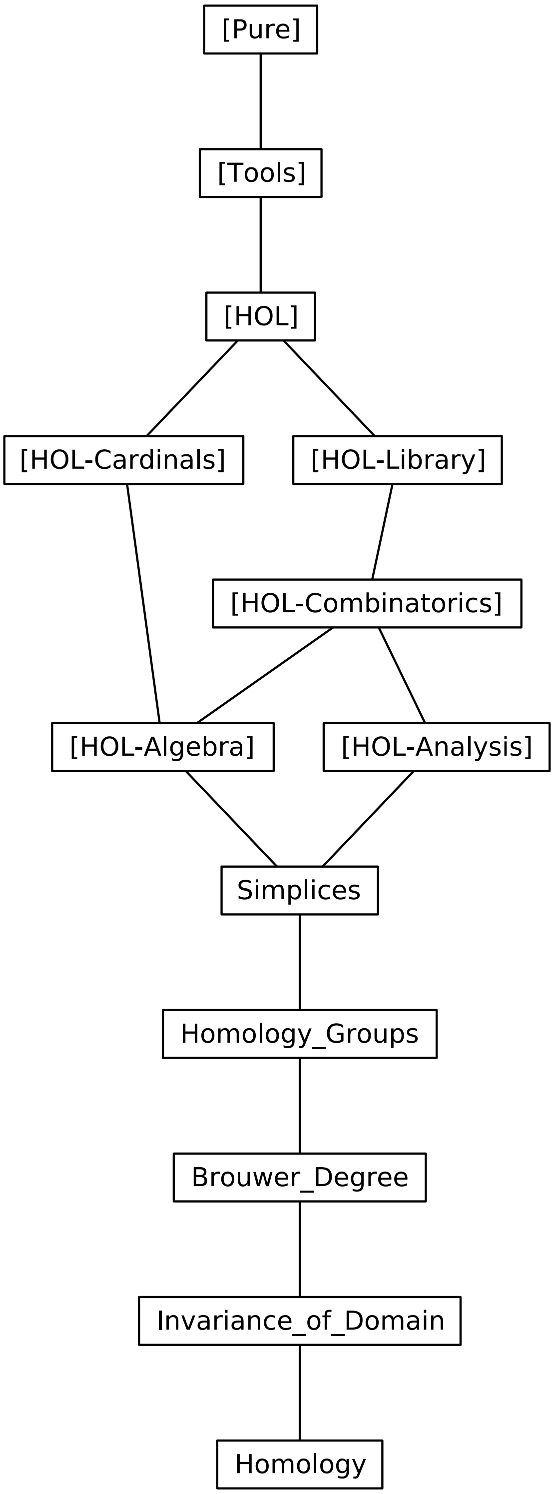
Homology

September 11, 2023

Contents


[Pure]
[Tools]
[HOL]
[HOL-Cardinals]
[HOL-Library]
[HOL-Combinatorics]
[HOL-Algebra]
[HOL-Analysis]
Simplices
Homology_Groups
Brouwer_Degree
Invariance_of_Domain
Homology

0.1 Homology, I: Simplices

```
theory Simplices
  imports
    HOL−Analysis.Function_Metric
    HOL−Analysis.Abstract_Euclidean_Space
    HOL−Algebra.Free_Abelian_Groups
begin
```

0.1.1 Standard simplices, all of which are topological subspaces of R^n.

```
type_synonym 'a chain = ((nat ⇒ real) ⇒ 'a) ⇒₀ int

definition standard_simplex :: nat ⇒ (nat ⇒ real) set where
  standard_simplex p ≡
    {x. (∀ i. 0 ≤ x i ∧ x i ≤ 1) ∧ (∀ i>p. x i = 0) ∧ (∑ i≤p. x i) = 1}

lemma topspace_standard_simplex:
  topspace(subtopology (powertop_real UNIV) (standard_simplex p))
    = standard_simplex p
  by simp

lemma basis_in_standard_simplex [simp]:
   (λj. if j = i then 1 else 0) ∈ standard_simplex p ⟷ i ≤ p
  by (auto simp: standard_simplex_def)

lemma nonempty_standard_simplex: standard_simplex p ≠ {}
  using basis_in_standard_simplex by blast

lemma standard_simplex_0: standard_simplex 0 = {(λj. if j = 0 then 1 else 0)}
  by (auto simp: standard_simplex_def)

lemma standard_simplex_mono:
  assumes p ≤ q
  shows standard_simplex p ⊆ standard_simplex q
  using assms
proof (clarsimp simp: standard_simplex_def)
  fix x :: nat ⇒ real
  assume ∀ i. 0 ≤ x i ∧ x i ≤ 1 and ∀ i>p. x i = 0 and sum x {..p} = 1
  then show sum x {..q} = 1
    using sum.mono_neutral_left [of {..q} {..p} x] assms by auto
qed

lemma closedin_standard_simplex:
   closedin (powertop_real UNIV) (standard_simplex p)
    (is closedin ?X ?S)
proof −
  have eq: standard_simplex p =
```

```
          (⋂ i. {x. x ∈ topspace ?X ∧ x i ∈ {0..1}}) ∩
          (⋂ i ∈ {p<..}. {x ∈ topspace ?X. x i ∈ {0}}) ∩
          {x ∈ topspace ?X. (∑ i≤p. x i) ∈ {1}}
    by (auto simp: standard_simplex_def topspace_product_topology)
  show ?thesis
    unfolding eq
    by (rule closedin_Int closedin_Inter continuous_map_sum
        continuous_map_product_projection closedin_continuous_map_preimage
| force | clarify)+
qed

lemma standard_simplex_01: standard_simplex p ⊆ UNIV →E {0..1}
  using standard_simplex_def by auto

lemma compactin_standard_simplex:
   compactin (powertop_real UNIV) (standard_simplex p)
proof (rule closed_compactin)
  show compactin (powertop_real UNIV) (UNIV →E {0..1})
    by (simp add: compactin_PiE)
  show standard_simplex p ⊆ UNIV →E {0..1}
    by (simp add: standard_simplex_01)
  show closedin (powertop_real UNIV) (standard_simplex p)
    by (simp add: closedin_standard_simplex)
qed

lemma convex_standard_simplex:
   ⟦x ∈ standard_simplex p; y ∈ standard_simplex p;
     0 ≤ u; u ≤ 1⟧
    ⟹ (λi. (1 − u) ∗ x i + u ∗ y i) ∈ standard_simplex p
 by (simp add: standard_simplex_def sum.distrib convex_bound_le flip: sum_distrib_left)

lemma path_connectedin_standard_simplex:
   path_connectedin (powertop_real UNIV) (standard_simplex p)
proof −
  define g where g ≡ λx y::nat⇒real. λu i. (1 − u) ∗ x i + u ∗ y i
  have continuous_map
           (subtopology euclideanreal {0..1}) (powertop_real UNIV)
           (g x y)
    if x ∈ standard_simplex p y ∈ standard_simplex p for x y
    unfolding g_def continuous_map_componentwise
    by (force intro: continuous_intros)
  moreover
  have g x y ‘ {0..1} ⊆ standard_simplex p g x y 0 = x g x y 1 = y
    if x ∈ standard_simplex p y ∈ standard_simplex p for x y
    using that by (auto simp: convex_standard_simplex g_def)
  ultimately
  show ?thesis
    unfolding path_connectedin_def path_connected_space_def pathin_def
   by (metis continuous_map_in_subtopology euclidean_product_topology top_greatest
```

```
topspace_euclidean topspace_euclidean_subtopology)
qed

lemma connectedin_standard_simplex:
  connectedin (powertop_real UNIV) (standard_simplex p)
 by (simp add: path_connectedin_imp_connectedin path_connectedin_standard_simplex)
```

0.1.2 Face map

```
definition simplical_face :: nat ⇒ (nat ⇒ 'a) ⇒ nat ⇒ 'a::comm_monoid_add
where
  simplical_face k x ≡ λi. if i < k then x i else if i = k then 0 else x(i −1)

lemma simplical_face_in_standard_simplex:
 assumes 1 ≤ p k ≤ p x ∈ standard_simplex (p − Suc 0)
 shows (simplical_face k x) ∈ standard_simplex p
proof −
 have x01: ⋀i. 0 ≤ x i ∧ x i ≤ 1 and sumx: sum x {..p − Suc 0} = 1
   using assms by (auto simp: standard_simplex_def simplical_face_def)
 have gg: ⋀g. sum g {..p} = sum g {..<k} + sum g {k..p}
   using ‹k ≤ p› sum.union_disjoint [of {..<k} {k..p}]
   by (force simp: ivl_disj_un ivl_disj_int)
 have eq: (∑ i≤p. if i < k then x i else if i = k then 0 else x (i −1))
        = (∑ i < k. x i) + (∑ i ∈ {k..p}. if i = k then 0 else x (i −1))
   by (simp add: gg)
 consider k ≤ p − Suc 0 | k = p
   using ‹k ≤ p› by linarith
 then have (∑ i≤p. if i < k then x i else if i = k then 0 else x (i −1)) = 1
 proof cases
   case 1
   have [simp]: Suc (p − Suc 0) = p
     using ‹1 ≤ p› by auto
   have (∑ i = k..p. if i = k then 0 else x (i −1)) = (∑ i = k+1..p. if i = k then
0 else x (i −1))
     by (rule sum.mono_neutral_right) auto
   also have … = (∑ i = k+1..p. x (i −1))
     by simp
   also have … = (∑ i = k..p−1. x i)
      using sum.atLeastAtMost_reindex [of Suc k p−1 λi. x (i − Suc 0)] 1 by
simp
   finally have eq2: (∑ i = k..p. if i = k then 0 else x (i −1)) = (∑ i = k..p−1.
x i) .
   with 1 show ?thesis
    by (metis (no_types, lifting) One_nat_def eq finite_atLeastAtMost finite_lessThan
ivl_disj_int(4) ivl_disj_un(10) sum.union_disjoint sumx)
 next
   case 2
   have [simp]: ({..p} ∩ {x. x < p}) = {..p − Suc 0}
     using assms by auto
```

have ($\sum i\leq p.$ *if* $i < p$ *then* x i *else if* $i = k$ *then* 0 *else* x $(i - 1)$) = ($\sum i\leq p.$ *if* $i < p$ *then* x i *else* 0)
by (*rule sum.cong*) (*auto simp*: *2*)
also have ... = *sum x* {..*p*−*1*}
by (*simp add*: *sum.If_cases*)
also have ... = *1*
by (*simp add*: *sumx*)
finally show *?thesis*
using *2* **by** *simp*
qed
then show *?thesis*
using *assms* **by** (*auto simp*: *standard_simplex_def simplical_face_def*)
qed

0.1.3 Singular simplices, forcing canonicity outside the intended domain

definition *singular_simplex* :: *nat* ⇒ *'a topology* ⇒ ((*nat* ⇒ *real*) ⇒ *'a*) ⇒ *bool*
where
singular_simplex p X f ≡
continuous_map(*subtopology* (*powertop_real UNIV*) (*standard_simplex p*)) *X f*
∧ *f* ∈ *extensional* (*standard_simplex p*)

abbreviation *singular_simplex_set* :: *nat* ⇒ *'a topology* ⇒ ((*nat* ⇒ *real*) ⇒ *'a*) *set* **where**
singular_simplex_set p X ≡ *Collect* (*singular_simplex p X*)

lemma *singular_simplex_empty*:
topspace X = {} ⟹ ¬ *singular_simplex p X f*
by (*simp add*: *singular_simplex_def continuous_map nonempty_standard_simplex*)

lemma *singular_simplex_mono*:
⟦*singular_simplex p* (*subtopology X T*) *f*; *T* ⊆ *S*⟧ ⟹ *singular_simplex p* (*subtopology X S*) *f*
by (*auto simp*: *singular_simplex_def continuous_map_in_subtopology*)

lemma *singular_simplex_subtopology*:
singular_simplex p (*subtopology X S*) *f* ⟷
singular_simplex p X f ∧ *f* ‘ (*standard_simplex p*) ⊆ *S*
by (*auto simp*: *singular_simplex_def continuous_map_in_subtopology*)

Singular face

definition *singular_face* :: *nat* ⇒ *nat* ⇒ ((*nat* ⇒ *real*) ⇒ *'a*) ⇒ (*nat* ⇒ *real*) ⇒ *'a*
where *singular_face p k f* ≡ *restrict* (*f* ∘ *simplical_face k*) (*standard_simplex* (*p* − *Suc 0*))

lemma *singular_simplex_singular_face*:

assumes *f*: *singular_simplex p X f* **and** *1* ≤ *p k* ≤ *p*
shows *singular_simplex* (*p* − *Suc 0*) *X* (*singular_face p k f*)
proof −
let *?PT* = (*powertop_real UNIV*)
have *0*: *simplical_face k* ' *standard_simplex* (*p* − *Suc 0*) ⊆ *standard_simplex p*
using *assms simplical_face_in_standard_simplex* **by** *auto*
have *1*: *continuous_map* (*subtopology ?PT* (*standard_simplex* (*p* − *Suc 0*)))
(*subtopology ?PT* (*standard_simplex p*))
(*simplical_face k*)
proof (*clarsimp simp add*: *continuous_map_in_subtopology simplical_face_in_standard_simplex continuous_map_componentwise 0*)
fix *i*
have *continuous_map ?PT euclideanreal* (λ*x*. *if i* < *k then x i else if i* = *k then 0 else x* (*i* −*1*))
by (*auto intro*: *continuous_map_product_projection*)
then show *continuous_map* (*subtopology ?PT* (*standard_simplex* (*p* − *Suc 0*))) *euclideanreal*
(λ*x*. *simplical_face k x i*)
by (*simp add*: *simplical_face_def continuous_map_from_subtopology*)
qed
have *2*: *continuous_map* (*subtopology ?PT* (*standard_simplex p*)) *X f*
using *assms*(*1*) *singular_simplex_def* **by** *blast*
show *?thesis*
by (*simp add*: *singular_simplex_def singular_face_def continuous_map_compose* [*OF 1 2*])
qed

0.1.4 Singular chains

definition *singular_chain* :: [*nat*, ′*a topology*, ′*a chain*] ⇒ *bool*
where *singular_chain p X c* ≡ *Poly_Mapping.keys c* ⊆ *singular_simplex_set p X*

abbreviation *singular_chain_set* :: [*nat*, ′*a topology*] ⇒ (′*a chain*) *set*
where *singular_chain_set p X* ≡ *Collect* (*singular_chain p X*)

lemma *singular_chain_empty*:
topspace X = {} ⟹ *singular_chain p X c* ⟷ *c* = *0*
by (*auto simp*: *singular_chain_def singular_simplex_empty subset_eq poly_mapping_eqI*)

lemma *singular_chain_mono*:
⟦*singular_chain p* (*subtopology X T*) *c*; *T* ⊆ *S*⟧
⟹ *singular_chain p* (*subtopology X S*) *c*
unfolding *singular_chain_def* **using** *singular_simplex_mono* **by** *blast*

lemma *singular_chain_subtopology*:
singular_chain p (*subtopology X S*) *c* ⟷
singular_chain p X c ∧ (∀*f* ∈ *Poly_Mapping.keys c*. *f* ' (*standard_simplex p*) ⊆ *S*)

unfolding *singular_chain_def*
by (*fastforce simp add*: *singular_simplex_subtopology subset_eq*)

lemma *singular_chain_0* [*iff*]: *singular_chain p X 0*
by (*auto simp*: *singular_chain_def*)

lemma *singular_chain_of*:
singular_chain p X (*frag_of c*) $\longleftrightarrow$ *singular_simplex p X c*
by (*auto simp*: *singular_chain_def*)

lemma *singular_chain_cmul*:
singular_chain p X c $\Longrightarrow$ *singular_chain p X* (*frag_cmul a c*)
by (*auto simp*: *singular_chain_def*)

lemma *singular_chain_minus*:
singular_chain p X ($-c$) $\longleftrightarrow$ *singular_chain p X c*
by (*auto simp*: *singular_chain_def*)

lemma *singular_chain_add*:
$[\![$*singular_chain p X a*; *singular_chain p X b*$]\!]$ $\Longrightarrow$ *singular_chain p X* ($a+b$)
unfolding *singular_chain_def*
using *keys_add* [*of a b*] **by** *blast*

lemma *singular_chain_diff*:
$[\![$*singular_chain p X a*; *singular_chain p X b*$]\!]$ $\Longrightarrow$ *singular_chain p X* ($a-b$)
unfolding *singular_chain_def*
using *keys_diff* [*of a b*] **by** *blast*

lemma *singular_chain_sum*:
($\bigwedge i.\ i \in I \Longrightarrow$ *singular_chain p X* (*f i*)) $\Longrightarrow$ *singular_chain p X* ($\sum i{\in}I.\ f\ i$)
unfolding *singular_chain_def*
using *keys_sum* [*of f I*] **by** *blast*

lemma *singular_chain_extend*:
($\bigwedge c.\ c \in$ *Poly_Mapping.keys x* $\Longrightarrow$ *singular_chain p X* (*f c*))
$\Longrightarrow$ *singular_chain p X* (*frag_extend f x*)
by (*simp add*: *frag_extend_def singular_chain_cmul singular_chain_sum*)

0.1.5 Boundary homomorphism for singular chains

definition *chain_boundary* :: *nat* $\Rightarrow$ ('*a chain*) $\Rightarrow$ '*a chain*
where *chain_boundary p c* $\equiv$
(*if p* = *0 then 0 else*
frag_extend (λf. ($\sum k{\le}p$. *frag_cmul* ((-1) ^ *k*) (*frag_of*(*singular_face p k f*)))) *c*)

lemma *singular_chain_boundary*:
assumes *singular_chain p X c*
shows *singular_chain* (*p* $-$ *Suc 0*) *X* (*chain_boundary p c*)

unfolding *chain_boundary_def*
proof (*clarsimp intro*!: *singular_chain_extend singular_chain_sum singular_chain_cmul*)
show $\bigwedge d\ k.$ ⟦*0* < *p*; *d* ∈ *Poly_Mapping.keys c*; *k* ≤ *p*⟧
⟹ *singular_chain* (*p* − *Suc 0*) *X* (*frag_of* (*singular_face p k d*))
using *assms* **by** (*auto simp*: *singular_chain_def intro*: *singular_simplex_singular_face*)
qed

lemma *singular_chain_boundary_alt*:
singular_chain (*Suc p*) *X c* ⟹ *singular_chain p X* (*chain_boundary* (*Suc p*) *c*)
using *singular_chain_boundary* **by** *force*

lemma *chain_boundary_0* [*simp*]: *chain_boundary p 0* = *0*
by (*simp add*: *chain_boundary_def*)

lemma *chain_boundary_cmul*:
chain_boundary p (*frag_cmul k c*) = *frag_cmul k* (*chain_boundary p c*)
by (*auto simp*: *chain_boundary_def frag_extend_cmul*)

lemma *chain_boundary_minus*:
chain_boundary p (− *c*) = − (*chain_boundary p c*)
by (*metis chain_boundary_cmul frag_cmul_minus_one*)

lemma *chain_boundary_add*:
chain_boundary p (*a*+*b*) = *chain_boundary p a* + *chain_boundary p b*
by (*simp add*: *chain_boundary_def frag_extend_add*)

lemma *chain_boundary_diff*:
chain_boundary p (*a*−*b*) = *chain_boundary p a* − *chain_boundary p b*
using *chain_boundary_add* [*of p a* −*b*]
by (*simp add*: *chain_boundary_minus*)

lemma *chain_boundary_sum*:
chain_boundary p (*sum g I*) = *sum* (*chain_boundary p* ∘ *g*) *I*
by (*induction I rule*: *infinite_finite_induct*) (*simp_all add*: *chain_boundary_add*)

lemma *chain_boundary_sum'*:
finite I ⟹ *chain_boundary p* (*sum' g I*) = *sum'* (*chain_boundary p* ∘ *g*) *I*
by (*induction I rule*: *finite_induct*) (*simp_all add*: *chain_boundary_add*)

lemma *chain_boundary_of*:
chain_boundary p (*frag_of f*) =
(*if p* = *0 then 0*
else ($\sum$ *k*≤*p*. *frag_cmul* ((−*1*) ^ *k*) (*frag_of*(*singular_face p k f*))))
by (*simp add*: *chain_boundary_def*)

0.1.6 Factoring out chains in a subtopology for relative homology

```
definition mod_subset
  where mod_subset p X ≡ {(a,b). singular_chain p X (a − b)}

lemma mod_subset_empty [simp]:
   (a,b) ∈ (mod_subset p (subtopology X {})) ⟷ a = b
  by (simp add: mod_subset_def singular_chain_empty)

lemma mod_subset_refl [simp]: (c,c) ∈ mod_subset p X
  by (auto simp: mod_subset_def)

lemma mod_subset_cmul:
  assumes (a,b) ∈ mod_subset p X
  shows (frag_cmul k a, frag_cmul k b) ∈ mod_subset p X
  using assms
 by (simp add: mod_subset_def) (metis (no_types, lifting) add_diff_cancel diff_add_cancel
frag_cmul_distrib2 singular_chain_cmul)

lemma mod_subset_add:
   ⟦(c1,c2) ∈ mod_subset p X; (d1,d2) ∈ mod_subset p X⟧ ⟹ (c1+d1, c2+d2)
∈ mod_subset p X
  by (simp add: mod_subset_def add_diff_add singular_chain_add)
```

0.1.7 Relative cycles $Z_pX(S)$ where X is a topology and S a subset

```
definition singular_relcycle :: nat ⇒ 'a topology ⇒ 'a set ⇒ ('a chain) ⇒ bool
  where singular_relcycle p X S ≡
        λc. singular_chain p X c ∧ (chain_boundary p c, 0) ∈ mod_subset (p−1)
(subtopology X S)

abbreviation singular_relcycle_set
  where singular_relcycle_set p X S ≡ Collect (singular_relcycle p X S)

lemma singular_relcycle_restrict [simp]:
   singular_relcycle p X (topspace X ∩ S) = singular_relcycle p X S
proof −
  have eq: subtopology X (topspace X ∩ S) = subtopology X S
    by (metis subtopology_subtopology subtopology_topspace)
  show ?thesis
    by (force simp: singular_relcycle_def eq)
qed

lemma singular_relcycle:
   singular_relcycle p X S c ⟷
   singular_chain p X c ∧ singular_chain (p−1) (subtopology X S) (chain_boundary
p c)
  by (simp add: singular_relcycle_def mod_subset_def)
```

lemma *singular_relcycle_0* [*simp*]: *singular_relcycle p X S 0*
by (*auto simp*: *singular_relcycle_def*)

lemma *singular_relcycle_cmul*:
singular_relcycle p X S c $\Longrightarrow$ *singular_relcycle p X S* (*frag_cmul k c*)
by (*auto simp*: *singular_relcycle_def chain_boundary_cmul dest*: *singular_chain_cmul mod_subset_cmul*)

lemma *singular_relcycle_minus*:
singular_relcycle p X S $(-c)$ $\longleftrightarrow$ *singular_relcycle p X S c*
by (*simp add*: *chain_boundary_minus singular_chain_minus singular_relcycle*)

lemma *singular_relcycle_add*:
⟦*singular_relcycle p X S a*; *singular_relcycle p X S b*⟧
$\Longrightarrow$ *singular_relcycle p X S* $(a+b)$
by (*simp add*: *singular_relcycle_def chain_boundary_add mod_subset_def singular_chain_add*)

lemma *singular_relcycle_sum*:
⟦$\bigwedge i.\ i \in I \Longrightarrow$ *singular_relcycle p X S* (*f i*)⟧
$\Longrightarrow$ *singular_relcycle p X S* (*sum f I*)
by (*induction I rule*: *infinite_finite_induct*) (*auto simp*: *singular_relcycle_add*)

lemma *singular_relcycle_diff*:
⟦*singular_relcycle p X S a*; *singular_relcycle p X S b*⟧
$\Longrightarrow$ *singular_relcycle p X S* $(a-b)$
by (*metis singular_relcycle_add singular_relcycle_minus uminus_add_conv_diff*)

lemma *singular_cycle*:
singular_relcycle p X {} *c* $\longleftrightarrow$ *singular_chain p X c* $\wedge$ *chain_boundary p c* = *0*
using *mod_subset_empty* **by** (*auto simp*: *singular_relcycle_def*)

lemma *singular_cycle_mono*:
⟦*singular_relcycle p* (*subtopology X T*) {} *c*; $T \subseteq S$⟧
$\Longrightarrow$ *singular_relcycle p* (*subtopology X S*) {} *c*
by (*auto simp*: *singular_cycle elim*: *singular_chain_mono*)

0.1.8 Relative boundaries $B_p X S$, where X is a topology and S a subset.

definition *singular_relboundary* :: *nat* $\Rightarrow$ *'a topology* $\Rightarrow$ *'a set* $\Rightarrow$ (*'a chain*) $\Rightarrow$ *bool*
where
singular_relboundary p X S $\equiv$
$\lambda c.\ \exists d.$ *singular_chain* (*Suc p*) *X d* $\wedge$ (*chain_boundary* (*Suc p*) *d*, *c*) $\in$ (*mod_subset p* (*subtopology X S*))

abbreviation *singular_relboundary_set* :: *nat* ⇒ *'a topology* ⇒ *'a set* ⇒ (*'a chain*) *set*
where *singular_relboundary_set p X S* ≡ *Collect* (*singular_relboundary p X S*)

lemma *singular_relboundary_restrict* [*simp*]:
singular_relboundary p X (*topspace X* ∩ *S*) = *singular_relboundary p X S*
unfolding *singular_relboundary_def*
by (*metis* (*no_types*, *opaque_lifting*) *subtopology_subtopology subtopology_topspace*)

lemma *singular_relboundary_alt*:
singular_relboundary p X S c ⟷
(∃ *d e*. *singular_chain* (*Suc p*) *X d* ∧ *singular_chain p* (*subtopology X S*) *e* ∧
chain_boundary (*Suc p*) *d* = *c* + *e*)
unfolding *singular_relboundary_def mod_subset_def* **by** *fastforce*

lemma *singular_relboundary*:
singular_relboundary p X S c ⟷
(∃ *d e*. *singular_chain* (*Suc p*) *X d* ∧ *singular_chain p* (*subtopology X S*) *e* ∧
(*chain_boundary* (*Suc p*) *d*) + *e* = *c*)
using *singular_chain_minus*
by (*fastforce simp add*: *singular_relboundary_alt*)

lemma *singular_boundary*:
singular_relboundary p X {} *c* ⟷
(∃ *d*. *singular_chain* (*Suc p*) *X d* ∧ *chain_boundary* (*Suc p*) *d* = *c*)
by (*meson mod_subset_empty singular_relboundary_def*)

lemma *singular_boundary_imp_chain*:
singular_relboundary p X {} *c* ⟹ *singular_chain p X c*
by (*auto simp*: *singular_relboundary singular_chain_boundary_alt singular_chain_empty*)

lemma *singular_boundary_mono*:
⟦*T* ⊆ *S*; *singular_relboundary p* (*subtopology X T*) {} *c*⟧
⟹ *singular_relboundary p* (*subtopology X S*) {} *c*
by (*metis mod_subset_empty singular_chain_mono singular_relboundary_def*)

lemma *singular_relboundary_imp_chain*:
singular_relboundary p X S c ⟹ *singular_chain p X c*
unfolding *singular_relboundary singular_chain_subtopology*
by (*blast intro*: *singular_chain_add singular_chain_boundary_alt*)

lemma *singular_chain_imp_relboundary*:
singular_chain p (*subtopology X S*) *c* ⟹ *singular_relboundary p X S c*
unfolding *singular_relboundary_def*
using *mod_subset_def singular_chain_minus* **by** *fastforce*

lemma *singular_relboundary_0* [*simp*]: *singular_relboundary p X S 0*
unfolding *singular_relboundary_def*
by (*rule_tac x=0* **in** *exI*) *auto*

lemma *singular_relboundary_cmul*:
singular_relboundary p X S c $\Longrightarrow$ *singular_relboundary p X S (frag_cmul a c)*
unfolding *singular_relboundary_def*
by (*metis chain_boundary_cmul mod_subset_cmul singular_chain_cmul*)

lemma *singular_relboundary_minus*:
singular_relboundary p X S $(-c)$ $\longleftrightarrow$ *singular_relboundary p X S c*
using *singular_relboundary_cmul*
by (*metis add.inverse_inverse frag_cmul_minus_one*)

lemma *singular_relboundary_add*:
⟦*singular_relboundary p X S a*; *singular_relboundary p X S b*⟧ $\Longrightarrow$ *singular_relboundary p X S* $(a+b)$
unfolding *singular_relboundary_def*
by (*metis chain_boundary_add mod_subset_add singular_chain_add*)

lemma *singular_relboundary_diff*:
⟦*singular_relboundary p X S a*; *singular_relboundary p X S b*⟧ $\Longrightarrow$ *singular_relboundary p X S* $(a-b)$
by (*metis uminus_add_conv_diff singular_relboundary_minus singular_relboundary_add*)

0.1.9 The (relative) homology relation

definition *homologous_rel* :: [*nat*,'*a topology*,'*a set*,'*a chain*,'*a chain*] $\Rightarrow$ *bool*
where *homologous_rel p X S* $\equiv$ λ*a b*. *singular_relboundary p X S* $(a-b)$

abbreviation *homologous_rel_set*
where *homologous_rel_set p X S a* $\equiv$ *Collect* (*homologous_rel p X S a*)

lemma *homologous_rel_restrict* [*simp*]:
homologous_rel p X (*topspace X* $\cap$ *S*) = *homologous_rel p X S*
unfolding *homologous_rel_def* **by** (*metis singular_relboundary_restrict*)

lemma *homologous_rel_refl* [*simp*]: *homologous_rel p X S c c*
unfolding *homologous_rel_def* **by** *auto*

lemma *homologous_rel_sym*:
homologous_rel p X S a b = *homologous_rel p X S b a*
unfolding *homologous_rel_def*
using *singular_relboundary_minus* **by** *fastforce*

lemma *homologous_rel_trans*:
assumes *homologous_rel p X S b c homologous_rel p X S a b*
shows *homologous_rel p X S a c*
using *homologous_rel_def*
proof –
have *singular_relboundary p X S* $(b - c)$
using *assms* **unfolding** *homologous_rel_def* **by** *blast*

moreover have *singular_relboundary p X S* (*b* − *a*)
  using *assms* **by** (*meson homologous_rel_def homologous_rel_sym*)
ultimately have *singular_relboundary p X S* (*c* − *a*)
  using *singular_relboundary_diff* **by** *fastforce*
then show *?thesis*
  by (*meson homologous_rel_def homologous_rel_sym*)
qed

lemma *homologous_rel_eq*:
  homologous_rel p X S a = *homologous_rel p X S b* ⟷
  homologous_rel p X S a b
using *homologous_rel_sym homologous_rel_trans* **by** *fastforce*

lemma *homologous_rel_set_eq*:
  homologous_rel_set p X S a = *homologous_rel_set p X S b* ⟷
  homologous_rel p X S a b
by (*metis homologous_rel_eq mem_Collect_eq*)

lemma *homologous_rel_singular_chain*:
homologous_rel p X S a b ⟹ (*singular_chain p X a* ⟷ *singular_chain p X b*)
unfolding *homologous_rel_def*
using *singular_chain_diff singular_chain_add*
by (*fastforce dest*: *singular_relboundary_imp_chain*)

lemma *homologous_rel_add*:
  ⟦*homologous_rel p X S a a*′; *homologous_rel p X S b b*′⟧
    ⟹ *homologous_rel p X S* (*a*+*b*) (*a*′+*b*′)
unfolding *homologous_rel_def*
by (*simp add*: *add_diff_add singular_relboundary_add*)

lemma *homologous_rel_diff*:
assumes *homologous_rel p X S a a*′ *homologous_rel p X S b b*′
shows *homologous_rel p X S* (*a* − *b*) (*a*′ − *b*′)
proof −
have *singular_relboundary p X S* ((*a* − *a*′) − (*b* − *b*′))
  using *assms singular_relboundary_diff* **unfolding** *homologous_rel_def* **by** *blast*
then show *?thesis*
  by (*simp add*: *homologous_rel_def algebra_simps*)
qed

lemma *homologous_rel_sum*:
assumes *f*: *finite* {*i* ∈ *I*. *f i* ≠ *0*} **and** *g*: *finite* {*i* ∈ *I*. *g i* ≠ *0*}
  and *h*: ⋀*i*. *i* ∈ *I* ⟹ *homologous_rel p X S* (*f i*) (*g i*)
shows *homologous_rel p X S* (*sum f I*) (*sum g I*)
proof (*cases finite I*)
case *True*
let *?L* = {*i* ∈ *I*. *f i* ≠ *0*} ∪ {*i* ∈ *I*. *g i* ≠ *0*}
have *L*: *finite ?L ?L* ⊆ *I*

```
    using f g by blast+
  have sum f I = sum f ?L
    by (rule comm_monoid_add_class.sum.mono_neutral_right [OF True]) auto
  moreover have sum g I = sum g ?L
    by (rule comm_monoid_add_class.sum.mono_neutral_right [OF True]) auto
  moreover have *: homologous_rel p X S (f i) (g i) if i ∈ ?L for i
    using h that by auto
  have homologous_rel p X S (sum f ?L) (sum g ?L)
    using L
  proof induction
    case (insert j J)
    then show ?case
      by (simp add: h homologous_rel_add)
  qed auto
  ultimately show ?thesis
    by simp
qed auto


lemma chain_homotopic_imp_homologous_rel:
  assumes
   ⋀c. singular_chain p X c ⟹ singular_chain (Suc p) X' (h c)
  ⋀c. singular_chain (p -1) (subtopology X S) c ⟹ singular_chain p (subtopology
X' T) (h' c)
   ⋀c. singular_chain p X c
           ⟹ (chain_boundary (Suc p) (h c)) + (h'(chain_boundary p c)) = f c
- g c
    singular_relcycle p X S c
  shows homologous_rel p X' T (f c) (g c)
proof -
  have singular_chain p (subtopology X' T) (chain_boundary (Suc p) (h c) - (f
c - g c))
    using assms
   by (metis (no_types, lifting) add_diff_cancel_left' minus_diff_eq singular_chain_minus
singular_relcycle)
  then show ?thesis
  using assms
    by (metis homologous_rel_def singular_relboundary singular_relcycle)
qed
```

0.1.10 Show that all boundaries are cycles, the key "chain complex" property.

```
lemma chain_boundary_boundary:
  assumes singular_chain p X c
  shows chain_boundary (p - Suc 0) (chain_boundary p c) = 0
proof (cases p -1 = 0)
  case False
  then have 2 ≤ p
```

```
    by auto
  show ?thesis
    using assms
    unfolding singular_chain_def
  proof (induction rule: frag_induction)
    case (one g)
    then have ss: singular_simplex p X g
      by simp
    have eql: {..p} × {..p - Suc 0} ∩ {(x, y). y < x} = (λ(j,i). (Suc i, j)) ` {(i,j).
i ≤ j ∧ j ≤ p -1}
      using False
       by (auto simp: image_def) (metis One_nat_def diff_Suc_1 diff_le_mono
le_refl lessE less_imp_le_nat)
    have eqr: {..p} × {..p - Suc 0} - {(x, y). y < x} = {(i,j). i ≤ j ∧ j ≤ p -1}
      by auto
    have eqf: singular_face (p - Suc 0) i (singular_face p (Suc j) g) =
              singular_face (p - Suc 0) j (singular_face p i g) if i ≤ j j ≤ p - Suc
0 for i j
    proof (rule ext)
      fix t
      show singular_face (p - Suc 0) i (singular_face p (Suc j) g) t =
            singular_face (p - Suc 0) j (singular_face p i g) t
      proof (cases t ∈ standard_simplex (p -1 -1))
        case True
        have fi: simplical_face i t ∈ standard_simplex (p - Suc 0)
          using False True simplical_face_in_standard_simplex that by force
        have fj: simplical_face j t ∈ standard_simplex (p - Suc 0)
          by (metis False One_nat_def True simplical_face_in_standard_simplex
less_one not_less that(2))
          have eq: simplical_face (Suc j) (simplical_face i t) = simplical_face i
(simplical_face j t)
          using True that ss
          unfolding standard_simplex_def simplical_face_def by fastforce
        show ?thesis by (simp add: singular_face_def fi fj eq)
      qed (simp add: singular_face_def)
    qed
    show ?case
    proof (cases p = 1)
      case False
      have eq0: frag_cmul (-1) a = b ⟹ a + b = 0 for a b
        by (simp add: neg_eq_iff_add_eq_0)
      have *: (∑x≤p. ∑i≤p - Suc 0.
               frag_cmul ((-1) ^ (x + i)) (frag_of (singular_face (p - Suc 0) i
(singular_face p x g))))
             = 0
       apply (simp add: sum.cartesian_product sum.Int_Diff [of _ × _ _ {(x,y).
y < x}])
        apply (rule eq0)
      unfolding frag_cmul_sum prod.case_distrib [of frag_cmul (-1)] frag_cmul_cmul
```

```
eql eqr
      apply (force simp: inj_on_def sum.reindex add.commute eqf intro: sum.cong)
        done
      show ?thesis
         using False by (simp add: chain_boundary_of chain_boundary_sum
chain_boundary_cmul frag_cmul_sum * flip: power_add)
    qed (simp add: chain_boundary_def)
  next
    case (diff a b)
    then show ?case
      by (simp add: chain_boundary_diff)
  qed auto
qed (simp add: chain_boundary_def)


lemma chain_boundary_boundary_alt:
   singular_chain (Suc p) X c ⟹ chain_boundary p (chain_boundary (Suc p) c)
= 0
  using chain_boundary_boundary by force

lemma singular_relboundary_imp_relcycle:
  assumes singular_relboundary p X S c
  shows singular_relcycle p X S c
proof -
  obtain d e where d: singular_chain (Suc p) X d
             and e: singular_chain p (subtopology X S) e
             and c: c = chain_boundary (Suc p) d + e
    using assms by (auto simp: singular_relboundary singular_relcycle)
   have 1: singular_chain (p - Suc 0) (subtopology X S) (chain_boundary p
(chain_boundary (Suc p) d))
    using d chain_boundary_boundary_alt by fastforce
  have 2: singular_chain (p - Suc 0) (subtopology X S) (chain_boundary p e)
     using ‹singular_chain p (subtopology X S) e› singular_chain_boundary by
auto
  have singular_chain p X c
    using assms singular_relboundary_imp_chain by auto
  moreover have singular_chain (p - Suc 0) (subtopology X S) (chain_boundary
p c)
    by (simp add: c chain_boundary_add singular_chain_add 1 2)
  ultimately show ?thesis
    by (simp add: singular_relcycle)
qed

lemma homologous_rel_singular_relcycle_1:
  assumes homologous_rel p X S c1 c2 singular_relcycle p X S c1
  shows singular_relcycle p X S c2
  using assms
 by (metis diff_add_cancel homologous_rel_def homologous_rel_sym singular_relboundary_imp_relcycle
singular_relcycle_add)
```

lemma *homologous_rel_singular_relcycle*:
assumes *homologous_rel p X S c1 c2*
shows *singular_relcycle p X S c1 = singular_relcycle p X S c2*
using *assms homologous_rel_singular_relcycle_1*
using *homologous_rel_sym* **by** *blast*

0.1.11 Operations induced by a continuous map g between topological spaces

definition *simplex_map* :: *nat* ⇒ (*'b* ⇒ *'a*) ⇒ ((*nat* ⇒ *real*) ⇒ *'b*) ⇒ (*nat* ⇒ *real*) ⇒ *'a*
where *simplex_map p g c* ≡ *restrict* (*g* ∘ *c*) (*standard_simplex p*)

lemma *singular_simplex_simplex_map*:
⟦*singular_simplex p X f*; *continuous_map X X' g*⟧
⟹ *singular_simplex p X'* (*simplex_map p g f*)
unfolding *singular_simplex_def simplex_map_def*
by (*auto simp*: *continuous_map_compose*)

lemma *simplex_map_eq*:
⟦*singular_simplex p X c*;
⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*⟧
⟹ *simplex_map p f c* = *simplex_map p g c*
by (*auto simp*: *singular_simplex_def simplex_map_def continuous_map_def Pi_iff*)

lemma *simplex_map_id_gen*:
⟦*singular_simplex p X c*;
⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *x*⟧
⟹ *simplex_map p f c* = *c*
unfolding *singular_simplex_def simplex_map_def continuous_map_def*
using *extensional_arb* **by** *fastforce*

lemma *simplex_map_id* [*simp*]:
simplex_map p id = (λ*c*. *restrict c* (*standard_simplex p*))
by (*auto simp*: *simplex_map_def*)

lemma *simplex_map_compose*:
simplex_map p (*h* ∘ *g*) = *simplex_map p h* ∘ *simplex_map p g*
unfolding *simplex_map_def* **by** *force*

lemma *singular_face_simplex_map*:
⟦*1* ≤ *p*; *k* ≤ *p*⟧
⟹ *singular_face p k* (*simplex_map p f c*) = *simplex_map* (*p* − *Suc 0*) *f* (*c* ∘ *simplical_face k*)
unfolding *simplex_map_def singular_face_def*
by (*force simp*: *simplical_face_in_standard_simplex*)

lemma *singular_face_restrict* [*simp*]:
assumes $p > 0$ $i \le p$
shows *singular_face p i* (*restrict f* (*standard_simplex p*)) = *singular_face p i f*
by (*metis assms One_nat_def Suc_leI simplex_map_id singular_face_def singular_face_simplex_map*)

definition *chain_map* :: *nat* ⇒ ('*b* ⇒ '*a*) ⇒ (((*nat* ⇒ *real*) ⇒ '*b*) $\Rightarrow_0$ *int*) ⇒ '*a chain*
where *chain_map p g c* ≡ *frag_extend* (*frag_of* ∘ *simplex_map p g*) *c*

lemma *singular_chain_chain_map*:
⟦*singular_chain p X c*; *continuous_map X X' g*⟧ ⟹ *singular_chain p X'* (*chain_map p g c*)
unfolding *chain_map_def*
by (*force simp add*: *singular_chain_def subset_iff*
intro!: *singular_chain_extend singular_simplex_simplex_map*)

lemma *chain_map_0* [*simp*]: *chain_map p g 0* = *0*
by (*auto simp*: *chain_map_def*)

lemma *chain_map_of* [*simp*]: *chain_map p g* (*frag_of f*) = *frag_of* (*simplex_map p g f*)
by (*simp add*: *chain_map_def*)

lemma *chain_map_cmul* [*simp*]:
chain_map p g (*frag_cmul a c*) = *frag_cmul a* (*chain_map p g c*)
by (*simp add*: *frag_extend_cmul chain_map_def*)

lemma *chain_map_minus*: *chain_map p g* (−*c*) = − (*chain_map p g c*)
by (*simp add*: *frag_extend_minus chain_map_def*)

lemma *chain_map_add*:
chain_map p g (*a*+*b*) = *chain_map p g a* + *chain_map p g b*
by (*simp add*: *frag_extend_add chain_map_def*)

lemma *chain_map_diff*:
chain_map p g (*a*−*b*) = *chain_map p g a* − *chain_map p g b*
by (*simp add*: *frag_extend_diff chain_map_def*)

lemma *chain_map_sum*:
finite I ⟹ *chain_map p g* (*sum f I*) = *sum* (*chain_map p g* ∘ *f*) *I*
by (*simp add*: *frag_extend_sum chain_map_def*)

lemma *chain_map_eq*:
⟦*singular_chain p X c*; ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*⟧
⟹ *chain_map p f c* = *chain_map p g c*
unfolding *singular_chain_def*
proof (*induction rule*: *frag_induction*)

case (*one x*)
then show *?case*
by (*metis* (*no_types*, *lifting*) *chain_map_of mem_Collect_eq simplex_map_eq*)
qed (*auto simp*: *chain_map_diff*)

lemma *chain_map_id_gen*:
⟦*singular_chain p X c*; $\bigwedge x.\ x \in topspace\ X \Longrightarrow f\ x = x$⟧
$\Longrightarrow$ *chain_map p f c* = *c*
unfolding *singular_chain_def*
by (*erule frag_induction*) (*auto simp*: *chain_map_diff simplex_map_id_gen*)

lemma *chain_map_ident*:
singular_chain p X c $\Longrightarrow$ *chain_map p id c* = *c*
by (*simp add*: *chain_map_id_gen*)

lemma *chain_map_id*:
chain_map p id = *frag_extend* (*frag_of* ∘ (λ*f*. *restrict f* (*standard_simplex p*)))
by (*auto simp*: *chain_map_def*)

lemma *chain_map_compose*:
chain_map p (*h* ∘ *g*) = *chain_map p h* ∘ *chain_map p g*
proof
show *chain_map p* (*h* ∘ *g*) *c* = (*chain_map p h* ∘ *chain_map p g*) *c* **for** *c*
using *subset_UNIV*
proof (*induction c rule*: *frag_induction*)
case (*one x*)
then show *?case*
by *simp* (*metis* (*mono_tags*, *lifting*) *comp_eq_dest_lhs restrict_apply simplex_map_def*)
next
case (*diff a b*)
then show *?case*
by (*simp add*: *chain_map_diff*)
qed *auto*
qed

lemma *singular_simplex_chain_map_id*:
assumes *singular_simplex p X f*
shows *chain_map p f* (*frag_of* (*restrict id* (*standard_simplex p*))) = *frag_of f*
proof –
have (*restrict* (*f* ∘ *restrict id* (*standard_simplex p*)) (*standard_simplex p*)) = *f*
by (*rule ext*) (*metis assms comp_apply extensional_arb id_apply restrict_apply singular_simplex_def*)
then show *?thesis*
by (*simp add*: *simplex_map_def*)
qed

lemma *chain_boundary_chain_map*:
assumes *singular_chain p X c*

shows *chain_boundary p (chain_map p g c) = chain_map (p − Suc 0) g (chain_boundary p c)*
using *assms* **unfolding** *singular_chain_def*
proof *(induction c rule: frag_induction)*
case *(one x)*
then have *singular_face p i (simplex_map p g x) = simplex_map (p − Suc 0) g (singular_face p i x)*
if *0 ≤ i i ≤ p p ≠ 0* **for** *i*
using *that*
by *(fastforce simp add: singular_face_def simplex_map_def simplical_face_in_standard_simplex)*
then show *?case*
by *(auto simp: chain_boundary_of chain_map_sum)*
next
case *(diff a b)*
then show *?case*
by *(simp add: chain_boundary_diff chain_map_diff)*
qed *auto*

lemma *singular_relcycle_chain_map*:
assumes *singular_relcycle p X S c continuous_map X X′ g g ‘ S ⊆ T*
shows *singular_relcycle p X′ T (chain_map p g c)*
proof −
have *continuous_map (subtopology X S) (subtopology X′ T) g*
using *assms*
using *continuous_map_from_subtopology continuous_map_in_subtopology topspace_subtopology* **by** *fastforce*
then show *?thesis*
using *chain_boundary_chain_map [of p X c g]*
by *(metis One_nat_def assms(1) assms(2) singular_chain_chain_map singular_relcycle)*
qed

lemma *singular_relboundary_chain_map*:
assumes *singular_relboundary p X S c continuous_map X X′ g g ‘ S ⊆ T*
shows *singular_relboundary p X′ T (chain_map p g c)*
proof −
obtain *d e* **where** *d: singular_chain (Suc p) X d*
and *e: singular_chain p (subtopology X S) e* **and** *c: c = chain_boundary (Suc p) d + e*
using *assms* **by** *(auto simp: singular_relboundary)*
have *singular_chain (Suc p) X′ (chain_map (Suc p) g d)*
using *assms(2) d singular_chain_chain_map* **by** *blast*
moreover have *singular_chain p (subtopology X′ T) (chain_map p g e)*
proof −
have *∀ t. g ‘ topspace (subtopology t S) ⊆ T*
by *(metis assms(3) closure_of_subset_subtopology closure_of_topspace dual_order.trans image_mono)*
then show *?thesis*
by *(meson assms(2) continuous_map_from_subtopology continuous_map_in_subtopology*

e singular_chain_chain_map)
 qed
 moreover have *chain_boundary* (*Suc p*) (*chain_map* (*Suc p*) *g d*) + *chain_map p g e* =
 chain_map p g (*chain_boundary* (*Suc p*) *d* + *e*)
 by (*metis One_nat_def chain_boundary_chain_map chain_map_add d diff_Suc_1*)
 ultimately show *?thesis*
 unfolding *singular_relboundary*
 using *c* **by** *blast*
qed

