 i t) ⊆ winding-triangle
      if t ∈ {0..1} for t
    proof −
      have jp: path-image (linepath 0 1 +++ linepath 1 i) ⊆ winding-triangle
        by (rule subset-path-image-join) (use edges in auto)
      have mem: (linepath 0 1 +++ linepath 1 i) t ∈ winding-triangle
        linepath 0 i t ∈ winding-triangle
        using jp edges(3) that by (auto simp: path-image-def)
      show ?thesis using convex-winding-triangle mem
        by (meson convex-contains-segment)
    qed
  qed
  have mapped: homotopic-paths (−{0}) (f ∘ (linepath 0 1 +++ linepath 1 i)) (f ∘ linepath 0 i)
    by (rule homotopic-paths-continuous-image[OF hom cont])
      (use im in ‹auto simp: circle-def›)
  have p: path (f ∘ linepath 0 1) path (f ∘ linepath 1 i)
    using cont edges by (intro path-continuous-image; auto intro: continuous-on-subset)+
  have images: path-image (f ∘ linepath 0 1) ⊆ circle
    path-image (f ∘ linepath 1 i) ⊆ circle
    using subset-trans[OF image-mono[OF edges(1), of f] im]
      subset-trans[OF image-mono[OF edges(2), of f] im]
    by (simp-all add: path-image-def image-comp)
  have nz: 0 ∉ path-image (f ∘ linepath 0 1) 0 ∉ path-image (f ∘ linepath 1 i)
    using images by (auto simp: circle-def sphere-def)
  have winding-number (f ∘ linepath 0 i) 0 =
      winding-number (f ∘ linepath 0 1) 0 + winding-number (f ∘ linepath 1 i) 0
    using winding-number-homotopic-paths[OF mapped]
      winding-number-join[OF p(1) nz(1) p(2) nz(2)]
    by (simp add: path-compose-join pathstart-def pathfinish-def linepath-def)
  then show ?thesis by simp
qed

lemma singular-triangle-winding:
  assumes singular-simplex 2 (top-of-set circle) f
  shows winding-number (singular-face 2 0 f ∘ edge-coordinates) 0 −
      winding-number (singular-face 2 1 f ∘ edge-coordinates) 0 +
```

```
    winding-number (singular-face 2 2 f ∘ edge-coordinates) 0 = 0
proof −
  have coords:
    simplical-face 0 (edge-coordinates t) = triangle-coordinates (linepath 1 i t)
    simplical-face 1 (edge-coordinates t) = triangle-coordinates (linepath 0 i t)
    simplical-face 2 (edge-coordinates t) = triangle-coordinates (linepath 0 1 t) for t
    by (auto simp: fun-eq-iff simplical-face-def edge-coordinates-def triangle-coordinates-def
        linepath-def scaleR-conv-of-real numeral-2-eq-2 split: if-splits; algebra)+
  have faces:
    (singular-face 2 0 f ∘ edge-coordinates) t =
      ((f ∘ triangle-coordinates) ∘ linepath 1 i) t
    (singular-face 2 1 f ∘ edge-coordinates) t =
      ((f ∘ triangle-coordinates) ∘ linepath 0 i) t
    (singular-face 2 2 f ∘ edge-coordinates) t =
      ((f ∘ triangle-coordinates) ∘ linepath 0 1) t
    if t ∈ {0..1} for t
    using edge-coordinates-mem[OF that]
    by (simp-all add: singular-face-def coords[unfolded One-nat-def, simplified])
  have eq:
    winding-number (singular-face 2 0 f ∘ edge-coordinates) 0 =
      winding-number ((f ∘ triangle-coordinates) ∘ linepath 1 i) 0
    winding-number (singular-face 2 1 f ∘ edge-coordinates) 0 =
      winding-number ((f ∘ triangle-coordinates) ∘ linepath 0 i) 0
    winding-number (singular-face 2 2 f ∘ edge-coordinates) 0 =
      winding-number ((f ∘ triangle-coordinates) ∘ linepath 0 1) 0
    by (intro winding-number-cong; use faces in auto)+
  show ?thesis
    unfolding eq by (rule triangle-winding-add[OF singular-triangle-continuous[OF assms]])
qed

definition complex-add-group :: complex monoid where
  complex-add-group = ⦇carrier = UNIV, mult = (+), one = 0⦈

lemma comm-group-complex-add: comm-group complex-add-group
proof (rule comm-groupI)
  show ∃ y∈carrier complex-add-group. y ⊗complex-add-group x = 1complex-add-group
    for x by (rule bexI[where x=−x]) (simp-all add: complex-add-group-def)
qed (simp-all add: complex-add-group-def add.assoc add.commute)

lemma winding-chain-functional:
  obtains w :: complex chain ⇒ complex where
    w 0 = 0 ⋀a b. w (a+b) = w a + w b ⋀a b. w (a−b) = w a − w b
    ⋀f. w (frag-of f) = winding-number (f ∘ edge-coordinates) 0
    ⋀c. singular-chain 2 (top-of-set circle) c ⟹ w (chain-boundary 2 c) = 0
proof −
  interpret C: comm-group complex-add-group by (rule comm-group-complex-add)
  obtain w where hom: w ∈ hom (free-Abelian-group UNIV) complex-add-group
    and gen: ⋀f. w (frag-of f) = winding-number (f ∘ edge-coordinates) 0
    by (rule C.free-Abelian-group-universal[where S=UNIV and
          f=λf. winding-number (f ∘ edge-coordinates) 0])
       (auto simp: complex-add-group-def)
  interpret W: group-hom free-Abelian-group UNIV complex-add-group w
    by (simp add: group-hom-def group-hom-axioms-def hom C.is-group)
  have zero: w 0 = 0 using W.hom-one by (simp add: complex-add-group-def)
  have add: w (a+b) = w a + w b for a b
    using W.hom-mult[of a b] by (simp add: complex-add-group-def)
  have minus: w (−a) = − w a for a
    using add[of a −a] zero by (simp add: add-eq-0-iff)
```

```
  have diff: w (a−b) = w a − w b for a b
    by (subst diff-conv-add-uminus, subst add, subst minus)
      (rule diff-conv-add-uminus[symmetric])
  have boundary: w (chain-boundary 2 c) = 0 if singular-chain 2 (top-of-set circle) c for c
  proof (rule free-Abelian-group-induct[OF that[unfolded singular-chain-def]])
    show w (chain-boundary 2 0) = 0 by (simp add: zero)
    show w (chain-boundary 2 (a−b)) = 0
      if Poly-Mapping.keys a ⊆ singular-simplex-set 2 (top-of-set circle)
        Poly-Mapping.keys b ⊆ singular-simplex-set 2 (top-of-set circle)
        w (chain-boundary 2 a) = 0 w (chain-boundary 2 b) = 0 for a b
      using that by (simp add: chain-boundary-diff diff)
    show w (chain-boundary 2 (frag-of f)) = 0
      if f ∈ singular-simplex-set 2 (top-of-set circle) for f
      using that singular-triangle-winding[of f]
      by (simp add: chain-boundary-of sum.atMost-Suc numeral-2-eq-2
          add minus diff gen zero)
  qed
  show thesis by (rule that[OF zero add diff]) (use gen boundary in auto)
qed

theorem circle-standard-loop-homology-nontrivial:
  circle-loop-homology (circlepath 0 1) ≠ 1_{homology-group 1 (top-of-set circle)}
proof
  assume eq: circle-loop-homology (circlepath 0 1) = 1_{homology-group 1 (top-of-set circle)}
  have singular-relboundary 1 (top-of-set circle) {} (frag-of (loop-simplex (circlepath 0 1)))
    using eq by (simp add: circle-loop-homology-def homologous-rel-set-eq-relboundary
        one-relative-homology-group[where p=1, simplified])
  then obtain c where c: singular-chain 2 (top-of-set circle) c
    chain-boundary 2 c = frag-of (loop-simplex (circlepath 0 1))
    by (auto simp: singular-boundary numeral-2-eq-2)
  obtain w :: complex chain ⇒ complex where
    w: ⋀f. w (frag-of f) = winding-number (f ∘ edge-coordinates) 0
    ⋀c. singular-chain 2 (top-of-set circle) c ⟹ w (chain-boundary 2 c) = 0
    by (rule winding-chain-functional) blast
  have winding-number (loop-simplex (circlepath 0 1) ∘ edge-coordinates) 0 = 0
    using w c by metis
  moreover have winding-number (loop-simplex (circlepath 0 1) ∘ edge-coordinates) 0 = 1
    using winding-number-cong[of loop-simplex (circlepath 0 1) ∘ edge-coordinates circlepath 0 1 0]
    by (simp add: loop-simplex-edge winding-number-circlepath-centre)
  ultimately show False by simp
qed

end
theory Circle-Homology-Action
  imports Circle-Winding-Cocycle Circle-Loop-Concatenation
begin
```

15 Winding number and the induced homology map

An endomorphism of an infinite cyclic group is determined by its action on any nonzero element. Thus the nonzero standard-loop class and the loop winding formula suffice; we need not first prove that this class is a primitive generator. The final theorem applies to arbitrary continuous circle maps, using a rotation homotopy to normalize the basepoint.