0.1.12 Homology of one-point spaces degenerates except for $p = 0$.

lemma *singular_simplex_singleton*:
 assumes *topspace X* = {*a*}
 shows *singular_simplex p X f* ⟷ *f* = *restrict* (λ*x*. *a*) (*standard_simplex p*) (**is** *?lhs* = *?rhs*)
proof
 assume *L*: *?lhs*
 then show *?rhs*
 proof −
 have *continuous_map* (*subtopology* (*product_topology* (λ*n*. *euclideanreal*) *UNIV*) (*standard_simplex p*)) *X f*
 using ‹*singular_simplex p X f*› *singular_simplex_def* **by** *blast*
 then have ⋀*c*. *c* ∉ *standard_simplex p* ∨ *f c* = *a*
 by (*simp add*: *assms continuous_map_def Pi_iff*)
 then show *?thesis*
 by (*metis* (*no_types*) *L extensional_restrict restrict_ext singular_simplex_def*)
 qed
next
 assume *?rhs*
 with *assms* **show** *?lhs*
 by (*auto simp*: *singular_simplex_def*)
qed

lemma *singular_chain_singleton*:
 assumes *topspace X* = {*a*}
 shows *singular_chain p X c* ⟷
 (∃ *b*. *c* = *frag_cmul b* (*frag_of*(*restrict* (λ*x*. *a*) (*standard_simplex p*))))
 (**is** *?lhs* = *?rhs*)
proof
 let *?f* = *restrict* (λ*x*. *a*) (*standard_simplex p*)
 assume *L*: *?lhs*
 with *assms* **have** *Poly_Mapping.keys c* ⊆ {*?f*}
 by (*auto simp*: *singular_chain_def singular_simplex_singleton*)
 then consider *Poly_Mapping.keys c* = {} | *Poly_Mapping.keys c* = {*?f*}
 by *blast*
 then show *?rhs*

```
  proof cases
    case 1
    with L show ?thesis
      by (metis frag_cmul_zero keys_eq_empty)
  next
    case 2
   then have ∃ b. frag_extend frag_of c = frag_cmul b (frag_of (λx∈standard_simplex
p. a))
      by (force simp: frag_extend_def)
    then show ?thesis
      by (metis frag_expansion)
  qed
next
  assume ?rhs
  with assms show ?lhs
    by (auto simp: singular_chain_def singular_simplex_singleton)
qed

lemma chain_boundary_of_singleton:
  assumes tX: topspace X = {a} and sc: singular_chain p X c
  shows chain_boundary p c =
         (if p = 0 ∨ odd p then 0
         else frag_extend (λf. frag_of(restrict (λx. a) (standard_simplex (p −1))))
c)
    (is ?lhs = ?rhs)
proof (cases p = 0)
  case False
   have ?lhs = frag_extend (λf. if odd p then 0 else frag_of(restrict (λx. a)
(standard_simplex (p −1)))) c
  proof (simp only: chain_boundary_def False if_False, rule frag_extend_eq)
    fix f
    assume f ∈ Poly_Mapping.keys c
    with assms have singular_simplex p X f
      by (auto simp: singular_chain_def)
    then have ∗: ⋀k. k ≤ p ⟹ singular_face p k f = (λx∈standard_simplex (p
−1). a)
      using False singular_simplex_singular_face
      by (fastforce simp flip: singular_simplex_singleton [OF tX])
    define c where c ≡ frag_of (λx∈standard_simplex (p −1). a)
    have (∑ k≤p. frag_cmul ((−1) ^ k) (frag_of (singular_face p k f)))
        = (∑ k≤p. frag_cmul ((−1) ^ k) c)
      by (auto simp: c_def ∗ intro: sum.cong)
    also have . . . = (if odd p then 0 else c)
      by (induction p) (auto simp: c_def restrict_def)
    finally show (∑ k≤p. frag_cmul ((−1) ^ k) (frag_of (singular_face p k f)))
                = (if odd p then 0 else frag_of (λx∈standard_simplex (p −1). a))
      unfolding c_def .
  qed
  also have . . . = ?rhs
```

```
    by (auto simp: False frag_extend_eq_0)
  finally show ?thesis .
qed (simp add: chain_boundary_def)


lemma singular_cycle_singleton:
  assumes topspace X = {a}
  shows singular_relcycle p X {} c ⟷ singular_chain p X c ∧ (p = 0 ∨ odd p
∨ c = 0)
proof −
  have c = 0 if singular_chain p X c and chain_boundary p c = 0 and even p
and p ≠ 0
   using that assms singular_chain_singleton [of X a p c] chain_boundary_of_singleton
[OF assms]
    by (auto simp: frag_extend_cmul)
  moreover
  have chain_boundary p c = 0 if sc: singular_chain p X c and odd p
    by (simp add: chain_boundary_of_singleton [OF assms sc] that)
  moreover have chain_boundary 0 c = 0 if singular_chain 0 X c and p = 0
    by (simp add: chain_boundary_def)
  ultimately show ?thesis
  using assms by (auto simp: singular_cycle)
qed


lemma singular_boundary_singleton:
  assumes topspace X = {a}
  shows singular_relboundary p X {} c ⟷ singular_chain p X c ∧ (odd p ∨ c
= 0)
proof (cases singular_chain p X c)
  case True
  have ∃ d. singular_chain (Suc p) X d ∧ chain_boundary (Suc p) d = c
    if singular_chain p X c and odd p
  proof −
   obtain b where b: c = frag_cmul b (frag_of(restrict (λx. a) (standard_simplex
p)))
      by (metis True assms singular_chain_singleton)
    let ?d = frag_cmul b (frag_of (λx∈standard_simplex (Suc p). a))
    have scd: singular_chain (Suc p) X ?d
      by (metis assms singular_chain_singleton)
    moreover have chain_boundary (Suc p) ?d = c
        by (simp add: assms scd chain_boundary_of_singleton [of X a Suc p] b
frag_extend_cmul ‹odd p›)
    ultimately show ?thesis
      by metis
  qed
  with True assms show ?thesis
    by (auto simp: singular_boundary chain_boundary_of_singleton)
next
```

case *False*
with *assms singular_boundary_imp_chain* **show** *?thesis*
by *metis*
qed

lemma *singular_boundary_eq_cycle_singleton*:
assumes *topspace X* = {*a*} *1* ≤ *p*
shows *singular_relboundary p X* {} *c* ⟷ *singular_relcycle p X* {} *c* (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
by (*simp add*: *singular_relboundary_imp_relcycle*)
show *?rhs* ⟹ *?lhs*
by (*metis assms not_one_le_zero singular_boundary_singleton singular_cycle_singleton*)
qed

lemma *singular_boundary_set_eq_cycle_singleton*:
assumes *topspace X* = {*a*} *1* ≤ *p*
shows *singular_relboundary_set p X* {} = *singular_relcycle_set p X* {}
using *singular_boundary_eq_cycle_singleton* [*OF assms*]
by *blast*

0.1.13 Simplicial chains

Simplicial chains, effectively those resulting from linear maps. We still allow the map to be singular, so the name is questionable. These are intended as building-blocks for singular subdivision, rather than as a axis for 1 simplicial homology.

definition *oriented_simplex*
where *oriented_simplex p l* ≡ (λ*x*∈*standard_simplex p*. λ*i*. (∑*j*≤*p*. *l j i* ∗ *x j*))

definition *simplicial_simplex*
where
simplicial_simplex p S f ≡
singular_simplex p (*subtopology* (*powertop_real UNIV*) *S*) *f* ∧
(∃ *l*. *f* = *oriented_simplex p l*)

lemma *simplicial_simplex*:
simplicial_simplex p S f ⟷ *f* ‘ (*standard_simplex p*) ⊆ *S* ∧ (∃ *l*. *f* = *oriented_simplex p l*)
(**is** *?lhs* = *?rhs*)
proof
assume *R*: *?rhs*
have *continuous_map* (*subtopology* (*powertop_real UNIV*) (*standard_simplex p*))
(*powertop_real UNIV*) (λ*x i*. ∑*j*≤*p*. *l j i* ∗ *x j*) **for** *l* :: *nat* ⇒ *'a* ⇒ *real*

unfolding *continuous_map_componentwise*
by (*force intro*: *continuous_intros continuous_map_from_subtopology continuous_map_product_projection*)
with *R* **show** *?lhs*
unfolding *simplicial_simplex_def singular_simplex_subtopology*
by (*auto simp add*: *singular_simplex_def oriented_simplex_def*)
qed (*simp add*: *simplicial_simplex_def singular_simplex_subtopology*)

lemma *simplicial_simplex_empty* [*simp*]: ¬ *simplicial_simplex p* {} *f*
by (*simp add*: *nonempty_standard_simplex simplicial_simplex*)

definition *simplicial_chain*
where *simplicial_chain p S c* ≡ *Poly_Mapping.keys c* ⊆ *Collect* (*simplicial_simplex p S*)

lemma *simplicial_chain_0* [*simp*]: *simplicial_chain p S 0*
by (*simp add*: *simplicial_chain_def*)

lemma *simplicial_chain_of* [*simp*]:
simplicial_chain p S (*frag_of c*) ⟷ *simplicial_simplex p S c*
by (*simp add*: *simplicial_chain_def*)

lemma *simplicial_chain_cmul*:
simplicial_chain p S c ⟹ *simplicial_chain p S* (*frag_cmul a c*)
by (*auto simp*: *simplicial_chain_def*)

lemma *simplicial_chain_diff*:
⟦*simplicial_chain p S c1*; *simplicial_chain p S c2*⟧ ⟹ *simplicial_chain p S* (*c1* − *c2*)
unfolding *simplicial_chain_def* **by** (*meson UnE keys_diff subset_iff*)

lemma *simplicial_chain_sum*:
(⋀*i*. *i* ∈ *I* ⟹ *simplicial_chain p S* (*f i*)) ⟹ *simplicial_chain p S* (*sum f I*)
unfolding *simplicial_chain_def*
using *order_trans* [*OF keys_sum* [*of f I*]]
by (*simp add*: *UN_least*)

lemma *simplicial_simplex_oriented_simplex*:
simplicial_simplex p S (*oriented_simplex p l*)
⟷ ((*λx i*. ∑*j*≤*p*. *l j i* ∗ *x j*) ‘ *standard_simplex p* ⊆ *S*)
by (*auto simp*: *simplicial_simplex oriented_simplex_def*)

lemma *simplicial_imp_singular_simplex*:
simplicial_simplex p S f
⟹ *singular_simplex p* (*subtopology* (*powertop_real UNIV*) *S*) *f*
by (*simp add*: *simplicial_simplex_def*)

lemma *simplicial_imp_singular_chain*:
simplicial_chain p S c

```
    ⟹ singular_chain p (subtopology (powertop_real UNIV) S) c
  unfolding simplicial_chain_def singular_chain_def
  by (auto intro: simplicial_imp_singular_simplex)

lemma oriented_simplex_eq:
  oriented_simplex p l = oriented_simplex p l' ⟷ (∀ i. i ≤ p ⟶ l i = l' i)
  (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof clarify
    fix i
    assume i ≤ p
    let ?fi = (λj. if j = i then 1 else 0)
    have (∑j≤p. l j k ∗ ?fi j) = (∑j≤p. l' j k ∗ ?fi j) for k
      using L ‹i ≤ p›
      by (simp add: fun_eq_iff oriented_simplex_def split: if_split_asm)
    with ‹i ≤ p› show l i = l' i
      by (simp add: if_distrib ext cong: if_cong)
  qed
qed (auto simp: oriented_simplex_def)

lemma singular_face_oriented_simplex:
  assumes 1 ≤ p k ≤ p
  shows singular_face p k (oriented_simplex p l) =
        oriented_simplex (p −1) (λj. if j < k then l j else l (Suc j))
proof −
  have (∑j≤p. l j i ∗ simplical_face k x j)
    = (∑j≤p − Suc 0. (if j < k then l j else l (Suc j)) i ∗ x j)
    if x ∈ standard_simplex (p − Suc 0) for i x
  proof −
    show ?thesis
      unfolding simplical_face_def
      using sum.zero_middle [OF assms, where 'a=real, symmetric]
     by (simp add: if_distrib [of λx. _ ∗ x] if_distrib [of λf. f i ∗ _] atLeast0AtMost
cong: if_cong)
  qed
  then show ?thesis
    using simplical_face_in_standard_simplex assms
    by (auto simp: singular_face_def oriented_simplex_def restrict_def)
qed

lemma simplicial_simplex_singular_face:
  fixes f :: (nat ⇒ real) ⇒ nat ⇒ real
  assumes ss: simplicial_simplex p S f and p: 1 ≤ p k ≤ p
  shows simplicial_simplex (p − Suc 0) S (singular_face p k f)
proof −
  let ?X = subtopology (powertop_real UNIV) S
  obtain m where l: singular_simplex p ?X (oriented_simplex p m)
```

```
      and feq: f = oriented_simplex p m
    using assms by (force simp: simplicial_simplex_def)
  moreover
  have singular_face p k f = oriented_simplex (p − Suc 0) (λi. if i < k then m i
else m (Suc i))
    unfolding feq singular_face_def oriented_simplex_def
   apply (simp add: simplical_face_in_standard_simplex [OF p] restrict_compose_left
subset_eq)
    using sum.zero_middle [OF p, where 'a=real, symmetric]  unfolding simpli-
cal_face_def o_def
   apply (simp add: if_distrib [of λx. _ ∗ x] if_distrib [of λf. f _ ∗ _] atLeast0At-
Most cong: if_cong)
    done
  ultimately
  show ?thesis
    using p simplicial_simplex_def singular_simplex_singular_face by blast
qed

lemma simplicial_chain_boundary:
   simplicial_chain p S c ⟹ simplicial_chain (p −1) S (chain_boundary p c)
  unfolding simplicial_chain_def
proof (induction rule: frag_induction)
  case (one f)
  then have simplicial_simplex p S f
    by simp
  have simplicial_chain (p − Suc 0) S (frag_of (singular_face p i f))
    if 0 < p i ≤ p for i
    using that one
    by (force simp: simplicial_simplex_def singular_simplex_singular_face singu-
lar_face_oriented_simplex)
  then have simplicial_chain (p − Suc 0) S (chain_boundary p (frag_of f))
    unfolding chain_boundary_def frag_extend_of
    by (auto intro!: simplicial_chain_cmul simplicial_chain_sum)
  then show ?case
    by (simp add: simplicial_chain_def [symmetric])
next
  case (diff a b)
  then show ?case
    by (metis chain_boundary_diff simplicial_chain_def simplicial_chain_diff)
qed auto
```

0.1.14 The cone construction on simplicial simplices.

```
consts simplex_cone :: [nat, nat ⇒ real, [nat ⇒ real, nat] ⇒ real, nat ⇒ real,
nat] ⇒ real
specification (simplex_cone)
  simplex_cone:
    ⋀p v l. simplex_cone p v (oriented_simplex p l) =
          oriented_simplex (Suc p) (λi. if i = 0 then v else l(i −1))
```

proof −
have ∗: $\bigwedge x.\ \exists y.\ \forall v.\ (\lambda l.\ oriented\_simplex\ (Suc\ x)\ (\lambda i.\ if\ i = 0\ then\ v\ else\ l\ (i - 1)))$
$= (y\ v \circ (oriented\_simplex\ x))$
apply (*subst choice_iff* [*symmetric*])
by (*simp add*: *oriented_simplex_eq choice_iff* [*symmetric*] *function_factors_left* [*symmetric*])
then show *?thesis*
unfolding *o_def* **by** (*metis*(*no_types*))
qed

lemma *simplicial_simplex_simplex_cone*:
assumes *f*: *simplicial_simplex p S f*
and *T*: $\bigwedge x\ u.\ [\![0 \le u;\ u \le 1;\ x \in S]\!] \Longrightarrow (\lambda i.\ (1 - u) * v\ i + u * x\ i) \in T$
shows *simplicial_simplex* (*Suc p*) *T* (*simplex_cone p v f*)
proof −
obtain *l* **where** *l*: $\bigwedge x.\ x \in standard\_simplex\ p \Longrightarrow oriented\_simplex\ p\ l\ x \in S$
and *feq*: $f = oriented\_simplex\ p\ l$
using *f* **by** (*auto simp*: *simplicial_simplex*)
have $oriented\_simplex\ p\ l\ x \in S$ **if** $x \in standard\_simplex\ p$ **for** *x*
using *f that* **by** (*auto simp*: *simplicial_simplex feq*)
then have *S*: $\bigwedge x.\ [\![\bigwedge i.\ 0 \le x\ i \wedge x\ i \le 1;\ \bigwedge i.\ i>p \Longrightarrow x\ i = 0;\ sum\ x\ \{..p\} = 1]\!]$
$\Longrightarrow (\lambda i.\ \sum j \le p.\ l\ j\ i * x\ j) \in S$
by (*simp add*: *oriented_simplex_def standard_simplex_def*)
have $oriented\_simplex\ (Suc\ p)\ (\lambda i.\ if\ i = 0\ then\ v\ else\ l\ (i - 1))\ x \in T$
if $x \in standard\_simplex\ (Suc\ p)$ **for** *x*
proof (*simp add*: *that oriented_simplex_def sum.atMost_Suc_shift del*: *sum.atMost_Suc*)
have *x01*: $\bigwedge i.\ 0 \le x\ i \wedge x\ i \le 1$ **and** *x0*: $\bigwedge i.\ i > Suc\ p \Longrightarrow x\ i = 0$ **and** *x1*: $sum\ x\ \{..Suc\ p\} = 1$
using *that* **by** (*auto simp*: *oriented_simplex_def standard_simplex_def*)
obtain *a* **where** $a \in S$
using *f* **by** *force*
show $(\lambda i.\ v\ i * x\ 0 + (\sum j \le p.\ l\ j\ i * x\ (Suc\ j))) \in T$
proof $(cases\ x\ 0 = 1)$
case *True*
then have $sum\ x\ \{Suc\ 0..Suc\ p\} = 0$
using *x1* **by** (*simp add*: *atMost_atLeast0 sum.atLeast_Suc_atMost*)
then have [*simp*]: $x\ (Suc\ j) = 0$ **if** $j \le p$ **for** *j*
unfolding *sum.atLeast_Suc_atMost_Suc_shift*
using *x01 that* **by** (*simp add*: *sum_nonneg_eq_0_iff*)
then show *?thesis*
using *T* [*of 0 a*] ‹$a \in S$› **by** (*auto simp*: *True*)
next
case *False*
then have $(\lambda i.\ v\ i * x\ 0 + (\sum j \le p.\ l\ j\ i * x\ (Suc\ j))) = (\lambda i.\ (1 - (1 - x\ 0)) * v\ i + (1 - x\ 0) * (inverse\ (1 - x\ 0) * (\sum j \le p.\ l\ j\ i * x\ (Suc\ j))))$
by (*force simp*: *field_simps*)
also have $\ldots \in T$

```
      proof (rule T)
        have x 0 < 1
          by (simp add: False less_le x01)
        have xle: x (Suc i) ≤ (1 − x 0) for i
        proof (cases i ≤ p)
          case True
          have sum x {0, Suc i} ≤ sum x {..Suc p}
            by (rule sum_mono2) (auto simp: True x01)
          then show ?thesis
           using x1 x01 by (simp add: algebra_simps not_less)
        qed (simp add: x0 x01)
        have (λi. (∑j≤p. l j i ∗ (x (Suc j) ∗ inverse (1 − x 0)))) ∈ S
        proof (rule S)
          have x 0 + (∑j≤p. x (Suc j)) = sum x {..Suc p}
            by (metis sum.atMost_Suc_shift)
          with x1 have (∑j≤p. x (Suc j)) = 1 − x 0
            by simp
          with False show (∑j≤p. x (Suc j) ∗ inverse (1 − x 0)) = 1
              by (metis add_diff_cancel_left' diff_diff_eq2 diff_zero right_inverse
sum_distrib_right)
      qed (use x01 x0 xle ‹x 0 < 1› in ‹auto simp: field_split_simps›)
      then show (λi. inverse (1 − x 0) ∗ (∑j≤p. l j i ∗ x (Suc j))) ∈ S
        by (simp add: field_simps sum_divide_distrib)
    qed (use x01 in auto)
    finally show ?thesis .
  qed
qed
  then show ?thesis
    by (auto simp: simplicial_simplex feq  simplex_cone)
qed

definition simplicial_cone
  where simplicial_cone p v ≡ frag_extend (frag_of ∘ simplex_cone p v)

lemma simplicial_chain_simplicial_cone:
  assumes c: simplicial_chain p S c
    and T: ⋀x u. ⟦0 ≤ u; u ≤ 1; x ∈ S⟧ ⟹ (λi. (1 − u) ∗ v i + u ∗ x i) ∈ T
  shows simplicial_chain (Suc p) T (simplicial_cone p v c)
  using c unfolding simplicial_chain_def simplicial_cone_def
proof (induction rule: frag_induction)
  case (one x)
  then show ?case
    by (simp add: T simplicial_simplex_simplex_cone)
next
  case (diff a b)
  then show ?case
    by (metis frag_extend_diff simplicial_chain_def simplicial_chain_diff)
qed auto
```

```
lemma chain_boundary_simplicial_cone_of':
  assumes f = oriented_simplex p l
  shows chain_boundary (Suc p) (simplicial_cone p v (frag_of f)) =
        frag_of f
        − (if p = 0 then frag_of (λu∈standard_simplex p. v)
           else simplicial_cone (p −1) v (chain_boundary p (frag_of f)))
proof (simp, intro impI conjI)
  assume p = 0
  have eq: (oriented_simplex 0 (λj. if j = 0 then v else l j)) = (λu∈standard_simplex
0. v)
    by (force simp: oriented_simplex_def standard_simplex_def)
  show chain_boundary (Suc 0) (simplicial_cone 0 v (frag_of f))
      = frag_of f − frag_of (λu∈standard_simplex 0. v)
    by (simp add: assms simplicial_cone_def chain_boundary_of ‹p = 0› sim-
plex_cone singular_face_oriented_simplex eq cong: if_cong)
next
  assume 0 < p
  have 0: simplex_cone (p − Suc 0) v (singular_face p x (oriented_simplex p l))
        = oriented_simplex p
             (λj. if j < Suc x
                  then if j = 0 then v else l (j −1)
                  else if Suc j = 0 then v else l (Suc j −1)) if x ≤ p for x
    using ‹0 < p› that
    by (auto simp: Suc_leI singular_face_oriented_simplex simplex_cone ori-
ented_simplex_eq)
  have 1: frag_extend (frag_of ∘ simplex_cone (p − Suc 0) v)
                     (∑k = 0..p. frag_cmul ((−1) ^ k) (frag_of (singular_face p k
(oriented_simplex p l))))
        = − (∑k = Suc 0..Suc p. frag_cmul ((−1) ^ k)
             (frag_of (singular_face (Suc p) k (simplex_cone p v (oriented_simplex
p l)))))
    unfolding sum.atLeast_Suc_atMost_Suc_shift
    by (auto simp: 0 simplex_cone singular_face_oriented_simplex frag_extend_sum
frag_extend_cmul simp flip: sum_negf)
  moreover have 2: singular_face (Suc p) 0 (simplex_cone p v (oriented_simplex
p l))
                    = oriented_simplex p l
    by (simp add: simplex_cone singular_face_oriented_simplex)
  show chain_boundary (Suc p) (simplicial_cone p v (frag_of f))
       = frag_of f − simplicial_cone (p − Suc 0) v (chain_boundary p (frag_of
f))
    using ‹p > 0›
    apply (simp add: assms simplicial_cone_def chain_boundary_of atMost_atLeast0
del: sum.atMost_Suc)
    apply (subst sum.atLeast_Suc_atMost [of 0])
    apply (simp_all add: 1 2 del: sum.atMost_Suc)
    done
qed
```

```
lemma chain_boundary_simplicial_cone_of:
  assumes simplicial_simplex p S f
  shows chain_boundary (Suc p) (simplicial_cone p v (frag_of f)) =
        frag_of f
        − (if p = 0 then frag_of (λu∈standard_simplex p. v)
           else simplicial_cone (p −1) v (chain_boundary p (frag_of f)))
  using chain_boundary_simplicial_cone_of' assms unfolding simplicial_simplex_def
  by blast

lemma chain_boundary_simplicial_cone:
  simplicial_chain p S c
   ⟹ chain_boundary (Suc p) (simplicial_cone p v c) =
       c − (if p = 0 then frag_extend (λf. frag_of (λu∈standard_simplex p. v)) c
           else simplicial_cone (p −1) v (chain_boundary p c))
  unfolding simplicial_chain_def
proof (induction rule: frag_induction)
  case (one x)
  then show ?case
    by (auto simp: chain_boundary_simplicial_cone_of)
qed (auto simp: chain_boundary_diff simplicial_cone_def frag_extend_diff)

lemma simplex_map_oriented_simplex:
  assumes l: simplicial_simplex p (standard_simplex q) (oriented_simplex p l)
    and g: simplicial_simplex r S g and q ≤ r
  shows simplex_map p g (oriented_simplex p l) = oriented_simplex p (g ∘ l)
proof −
  obtain m where geq: g = oriented_simplex r m
    using g by (auto simp: simplicial_simplex_def)
  have g (λi. ∑j≤p. l j i ∗ x j) i = (∑j≤p. g (l j) i ∗ x j)
    if x ∈ standard_simplex p for x i
  proof −
    have ssr: (λi. ∑j≤p. l j i ∗ x j) ∈ standard_simplex r
      using l that standard_simplex_mono [OF ‹q ≤ r›]
      unfolding simplicial_simplex_oriented_simplex by auto
    have lss: l j ∈ standard_simplex r if j≤p for j
    proof −
      have q: (λx i. ∑j≤p. l j i ∗ x j) ‘ standard_simplex p ⊆ standard_simplex q
        using l by (simp add: simplicial_simplex_oriented_simplex)
      let ?x = (λi. if i = j then 1 else 0)
      have p: l j ∈ (λx i. ∑j≤p. l j i ∗ x j) ‘ standard_simplex p
      proof
        show l j = (λi. ∑j≤p. l j i ∗ ?x j)
          using ‹j≤p› by (force simp: if_distrib cong: if_cong)
        show ?x ∈ standard_simplex p
          by (simp add: that)
      qed
      show ?thesis
        using standard_simplex_mono [OF ‹q ≤ r›] q p
```

```
        by blast
    qed
    have g (λi. ∑j≤p. l j i ∗ x j) i = (∑j≤r. ∑n≤p. m j i ∗ (l n j ∗ x n))
      by (simp add: geq oriented_simplex_def sum_distrib_left ssr)
    also have ... =  (∑j≤p. ∑n≤r. m n i ∗ (l j n ∗ x j))
      by (rule sum.swap)
    also have ... = (∑j≤p. g (l j) i ∗ x j)
      by (simp add: geq oriented_simplex_def sum_distrib_right mult.assoc lss)
    finally show ?thesis .
  qed
  then show ?thesis
    by (force simp: oriented_simplex_def simplex_map_def o_def)
qed


lemma chain_map_simplicial_cone:
  assumes g: simplicial_simplex r S g
      and c: simplicial_chain p (standard_simplex q) c
      and v: v ∈ standard_simplex q and q ≤ r
  shows chain_map (Suc p) g (simplicial_cone p v c) = simplicial_cone p (g v)
(chain_map p g c)
proof −
  have ∗: simplex_map (Suc p) g (simplex_cone p v f) = simplex_cone p (g v)
(simplex_map p g f)
    if f ∈ Poly_Mapping.keys c for f
  proof −
    have simplicial_simplex p (standard_simplex q) f
      using c that by (auto simp: simplicial_chain_def)
    then obtain m where feq: f = oriented_simplex p m
      by (auto simp: simplicial_simplex)
    have 0: simplicial_simplex p (standard_simplex q) (oriented_simplex p m)
      using ‹simplicial_simplex p (standard_simplex q) f› feq by blast
    then have 1: simplicial_simplex (Suc p) (standard_simplex q)
                     (oriented_simplex (Suc p) (λi. if i = 0 then v else m (i −1)))
      using convex_standard_simplex v
      by (simp flip: simplex_cone add: simplicial_simplex_simplex_cone)
    show ?thesis
      using simplex_map_oriented_simplex [OF 1 g ‹q ≤ r›]
            simplex_map_oriented_simplex [of p q m r S g, OF 0 g ‹q ≤ r›]
      by (simp add: feq oriented_simplex_eq simplex_cone)
  qed
  show ?thesis
     by (auto simp: chain_map_def simplicial_cone_def frag_extend_compose ∗
intro: frag_extend_eq)
qed
```

0.1.15 Barycentric subdivision of a linear ("simplicial") simplex's image

```
definition simplicial_vertex
  where simplicial_vertex i f = f(λj. if j = i then 1 else 0)

lemma simplicial_vertex_oriented_simplex:
   simplicial_vertex i (oriented_simplex p l) = (if i ≤ p then l i else undefined)
  by (simp add: simplicial_vertex_def oriented_simplex_def if_distrib cong: if_cong)


primrec simplicial_subdivision
where
  simplicial_subdivision 0 = id
| simplicial_subdivision (Suc p) =
     frag_extend
      (λf. simplicial_cone p
           (λi. (∑j≤Suc p. simplicial_vertex j f i) / (p + 2))
           (simplicial_subdivision p (chain_boundary (Suc p) (frag_of f))))


lemma simplicial_subdivision_0 [simp]:
   simplicial_subdivision p 0 = 0
  by (induction p) auto

lemma simplicial_subdivision_diff:
  simplicial_subdivision p (c1−c2) = simplicial_subdivision p c1 − simplicial_subdivision
p c2
  by (induction p) (auto simp: frag_extend_diff)

lemma simplicial_subdivision_of:
   simplicial_subdivision p (frag_of f) =
        (if p = 0 then frag_of f
        else simplicial_cone (p −1)
              (λi. (∑j≤p. simplicial_vertex j f i) / (Suc p))
              (simplicial_subdivision (p −1) (chain_boundary p (frag_of f))))
  by (induction p) (auto simp: add.commute)


lemma simplicial_chain_simplicial_subdivision:
   simplicial_chain p S c
          ⟹ simplicial_chain p S (simplicial_subdivision p c)
proof (induction p arbitrary: S c)
  case (Suc p)
  show ?case
    using Suc.prems [unfolded simplicial_chain_def]
  proof (induction c rule: frag_induction)
    case (one f)
    then have f: simplicial_simplex (Suc p) S f
      by auto
```

then have *simplicial_chain p (f ‘ standard_simplex (Suc p))*
(simplicial_subdivision p (chain_boundary (Suc p) (frag_of f)))
by (*metis Suc.IH diff_Suc_1 simplicial_chain_boundary simplicial_chain_of simplicial_simplex subsetI*)
moreover
obtain *l* **where** *l*: $\bigwedge x.\ x \in$ *standard_simplex (Suc p)* $\Longrightarrow$ ($\lambda i.$ ($\sum j\leq$*Suc p. l j i* $*$ *x j*)) $\in S$
and *feq*: *f* = *oriented_simplex (Suc p) l*
using *f* **by** (*fastforce simp*: *simplicial_simplex oriented_simplex_def simp del*: *sum.atMost_Suc*)
have ($\lambda i.$ (*1* − *u*) $*$ (($\sum j\leq$*Suc p. simplicial_vertex j f i*) / (*real p* + *2*)) + *u* $*$ *y i*) $\in S$
if *0* ≤ *u u* ≤ *1* **and** *y*: *y* ∈ *f ‘ standard_simplex (Suc p)* **for** *y u*
proof −
obtain *x* **where** *x*: *x* ∈ *standard_simplex (Suc p)* **and** *yeq*: *y* = *oriented_simplex (Suc p) l x*
using *y feq* **by** *blast*
have ($\lambda i.$ $\sum j\leq$*Suc p. l j i* $*$ ((*if j* ≤ *Suc p then* (*1* − *u*) $*$ *inverse* (*p* + *2*) + *u* $*$ *x j else 0*))) $\in S$
proof (*rule l*)
have *inverse* (*2* + *real p*) ≤ *1* (*2* + *real p*) $*$ ((*1* − *u*) $*$ *inverse* (*2* + *real p*)) + *u* = *1*
by (*auto simp add*: *field_split_simps*)
then show ($\lambda j.$ *if j* ≤ *Suc p then* (*1* − *u*) $*$ *inverse* (*real* (*p* + *2*)) + *u* $*$ *x j else 0*) ∈ *standard_simplex (Suc p)*
using *x* ‹*0* ≤ *u*› ‹*u* ≤ *1*›
by (*simp add*: *sum.distrib standard_simplex_def linepath_le_1 flip*: *sum_distrib_left del*: *sum.atMost_Suc*)
qed
moreover have ($\lambda i.$ $\sum j\leq$*Suc p. l j i* $*$ ((*1* − *u*) $*$ *inverse* (*2* + *real p*) + *u* $*$ *x j*))
= ($\lambda i.$ (*1* − *u*) $*$ ($\sum j\leq$*Suc p. l j i*) / (*real p* + *2*) + *u* $*$ ($\sum j\leq$*Suc p. l j i* $*$ *x j*))
proof
fix *i*
have ($\sum j\leq$*Suc p. l j i* $*$ ((*1* − *u*) $*$ *inverse* (*2* + *real p*) + *u* $*$ *x j*))
= ($\sum j\leq$*Suc p.* (*1* − *u*) $*$ *l j i* / (*real p* + *2*) + *u* $*$ *l j i* $*$ *x j*) (**is** *?lhs* = _)
by (*simp add*: *field_simps cong*: *sum.cong*)
also have . . . = (*1* − *u*) $*$ ($\sum j\leq$*Suc p. l j i*) / (*real p* + *2*) + *u* $*$ ($\sum j\leq$*Suc p. l j i* $*$ *x j*) (**is** _ = *?rhs*)
by (*simp add*: *sum_distrib_left sum.distrib sum_divide_distrib mult.assoc del*: *sum.atMost_Suc*)
finally show *?lhs* = *?rhs* **.**
qed
ultimately show *?thesis*
using *feq x yeq*
by (*simp add*: *simplicial_vertex_oriented_simplex*) (*simp add*: *oriented_simplex_def*)
qed

ultimately show *?case*
by (*simp add*: *simplicial_chain_simplicial_cone*)
next
case (*diff a b*)
then show *?case*
by (*metis simplicial_chain_diff simplicial_subdivision_diff*)
qed *auto*
qed *auto*

lemma *chain_boundary_simplicial_subdivision*:
simplicial_chain p S c
⟹ *chain_boundary p* (*simplicial_subdivision p c*) = *simplicial_subdivision* (*p* −*1*) (*chain_boundary p c*)
proof (*induction p arbitrary*: *c*)
case (*Suc p*)
show *?case*
using *Suc.prems* [*unfolded simplicial_chain_def*]
proof (*induction c rule*: *frag_induction*)
case (*one f*)
then have *f*: *simplicial_simplex* (*Suc p*) *S f*
by *simp*
then have *simplicial_chain p S* (*simplicial_subdivision p* (*chain_boundary* (*Suc p*) (*frag_of f*)))
by (*metis diff_Suc_1 simplicial_chain_boundary simplicial_chain_of simplicial_chain_simplicial_subdivision*)
moreover have *simplicial_chain p S* (*chain_boundary* (*Suc p*) (*frag_of f*))
using *one simplicial_chain_boundary simplicial_chain_of* **by** *fastforce*
moreover have *simplicial_subdivision* (*p* − *Suc 0*) (*chain_boundary p* (*chain_boundary* (*Suc p*) (*frag_of f*))) = *0*
by (*metis f chain_boundary_boundary_alt simplicial_simplex_def simplicial_subdivision_0 singular_chain_of*)
ultimately show *?case*
using *chain_boundary_simplicial_cone Suc*
by (*auto simp*: *chain_boundary_of frag_extend_diff simplicial_cone_def*)
next
case (*diff a b*)
then show *?case*
by (*simp add*: *simplicial_subdivision_diff chain_boundary_diff frag_extend_diff*)
qed *auto*
qed *auto*

A MESS AND USED ONLY ONCE

lemma *simplicial_subdivision_shrinks*:
⟦*simplicial_chain p S c*;
⋀*f x y*. ⟦*f* ∈ *Poly_Mapping.keys c*; *x* ∈ *standard_simplex p*; *y* ∈ *standard_simplex p*⟧ ⟹ |*f x k* − *f y k*| ≤ *d*;
f ∈ *Poly_Mapping.keys*(*simplicial_subdivision p c*);
x ∈ *standard_simplex p*; *y* ∈ *standard_simplex p*⟧
⟹ |*f x k* − *f y k*| ≤ (*p* / (*Suc p*)) ∗ *d*

```
proof (induction p arbitrary: d c f x y)
  case (Suc p)
  define Sigp where Sigp ≡ λf:: (nat ⇒ real) ⇒ nat ⇒ real. λi. (∑j≤Suc p.
simplicial_vertex j f i) / real (p + 2)
  let ?CB = λf. chain_boundary (Suc p) (frag_of f)
  have *: Poly_Mapping.keys
            (simplicial_cone p (Sigp f)
              (simplicial_subdivision p (?CB f)))
          ⊆ {f. ∀x∈standard_simplex (Suc p). ∀y∈standard_simplex (Suc p).
                  |f x k − f y k| ≤ real (Suc p) / (real p + 2) ∗ d} (is ?lhs ⊆ ?rhs)
    if f: f ∈ Poly_Mapping.keys c for f
  proof −
    have ssf: simplicial_simplex (Suc p) S f
      using Suc.prems(1) simplicial_chain_def that by auto
    have 2: ⋀x y. ⟦x ∈ standard_simplex (Suc p); y ∈ standard_simplex (Suc p)⟧
⟹ |f x k − f y k| ≤ d
     by (meson Suc.prems(2) f subsetD le_Suc_eq order_refl standard_simplex_mono)
    have sub: Poly_Mapping.keys ((frag_of ∘ simplex_cone p (Sigp f)) g) ⊆ ?rhs
      if g ∈ Poly_Mapping.keys (simplicial_subdivision p (?CB f)) for g
    proof −
      have 1: simplicial_chain p S (?CB f)
        using ssf simplicial_chain_boundary simplicial_chain_of by fastforce
      have simplicial_chain (Suc p) (f ‘ standard_simplex(Suc p)) (frag_of f)
        by (metis simplicial_chain_of simplicial_simplex ssf subset_refl)
      then have sc_sub: Poly_Mapping.keys (?CB f)
                        ⊆ Collect (simplicial_simplex p (f ‘ standard_simplex (Suc p)))
        by (metis diff_Suc_1 simplicial_chain_boundary simplicial_chain_def)
     have led: ⋀h x y. ⟦h ∈ Poly_Mapping.keys (chain_boundary (Suc p) (frag_of
f));
                            x ∈ standard_simplex p; y ∈ standard_simplex p⟧ ⟹ |h x k
− h y k| ≤ d
        using Suc.prems(2) f sc_sub
        by (simp add: simplicial_simplex subset_iff image_iff) metis
      have ⋀f' x y. ⟦f' ∈ Poly_Mapping.keys (simplicial_subdivision p (?CB f));
x ∈ standard_simplex p; y ∈ standard_simplex p⟧
            ⟹ |f' x k − f' y k| ≤ (p / (Suc p)) ∗ d
        by (blast intro: led Suc.IH [of chain_boundary (Suc p) (frag_of f), OF 1])
      then have g: ⋀x y. ⟦x ∈ standard_simplex p; y ∈ standard_simplex p⟧ ⟹
|g x k − g y k| ≤ (p / (Suc p)) ∗ d
        using that by blast
      have d ≥ 0
        using Suc.prems(2)[OF f] ‹x ∈ standard_simplex (Suc p)› by force
      have 3: simplex_cone p (Sigp f) g ∈ ?rhs
      proof −
        have simplicial_simplex p (f ‘ standard_simplex(Suc p)) g
          by (metis (mono_tags, opaque_lifting) sc_sub mem_Collect_eq simpli-
cial_chain_def simplicial_chain_simplicial_subdivision subsetD that)
        then obtain m where m: g ‘ standard_simplex p ⊆ f ‘ standard_simplex
(Suc p)
```

```
      and geq: g = oriented_simplex p m
      using ssf by (auto simp: simplicial_simplex)
    have m_in_gim: m i ∈ g ` standard_simplex p if i ≤ p for i
    proof
      show m i = g (λj. if j = i then 1 else 0)
        by (simp add: geq oriented_simplex_def that if_distrib cong: if_cong)
      show (λj. if j = i then 1 else 0) ∈ standard_simplex p
        by (simp add: oriented_simplex_def that)
    qed
    obtain l where l: f ` standard_simplex (Suc p) ⊆ S
      and feq: f = oriented_simplex (Suc p) l
      using ssf by (auto simp: simplicial_simplex)
    show ?thesis
    proof (clarsimp simp add: geq simp del: sum.atMost_Suc)
      fix x y
       assume x: x ∈ standard_simplex (Suc p) and y: y ∈ standard_simplex (Suc p)
     then have x': (∀ i. 0 ≤ x i ∧ x i ≤ 1) ∧ (∀ i>Suc p. x i = 0) ∧ (∑ i≤Suc p. x i) = 1
        and y': (∀ i. 0 ≤ y i ∧ y i ≤ 1) ∧ (∀ i>Suc p. y i = 0) ∧ (∑ i≤Suc p. y i) = 1
        by (auto simp: standard_simplex_def)
     have |(∑ j≤Suc p. (if j = 0 then λi. (∑ j≤Suc p. l j i) / (2 + real p) else m (j -1)) k * x j) -
          (∑ j≤Suc p. (if j = 0 then λi. (∑ j≤Suc p. l j i) / (2 + real p) else m (j -1)) k * y j)|
          ≤ (1 + real p) * d / (2 + real p)
      proof -
        have zero: |m (s - Suc 0) k - (∑ j≤Suc p. l j k) / (2 + real p)| ≤ (1 + real p) * d / (2 + real p)
          if 0 < s and s ≤ Suc p for s
        proof -
          have m (s - Suc 0) ∈ f ` standard_simplex (Suc p)
            using m m_in_gim that(2) by auto
           then obtain z where eq: m (s - Suc 0) = (λi. ∑ j≤Suc p. l j i * z j) and z: z ∈ standard_simplex (Suc p)
            using feq unfolding oriented_simplex_def by auto
          show ?thesis
            unfolding eq
          proof (rule convex_sum_bound_le)
            fix i
            assume i: i ∈ {..Suc p}
            then have [simp]: card ({..Suc p} - {i}) = Suc p
              by (simp add: card_Suc_Diff1)
           have (∑ j≤Suc p. |l i k / (p + 2) - l j k / (p + 2)|) = (∑ j≤Suc p. |l i k - l j k| / (p + 2))
              by (rule sum.cong) (simp_all add: flip: diff_divide_distrib)
            also have ... = (∑ j ∈ {..Suc p} - {i}. |l i k - l j k| / (p + 2))
              by (rule sum.mono_neutral_right) auto
```

```
                  also have ... ≤ (1 + real p) ∗ d / (p + 2)
                  proof (rule sum_bounded_above_divide)
                    fix i' :: nat
                    assume i': i' ∈ {..Suc p} − {i}
                    have lf: l r ∈ f ‘ standard_simplex(Suc p) if r ≤ Suc p for r
                    proof
                      show l r = f (λj. if j = r then 1 else 0)
                          using that by (simp add: feq oriented_simplex_def if_distrib
cong: if_cong)
                      show (λj. if j = r then 1 else 0) ∈ standard_simplex (Suc p)
                        by (auto simp: oriented_simplex_def that)
                    qed
                    show |l i k − l i' k| / real (p + 2) ≤ (1 + real p) ∗ d / real (p +
2) / real (card ({..Suc p} − {i}))
                      using i i' lf [of i] lf [of i'] 2
                      by (auto simp: image_iff divide_simps)
                  qed auto
                  finally have (∑j≤Suc p. |l i k / (p + 2) − l j k / (p + 2)|) ≤ (1
+ real p) ∗ d / (p + 2) .
                  then have |∑j≤Suc p. l i k / (p + 2) − l j k / (p + 2)| ≤ (1 +
real p) ∗ d / (p + 2)
                    by (rule order_trans [OF sum_abs])
                  then show |l i k − (∑j≤Suc p. l j k) / (2 + real p)| ≤ (1 + real p)
∗ d / (2 + real p)
                by (simp add: sum_subtractf sum_divide_distrib del: sum.atMost_Suc)
                qed (use standard_simplex_def z in auto)
              qed
              have nonz: |m (s − Suc 0) k − m (r − Suc 0) k| ≤ (1 + real p) ∗ d /
(2 + real p) (is ?lhs ≤ ?rhs)
                if r < s and 0 < r and r ≤ Suc p and s ≤ Suc p for r s
              proof −
                have ?lhs ≤ (p / (Suc p)) ∗ d
                  using m_in_gim [of r − Suc 0] m_in_gim [of s − Suc 0] that g by
fastforce
                also have ... ≤ ?rhs
                  by (simp add: field_simps ‹0 ≤ d›)
                finally show ?thesis .
              qed
              have jj: j ≤ Suc p ∧ j' ≤ Suc p
                    ⟶ |(if j' = 0 then λi. (∑j≤Suc p. l j i) / (2 + real p) else m (j'
−1)) k −
                      (if j = 0 then λi. (∑j≤Suc p. l j i) / (2 + real p) else m (j −1))
k|
                      ≤ (1 + real p) ∗ d / (2 + real p) for j j'
                using ‹0 ≤ d›
                by (rule_tac a=j and b = j' in linorder_less_wlog; force simp: zero
nonz simp del: sum.atMost_Suc)
              show ?thesis
                apply (rule convex_sum_bound_le)
```

```
            using x′ apply blast
            using x′ apply blast
            apply (subst abs_minus_commute)
            apply (rule convex_sum_bound_le)
            using y′ apply blast
            using y′ apply blast
            using jj by blast
          qed
            then show |simplex_cone p (Sigp f) (oriented_simplex p m) x k −
simplex_cone p (Sigp f) (oriented_simplex p m) y k|
              ≤ (1 + real p) ∗ d / (real p + 2)
              apply (simp add: feq Sigp_def simplicial_vertex_oriented_simplex
simplex_cone del: sum.atMost_Suc)
          apply (simp add: oriented_simplex_def algebra_simps x y del: sum.atMost_Suc)
            done
        qed
      qed
      show ?thesis
        using Suc.IH [OF 1, where f=g] 2 3 by simp
    qed
    then show ?thesis
      unfolding simplicial_chain_def simplicial_cone_def
      by (simp add: order_trans [OF keys_frag_extend] sub UN_subset_iff)
  qed
  show ?case
    using Suc
    apply (simp del: sum.atMost_Suc)
    apply (drule subsetD [OF keys_frag_extend])
    apply (simp del: sum.atMost_Suc)
    apply clarify
    apply (rename_tac FFF)
    using ∗
    apply (simp add: add.commute Sigp_def subset_iff)
    done
qed (auto simp: standard_simplex_0)
```

0.1.16 Singular subdivision

```
definition singular_subdivision
  where singular_subdivision p ≡
      frag_extend
         (λf. chain_map p f
               (simplicial_subdivision p
                     (frag_of(restrict id (standard_simplex p)))))

lemma singular_subdivision_0 [simp]: singular_subdivision p 0 = 0
  by (simp add: singular_subdivision_def)

lemma singular_subdivision_add:
```

singular_subdivision p (*a* + *b*) = *singular_subdivision p a* + *singular_subdivision p b*
by (*simp add*: *singular_subdivision_def frag_extend_add*)

lemma *singular_subdivision_diff*:
singular_subdivision p (*a* − *b*) = *singular_subdivision p a* − *singular_subdivision p b*
by (*simp add*: *singular_subdivision_def frag_extend_diff*)

lemma *simplicial_simplex_id* [*simp*]:
simplicial_simplex p S (*restrict id* (*standard_simplex p*)) ⟷ *standard_simplex p* ⊆ *S*
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*simp add*: *simplicial_simplex_def singular_simplex_def continuous_map_in_subtopology set_mp*)
next
assume *R*: *?rhs*
then have *cm*: *continuous_map*
(*subtopology* (*powertop_real UNIV*) (*standard_simplex p*))
(*subtopology* (*powertop_real UNIV*) *S*) *id*
using *continuous_map_from_subtopology_mono continuous_map_id* **by** *blast*
moreover have ∃ *l*. *restrict id* (*standard_simplex p*) = *oriented_simplex p l*
apply (*rule_tac x*=λ*i j*. *if i* = *j then 1 else 0* **in** *exI*)
apply (*force simp*: *oriented_simplex_def standard_simplex_def if_distrib* [*of* λ*u*. *u* ∗ _] *cong*: *if_cong*)
done
ultimately show *?lhs*
by (*simp add*: *simplicial_simplex_def singular_simplex_def*)
qed

lemma *singular_chain_singular_subdivision*:
singular_chain p X c
⟹ *singular_chain p X* (*singular_subdivision p c*)
unfolding *singular_subdivision_def*
apply (*rule singular_chain_extend*)
apply (*rule singular_chain_chain_map* [**where** *X* = *subtopology* (*powertop_real UNIV*)
(*standard_simplex p*)])
apply (*simp add*: *simplicial_chain_simplicial_subdivision simplicial_imp_singular_chain*)
by (*simp add*: *singular_chain_def singular_simplex_def subset_iff*)

lemma *naturality_singular_subdivision*:
singular_chain p X c
⟹ *singular_subdivision p* (*chain_map p g c*) = *chain_map p g* (*singular_subdivision p c*)
unfolding *singular_chain_def*

proof (*induction rule*: *frag_induction*)
  case (*one f*)
  then have *singular_simplex p X f*
    by *auto*
  have ⟦*simplicial_chain p* (*standard_simplex p*) *d*⟧
    ⟹ *chain_map p* (*simplex_map p g f*) *d* = *chain_map p g* (*chain_map p f d*)
for *d*
    unfolding *simplicial_chain_def*
  proof (*induction rule*: *frag_induction*)
    case (*one x*)
   then have *simplex_map p* (*simplex_map p g f*) *x* = *simplex_map p g* (*simplex_map p f x*)
      by (*force simp*: *simplex_map_def restrict_compose_left simplicial_simplex*)
    then show *?case*
      by *auto*
  qed (*auto simp*: *chain_map_diff*)
  then show *?case*
   using *simplicial_chain_simplicial_subdivision* [*of p standard_simplex p frag_of* (*restrict id* (*standard_simplex p*))]
    by (*simp add*: *singular_subdivision_def*)
next
  case (*diff a b*)
  then show *?case*
    by (*simp add*: *chain_map_diff singular_subdivision_diff*)
qed *auto*

lemma *simplicial_chain_chain_map*:
 assumes *f*: *simplicial_simplex q X f* **and** *c*: *simplicial_chain p* (*standard_simplex q*) *c*
  shows *simplicial_chain p X* (*chain_map p f c*)
  using *c* **unfolding** *simplicial_chain_def*
proof (*induction c rule*: *frag_induction*)
  case (*one g*)
  have ∃ *n*. *simplex_map p* (*oriented_simplex q l*)
               (*oriented_simplex p m*) = *oriented_simplex p n*
    if *m*: *singular_simplex p*
          (*subtopology* (*powertop_real UNIV*) (*standard_simplex q*)) (*oriented_simplex p m*)
    for *l m*
  proof −
    have (*λi*. ∑*j*≤*p*. *m j i* ∗ *x j*) ∈ *standard_simplex q*
      if *x* ∈ *standard_simplex p* **for** *x*
      using *that m* **unfolding** *oriented_simplex_def singular_simplex_def*
      by (*auto simp*: *continuous_map_in_subtopology image_subset_iff*)
    then show *?thesis*
      unfolding *oriented_simplex_def simplex_map_def*
      apply (*rule_tac x*=*λj k*. (∑ *i*≤*q*. *l i k* ∗ *m j i*) **in** *exI*)
        apply (*force simp*: *sum_distrib_left sum_distrib_right mult.assoc intro*: *sum.swap*)

```
      done
    qed
    then show ?case
      using f one
      apply (auto simp: simplicial_simplex_def)
      apply (rule singular_simplex_simplex_map
          [where X = subtopology (powertop_real UNIV) (standard_simplex q)])
      unfolding singular_simplex_def apply (fastforce simp add:)+
      done
next
  case (diff a b)
  then show ?case
    by (metis chain_map_diff simplicial_chain_def simplicial_chain_diff)
qed auto


lemma singular_subdivision_simplicial_simplex:
   simplicial_chain p S c
        ⟹ singular_subdivision p c = simplicial_subdivision p c
proof (induction p arbitrary: S c)
  case 0
  then show ?case
    unfolding simplicial_chain_def
  proof (induction rule: frag_induction)
    case (one x)
    then show ?case
      using singular_simplex_chain_map_id simplicial_imp_singular_simplex
      by (fastforce simp: singular_subdivision_def simplicial_subdivision_def)
  qed (auto simp: singular_subdivision_diff)
next
  case (Suc p)
  show ?case
    using Suc.prems unfolding simplicial_chain_def
  proof (induction rule: frag_induction)
    case (one f)
    then have ssf: simplicial_simplex (Suc p) S f
      by (auto simp: simplicial_simplex)
    then have 1: simplicial_chain p (standard_simplex (Suc p))
                  (simplicial_subdivision p
                    (chain_boundary (Suc p)
                      (frag_of (restrict id (standard_simplex (Suc p))))))
     by (metis diff_Suc_1 order_refl simplicial_chain_boundary simplicial_chain_of
simplicial_chain_simplicial_subdivision simplicial_simplex_id)
     have 2: (λi. (∑j≤Suc p. simplicial_vertex j (restrict id (standard_simplex
(Suc p))) i) / (real p + 2))
                 ∈ standard_simplex (Suc p)
     by (simp add: simplicial_vertex_def standard_simplex_def del: sum.atMost_Suc)
    have ss_Sp: (λi. (if i ≤ Suc p then 1 else 0) / (real p + 2)) ∈ standard_simplex
(Suc p)
```

by (*simp add*: *standard_simplex_def field_split_simps*)
obtain *l* **where** *feq*: *f* = *oriented_simplex* (*Suc p*) *l*
using *one* **unfolding** *simplicial_simplex* **by** *blast*
then have *3*: *f* (λ*i*. (∑*j*≤*Suc p*. *simplicial_vertex j* (*restrict id* (*standard_simplex* (*Suc p*))) *i*) / (*real p* + *2*))
= (λ*i*. (∑*j*≤*Suc p*. *simplicial_vertex j f i*) / (*real p* + *2*))
unfolding *simplicial_vertex_def oriented_simplex_def*
by (*simp add*: *ss_Sp if_distrib* [*of* λ*x*. _ ∗ *x*] *sum_divide_distrib del*: *sum.atMost_Suc cong*: *if_cong*)
have *scp*: *singular_chain* (*Suc p*)
(*subtopology* (*powertop_real UNIV*) (*standard_simplex* (*Suc p*)))
(*frag_of* (*restrict id* (*standard_simplex* (*Suc p*))))
by (*simp add*: *simplicial_imp_singular_chain*)
have *scps*: *simplicial_chain p* (*standard_simplex* (*Suc p*))
(*chain_boundary* (*Suc p*) (*frag_of* (*restrict id* (*standard_simplex* (*Suc p*)))))
by (*metis diff_Suc_1 order_refl simplicial_chain_boundary simplicial_chain_of simplicial_simplex_id*)
have *scpf*: *simplicial_chain p S*
(*chain_map p f*
(*chain_boundary* (*Suc p*) (*frag_of* (*restrict id* (*standard_simplex* (*Suc p*))))))
using *scps simplicial_chain_chain_map ssf* **by** *blast*
have *4*: *chain_map p f*
(*simplicial_subdivision p*
(*chain_boundary* (*Suc p*) (*frag_of* (*restrict id* (*standard_simplex* (*Suc p*))))))
= *simplicial_subdivision p* (*chain_boundary* (*Suc p*) (*frag_of f*))
apply (*simp add*: *chain_boundary_chain_map* [*OF scp*] *del*: *chain_map_of flip*: *singular_simplex_chain_map_id* [*OF simplicial_imp_singular_simplex* [*OF ssf*]])
by (*metis* (*no_types*) *scp singular_chain_boundary_alt Suc.IH* [*OF scps*] *Suc.IH* [*OF scpf*] *naturality_singular_subdivision*)
show *?case*
apply (*simp add*: *singular_subdivision_def del*: *sum.atMost_Suc*)
apply (*simp only*: *ssf 1 2 3 4 chain_map_simplicial_cone* [*of Suc p S* _ *p Suc p*])
done
qed (*auto simp*: *frag_extend_diff singular_subdivision_diff*)
qed

lemma *naturality_simplicial_subdivision*:
⟦*simplicial_chain p* (*standard_simplex q*) *c*; *simplicial_simplex q S g*⟧
⟹ *simplicial_subdivision p* (*chain_map p g c*) = *chain_map p g* (*simplicial_subdivision p c*)
apply (*simp flip*: *singular_subdivision_simplicial_simplex*)
by (*metis naturality_singular_subdivision simplicial_chain_chain_map simplicial_imp_singular_chain singular_subdivision_simplicial_simplex*)

lemma *chain_boundary_singular_subdivision*:
singular_chain p X c
⟹ *chain_boundary p* (*singular_subdivision p c*) =
singular_subdivision (*p* − *Suc 0*) (*chain_boundary p c*)
unfolding *singular_chain_def*
proof (*induction rule*: *frag_induction*)
case (*one f*)
then have *ssf*: *singular_simplex p X f*
by (*auto simp*: *singular_simplex_def*)
then have *scp*: *simplicial_chain p* (*standard_simplex p*) (*frag_of* (*restrict id* (*standard_simplex p*)))
by *simp*
have *scp1*: *simplicial_chain* (*p* − *Suc 0*) (*standard_simplex p*)
(*chain_boundary p* (*frag_of* (*restrict id* (*standard_simplex p*))))
using *simplicial_chain_boundary* **by** *force*
have *sgp1*: *singular_chain* (*p* − *Suc 0*)
(*subtopology* (*powertop_real UNIV*) (*standard_simplex p*))
(*chain_boundary p* (*frag_of* (*restrict id* (*standard_simplex p*))))
using *scp1 simplicial_imp_singular_chain* **by** *blast*
have *scpp*: *singular_chain p* (*subtopology* (*powertop_real UNIV*) (*standard_simplex p*))
(*frag_of* (*restrict id* (*standard_simplex p*)))
using *scp simplicial_imp_singular_chain* **by** *blast*
then show *?case*
unfolding *singular_subdivision_def*
using *chain_boundary_chain_map* [*of p subtopology* (*powertop_real UNIV*)
(*standard_simplex p*) _ *f*]
apply (*simp add*: *simplicial_chain_simplicial_subdivision*
simplicial_imp_singular_chain chain_boundary_simplicial_subdivision [*OF scp*]
flip: *singular_subdivision_simplicial_simplex* [*OF scp1*] *naturality_singular_subdivision* [*OF sgp1*])
by (*metis* (*full_types*) *singular_subdivision_def chain_boundary_chain_map* [*OF scpp*] *singular_simplex_chain_map_id* [*OF ssf*])
qed (*auto simp*: *singular_subdivision_def frag_extend_diff chain_boundary_diff*)

lemma *singular_subdivision_zero*:
singular_chain 0 X c ⟹ *singular_subdivision 0 c* = *c*
unfolding *singular_chain_def*
proof (*induction rule*: *frag_induction*)
case (*one f*)
then have *restrict* (*f* ∘ *restrict id* (*standard_simplex 0*)) (*standard_simplex 0*) = *f*
by (*simp add*: *extensional_restrict restrict_compose_right singular_simplex_def*)
then show *?case*
by (*auto simp*: *singular_subdivision_def simplex_map_def*)
qed (*auto simp*: *singular_subdivision_def frag_extend_diff*)

```
primrec subd where
  subd 0 = (λx. 0)
| subd (Suc p) =
      frag_extend
       (λf. simplicial_cone (Suc p) (λi. (∑j≤Suc p. simplicial_vertex j f i) / real
(Suc p + 1))
              (simplicial_subdivision (Suc p) (frag_of f) − frag_of f −
               subd p (chain_boundary (Suc p) (frag_of f))))

lemma subd_0 [simp]: subd p 0 = 0
  by (induction p) auto

lemma subd_diff [simp]: subd p (c1 − c2) = subd p c1 − subd p c2
  by (induction p) (auto simp: frag_extend_diff)

lemma subd_uminus [simp]: subd p (−c) = − subd p c
  by (metis diff_0 subd_0 subd_diff)

lemma subd_power_uminus: subd p (frag_cmul ((−1) ^ k) c) = frag_cmul ((−1)
^ k) (subd p c)
  apply (induction k, simp_all)
  by (metis minus_frag_cmul subd_uminus)

lemma subd_power_sum: subd p (sum f I) = sum (subd p ∘ f) I
  apply (induction I rule: infinite_finite_induct)
  by auto (metis add_diff_cancel_left' diff_add_cancel subd_diff)

lemma subd: simplicial_chain p (standard_simplex s) c
     ⟹ (∀ r g. simplicial_simplex s (standard_simplex r) g ⟶ chain_map (Suc
p) g (subd p c) = subd p (chain_map p g c))
         ∧ simplicial_chain (Suc p) (standard_simplex s) (subd p c)
        ∧ (chain_boundary (Suc p) (subd p c)) + (subd (p − Suc 0) (chain_boundary
p c)) = (simplicial_subdivision p c) − c
proof (induction p arbitrary: c)
  case (Suc p)
  show ?case
    using Suc.prems [unfolded simplicial_chain_def]
  proof (induction rule: frag_induction)
    case (one f)
    then obtain l where l: (λx i. ∑j≤Suc p. l j i ∗ x j) ‘ standard_simplex (Suc
p) ⊆ standard_simplex s
                  and feq: f = oriented_simplex (Suc p) l
     by (metis (mono_tags) mem_Collect_eq simplicial_simplex simplicial_simplex_oriented_simplex)
    have scf: simplicial_chain (Suc p) (standard_simplex s) (frag_of f)
      using one by simp
    have lss: l i ∈ standard_simplex s if i ≤ Suc p for i
    proof −
      have (λi'. ∑j≤Suc p. l j i' ∗ (if j = i then 1 else 0)) ∈ standard_simplex s
```

```
        using subsetD [OF l] basis_in_standard_simplex that by blast
      moreover have (λi'. ∑j≤Suc p. l j i' * (if j = i then 1 else 0)) = l i
         using that by (simp add: if_distrib [of λx. _ * x] del: sum.atMost_Suc
cong: if_cong)
      ultimately show ?thesis
        by simp
    qed
    have *: (⋀i. i ≤ n ⟹ l i ∈ standard_simplex s)
     ⟹ (λi. (∑j≤n. l j i) / (Suc n)) ∈ standard_simplex s for n
    proof (induction n)
      case (Suc n)
      let ?x = λi. (1 − inverse (n + 2)) * ((∑j≤n. l j i) / (Suc n)) + inverse (n
+ 2) * l (Suc n) i
      have ?x ∈ standard_simplex s
      proof (rule convex_standard_simplex)
        show (λi. (∑j≤n. l j i) / real (Suc n)) ∈ standard_simplex s
          using Suc by simp
      qed (auto simp: lss Suc inverse_le_1_iff)
      moreover have ?x = (λi. (∑j≤Suc n. l j i) / real (Suc (Suc n)))
        by (force simp: divide_simps)
      ultimately show ?case
        by simp
    qed auto
     have **: (λi. (∑j≤Suc p. simplicial_vertex j f i) / (2 + real p)) ∈ stan-
dard_simplex s
      using * [of Suc p] lss by (simp add: simplicial_vertex_oriented_simplex feq)
    show ?case
    proof (intro conjI impI allI)
      fix r g
      assume g: simplicial_simplex s (standard_simplex r) g
      then obtain m where geq: g = oriented_simplex s m
        using simplicial_simplex by blast
     have 1: simplicial_chain (Suc p) (standard_simplex s) (simplicial_subdivision
(Suc p) (frag_of f))
      by (metis mem_Collect_eq one.hyps simplicial_chain_of simplicial_chain_simplicial_subdivision)
      have 2: (∑j≤Suc p. ∑i≤s. m i k * simplicial_vertex j f i)
            = (∑j≤Suc p. simplicial_vertex j
                          (simplex_map (Suc p) (oriented_simplex s m) f) k) for k
      proof (rule sum.cong [OF refl])
        fix j
        assume j: j ∈ {..Suc p}
         have eq: simplex_map (Suc p) (oriented_simplex s m) (oriented_simplex
(Suc p) l)
              = oriented_simplex (Suc p) (oriented_simplex s m ∘ l)
        proof (rule simplex_map_oriented_simplex)
          show simplicial_simplex (Suc p) (standard_simplex s) (oriented_simplex
(Suc p) l)
            using one by (simp add: feq flip: oriented_simplex_def)
          show simplicial_simplex s (standard_simplex r) (oriented_simplex s m)
```

```
          using g by (simp add: geq)
        qed auto
        show (∑ i≤s. m i k * simplicial_vertex j f i)
           = simplicial_vertex j (simplex_map (Suc p) (oriented_simplex s m) f) k
          using one j
            apply (simp add: feq eq simplicial_vertex_oriented_simplex simpli-
cial_simplex_oriented_simplex image_subset_iff)
          apply (drule_tac x=(λi. if i = j then 1 else 0) in bspec)
           apply (auto simp: oriented_simplex_def lss)
          done
      qed
      have 4: chain_map (Suc p) g (subd p (chain_boundary (Suc p) (frag_of f)))
              = subd p (chain_boundary (Suc p) (frag_of (simplex_map (Suc p) g
f)))
        by (metis (no_types) One_nat_def scf Suc.IH chain_boundary_chain_map
chain_map_of diff_Suc_Suc diff_zero g simplicial_chain_boundary simplicial_imp_singular_chain)
      show chain_map (Suc (Suc p)) g (subd (Suc p) (frag_of f)) = subd (Suc p)
(chain_map (Suc p) g (frag_of f))
        using g
        apply (simp only: subd.simps frag_extend_of)
        apply (subst chain_map_simplicial_cone [of s standard_simplex r _ Suc p
s], assumption)
           apply (intro simplicial_chain_diff)
        using 1 apply auto[1]
        using one.hyps apply auto[1]
      apply (metis Suc.IH diff_Suc_1 mem_Collect_eq one.hyps simplicial_chain_boundary
simplicial_chain_of)
        using ** apply auto[1]
         apply (rule order_refl)
         apply (simp only: chain_map_of frag_extend_of)
        apply (rule arg_cong2 [where f = simplicial_cone (Suc p)])
          apply (simp add: geq sum_distrib_left oriented_simplex_def ** del:
sum.atMost_Suc flip: sum_divide_distrib)
        using 2  apply (simp only: oriented_simplex_def sum.swap [where A =
{..s}])
         using naturality_simplicial_subdivision scf apply (fastforce simp add: 4
chain_map_diff)
        done
    next
      have sc: simplicial_chain (Suc p) (standard_simplex s)
              (simplicial_cone p
                (λi. (∑ j≤Suc p. simplicial_vertex j f i) / (Suc (Suc p)))
                (simplicial_subdivision p
                  (chain_boundary (Suc p) (frag_of f))))
           by (metis diff_Suc_1 nat.simps(3) simplicial_subdivision_of scf simpli-
cial_chain_simplicial_subdivision)
     have ff: simplicial_chain (Suc p) (standard_simplex s) (subd p (chain_boundary
(Suc p) (frag_of f)))
        by (metis (no_types) Suc.IH diff_Suc_1 scf simplicial_chain_boundary)
```

show *simplicial_chain* (*Suc* (*Suc* *p*)) (*standard_simplex* *s*) (*subd* (*Suc* *p*) (*frag_of* *f*))
using *one*
apply (*simp only*: *subd.simps frag_extend_of*)
apply (*rule_tac S*=*standard_simplex s* **in** *simplicial_chain_simplicial_cone*)
apply (*intro simplicial_chain_diff ff*)
using *sc* **apply** (*simp add*: *algebra_simps*)
using ∗∗ *convex_standard_simplex* **apply** *force*+
done
have *simplicial_chain p* (*standard_simplex s*) (*chain_boundary* (*Suc p*) (*frag_of f*))
using *scf simplicial_chain_boundary* **by** *fastforce*
then have *chain_boundary* (*Suc p*) (*simplicial_subdivision* (*Suc p*) (*frag_of f*) − *frag_of f*
− *subd p* (*chain_boundary* (*Suc p*) (*frag_of f*))) = *0*
apply (*simp only*: *chain_boundary_diff*)
using *Suc.IH chain_boundary_boundary* [*of Suc p subtopology* (*powertop_real UNIV*)
(*standard_simplex s*) *frag_of f*]
by (*metis One_nat_def add_diff_cancel_left′ subd_0 chain_boundary_simplicial_subdivision plus_1_eq_Suc scf simplicial_imp_singular_chain*)
then show *chain_boundary* (*Suc* (*Suc p*)) (*subd* (*Suc p*) (*frag_of f*))
+ *subd* (*Suc p* − *Suc 0*) (*chain_boundary* (*Suc p*) (*frag_of f*))
= *simplicial_subdivision* (*Suc p*) (*frag_of f*) − *frag_of f*
apply (*simp only*: *subd.simps frag_extend_of*)
apply (*subst chain_boundary_simplicial_cone* [*of Suc p standard_simplex s*])
apply (*meson ff scf simplicial_chain_diff simplicial_chain_simplicial_subdivision*)
apply (*simp add*: *simplicial_cone_def del*: *sum.atMost_Suc simplicial_subdivision.simps*)
done
qed
next
case (*diff a b*)
then show *?case*
apply *safe*
apply (*metis chain_map_diff subd_diff*)
apply (*metis simplicial_chain_diff subd_diff*)
apply (*auto simp*: *simplicial_subdivision_diff chain_boundary_diff*
simp del: *simplicial_subdivision.simps subd.simps*)
by (*metis* (*no_types, lifting*) *add_diff_add add_uminus_conv_diff diff_0 diff_diff_add*)
qed *auto*
qed *simp*

lemma *chain_homotopic_simplicial_subdivision1*:
⟦*simplicial_chain p* (*standard_simplex q*) *c*; *simplicial_simplex q* (*standard_simplex r*) *g*⟧
⟹ *chain_map* (*Suc p*) *g* (*subd p c*) = *subd p* (*chain_map p g c*)

by (*simp add*: *subd*)

lemma *chain_homotopic_simplicial_subdivision2*:
simplicial_chain p (*standard_simplex q*) *c*
⟹ *simplicial_chain* (*Suc p*) (*standard_simplex q*) (*subd p c*)
by (*simp add*: *subd*)

lemma *chain_homotopic_simplicial_subdivision3*:
simplicial_chain p (*standard_simplex q*) *c*
⟹ *chain_boundary* (*Suc p*) (*subd p c*) = (*simplicial_subdivision p c*) − *c* − *subd* (*p* − *Suc 0*) (*chain_boundary p c*)
by (*simp add*: *subd algebra_simps*)

lemma *chain_homotopic_simplicial_subdivision*:
∃ *h*. (∀ *p*. *h p 0* = *0*) ∧
(∀ *p c1 c2*. *h p* (*c1*−*c2*) = *h p c1* − *h p c2*) ∧
(∀ *p q r g c*.
simplicial_chain p (*standard_simplex q*) *c*
⟶ *simplicial_simplex q* (*standard_simplex r*) *g*
⟶ *chain_map* (*Suc p*) *g* (*h p c*) = *h p* (*chain_map p g c*)) ∧
(∀ *p q c*. *simplicial_chain p* (*standard_simplex q*) *c*
⟶ *simplicial_chain* (*Suc p*) (*standard_simplex q*) (*h p c*)) ∧
(∀ *p q c*. *simplicial_chain p* (*standard_simplex q*) *c*
⟶ *chain_boundary* (*Suc p*) (*h p c*) + *h* (*p* − *Suc 0*) (*chain_boundary p c*)
= (*simplicial_subdivision p c*) − *c*)
by (*rule_tac x*=*subd* **in** *exI*) (*fastforce simp*: *subd*)

lemma *chain_homotopic_singular_subdivision*:
obtains *h* **where**
⋀*p*. *h p 0* = *0*
⋀*p c1 c2*. *h p* (*c1*−*c2*) = *h p c1* − *h p c2*
⋀*p X c*. *singular_chain p X c* ⟹ *singular_chain* (*Suc p*) *X* (*h p c*)
⋀*p X c*. *singular_chain p X c*
⟹ *chain_boundary* (*Suc p*) (*h p c*) + *h* (*p* − *Suc 0*) (*chain_boundary p c*) = *singular_subdivision p c* − *c*
proof −
define *k* **where** *k* ≡ λ*p*. *frag_extend* (λ*f*:: (*nat* ⇒ *real*) ⇒ ′*a*. *chain_map* (*Suc p*) *f* (*subd p* (*frag_of*(*restrict id* (*standard_simplex p*)))))
show *?thesis*
proof
fix *p X* **and** *c* :: ′*a chain*
assume *c*: *singular_chain p X c*
have *singular_chain* (*Suc p*) *X* (*k p c*) ∧
chain_boundary (*Suc p*) (*k p c*) + *k* (*p* − *Suc 0*) (*chain_boundary p c*) = *singular_subdivision p c* − *c*
using *c* [*unfolded singular_chain_def*]
proof (*induction rule*: *frag_induction*)
case (*one f*)

let *?X = subtopology (powertop_real UNIV) (standard_simplex p)*
show *?case*
proof *(simp add: k_def, intro conjI)*
show *singular_chain (Suc p) X (chain_map (Suc p) f (subd p (frag_of (restrict id (standard_simplex p)))))*
proof *(rule singular_chain_chain_map)*
show *singular_chain (Suc p) ?X (subd p (frag_of (restrict id (standard_simplex p))))*
by *(simp add: chain_homotopic_simplicial_subdivision2 simplicial_imp_singular_chain)*
show *continuous_map ?X X f*
using *one.hyps singular_simplex_def* **by** *auto*
qed
next
have *scp: singular_chain (Suc p) ?X (subd p (frag_of (restrict id (standard_simplex p))))*
by *(simp add: chain_homotopic_simplicial_subdivision2 simplicial_imp_singular_chain)*
have *feqf: frag_of (simplex_map p f (restrict id (standard_simplex p))) = frag_of f*
using *one.hyps singular_simplex_chain_map_id* **by** *auto*
have *∗: chain_map p f*
(subd (p − Suc 0)
(∑ k≤p. frag_cmul ((−1) ^ k) (frag_of (singular_face p k id))))
= (∑ x≤p. frag_cmul ((−1) ^ x)
(chain_map p (singular_face p x f)
(subd (p − Suc 0) (frag_of (restrict id (standard_simplex (p − Suc 0)))))))
(**is** *?lhs = ?rhs*)
if *p > 0*
proof −
have *eqc: subd (p − Suc 0) (frag_of (singular_face p i id))*
= chain_map p (singular_face p i id)
(subd (p − Suc 0) (frag_of (restrict id (standard_simplex (p − Suc 0)))))
if *i ≤ p* **for** *i*
proof −
have *1: simplicial_chain (p − Suc 0) (standard_simplex (p − Suc 0))*
(frag_of (restrict id (standard_simplex (p − Suc 0))))
by *simp*
have *2: simplicial_simplex (p − Suc 0) (standard_simplex p) (singular_face p i id)*
by *(metis One_nat_def Suc_leI ‹0 < p› simplicial_simplex_id simplicial_simplex_singular_face singular_face_restrict subsetI that)*
have *3: simplex_map (p − Suc 0) (singular_face p i id) (restrict id (standard_simplex (p − Suc 0)))*
= singular_face p i id
by *(force simp: simplex_map_def singular_face_def)*
show *?thesis*
using *chain_homotopic_simplicial_subdivision1 [OF 1 2]*
that ‹p > 0› **by** *(simp add: 3)*

qed
have *xx*: *simplicial_chain p* (*standard_simplex*(*p* − *Suc 0*))
(*subd* (*p* − *Suc 0*) (*frag_of* (*restrict id* (*standard_simplex* (*p* − *Suc 0*)))))
by (*metis Suc_pred chain_homotopic_simplicial_subdivision2 order_refl simplicial_chain_of simplicial_simplex_id that*)
have *yy*: ⋀*k*. *k* ≤ *p* ⟹
chain_map p f
(*chain_map p* (*singular_face p k id*) *h*) = *chain_map p* (*singular_face p k f*) *h*
if *simplicial_chain p* (*standard_simplex*(*p* − *Suc 0*)) *h* **for** *h*
using *that* **unfolding** *simplicial_chain_def*
proof (*induction h rule*: *frag_induction*)
case (*one x*)
then show *?case*
using *one*
apply (*simp add*: *chain_map_of singular_simplex_def simplicial_simplex_def*, *auto*)
apply (*rule_tac f=frag_of* **in** *arg_cong*, *rule*)
apply (*simp add*: *simplex_map_def*)
by (*simp add*: *continuous_map_in_subtopology image_subset_iff singular_face_def*)
qed (*auto simp*: *chain_map_diff*)
have *?lhs*
= *chain_map p f*
(∑*k*≤*p*. *frag_cmul* ((−*1*) ^ *k*)
(*chain_map p* (*singular_face p k id*)
(*subd* (*p* − *Suc 0*) (*frag_of* (*restrict id* (*standard_simplex* (*p* − *Suc 0*)))))))
by (*simp add*: *subd_power_sum subd_power_uminus eqc*)
also have ... = *?rhs*
by (*simp add*: *chain_map_sum xx yy*)
finally show *?thesis* **.**
qed
have *chain_map p f*
(*simplicial_subdivision p* (*frag_of* (*restrict id* (*standard_simplex p*)))
− *subd* (*p* − *Suc 0*) (*chain_boundary p* (*frag_of* (*restrict id* (*standard_simplex p*)))))
= *singular_subdivision p* (*frag_of f*)
− *frag_extend*
(λ*f*. *chain_map* (*Suc* (*p* − *Suc 0*)) *f*
(*subd* (*p* − *Suc 0*) (*frag_of* (*restrict id* (*standard_simplex* (*p* − *Suc 0*))))))
(*chain_boundary p* (*frag_of f*))
apply (*simp add*: *singular_subdivision_def chain_map_diff*)
apply (*clarsimp simp add*: *chain_boundary_def*)
apply (*simp add*: *frag_extend_sum frag_extend_cmul* ∗)
done

```
        then show chain_boundary (Suc p) (chain_map (Suc p) f (subd p (frag_of
(restrict id (standard_simplex p)))))
                   + frag_extend
                     (λf. chain_map (Suc (p − Suc 0)) f
                            (subd (p − Suc 0) (frag_of (restrict id (standard_simplex (p
− Suc 0))))))
                     (chain_boundary p (frag_of f))
                   = singular_subdivision p (frag_of f) − frag_of f
        by (simp add: chain_boundary_chain_map [OF scp] chain_homotopic_simplicial_subdivision3
[where q=p] chain_map_diff feqf)
      qed
    next
      case (diff a b)
      then show ?case
      apply (simp only: k_def singular_chain_diff chain_boundary_diff frag_extend_diff
singular_subdivision_diff)
        by (metis (no_types, lifting) add_diff_add diff_add_cancel)
    qed (auto simp: k_def)
    then show singular_chain (Suc p) X (k p c) chain_boundary (Suc p) (k p c)
+ k (p − Suc 0) (chain_boundary p c) = singular_subdivision p c − c
        by auto
  qed (auto simp: k_def frag_extend_diff)
qed


lemma homologous_rel_singular_subdivision:
  assumes singular_relcycle p X T c
  shows homologous_rel p X T (singular_subdivision p c) c
proof (cases p = 0)
  case True
  with assms show ?thesis
    by (auto simp: singular_relcycle_def singular_subdivision_zero)
next
  case False
  with assms show ?thesis
    unfolding homologous_rel_def singular_relboundary singular_relcycle
     by (metis One_nat_def Suc_diff_1 chain_homotopic_singular_subdivision
gr_zeroI)
qed
```