```
lemma infinite-cyclic-hom-determined:
  assumes grp: group G and iso: h ∈ iso G Elementary-Groups.integer-group
    and hom: F ∈ hom G G
```

```
    and a: a ∈ carrier G a ≠ 𝟭_G
    and image: F a = a [^]_G (n::int)
    and x: x ∈ carrier G
  shows h (F x) = n * h x
proof −
  interpret G: group G by (rule grp)
  interpret H: group-hom G Elementary-Groups.integer-group h
    using iso by (simp add: group-hom-def group-hom-axioms-def grp iso-def)
  interpret F: group-hom G G F
    by (simp add: group-hom-def group-hom-axioms-def grp hom)
  obtain k where inv: group-isomorphisms G Elementary-Groups.integer-group h k
    using iso G.iso-iff-group-isomorphisms by blast
  interpret K: group-hom Elementary-Groups.integer-group G k
    using inv by (auto simp: group-hom-def group-hom-axioms-def group-isomorphisms-def grp)
  have k1: k 1 ∈ carrier G using K.hom-closed[of 1] by simp
  have repr: y = k 1 [^]_G h y if y ∈ carrier G for y
  proof −
    have k (h y) = y using inv that by (simp add: group-isomorphisms-def)
    moreover have k (h y) = k 1 [^]_G h y
      using K.hom-int-pow[of 1 h y] by simp
    ultimately show ?thesis by simp
  qed
  have action: h (F y) = h (F (k 1)) * h y if y ∈ carrier G for y
    by (subst repr[OF that]) (simp add: F.hom-int-pow H.hom-int-pow k1 F.hom-closed)
  have inj: inj-on h (carrier G)
    using iso by (simp add: iso-def bij-betw-def)
  then have ha: h a ≠ 0
    using H.inj-on-one-iff a by force
  have h (F (k 1)) * h a = n * h a
    using action[OF a(1)] image H.hom-int-pow[OF a(1), of n]
    by (auto simp: mult.commute)
  then have h (F (k 1)) = n using ha by simp
  then show ?thesis using action[OF x] by simp
qed

lemma circle-first-homology-iso:
  homology-group 1 (top-of-set circle) ≅ Elementary-Groups.integer-group
  using reduced-homology-group-circle-one
  by (simp add: un-reduced-homology-group)

lemma circle-based-map-loop:
  assumes continuous-map (top-of-set circle) (top-of-set circle) f f 1 = 1 is-loop p
  shows is-loop (f ∘ p)
proof −
  have c: continuous-on circle f using assms(1) by (simp add: continuous-map-subtopology-eu)
  have path: path (f ∘ p)
    by (rule path-continuous-image[OF is-loop-path[OF assms(3)]]
      (rule continuous-on-subset[OF c is-loop-image[OF assms(3)]]))
  show ?thesis using assms path
    by (auto simp: continuous-map-subtopology-eu is-loop-def pathstart-def pathfinish-def path-image-def)
qed