0.1.17 Excision argument that we keep doing singular subdivision

```
lemma singular_subdivision_power_0 [simp]: (singular_subdivision p ^^ n) 0 =
0
  by (induction n) auto

lemma singular_subdivision_power_diff:
   (singular_subdivision p ^^ n) (a − b) = (singular_subdivision p ^^ n) a −
```

```
(singular_subdivision p ^^ n) b
  by (induction n) (auto simp: singular_subdivision_diff)

lemma iterated_singular_subdivision:
   singular_chain p X c
       ⟹ (singular_subdivision p ^^ n) c =
           frag_extend
            (λf. chain_map p f
                     ((simplicial_subdivision p ^^ n)
                       (frag_of(restrict id (standard_simplex p))))) c
proof (induction n arbitrary: c)
  case 0
  then show ?case
    unfolding singular_chain_def
  proof (induction c rule: frag_induction)
    case (one f)
    then have restrict f (standard_simplex p) = f
      by (simp add: extensional_restrict singular_simplex_def)
    then show ?case
      by (auto simp: simplex_map_def cong: restrict_cong)
  qed (auto simp: frag_extend_diff)
next
  case (Suc n)
  show ?case
    using Suc.prems unfolding singular_chain_def
  proof (induction c rule: frag_induction)
    case (one f)
    then have singular_simplex p X f
      by simp
    have scp: simplicial_chain p (standard_simplex p)
            ((simplicial_subdivision p ^^ n) (frag_of (restrict id (standard_simplex
p))))
    proof (induction n)
      case 0
      then show ?case
        by (metis funpow_0 order_refl simplicial_chain_of simplicial_simplex_id)
    next
      case (Suc n)
      then show ?case
        by (simp add: simplicial_chain_simplicial_subdivision)
    qed
    have scnp: simplicial_chain p (standard_simplex p)
              ((simplicial_subdivision p ^^ n) (frag_of (λx∈standard_simplex p.
x)))
    proof (induction n)
      case 0
      then show ?case
           by (metis eq_id_iff funpow_0 order_refl simplicial_chain_of simpli-
cial_simplex_id)
```

```
    next
      case (Suc n)
      then show ?case
        by (simp add: simplicial_chain_simplicial_subdivision)
    qed
    have sff: singular_chain p X (frag_of f)
      by (simp add: ‹singular_simplex p X f› singular_chain_of)
    then show ?case
     using Suc.IH [OF sff] naturality_singular_subdivision [OF simplicial_imp_singular_chain
[OF scp], of f] singular_subdivision_simplicial_simplex [OF scnp]
      by (simp add: singular_chain_of id_def del: restrict_apply)
  qed (auto simp: singular_subdivision_power_diff singular_subdivision_diff frag_extend_diff)
qed


lemma chain_homotopic_iterated_singular_subdivision:
  obtains h where
        ⋀p. h p 0 = (0 :: 'a chain)
        ⋀p c1 c2. h p (c1−c2) = h p c1 − h p c2
        ⋀p X c. singular_chain p X c ⟹ singular_chain (Suc p) X (h p c)
        ⋀p X c. singular_chain p X c
             ⟹ chain_boundary (Suc p) (h p c) + h (p − Suc 0) (chain_boundary
p c)
                  = (singular_subdivision p ^^ n) c − c
proof (induction n arbitrary: thesis)
  case 0
  show ?case
    by (rule 0 [of (λp x. 0)]) auto
next
  case (Suc n)
  then obtain k where k:
        ⋀p. k p 0 = (0 :: 'a chain)
        ⋀p c1 c2. k p (c1−c2) = k p c1 − k p c2
        ⋀p X c. singular_chain p X c ⟹ singular_chain (Suc p) X (k p c)
        ⋀p X c. singular_chain p X c
             ⟹ chain_boundary (Suc p) (k p c) + k (p − Suc 0) (chain_boundary
p c)
                   = (singular_subdivision p ^^ n) c − c
    by metis
  obtain h where h:
        ⋀p. h p 0 = (0 :: 'a chain)
        ⋀p c1 c2. h p (c1−c2) = h p c1 − h p c2
        ⋀p X c. singular_chain p X c ⟹ singular_chain (Suc p) X (h p c)
        ⋀p X c. singular_chain p X c
             ⟹ chain_boundary (Suc p) (h p c) + h (p − Suc 0) (chain_boundary
p c) = singular_subdivision p c − c
    by (blast intro: chain_homotopic_singular_subdivision)
  let ?h = (λp c. singular_subdivision (Suc p) (k p c) + h p c)
  show ?case
```

```
  proof (rule Suc.prems)
    fix p X and c :: 'a chain
    assume singular_chain p X c
    then show singular_chain (Suc p) X (?h p c)
      by (simp add: h k singular_chain_add singular_chain_singular_subdivision)
  next
    fix p :: nat and X :: 'a topology and c :: 'a chain
    assume sc: singular_chain p X c
     have f5: chain_boundary (Suc p) (singular_subdivision (Suc p) (k p c)) =
singular_subdivision p (chain_boundary (Suc p) (k p c))
      using chain_boundary_singular_subdivision k(3) sc by fastforce
   have [simp]: singular_subdivision (Suc (p − Suc 0)) (k (p − Suc 0) (chain_boundary
p c)) =
                singular_subdivision p (k (p − Suc 0) (chain_boundary p c))
    proof (cases p)
      case 0
      then show ?thesis
        by (simp add: k chain_boundary_def)
    qed auto
    show chain_boundary (Suc p) (?h p c) + ?h (p − Suc 0) (chain_boundary p
c) = (singular_subdivision p ^^ Suc n) c − c
      using chain_boundary_singular_subdivision [of Suc p X]
      apply (simp add: chain_boundary_add f5 h k algebra_simps)
     apply (smt (verit, ccfv_threshold) add.commute add_diff_eq diff_add_cancel
diff_diff_eq2 h(4) k(4) sc singular_subdivision_add)
      done
  qed (auto simp: k h singular_subdivision_diff)
qed

lemma llemma:
  assumes p: standard_simplex p ⊆ ⋃𝒞
      and 𝒞: ⋀U. U ∈ 𝒞 ⟹ openin (powertop_real UNIV) U
  obtains d where 0 < d
                ⋀K. ⟦K ⊆ standard_simplex p;
                     ⋀x y i. ⟦i ≤ p; x ∈ K; y ∈ K⟧ ⟹ |x i − y i| ≤ d⟧
                    ⟹ ∃U. U ∈ 𝒞 ∧ K ⊆ U
proof −
  have ∃e U. 0 < e ∧ U ∈ 𝒞 ∧ x ∈ U ∧
              (∀y. (∀i≤p. |y i − x i| ≤ 2 ∗ e) ∧ (∀i>p. y i = 0) ⟶ y ∈ U)
    if x: x ∈ standard_simplex p for x
  proof−
    obtain U where U: U ∈ 𝒞 x ∈ U
      using x p by blast
     then obtain V where finV: finite {i. V i ≠ UNIV} and openV: ⋀i. open
(V i)
                and xV: x ∈ Pi_E UNIV V and UV: Pi_E UNIV V ⊆ U
      using 𝒞 unfolding openin_product_topology_alt by force
    have xVi: x i ∈ V i for i
      using PiE_mem [OF xV] by simp
```

```
    have ⋀i. ∃e>0. ∀x'. |x' − x i| < e ⟶ x' ∈ V i
      by (rule openV [unfolded open_real, rule_format, OF xVi])
    then obtain d where d: ⋀i. d i > 0 and dV: ⋀i x'. |x' − x i| < d i ⟹ x'
∈ V i
      by metis
    define e where e ≡ Inf (insert 1 (d ` {i. V i ≠ UNIV})) / 3
    have ed3: e ≤ d i / 3 if V i ≠ UNIV for i
      using that finV by (auto simp: e_def intro: cInf_le_finite)
    show ∃e U. 0 < e ∧ U ∈ 𝒞 ∧ x ∈ U ∧
             (∀y. (∀i≤p. |y i − x i| ≤ 2 ∗ e) ∧ (∀i>p. y i = 0) ⟶ y ∈ U)
    proof (intro exI conjI allI impI)
      show e > 0
        using d finV by (simp add: e_def finite_less_Inf_iff)
      fix y assume y: (∀i≤p. |y i − x i| ≤ 2 ∗ e) ∧ (∀i>p. y i = 0)
      have y ∈ Pi_E UNIV V
      proof
        show y i ∈ V i for i
        proof (cases p < i)
          case True
          then show ?thesis
           by (metis (mono_tags, lifting) y x mem_Collect_eq standard_simplex_def
xVi)
        next
          case False show ?thesis
          proof (cases V i = UNIV)
            case False show ?thesis
            proof (rule dV)
              have |y i − x i| ≤ 2 ∗ e
                using y ‹¬ p < i› by simp
              also have ... < d i
                using ed3 [OF False] ‹e > 0› by simp
              finally show |y i − x i| < d i .
            qed
          qed auto
        qed
      qed auto
      with UV show y ∈ U
        by blast
    qed (use U in auto)
  qed
  then obtain e U where
      eU: ⋀x. x ∈ standard_simplex p ⟹
               0 < e x ∧ U x ∈ 𝒞 ∧ x ∈ U x
     and UI: ⋀x y. ⟦x ∈ standard_simplex p; ⋀i. i ≤ p ⟹ |y i − x i| ≤ 2 ∗ e
x; ⋀i. i > p ⟹ y i = 0⟧
                   ⟹ y ∈ U x
    by metis
  define F where F ≡ λx. Pi_E UNIV (λi. if i ≤ p then {x i − e x<..<x i + e
x} else UNIV)
```

```
 have ∀ S ∈ F ‘ standard_simplex p. openin (powertop_real UNIV) S
   by (simp add: F_def openin_PiE_gen)
 moreover have pF: standard_simplex p ⊆ ⋃(F ‘ standard_simplex p)
   by (force simp: F_def PiE_iff eU)
 ultimately have ∃ℱ. finite ℱ ∧ ℱ ⊆ F ‘ standard_simplex p ∧ standard_simplex
p ⊆ ⋃ℱ
   using compactin_standard_simplex [of p]
   unfolding compactin_def by force
 then obtain S where finite S and ssp: S ⊆ standard_simplex p standard_simplex
p ⊆ ⋃(F ‘ S)
   unfolding ex_finite_subset_image by (auto simp: ex_finite_subset_image)
 then have S ≠ {}
   by (auto simp: nonempty_standard_simplex)
 show ?thesis
 proof
   show Inf (e ‘ S) > 0
     using ‹finite S› ‹S ≠ {}› ssp eU by (auto simp: finite_less_Inf_iff)
   fix k :: (nat ⇒ real) set
   assume k: k ⊆ standard_simplex p
       and kle: ⋀x y i. ⟦i ≤ p; x ∈ k; y ∈ k⟧ ⟹ |x i − y i| ≤ Inf (e ‘ S)
   show ∃ U. U ∈ 𝒞 ∧ k ⊆ U
   proof (cases k = {})
     case True
     then show ?thesis
       using ‹S ≠ {}› eU equals0I ssp(1) subset_eq p by auto
   next
     case False
     with k ssp obtain x a where x ∈ k x ∈ standard_simplex p
                     and a: a ∈ S and Fa: x ∈ F a
       by blast
     then have le_ea: ⋀i. i ≤ p ⟹ abs (x i − a i) < e a
       by (simp add: F_def PiE_iff if_distrib abs_diff_less_iff cong: if_cong)
     show ?thesis
     proof (intro exI conjI)
       show U a ∈ 𝒞
         using a eU ssp(1) by auto
       show k ⊆ U a
       proof clarify
         fix y assume y ∈ k
         with k have y: y ∈ standard_simplex p
           by blast
         show y ∈ U a
         proof (rule UI)
           show a ∈ standard_simplex p
             using a ssp(1) by auto
           fix i :: nat
           assume i ≤ p
           then have |x i − y i| ≤ e a
             by (meson kle [OF ‹i ≤ p›] a ‹finite S› ‹x ∈ k› ‹y ∈ k› cInf_le_finite
```

```
finite_imageI imageI order_trans)
            then show |y i − a i| ≤ 2 ∗ e a
              using le_ea [OF ‹i ≤ p›] by linarith
          next
            fix i assume p < i
            then show y i = 0
              using standard_simplex_def y by auto
          qed
        qed
      qed
    qed
  qed
qed


proposition sufficient_iterated_singular_subdivision_exists:
  assumes 𝒞: ⋀U. U ∈ 𝒞 ⟹ openin X U
      and X: topspace X ⊆ ⋃𝒞
      and p: singular_chain p X c
 obtains n where ⋀m f. ⟦n ≤ m; f ∈ Poly_Mapping.keys ((singular_subdivision
p ^^ m) c)⟧
                      ⟹ ∃ V ∈ 𝒞. f ‘ (standard_simplex p) ⊆ V
proof (cases c = 0)
  case False
  then show ?thesis
  proof (cases topspace X = {})
    case True
    show ?thesis
      using p that by (force simp: singular_chain_empty True)
  next
    case False
    show ?thesis
    proof (cases 𝒞 = {})
      case True
      then show ?thesis
        using False X by blast
    next
      case False
      have ∃ e. 0 < e ∧
               (∀ K. K ⊆ standard_simplex p ⟶ (∀ x y i. x ∈ K ∧ y ∈ K ∧ i ≤ p
⟶ |x i − y i| ≤ e)
                      ⟶ (∃ V. V ∈ 𝒞 ∧ f ‘ K ⊆ V))
        if f: f ∈ Poly_Mapping.keys c for f
      proof −
        have ssf: singular_simplex p X f
          using f p by (auto simp: singular_chain_def)
        then have fp: ⋀x. x ∈ standard_simplex p ⟹ f x ∈ topspace X
             by (auto simp: singular_simplex_def image_subset_iff dest: continu-
ous_map_image_subset_topspace)
```

have ∃ *T*. *openin* (*powertop_real UNIV*) *T* ∧
standard_simplex p ∩ *f* −' *V* = *T* ∩ *standard_simplex p*
if *V*: *V* ∈ 𝒞 **for** *V*
proof −
have *singular_simplex p X f*
using *p f* **unfolding** *singular_chain_def* **by** *blast*
then have *openin* (*subtopology* (*powertop_real UNIV*) (*standard_simplex p*))
{*x* ∈ *standard_simplex p*. *f x* ∈ *V*}
using 𝒞 [*OF* ‹*V* ∈ 𝒞›] **by** (*simp add*: *singular_simplex_def continuous_map_def*)
moreover have *standard_simplex p* ∩ *f* −' *V* = {*x* ∈ *standard_simplex p*. *f x* ∈ *V*}
by *blast*
ultimately show *?thesis*
by (*simp add*: *openin_subtopology*)
qed
then obtain *g* **where** *gope*: ⋀*V*. *V* ∈ 𝒞 ⟹ *openin* (*powertop_real UNIV*) (*g V*)
and *geq*: ⋀*V*. *V* ∈ 𝒞 ⟹ *standard_simplex p* ∩ *f* −' *V* = *g V* ∩ *standard_simplex p*
by *metis*
obtain *d* **where** *0* < *d*
and *d*: ⋀*K*. ⟦*K* ⊆ *standard_simplex p*; ⋀*x y i*. ⟦*i* ≤ *p*; *x* ∈ *K*; *y* ∈ *K*⟧ ⟹ |*x i* − *y i*| ≤ *d*⟧
⟹ ∃ *U*. *U* ∈ *g* ' 𝒞 ∧ *K* ⊆ *U*
proof (*rule llemma* [*of p g* ' 𝒞])
show *standard_simplex p* ⊆ ⋃(*g* ' 𝒞)
using *geq X fp* **by** (*fastforce simp add*:)
show *openin* (*powertop_real UNIV*) *U* **if** *U* ∈ *g* ' 𝒞 **for** *U* :: (*nat* ⇒ *real*) *set*
using *gope that* **by** *blast*
qed *auto*
show *?thesis*
proof (*rule exI*, *intro allI conjI impI*)
fix *K* :: (*nat* ⇒ *real*) *set*
assume *K*: *K* ⊆ *standard_simplex p*
and *Kd*: ∀ *x y i*. *x* ∈ *K* ∧ *y* ∈ *K* ∧ *i* ≤ *p* ⟶ |*x i* − *y i*| ≤ *d*
then have ∃ *U*. *U* ∈ *g* ' 𝒞 ∧ *K* ⊆ *U*
using *d* [*OF K*] **by** *auto*
then show ∃ *V*. *V* ∈ 𝒞 ∧ *f* ' *K* ⊆ *V*
using *K geq* **by** *fastforce*
qed (*rule* ‹*d* > *0*›)
qed
then obtain *ψ* **where** *epos*: ∀ *f* ∈ *Poly_Mapping.keys c*. *0* < *ψ f*
and *e*: ⋀*f K*. ⟦*f* ∈ *Poly_Mapping.keys c*; *K* ⊆ *standard_simplex p*;
⋀*x y i*. *x* ∈ *K* ∧ *y* ∈ *K* ∧ *i* ≤ *p* ⟹ |*x i* − *y i*| ≤ *ψ f*⟧
⟹ ∃ *V*. *V* ∈ 𝒞 ∧ *f* ' *K* ⊆ *V*
by *metis*

```
    obtain d where 0 < d
          and d: ⋀f K. ⟦f ∈ Poly_Mapping.keys c; K ⊆ standard_simplex p;
                   ⋀x y i. ⟦x ∈ K; y ∈ K; i ≤ p⟧ ⟹ |x i − y i| ≤ d⟧
                   ⟹ ∃ V. V ∈ 𝒞 ∧ f ‘ K ⊆ V
    proof
      show Inf (ψ ‘ Poly_Mapping.keys c) > 0
        by (simp add: finite_less_Inf_iff ‹c ≠ 0› epos)
      fix f K
      assume fK: f ∈ Poly_Mapping.keys c K ⊆ standard_simplex p
          and le: ⋀x y i. ⟦x ∈ K; y ∈ K; i ≤ p⟧ ⟹ |x i − y i| ≤ Inf (ψ ‘
Poly_Mapping.keys c)
      then have lef: Inf (ψ ‘ Poly_Mapping.keys c) ≤ ψ f
        by (auto intro: cInf_le_finite)
      show ∃ V. V ∈ 𝒞 ∧ f ‘ K ⊆ V
        using le lef by (blast intro: dual_order.trans e [OF fK])
    qed
  let ?d = λm. (simplicial_subdivision p ^^ m) (frag_of (restrict id (standard_simplex
p)))
    obtain n where n: (p / (Suc p)) ^ n < d
      using real_arch_pow_inv ‹0 < d› by fastforce
    show ?thesis
    proof
      fix m h
       assume n ≤ m and h ∈ Poly_Mapping.keys ((singular_subdivision p ^^
m) c)
       then obtain f where f ∈ Poly_Mapping.keys c h ∈ Poly_Mapping.keys
(chain_map p f (?d m))
         using subsetD [OF keys_frag_extend] iterated_singular_subdivision [OF
p, of m] by force
        then obtain g where g: g ∈ Poly_Mapping.keys (?d m) and heq: h =
restrict (f ∘ g) (standard_simplex p)
     using keys_frag_extend by (force simp: chain_map_def simplex_map_def)
      have xx: simplicial_chain p (standard_simplex p) (?d n) ∧
                 (∀ f ∈ Poly_Mapping.keys(?d n). ∀ x ∈ standard_simplex p. ∀ y ∈
standard_simplex p.
                   |f x i − f y i| ≤ (p / (Suc p)) ^ n)
        for n i
      proof (induction n)
        case 0
        have simplicial_simplex p (standard_simplex p) (λa∈standard_simplex p.
a)
          by (metis eq_id_iff order_refl simplicial_simplex_id)
        moreover have (∀ x∈standard_simplex p. ∀ y∈standard_simplex p. |x i −
y i| ≤ 1)
          unfolding standard_simplex_def
          by (auto simp: abs_if dest!: spec [where x=i])
        ultimately show ?case
          unfolding power_0 funpow_0 by simp
      next
```

```
        case (Suc n)
        show ?case
          unfolding power_Suc funpow.simps o_def
        proof (intro conjI ballI)
          show simplicial_chain p (standard_simplex p) (simplicial_subdivision p
(?d n))
            by (simp add: Suc simplicial_chain_simplicial_subdivision)
          show |f x i − f y i| ≤ real p / real (Suc p) ∗ (real p / real (Suc p)) ^ n
            if f ∈ Poly_Mapping.keys (simplicial_subdivision p (?d n))
              and x ∈ standard_simplex p and y ∈ standard_simplex p for f x y
            using Suc that by (blast intro: simplicial_subdivision_shrinks)
        qed
      qed
      have g ‘ standard_simplex p ⊆ standard_simplex p
        using g xx [of m] unfolding simplicial_chain_def simplicial_simplex by
auto
      moreover
     have |g x i − g y i| ≤ d if i ≤ p x ∈ standard_simplex p y ∈ standard_simplex
p for x y i
      proof −
        have |g x i − g y i| ≤ (p / (Suc p)) ^ m
          using g xx [of m] that by blast
        also have . . . ≤ (p / (Suc p)) ^ n
          by (auto intro: power_decreasing [OF ‹n ≤ m›])
        finally show ?thesis using n by simp
      qed
      then have |x i − y i| ≤ d
        if x ∈ g ‘ (standard_simplex p) y ∈ g ‘ (standard_simplex p) i ≤ p for i
x y
        using that by blast
      ultimately show ∃ V∈C. h ‘ standard_simplex p ⊆ V
        using ‹f ∈ Poly_Mapping.keys c› d [of f g ‘ standard_simplex p]
        by (simp add: Bex_def heq image_image)
    qed
  qed
 qed
qed force


lemma small_homologous_rel_relcycle_exists:
 assumes C: ⋀U. U ∈ C =⇒ openin X U
   and X: topspace X ⊆ ⋃C
   and p: singular_relcycle p X S c
  obtains c′ where singular_relcycle p X S c′ homologous_rel p X S c c′
             ⋀f. f ∈ Poly_Mapping.keys c′ =⇒ ∃ V ∈ C. f ‘ (standard_simplex
p) ⊆ V
proof −
 have singular_chain p X c
   (chain_boundary p c, 0) ∈ (mod_subset (p − Suc 0) (subtopology X S))
```

```
    using p unfolding singular_relcycle_def by auto
  then obtain n where n: ⋀m f. ⟦n ≤ m; f ∈ Poly_Mapping.keys ((singular_subdivision
p ^^ m) c)⟧
                         ⟹ ∃ V ∈ 𝒞. f ‘ (standard_simplex p) ⊆ V
    by (blast intro: sufficient_iterated_singular_subdivision_exists [OF 𝒞 X])
  let ?c' = (singular_subdivision p ^^ n) c
  show ?thesis
  proof
    show homologous_rel p X S c ?c'
      apply (induction n, simp_all)
     by (metis p homologous_rel_singular_subdivision homologous_rel_singular_relcycle
homologous_rel_trans homologous_rel_sym)
    then show singular_relcycle p X S ?c'
      by (metis homologous_rel_singular_relcycle p)
  next
    fix f :: (nat ⇒ real) ⇒ 'a
    assume f ∈ Poly_Mapping.keys ?c'
    then show ∃ V∈𝒞. f ‘ standard_simplex p ⊆ V
      by (rule n [OF order_refl])
  qed
qed

lemma excised_chain_exists:
  fixes S :: 'a set
  assumes X closure_of U ⊆ X interior_of T T ⊆ S singular_chain p (subtopology
X S) c
  obtains n d e where singular_chain p (subtopology X (S − U)) d
                      singular_chain p (subtopology X T) e
                      (singular_subdivision p ^^ n) c = d + e
proof −
  have *: ∃ n d e. singular_chain p (subtopology X (S − U)) d ∧
                 singular_chain p (subtopology X T) e ∧
                 (singular_subdivision p ^^ n) c = d + e
    if c: singular_chain p (subtopology X S) c
       and X: X closure_of U ⊆ X interior_of T U ⊆ topspace X and S: T ⊆ S
S ⊆ topspace X
       for p X c S and T U :: 'a set
  proof −
   obtain n where n: ⋀m f. ⟦n ≤ m; f ∈ Poly_Mapping.keys ((singular_subdivision
p ^^ m) c)⟧
                         ⟹ ∃ V ∈ {S ∩ X interior_of T, S − X closure_of U}. f
‘ standard_simplex p ⊆ V
      apply (rule sufficient_iterated_singular_subdivision_exists
                  [of {S ∩ X interior_of T, S − X closure_of U}])
      using X S c
     by (auto simp: topspace_subtopology openin_subtopology_Int2 openin_subtopology_diff_closed)
    let ?c' = λn. (singular_subdivision p ^^ n) c
    have singular_chain p (subtopology X S) (?c' m) for m
      by (induction m) (auto simp: singular_chain_singular_subdivision c)
```

then have *scp*: *singular_chain p* (*subtopology X S*) (*?c′ n*) **.**

have *SS*: *Poly_Mapping.keys* (*?c′ n*) ⊆ *singular_simplex_set p* (*subtopology X* (*S* − *U*))
∪ *singular_simplex_set p* (*subtopology X T*)
proof (*clarsimp*)
fix *f*
assume *f*: *f* ∈ *Poly_Mapping.keys* ((*singular_subdivision p* ^^ *n*) *c*)
and *non*: ¬ *singular_simplex p* (*subtopology X T*) *f*
show *singular_simplex p* (*subtopology X* (*S* − *U*)) *f*
using *n* [*OF order_refl f*] *scp f non closure_of_subset* [*OF* ‹*U* ⊆ *topspace X*›] *interior_of_subset* [*of X T*]
by (*fastforce simp*: *image_subset_iff singular_simplex_subtopology singular_chain_def*)
qed
show *?thesis*
unfolding *singular_chain_def* **using** *frag_split* [*OF SS*] **by** *metis*
qed
have (*subtopology X* (*topspace X* ∩ *S*)) = (*subtopology X S*)
by (*metis subtopology_subtopology subtopology_topspace*)
with *assms* **have** *c*: *singular_chain p* (*subtopology X* (*topspace X* ∩ *S*)) *c*
by *simp*
have *Xsub*: *X closure_of* (*topspace X* ∩ *U*) ⊆ *X interior_of* (*topspace X* ∩ *T*)
using *assms closure_of_restrict interior_of_restrict* **by** *fastforce*
obtain *n d e* **where**
d: *singular_chain p* (*subtopology X* (*topspace X* ∩ *S* − *topspace X* ∩ *U*)) *d*
and *e*: *singular_chain p* (*subtopology X* (*topspace X* ∩ *T*)) *e*
and *de*: (*singular_subdivision p* ^^ *n*) *c* = *d* + *e*
using ∗[*OF c Xsub*, *simplified*] *assms* **by** *force*
show *thesis*
proof
show *singular_chain p* (*subtopology X* (*S* − *U*)) *d*
by (*metis d Diff_Int_distrib inf.cobounded2 singular_chain_mono*)
show *singular_chain p* (*subtopology X T*) *e*
by (*metis e inf.cobounded2 singular_chain_mono*)
show (*singular_subdivision p* ^^ *n*) *c* = *d* + *e*
by (*rule de*)
qed
qed

lemma *excised_relcycle_exists*:
fixes *S* :: *′a set*
assumes *X*: *X closure_of U* ⊆ *X interior_of T* **and** *T* ⊆ *S*
and *c*: *singular_relcycle p* (*subtopology X S*) *T c*
obtains *c′* **where** *singular_relcycle p* (*subtopology X* (*S* − *U*)) (*T* − *U*) *c′*
homologous_rel p (*subtopology X S*) *T c c′*
proof −
have [*simp*]: (*S* − *U*) ∩ (*T* − *U*) = *T* − *U* *S* ∩ *T* = *T*

```
    using ‹T ⊆ S› by auto
  have scc: singular_chain p (subtopology X S) c
    and scp1: singular_chain (p - Suc 0) (subtopology X T) (chain_boundary p
c)
   using c by (auto simp: singular_relcycle_def mod_subset_def subtopology_subtopology)
  obtain n d e where d: singular_chain p (subtopology X (S - U)) d
    and e: singular_chain p (subtopology X T) e
    and de: (singular_subdivision p ^^ n) c = d + e
    using excised_chain_exists [OF X ‹T ⊆ S› scc] .
 have scSUd: singular_chain (p - Suc 0) (subtopology X (S - U)) (chain_boundary
p d)
    by (simp add: singular_chain_boundary d)
  have sccn: singular_chain p (subtopology X S) ((singular_subdivision p ^^ n) c)
for n
    by (induction n) (auto simp: singular_chain_singular_subdivision scc)
 have singular_chain (p - Suc 0) (subtopology X T) (chain_boundary p ((singular_subdivision
p ^^ n) c))
  proof (induction n)
    case (Suc n)
    then show ?case
     by (simp add: singular_chain_singular_subdivision chain_boundary_singular_subdivision
[OF sccn])
  qed (auto simp: scp1)
   then have singular_chain (p - Suc 0) (subtopology X T) (chain_boundary p
((singular_subdivision p ^^ n) c - e))
   by (simp add: chain_boundary_diff singular_chain_diff singular_chain_boundary
e)
 with de have scTd: singular_chain (p - Suc 0) (subtopology X T) (chain_boundary
p d)
    by simp
  show thesis
  proof
    have singular_chain (p - Suc 0) X (chain_boundary p d)
      using scTd singular_chain_subtopology by blast
    with scSUd scTd have singular_chain (p - Suc 0) (subtopology X (T - U))
(chain_boundary p d)
      by (fastforce simp add: singular_chain_subtopology)
    then show singular_relcycle p (subtopology X (S - U)) (T - U) d
     by (auto simp: singular_relcycle_def mod_subset_def subtopology_subtopology
d)
    have homologous_rel p (subtopology X S) T (c-0) ((singular_subdivision p ^^
n) c - e)
    proof (rule homologous_rel_diff)
     show homologous_rel p (subtopology X S) T c ((singular_subdivision p ^^ n)
c)
      proof (induction n)
        case (Suc n)
        then show ?case
          apply simp
```

apply (*rule homologous_rel_trans*)
using *c homologous_rel_singular_relcycle_1 homologous_rel_singular_subdivision homologous_rel_sym* **by** *blast*
qed *auto*
show *homologous_rel p* (*subtopology X S*) *T 0 e*
unfolding *homologous_rel_def* **using** *e*
by (*intro singular_relboundary_diff singular_chain_imp_relboundary*; *simp add*: *subtopology_subtopology*)
qed
with *de* **show** *homologous_rel p* (*subtopology X S*) *T c d*
by *simp*
qed
qed

0.1.18 Homotopy invariance

theorem *homotopic_imp_homologous_rel_chain_maps*:
assumes *hom*: *homotopic_with* (λ*h. h ‘ T* ⊆ *V*) *S U f g* **and** *c*: *singular_relcycle p S T c*
shows *homologous_rel p U V* (*chain_map p f c*) (*chain_map p g c*)
proof −
note *sum.atMost_Suc* [*simp del*]
have *contf*: *continuous_map S U f* **and** *contg*: *continuous_map S U g*
using *homotopic_with_imp_continuous_maps* [*OF hom*] **by** *metis+*
obtain *h* **where** *conth*: *continuous_map* (*prod_topology* (*top_of_set* {*0..1*::*real*}) *S*) *U h*
and *h0*: ⋀*x. h*(*0*, *x*) = *f x*
and *h1*: ⋀*x. h*(*1*, *x*) = *g x*
and *hV*: ⋀*t x.* ⟦*0* ≤ *t*; *t* ≤ *1*; *x* ∈ *T*⟧ ⟹ *h*(*t*,*x*) ∈ *V*
using *hom* **by** (*fastforce simp*: *homotopic_with_def*)
define *vv* **where** *vv* ≡ λ*j i. if i* = *Suc j then 1 else* (*0*::*real*)
define *ww* **where** *ww* ≡ λ*j i. if i*=*0* ∨ *i* = *Suc j then 1 else* (*0*::*real*)
define *simp* **where** *simp* ≡ λ*q i. oriented_simplex* (*Suc q*) (λ*j. if j* ≤ *i then vv j else ww*(*j* −*1*))
define *pr* **where** *pr* ≡ λ*q c.* ∑ *i*≤*q. frag_cmul* ((−*1*) ^ *i*)
(*frag_of* (*simplex_map* (*Suc q*) (λ*z. h*(*z 0*, *c*(*z* ∘ *Suc*))) (*simp q i*)))
have *ss_ss*: *simplicial_simplex* (*Suc q*) ({*x. x 0* ∈ {*0..1*} ∧ (*x* ∘ *Suc*) ∈ *standard_simplex q*}) (*simp q i*)
if *i* ≤ *q* **for** *q i*
proof −
have (∑ *j*≤*Suc q.* (*if j* ≤ *i then vv j 0 else ww* (*j* −*1*) *0*) ∗ *x j*) ∈ {*0..1*}
if *x* ∈ *standard_simplex* (*Suc q*) **for** *x*
proof −
have (∑ *j*≤*Suc q. if j* ≤ *i then 0 else x j*) ≤ *sum x* {*..Suc q*}
using *that* **unfolding** *standard_simplex_def*
by (*force intro*!: *sum_mono*)
with ‹*i* ≤ *q*› *that* **show** *?thesis*
by (*simp add*: *vv_def ww_def standard_simplex_def if_distrib* [*of* λ*u. u* ∗

```
_] sum_nonneg cong: if_cong)
    qed
    moreover
    have (λk. ∑j≤Suc q. (if j ≤ i then vv j k else ww (j −1) k) ∗ x j) ∘ Suc ∈
standard_simplex q
      if x ∈ standard_simplex (Suc q) for x
    proof −
      have card: ({..q} ∩ {k. Suc k = j}) = {j−1} if 0 < j j ≤ Suc q for j
        using that by auto
      have eq: (∑j≤Suc q. ∑k≤q. if j ≤ i then if k = j then x j else 0 else if Suc
k = j then x j else 0)
              = (∑j≤Suc q. x j)
        by (rule sum.cong [OF refl]) (use ‹i ≤ q› in ‹simp add: sum.If_cases card›)
      have (∑j≤Suc q. if j ≤ i then if k = j then x j else 0 else if Suc k = j then
x j else 0)
            ≤ sum x {..Suc q} for k
        using that unfolding standard_simplex_def
        by (force intro!: sum_mono)
      then show ?thesis
        using ‹i ≤ q› that
        by (simp add: vv_def ww_def standard_simplex_def if_distrib [of λu. u ∗
_] sum_nonneg
            sum.swap [where A = atMost q] eq cong: if_cong)
    qed
    ultimately show ?thesis
     by (simp add: that simplicial_simplex_oriented_simplex simp_def image_subset_iff
if_distribR)
  qed
  obtain prism where prism: ⋀q. prism q 0 = 0
    ⋀q c. singular_chain q S c ⟹ singular_chain (Suc q) U (prism q c)
    ⋀q c. singular_chain q (subtopology S T) c
                        ⟹ singular_chain (Suc q) (subtopology U V) (prism q c)
    ⋀q c. singular_chain q S c
                        ⟹ chain_boundary (Suc q) (prism q c) =
                            chain_map q g c − chain_map q f c − prism (q −1)
(chain_boundary q c)
  proof
    show (frag_extend ∘ pr) q 0 = 0 for q
      by (simp add: pr_def)
  next
    show singular_chain (Suc q) U ((frag_extend ∘ pr) q c)
      if singular_chain q S c for q c
      using that [unfolded singular_chain_def]
    proof (induction c rule: frag_induction)
      case (one m)
      show ?case
      proof (simp add: pr_def, intro singular_chain_cmul singular_chain_sum)
        fix i :: nat
        assume i ∈ {..q}
```

```
        define X where X = subtopology (powertop_real UNIV) {x. x 0 ∈ {0..1}
∧ (x ∘ Suc) ∈ standard_simplex q}
        show singular_chain (Suc q) U
                (frag_of (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q
i)))
          unfolding singular_chain_of
        proof (rule singular_simplex_simplex_map)
          show singular_simplex (Suc q) X (simp q i)
             unfolding X_def using ‹i ∈ {..q}› simplicial_imp_singular_simplex
ss_ss by blast
          have 0: continuous_map X (top_of_set {0..1}) (λx. x 0)
           unfolding continuous_map_in_subtopology topspace_subtopology X_def
         by (auto intro: continuous_map_product_projection continuous_map_from_subtopology)
          have 1: continuous_map X S (m ∘ (λx j. x (Suc j)))
          proof (rule continuous_map_compose)
              have continuous_map (powertop_real UNIV) (powertop_real UNIV)
(λx j. x (Suc j))
              by (auto intro: continuous_map_product_projection)
               then show continuous_map X (subtopology (powertop_real UNIV)
(standard_simplex q)) (λx j. x (Suc j))
              unfolding X_def o_def
                  by (auto simp: continuous_map_in_subtopology intro: continu-
ous_map_from_subtopology continuous_map_product_projection)
          qed (use one in ‹simp add: singular_simplex_def›)
          show continuous_map X U (λz. h (z 0, m (z ∘ Suc)))
            apply (rule continuous_map_compose [unfolded o_def, OF _ conth])
            using 0 1 by (simp add: continuous_map_pairwise o_def)
        qed
      qed
    next
      case (diff a b)
      then show ?case
        apply (simp add: frag_extend_diff keys_diff)
        using singular_chain_def singular_chain_diff by blast
    qed auto
  next
    show singular_chain (Suc q) (subtopology U V) ((frag_extend ∘ pr) q c)
      if singular_chain q (subtopology S T) c for q c
      using that [unfolded singular_chain_def]
    proof (induction c rule: frag_induction)
      case (one m)
      show ?case
      proof (simp add: pr_def, intro singular_chain_cmul singular_chain_sum)
        fix i :: nat
        assume i ∈ {..q}
        define X where X = subtopology (powertop_real UNIV) {x. x 0 ∈ {0..1}
∧ (x ∘ Suc) ∈ standard_simplex q}
        show singular_chain (Suc q) (subtopology U V)
                (frag_of (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q
```

```
i)))
          unfolding singular_chain_of
        proof (rule singular_simplex_simplex_map)
          show singular_simplex (Suc q) X (simp q i)
            unfolding X_def using ‹i ∈ {..q}› simplicial_imp_singular_simplex
ss_ss by blast
          have 0: continuous_map X (top_of_set {0..1}) (λx. x 0)
            unfolding continuous_map_in_subtopology topspace_subtopology X_def
          by (auto intro: continuous_map_product_projection continuous_map_from_subtopology)
          have 1: continuous_map X (subtopology S T) (m ∘ (λx j. x (Suc j)))
          proof (rule continuous_map_compose)
            have continuous_map (powertop_real UNIV) (powertop_real UNIV)
(λx j. x (Suc j))
              by (auto intro: continuous_map_product_projection)
            then show continuous_map X (subtopology (powertop_real UNIV)
(standard_simplex q)) (λx j. x (Suc j))
              unfolding X_def o_def
                by (auto simp: continuous_map_in_subtopology intro: continu-
ous_map_from_subtopology continuous_map_product_projection)
          show continuous_map (subtopology (powertop_real UNIV) (standard_simplex
q)) (subtopology S T) m
              using one continuous_map_into_fulltopology by (auto simp: singu-
lar_simplex_def)
          qed
          have continuous_map X (subtopology U V) (h ∘ (λz. (z 0, m (z ∘ Suc))))
          proof (rule continuous_map_compose)
              show continuous_map X (prod_topology (top_of_set {0..1::real})
(subtopology S T)) (λz. (z 0, m (z ∘ Suc)))
              using 0 1 by (simp add: continuous_map_pairwise o_def)
              have continuous_map (subtopology (prod_topology euclideanreal S)
({0..1} × T)) U h
              by (metis conth continuous_map_from_subtopology subtopology_Times
subtopology_topspace)
          with hV show continuous_map (prod_topology (top_of_set {0..1::real})
(subtopology S T)) (subtopology U V) h
              by (force simp: topspace_subtopology continuous_map_in_subtopology
subtopology_restrict subtopology_Times)
          qed
          then show continuous_map X (subtopology U V) (λz. h (z 0, m (z ∘
Suc)))
            by (simp add: o_def)
        qed
      qed
    next
      case (diff a b)
      then show ?case
        by (metis comp_apply frag_extend_diff singular_chain_diff)
    qed auto
  next
```

show *chain_boundary* (*Suc q*) ((*frag_extend* ∘ *pr*) *q c*) =
chain_map q g c − *chain_map q f c* − (*frag_extend* ∘ *pr*) (*q* −*1*) (*chain_boundary q c*)
if *singular_chain q S c* **for** *q c*
using *that* [*unfolded singular_chain_def*]
proof (*induction c rule*: *frag_induction*)
case (*one m*)
have *eq2*: *Sigma S T* = (λ*i*. (*i*,*i*)) ' {*i* ∈ *S*. *i* ∈ *T i*} ∪ (*Sigma S* (λ*i*. *T i* − {*i*})) **for** *S* :: *nat set* **and** *T*
by *force*
have *1*: (∑(*i*,*j*)∈(λ*i*. (*i*, *i*)) ' {*i*. *i* ≤ *q* ∧ *i* ≤ *Suc q*}.
frag_cmul (((−*1*) ^ *i*) ∗ (−*1*) ^ *j*)
(*frag_of*
(*singular_face* (*Suc q*) *j*
(*simplex_map* (*Suc q*) (λ*z*. *h* (*z 0*, *m* (*z* ∘ *Suc*))) (*simp q i*)))))
+ (∑(*i*,*j*)∈(λ*i*. (*i*, *i*)) ' {*i*. *i* ≤ *q*}.
frag_cmul (− ((−*1*) ^ *i* ∗ (−*1*) ^ *j*))
(*frag_of*
(*singular_face* (*Suc q*) (*Suc j*)
(*simplex_map* (*Suc q*) (λ*z*. *h* (*z 0*, *m* (*z* ∘ *Suc*))) (*simp q i*)))))
= *frag_of* (*simplex_map q g m*) − *frag_of* (*simplex_map q f m*)
proof −
have *restrict* ((λ*z*. *h* (*z 0*, *m* (*z* ∘ *Suc*))) ∘ (*simp q 0* ∘ *simplical_face 0*)) (*standard_simplex q*)
= *restrict* (*g* ∘ *m*) (*standard_simplex q*)
proof (*rule restrict_ext*)
fix *x*
assume *x*: *x* ∈ *standard_simplex q*
have (∑*j*≤*Suc q*. *if j* = *0 then 0 else x* (*j* − *Suc 0*)) = (∑*j*≤*q*. *x j*)
by (*simp add*: *sum.atMost_Suc_shift*)
with *x* **have** *simp q 0* (*simplical_face 0 x*) *0* = *1*
apply (*simp add*: *oriented_simplex_def simp_def simplical_face_in_standard_simplex*)
apply (*simp add*: *simplical_face_def if_distrib ww_def standard_simplex_def cong*: *if_cong*)
done
moreover
have (λ*n*. *if n* ≤ *q then x n else 0*) = *x*
using *standard_simplex_def x* **by** *auto*
then have (λ*n*. *simp q 0* (*simplical_face 0 x*) (*Suc n*)) = *x*
unfolding *oriented_simplex_def simp_def ww_def* **using** *x*
apply (*simp add*: *simplical_face_in_standard_simplex*)
apply (*simp add*: *simplical_face_def if_distrib*)
apply (*simp add*: *if_distribR if_distrib cong*: *if_cong*)
done
ultimately show ((λ*z*. *h* (*z 0*, *m* (*z* ∘ *Suc*))) ∘ (*simp q 0* ∘ *simplical_face 0*)) *x* = (*g* ∘ *m*) *x*
by (*simp add*: *o_def h1*)
qed

then have *a*: *frag_of* (*singular_face* (*Suc q*) *0* (*simplex_map* (*Suc q*) (λz. *h* (*z 0*, *m* (*z* ∘ *Suc*))) (*simp q 0*)))
= *frag_of* (*simplex_map q g m*)
by (*simp add*: *singular_face_simplex_map*) (*simp add*: *simplex_map_def*)
have *restrict* ((λz. *h* (*z 0*, *m* (*z* ∘ *Suc*))) ∘ (*simp q q* ∘ *simplical_face* (*Suc q*))) (*standard_simplex q*)
= *restrict* (*f* ∘ *m*) (*standard_simplex q*)
proof (*rule restrict_ext*)
fix *x*
assume *x*: *x* ∈ *standard_simplex q*
then have *simp q q* (*simplical_face* (*Suc q*) *x*) *0* = *0*
unfolding *oriented_simplex_def simp_def*
by (*simp add*: *simplical_face_in_standard_simplex sum.atMost_Suc*) (*simp add*: *simplical_face_def vv_def*)
moreover have (λn. *simp q q* (*simplical_face* (*Suc q*) *x*) (*Suc n*)) = *x*
unfolding *oriented_simplex_def simp_def vv_def* **using** *x*
apply (*simp add*: *simplical_face_in_standard_simplex*)
apply (*force simp*: *standard_simplex_def simplical_face_def if_distribR if_distrib* [*of* λx. *x* ∗ _] *sum.atMost_Suc cong*: *if_cong*)
done
ultimately show ((λz. *h* (*z 0*, *m* (*z* ∘ *Suc*))) ∘ (*simp q q* ∘ *simplical_face* (*Suc q*))) *x* = (*f* ∘ *m*) *x*
by (*simp add*: *o_def h0*)
qed
then have *b*: *frag_of* (*singular_face* (*Suc q*) (*Suc q*)
(*simplex_map* (*Suc q*) (λz. *h* (*z 0*, *m* (*z* ∘ *Suc*))) (*simp q q*)))
= *frag_of* (*simplex_map q f m*)
by (*simp add*: *singular_face_simplex_map*) (*simp add*: *simplex_map_def*)
have *sfeq*: *simplex_map q* (λz. *h* (*z 0*, *m* (*z* ∘ *Suc*))) (*simp q* (*Suc i*) ∘ *simplical_face* (*Suc i*))
= *simplex_map q* (λz. *h* (*z 0*, *m* (*z* ∘ *Suc*))) (*simp q i* ∘ *simplical_face* (*Suc i*))
if *i* < *q* **for** *i*
unfolding *simplex_map_def*
proof (*rule restrict_ext*)
fix *x*
assume *x* ∈ *standard_simplex q*
then have (*simp q* (*Suc i*) ∘ *simplical_face* (*Suc i*)) *x* = (*simp q i* ∘ *simplical_face* (*Suc i*)) *x*
unfolding *oriented_simplex_def simp_def simplical_face_def*
by (*force intro*: *sum.cong*)
then show ((λz. *h* (*z 0*, *m* (*z* ∘ *Suc*))) ∘ (*simp q* (*Suc i*) ∘ *simplical_face* (*Suc i*))) *x*
= ((λz. *h* (*z 0*, *m* (*z* ∘ *Suc*))) ∘ (*simp q i* ∘ *simplical_face* (*Suc i*))) *x*
by *simp*
qed
have *eqq*: {*i*. *i* ≤ *q* ∧ *i* ≤ *Suc q*} = {..*q*}
by *force*
have *qeq*: {..*q*} = *insert 0* ((λi. *Suc i*) ‘ {*i*. *i* < *q*}) {*i*. *i* ≤ *q*} = *insert q*

```
{i. i < q}
          using le_imp_less_Suc less_Suc_eq_0_disj by auto
        show ?thesis
          using a b
          apply (simp add: sum.reindex inj_on_def eqq)
       apply (simp add: qeq sum.insert_if sum.reindex sum_negf singular_face_simplex_map
sfeq)
          done
      qed
      have 2: (∑(i,j)∈(SIGMA i:{..q}. {0..min (Suc q) i} − {i}).
                    frag_cmul ((−1) ^ i ∗ (−1) ^ j)
                     (frag_of
                       (singular_face (Suc q) j
                       (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q i)))))
               + (∑(i,j)∈(SIGMA i:{..q}. {i..q} − {i}).
                  frag_cmul (− ((−1) ^ i ∗ (−1) ^ j))
                   (frag_of
                     (singular_face (Suc q) (Suc j)
                       (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q i)))))
               = − frag_extend (pr (q − Suc 0)) (chain_boundary q (frag_of m))
      proof (cases q=0)
        case True
        then show ?thesis
          by (simp add: chain_boundary_def flip: sum.Sigma)
      next
        case False
        have eq: {..q − Suc 0} × {..q} = Sigma {..q − Suc 0} (λi. {0..min q i})
∪ Sigma {..q} (λi. {i<..q})
          by force
        have I: (∑(i,j)∈(SIGMA i:{..q}. {0..min (Suc q) i} − {i}).
                  frag_cmul ((−1) ^ (i + j))
                     (frag_of
                       (singular_face (Suc q) j
                       (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q i)))))
              = (∑(i,j)∈(SIGMA i:{..q − Suc 0}. {0..min q i}).
                  frag_cmul (− ((−1) ^ (j + i)))
                   (frag_of
                     (simplex_map q (λz. h (z 0, singular_face q j m (z ∘ Suc)))
                       (simp (q − Suc 0) i))))
        proof −
          have seq: simplex_map q (λz. h (z 0, singular_face q j m (z ∘ Suc)))
                     (simp (q − Suc 0) (i − Suc 0))
                = simplex_map q (λz. h (z 0, m (z ∘ Suc))) (simp q i ∘ simplical_face
j)
            if ij: i ≤ q j ≠ i j ≤ i for i j
            unfolding simplex_map_def
          proof (rule restrict_ext)
            fix x
            assume x: x ∈ standard_simplex q
```

```
          have i > 0
            using that by force
          then have iq: i − Suc 0 ≤ q − Suc 0
            using ‹i ≤ q› False by simp
          have q0_eq: {..Suc q} = insert 0 (Suc ‘ {..q})
            by (auto simp: image_def gr0_conv_Suc)
          have α: simp (q − Suc 0) (i − Suc 0) x 0 = simp q i (simplical_face j
x) 0
            using False x ij
            unfolding oriented_simplex_def simp_def vv_def ww_def
            apply (simp add: simplical_face_in_standard_simplex)
                apply (force simp: simplical_face_def q0_eq sum.reindex intro!:
sum.cong)
            done
         have β: simplical_face j (simp (q − Suc 0) (i − Suc 0) x ∘ Suc) = simp
q i (simplical_face j x) ∘ Suc
          proof
            fix k
            show simplical_face j (simp (q − Suc 0) (i − Suc 0) x ∘ Suc) k
                = (simp q i (simplical_face j x) ∘ Suc) k
              using False x ij
              unfolding oriented_simplex_def simp_def o_def vv_def ww_def
              apply (simp add: simplical_face_in_standard_simplex if_distribR)
               apply (simp add: simplical_face_def if_distrib [of λu. u ∗ _] cong:
if_cong)
              apply (intro impI conjI)
               apply (force simp: sum.atMost_Suc intro: sum.cong)
              apply (force simp: q0_eq sum.reindex intro!: sum.cong)
              done
          qed
          have simp (q − Suc 0) (i − Suc 0) x ∘ Suc ∈ standard_simplex (q −
Suc 0)
            using ss_ss [OF iq] ‹i ≤ q› False ‹i > 0›
            apply (simp add: simplicial_simplex image_subset_iff)
            using ‹x ∈ standard_simplex q› by blast
          then show ((λz. h (z 0, singular_face q j m (z ∘ Suc))) ∘ simp (q −
Suc 0) (i − Suc 0)) x
              = ((λz. h (z 0, m (z ∘ Suc))) ∘ (simp q i ∘ simplical_face j)) x
            by (simp add: singular_face_def α β)
        qed
        have [simp]: (−1::int) ^(i + j − Suc 0) = − ((−1) ^(i + j)) if i ≠ j
for i j::nat
        proof −
          have i + j > 0
            using that by blast
          then show ?thesis
                by (metis (no_types, opaque_lifting) One_nat_def Suc_diff_1
add.inverse_inverse mult.left_neutral mult_minus_left power_Suc)
        qed
```

```
          show ?thesis
            apply (rule sum.eq_general_inverses [where h = λ(a,b). (a−1,b) and
k = λ(a,b). (Suc a,b)])
         using False apply (auto simp: singular_face_simplex_map seq add.commute)
            done
        qed
       have *: singular_face (Suc q) (Suc j) (simplex_map (Suc q) (λz. h (z 0, m
(z ∘ Suc))) (simp q i))
              = simplex_map q (λz. h (z 0, singular_face q j m (z ∘ Suc))) (simp
(q − Suc 0) i)
          if ij: i < j j ≤ q for i j
        proof −
          have iq: i ≤ q − Suc 0
            using that by auto
          have sf_eqh: singular_face (Suc q) (Suc j)
                       (λx. if x ∈ standard_simplex (Suc q)
                            then ((λz. h (z 0, m (z ∘ Suc))) ∘ simp q i) x else
undefined) x
                    = h (simp (q − Suc 0) i x 0,
                       singular_face q j m (λxa. simp (q − Suc 0) i x (Suc xa)))
            if x: x ∈ standard_simplex q for x
          proof −
            let ?f = λk. ∑j≤q. if j ≤ i then if k = j then x j else 0
                          else if Suc k = j then x j else 0
            have fm: simplical_face (Suc j) x ∈ standard_simplex (Suc q)
              using ss_ss [OF iq] that ij
              by (simp add: simplical_face_in_standard_simplex)
            have ss: ?f ∈ standard_simplex (q − Suc 0)
              unfolding standard_simplex_def
            proof (intro CollectI conjI impI allI)
              fix k
              show 0 ≤ ?f k
                using that by (simp add: sum_nonneg standard_simplex_def)
              show ?f k ≤ 1
                using x sum_le_included [of {..q} {..q} x id]
                by (simp add: standard_simplex_def)
              assume k: q − Suc 0 < k
              show ?f k = 0
               by (rule sum.neutral) (use that x iq k standard_simplex_def in auto)
            next
              have (∑k≤q − Suc 0. ?f k)
                  = (∑(k,j) ∈ ({..q − Suc 0} × {..q}) ∩ {(k,j). if j ≤ i then k = j
else Suc k = j}. x j)
                apply (simp add: sum.Sigma)
                by (rule sum.mono_neutral_cong) (auto simp: split: if_split_asm)
              also have ... = sum x {..q}
                apply (rule sum.eq_general_inverses
                   [where h = λ(k,j). if j≤i ∧ k=j ∨ j>i ∧ Suc k = j then j else Suc
q
```

```
            and k = λj. if j ≤ i then (j,j) else (j − Suc 0, j)])
        using ij by auto
      also have ... = 1
        using x by (simp add: standard_simplex_def)
      finally show (∑k≤q − Suc 0. ?f k) = 1
        by (simp add: standard_simplex_def)
    qed
    let ?g = λk. if k ≤ i then 0
                  else if k < Suc j then x k
                       else if k = Suc j then 0 else x (k − Suc 0)
     have eq: {..Suc q} = {..j} ∪ {Suc j} ∪ Suc‘{j<..q} {..q} = {..j} ∪
{j<..q}
      using ij image_iff less_Suc_eq_0_disj less_Suc_eq_le
      by (force simp: image_iff)+
     then have (∑k≤Suc q. ?g k) = (∑k∈{..j} ∪ {Suc j} ∪ Suc‘{j<..q}.
?g k)
      by simp
    also have ... = (∑k∈{..j} ∪ Suc‘{j<..q}. ?g k)
      by (rule sum.mono_neutral_right) auto
    also have ... = (∑k∈{..j}. ?g k) + (∑k∈Suc‘{j<..q}. ?g k)
      by (rule sum.union_disjoint) auto
    also have ... = (∑k∈{..j}. ?g k) + (∑k∈{j<..q}. ?g (Suc k))
      by (auto simp: sum.reindex)
    also have ... = (∑k∈{..j}. if k ≤ i then 0 else x k)
                  + (∑k∈{j<..q}. if k ≤ i then 0 else x k)
      by (intro sum.cong arg_cong2 [of concl: (+)]) (use ij in auto)
    also have ... = (∑k≤q. if k ≤ i then 0 else x k)
      unfolding eq by (subst sum.union_disjoint) auto
    finally have (∑k≤Suc q. ?g k) = (∑k≤q. if k ≤ i then 0 else x k) .
     then have QQ: (∑l≤Suc q. if l ≤ i then 0 else simplical_face (Suc j)
x l) = (∑j≤q. if j ≤ i then 0 else x j)
      by (simp add: simplical_face_def cong: if_cong)
    have WW: (λk. ∑l≤Suc q. if l ≤ i
                       then if k = l then simplical_face (Suc j) x l else 0
                       else if Suc k = l then simplical_face (Suc j) x l
                       else 0)
        = simplical_face j
          (λk. ∑j≤q. if j ≤ i then if k = j then x j else 0
                    else if Suc k = j then x j else 0)
    proof −
        have *: (∑l≤q. if l ≤ i then 0 else if Suc k = l then x (l − Suc 0)
else 0)
          = (∑l≤q. if l ≤ i then if k − Suc 0 = l then x l else 0 else if k =
l then x l else 0)
        (is ?lhs = ?rhs)
        if k ≠ q k > j for k
      proof (cases k ≤ q)
        case True
        have ?lhs = sum (λl. x (l − Suc 0)) {Suc k} ?rhs = sum x {k}
```

```
            by (rule sum.mono_neutral_cong_right; use True ij that in auto)+
          then show ?thesis
            by simp
        next
          case False
          have ?lhs = 0 ?rhs = 0
            by (rule sum.neutral; use False ij in auto)+
          then show ?thesis
            by simp
        qed
        show ?thesis
          apply (rule ext)
          unfolding simplical_face_def using ij
          apply (auto simp: sum.atMost_Suc cong: if_cong)
           apply (force simp flip: ivl_disj_un(2) intro: sum.neutral)
           apply (auto simp: *)
          done
      qed
      show ?thesis
        using False that iq
        unfolding oriented_simplex_def simp_def vv_def ww_def
        apply (simp add: if_distribR cong: if_cong)
         apply (simp add: simplical_face_def if_distrib [of λu. u * _] o_def
cong: if_cong)
        apply (simp add: singular_face_def fm ss QQ WW)
        done
    qed
    show ?thesis
      unfolding simplex_map_def restrict_def
        apply (simp add: simplicial_simplex image_subset_iff o_def sf_eqh
fun_eq_iff)
      apply (simp add: singular_face_def)
      done
  qed
  have sgeq: (SIGMA i:{..q}. {i..q} − {i})  = (SIGMA i:{..q}. {i<..q})
    by force
  have II: (∑ (i,j)∈(SIGMA i:{..q}. {i..q} − {i}).
             frag_cmul (− ((−1) ^ (i + j)))
              (frag_of
                (singular_face (Suc q) (Suc j)
                    (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q
i))))) =
           (∑ (i,j)∈(SIGMA i:{..q}. {i<..q}).
             frag_cmul (− ((−1) ^ (j + i)))
              (frag_of
                (simplex_map q (λz. h (z 0, singular_face q j m (z ∘ Suc)))
                  (simp (q − Suc 0) i))))
    by (force simp: * sgeq add.commute intro: sum.cong)
  show ?thesis
```

```
          using False
        apply (simp add: chain_boundary_def frag_extend_sum frag_extend_cmul
frag_cmul_sum pr_def flip: sum_negf power_add)
          apply (subst sum.swap [where A = {..q}])
             apply (simp add: sum.cartesian_product eq sum.union_disjoint dis-
joint_iff_not_equal I II)
          done
       qed
       have *: ⟦a+b = w; c+d = −z⟧ ⟹ (a + c) + (b+d) = w−z for a b w c d z
:: 'c ⇒₀ int
         by (auto simp: algebra_simps)
       have eq: {..q} × {..Suc q} =
                 Sigma {..q} (λi. {0..min (Suc q) i})
               ∪ Sigma {..q} (λi. {Suc i..Suc q})
         by force
       show ?case
         apply (subst pr_def)
         apply (simp add: chain_boundary_sum chain_boundary_cmul)
         apply (subst chain_boundary_def)
         apply simp
       apply (simp add: frag_cmul_sum sum.cartesian_product eq sum.union_disjoint
disjoint_iff_not_equal
             sum.atLeast_Suc_atMost_Suc_shift del: sum.cl_ivl_Suc min.absorb2
min.absorb4
           flip: comm_monoid_add_class.sum.Sigma)
         apply (simp add: sum.Sigma eq2 [of _ λi. {_ i.._ i}]
           del: min.absorb2 min.absorb4)
         apply (simp add: sum.union_disjoint disjoint_iff_not_equal * [OF 1 2])
         done
     next
       case (diff a b)
       then show ?case
         by (simp add: chain_boundary_diff frag_extend_diff chain_map_diff)
     qed auto
   qed
  have *: singular_chain p (subtopology U V) (prism (p − Suc 0) (chain_boundary
p c))
    if singular_chain p S c singular_chain (p − Suc 0) (subtopology S T) (chain_boundary
p c)
   proof (cases p)
     case 0 then show ?thesis by (simp add: chain_boundary_def prism)
   next
     case (Suc p')
     with prism that show ?thesis by auto
   qed
   then show ?thesis
     using c
    unfolding singular_relcycle_def homologous_rel_def singular_relboundary_def
mod_subset_def
```

apply (*rule_tac x*=− *prism p c* **in** *exI*)
by (*simp add*: *chain_boundary_minus prism*(*2*) *prism*(*4*) *singular_chain_minus*)
qed

end

0.2 Homology, II: Homology Groups

theory *Homology_Groups*
imports *Simplices HOL*−*Algebra.Exact_Sequence*

begin

0.2.1 Homology Groups

Now actually connect to group theory and set up homology groups. Note that we define homomogy groups for all *integers p*, since this seems to avoid some special-case reasoning, though they are trivial for $p < (0::'a)$.

definition *chain_group* :: *nat* ⇒ *'a topology* ⇒ *'a chain monoid*
where *chain_group p X* ≡ *free_Abelian_group* (*singular_simplex_set p X*)

lemma *carrier_chain_group* [*simp*]: *carrier*(*chain_group p X*) = *singular_chain_set p X*
by (*auto simp*: *chain_group_def singular_chain_def free_Abelian_group_def*)

lemma *one_chain_group* [*simp*]: *one*(*chain_group p X*) = *0*
by (*auto simp*: *chain_group_def free_Abelian_group_def*)

lemma *mult_chain_group* [*simp*]: *monoid.mult*(*chain_group p X*) = (+)
by (*auto simp*: *chain_group_def free_Abelian_group_def*)

lemma *m_inv_chain_group* [*simp*]: *Poly_Mapping.keys a* ⊆ *singular_simplex_set p X* ⟹ $inv_{chain\_group\ p\ X}\ a = -a$
unfolding *chain_group_def* **by** *simp*

lemma *group_chain_group* [*simp*]: *Group.group* (*chain_group p X*)
by (*simp add*: *chain_group_def*)

lemma *abelian_chain_group*: *comm_group*(*chain_group p X*)
by (*simp add*: *free_Abelian_group_def group.group_comm_groupI* [*OF group_chain_group*])

lemma *subgroup_singular_relcycle*:
subgroup (*singular_relcycle_set p X S*) (*chain_group p X*)
proof
show $x \otimes_{chain\_group\ p\ X} y$ ∈ *singular_relcycle_set p X S*
if *x* ∈ *singular_relcycle_set p X S* **and** *y* ∈ *singular_relcycle_set p X S* **for** *x y*
using *that* **by** (*simp add*: *singular_relcycle_add*)

```
next
  show invchain_group p X x ∈ singular_relcycle_set p X S
    if x ∈ singular_relcycle_set p X S for x
    using that
    by clarsimp (metis m_inv_chain_group singular_chain_def singular_relcycle
singular_relcycle_minus)
qed (auto simp: singular_relcycle)


definition relcycle_group :: nat ⇒ 'a topology ⇒ 'a set ⇒ ('a chain) monoid
  where relcycle_group p X S ≡
      subgroup_generated (chain_group p X) (Collect(singular_relcycle p X S))

lemma carrier_relcycle_group [simp]:
  carrier (relcycle_group p X S) = singular_relcycle_set p X S
proof −
 have carrier (chain_group p X) ∩ singular_relcycle_set p X S = singular_relcycle_set
p X S
    using subgroup.subset subgroup_singular_relcycle by blast
  moreover have generate (chain_group p X) (singular_relcycle_set p X S) ⊆
singular_relcycle_set p X S
   by (simp add: group.generate_subgroup_incl group_chain_group subgroup_singular_relcycle)
  ultimately show ?thesis
    by (auto simp: relcycle_group_def subgroup_generated_def generate.incl)
qed

lemma one_relcycle_group [simp]: one(relcycle_group p X S) = 0
  by (simp add: relcycle_group_def)

lemma mult_relcycle_group [simp]: (⊗relcycle_group p X S) = (+)
  by (simp add: relcycle_group_def)

lemma abelian_relcycle_group [simp]:
   comm_group(relcycle_group p X S)
  unfolding relcycle_group_def
   by (intro group.abelian_subgroup_generated group_chain_group) (auto simp:
abelian_chain_group singular_relcycle)

lemma group_relcycle_group [simp]: group(relcycle_group p X S)
  by (simp add: comm_group.axioms(2))

lemma relcycle_group_restrict [simp]:
   relcycle_group p X (topspace X ∩ S) = relcycle_group p X S
  by (metis relcycle_group_def singular_relcycle_restrict)


definition relative_homology_group :: int ⇒ 'a topology ⇒ 'a set ⇒ ('a chain)
set monoid
  where
```

relative_homology_group p X S ≡
if p < 0 then singleton_group undefined else
(relcycle_group (nat p) X S) Mod (singular_relboundary_set (nat p) X S)

abbreviation *homology_group*
where *homology_group p X ≡ relative_homology_group p X {}*

lemma *relative_homology_group_restrict* [*simp*]:
relative_homology_group p X (topspace X ∩ S) = relative_homology_group p X S
by (*simp add*: *relative_homology_group_def*)

lemma *nontrivial_relative_homology_group*:
fixes *p*::*nat*
shows *relative_homology_group p X S*
= relcycle_group p X S Mod singular_relboundary_set p X S
by (*simp add*: *relative_homology_group_def*)

lemma *singular_relboundary_ss*:
singular_relboundary p X S x ⟹ Poly_Mapping.keys x ⊆ singular_simplex_set p X
using *singular_chain_def singular_relboundary_imp_chain* **by** *blast*

lemma *trivial_relative_homology_group* [*simp*]:
p < 0 ⟹ trivial_group(relative_homology_group p X S)
by (*simp add*: *relative_homology_group_def*)

lemma *subgroup_singular_relboundary*:
subgroup (singular_relboundary_set p X S) (chain_group p X)
unfolding *chain_group_def*
proof *unfold_locales*
show *singular_relboundary_set p X S*
⊆ carrier (free_Abelian_group (singular_simplex_set p X))
using *singular_chain_def singular_relboundary_imp_chain* **by** *fastforce*
next
fix *x*
assume *x ∈ singular_relboundary_set p X S*
then show $inv_{free\_Abelian\_group\ (singular\_simplex\_set\ p\ X)}\ x$
∈ singular_relboundary_set p X S
by (*simp add*: *singular_relboundary_ss singular_relboundary_minus*)
qed (*auto simp*: *free_Abelian_group_def singular_relboundary_add*)

lemma *subgroup_singular_relboundary_relcycle*:
subgroup (singular_relboundary_set p X S) (relcycle_group p X S)
unfolding *relcycle_group_def*
apply (*rule group.subgroup_of_subgroup_generated*)
by (*auto simp*: *singular_relcycle subgroup_singular_relboundary intro*: *singular_relboundary_imp_relcycle*)

lemma *normal_subgroup_singular_relboundary_relcycle*:
(*singular_relboundary_set p X S*) ◁ (*relcycle_group p X S*)
by (*simp add*: *comm_group.normal_iff_subgroup subgroup_singular_relboundary_relcycle*)

lemma *group_relative_homology_group* [*simp*]:
group (*relative_homology_group p X S*)
by (*simp add*: *relative_homology_group_def normal.factorgroup_is_group*
normal_subgroup_singular_relboundary_relcycle)

lemma *right_coset_singular_relboundary*:
r_coset (*relcycle_group p X S*) (*singular_relboundary_set p X S*)
= (λ*a*. {*b*. *homologous_rel p X S a b*})
using *singular_relboundary_minus*
by (*force simp*: *r_coset_def homologous_rel_def relcycle_group_def subgroup_generated_def*)

lemma *carrier_relative_homology_group*:
carrier(*relative_homology_group* (*int p*) *X S*)
= (*homologous_rel_set p X S*) ‘ *singular_relcycle_set p X S*
by (*auto simp*: *set_eq_iff image_iff relative_homology_group_def FactGroup_def*
RCOSETS_def right_coset_singular_relboundary)

lemma *carrier_relative_homology_group_0*:
carrier(*relative_homology_group 0 X S*)
= (*homologous_rel_set 0 X S*) ‘ *singular_relcycle_set 0 X S*
using *carrier_relative_homology_group* [*of 0 X S*] **by** *simp*

lemma *one_relative_homology_group* [*simp*]:
one(*relative_homology_group* (*int p*) *X S*) = *singular_relboundary_set p X S*
by (*simp add*: *relative_homology_group_def FactGroup_def*)

lemma *mult_relative_homology_group*:
$(\otimes_{relative\_homology\_group\ (int\ p)\ X\ S}) = (\lambda R\ S.\ (\bigcup r{\in}R.\ \bigcup s{\in}S.\ \{r + s\}))$
unfolding *relcycle_group_def subgroup_generated_def chain_group_def free_Abelian_group_def*
set_mult_def relative_homology_group_def FactGroup_def
by *force*

lemma *inv_relative_homology_group*:
assumes *R* ∈ *carrier* (*relative_homology_group* (*int p*) *X S*)
shows *m_inv*(*relative_homology_group* (*int p*) *X S*) *R* = *uminus* ‘ *R*
proof (*rule group.inv_equality* [*OF group_relative_homology_group _ assms*])
obtain *c* **where** *c*: *R* = *homologous_rel_set p X S c singular_relcycle p X S c*
using *assms* **by** (*auto simp*: *carrier_relative_homology_group*)
have *singular_relboundary p X S* (*b* − *a*)
if *a* ∈ *R* **and** *b* ∈ *R* **for** *a b*
using *c that*
by *clarify* (*metis homologous_rel_def homologous_rel_eq*)
moreover
have $x \in (\bigcup x{\in}R.\ \bigcup y{\in}R.\ \{y - x\})$
if *singular_relboundary p X S x* **for** *x*

using *c*
by *simp* (*metis diff_eq_eq homologous_rel_def homologous_rel_refl homologous_rel_sym that*)
ultimately
have $(\bigcup x{\in}R.\ \bigcup xa{\in}R.\ \{xa - x\})$ = *singular_relboundary_set p X S*
by *auto*
then show *uminus* ' $R \otimes_{relative\_homology\_group\ (int\ p)\ X\ S} R = \mathbf{1}_{relative\_homology\_group\ (int\ p)\ X\ S}$
by (*auto simp*: *carrier_relative_homology_group mult_relative_homology_group*)
have *singular_relcycle p X S* (−*c*)
using *c* **by** (*simp add*: *singular_relcycle_minus*)
moreover have *homologous_rel p X S c x* ⟹ *homologous_rel p X S* (−*c*) (−*x*) **for** *x*
by (*metis homologous_rel_def homologous_rel_sym minus_diff_eq minus_diff_minus*)
moreover have *homologous_rel p X S* (−*c*) *x* ⟹ *x* ∈ *uminus* ' *homologous_rel_set p X S c* **for** *x*
by (*clarsimp simp*: *image_iff*) (*metis add.inverse_inverse diff_0 homologous_rel_diff homologous_rel_refl*)
ultimately show *uminus* ' *R* ∈ *carrier* (*relative_homology_group* (*int p*) *X S*)
using *c* **by** (*auto simp*: *carrier_relative_homology_group*)
qed

lemma *homologous_rel_eq_relboundary*:
homologous_rel p X S c = *singular_relboundary p X S* ⟷ *singular_relboundary p X S c* (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
unfolding *homologous_rel_def*
by (*metis diff_zero singular_relboundary_0*)
next
assume *R*: *?rhs*
show *?lhs*
unfolding *homologous_rel_def*
using *singular_relboundary_diff R* **by** *fastforce*
qed

lemma *homologous_rel_set_eq_relboundary*:
homologous_rel_set p X S c = *singular_relboundary_set p X S* ⟷ *singular_relboundary p X S c*
by (*auto simp flip*: *homologous_rel_eq_relboundary*)

Lift the boundary and induced maps to homology groups. We totalize both quite aggressively to the appropriate group identity in all "undefined" situations, which makes several of the properties cleaner and simpler.

lemma *homomorphism_chain_boundary*:
chain_boundary p ∈ *hom* (*relcycle_group p X S*) (*relcycle_group*(*p* − *Suc 0*) (*subtopology X S*) {})
(**is** *?h* ∈ *hom ?G ?H*)

```
proof (rule homI)
  show ⋀x. x ∈ carrier ?G ⟹ ?h x ∈ carrier ?H
   by (auto simp: singular_relcycle_def mod_subset_def chain_boundary_boundary)
qed (simp add: relcycle_group_def subgroup_generated_def chain_boundary_add)


lemma hom_boundary1:
    ∃ d. ∀ p X S.
          d p X S ∈ hom (relative_homology_group (int p) X S)
                         (homology_group (int (p − Suc 0)) (subtopology X S))
       ∧ (∀ c. singular_relcycle p X S c
             ⟶ d p X S (homologous_rel_set p X S c)
            = homologous_rel_set (p − Suc 0) (subtopology X S) {} (chain_boundary
p c))
    (is ∃ d. ∀ p X S. ?Φ (d p X S) p X S)
proof ((subst choice_iff [symmetric])+, clarify)
  fix p X and S :: 'a set
  define ϑ where ϑ ≡ r_coset (relcycle_group(p − Suc 0) (subtopology X S) {})
                      (singular_relboundary_set (p − Suc 0) (subtopology X S) {}) ∘
chain_boundary p
  define H where H ≡ relative_homology_group (int (p − Suc 0)) (subtopology
X S) {}
  define J where J ≡ relcycle_group (p − Suc 0) (subtopology X S) {}

  have ϑ: ϑ ∈ hom (relcycle_group p X S) H
    unfolding ϑ_def
  proof (rule hom_compose)
    show chain_boundary p ∈ hom (relcycle_group p X S) J
      by (simp add: J_def homomorphism_chain_boundary)
    show (#>relcycle_group (p − Suc 0) (subtopology X S) {})
         (singular_relboundary_set (p − Suc 0) (subtopology X S) {}) ∈ hom J H
      by (simp add: H_def J_def nontrivial_relative_homology_group
        normal.r_coset_hom_Mod normal_subgroup_singular_relboundary_relcycle)
  qed
  have ∗: singular_relboundary (p − Suc 0) (subtopology X S) {} (chain_boundary
p c)
    if singular_relboundary p X S c for c
  proof (cases p=0)
    case True
    then show ?thesis
      by (metis chain_boundary_def singular_relboundary_0)
  next
    case False
    with that have ∃ d. singular_chain p (subtopology X S) d ∧ chain_boundary
p d = chain_boundary p c
     by (metis add.left_neutral chain_boundary_add chain_boundary_boundary_alt
singular_relboundary)
    with that False show ?thesis
      by (auto simp: singular_boundary)
```

qed
have *ϑ_eq*: *ϑ x* = *ϑ y*
if *x*: *x* ∈ *singular_relcycle_set p X S* **and** *y*: *y* ∈ *singular_relcycle_set p X S*
and *eq*: *singular_relboundary_set p X S* $\#>_{relcycle\_group\ p\ X\ S}$ *x*
= *singular_relboundary_set p X S* $\#>_{relcycle\_group\ p\ X\ S}$ *y* **for** *x y*
proof −
have *singular_relboundary p X S* (*x*−*y*)
by (*metis eq homologous_rel_def homologous_rel_eq mem_Collect_eq right_coset_singular_relboun*
with ∗ **have** (*singular_relboundary* (*p* − *Suc 0*) (*subtopology X S*) {}) (*chain_boundary p* (*x*−*y*))
by *blast*
then show *?thesis*
unfolding *ϑ_def comp_def*
by (*metis chain_boundary_diff homologous_rel_def homologous_rel_eq right_coset_singular_relboun*
qed
obtain *d*
where *d* ∈ *hom* ((*relcycle_group p X S*) *Mod* (*singular_relboundary_set p X S*)) *H*
and *d*: ⋀*u*. *u* ∈ *singular_relcycle_set p X S* ⟹ *d* (*homologous_rel_set p X S u*) = *ϑ u*
by (*metis FactGroup_universal* [*OF ϑ normal_subgroup_singular_relboundary_relcycle ϑ_eq*] *right_coset_singular_relboundary carrier_relcycle_group*)
then have *d* ∈ *hom* (*relative_homology_group p X S*) *H*
by (*simp add*: *nontrivial_relative_homology_group*)
then show ∃ *d*. *?Φ d p X S*
by (*force simp*: *H_def right_coset_singular_relboundary d ϑ_def*)
qed

lemma *hom_boundary2*:
∃ *d*. (∀ *p X S*.
(*d p X S*) ∈ *hom* (*relative_homology_group p X S*)
(*homology_group* (*p* − *1*) (*subtopology X S*)))
∧ (∀ *p X S c*. *singular_relcycle p X S c* ∧ *Suc 0* ≤ *p*
⟶ *d p X S* (*homologous_rel_set p X S c*)
= *homologous_rel_set* (*p* − *Suc 0*) (*subtopology X S*) {} (*chain_boundary p c*))
(**is** ∃ *d*. *?Φ d*)
proof −
have ∗: ∃ *f*. Φ(λ*p*. *if p* ≤ *0 then* λ*q r t*. *undefined else f*(*nat p*)) ⟹ ∃ *f*. Φ *f* **for** Φ
by *blast*
show *?thesis*
apply (*rule* ∗ [*OF ex_forward* [*OF hom_boundary1*]])
apply (*simp add*: *not_le relative_homology_group_def nat_diff_distrib′ int_eq_iff nat_diff_distrib flip*: *nat_1*)
by (*simp add*: *hom_def singleton_group_def*)
qed

lemma *hom_boundary3*:

∃ *d*. ((∀ *p X S c*. *c* ∉ *carrier*(*relative_homology_group p X S*)
⟶ *d p X S c* = *one*(*homology_group* (*p*−*1*) (*subtopology X S*))) ∧
(∀ *p X S*.
d p X S ∈ *hom* (*relative_homology_group p X S*)
(*homology_group* (*p*−*1*) (*subtopology X S*))) ∧
(∀ *p X S c*.
singular_relcycle p X S c ∧ *1* ≤ *p*
⟶ *d p X S* (*homologous_rel_set p X S c*)
= *homologous_rel_set* (*p* − *Suc 0*) (*subtopology X S*) {} (*chain_boundary p c*)) ∧
(∀ *p X S*. *d p X S* = *d p X* (*topspace X* ∩ *S*))) ∧
(∀ *p X S c*. *d p X S c* ∈ *carrier*(*homology_group* (*p*−*1*) (*subtopology X S*)))
∧
(∀ *p*. *p* ≤ *0* ⟶ *d p* = (λ*q r t*. *undefined*))
(**is** ∃ *x*. *?P x* ∧ *?Q x* ∧ *?R x*)
proof −
have ⋀*x*. *?Q x* ⟹ *?R x*
by (*erule all_forward*) (*force simp*: *relative_homology_group_def*)
moreover have ∃ *x*. *?P x* ∧ *?Q x*
proof −
obtain *d*:: [*int*, *'a topology*, *'a set*, (*'a chain*) *set*] ⇒ (*'a chain*) *set*
where *1*: ⋀*p X S*. *d p X S* ∈ *hom* (*relative_homology_group p X S*)
(*homology_group* (*p* − *1*) (*subtopology X S*))
and *2*: ⋀*n X S c*. *singular_relcycle n X S c* ∧ *Suc 0* ≤ *n*
⟹ *d n X S* (*homologous_rel_set n X S c*)
= *homologous_rel_set* (*n* − *Suc 0*) (*subtopology X S*) {} (*chain_boundary n c*)
using *hom_boundary2* **by** *blast*
have *4*: *c* ∈ *carrier* (*relative_homology_group p X S*) ⟹
d p X (*topspace X* ∩ *S*) *c* ∈ *carrier* (*relative_homology_group* (*p*−*1*) (*subtopology X S*) {})
for *p X S c*
using *hom_carrier* [*OF 1* [*of p X topspace X* ∩ *S*]]
by (*simp add*: *image_subset_iff subtopology_restrict*)
show *?thesis*
apply (*rule_tac x*=λ*p X S c*.
if c ∈ *carrier*(*relative_homology_group p X S*)
then d p X (*topspace X* ∩ *S*) *c*
else one(*homology_group* (*p* − *1*) (*subtopology X S*)) **in** *exI*)
apply (*simp add*: *Int_left_absorb subtopology_restrict carrier_relative_homology_group group.is_monoid group.restrict_hom_iff 4 cong*: *if_cong*)
apply (*rule conjI*)
apply (*metis 1 relative_homology_group_restrict subtopology_restrict*)
apply (*metis 2 homologous_rel_restrict singular_relcycle_def subtopology_restrict*)
done
qed
ultimately show *?thesis*
by *auto*
qed

consts *hom_boundary* :: [*int*,′*a topology*,′*a set*,′*a chain set*] ⇒ ′*a chain set*
specification (*hom_boundary*)
hom_boundary:
((∀ *p X S c*. *c* ∉ *carrier*(*relative_homology_group p X S*)
⟶ *hom_boundary p X S c* = *one*(*homology_group* (*p*−*1*) (*subtopology X* (*S*::′*a set*)))) ∧
(∀ *p X S*.
hom_boundary p X S ∈ *hom* (*relative_homology_group p X S*)
(*homology_group* (*p*−*1*) (*subtopology X* (*S*::′*a set*)))) ∧
(∀ *p X S c*.
singular_relcycle p X S c ∧ *1* ≤ *p*
⟶ *hom_boundary p X S* (*homologous_rel_set p X S c*)
= *homologous_rel_set* (*p* − *Suc 0*) (*subtopology X* (*S*::′*a set*)) {} (*chain_boundary p c*)) ∧
(∀ *p X S*. *hom_boundary p X S* = *hom_boundary p X* (*topspace X* ∩ (*S*::′*a set*)))) ∧
(∀ *p X S c*. *hom_boundary p X S c* ∈ *carrier*(*homology_group* (*p*−*1*) (*subtopology X* (*S*::′*a set*)))) ∧
(∀ *p*. *p* ≤ *0* ⟶ *hom_boundary p* = (λ*q r*. λ*t*::′*a chain set*. *undefined*))
by (*fact hom_boundary3*)

lemma *hom_boundary_default*:
c ∉ *carrier*(*relative_homology_group p X S*)
⟹ *hom_boundary p X S c* = *one*(*homology_group* (*p*−*1*) (*subtopology X S*))
and *hom_boundary_hom*: *hom_boundary p X S* ∈ *hom* (*relative_homology_group p X S*) (*homology_group* (*p*−*1*) (*subtopology X S*))
and *hom_boundary_restrict* [*simp*]: *hom_boundary p X* (*topspace X* ∩ *S*) = *hom_boundary p X S*
and *hom_boundary_carrier*: *hom_boundary p X S c* ∈ *carrier*(*homology_group* (*p*−*1*) (*subtopology X S*))
and *hom_boundary_trivial*: *p* ≤ *0* ⟹ *hom_boundary p* = (λ*q r t*. *undefined*)
by (*metis hom_boundary*)+

lemma *hom_boundary_chain_boundary*:
⟦*singular_relcycle p X S c*; *1* ≤ *p*⟧
⟹ *hom_boundary* (*int p*) *X S* (*homologous_rel_set p X S c*) =
homologous_rel_set (*p* − *Suc 0*) (*subtopology X S*) {} (*chain_boundary p c*)
by (*metis hom_boundary*)+

lemma *hom_chain_map*:
⟦*continuous_map X Y f*; *f* ‘ *S* ⊆ *T*⟧
⟹ (*chain_map p f*) ∈ *hom* (*relcycle_group p X S*) (*relcycle_group p Y T*)
by (*force simp*: *chain_map_add singular_relcycle_chain_map hom_def*)

lemma *hom_induced1*:

```
 ∃ hom_relmap.
   (∀ p X S Y T f.
       continuous_map X Y f ∧ f ` (topspace X ∩ S) ⊆ T
       ⟶ (hom_relmap p X S Y T f) ∈ hom (relative_homology_group (int p) X
S)
                                  (relative_homology_group (int p) Y T)) ∧
   (∀ p X S Y T f c.
       continuous_map X Y f ∧ f ` (topspace X ∩ S) ⊆ T ∧
       singular_relcycle p X S c
       ⟶ hom_relmap p X S Y T f (homologous_rel_set p X S c) =
          homologous_rel_set p Y T (chain_map p f c))
proof −
 have ∃ y. (y ∈ hom (relative_homology_group (int p) X S) (relative_homology_group
(int p) Y T)) ∧
           (∀ c. singular_relcycle p X S c ⟶
                 y (homologous_rel_set p X S c) = homologous_rel_set p Y T
(chain_map p f c))
   if contf: continuous_map X Y f and fim: f ` (topspace X ∩ S) ⊆ T
   for p X S Y T and f :: 'a ⇒ 'b
  proof −
  let ?f = (#>relcycle_group p Y T) (singular_relboundary_set p Y T) ∘ chain_map
p f
   let ?F = λx. singular_relboundary_set p X S #>relcycle_group p X S x
   have 1: ?f ∈ hom (relcycle_group p X S) (relative_homology_group (int p) Y
T)
     apply (rule hom_compose [where H = relcycle_group p Y T])
      apply (metis contf fim hom_chain_map relcycle_group_restrict)
    by (simp add: nontrivial_relative_homology_group normal.r_coset_hom_Mod
normal_subgroup_singular_relboundary_relcycle)
   have 2: singular_relboundary_set p X S ◁ relcycle_group p X S
     using normal_subgroup_singular_relboundary_relcycle by blast
   have 3: ?f x = ?f y
     if singular_relcycle p X S x singular_relcycle p X S y ?F x = ?F y for x y
   proof −
     have singular_relboundary p Y T (chain_map p f (x − y))
       apply (rule singular_relboundary_chain_map [OF _ contf fim])
      by (metis homologous_rel_def homologous_rel_eq mem_Collect_eq right_coset_singular_relboundary
singular_relboundary_restrict that(3))
     then have singular_relboundary p Y T (chain_map p f x − chain_map p f
y)
       by (simp add: chain_map_diff)
     with that
     show ?thesis
     apply (simp add: right_coset_singular_relboundary homologous_rel_set_eq)
       apply (simp add: homologous_rel_def)
       done
   qed
   obtain g where g ∈ hom (relcycle_group p X S Mod singular_relboundary_set
p X S)
```

(*relative_homology_group* (*int p*) *Y T*)
$\bigwedge$*x*. *x* ∈ *singular_relcycle_set p X S* ⟹ *g* (*?F x*) = *?f x*
using *FactGroup_universal* [*OF 1 2 3*, *unfolded carrier_relcycle_group*] **by** *blast*
then show *?thesis*
by (*force simp*: *right_coset_singular_relboundary nontrivial_relative_homology_group*)
qed
then show *?thesis*
apply (*simp flip*: *all_conj_distrib*)
apply ((*subst choice_iff* [*symmetric*])+)
apply *metis*
done
qed

lemma *hom_induced2*:
∃ *hom_relmap*.
(∀ *p X S Y T f*.
continuous_map X Y f ∧
f ‘ (*topspace X* ∩ *S*) ⊆ *T*
⟶ (*hom_relmap p X S Y T f*) ∈ *hom* (*relative_homology_group p X S*)
(*relative_homology_group p Y T*)) ∧
(∀ *p X S Y T f c*.
continuous_map X Y f ∧
f ‘ (*topspace X* ∩ *S*) ⊆ *T* ∧
singular_relcycle p X S c
⟶ *hom_relmap p X S Y T f* (*homologous_rel_set p X S c*) =
homologous_rel_set p Y T (*chain_map p f c*)) ∧
(∀ *p*. *p* < *0* ⟶ *hom_relmap p* = (λ*X S Y T f c*. *undefined*))
(**is** ∃ *d*. *?Φ d*)
proof −
have ∗: ∃ *f*. Φ(λ*p*. *if p* < *0 then* λ*X S Y T f c*. *undefined else f*(*nat p*)) ⟹ ∃ *f*. Φ *f* **for** Φ
by *blast*
show *?thesis*
apply (*rule* ∗ [*OF ex_forward* [*OF hom_induced1*]])
apply (*simp add*: *not_le relative_homology_group_def nat_diff_distrib′ int_eq_iff nat_diff_distrib flip*: *nat_1*)
done
qed

lemma *hom_induced3*:
∃ *hom_relmap*.
((∀ *p X S Y T f c*.
~(*continuous_map X Y f* ∧ *f* ‘ (*topspace X* ∩ *S*) ⊆ *T* ∧
c ∈ *carrier*(*relative_homology_group p X S*))
⟶ *hom_relmap p X S Y T f c* = *one*(*relative_homology_group p Y T*)) ∧
(∀ *p X S Y T f*.
hom_relmap p X S Y T f ∈ *hom* (*relative_homology_group p X S*) (*relative_homology_group p Y T*)) ∧