theorem circle-homology-action-winding-based:
  assumes coordinates: h ∈ iso (homology-group 1 (top-of-set circle)) Elementary-Groups.integer-group
    and f: continuous-map (top-of-set circle) (top-of-set circle) f f 1 = 1
    and x: x ∈ carrier (homology-group 1 (top-of-set circle))
  shows h (hom-induced 1 (top-of-set circle) {} (top-of-set circle) {} f x) =
    wnd (f ∘ circlepath 0 1) * h x
```

proof (*rule infinite-cyclic-hom-determined*[*OF - coordinates - - circle-standard-loop-homology-nontrivial - x*])
show *group* (*homology-group 1* (*top-of-set circle*)) **by** *simp*
show *hom-induced 1* (*top-of-set circle*) {} (*top-of-set circle*) {} *f* ∈
hom (*homology-group 1* (*top-of-set circle*)) (*homology-group 1* (*top-of-set circle*))
by (*rule hom-induced-empty-hom*)
show *circle-loop-homology* (*circlepath 0 1*) ∈ *carrier* (*homology-group 1* (*top-of-set circle*))
by (*rule circle-loop-homology-carrier*[*OF is-loop-circlepath*])
show *hom-induced 1* (*top-of-set circle*) {} (*top-of-set circle*) {} *f*
(*circle-loop-homology* (*circlepath 0 1*)) =
circle-loop-homology (*circlepath 0 1*) [⁀]$_{\textit{homology-group 1 (top-of-set circle)}}$
wnd (*f* ∘ *circlepath 0 1*)
using *circle-loop-homology-natural*[*OF f*(*1*) *is-loop-circlepath*]
circle-loop-homology-winding[*OF circle-based-map-loop*[*OF f is-loop-circlepath*]] **by** *simp*
qed

lemma *circle-map-winding-homotopic*:
assumes *homotopic-with-canon* (λ*h*. *True*) *circle circle f g*
shows *wnd* (*f* ∘ *circlepath 0 1*) = *wnd* (*g* ∘ *circlepath 0 1*)
proof −
have *param*: *circlepath 0 1* = *exp* ∘ (λ*t*. *2* ∗ *of-real pi* ∗ *of-real t* ∗ i)
by (*auto simp*: *fun-eq-iff circlepath-def part-circlepath-def linepath-def scaleR-conv-of-real algebra-simps*)
have *hom*: *homotopic-loops circle* (*f* ∘ *circlepath 0 1*) (*g* ∘ *circlepath 0 1*)
using *homotopic-circlemaps-imp-homotopic-loops*[**where** *f*=*f* **and** *g*=*g* **and** *S*=*circle*] *assms*
by (*simp add*: *circle-def param o-assoc*)
have *homotopic-loops* (−{*0*}) (*f* ∘ *circlepath 0 1*) (*g* ∘ *circlepath 0 1*)
by (*rule homotopic-loops-subset*[*OF hom*]) (*auto simp*: *circle-def sphere-def*)
then have *winding-number* (*f* ∘ *circlepath 0 1*) *0* = *winding-number* (*g* ∘ *circlepath 0 1*) *0*
by (*rule winding-number-homotopic-loops*)
then show *?thesis* **by** (*simp add*: *wnd-def*)
qed

lemma *circle-map-based-normalization*:
assumes *f*: *continuous-map* (*top-of-set circle*) (*top-of-set circle*) *f*
obtains *g* **where** *continuous-map* (*top-of-set circle*) (*top-of-set circle*) *g g 1* = *1*
homotopic-with-canon (λ*h*. *True*) *circle circle f g*
proof −
have *fc*: *continuous-on circle f* **and** *fim*: *f* ∈ *circle* → *circle*
using *f* **by** *auto*
have *one*: (*1*::*complex*) ∈ *circle* **by** (*simp add*: *circle-def*)
have *f1*: *norm* (*f 1*) = *1* **using** *fim one norm-of-circle* **by** *blast*
have *nz*: *f 1* ≠ *0* **using** *f1* **by** *auto*
have *inv*: *inverse* (*f 1*) ∈ *circle* **using** *f1* **by** (*simp add*: *circle-def sphere-def norm-inverse*)
have *pc*: *path-connected circle*
unfolding *circle-def* **by** (*rule path-connected-sphere*) *simp*
obtain *r* **where** *r*: *path r path-image r* ⊆ *circle*
pathstart r = *1 pathfinish r* = *inverse* (*f 1*)
using *pc one inv* **unfolding** *path-connected-def* **by** *blast*
define *g* **where** *g z* = *inverse* (*f 1*) ∗ *f z* **for** *z*
define *H* **where** *H x* = *r* (*fst x*) ∗ *f* (*snd x*) **for** *x*
have *rc*: *continuous-on* {*0..1*} *r* **using** *r*(*1*) **by** (*simp only*: *path-def*)
have *rcomp*: *continuous-on* ({*0..1*} × *circle*) (λ*x*. *r* (*fst x*))
by (*rule continuous-on-compose2*[*OF rc*]) (*auto intro*: *continuous-on-fst*)
have *fcomp*: *continuous-on* ({*0..1*} × *circle*) (λ*x*. *f* (*snd x*))
by (*rule continuous-on-compose2*[*OF fc*]) (*auto intro*: *continuous-on-snd*)
have *hc*: *continuous-on* ({*0..1*} × *circle*) *H*
unfolding *H-def* **by** (*rule continuous-on-mult*[*OF rcomp fcomp*])