(∀ *p X S Y T f c*.
continuous_map X Y f ∧ *f* ‘ (*topspace X* ∩ *S*) ⊆ *T* ∧ *singular_relcycle p X S c*
⟶ *hom_relmap p X S Y T f* (*homologous_rel_set p X S c*) =
homologous_rel_set p Y T (*chain_map p f c*)) ∧
(∀ *p X S Y T*.
hom_relmap p X S Y T =
hom_relmap p X (*topspace X* ∩ *S*) *Y* (*topspace Y* ∩ *T*))) ∧
(∀ *p X S Y f T c*.
hom_relmap p X S Y T f c ∈ *carrier*(*relative_homology_group p Y T*)) ∧
(∀ *p*. *p* < *0* ⟶ *hom_relmap p* = (λ*X S Y T f c*. *undefined*))
(**is** ∃ *x*. *?P x* ∧ *?Q x* ∧ *?R x*)
proof −
have ⋀*x*. *?Q x* ⟹ *?R x*
by (*erule all_forward*) (*fastforce simp*: *relative_homology_group_def*)
moreover have ∃ *x*. *?P x* ∧ *?Q x*
proof −
obtain *hom_relmap*:: [*int*,'*a topology*,'*a set*,'*b topology*,'*b set*,'*a* ⇒ '*b*,('*a chain*) *set*] ⇒ ('*b chain*) *set*
where *1*: ⋀*p X S Y T f*. ⟦*continuous_map X Y f*; *f* ‘ (*topspace X* ∩ *S*) ⊆ *T*⟧ ⟹
hom_relmap p X S Y T f
∈ *hom* (*relative_homology_group p X S*) (*relative_homology_group p Y T*)
and *2*: ⋀*p X S Y T f c*.
⟦*continuous_map X Y f*; *f* ‘ (*topspace X* ∩ *S*) ⊆ *T*; *singular_relcycle p X S c*⟧
⟹
hom_relmap (*int p*) *X S Y T f* (*homologous_rel_set p X S c*) =
homologous_rel_set p Y T (*chain_map p f c*)
and *3*: (∀ *p*. *p* < *0* ⟶ *hom_relmap p* = (λ*X S Y T f c*. *undefined*))
using *hom_induced2* [**where** *?'a*='*a* **and** *?'b*='*b*]
apply *clarify*
apply (*rule_tac hom_relmap*=*hom_relmap* **in** *that*, *auto*)
done
have *4*: ⟦*continuous_map X Y f*; *f* ‘ (*topspace X* ∩ *S*) ⊆ *T*; *c* ∈ *carrier* (*relative_homology_group p X S*)⟧ ⟹
hom_relmap p X (*topspace X* ∩ *S*) *Y* (*topspace Y* ∩ *T*) *f c*
∈ *carrier* (*relative_homology_group p Y T*)
for *p X S Y f T c*
using *hom_carrier* [*OF 1* [*of X Y f topspace X* ∩ *S topspace Y* ∩ *T p*]]
continuous_map_image_subset_topspace **by** *fastforce*
have *inhom*: (λ*c*. *if continuous_map X Y f* ∧ *f* ‘ (*topspace X* ∩ *S*) ⊆ *T* ∧
c ∈ *carrier* (*relative_homology_group p X S*)
then hom_relmap p X (*topspace X* ∩ *S*) *Y* (*topspace Y* ∩ *T*) *f c*
else $\mathbf{1}_{relative\_homology\_group\ p\ Y\ T}$)
∈ *hom* (*relative_homology_group p X S*) (*relative_homology_group p Y T*)
(**is** *?h* ∈ *hom ?GX ?GY*)
for *p X S Y T f*

```
    proof (rule homI)
      show ⋀x. x ∈ carrier ?GX ⟹ ?h x ∈ carrier ?GY
        by (auto simp: 4 group.is_monoid)
      show ?h (x ⊗?GX y) = ?h x ⊗?GY?h y if x ∈ carrier ?GX y ∈ carrier ?GX
for x y
      proof (cases p < 0)
        case True
        with that show ?thesis
          by (simp add: relative_homology_group_def singleton_group_def 3)
      next
        case False
        show ?thesis
        proof (cases continuous_map X Y f)
          case True
          then have f ‘ (topspace X ∩ S) ⊆ topspace Y
            using continuous_map_image_subset_topspace by blast
          then show ?thesis
            using True False that
            using 1 [of X Y f topspace X ∩ S topspace Y ∩ T p]
             by (simp add: 4 continuous_map_image_subset_topspace hom_mult
not_less group.is_monoid monoid.m_closed Int_left_absorb)
        qed (simp add: group.is_monoid)
      qed
    qed
   have hrel: ⟦continuous_map X Y f; f ‘ (topspace X ∩ S) ⊆ T; singular_relcycle
p X S c⟧
           ⟹ hom_relmap (int p) X (topspace X ∩ S) Y (topspace Y ∩ T)
          f (homologous_rel_set p X S c) = homologous_rel_set p Y T (chain_map
p f c)
        for p X S Y T f c
      using 2 [of X Y f topspace X ∩ S topspace Y ∩ T p c]
            continuous_map_image_subset_topspace by fastforce
    show ?thesis
      apply (rule_tac x=λp X S Y T f c.
                if continuous_map X Y f ∧ f ‘ (topspace X ∩ S) ⊆ T ∧
                   c ∈ carrier(relative_homology_group p X S)
                then hom_relmap p X (topspace X ∩ S) Y (topspace Y ∩ T) f c
                else one(relative_homology_group p Y T) in exI)
    apply (simp add: Int_left_absorb subtopology_restrict carrier_relative_homology_group
          group.is_monoid group.restrict_hom_iff 4 inhom hrel cong: if_cong)
      apply (force simp: continuous_map_def intro!: ext)
      done
  qed
  ultimately show ?thesis
    by auto
qed


consts hom_induced:: [int,'a topology,'a set,'b topology,'b set,'a ⇒ 'b,('a chain)
```

set] ⇒ (′*b chain*) *set*
specification (*hom_induced*)
hom_induced:
((∀ *p X S Y T f c*.
~(*continuous_map X Y f* ∧
f ‘ (*topspace X* ∩ *S*) ⊆ *T* ∧
c ∈ *carrier*(*relative_homology_group p X S*))
⟶ *hom_induced p X* (*S*::′*a set*) *Y* (*T*::′*b set*) *f c* =
one(*relative_homology_group p Y T*)) ∧
(∀ *p X S Y T f*.
(*hom_induced p X* (*S*::′*a set*) *Y* (*T*::′*b set*) *f*) ∈ *hom* (*relative_homology_group p X S*)
(*relative_homology_group p Y T*)) ∧
(∀ *p X S Y T f c*.
continuous_map X Y f ∧
f ‘ (*topspace X* ∩ *S*) ⊆ *T* ∧
singular_relcycle p X S c
⟶ *hom_induced p X* (*S*::′*a set*) *Y* (*T*::′*b set*) *f* (*homologous_rel_set p X S c*) =
homologous_rel_set p Y T (*chain_map p f c*)) ∧
(∀ *p X S Y T*.
hom_induced p X (*S*::′*a set*) *Y* (*T*::′*b set*) =
hom_induced p X (*topspace X* ∩ *S*) *Y* (*topspace Y* ∩ *T*))) ∧
(∀ *p X S Y f T c*.
hom_induced p X (*S*::′*a set*) *Y* (*T*::′*b set*) *f c* ∈
carrier(*relative_homology_group p Y T*)) ∧
(∀ *p*. *p* < *0* ⟶ *hom_induced p* = (λ*X S Y T*. λ*f*::′*a*⇒′*b*. λ*c*. *undefined*))
by (*fact hom_induced3*)

lemma *hom_induced_default*:
~(*continuous_map X Y f* ∧ *f* ‘ (*topspace X* ∩ *S*) ⊆ *T* ∧ *c* ∈ *carrier*(*relative_homology_group p X S*))
⟹ *hom_induced p X S Y T f c* = *one*(*relative_homology_group p Y T*)
and *hom_induced_hom*:
hom_induced p X S Y T f ∈ *hom* (*relative_homology_group p X S*) (*relative_homology_group p Y T*)
and *hom_induced_restrict* [*simp*]:
hom_induced p X (*topspace X* ∩ *S*) *Y* (*topspace Y* ∩ *T*) = *hom_induced p X S Y T*
and *hom_induced_carrier*:
hom_induced p X S Y T f c ∈ *carrier*(*relative_homology_group p Y T*)
and *hom_induced_trivial*: *p* < *0* ⟹ *hom_induced p* = (λ*X S Y T f c*. *undefined*)
by (*metis hom_induced*)+

lemma *hom_induced_chain_map_gen*:
⟦*continuous_map X Y f*; *f* ‘ (*topspace X* ∩ *S*) ⊆ *T*; *singular_relcycle p X S c*⟧
⟹ *hom_induced p X S Y T f* (*homologous_rel_set p X S c*) = *homologous_rel_set p Y T* (*chain_map p f c*)
by (*metis hom_induced*)

lemma *hom_induced_chain_map*:
⟦*continuous_map X Y f*; *f ‘ S* ⊆ *T*; *singular_relcycle p X S c*⟧
⟹ *hom_induced p X S Y T f* (*homologous_rel_set p X S c*)
= *homologous_rel_set p Y T* (*chain_map p f c*)
by (*meson Int_lower2 hom_induced image_subsetI image_subset_iff subset_iff*)

lemma *hom_induced_eq*:
assumes ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*
shows *hom_induced p X S Y T f* = *hom_induced p X S Y T g*
proof −
consider *p* < *0* | *n* **where** *p* = *int n*
by (*metis int_nat_eq not_less*)
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
by (*simp add*: *hom_induced_trivial*)
next
case *2*
have *hom_induced n X S Y T f C* = *hom_induced n X S Y T g C* **for** *C*
proof −
have *continuous_map X Y f* ∧ *f ‘* (*topspace X* ∩ *S*) ⊆ *T* ∧ *C* ∈ *carrier* (*relative_homology_group n X S*)
⟷ *continuous_map X Y g* ∧ *g ‘* (*topspace X* ∩ *S*) ⊆ *T* ∧ *C* ∈ *carrier* (*relative_homology_group n X S*)
(**is** *?P* = *?Q*)
by (*metis IntD1 assms continuous_map_eq image_cong*)
then consider ¬ *?P* ∧ ¬ *?Q* | *?P* ∧ *?Q*
by *blast*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
by (*simp add*: *hom_induced_default*)
next
case *2*
have *homologous_rel_set n Y T* (*chain_map n f c*) = *homologous_rel_set n Y T* (*chain_map n g c*)
if *continuous_map X Y f f ‘* (*topspace X* ∩ *S*) ⊆ *T*
continuous_map X Y g g ‘ (*topspace X* ∩ *S*) ⊆ *T*
C = *homologous_rel_set n X S c singular_relcycle n X S c*
for *c*
proof −
have *chain_map n f c* = *chain_map n g c*
using *assms chain_map_eq singular_relcycle that* **by** *blast*
then show *?thesis*
by *simp*

```
        qed
        with 2 show ?thesis
          by (auto simp: relative_homology_group_def carrier_FactGroup
              right_coset_singular_relboundary hom_induced_chain_map_gen)
      qed
    qed
    with 2 show ?thesis
      by auto
  qed
qed
```

0.2.2 Towards the Eilenberg-Steenrod axioms

First prove we get functors into abelian groups with the boundary map being a natural transformation between them, and prove Eilenberg-Steenrod axioms (we also prove additivity a bit later on if one counts that).

```
lemma abelian_relative_homology_group [simp]:
   comm_group(relative_homology_group p X S)
  apply (simp add: relative_homology_group_def)
 apply (metis comm_group.abelian_FactGroup abelian_relcycle_group subgroup_singular_relboundary_relcycle)
  done

lemma abelian_homology_group: comm_group(homology_group p X)
  by simp


lemma hom_induced_id_gen:
  assumes contf: continuous_map X X f and feq: ⋀x. x ∈ topspace X ⟹ f x =
x
    and c: c ∈ carrier (relative_homology_group p X S)
  shows hom_induced p X S X S f c = c
proof –
  consider p < 0 | n where p = int n
    by (metis int_nat_eq not_less)
  then show ?thesis
  proof cases
    case 1
    with c show ?thesis
      by (simp add: hom_induced_trivial relative_homology_group_def)
  next
    case 2
    have cm: chain_map n f d = d if singular_relcycle n X S d for d
      using that assms by (auto simp: chain_map_id_gen singular_relcycle)
    have f ‘ (topspace X ∩ S) ⊆ S
      using feq by auto
    with 2 c show ?thesis
      by (auto simp: nontrivial_relative_homology_group carrier_FactGroup
            cm right_coset_singular_relboundary hom_induced_chain_map_gen
assms)
```

qed
qed

lemma *hom_induced_id*:
c ∈ *carrier* (*relative_homology_group p X S*) ⟹ *hom_induced p X S X S id c* = *c*
by (*rule hom_induced_id_gen*) *auto*

lemma *hom_induced_compose*:
assumes *continuous_map X Y f f* ‘ *S* ⊆ *T continuous_map Y Z g g* ‘ *T* ⊆ *U*
shows *hom_induced p X S Z U* (*g* ∘ *f*) = *hom_induced p Y T Z U g* ∘ *hom_induced p X S Y T f*
proof −
consider (*neg*) *p* < *0* | (*int*) *n* **where** *p* = *int n*
by (*metis int_nat_eq not_less*)
then show *?thesis*
proof *cases*
case *int*
have *gf*: *continuous_map X Z* (*g* ∘ *f*)
using *assms continuous_map_compose* **by** *fastforce*
have *gfim*: (*g* ∘ *f*) ‘ *S* ⊆ *U*
unfolding *o_def* **using** *assms* **by** *blast*
have *sr*: ⋀*a*. *singular_relcycle n X S a* ⟹ *singular_relcycle n Y T* (*chain_map n f a*)
by (*simp add*: *assms singular_relcycle_chain_map*)
show *?thesis*
proof
fix *c*
show *hom_induced p X S Z U* (*g* ∘ *f*) *c* = (*hom_induced p Y T Z U g* ∘ *hom_induced p X S Y T f*) *c*
proof (*cases c* ∈ *carrier*(*relative_homology_group p X S*))
case *True*
with *gfim* **show** *?thesis*
unfolding *int*
by (*auto simp*: *carrier_relative_homology_group gf gfim assms sr chain_map_compose hom_induced_chain_map*)
next
case *False*
then show *?thesis*
by (*simp add*: *hom_induced_default hom_one* [*OF hom_induced_hom*])
qed
qed
qed (*force simp*: *hom_induced_trivial*)
qed

lemma *hom_induced_compose′*:
assumes *continuous_map X Y f f* ‘ *S* ⊆ *T continuous_map Y Z g g* ‘ *T* ⊆ *U*
shows *hom_induced p Y T Z U g* (*hom_induced p X S Y T f x*) = *hom_induced*

```
p X S Z U (g ∘ f) x
  using hom_induced_compose [OF assms] by simp

lemma naturality_hom_induced:
  assumes continuous_map X Y f f ` S ⊆ T
  shows hom_boundary q Y T ∘ hom_induced q X S Y T f
       = hom_induced (q − 1) (subtopology X S) {} (subtopology Y T) {} f ∘
hom_boundary q X S
proof (cases q ≤ 0)
  case False
  then obtain p where p1: p ≥ Suc 0 and q: q = int p
    using zero_le_imp_eq_int by force
  show ?thesis
  proof
    fix c
    show (hom_boundary q Y T ∘ hom_induced q X S Y T f) c =
          (hom_induced (q − 1) (subtopology X S) {} (subtopology Y T) {} f ∘
hom_boundary q X S) c
    proof (cases c ∈ carrier(relative_homology_group p X S))
      case True
       then obtain a where ceq: c = homologous_rel_set p X S a and a: singu-
lar_relcycle p X S a
        by (force simp: carrier_relative_homology_group)
      then have sr: singular_relcycle p Y T (chain_map p f a)
        using assms singular_relcycle_chain_map by fastforce
    then have sb: singular_relcycle (p − Suc 0) (subtopology X S) {} (chain_boundary
p a)
          by (metis One_nat_def a chain_boundary_boundary singular_chain_0
singular_relcycle)
      have p1_eq: int p − 1 = int (p − Suc 0)
        using p1 by auto
      have cbm: (chain_boundary p (chain_map p f a))
              = (chain_map (p − Suc 0) f (chain_boundary p a))
        using a chain_boundary_chain_map singular_relcycle by blast
      have contf: continuous_map (subtopology X S) (subtopology Y T) f
        using assms
        by (auto simp: continuous_map_in_subtopology topspace_subtopology
          continuous_map_from_subtopology)
      show ?thesis
        unfolding q using assms p1 a
      apply (simp add: ceq assms hom_induced_chain_map hom_boundary_chain_boundary
                      hom_boundary_chain_boundary [OF sr] singular_relcycle_def
mod_subset_def)
        apply (simp add: p1_eq contf sb cbm hom_induced_chain_map)
        done
    next
      case False
      with assms show ?thesis
        unfolding q o_def using assms
```

apply (*simp add*: *hom_induced_default hom_boundary_default*)
by (*metis group_relative_homology_group hom_boundary hom_induced hom_one one_relative_homology_group*)
qed
qed
qed (*force simp*: *hom_induced_trivial hom_boundary_trivial*)

lemma *homology_exactness_axiom_1*:
exact_seq ([*homology_group* (*p*−*1*) (*subtopology X S*), *relative_homology_group p X S*, *homology_group p X*],
[*hom_boundary p X S*,*hom_induced p X* {} *X S id*])
proof −
consider (*neg*) *p* < *0* | (*int*) *n* **where** *p* = *int n*
by (*metis int_nat_eq not_less*)
then have (*hom_induced p X* {} *X S id*) ‘ *carrier* (*homology_group p X*)
= *kernel* (*relative_homology_group p X S*) (*homology_group* (*p*−*1*) (*subtopology X S*))
(*hom_boundary p X S*)
proof *cases*
case *neg*
then show *?thesis*
unfolding *kernel_def singleton_group_def relative_homology_group_def*
by (*auto simp*: *hom_induced_trivial hom_boundary_trivial*)
next
case *int*
have *hom_induced* (*int m*) *X* {} *X S id* ‘ *carrier* (*relative_homology_group* (*int m*) *X* {})
= *carrier* (*relative_homology_group* (*int m*) *X S*) ∩
{*c*. *hom_boundary* (*int m*) *X S c* = $\mathbf{1}_{relative\_homology\_group\ (int\ m\ -\ 1)\ (subtopology\ X\ S)\ \{\}}$}
for *m*
proof (*cases m*)
case *0*
have *hom_induced 0 X* {} *X S id* ‘ *carrier* (*relative_homology_group 0 X* {})
= *carrier* (*relative_homology_group 0 X S*) (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
using *hom_induced_hom* [*of 0 X* {} *X S id*]
by (*simp add*: *hom_induced_hom hom_carrier*)
show *?rhs* ⊆ *?lhs*
apply (*clarsimp simp add*: *image_iff carrier_relative_homology_group* [*of 0*, *simplified*] *singular_relcycle*)
apply (*force simp*: *chain_map_id_gen chain_boundary_def singular_relcycle*
hom_induced_chain_map [*of concl*: *0*, *simplified*])
done
qed

```
    with 0 show ?thesis
     by (simp add: hom_boundary_trivial relative_homology_group_def [of -1]
singleton_group_def)
  next
    case (Suc n)
    have (hom_induced (int (Suc n)) X {} X S id ∘
          homologous_rel_set (Suc n) X {}) ` singular_relcycle_set (Suc n) X {}
        = homologous_rel_set (Suc n) X S `
          (singular_relcycle_set (Suc n) X S ∩
           {c. hom_boundary (int (Suc n)) X S (homologous_rel_set (Suc n) X S
c)
              = singular_relboundary_set n (subtopology X S) {}})
      (is ?lhs = ?rhs)
    proof -
      have 1: (⋀x. x ∈ A ⟹ x ∈ B ⟷ x ∈ C) ⟹ f ` (A ∩ B) = f ` (A ∩ C)
for f A B C
        by blast
      have 2: ⟦⋀x. x ∈ A ⟹ ∃ y. y ∈ B ∧ f x = f y; ⋀x. x ∈ B ⟹ ∃ y. y ∈ A
∧ f x = f y⟧
  ⟹ f ` A = f ` B for f A B
        by blast
      have ?lhs = homologous_rel_set (Suc n) X S ` singular_relcycle_set (Suc
n) X {}
        apply (rule image_cong [OF refl])
        apply (simp add: o_def hom_induced_chain_map chain_map_ident [of
_ X] singular_relcycle
              del: of_nat_Suc)
        done
      also have ... = homologous_rel_set (Suc n) X S `
                    (singular_relcycle_set (Suc n) X S ∩
                  {c. singular_relboundary n (subtopology X S) {} (chain_boundary
(Suc n) c)})
      proof (rule 2)
        fix c
        assume c ∈ singular_relcycle_set (Suc n) X {}
        then show ∃ y. y ∈ singular_relcycle_set (Suc n) X S ∩
                {c. singular_relboundary n (subtopology X S) {} (chain_boundary
(Suc n) c)} ∧
          homologous_rel_set (Suc n) X S c = homologous_rel_set (Suc n) X S y
          apply (rule_tac x=c in exI)
            by (simp add: singular_boundary) (metis chain_boundary_0 singu-
lar_cycle singular_relcycle singular_relcycle_0)
      next
        fix c
        assume c: c ∈ singular_relcycle_set (Suc n) X S ∩
                {c. singular_relboundary n (subtopology X S) {} (chain_boundary
(Suc n) c)}
        then obtain d where d: singular_chain (Suc n) (subtopology X S) d
          chain_boundary (Suc n) d = chain_boundary (Suc n) c
```

```
          by (auto simp: singular_boundary)
         with c have c − d ∈ singular_relcycle_set (Suc n) X {}
        by (auto simp: singular_cycle chain_boundary_diff singular_chain_subtopology
singular_relcycle singular_chain_diff)
        moreover have homologous_rel_set (Suc n) X S c = homologous_rel_set
(Suc n) X S (c − d)
         proof (simp add: homologous_rel_set_eq)
          show homologous_rel (Suc n) X S c (c − d)
         using d by (simp add: homologous_rel_def singular_chain_imp_relboundary)
         qed
         ultimately show ∃ y. y ∈ singular_relcycle_set (Suc n) X {} ∧
                 homologous_rel_set (Suc n) X S c = homologous_rel_set (Suc n)
X S y
          by blast
       qed
       also have . . . = ?rhs
       by (rule 1) (simp add: hom_boundary_chain_boundary homologous_rel_set_eq_relboundary
del: of_nat_Suc)
       finally show ?lhs = ?rhs .
     qed
     with Suc show ?thesis
       unfolding carrier_relative_homology_group image_comp id_def by auto
   qed
   then show ?thesis
     by (auto simp: kernel_def int)
 qed
 then show ?thesis
   using hom_boundary_hom hom_induced_hom
   by (force simp: group_hom_def group_hom_axioms_def)
qed


lemma homology_exactness_axiom_2:
  exact_seq ([homology_group (p−1) X, homology_group (p−1) (subtopology X
S), relative_homology_group p X S],
           [hom_induced (p−1) (subtopology X S) {} X {} id, hom_boundary p
X S])
proof −
 consider (neg) p ≤ 0 | (int) n where p = int (Suc n)
   by (metis linear not0_implies_Suc of_nat_0 zero_le_imp_eq_int)
 then have kernel (relative_homology_group (p − 1) (subtopology X S) {})
                (relative_homology_group (p − 1) X {})
                (hom_induced (p − 1) (subtopology X S) {} X {} id)
          = hom_boundary p X S ‘ carrier (relative_homology_group p X S)
 proof cases
   case neg
   obtain x where x ∈ carrier (relative_homology_group p X S)
     using group_relative_homology_group group.is_monoid by blast
   with neg show ?thesis
```

```
      unfolding kernel_def singleton_group_def relative_homology_group_def
      by (force simp: hom_induced_trivial hom_boundary_trivial)
  next
    case int
    have hom_boundary (int (Suc n)) X S ‘ carrier (relative_homology_group (int
(Suc n)) X S)
        = carrier (relative_homology_group n (subtopology X S) {}) ∩
          {c. hom_induced n (subtopology X S) {} X {} id c =
           1_relative_homology_group n X {}}
        (is ?lhs = ?rhs)
    proof −
      have 1: (⋀x. x ∈ A ⟹ x ∈ B ⟷ x ∈ C) ⟹ f ‘ (A ∩ B) = f ‘ (A ∩ C)
for f A B C
        by blast
      have 2: (⋀x. x ∈ A ⟹ x ∈ B ⟷ x ∈ f −‘ C) ⟹ f ‘ (A ∩ B) = f ‘ A ∩
C for f A B C
        by blast
      have ?lhs = homologous_rel_set n (subtopology X S) {}
                 ‘ (chain_boundary (Suc n) ‘ singular_relcycle_set (Suc n) X S)
        unfolding carrier_relative_homology_group image_comp
      by (rule image_cong [OF refl]) (simp add: o_def hom_boundary_chain_boundary
del: of_nat_Suc)
      also have ... = homologous_rel_set n (subtopology X S) {} ‘
                        (singular_relcycle_set n (subtopology X S) {} ∩ singu-
lar_relboundary_set n X {})
      by (force simp: singular_relcycle singular_boundary chain_boundary_boundary_alt)
      also have ... = ?rhs
        unfolding carrier_relative_homology_group vimage_def
        apply (rule 2)
          apply (auto simp: hom_induced_chain_map chain_map_ident homolo-
gous_rel_set_eq_relboundary singular_relcycle)
        done
      finally show ?thesis .
    qed
    then show ?thesis
      by (auto simp: kernel_def int)
  qed
  then show ?thesis
    using hom_boundary_hom hom_induced_hom
    by (force simp: group_hom_def group_hom_axioms_def)
qed


lemma homology_exactness_axiom_3:
    exact_seq ([relative_homology_group p X S, homology_group p X, homol-
ogy_group p (subtopology X S)],
              [hom_induced p X {} X S id, hom_induced p (subtopology X S) {} X
{} id])
proof (cases p < 0)
```

```
  case True
  then show ?thesis
   apply (simp add: relative_homology_group_def hom_induced_trivial group_hom_def
group_hom_axioms_def)
    apply (auto simp: kernel_def singleton_group_def)
    done
next
  case False
  then obtain n where peq: p = int n
    by (metis int_ops(1) linorder_neqE_linordered_idom pos_int_cases)
  have hom_induced n (subtopology X S) {} X {} id `
        (homologous_rel_set n (subtopology X S) {} `
        singular_relcycle_set n (subtopology X S) {})
      = {c ∈ homologous_rel_set n X {} ` singular_relcycle_set n X {}.
         hom_induced n X {} X S id c = singular_relboundary_set n X S}
        (is ?lhs = ?rhs)
  proof -
    have 2: ⟦⋀x. x ∈ A ⟹ ∃ y. y ∈ B ∧ f x = f y; ⋀x. x ∈ B ⟹ ∃ y. y ∈ A ∧
f x = f y⟧
    ⟹ f ` A = f ` B for f A B
      by blast
   have ?lhs = homologous_rel_set n X {} ` (singular_relcycle_set n (subtopology
X S) {})
      unfolding image_comp o_def
      apply (rule image_cong [OF refl])
      apply (simp add: hom_induced_chain_map singular_relcycle)
       apply (metis chain_map_ident)
      done
    also have ... = homologous_rel_set n X {} ` (singular_relcycle_set n X {}
∩ singular_relboundary_set n X S)
    proof (rule 2)
      fix c
      assume c ∈ singular_relcycle_set n (subtopology X S) {}
     then show ∃ y. y ∈ singular_relcycle_set n X {} ∩ singular_relboundary_set
n X S ∧
             homologous_rel_set n X {} c = homologous_rel_set n X {} y
      using singular_chain_imp_relboundary singular_cycle singular_relboundary_imp_chain
singular_relcycle by fastforce
    next
      fix c
      assume c ∈ singular_relcycle_set n X {} ∩ singular_relboundary_set n X S
      then obtain d e where c: singular_relcycle n X {} c singular_relboundary
n X S c
        and d:  singular_chain n (subtopology X S) d
        and e: singular_chain (Suc n) X e chain_boundary (Suc n) e = c + d
        using singular_relboundary_alt by blast
      then have chain_boundary n (c + d) = 0
        using chain_boundary_boundary_alt by fastforce
      then have chain_boundary n c + chain_boundary n d = 0
```

```
        by (metis chain_boundary_add)
      with c have singular_relcycle n (subtopology X S) {} (− d)
       by (metis (no_types) d eq_add_iff singular_cycle singular_relcycle_minus)
      moreover have homologous_rel n X {} c (− d)
        using c
        by (metis diff_minus_eq_add e homologous_rel_def singular_boundary)
      ultimately
      show ∃ y. y ∈ singular_relcycle_set n (subtopology X S) {} ∧
           homologous_rel_set n X {} c = homologous_rel_set n X {} y
        by (force simp: homologous_rel_set_eq)
    qed
    also have ... = homologous_rel_set n X {} `
                  (singular_relcycle_set n X {} ∩ homologous_rel_set n X {} −` {x.
hom_induced n X {} X S id x = singular_relboundary_set n X S})
     by (rule 2) (auto simp: hom_induced_chain_map homologous_rel_set_eq_relboundary
chain_map_ident [of _ X] singular_cycle cong: conj_cong)
    also have ... = ?rhs
      by blast
    finally show ?thesis .
  qed
  then have kernel (relative_homology_group p X {}) (relative_homology_group
p X S) (hom_induced p X {} X S id)
     = hom_induced p (subtopology X S) {} X {} id ` carrier (relative_homology_group
p (subtopology X S) {})
    by (simp add: kernel_def carrier_relative_homology_group peq)
  then show ?thesis
   by (simp add: not_less group_hom_def group_hom_axioms_def hom_induced_hom)
qed


lemma homology_dimension_axiom:
  assumes X: topspace X = {a} and p ≠ 0
  shows trivial_group(homology_group p X)
proof (cases p < 0)
  case True
  then show ?thesis
    by simp
next
  case False
  then obtain n where peq: p = int n n > 0
    by (metis assms(2) neq0_conv nonneg_int_cases not_less of_nat_0)
 have homologous_rel_set n X {} ` singular_relcycle_set n X {} = {singular_relcycle_set
n X {}}
        (is ?lhs = ?rhs)
  proof
    show ?lhs ⊆ ?rhs
      using peq assms
       by (auto simp: image_subset_iff homologous_rel_set_eq_relboundary simp
flip: singular_boundary_set_eq_cycle_singleton)
```

```
    have singular_relboundary n X {} 0
      by simp
    with peq assms
    show ?rhs ⊆ ?lhs
       by (auto simp: image_iff simp flip: homologous_rel_eq_relboundary singu-
lar_boundary_set_eq_cycle_singleton)
  qed
  with peq assms show ?thesis
    unfolding trivial_group_def
   by (simp add: carrier_relative_homology_group singular_boundary_set_eq_cycle_singleton
[OF X])
qed


proposition homology_homotopy_axiom:
  assumes homotopic_with (λh. h ` S ⊆ T) X Y f g
  shows hom_induced p X S Y T f = hom_induced p X S Y T g
proof (cases p < 0)
  case True
  then show ?thesis
    by (simp add: hom_induced_trivial)
next
  case False
  then obtain n where peq: p = int n
    by (metis int_nat_eq not_le)
  have cont: continuous_map X Y f continuous_map X Y g
    using assms homotopic_with_imp_continuous_maps by blast+
  have im: f ` (topspace X ∩ S) ⊆ T g ` (topspace X ∩ S) ⊆ T
    using homotopic_with_imp_property assms by blast+
  show ?thesis
  proof
    fix c show hom_induced p X S Y T f c = hom_induced p X S Y T g c
    proof (cases c ∈ carrier(relative_homology_group p X S))
      case True
      then obtain a where a: c = homologous_rel_set n X S a singular_relcycle
n X S a
        unfolding carrier_relative_homology_group peq by auto
      then show ?thesis
          apply (simp add: peq hom_induced_chain_map_gen cont im homolo-
gous_rel_set_eq)
        apply (blast intro: assms homotopic_imp_homologous_rel_chain_maps)
        done
    qed (simp add: hom_induced_default)
  qed
qed

proposition homology_excision_axiom:
  assumes X closure_of U ⊆ X interior_of T T ⊆ S
  shows
```

```
  hom_induced p (subtopology X (S − U)) (T − U) (subtopology X S) T id
  ∈ iso (relative_homology_group p (subtopology X (S − U)) (T − U))
        (relative_homology_group p (subtopology X S) T)
proof (cases p < 0)
  case True
  then show ?thesis
    unfolding iso_def bij_betw_def relative_homology_group_def by (simp add:
hom_induced_trivial)
next
  case False
  then obtain n where peq: p = int n
    by (metis int_nat_eq not_le)
  have cont: continuous_map (subtopology X (S − U)) (subtopology X S) id
   by (simp add: closure_of_subtopology_mono continuous_map_eq_image_closure_subset)
  have TU: topspace X ∩ (S − U) ∩ (T − U) ⊆ T
    by auto
  show ?thesis
   proof (simp add: iso_def peq carrier_relative_homology_group bij_betw_def
hom_induced_hom, intro conjI)
    show inj_on (hom_induced n (subtopology X (S − U)) (T − U) (subtopology
X S) T id)
         (homologous_rel_set n (subtopology X (S − U)) (T − U) `
          singular_relcycle_set n (subtopology X (S − U)) (T − U))
      unfolding inj_on_def
    proof (clarsimp simp add: homologous_rel_set_eq)
      fix c d
      assume c: singular_relcycle n (subtopology X (S − U)) (T − U) c
        and d: singular_relcycle n (subtopology X (S − U)) (T − U) d
        and hh: hom_induced n (subtopology X (S − U)) (T − U) (subtopology X
S) T id
                 (homologous_rel_set n (subtopology X (S − U)) (T − U) c)
               = hom_induced n (subtopology X (S − U)) (T − U) (subtopology X
S) T id
                 (homologous_rel_set n (subtopology X (S − U)) (T − U) d)
      then have scc: singular_chain n (subtopology X (S − U)) c
           and  scd: singular_chain n (subtopology X (S − U)) d
        using singular_relcycle by blast+
      have singular_relboundary n (subtopology X (S − U)) (T − U) c
        if srb: singular_relboundary n (subtopology X S) T c
          and src: singular_relcycle n (subtopology X (S − U)) (T − U) c for c
      proof −
        have [simp]: (S − U) ∩ (T − U) = T − U S ∩ T = T
          using ‹T ⊆ S› by blast+
        have c: singular_chain n (subtopology X (S − U)) c
            singular_chain (n − Suc 0) (subtopology X (T − U)) (chain_boundary
n c)
         using that by (auto simp: singular_relcycle_def mod_subset_def subtopol-
ogy_subtopology)
        obtain d e where d: singular_chain (Suc n) (subtopology X S) d
```

```
      and e: singular_chain n (subtopology X T) e
      and dce: chain_boundary (Suc n) d = c + e
    using srb by (auto simp: singular_relboundary_alt subtopology_subtopology)
    obtain m f g where f: singular_chain (Suc n) (subtopology X (S − U)) f
                 and g: singular_chain (Suc n) (subtopology X T) g
                 and dfg: (singular_subdivision (Suc n) ^^ m) d = f + g
      using excised_chain_exists [OF assms d] .
    obtain h where
        h0: ⋀p. h p 0 = (0 :: 'a chain)
      and hdiff: ⋀p c1 c2. h p (c1−c2) = h p c1 − h p c2
      and hSuc: ⋀p X c. singular_chain p X c ⟹ singular_chain (Suc p) X (h
p c)
      and hchain: ⋀p X c. singular_chain p X c
                   ⟹ chain_boundary (Suc p) (h p c) + h (p − Suc 0)
(chain_boundary p c)
                 = (singular_subdivision p ^^ m) c − c
      using chain_homotopic_iterated_singular_subdivision by blast
    have hadd: ⋀p c1 c2. h p (c1 + c2) = h p c1 + h p c2
      by (metis add_diff_cancel diff_add_cancel hdiff)
    define c1 where c1 ≡ f − h n c
    define c2 where c2 ≡ chain_boundary (Suc n) (h n e) − (chain_boundary
(Suc n) g − e)
    show ?thesis
      unfolding singular_relboundary_alt
    proof (intro exI conjI)
      show c1: singular_chain (Suc n) (subtopology X (S − U)) c1
        by (simp add: ‹singular_chain n (subtopology X (S − U)) c› c1_def f
hSuc singular_chain_diff)
      have chain_boundary (Suc n) (chain_boundary (Suc (Suc n)) (h (Suc n)
d) + h n (c+e))
          = chain_boundary (Suc n) (f + g − d)
          using hchain [OF d] by (simp add: dce dfg)
        then have chain_boundary (Suc n) (h n (c + e))
            = chain_boundary (Suc n) f + chain_boundary (Suc n) g − (c + e)
          using chain_boundary_boundary_alt [of Suc n subtopology X S]
          by (simp add: chain_boundary_add chain_boundary_diff d hSuc dce)
        then have chain_boundary (Suc n) (h n c) + chain_boundary (Suc n)
(h n e)
            = chain_boundary (Suc n) f + chain_boundary (Suc n) g − (c + e)
          by (simp add: chain_boundary_add hadd)
      then have ∗: chain_boundary (Suc n) (f − h n c) = c + (chain_boundary
(Suc n) (h n e) − (chain_boundary (Suc n) g − e))
          by (simp add: algebra_simps chain_boundary_diff)
        then show chain_boundary (Suc n) c1 = c + c2
        unfolding c1_def c2_def
          by (simp add: algebra_simps chain_boundary_diff)
        have singular_chain n (subtopology X (S − U)) c2 singular_chain n
(subtopology X T) c2
          using singular_chain_diff c c1 ∗
```

```
            unfolding c1_def c2_def
             apply (metis add_diff_cancel_left' singular_chain_boundary_alt)
          by (simp add: e g hSuc singular_chain_boundary_alt singular_chain_diff)
          then show singular_chain n (subtopology (subtopology X (S − U)) (T
− U)) c2
            by (fastforce simp add: singular_chain_subtopology)
        qed
      qed
      then have singular_relboundary n (subtopology X S) T (c − d) ⟹
                singular_relboundary n (subtopology X (S − U)) (T − U) (c − d)
        using c d singular_relcycle_diff by metis
      with hh show homologous_rel n (subtopology X (S − U)) (T − U) c d
       apply (simp add: hom_induced_chain_map cont c d chain_map_ident [OF
scc] chain_map_ident [OF scd])
        using homologous_rel_set_eq homologous_rel_def by metis
    qed
  next
    have h: homologous_rel_set n (subtopology X S) T a
          ∈ (λx. homologous_rel_set n (subtopology X S) T (chain_map n id x)) `
            singular_relcycle_set n (subtopology X (S − U)) (T − U)
      if a: singular_relcycle n (subtopology X S) T a for a
    proof −
      obtain c' where c': singular_relcycle n (subtopology X (S − U)) (T − U)
c'
                          homologous_rel n (subtopology X S) T a c'
        using a by (blast intro: excised_relcycle_exists [OF assms])
      then have scc': singular_chain n (subtopology X S) c'
        using homologous_rel_singular_chain singular_relcycle that by blast
      then show ?thesis
        apply (rule_tac x=c' in image_eqI)
         apply (auto simp: scc' chain_map_ident [of _ subtopology X S] c' homol-
ogous_rel_set_eq)
        done
    qed
    show hom_induced n (subtopology X (S − U)) (T − U) (subtopology X S) T
id `
          homologous_rel_set n (subtopology X (S − U)) (T − U) `
          singular_relcycle_set n (subtopology X (S − U)) (T − U)
          = homologous_rel_set n (subtopology X S) T ` singular_relcycle_set n
(subtopology X S) T
       apply (simp add: image_comp o_def hom_induced_chain_map_gen cont
TU topspace_subtopology
                      cong: image_cong_simp)
      apply (force simp: cont h singular_relcycle_chain_map)
      done
  qed
qed
```

0.2.3 Additivity axiom

Not in the original Eilenberg-Steenrod list but usually included nowadays, following Milnor's "On Axiomatic Homology Theory".

lemma *iso_chain_group_sum*:
  assumes *disj*: *pairwise disjnt* $\mathcal{U}$ **and** *UU*: $\bigcup\mathcal{U}$ = *topspace X*
    and *subs*: $\bigwedge C\ T.$ ⟦*compactin X C*; *path_connectedin X C*; $T \in \mathcal{U}$; ~ *disjnt C T*⟧ $\Longrightarrow C \subseteq T$
  shows ($\lambda f.$ *sum′ f* $\mathcal{U}$) $\in$ *iso* (*sum_group* $\mathcal{U}$ ($\lambda S.$ *chain_group p* (*subtopology X S*))) (*chain_group p X*)
proof −
  have *pw*: *pairwise* ($\lambda i\ j.$ *disjnt* (*singular_simplex_set p* (*subtopology X i*))
                    (*singular_simplex_set p* (*subtopology X j*))) $\mathcal{U}$
  proof
    fix *S T*
    assume $S \in \mathcal{U}$ $T \in \mathcal{U}$ $S \neq T$
    then show *disjnt* (*singular_simplex_set p* (*subtopology X S*))
              (*singular_simplex_set p* (*subtopology X T*))
      using *nonempty_standard_simplex* [*of p*] *disj*
        by (*fastforce simp*: *pairwise_def disjnt_def singular_simplex_subtopology image_subset_iff*)
  qed
  have $\exists S\in\mathcal{U}.$ *singular_simplex p* (*subtopology X S*) *f*
    if *f*: *singular_simplex p X f* **for** *f*
  proof −
    obtain *x* **where** *x*: $x \in$ *topspace X* $x \in f$ ‘ *standard_simplex p*
     using *f nonempty_standard_simplex* [*of p*] *continuous_map_image_subset_topspace*
      unfolding *singular_simplex_def* **by** *fastforce*
    then obtain *S* **where** $S \in \mathcal{U}$ $x \in S$
      using *UU* **by** *auto*
    have *f* ‘ *standard_simplex p* $\subseteq S$
    proof (*rule subs*)
      have *cont*: *continuous_map* (*subtopology* (*powertop_real UNIV*)
                    (*standard_simplex p*)) *X f*
        using *f singular_simplex_def* **by** *auto*
      show *compactin X* (*f* ‘ *standard_simplex p*)
          by (*simp add*: *compactin_subtopology compactin_standard_simplex image_compactin* [*OF _ cont*])
      show *path_connectedin X* (*f* ‘ *standard_simplex p*)
      by (*simp add*: *path_connectedin_subtopology path_connectedin_standard_simplex path_connectedin_continuous_map_image* [*OF cont*])
      have *standard_simplex p* $\neq$ {}
        by (*simp add*: *nonempty_standard_simplex*)
      then
      show ¬ *disjnt* (*f* ‘ *standard_simplex p*) *S*
        using *x* ‹$x \in S$› **by** (*auto simp*: *disjnt_def*)
    qed (*auto simp*: ‹$S \in \mathcal{U}$›)
    then show *?thesis*
      by (*meson* ‹$S \in \mathcal{U}$› *singular_simplex_subtopology that*)

```
  qed
 then have (⋃ i∈𝒰. singular_simplex_set p (subtopology X i)) = singular_simplex_set
p X
    by (auto simp: singular_simplex_subtopology)
  then show ?thesis
    using iso_free_Abelian_group_sum [OF pw] by (simp add: chain_group_def)
qed

lemma relcycle_group_0_eq_chain_group: relcycle_group 0 X {} = chain_group
0 X
  apply (rule monoid.equality, simp)
     apply (simp_all add: relcycle_group_def chain_group_def)
  by (metis chain_boundary_def singular_cycle)


proposition iso_cycle_group_sum:
  assumes disj: pairwise disjnt 𝒰 and UU: ⋃𝒰 = topspace X
    and subs: ⋀C T. ⟦compactin X C; path_connectedin X C; T ∈ 𝒰; ¬ disjnt C
T⟧ ⟹ C ⊆ T
  shows (λf. sum' f 𝒰) ∈ iso (sum_group 𝒰 (λT. relcycle_group p (subtopology
X T) {}))
                          (relcycle_group p X {})
proof (cases p = 0)
  case True
  then show ?thesis
    by (simp add: relcycle_group_0_eq_chain_group iso_chain_group_sum [OF
assms])
next
  case False
  let ?SG = (sum_group 𝒰 (λT. chain_group p (subtopology X T)))
  let ?PI = (Π_E T∈𝒰. singular_relcycle_set p (subtopology X T) {})
 have (λf. sum' f 𝒰) ∈ Group.iso (subgroup_generated ?SG (carrier ?SG ∩ ?PI))
                    (subgroup_generated (chain_group p X) (singular_relcycle_set
p X {}))
  proof (rule group_hom.iso_between_subgroups)
    have iso: (λf. sum' f 𝒰) ∈ Group.iso ?SG (chain_group p X)
      by (auto simp: assms iso_chain_group_sum)
    then show group_hom ?SG (chain_group p X) (λf. sum' f 𝒰)
     by (auto simp: iso_imp_homomorphism group_hom_def group_hom_axioms_def)
     have B: sum' f 𝒰 ∈ singular_relcycle_set p X {} ⟷ f ∈ (carrier ?SG ∩
?PI)
      if f ∈ (carrier ?SG) for f
    proof -
      have f: ⋀S. S ∈ 𝒰 ⟶ singular_chain p (subtopology X S) (f S)
              f ∈ extensional 𝒰 finite {i ∈ 𝒰. f i ≠ 0}
        using that by (auto simp: carrier_sum_group PiE_def Pi_def)
      then have rfin: finite {S ∈ 𝒰. restrict (chain_boundary p ∘ f) 𝒰 S ≠ 0}
        by (auto elim: rev_finite_subset)
      have chain_boundary p ((∑x | x ∈ 𝒰 ∧ f x ≠ 0. f x)) = 0
```

```
        ⟷ (∀ S ∈ 𝒰. chain_boundary p (f S) = 0) (is ?cb = 0 ⟷ ?rhs)
      proof
        assume ?cb = 0
        moreover have ?cb = sum' (λS. chain_boundary p (f S)) 𝒰
          unfolding sum.G_def using rfin f
        by (force simp: chain_boundary_sum intro: sum.mono_neutral_right cong:
conj_cong)
        ultimately have eq0: sum' (λS. chain_boundary p (f S)) 𝒰 = 0
          by simp
        have (λf. sum' f 𝒰) ∈ hom (sum_group 𝒰 (λS. chain_group (p − Suc 0)
(subtopology X S)))
                                  (chain_group (p − Suc 0) X)
          and inj: inj_on (λf. sum' f 𝒰) (carrier (sum_group 𝒰 (λS. chain_group
(p − Suc 0) (subtopology X S))))
          using iso_chain_group_sum [OF assms, of p−1] by (auto simp: iso_def
bij_betw_def)
        then have eq: ⟦f ∈ (Π_E i∈𝒰. singular_chain_set (p − Suc 0) (subtopology
X i));
                  finite {S ∈ 𝒰. f S ≠ 0}; sum' f 𝒰 = 0; S ∈ 𝒰⟧ ⟹ f S = 0 for f S
        apply (simp add: group_hom_def group_hom_axioms_def group_hom.inj_on_one_iff
[of _ chain_group (p−1) X])
          apply (auto simp: carrier_sum_group fun_eq_iff that)
          done
        show ?rhs
        proof clarify
          fix S assume S ∈ 𝒰
          then show chain_boundary p (f S) = 0
            using eq [of restrict (chain_boundary p ∘ f) 𝒰 S] rfin f eq0
            by (simp add: singular_chain_boundary cong: conj_cong)
        qed
      next
        assume ?rhs
        then show ?cb = 0
          by (force simp: chain_boundary_sum intro: sum.mono_neutral_right)
      qed
      moreover
      have (⋀S. S ∈ 𝒰 ⟶ singular_chain p (subtopology X S) (f S))
          ⟹ singular_chain p X (∑ x | x ∈ 𝒰 ∧ f x ≠ 0. f x)
        by (metis (no_types, lifting) mem_Collect_eq singular_chain_subtopology
singular_chain_sum)
      ultimately show ?thesis
          using f by (auto simp: carrier_sum_group sum.G_def singular_cycle
PiE_iff)
    qed
    have singular_relcycle_set p X {} ⊆ carrier (chain_group p X)
      using subgroup.subset subgroup_singular_relcycle by blast
    then show (λf. sum' f 𝒰) ‘ (carrier ?SG ∩ ?PI) = singular_relcycle_set p X
{}
      using iso B
```

```
      apply (auto simp: iso_def bij_betw_def)
      apply (force simp: singular_relcycle)
      done
  qed (auto simp: assms iso_chain_group_sum)
  then show ?thesis
   by (simp add: relcycle_group_def sum_group_subgroup_generated subgroup_singular_relcycle)
qed


proposition homology_additivity_axiom_gen:
  assumes disj: pairwise disjnt 𝒰 and UU: ⋃𝒰 = topspace X
    and subs: ⋀C T. ⟦compactin X C; path_connectedin X C; T ∈ 𝒰; ¬ disjnt C
T⟧ ⟹ C ⊆ T
  shows (λx. gfinprod (homology_group p X)
                      (λV. hom_induced p (subtopology X V) {} X {} id (x V)) 𝒰)
    ∈ iso (sum_group 𝒰 (λS. homology_group p (subtopology X S))) (homology_group
p X)
     (is ?h ∈ iso ?SG ?HG)
proof (cases p < 0)
  case True
   then have [simp]: gfinprod (singleton_group undefined) (λv. undefined) 𝒰 =
undefined
   by (metis Pi_I carrier_singleton_group comm_group_def comm_monoid.gfinprod_closed
singletonD singleton_abelian_group)
  show ?thesis
    using True
    apply (simp add: iso_def relative_homology_group_def hom_induced_trivial
carrier_sum_group)
    apply (auto simp: singleton_group_def bij_betw_def inj_on_def fun_eq_iff)
    done
next
  case False
  then obtain n where peq: p = int n
    by (metis int_ops(1) linorder_neqE_linordered_idom pos_int_cases)
  interpret comm_group homology_group p X
    by (rule abelian_homology_group)
  show ?thesis
  proof (simp add: iso_def bij_betw_def, intro conjI)
    show ?h ∈ hom ?SG ?HG
      by (rule hom_group_sum) (simp_all add: hom_induced_hom)
    then interpret group_hom ?SG ?HG ?h
      by (simp add: group_hom_def group_hom_axioms_def)
    have carrSG: carrier ?SG
        = (λx. λS∈𝒰. homologous_rel_set n (subtopology X S) {} (x S))
          ‘ (carrier (sum_group 𝒰 (λS. relcycle_group n (subtopology X S) {}))) (is
?lhs = ?rhs)
    proof
      show ?lhs ⊆ ?rhs
      proof (clarsimp simp: carrier_sum_group carrier_relative_homology_group
```

peq)
fix *z*
assume *z*: $z \in (\Pi_E\ S{\in}\mathcal{U}.\ homologous\_rel\_set\ n\ (subtopology\ X\ S)\ \{\}$ ' *singular_relcycle_set n* (*subtopology X S*) {})
and *fin*: *finite* $\{S \in \mathcal{U}.\ z\ S \neq singular\_relboundary\_set\ n\ (subtopology\ X\ S)\ \{\}\}$
then obtain *c* **where** *c*: $\forall S{\in}\mathcal{U}.$ *singular_relcycle n* (*subtopology X S*) {} (*c S*)
$\wedge$ *z S* = *homologous_rel_set n* (*subtopology X S*) {} (*c S*)
by (*simp add*: *PiE_def Pi_def image_def*) *metis*
let *?f* = $\lambda S{\in}\mathcal{U}.$ *if singular_relboundary n* (*subtopology X S*) {} (*c S*) *then 0 else c S*
have *z* = ($\lambda S{\in}\mathcal{U}.$ *homologous_rel_set n* (*subtopology X S*) {} (*?f S*))
apply (*simp_all add*: *c fun_eq_iff PiE_arb* [*OF z*])
apply (*metis homologous_rel_eq_relboundary singular_boundary singular_relboundary_0*)
done
moreover have *?f* $\in (\Pi_E\ i{\in}\mathcal{U}.$ *singular_relcycle_set n* (*subtopology X i*) {})
by (*simp add*: *c fun_eq_iff PiE_arb* [*OF z*])
moreover have *finite* $\{i \in \mathcal{U}.\ ?f\ i \neq 0\}$
apply (*rule finite_subset* [*OF _ fin*])
using *z* **apply** (*clarsimp simp*: *PiE_def Pi_def image_def*)
by (*metis c homologous_rel_set_eq_relboundary singular_boundary*)
ultimately
show $z \in (\lambda x.\ \lambda S{\in}\mathcal{U}.$ *homologous_rel_set n* (*subtopology X S*) {} (*x S*)) '
$\{x \in \Pi_E\ i{\in}\mathcal{U}.$ *singular_relcycle_set n* (*subtopology X i*) {}. *finite* $\{i \in \mathcal{U}.\ x\ i \neq 0\}\}$
by *blast*
qed
show *?rhs* $\subseteq$ *?lhs*
by (*force simp*: *peq carrier_sum_group carrier_relative_homology_group homologous_rel_set_eq_relboundary*
elim: *rev_finite_subset*)
qed
have *gf*: *gfinprod* (*homology_group p X*)
($\lambda V.$ *hom_induced n* (*subtopology X V*) {} *X* {} *id*
(($\lambda S{\in}\mathcal{U}.$ *homologous_rel_set n* (*subtopology X S*) {} (*z S*)) *V*)) $\mathcal{U}$
= *homologous_rel_set n X* {} (*sum' z* $\mathcal{U}$) (**is** *?lhs* = *?rhs*)
if *z*: $z \in$ *carrier* (*sum_group* $\mathcal{U}$ ($\lambda S.$ *relcycle_group n* (*subtopology X S*) {})) **for** *z*
proof −
have *hom_pi*: ($\lambda S.$ *homologous_rel_set n X* {} (*z S*)) $\in \mathcal{U} \to$ *carrier* (*homology_group p X*)
apply (*rule Pi_I*)
using *z*
apply (*force simp*: *peq carrier_sum_group carrier_relative_homology_group singular_chain_subtopology singular_cycle*)

```
      done
    have fin: finite {S ∈ 𝒰. z S ≠ 0}
      using that by (force simp: carrier_sum_group)
     have ?lhs = gfinprod (homology_group p X) (λS. homologous_rel_set n X
{} (z S)) 𝒰
      apply (rule gfinprod_cong [OF refl Pi_I])
       apply (simp add: hom_induced_carrier peq)
      using that
          apply (auto simp: peq simp_implies_def carrier_sum_group PiE_def
Pi_def chain_map_ident singular_cycle hom_induced_chain_map)
      done
    also have ... = gfinprod (homology_group p X)
                          (λS. homologous_rel_set n X {} (z S)) {S ∈ 𝒰. z S ≠ 0}
      apply (rule gfinprod_mono_neutral_cong_right, simp_all add: hom_pi)
      apply (simp add: relative_homology_group_def peq)
      apply (metis homologous_rel_eq_relboundary singular_relboundary_0)
      done
    also have ... = ?rhs
    proof –
      have gfinprod (homology_group p X) (λS. homologous_rel_set n X {} (z
S)) ℱ
        = homologous_rel_set n X {} (sum z ℱ)
        if finite ℱ ℱ ⊆ {S ∈ 𝒰. z S ≠ 0} for ℱ
        using that
      proof (induction ℱ)
        case empty
        have 1_{homology_group p X} = homologous_rel_set n X {} 0
          apply (simp add: relative_homology_group_def peq)
          by (metis diff_zero homologous_rel_def homologous_rel_sym)
        then show ?case
          by simp
      next
        case (insert S ℱ)
         with z have pi: (λS. homologous_rel_set n X {} (z S)) ∈ ℱ → carrier
(homology_group p X)
          homologous_rel_set n X {} (z S) ∈ carrier (homology_group p X)
         by (force simp: peq carrier_sum_group carrier_relative_homology_group
singular_chain_subtopology singular_cycle)+
        have hom: homologous_rel_set n X {} (z S) ∈ carrier (homology_group
p X)
          using insert z
         by (force simp: peq carrier_sum_group carrier_relative_homology_group
singular_chain_subtopology singular_cycle)
        show ?case
          using insert z
        proof (simp add: pi)
          show homologous_rel_set n X {} (z S) ⊗_{homology_group p X} homolo-
gous_rel_set n X {} (sum z ℱ)
            = homologous_rel_set n X {} (z S + sum z ℱ)
```

using *insert z* **apply** (*auto simp*: *peq homologous_rel_add mult_relative_homology_group*)
by (*metis* (*no_types*, *lifting*) *diff_add_cancel diff_diff_eq2 homologous_rel_def homologous_rel_refl*)
qed
qed
with *fin* **show** *?thesis*
by (*simp add*: *sum.G_def*)
qed
finally show *?thesis* **.**
qed
show *inj_on ?h* (*carrier ?SG*)
proof (*clarsimp simp add*: *inj_on_one_iff*)
fix *x*
assume *x*: $x \in$ *carrier* (*sum_group* $\mathcal{U}$ (λS. *homology_group p* (*subtopology X S*)))
and *1*: *gfinprod* (*homology_group p X*) (λV. *hom_induced p* (*subtopology X V*) {} *X* {} *id* (*x V*)) $\mathcal{U}$
= $\mathbf{1}_{homology\_group\ p\ X}$
have *feq*: ($\lambda S \in \mathcal{U}$. *homologous_rel_set n* (*subtopology X S*) {} (*z S*))
= ($\lambda S \in \mathcal{U}$. $\mathbf{1}_{homology\_group\ p\ (subtopology\ X\ S)}$)
if *z*: $z \in$ *carrier* (*sum_group* $\mathcal{U}$ (λS. *relcycle_group n* (*subtopology X S*) {}))
and *eq*: *homologous_rel_set n X* {} (*sum' z* $\mathcal{U}$) = $\mathbf{1}_{homology\_group\ p\ X}$
for *z*
proof −
have $z \in (\Pi_E\ S \in \mathcal{U}$. *singular_relcycle_set n* (*subtopology X S*) {}) *finite* {$S \in \mathcal{U}$. $z\ S \neq 0$}
using *z* **by** (*auto simp*: *carrier_sum_group*)
have *singular_relboundary n X* {} (*sum' z* $\mathcal{U}$)
using *eq singular_chain_imp_relboundary* **by** (*auto simp*: *relative_homology_group_def peq*)
then obtain *d* **where** *scd*: *singular_chain* (*Suc n*) *X d* **and** *cbd*: *chain_boundary* (*Suc n*) *d* = *sum' z* $\mathcal{U}$
by (*auto simp*: *singular_boundary*)
have $*$: $\exists\, d$. *singular_chain* (*Suc n*) (*subtopology X S*) *d* $\wedge$ *chain_boundary* (*Suc n*) *d* = *z S*
if $S \in \mathcal{U}$ **for** *S*
proof −
have *inj'*: *inj_on* (λf. *sum' f* $\mathcal{U}$) {$x \in \Pi_E\ S \in \mathcal{U}$. *singular_chain_set* (*Suc n*) (*subtopology X S*). *finite* {$S \in \mathcal{U}$. $x\ S \neq 0$}}
using *iso_chain_group_sum* [*OF assms*, *of Suc n*]
by (*simp add*: *iso_iff_mon_epi mon_def carrier_sum_group*)
obtain *w* **where** *w*: $w \in (\Pi_E\ S \in \mathcal{U}$. *singular_chain_set* (*Suc n*) (*subtopology X S*))
and *finw*: *finite* {$S \in \mathcal{U}$. $w\ S \neq 0$}
and *deq*: *d* = *sum' w* $\mathcal{U}$
using *iso_chain_group_sum* [*OF assms*, *of Suc n*] *scd*
by (*auto simp*: *iso_iff_mon_epi epi_def carrier_sum_group set_eq_iff*)
with ‹$S \in \mathcal{U}$› **have** *scwS*: *singular_chain* (*Suc n*) (*subtopology X S*) (*w S*)
by *blast*

have *inj_on* (λ*f*. *sum′ f* 𝒰) {*x* ∈ Π_E *S*∈𝒰. *singular_chain_set n* (*subtopology X S*). *finite* {*S* ∈ 𝒰. *x S* ≠ *0*}}
using *iso_chain_group_sum* [*OF assms*, *of n*]
by (*simp add*: *iso_iff_mon_epi mon_def carrier_sum_group*)
then have (λ*S*∈𝒰. *chain_boundary* (*Suc n*) (*w S*)) = *z*
proof (*rule inj_onD*)
have *sum′* (λ*S*∈𝒰. *chain_boundary* (*Suc n*) (*w S*)) 𝒰 = *sum′* (*chain_boundary* (*Suc n*) ∘ *w*) {*S* ∈ 𝒰. *w S* ≠ *0*}
by (*auto simp*: *o_def intro*: *sum.mono_neutral_right′*)
also have ... = *chain_boundary* (*Suc n*) *d*
by (*auto simp*: *sum.G_def deq chain_boundary_sum finw intro*: *finite_subset* [*OF _ finw*] *sum.mono_neutral_left*)
finally show *sum′* (λ*S*∈𝒰. *chain_boundary* (*Suc n*) (*w S*)) 𝒰 = *sum′ z* 𝒰
by (*simp add*: *cbd*)
show (λ*S*∈𝒰. *chain_boundary* (*Suc n*) (*w S*)) ∈ {*x* ∈ Π_E *S*∈𝒰. *singular_chain_set n* (*subtopology X S*). *finite* {*S* ∈ 𝒰. *x S* ≠ *0*}}
using *w* **by** (*auto simp*: *PiE_iff singular_chain_boundary_alt cong*: *rev_conj_cong intro*: *finite_subset* [*OF _ finw*])
show *z* ∈ {*x* ∈ Π_E *S*∈𝒰. *singular_chain_set n* (*subtopology X S*). *finite* {*S* ∈ 𝒰. *x S* ≠ *0*}}
using *z* **by** (*simp_all add*: *carrier_sum_group PiE_iff singular_cycle*)
qed
with ‹*S* ∈ 𝒰› *scwS* **show** *?thesis*
by *force*
qed
show *?thesis*
apply (*rule restrict_ext*)
using *that* ∗
apply (*simp add*: *singular_boundary relative_homology_group_def homologous_rel_set_eq_relboundary peq*)
done
qed
show *x* = (λ*S*∈𝒰. $\mathbf{1}_{homology\_group\ p\ (subtopology\ X\ S)}$)
using *x 1 carrSG gf*
by (*auto simp*: *peq feq*)
qed
show *?h* ‘ *carrier ?SG* = *carrier ?HG*
proof *safe*
fix *A*
assume *A* ∈ *carrier* (*homology_group p X*)
then obtain *y* **where** *y*: *singular_relcycle n X* {} *y* **and** *xeq*: *A* = *homologous_rel_set n X* {} *y*
by (*auto simp*: *peq carrier_relative_homology_group*)
then obtain *x* **where** *x* ∈ *carrier* (*sum_group* 𝒰 (λ*T*. *relcycle_group n* (*subtopology X T*) {}))
y = *sum′ x* 𝒰
using *iso_cycle_group_sum* [*OF assms*, *of n*] *that* **by** (*force simp*: *iso_iff_mon_epi epi_def*)

```
    then show A ∈ (λx. gfinprod (homology_group p X) (λV. hom_induced p
(subtopology X V) {} X {} id (x V)) 𝒰) `
          carrier (sum_group 𝒰 (λS. homology_group p (subtopology X S)))
     apply (simp add: carrSG image_comp o_def xeq)
     apply (simp add: hom_induced_carrier peq flip: gf cong: gfinprod_cong)
     done
   qed auto
  qed
qed


corollary homology_additivity_axiom:
  assumes disj: pairwise disjnt 𝒰 and UU: ⋃𝒰 = topspace X
   and ope: ⋀v. v ∈ 𝒰 ⟹ openin X v
 shows (λx. gfinprod (homology_group p X)
                 (λv. hom_induced p (subtopology X v) {} X {} id (x v)) 𝒰)
   ∈ iso (sum_group 𝒰 (λS. homology_group p (subtopology X S))) (homology_group
p X)
proof (rule homology_additivity_axiom_gen [OF disj UU])
  fix C T
  assume
    compactin X C and
    path_connectedin X C and
    T ∈ 𝒰 and
    ¬ disjnt C T
  then have C ⊆ topspace X
    and *: ⋀B. ⟦openin X T; T ∩ B ∩ C = {}; C ⊆ T ∪ B; openin X B⟧ ⟹ B
∩ C = {}
    apply (auto simp: connectedin disjnt_def dest!: path_connectedin_imp_connectedin,
blast)
    done
  have C ⊆ Union 𝒰
    using ‹C ⊆ topspace X› UU by blast
  moreover have ⋃ (𝒰 − {T}) ∩ C = {}
  proof (rule *)
    show T ∩ ⋃ (𝒰 − {T}) ∩ C = {}
      using ‹T ∈ 𝒰› disj disjointD by fastforce
    show C ⊆ T ∪ ⋃ (𝒰 − {T})
      using ‹C ⊆ ⋃ 𝒰› by fastforce
  qed (auto simp: ‹T ∈ 𝒰› ope)
  ultimately show C ⊆ T
    by blast
qed
```

0.2.4 Special properties of singular homology

In particular: the zeroth homology group is isomorphic to the free abelian group generated by the path components. So, the "coefficient group" is the integers.

```
lemma iso_integer_zeroth_homology_group_aux:
  assumes X: path_connected_space X and f: singular_simplex 0 X f and f':
singular_simplex 0 X f'
  shows homologous_rel 0 X {} (frag_of f) (frag_of f')
proof -
  let ?p = λj. if j = 0 then 1 else 0
  have f ?p ∈ topspace X f' ?p ∈ topspace X
  using assms by (auto simp: singular_simplex_def continuous_map_def)
  then obtain g where g: pathin X g
               and g0: g 0 = f ?p
               and g1: g 1 = f' ?p
    using assms by (force simp: path_connected_space_def)
  then have contg: continuous_map (subtopology euclideanreal {0..1}) X g
    by (simp add: pathin_def)
 have singular_chain (Suc 0) X (frag_of (restrict (g ∘ (λx. x 0)) (standard_simplex
1)))
  proof -
    have continuous_map (subtopology (powertop_real UNIV) (standard_simplex
(Suc 0)))
                        (top_of_set {0..1}) (λx. x 0)
      apply (auto simp: continuous_map_in_subtopology g)
          apply (metis (mono_tags) UNIV_I continuous_map_from_subtopology
continuous_map_product_projection)
       apply (simp_all add: standard_simplex_def)
      done
    moreover have continuous_map (top_of_set {0..1}) X g
      using contg by blast
    ultimately show ?thesis
      by (force simp: singular_chain_of chain_boundary_of singular_simplex_def
continuous_map_compose)
  qed
  moreover
 have chain_boundary (Suc 0) (frag_of (restrict (g ∘ (λx. x 0)) (standard_simplex
1))) =
      frag_of f - frag_of f'
  proof -
    have singular_face (Suc 0) 0 (g ∘ (λx. x 0)) = f
         singular_face (Suc 0) (Suc 0) (g ∘ (λx. x 0)) = f'
      using assms
      by (auto simp: singular_face_def singular_simplex_def extensional_def sim-
plical_face_def standard_simplex_0 g0 g1)
    then show ?thesis
      by (simp add: singular_chain_of chain_boundary_of)
  qed
  ultimately
  show ?thesis
    by (auto simp: homologous_rel_def singular_boundary)
qed
```

```
proposition iso_integer_zeroth_homology_group:
  assumes X: path_connected_space X and f: singular_simplex 0 X f
  shows pow (homology_group 0 X) (homologous_rel_set 0 X {} (frag_of f))
      ∈ iso integer_group (homology_group 0 X) (is pow ?H ?q ∈ iso _ ?H)
proof -
  have srf: singular_relcycle 0 X {} (frag_of f)
    by (simp add: chain_boundary_def f singular_chain_of singular_cycle)
  then have qcarr: ?q ∈ carrier ?H
    by (simp add: carrier_relative_homology_group_0)
  have 1: homologous_rel_set 0 X {} a ∈ range (λn. homologous_rel_set 0 X {}
(frag_cmul n (frag_of f)))
    if singular_relcycle 0 X {} a for a
  proof -
    have singular_chain 0 X d ⟹
          homologous_rel_set 0 X {} d ∈ range (λn. homologous_rel_set 0 X {}
(frag_cmul n (frag_of f))) for d
      unfolding singular_chain_def
    proof (induction d rule: frag_induction)
      case zero
      then show ?case
        by (metis frag_cmul_zero rangeI)
    next
      case (one x)
      then have ∃i. homologous_rel_set 0 X {} (frag_cmul i (frag_of f))
                = homologous_rel_set 0 X {} (frag_of x)
         by (metis (no_types) iso_integer_zeroth_homology_group_aux [OF X] f
frag_cmul_one homologous_rel_eq mem_Collect_eq)
      with one show ?case
        by auto
    next
      case (diff a b)
      then obtain c d where
         homologous_rel 0 X {} (a - b) (frag_cmul c (frag_of f) - frag_cmul d
(frag_of f))
       using homologous_rel_diff by (fastforce simp add: homologous_rel_set_eq)
      then show ?case
         by (rule_tac x=c-d in image_eqI) (auto simp: homologous_rel_set_eq
frag_cmul_diff_distrib)
    qed
    with that show ?thesis
      unfolding singular_relcycle_def by blast
  qed
  have 2: n = 0
   if homologous_rel_set 0 X {} (frag_cmul n (frag_of f)) = 𝟭_(relative_homology_group 0 X {})
    for n
  proof -
    have singular_chain (Suc 0) X d
        ⟹ frag_extend (λx. frag_of f) (chain_boundary (Suc 0) d) = 0 for d
```

```
      unfolding singular_chain_def
    proof (induction d rule: frag_induction)
      case (one x)
      then show ?case
        by (simp add: frag_extend_diff chain_boundary_of)
    next
      case (diff a b)
      then show ?case
        by (simp add: chain_boundary_diff frag_extend_diff)
    qed auto
    with that show ?thesis
        by (force simp: singular_boundary relative_homology_group_def homolo-
gous_rel_set_eq_relboundary frag_extend_cmul)
  qed
  interpret GH : group_hom integer_group ?H ([^]?H) ?q
   by (simp add: group_hom_def group_hom_axioms_def qcarr group.hom_integer_group_pow)
  have eq: pow ?H ?q = (λn. homologous_rel_set 0 X {} (frag_cmul n (frag_of
f)))
  proof
    fix n
    have frag_of f
          ∈ carrier (subgroup_generated
            (free_Abelian_group (singular_simplex_set 0 X)) (singular_relcycle_set
0 X {}))
        by (metis carrier_relcycle_group chain_group_def mem_Collect_eq relcy-
cle_group_def srf)
    then have ff: frag_of f [^]relcycle_group 0 X {} n = frag_cmul n (frag_of f)
     by (simp add: relcycle_group_def chain_group_def group.int_pow_subgroup_generated
f)
    show pow ?H ?q n = homologous_rel_set 0 X {} (frag_cmul n (frag_of f))
      apply (rule subst [OF right_coset_singular_relboundary])
      apply (simp add: relative_homology_group_def)
     apply (simp add: srf ff normal.FactGroup_int_pow normal_subgroup_singular_relboundary_relcycle)
      done
  qed
  show ?thesis
    apply (subst GH.iso_iff)
    apply (simp add: eq)
    apply (auto simp: carrier_relative_homology_group_0 1 2)
    done
qed


corollary isomorphic_integer_zeroth_homology_group:
  assumes X: path_connected_space X topspace X ≠ {}
  shows homology_group 0 X ≅ integer_group
proof –
  obtain a where a: a ∈ topspace X
    using assms by blast
```

have *singular_simplex 0 X* (*restrict* ($\lambda x.$ *a*) (*standard_simplex 0*))
by (*simp add*: *singular_simplex_def a*)
then show *?thesis*
using *X group.iso_sym group_integer_group is_isoI iso_integer_zeroth_homology_group*
by *blast*
qed

corollary *homology_coefficients*:
topspace X = {*a*} $\Longrightarrow$ *homology_group 0 X* $\cong$ *integer_group*
using *isomorphic_integer_zeroth_homology_group path_connectedin_topspace*
by *fastforce*

proposition *zeroth_homology_group*:
homology_group 0 X $\cong$ *free_Abelian_group* (*path_components_of X*)
proof –
obtain *h* **where** *h*: *h* $\in$ *iso* (*sum_group* (*path_components_of X*) ($\lambda S.$ *homology_group 0* (*subtopology X S*)))
(*homology_group 0 X*)
proof (*rule that* [*OF homology_additivity_axiom_gen*])
show *disjoint* (*path_components_of X*)
by (*simp add*: *pairwise_disjoint_path_components_of*)
show $\bigcup$(*path_components_of X*) = *topspace X*
by (*rule Union_path_components_of*)
next
fix *C T*
assume *path_connectedin X C T* $\in$ *path_components_of X* $\neg$ *disjnt C T*
then show *C* $\subseteq$ *T*
by (*metis path_components_of_maximal disjnt_sym*)+
qed
have *homology_group 0 X* $\cong$ *sum_group* (*path_components_of X*) ($\lambda S.$ *homology_group 0* (*subtopology X S*))
by (*rule group.iso_sym*) (*use h is_iso_def* **in** *auto*)
also have ... $\cong$ *sum_group* (*path_components_of X*) ($\lambda i.$ *integer_group*)
proof (*rule iso_sum_groupI*)
show *homology_group 0* (*subtopology X i*) $\cong$ *integer_group* **if** *i* $\in$ *path_components_of X* **for** *i*
by (*metis that isomorphic_integer_zeroth_homology_group nonempty_path_components_of path_connectedin_def path_connectedin_path_components_of topspace_subtopology_subset*)
qed *auto*
also have ... $\cong$ *free_Abelian_group* (*path_components_of X*)
using *path_connectedin_path_components_of nonempty_path_components_of*
by (*simp add*: *isomorphic_sum_integer_group path_connectedin_def*)
finally show *?thesis* **.**
qed

lemma *isomorphic_homology_imp_path_components*:
assumes *homology_group 0 X* $\cong$ *homology_group 0 Y*

shows *path_components_of X* ≈ *path_components_of Y*
proof –
have *free_Abelian_group* (*path_components_of X*) ≅ *homology_group 0 X*
by (*rule group.iso_sym*) (*auto simp*: *zeroth_homology_group*)
also have ... ≅ *homology_group 0 Y*
by (*rule assms*)
also have ... ≅ *free_Abelian_group* (*path_components_of Y*)
by (*rule zeroth_homology_group*)
finally have *free_Abelian_group* (*path_components_of X*) ≅ *free_Abelian_group* (*path_components_of Y*) **.**
then show *?thesis*
by (*simp add*: *isomorphic_free_Abelian_groups*)
qed

lemma *isomorphic_homology_imp_path_connectedness*:
assumes *homology_group 0 X* ≅ *homology_group 0 Y*
shows *path_connected_space X* ⟷ *path_connected_space Y*
proof –
obtain *h* **where** *h*: *bij_betw h* (*path_components_of X*) (*path_components_of Y*)
using *assms isomorphic_homology_imp_path_components eqpoll_def* **by** *blast*
have *1*: *path_components_of X* ⊆ {*a*} ⟹ *path_components_of Y* ⊆ {*h a*} **for** *a*
using *h* **unfolding** *bij_betw_def* **by** *blast*
have *2*: *path_components_of Y* ⊆ {*a*}
⟹ *path_components_of X* ⊆ {*inv_into* (*path_components_of X*) *h a*}
for *a*
using *h* [*THEN bij_betw_inv_into*] **unfolding** *bij_betw_def* **by** *blast*
show *?thesis*
unfolding *path_connected_space_iff_components_subset_singleton*
by (*blast intro*: *dest*: *1 2*)
qed

0.2.5 More basic properties of homology groups, deduced from the E-S axioms

lemma *trivial_homology_group*:
$p < 0$ ⟹ *trivial_group*(*homology_group p X*)
by *simp*

lemma *hom_induced_empty_hom*:
(*hom_induced p X* {} *X′* {} *f*) ∈ *hom* (*homology_group p X*) (*homology_group p X′*)
by (*simp add*: *hom_induced_hom*)

lemma *hom_induced_compose_empty*:
⟦*continuous_map X Y f*; *continuous_map Y Z g*⟧
⟹ *hom_induced p X* {} *Z* {} (*g* ∘ *f*) = *hom_induced p Y* {} *Z* {} *g* ∘

hom_induced p X {} Y {} f
by (*simp add*: *hom_induced_compose*)

lemma *homology_homotopy_empty*:
homotopic_with (*λh. True*) *X Y f g* ⟹ *hom_induced p X {} Y {} f* = *hom_induced p X {} Y {} g*
by (*simp add*: *homology_homotopy_axiom*)

lemma *homotopy_equivalence_relative_homology_group_isomorphisms*:
assumes *contf*: *continuous_map X Y f* **and** *fim*: *f ‘ S ⊆ T*
and *contg*: *continuous_map Y X g* **and** *gim*: *g ‘ T ⊆ S*
and *gf*: *homotopic_with* (*λh. h ‘ S ⊆ S*) *X X* (*g ∘ f*) *id*
and *fg*: *homotopic_with* (*λk. k ‘ T ⊆ T*) *Y Y* (*f ∘ g*) *id*
shows *group_isomorphisms* (*relative_homology_group p X S*) (*relative_homology_group p Y T*)
(*hom_induced p X S Y T f*) (*hom_induced p Y T X S g*)
unfolding *group_isomorphisms_def*
proof (*intro conjI ballI*)
fix *x*
assume *x*: *x ∈ carrier* (*relative_homology_group p X S*)
then show *hom_induced p Y T X S g* (*hom_induced p X S Y T f x*) = *x*
using *homology_homotopy_axiom* [*OF gf*, *of p*]
apply (*simp add*: *hom_induced_compose* [*OF contf fim contg gim*])
apply (*metis comp_apply hom_induced_id*)
done
next
fix *y*
assume *y ∈ carrier* (*relative_homology_group p Y T*)
then show *hom_induced p X S Y T f* (*hom_induced p Y T X S g y*) = *y*
using *homology_homotopy_axiom* [*OF fg*, *of p*]
apply (*simp add*: *hom_induced_compose* [*OF contg gim contf fim*])
apply (*metis comp_apply hom_induced_id*)
done
qed (*auto simp*: *hom_induced_hom*)

lemma *homotopy_equivalence_relative_homology_group_isomorphism*:
assumes *continuous_map X Y f* **and** *fim*: *f ‘ S ⊆ T*
and *continuous_map Y X g* **and** *gim*: *g ‘ T ⊆ S*
and *homotopic_with* (*λh. h ‘ S ⊆ S*) *X X* (*g ∘ f*) *id*
and *homotopic_with* (*λk. k ‘ T ⊆ T*) *Y Y* (*f ∘ g*) *id*
shows (*hom_induced p X S Y T f*) ∈ *iso* (*relative_homology_group p X S*) (*relative_homology_group p Y T*)
using *homotopy_equivalence_relative_homology_group_isomorphisms* [*OF assms*] *group_isomorphisms_imp_iso*
by *metis*

lemma *homotopy_equivalence_homology_group_isomorphism*:
assumes *continuous_map X Y f*

```
    and continuous_map Y X g
    and homotopic_with (λh. True) X X (g ∘ f) id
    and homotopic_with (λk. True) Y Y (f ∘ g) id
  shows (hom_induced p X {} Y {} f) ∈ iso (homology_group p X) (homology_group
p Y)
  apply (rule homotopy_equivalence_relative_homology_group_isomorphism)
  using assms by auto

lemma homotopy_equivalent_space_imp_isomorphic_relative_homology_groups:
  assumes continuous_map X Y f and fim: f ` S ⊆ T
    and continuous_map Y X g and gim: g ` T ⊆ S
    and homotopic_with (λh. h ` S ⊆ S) X X (g ∘ f) id
    and homotopic_with (λk. k ` T ⊆ T) Y Y (f ∘ g) id
   shows relative_homology_group p X S ≅ relative_homology_group p Y T
 using homotopy_equivalence_relative_homology_group_isomorphism [OF assms]
  unfolding is_iso_def by blast

lemma homotopy_equivalent_space_imp_isomorphic_homology_groups:
   X homotopy_equivalent_space Y ⟹ homology_group p X ≅ homology_group
p Y
  unfolding homotopy_equivalent_space_def
 by (auto intro: homotopy_equivalent_space_imp_isomorphic_relative_homology_groups)

lemma homeomorphic_space_imp_isomorphic_homology_groups:
   X homeomorphic_space Y ⟹ homology_group p X ≅ homology_group p Y
 by (simp add: homeomorphic_imp_homotopy_equivalent_space homotopy_equivalent_space_imp_isomorphic_ho

lemma trivial_relative_homology_group_gen:
  assumes continuous_map X (subtopology X S) f
   homotopic_with (λh. True) (subtopology X S) (subtopology X S) f id
   homotopic_with (λk. True) X X f id
  shows trivial_group(relative_homology_group p X S)
proof (rule exact_seq_imp_triviality)
  show exact_seq ([homology_group (p−1) X,
                   homology_group (p−1) (subtopology X S),
                     relative_homology_group p X S, homology_group p X, homol-
ogy_group p (subtopology X S)],
                  [hom_induced (p−1) (subtopology X S) {} X {} id,
                   hom_boundary p X S,
                   hom_induced p X {} X S id,
                   hom_induced p (subtopology X S) {} X {} id])
     using homology_exactness_axiom_1 homology_exactness_axiom_2 homol-
ogy_exactness_axiom_3
   by (metis exact_seq_cons_iff)
next
  show hom_induced p (subtopology X S) {} X {} id
       ∈ iso (homology_group p (subtopology X S)) (homology_group p X)
      hom_induced (p − 1) (subtopology X S) {} X {} id
       ∈ iso (homology_group (p − 1) (subtopology X S)) (homology_group (p −
```

1) *X*)
using *assms*
by (*auto intro*: *homotopy_equivalence_relative_homology_group_isomorphism*)
qed

lemma *trivial_relative_homology_group_topspace*:
trivial_group(*relative_homology_group p X* (*topspace X*))
by (*rule trivial_relative_homology_group_gen* [**where** *f*=*id*]) *auto*

lemma *trivial_relative_homology_group_empty*:
topspace X = {} ⟹ *trivial_group*(*relative_homology_group p X S*)
by (*metis Int_absorb2 empty_subsetI relative_homology_group_restrict trivial_relative_homology_group_topspace*)

lemma *trivial_homology_group_empty*:
topspace X = {} ⟹ *trivial_group*(*homology_group p X*)
by (*simp add*: *trivial_relative_homology_group_empty*)

lemma *homeomorphic_maps_relative_homology_group_isomorphisms*:
assumes *homeomorphic_maps X Y f g* **and** *im*: *f* ‘ *S* ⊆ *T g* ‘ *T* ⊆ *S*
shows *group_isomorphisms* (*relative_homology_group p X S*) (*relative_homology_group p Y T*)
(*hom_induced p X S Y T f*) (*hom_induced p Y T X S g*)
proof −
have *fg*: *continuous_map X Y f continuous_map Y X g*
(∀ *x*∈*topspace X*. *g* (*f x*) = *x*) (∀ *y*∈*topspace Y*. *f* (*g y*) = *y*)
using *assms* **by** (*simp_all add*: *homeomorphic_maps_def*)
have *group_isomorphisms*
(*relative_homology_group p X* (*topspace X* ∩ *S*))
(*relative_homology_group p Y* (*topspace Y* ∩ *T*))
(*hom_induced p X* (*topspace X* ∩ *S*) *Y* (*topspace Y* ∩ *T*) *f*)
(*hom_induced p Y* (*topspace Y* ∩ *T*) *X* (*topspace X* ∩ *S*) *g*)
proof (*rule homotopy_equivalence_relative_homology_group_isomorphisms*)
show *homotopic_with* (λ*h*. *h* ‘ (*topspace X* ∩ *S*) ⊆ *topspace X* ∩ *S*) *X X* (*g* ∘ *f*) *id*
using *fg im* **by** (*auto intro*: *homotopic_with_equal continuous_map_compose*)
next
show *homotopic_with* (λ*k*. *k* ‘ (*topspace Y* ∩ *T*) ⊆ *topspace Y* ∩ *T*) *Y Y* (*f* ∘ *g*) *id*
using *fg im* **by** (*auto intro*: *homotopic_with_equal continuous_map_compose*)
qed (*use im fg* **in** ‹*auto simp*: *continuous_map_def*›)
then show *?thesis*
by *simp*
qed

lemma *homeomorphic_map_relative_homology_iso*:
assumes *f*: *homeomorphic_map X Y f* **and** *S*: *S* ⊆ *topspace X f* ‘ *S* = *T*
shows (*hom_induced p X S Y T f*) ∈ *iso* (*relative_homology_group p X S*)

```
(relative_homology_group p Y T)
proof –
  obtain g where g: homeomorphic_maps X Y f g
    using homeomorphic_map_maps f by metis
 then have group_isomorphisms (relative_homology_group p X S) (relative_homology_group
p Y T)
                                  (hom_induced p X S Y T f) (hom_induced p Y T X S g)
   using S g by (auto simp: homeomorphic_maps_def intro!: homeomorphic_maps_relative_homology_group_isom
  then show ?thesis
    by (rule group_isomorphisms_imp_iso)
qed

lemma inj_on_hom_induced_section_map:
  assumes section_map X Y f
  shows inj_on (hom_induced p X {} Y {} f) (carrier (homology_group p X))
proof –
  obtain g where cont: continuous_map X Y f continuous_map Y X g
           and gf: ⋀x. x ∈ topspace X ⟹ g (f x) = x
    using assms by (auto simp: section_map_def retraction_maps_def)
  show ?thesis
  proof (rule inj_on_inverseI)
    fix x
    assume x: x ∈ carrier (homology_group p X)
    have continuous_map X X (λx. g (f x))
        by (metis (no_types, lifting) continuous_map_eq continuous_map_id gf
id_apply)
    with x show hom_induced p Y {} X {} g (hom_induced p X {} Y {} f x) =
x
      using hom_induced_compose_empty [OF cont, symmetric]
      apply (simp add: o_def fun_eq_iff)
      apply (rule hom_induced_id_gen)
      apply (auto simp: gf)
      done
  qed
qed

corollary mon_hom_induced_section_map:
  assumes section_map X Y f
 shows (hom_induced p X {} Y {} f) ∈ mon (homology_group p X) (homology_group
p Y)
   by (simp add: hom_induced_empty_hom inj_on_hom_induced_section_map
[OF assms] mon_def)

lemma surj_hom_induced_retraction_map:
  assumes retraction_map X Y f
  shows carrier (homology_group p Y) = (hom_induced p X {} Y {} f) ` carrier
(homology_group p X)
         (is ?lhs = ?rhs)
proof –
```

obtain *g* **where** *cont*: *continuous_map Y X g* *continuous_map X Y f*
and *fg*: $\bigwedge x.\ x \in topspace\ Y \Longrightarrow f\ (g\ x) = x$
using *assms* **by** (*auto simp*: *retraction_map_def retraction_maps_def*)
have $x =$ *hom_induced p X* {} *Y* {} *f* (*hom_induced p Y* {} *X* {} *g x*)
if *x*: $x \in$ *carrier* (*homology_group p Y*) **for** *x*
proof −
have *continuous_map Y Y* (λ*x*. *f* (*g x*))
by (*metis* (*no_types*, *lifting*) *continuous_map_eq continuous_map_id fg id_apply*)
with *x* **show** *?thesis*
using *hom_induced_compose_empty* [*OF cont*, *symmetric*]
apply (*simp add*: *o_def fun_eq_iff*)
apply (*rule hom_induced_id_gen* [*symmetric*])
apply (*auto simp*: *fg*)
done
qed
moreover
have (*hom_induced p Y* {} *X* {} *g x*) $\in$ *carrier* (*homology_group p X*)
if $x \in$ *carrier* (*homology_group p Y*) **for** *x*
by (*metis hom_induced*)
ultimately have *?lhs* $\subseteq$ *?rhs*
by *auto*
moreover have *?rhs* $\subseteq$ *?lhs*
using *hom_induced_hom* [*of p X* {} *Y* {} *f*]
by (*simp add*: *hom_def flip*: *image_subset_iff_funcset*)
ultimately show *?thesis*
by *auto*
qed

corollary *epi_hom_induced_retraction_map*:
assumes *retraction_map X Y f*
shows (*hom_induced p X* {} *Y* {} *f*) $\in$ *epi* (*homology_group p X*) (*homology_group p Y*)
using *assms epi_iff_subset hom_induced_empty_hom surj_hom_induced_retraction_map*
by *fastforce*

lemma *homeomorphic_map_homology_iso*:
assumes *homeomorphic_map X Y f*
shows (*hom_induced p X* {} *Y* {} *f*) $\in$ *iso* (*homology_group p X*) (*homology_group p Y*)
using *assms*
apply (*simp add*: *iso_def bij_betw_def flip*: *section_and_retraction_eq_homeomorphic_map*)
by (*metis inj_on_hom_induced_section_map surj_hom_induced_retraction_map hom_induced_hom*)

lemma *inj_on_hom_induced_inclusion*:

assumes *S* = {} ∨ *S retract_of_space X*
shows *inj_on* (*hom_induced p* (*subtopology X S*) {} *X* {} *id*) (*carrier* (*homology_group p* (*subtopology X S*)))
using *assms*
proof
assume *S* = {}
then have *trivial_group*(*homology_group p* (*subtopology X S*))
by (*auto simp*: *topspace_subtopology intro*: *trivial_homology_group_empty*)
then show *?thesis*
by (*auto simp*: *inj_on_def trivial_group_def*)
next
assume *S retract_of_space X*
then show *?thesis*
by (*simp add*: *retract_of_space_section_map inj_on_hom_induced_section_map*)
qed

lemma *trivial_homomorphism_hom_boundary_inclusion*:
assumes *S* = {} ∨ *S retract_of_space X*
shows *trivial_homomorphism*
(*relative_homology_group p X S*) (*homology_group* (*p*−*1*) (*subtopology X S*))
(*hom_boundary p X S*)
apply (*rule iffD1* [*OF exact_seq_mon_eq_triviality inj_on_hom_induced_inclusion* [*OF assms*]])
apply (*rule exact_seq.intros*)
apply (*rule homology_exactness_axiom_1* [*of p*])
using *homology_exactness_axiom_2* [*of p*]
by *auto*

lemma *epi_hom_induced_relativization*:
assumes *S* = {} ∨ *S retract_of_space X*
shows (*hom_induced p X* {} *X S id*) ‘ *carrier* (*homology_group p X*) = *carrier* (*relative_homology_group p X S*)
apply (*rule iffD2* [*OF exact_seq_epi_eq_triviality trivial_homomorphism_hom_boundary_inclusion*])
apply (*rule exact_seq.intros*)
apply (*rule homology_exactness_axiom_1* [*of p*])
using *homology_exactness_axiom_2* [*of p*] **apply** (*auto simp*: *assms*)
done

lemmas *short_exact_sequence_hom_induced_inclusion* = *homology_exactness_axiom_3*

lemma *group_isomorphisms_homology_group_prod_retract*:
assumes *S* = {} ∨ *S retract_of_space X*
obtains $\mathcal{H}$ $\mathcal{K}$ **where**
subgroup $\mathcal{H}$ (*homology_group p X*)
subgroup $\mathcal{K}$ (*homology_group p X*)
$(\lambda(x, y).\ x \otimes_{homology\_group\ p\ X} y)$
∈ *iso* (*DirProd* (*subgroup_generated* (*homology_group p X*) $\mathcal{H}$) (*subgroup_generated*