```
  have rn: norm (r t) = 1 if t ∈ {0..1} for t
    using r(2) that by (auto simp: path-image-def circle-def sphere-def subset-iff)
  have fn: norm (f z) = 1 if z ∈ circle for z
    using fim that norm-of-circle by blast
  have hi: H ∈ ({0..1} × circle) → circle
  proof (rule Pi-I)
    fix x :: real × complex assume x: x ∈ {0..1} × circle
    have xt: fst x ∈ {0..1} and xz: snd x ∈ circle using x by auto
    have norm (H x) = 1
      unfolding H-def by (simp add: norm-mult rn[OF xt] fn[OF xz])
    then show H x ∈ circle by (simp add: circle-def sphere-def)
  qed
  have cm: continuous-map (prod-topology (top-of-set {0..1}) (top-of-set circle)) (top-of-set circle) H
    using hc hi by (simp add: continuous-map-in-subtopology prod-topology-subtopology-eu)
  have hom: homotopic-with-canon (λh. True) circle circle f g
    unfolding homotopic-with-def
    by (rule exI[where x=H])
       (use cm r in ‹auto simp: H-def g-def pathstart-def pathfinish-def›)
  have gc: continuous-map (top-of-set circle) (top-of-set circle) g
    using hom homotopic-with-imp-continuous-maps by blast
  show thesis by (rule that[OF gc - hom]) (simp add: g-def nz)
qed

theorem circle-homology-action-winding:
  assumes coordinates: h ∈ iso (homology-group 1 (top-of-set circle)) Elementary-Groups.integer-group
    and f: continuous-map (top-of-set circle) (top-of-set circle) f
    and x: x ∈ carrier (homology-group 1 (top-of-set circle))
  shows h (hom-induced 1 (top-of-set circle) {} (top-of-set circle) {} f x) =
    wnd (f ∘ circlepath 0 1) ∗ h x
proof −
  obtain g where g: continuous-map (top-of-set circle) (top-of-set circle) g g 1 = 1
    and hom: homotopic-with-canon (λh. True) circle circle f g
    by (rule circle-map-based-normalization[OF f])
  have eq: hom-induced 1 (top-of-set circle) {} (top-of-set circle) {} f =
      hom-induced 1 (top-of-set circle) {} (top-of-set circle) {} g
    by (rule homology-homotopy-axiom) (use hom in simp)
  show ?thesis
    using circle-homology-action-winding-based[OF coordinates g x]
      circle-map-winding-homotopic[OF hom] eq by simp
qed

end
theory Circle-Torus-Algebraic-Topology
  imports Torus-Fundamental-Group Circle-SVK Circle-Homology-Action
begin
```

16 Circle, torus, and punctured-plane invariants: collected results

This theory collects the circle, torus, and punctured-plane results. Its component theories separate the reusable covering-space API, product and retract constructions, the Seifert–van Kampen interface, and the homology/degree consequences.

```
theorem circle-fundamental-group-summary:
  pi1-circle ≅ Fundamental-Group-Circle.integer-group
  by (rule pi1-circle-iso-integers)
```

theorem *circle-reduced-homology-summary*:
reduced-homology-group 1 (*top-of-set circle*) ≅
Elementary-Groups.integer-group
using *reduced-homology-group-circle-one* **by** *auto*

theorem *punctured-plane-homology-and-noncontractibility*:
reduced-homology-group 1 (*top-of-set punctured-plane*) ≅ *integer-group* ∧
¬ *contractible-space* (*top-of-set punctured-plane*)
using *reduced-homology-group-punctured-plane-one*
punctured-plane-not-contractible **by** *auto*

end

References

[1] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[2] A. F. Ramos, D. B. Hulak, and R. J. G. B. de Queiroz. The classical Seifert–van Kampen theorem. *Archive of Formal Proofs*, May 2026. `https://isa-afp.org/entries/Seifert-Van-Kampen.html`, Formal proof development.

[3] A. F. Ramos, D. B. Hulak, and R. J. G. B. de Queiroz. The fundamental group of the circle. *Archive of Formal Proofs*, July 2026. `https://isa-afp.org/entries/Fundamental_Group_Circle.html`, Formal proof development.