(*homology_group p X*) $\mathcal{K}$))
(*homology_group p X*)
(*hom_induced p* (*subtopology X S*) {} *X* {} *id*)
∈ *iso* (*homology_group p* (*subtopology X S*)) (*subgroup_generated* (*homology_group p X*) $\mathcal{H}$)
(*hom_induced p X* {} *X S id*)
∈ *iso* (*subgroup_generated* (*homology_group p X*) $\mathcal{K}$) (*relative_homology_group p X S*)
using *assms*
proof
assume $S = \{\}$
show *thesis*
proof (*rule splitting_lemma_left* [*OF homology_exactness_axiom_3* [*of p*]])
let *?f* = λ*x*. *one*(*homology_group p* (*subtopology X* {}))
show *?f* ∈ *hom* (*homology_group p X*) (*homology_group p* (*subtopology X* {}))
by (*simp add*: *trivial_hom*)
have *tg*: *trivial_group* (*homology_group p* (*subtopology X* {}))
by (*auto simp*: *topspace_subtopology trivial_homology_group_empty*)
then have [*simp*]: *carrier* (*homology_group p* (*subtopology X* {})) = {*one* (*homology_group p* (*subtopology X* {}))}
by (*auto simp*: *trivial_group_def*)
then show *?f* (*hom_induced p* (*subtopology X* {}) {} *X* {} *id x*) = *x*
if *x* ∈ *carrier* (*homology_group p* (*subtopology X* {})) **for** *x*
using *that* **by** *auto*
show *inj_on* (*hom_induced p* (*subtopology X* {}) {} *X* {} *id*)
(*carrier* (*homology_group p* (*subtopology X* {})))
by (*meson inj_on_hom_induced_inclusion*)
show *hom_induced p X* {} *X* {} *id* ‘ *carrier* (*homology_group p X*) = *carrier* (*homology_group p X*)
by (*metis epi_hom_induced_relativization*)
next
fix $\mathcal{H}$ $\mathcal{K}$
assume ∗: $\mathcal{H}$ ◁ *homology_group p X* $\mathcal{K}$ ◁ *homology_group p X*
$\mathcal{H} \cap \mathcal{K} \subseteq \{\mathbf{1}_{homology\_group\ p\ X}\}$
hom_induced p (*subtopology X* {}) {} *X* {} *id*
∈ *Group.iso* (*homology_group p* (*subtopology X* {})) (*subgroup_generated* (*homology_group p X*) $\mathcal{H}$)
hom_induced p X {} *X* {} *id*
∈ *Group.iso* (*subgroup_generated* (*homology_group p X*) $\mathcal{K}$) (*relative_homology_group p X* {})
$\mathcal{H}$ <#>$_{homology\_group\ p\ X}$ $\mathcal{K}$ = *carrier* (*homology_group p X*)
show *thesis*
proof (*rule that*)
show (λ(*x*, *y*). $x \otimes_{homology\_group\ p\ X} y$)
∈ *iso* (*subgroup_generated* (*homology_group p X*) $\mathcal{H}$ ×× *subgroup_generated* (*homology_group p X*) $\mathcal{K}$)
(*homology_group p X*)
using ∗ **by** (*simp add*: *group_disjoint_sum.iso_group_mul normal_def group_disjoint_sum_def*)

```
    qed (use ‹S = {}› * in ‹auto simp: normal_def›)
  qed
next
  assume S retract_of_space X
  then obtain r where S ⊆ topspace X and r: continuous_map X (subtopology
X S) r
                  and req: ∀ x ∈ S. r x = x
    by (auto simp: retract_of_space_def)
  show thesis
  proof (rule splitting_lemma_left [OF homology_exactness_axiom_3 [of p]])
    let ?f = hom_induced p X {} (subtopology X S) {} r
    show ?f ∈ hom (homology_group p X) (homology_group p (subtopology X S))
      by (simp add: hom_induced_empty_hom)
    show eqx: ?f (hom_induced p (subtopology X S) {} X {} id x) = x
      if x ∈ carrier (homology_group p (subtopology X S)) for x
    proof -
      have hom_induced p (subtopology X S) {} (subtopology X S) {} r x = x
      by (metis ‹S ⊆ topspace X› continuous_map_from_subtopology hom_induced_id_gen
inf.absorb_iff2 r req that topspace_subtopology)
      then show ?thesis
      by (simp add: r hom_induced_compose [unfolded o_def fun_eq_iff, rule_format,
symmetric])
    qed
    then show inj_on (hom_induced p (subtopology X S) {} X {} id)
              (carrier (homology_group p (subtopology X S)))
      unfolding inj_on_def by metis
    show hom_induced p X {} X S id ‘ carrier (homology_group p X) = carrier
(relative_homology_group p X S)
      by (simp add: ‹S retract_of_space X› epi_hom_induced_relativization)
  next
    fix ℋ 𝒦
    assume *: ℋ ⊲ homology_group p X 𝒦 ⊲ homology_group p X
      ℋ ∩ 𝒦 ⊆ {𝟏homology_group p X}
      ℋ <#>homology_group p X 𝒦 = carrier (homology_group p X)
      hom_induced p (subtopology X S) {} X {} id
        ∈ Group.iso (homology_group p (subtopology X S)) (subgroup_generated
(homology_group p X) ℋ)
      hom_induced p X {} X S id
   ∈ Group.iso (subgroup_generated (homology_group p X) 𝒦) (relative_homology_group
p X S)
    show thesis
    proof (rule that)
      show (λ(x, y). x ⊗homology_group p X y)
        ∈ iso (subgroup_generated (homology_group p X) ℋ ×× subgroup_generated
(homology_group p X) 𝒦)
               (homology_group p X)
        using *
      by (simp add: group_disjoint_sum.iso_group_mul normal_def group_disjoint_sum_def)
    qed (use * in ‹auto simp: normal_def›)
```

  qed
qed

lemma *isomorphic_group_homology_group_prod_retract*:
  assumes *S* = {} ∨ *S retract_of_space X*
  shows *homology_group p X* ≅ *homology_group p* (*subtopology X S*) ×× *relative_homology_group p X S*
        (**is** *?lhs* ≅ *?rhs*)
proof −
  obtain 𝓗 𝓚 **where**
    subgroup 𝓗 (*homology_group p X*)
    subgroup 𝓚 (*homology_group p X*)
   and *1*: (λ(*x*, *y*). $x \otimes_{homology\_group\ p\ X}$ *y*)
   ∈ *iso* (*DirProd* (*subgroup_generated* (*homology_group p X*) 𝓗) (*subgroup_generated* (*homology_group p X*) 𝓚))
          (*homology_group p X*)
    (*hom_induced p* (*subtopology X S*) {} *X* {} *id*)
   ∈ *iso* (*homology_group p* (*subtopology X S*)) (*subgroup_generated* (*homology_group p X*) 𝓗)
    (*hom_induced p X* {} *X S id*)
   ∈ *iso* (*subgroup_generated* (*homology_group p X*) 𝓚) (*relative_homology_group p X S*)
   using *group_isomorphisms_homology_group_prod_retract* [*OF assms*] **by** *blast*
  have *?lhs* ≅ *subgroup_generated* (*homology_group p X*) 𝓗 ×× *subgroup_generated* (*homology_group p X*) 𝓚
   by (*meson DirProd_group 1 abelian_homology_group comm_group_def group.abelian_subgroup_gene*
group.iso_sym is_isoI)
  also have … ≅ *?rhs*
   by (*meson 1*(*2*) *1*(*3*) *abelian_homology_group comm_group_def group.DirProd_iso_trans group.abelian_subgroup_generated group.iso_sym is_isoI*)
  finally show *?thesis* **.**
qed

lemma *homology_additivity_explicit*:
  assumes *openin X S openin X T disjnt S T* **and** *SUT*: *S* ∪ *T* = *topspace X*
  shows (λ(*a*,*b*).(*hom_induced p* (*subtopology X S*) {} *X* {} *id a*)
                $\otimes_{homology\_group\ p\ X}$
                (*hom_induced p* (*subtopology X T*) {} *X* {} *id b*))
        ∈ *iso* (*DirProd* (*homology_group p* (*subtopology X S*)) (*homology_group p* (*subtopology X T*)))
            (*homology_group p X*)
proof −
  have *closedin X S closedin X T*
    using *assms Un_commute disjnt_sym*
   by (*metis Diff_cancel Diff_triv Un_Diff disjnt_def openin_closedin_eq sup_bot.right_neutral*)+
  with ‹*openin X S*› ‹*openin X T*› **have** *SS*: *X closure_of S* ⊆ *X interior_of S*

and *TT*: *X closure_of T* ⊆ *X interior_of T*
  by (*simp_all add*: *closure_of_closedin interior_of_openin*)
 have [*simp*]: *S* ∪ *T* − *T* = *S S* ∪ *T* − *S* = *T*
  using ‹*disjnt S T*›
  by (*auto simp*: *Diff_triv Un_Diff disjnt_def*)
 let *?f* = *hom_induced p X* {} *X T id*
 let *?g* = *hom_induced p X* {} *X S id*
 let *?h* = *hom_induced p* (*subtopology X S*) {} *X T id*
 let *?i* = *hom_induced p* (*subtopology X S*) {} *X* {} *id*
 let *?j* = *hom_induced p* (*subtopology X T*) {} *X* {} *id*
 let *?k* = *hom_induced p* (*subtopology X T*) {} *X S id*
 let *?A* = *homology_group p* (*subtopology X S*)
 let *?B* = *homology_group p* (*subtopology X T*)
 let *?C* = *relative_homology_group p X T*
 let *?D* = *relative_homology_group p X S*
 let *?G* = *homology_group p X*
 have *h*: *?h* ∈ *iso ?A ?C* **and** *k*: *?k* ∈ *iso ?B ?D*
  using *homology_excision_axiom* [*OF TT*, *of S* ∪ *T p*]
  using *homology_excision_axiom* [*OF SS*, *of S* ∪ *T p*]
  by *auto* (*simp_all add*: *SUT*)
 have *1*: ⋀*x*. (*hom_induced p X* {} *X T id* ∘ *hom_induced p* (*subtopology X S*) {} *X* {} *id*) *x*
   = *hom_induced p* (*subtopology X S*) {} *X T id x*
  by (*simp flip*: *hom_induced_compose*)
 have *2*: ⋀*x*. (*hom_induced p X* {} *X S id* ∘ *hom_induced p* (*subtopology X T*) {} *X* {} *id*) *x*
   = *hom_induced p* (*subtopology X T*) {} *X S id x*
  by (*simp flip*: *hom_induced_compose*)
 show *?thesis*
  using *exact_sequence_sum_lemma*
   [*OF abelian_homology_group h k homology_exactness_axiom_3 homology_exactness_axiom_3*] *1 2*
  by *auto*
qed

0.2.6 Generalize exact homology sequence to triples

definition *hom_relboundary* :: [*int*,'*a topology*,'*a set*,'*a set*,'*a chain set*] ⇒ '*a chain set*
 where
 hom_relboundary p X S T =
  hom_induced (*p* − *1*) (*subtopology X S*) {} (*subtopology X S*) *T id* ∘
  hom_boundary p X S

lemma *group_homomorphism_hom_relboundary*:
 hom_relboundary p X S T
 ∈ *hom* (*relative_homology_group p X S*) (*relative_homology_group* (*p* − *1*) (*subtopology X S*) *T*)
 unfolding *hom_relboundary_def*

```
  proof (rule hom_compose)
   show hom_boundary p X S ∈ hom (relative_homology_group p X S) (homology_group(p
− 1) (subtopology X S))
      by (simp add: hom_boundary_hom)
  show hom_induced (p − 1) (subtopology X S) {} (subtopology X S) T id
     ∈ hom (homology_group(p − 1) (subtopology X S)) (relative_homology_group
(p − 1) (subtopology X S) T)
    by (simp add: hom_induced_hom)
qed

lemma hom_relboundary:
    hom_relboundary p X S T c ∈ carrier (relative_homology_group (p − 1)
(subtopology X S) T)
  by (simp add: hom_relboundary_def hom_induced_carrier)

lemma hom_relboundary_empty: hom_relboundary p X S {} = hom_boundary p
X S
  apply (simp add: hom_relboundary_def o_def)
  apply (subst hom_induced_id)
  apply (metis hom_boundary_carrier, auto)
  done

lemma naturality_hom_induced_relboundary:
  assumes continuous_map X Y f f ‘ S ⊆ U f ‘ T ⊆ V
  shows hom_relboundary p Y U V ∘
        hom_induced p X S Y (U) f =
        hom_induced (p − 1) (subtopology X S) T (subtopology Y U) V f ∘
        hom_relboundary p X S T
proof −
  have [simp]: continuous_map (subtopology X S) (subtopology Y U) f
   using assms continuous_map_from_subtopology continuous_map_in_subtopology
topspace_subtopology by fastforce
  have hom_induced (p − 1) (subtopology Y U) {} (subtopology Y U) V id ∘
       hom_induced (p − 1) (subtopology X S) {} (subtopology Y U) {} f
     = hom_induced (p − 1) (subtopology X S) T (subtopology Y U) V f ∘
       hom_induced (p − 1) (subtopology X S) {} (subtopology X S) T id
    using assms by (simp flip: hom_induced_compose)
  then show ?thesis
    apply (simp add: hom_relboundary_def comp_assoc naturality_hom_induced
assms)
    apply (simp flip: comp_assoc)
    done
qed

proposition homology_exactness_triple_1:
  assumes T ⊆ S
  shows exact_seq ([relative_homology_group(p − 1) (subtopology X S) T,
                    relative_homology_group p X S,
                    relative_homology_group p X T],
```

[*hom_relboundary p X S T*, *hom_induced p X T X S id*])
(**is** *exact_seq* ([*?G1*,*?G2*,*?G3*], [*?h1*,*?h2*]))
proof −
have *iTS*: *id* ‘ *T* ⊆ *S* **and** [*simp*]: *S* ∩ *T* = *T*
using *assms* **by** *auto*
have *?h2 B* ∈ *kernel ?G2 ?G1 ?h1* **for** *B*
proof −
have *hom_boundary p X T B* ∈ *carrier* (*relative_homology_group* (*p* − *1*) (*subtopology X T*) {})
by (*metis* (*no_types*) *hom_boundary*)
then have ∗: *hom_induced* (*p* − *1*) (*subtopology X S*) {} (*subtopology X S*) *T id*
(*hom_induced* (*p* − *1*) (*subtopology X T*) {} (*subtopology X S*) {} *id*
(*hom_boundary p X T B*))
= $\mathbf{1}_{?G1}$
using *homology_exactness_axiom_3* [*of p*−*1 subtopology X S T*]
by (*auto simp*: *subtopology_subtopology kernel_def*)
show *?thesis*
apply (*simp add*: *kernel_def hom_induced_carrier hom_relboundary_def flip*: ∗)
by (*metis comp_def naturality_hom_induced* [*OF continuous_map_id iTS*])
qed
moreover have *B* ∈ *?h2* ‘ *carrier ?G3* **if** *B* ∈ *kernel ?G2 ?G1 ?h1* **for** *B*
proof −
have *Bcarr*: *B* ∈ *carrier ?G2*
and *Beq*: *?h1 B* = $\mathbf{1}_{?G1}$
using *that* **by** (*auto simp*: *kernel_def*)
have ∃ *A'* ∈ *carrier* (*homology_group* (*p* − *1*) (*subtopology X T*)). *hom_induced* (*p* − *1*) (*subtopology X T*) {} (*subtopology X S*) {} *id A'* = *A*
if *A* ∈ *carrier* (*homology_group* (*p* − *1*) (*subtopology X S*))
hom_induced (*p* − *1*) (*subtopology X S*) {} (*subtopology X S*) *T id A* =
$\mathbf{1}_{?G1}$ **for** *A*
using *homology_exactness_axiom_3* [*of p*−*1 subtopology X S T*] *that*
by (*simp add*: *kernel_def subtopology_subtopology image_iff set_eq_iff*) *meson*
then obtain *C* **where** *Ccarr*: *C* ∈ *carrier* (*homology_group* (*p* − *1*) (*subtopology X T*))
and *Ceq*: *hom_induced* (*p* − *1*) (*subtopology X T*) {} (*subtopology X S*) {} *id C* = *hom_boundary p X S B*
using *Beq* **by** (*simp add*: *hom_relboundary_def*) (*metis hom_boundary_carrier*)
let *?hi_XT* = *hom_induced* (*p* − *1*) (*subtopology X T*) {} *X* {} *id*
have *?hi_XT*
= *hom_induced* (*p* − *1*) (*subtopology X S*) {} *X* {} *id*
∘ (*hom_induced* (*p* − *1*) (*subtopology X T*) {} (*subtopology X S*) {} *id*)
by (*metis assms comp_id continuous_map_id_subt hom_induced_compose_empty inf.absorb_iff2 subtopology_subtopology*)
then have *?hi_XT C*
= *hom_induced* (*p* − *1*) (*subtopology X S*) {} *X* {} *id* (*hom_boundary p X S B*)

by (*simp add*: *Ceq*)
also have *eq*: . . . = $\mathbf{1}_{homology\_group\ (p\ -\ 1)\ X}$
using *homology_exactness_axiom_2* [*of p X S*] *Bcarr* **by** (*auto simp*: *kernel_def*)
finally have *?hi_XT C* = $\mathbf{1}_{homology\_group\ (p\ -\ 1)\ X}$.
then obtain *D* **where** *Dcarr*: *D* ∈ *carrier ?G3* **and** *Deq*: *hom_boundary p X T D* = *C*
using *homology_exactness_axiom_2* [*of p X T*] *Ccarr*
by (*auto simp*: *kernel_def image_iff set_eq_iff*) *meson*
interpret *hb*: *group_hom ?G2 homology_group* (*p*−*1*) (*subtopology X S*)
hom_boundary p X S
using *hom_boundary_hom group_hom_axioms_def group_hom_def* **by** *fastforce*
let *?A* = $B \otimes_{?G2} inv_{?G2}$ *?h2 D*
have ∃ *A′* ∈ *carrier* (*homology_group p X*). *hom_induced p X* {} *X S id A′* = *A*
if *A* ∈ *carrier ?G2*
hom_boundary p X S A = *one* (*homology_group* (*p* − *1*) (*subtopology X S*)) **for** *A*
using *that homology_exactness_axiom_1* [*of p X S*]
by (*simp add*: *kernel_def subtopology_subtopology image_iff set_eq_iff*) *meson*
moreover
have *?A* ∈ *carrier ?G2*
by (*simp add*: *Bcarr abelian_relative_homology_group comm_groupE*(*1*) *hom_induced_carrier*)
moreover have *hom_boundary p X S* (*?h2 D*) = *hom_boundary p X S B*
by (*metis* (*mono_tags*, *lifting*) *Ceq Deq comp_eq_dest continuous_map_id iTS naturality_hom_induced*)
then have *hom_boundary p X S ?A* = *one* (*homology_group* (*p* − *1*) (*subtopology X S*))
by (*simp add*: *hom_induced_carrier Bcarr*)
ultimately obtain *W* **where** *Wcarr*: *W* ∈ *carrier* (*homology_group p X*)
and *Weq*: *hom_induced p X* {} *X S id W* = *?A*
by *blast*
let *?W* = $D \otimes_{?G3}$ *hom_induced p X* {} *X T id W*
show *?thesis*
proof
interpret *comm_group ?G2*
by (*rule abelian_relative_homology_group*)
have *B* = (*?h2* ∘ *hom_induced p X* {} *X T id*) $W \otimes_{?G2}$ *?h2 D*
apply (*simp add*: *hom_induced_compose* [*symmetric*] *assms*)
by (*metis Bcarr Weq hb.G.inv_solve_right hom_induced_carrier*)
then have $B \otimes_{?G2} inv_{?G2}$ *?h2 D*
= *?h2* (*hom_induced p X* {} *X T id W*)
by (*simp add*: *hb.G.m_assoc hom_induced_carrier*)
then show *B* = *?h2 ?W*
apply (*simp add*: *Dcarr hom_induced_carrier hom_mult* [*OF hom_induced_hom*])
by (*metis Bcarr hb.G.inv_solve_right hom_induced_carrier m_comm*)

```
      show ?W ∈ carrier ?G3
        by (simp add: Dcarr abelian_relative_homology_group comm_groupE(1)
hom_induced_carrier)
    qed
  qed
  ultimately show ?thesis
    by (auto simp: group_hom_def group_hom_axioms_def hom_induced_hom
group_homomorphism_hom_relboundary)
qed


proposition homology_exactness_triple_2:
  assumes T ⊆ S
  shows exact_seq ([relative_homology_group(p − 1) X T,
                    relative_homology_group(p − 1) (subtopology X S) T,
                    relative_homology_group p X S],
                  [hom_induced (p − 1) (subtopology X S) T X T id, hom_relboundary
p X S T])
    (is exact_seq ([?G1,?G2,?G3], [?h1,?h2]))
proof −
  let ?H2 = homology_group (p − 1) (subtopology X S)
  have iTS: id ‘ T ⊆ S and [simp]: S ∩ T = T
    using assms by auto
  have ?h2 C ∈ kernel ?G2 ?G1 ?h1 for C
  proof −
    have ?h1 (?h2 C)
       = (hom_induced (p − 1) X {} X T id ∘ hom_induced (p − 1) (subtopology
X S) {} X {} id ∘ hom_boundary p X S) C
      unfolding hom_relboundary_def
     by (metis (no_types, lifting) comp_apply continuous_map_id continuous_map_id_subt
empty_subsetI hom_induced_compose id_apply image_empty image_id order_refl)
    also have ... = 1_?G1
    proof −
      have ∗: hom_boundary p X S C ∈ carrier ?H2
        by (simp add: hom_boundary_carrier)
      moreover have hom_boundary p X S C ∈ hom_boundary p X S ‘ carrier
?G3
        using homology_exactness_axiom_2 [of p X S] ∗
        apply (simp add: kernel_def set_eq_iff)
      by (metis group_relative_homology_group hom_boundary_default hom_one
image_eqI)
      ultimately
      have 1: hom_induced (p − 1) (subtopology X S) {} X {} id (hom_boundary
p X S C)
            = 1_homology_group (p − 1) X
       using homology_exactness_axiom_2 [of p X S] by (simp add: kernel_def)
blast
      show ?thesis
        by (simp add: 1 hom_one [OF hom_induced_hom])
```

qed
finally have *?h1* (*?h2 C*) = $\mathbf{1}_{?G1}$.
then show *?thesis*
by (*simp add*: *kernel_def hom_relboundary_def hom_induced_carrier*)
qed
moreover have *x* ∈ *?h2* ‘ *carrier ?G3* **if** *x* ∈ *kernel ?G2 ?G1 ?h1* **for** *x*
proof −
let *?homX* = *hom_induced* (*p* − *1*) (*subtopology X S*) {} *X* {} *id*
let *?homXS* = *hom_induced* (*p* − *1*) (*subtopology X S*) {} (*subtopology X S*)
T id
have *x* ∈ *carrier* (*relative_homology_group* (*p* − *1*) (*subtopology X S*) *T*)
using *that* **by** (*simp add*: *kernel_def*)
moreover
have *hom_boundary* (*p*−*1*) *X T* ∘ *hom_induced* (*p*−*1*) (*subtopology X S*) *T*
X T id = *hom_boundary* (*p*−*1*) (*subtopology X S*) *T*
by (*metis Int_lower2* ‹*S* ∩ *T* = *T*› *continuous_map_id_subt hom_relboundary_def*
hom_relboundary_empty id_apply image_id naturality_hom_induced subtopology_subtopology)
then have *hom_boundary* (*p* − *1*) (*subtopology X S*) *T x* = $\mathbf{1}$*homology_group* (*p* − *2*) (*subtopology* (*subt*
using *naturality_hom_induced* [*of subtopology X S X id T T p*−*1*] *that*
hom_one [*OF hom_boundary_hom group_relative_homology_group group_relative_homology_gro*
of p−*1 X T*]
apply (*simp add*: *kernel_def subtopology_subtopology*)
by (*metis comp_apply*)
ultimately
obtain *y* **where** *ycarr*: *y* ∈ *carrier ?H2*
and *yeq*: *?homXS y* = *x*
using *homology_exactness_axiom_1* [*of p*−*1 subtopology X S T*]
by (*simp add*: *kernel_def image_def set_eq_iff*) *meson*
have *?homX y* ∈ *carrier* (*homology_group* (*p* − *1*) *X*)
by (*simp add*: *hom_induced_carrier*)
moreover
have (*hom_induced* (*p* − *1*) *X* {} *X T id* ∘ *?homX*) *y* = $\mathbf{1}$*relative_homology_group* (*p* − *1*) *X T*
apply (*simp flip*: *hom_induced_compose*)
using *hom_induced_compose* [*of subtopology X S subtopology X S id* {} *T X*
id T p−*1*]
apply *simp*
by (*metis* (*mono_tags*, *lifting*) *kernel_def mem_Collect_eq that yeq*)
then have *hom_induced* (*p* − *1*) *X* {} *X T id* (*?homX y*) = $\mathbf{1}$*relative_homology_group* (*p* − *1*) *X T*
by *simp*
ultimately obtain *z* **where** *zcarr*: *z* ∈ *carrier* (*homology_group* (*p* − *1*)
(*subtopology X T*))
and *zeq*: *hom_induced* (*p* − *1*) (*subtopology X T*) {} *X* {} *id z* =
?homX y
using *homology_exactness_axiom_3* [*of p*−*1 X T*]
by (*auto simp*: *kernel_def dest*!: *equalityD1* [*of Collect _*])
have ∗: ⋀*t*. ⟦*t* ∈ *carrier ?H2*;
hom_induced (*p* − *1*) (*subtopology X S*) {} *X* {} *id t* =
$\mathbf{1}$*homology_group* (*p* − *1*) *X*⟧
⟹ *t* ∈ *hom_boundary p X S* ‘ *carrier ?G3*

using *homology_exactness_axiom_2* [*of p X S*]
by (*auto simp*: *kernel_def dest*!: *equalityD1* [*of Collect* _])
interpret *comm_group ?H2*
by (*rule abelian_relative_homology_group*)
interpret *gh*: *group_hom ?H2 homology_group* (*p* − *1*) *X hom_induced* (*p*−*1*) (*subtopology X S*) {} *X* {} *id*
by (*meson group_hom_axioms_def group_hom_def group_relative_homology_group hom_induced*)
let *?yz* = *y* $\otimes_{?H2}$ $inv_{?H2}$ *hom_induced* (*p* − *1*) (*subtopology X T*) {} (*subtopology X S*) {} *id z*
have *yzcarr*: *?yz* ∈ *carrier ?H2*
by (*simp add*: *hom_induced_carrier ycarr*)
have *yzeq*: *hom_induced* (*p* − *1*) (*subtopology X S*) {} *X* {} *id ?yz* = $\mathbf{1}_{homology\_group\ (p\ -\ 1)\ X}$
apply (*simp add*: *hom_induced_carrier ycarr gh.inv_solve_right*′)
by (*metis assms continuous_map_id_subt hom_induced_compose_empty inf.absorb_iff2 o_apply o_id subtopology_subtopology zeq*)
obtain *w* **where** *wcarr*: *w* ∈ *carrier ?G3* **and** *weq*: *hom_boundary p X S w* = *?yz*
using ∗ [*OF yzcarr yzeq*] **by** *blast*
interpret *gh2*: *group_hom ?H2 ?G2 ?homXS*
by (*simp add*: *group_hom_axioms_def group_hom_def hom_induced_hom*)
have *?homXS* (*hom_induced* (*p* − *1*) (*subtopology X T*) {} (*subtopology X S*) {} *id z*)
= $\mathbf{1}_{relative\_homology\_group\ (p\ -\ 1)\ (subtopology\ X\ S)\ T}$
using *homology_exactness_axiom_3* [*of p*−*1 subtopology X S T*] *zcarr*
by (*auto simp*: *kernel_def subtopology_subtopology*)
then show *?thesis*
apply (*rule_tac x*=*w* **in** *image_eqI*)
apply (*simp_all add*: *hom_relboundary_def weq wcarr*)
by (*metis gh2.hom_inv gh2.hom_mult gh2.inv_one gh2.r_one group.inv_closed group_l_invI hom_induced_carrier l_inv_ex ycarr yeq*)
qed
ultimately show *?thesis*
by (*auto simp*: *group_hom_axioms_def group_hom_def group_homomorphism_hom_relboundary hom_induced_hom*)
qed

proposition *homology_exactness_triple_3*:
assumes *T* ⊆ *S*
shows *exact_seq* ([*relative_homology_group p X S*,
relative_homology_group p X T,
relative_homology_group p (*subtopology X S*) *T*],
[*hom_induced p X T X S id*, *hom_induced p* (*subtopology X S*) *T X T id*])
(**is** *exact_seq* ([*?G1*,*?G2*,*?G3*], [*?h1*,*?h2*]))
proof −
have *iTS*: *id* ‘ *T* ⊆ *S* **and** [*simp*]: *S* ∩ *T* = *T*
using *assms* **by** *auto*
have *1*: *?h2 x* ∈ *kernel ?G2 ?G1 ?h1* **for** *x*

proof −
have *?h1* (*?h2 x*)
= (*hom_induced p* (*subtopology X S*) *S X S id* ∘
hom_induced p (*subtopology X S*) *T* (*subtopology X S*) *S id*) *x*
by (*metis comp_eq_dest_lhs continuous_map_id continuous_map_id_subt hom_induced_compose iTS id_apply image_subsetI*)
also have ... $= \mathbf{1}_{relative\_homology\_group\ p\ X\ S}$
proof −
have *trivial_group* (*relative_homology_group p* (*subtopology X S*) *S*)
using *trivial_relative_homology_group_topspace* [*of p subtopology X S*]
by (*metis inf_right_idem relative_homology_group_restrict topspace_subtopology*)
then have *1*: *hom_induced p* (*subtopology X S*) *T* (*subtopology X S*) *S id x*
$= \mathbf{1}_{relative\_homology\_group\ p\ (subtopology\ X\ S)\ S}$
using *hom_induced_carrier* **by** (*fastforce simp add*: *trivial_group_def*)
show *?thesis*
by (*simp add*: *1 hom_one* [*OF hom_induced_hom*])
qed
finally have *?h1* (*?h2 x*) $= \mathbf{1}_{relative\_homology\_group\ p\ X\ S}$.
then show *?thesis*
by (*simp add*: *hom_induced_carrier kernel_def*)
qed
moreover have *x* ∈ *?h2* ‘ *carrier ?G3* **if** *x*: *x* ∈ *kernel ?G2 ?G1 ?h1* **for** *x*
proof −
have *xcarr*: *x* ∈ *carrier ?G2*
using *that* **by** (*auto simp*: *kernel_def*)
interpret *G2*: *comm_group ?G2*
by (*rule abelian_relative_homology_group*)
let *?b* = *hom_boundary p X T x*
have *bcarr*: *?b* ∈ *carrier*(*homology_group*(*p* − *1*) (*subtopology X T*))
by (*simp add*: *hom_boundary_carrier*)
have *hom_boundary p X S* (*hom_induced p X T X S id x*)
= *hom_induced* (*p* − *1*) (*subtopology X T*) {} (*subtopology X S*) {} *id*
(*hom_boundary p X T x*)
using *naturality_hom_induced* [*of X X id T S p*] **by** (*simp add*: *assms o_def*) *meson*
with *bcarr* **have** *hom_boundary p X T x* ∈ *hom_boundary p* (*subtopology X S*) *T* ‘ *carrier ?G3*
using *homology_exactness_axiom_2* [*of p subtopology X S T*] *x*
apply (*simp add*: *kernel_def set_eq_iff subtopology_subtopology*)
by (*metis group_relative_homology_group hom_boundary_hom hom_one set_eq_iff*)
then obtain *u* **where** *ucarr*: *u* ∈ *carrier ?G3*
and *ueq*: *hom_boundary p X T x* = *hom_boundary p* (*subtopology X S*) *T u*
by (*auto simp*: *kernel_def set_eq_iff subtopology_subtopology hom_boundary_carrier*)
define *y* **where** $y = x \otimes_{?G2} inv_{?G2}\ ?h2\ u$
have *ycarr*: *y* ∈ *carrier ?G2*
using *x* **by** (*simp add*: *y_def kernel_def hom_induced_carrier*)
interpret *hb*: *group_hom ?G2 homology_group* (*p*−*1*) (*subtopology X T*)

hom_boundary p X T
by (*simp add*: *group_hom_axioms_def group_hom_def hom_boundary_hom*)
have *yyy*: *hom_boundary p X T y* = $\mathbf{1}_{homology\_group\ (p\ -\ 1)\ (subtopology\ X\ T)}$
apply (*simp add*: *y_def bcarr xcarr hom_induced_carrier hom_boundary_carrier hb.inv_solve_right'*)
using *naturality_hom_induced* [*of concl*: *p X T subtopology X S T id*]
apply (*simp add*: *o_def fun_eq_iff subtopology_subtopology*)
by (*metis hom_boundary_carrier hom_induced_id ueq*)
then have *y* ∈ *hom_induced p X* {} *X T id* ' *carrier* (*homology_group p X*)
using *homology_exactness_axiom_1* [*of p X T*] *x ycarr* **by** (*auto simp*: *kernel_def*)
then obtain *z* **where** *zcarr*: *z* ∈ *carrier* (*homology_group p X*)
and *zeq*: *hom_induced p X* {} *X T id z* = *y*
by *auto*
interpret *gh1*: *group_hom ?G2 ?G1 ?h1*
by (*meson group_hom_axioms_def group_hom_def group_relative_homology_group hom_induced*)

have *hom_induced p X* {} *X S id z* = (*hom_induced p X T X S id* ∘ *hom_induced p X* {} *X T id*) *z*
by (*simp add*: *assms flip*: *hom_induced_compose*)
also have ... = $\mathbf{1}_{relative\_homology\_group\ p\ X\ S}$
using *x 1* **by** (*simp add*: *kernel_def zeq y_def*)
finally have *hom_induced p X* {} *X S id z* = $\mathbf{1}_{relative\_homology\_group\ p\ X\ S}$.
then have *z* ∈ *hom_induced p* (*subtopology X S*) {} *X* {} *id* '
carrier (*homology_group p* (*subtopology X S*))
using *homology_exactness_axiom_3* [*of p X S*] *zcarr* **by** (*auto simp*: *kernel_def*)
then obtain *w* **where** *wcarr*: *w* ∈ *carrier* (*homology_group p* (*subtopology X S*))
and *weq*: *hom_induced p* (*subtopology X S*) {} *X* {} *id w* = *z*
by *blast*
let *?u* = *hom_induced p* (*subtopology X S*) {} (*subtopology X S*) *T id w* $\otimes_{?G3}$ *u*
show *?thesis*
proof
have ∗: *x* = *z* $\otimes_{?G2}$ *u*
if *z* = *x* $\otimes_{?G2}$ $inv_{?G2}$ *u z* ∈ *carrier ?G2 u* ∈ *carrier ?G2* **for** *z u*
using *that* **by** (*simp add*: *group.inv_solve_right xcarr*)
have *eq*: *?h2* ∘ *hom_induced p* (*subtopology X S*) {} (*subtopology X S*) *T id*
= *hom_induced p X* {} *X T id* ∘ *hom_induced p* (*subtopology X S*) {} *X* {} *id*
by (*simp flip*: *hom_induced_compose*)
show *x* = *hom_induced p* (*subtopology X S*) *T X T id ?u*
apply (*simp add*: *hom_mult* [*OF hom_induced_hom*] *hom_induced_carrier ucarr*)
apply (*rule* ∗)
using *eq* **apply** (*simp_all add*: *fun_eq_iff hom_induced_carrier flip*: *y_def zeq weq*)

```
      done
    show ?u ∈ carrier (relative_homology_group p (subtopology X S) T)
    by (simp add: abelian_relative_homology_group comm_groupE(1) hom_induced_carrier
ucarr)
   qed
  qed
  ultimately show ?thesis
   by (auto simp: group_hom_axioms_def group_hom_def hom_induced_hom)
qed

end
```

0.3 Homology, III: Brouwer Degree

```
theory Brouwer_Degree
  imports Homology_Groups HOL−Algebra.Multiplicative_Group

begin
```

0.3.1 Reduced Homology

```
definition reduced_homology_group :: int ⇒ 'a topology ⇒ 'a chain set monoid
  where reduced_homology_group p X ≡
         subgroup_generated (homology_group p X)
           (kernel (homology_group p X) (homology_group p (discrete_topology
{()}))
                (hom_induced p X {} (discrete_topology {()}) {} (λx. ())))

lemma one_reduced_homology_group: 1_(reduced_homology_group p X) = 1_(homology_group p X)
   by (simp add: reduced_homology_group_def)

lemma group_reduced_homology_group [simp]: group (reduced_homology_group
p X)
   by (simp add: reduced_homology_group_def group.group_subgroup_generated)

lemma carrier_reduced_homology_group:
  carrier (reduced_homology_group p X) =
   kernel (homology_group p X) (homology_group p (discrete_topology {()}))
        (hom_induced p X {} (discrete_topology {()}) {} (λx. ()))
   (is _ = kernel ?G ?H ?h)
proof −
  interpret subgroup kernel ?G ?H ?h ?G
 by (simp add: hom_induced_empty_hom group_hom_axioms_def group_hom_def
group_hom.subgroup_kernel)
  show ?thesis
   unfolding reduced_homology_group_def
   using carrier_subgroup_generated_subgroup by blast
qed
```

lemma *carrier_reduced_homology_group_subset*:
carrier (*reduced_homology_group p X*) ⊆ *carrier* (*homology_group p X*)
by (*simp add*: *group.carrier_subgroup_generated_subset reduced_homology_group_def*)

lemma *un_reduced_homology_group*:
assumes $p \neq 0$
shows *reduced_homology_group p X* = *homology_group p X*
proof –
have (*kernel* (*homology_group p X*) (*homology_group p* (*discrete_topology* {()}))
(*hom_induced p X* {} (*discrete_topology* {()}) {} (λ*x*. ())))
= *carrier* (*homology_group p X*)
proof (*rule group_hom.kernel_to_trivial_group*)
show *group_hom* (*homology_group p X*) (*homology_group p* (*discrete_topology*
{()}))
(*hom_induced p X* {} (*discrete_topology* {()}) {} (λ*x*. ()))
by (*auto simp*: *hom_induced_empty_hom group_hom_def group_hom_axioms_def*)
show *trivial_group* (*homology_group p* (*discrete_topology* {()}))
by (*simp add*: *homology_dimension_axiom* [*OF* _ *assms*])
qed
then show *?thesis*
by (*simp add*: *reduced_homology_group_def group.subgroup_generated_group_carrier*)
qed

lemma *trivial_reduced_homology_group*:
$p < 0$ ⟹ *trivial_group*(*reduced_homology_group p X*)
by (*simp add*: *trivial_homology_group un_reduced_homology_group*)

lemma *hom_induced_reduced_hom*:
(*hom_induced p X* {} *Y* {} *f*) ∈ *hom* (*reduced_homology_group p X*) (*reduced_homology_group
p Y*)
proof (*cases continuous_map X Y f*)
case *True*
have *eq*: *continuous_map X Y f*
⟹ *hom_induced p X* {} (*discrete_topology* {()}) {} (λ*x*. ())
= (*hom_induced p Y* {} (*discrete_topology* {()}) {} (λ*x*. ()) ∘ *hom_induced
p X* {} *Y* {} *f*)
by (*simp flip*: *hom_induced_compose_empty*)
interpret *subgroup kernel* (*homology_group p X*)
(*homology_group p* (*discrete_topology* {()}))
(*hom_induced p X* {} (*discrete_topology* {()}) {} (λ*x*. ()))
homology_group p X
by (*meson group_hom.subgroup_kernel group_hom_axioms_def group_hom_def
group_relative_homology_group hom_induced*)
have *sb*: *hom_induced p X* {} *Y* {} *f* ‘ *carrier* (*homology_group p X*) ⊆ *carrier*
(*homology_group p Y*)
using *hom_induced_carrier* **by** *blast*
show *?thesis*
using *True*
unfolding *reduced_homology_group_def*

apply (*simp add*: *hom_into_subgroup_eq group_hom.subgroup_kernel hom_induced_empty_hom group.hom_from_subgroup_generated group_hom_def group_hom_axioms_def*)
unfolding *kernel_def* **using** *eq sb* **by** *auto*
next
case *False*
then have *hom_induced p X {} Y {} f* = (λ*c. one*(*reduced_homology_group p Y*))
by (*force simp*: *hom_induced_default reduced_homology_group_def*)
then show *?thesis*
by (*simp add*: *trivial_hom*)
qed

lemma *hom_induced_reduced*:
c ∈ *carrier*(*reduced_homology_group p X*)
⟹ *hom_induced p X {} Y {} f c* ∈ *carrier*(*reduced_homology_group p Y*)
by (*meson hom_in_carrier hom_induced_reduced_hom*)

lemma *hom_boundary_reduced_hom*:
hom_boundary p X S
∈ *hom* (*relative_homology_group p X S*) (*reduced_homology_group* (*p−1*) (*subtopology X S*))
proof −
have ∗: *continuous_map X* (*discrete_topology* {()}) (λ*x.* ()) (λ*x.* ()) ' *S* ⊆ {()}
by *auto*
interpret *group_hom relative_homology_group p* (*discrete_topology* {()}) {()}
homology_group (*p−1*) (*discrete_topology* {()})
hom_boundary p (*discrete_topology* {()}) {()}
apply (*clarsimp simp*: *group_hom_def group_hom_axioms_def*)
by (*metis UNIV_unit hom_boundary_hom subtopology_UNIV*)
have *hom_boundary p X S* '
carrier (*relative_homology_group p X S*)
⊆ *kernel* (*homology_group* (*p* − *1*) (*subtopology X S*))
(*homology_group* (*p* − *1*) (*discrete_topology* {()}))
(*hom_induced* (*p* − *1*) (*subtopology X S*) {}
(*discrete_topology* {()}) {} (λ*x.* ()))
proof (*clarsimp simp add*: *kernel_def hom_boundary_carrier*)
fix *c*
assume *c*: *c* ∈ *carrier* (*relative_homology_group p X S*)
have *triv*: *trivial_group* (*relative_homology_group p* (*discrete_topology* {()}) {()})
by (*metis topspace_discrete_topology trivial_relative_homology_group_topspace*)
have *hom_boundary p* (*discrete_topology* {()}) {()}
(*hom_induced p X S* (*discrete_topology* {()}) {()} (λ*x.* ()) *c*)
= $\mathbf{1}_{homology\_group\ (p\ -\ 1)\ (discrete\_topology\ \{()\})}$
by (*metis hom_induced_carrier local.hom_one singletonD triv trivial_group_def*)
then show *hom_induced* (*p* − *1*) (*subtopology X S*) {} (*discrete_topology* {()}) {} (λ*x.* ()) (*hom_boundary p X S c*) =
$\mathbf{1}_{homology\_group\ (p\ -\ 1)\ (discrete\_topology\ \{()\})}$

```
    using naturality_hom_induced [OF *, of p, symmetric] by (simp add: o_def
fun_eq_iff)
  qed
  then show ?thesis
   by (simp add: reduced_homology_group_def hom_boundary_hom hom_into_subgroup)
qed


lemma homotopy_equivalence_reduced_homology_group_isomorphisms:
  assumes contf: continuous_map X Y f and contg: continuous_map Y X g
    and gf: homotopic_with (λh. True) X X (g ∘ f) id
    and fg: homotopic_with (λk. True) Y Y (f ∘ g) id
 shows group_isomorphisms (reduced_homology_group p X) (reduced_homology_group
p Y)
                                (hom_induced p X {} Y {} f) (hom_induced p Y {} X
{} g)
proof (simp add: hom_induced_reduced_hom group_isomorphisms_def, intro conjI
ballI)
  fix a
  assume a ∈ carrier (reduced_homology_group p X)
  then have (hom_induced p Y {} X {} g ∘ hom_induced p X {} Y {} f) a = a
    apply (simp add: contf contg flip: hom_induced_compose)
      using carrier_reduced_homology_group_subset gf hom_induced_id homol-
ogy_homotopy_empty by fastforce
  then show hom_induced p Y {} X {} g (hom_induced p X {} Y {} f a) = a
    by simp
next
  fix b
  assume b ∈ carrier (reduced_homology_group p Y)
  then have (hom_induced p X {} Y {} f ∘ hom_induced p Y {} X {} g) b = b
    apply (simp add: contf contg flip: hom_induced_compose)
      using carrier_reduced_homology_group_subset fg hom_induced_id homol-
ogy_homotopy_empty by fastforce
  then show hom_induced p X {} Y {} f (hom_induced p Y {} X {} g b) = b
    by (simp add: carrier_reduced_homology_group)
qed

lemma homotopy_equivalence_reduced_homology_group_isomorphism:
  assumes continuous_map X Y f continuous_map Y X g
      and homotopic_with (λh. True) X X (g ∘ f) id homotopic_with (λk. True)
Y Y (f ∘ g) id
  shows (hom_induced p X {} Y {} f)
          ∈ iso (reduced_homology_group p X) (reduced_homology_group p Y)
proof (rule group_isomorphisms_imp_iso)
 show group_isomorphisms (reduced_homology_group p X) (reduced_homology_group
p Y)
         (hom_induced p X {} Y {} f) (hom_induced p Y {} X {} g)
   by (simp add: assms homotopy_equivalence_reduced_homology_group_isomorphisms)
qed
```

lemma *homotopy_equivalent_space_imp_isomorphic_reduced_homology_groups*:
X homotopy_equivalent_space Y
⟹ *reduced_homology_group p X* ≅ *reduced_homology_group p Y*
unfolding *homotopy_equivalent_space_def*
using *homotopy_equivalence_reduced_homology_group_isomorphism is_isoI* **by** *blast*

lemma *homeomorphic_space_imp_isomorphic_reduced_homology_groups*:
X homeomorphic_space Y ⟹ *reduced_homology_group p X* ≅ *reduced_homology_group p Y*
by (*simp add*: *homeomorphic_imp_homotopy_equivalent_space homotopy_equivalent_space_imp_isor*

lemma *trivial_reduced_homology_group_empty*:
topspace X = {} ⟹ *trivial_group*(*reduced_homology_group p X*)
by (*metis carrier_reduced_homology_group_subset group.trivial_group_alt group_reduced_homology_ trivial_group_def trivial_homology_group_empty*)

lemma *homology_dimension_reduced*:
assumes *topspace X* = {*a*}
shows *trivial_group* (*reduced_homology_group p X*)
proof −
have *iso*: (*hom_induced p X* {} (*discrete_topology* {()}) {} (λ*x*. ()))
∈ *iso* (*homology_group p X*) (*homology_group p* (*discrete_topology* {()}))
apply (*rule homeomorphic_map_homology_iso*)
apply (*force simp*: *homeomorphic_map_maps homeomorphic_maps_def assms*)
done
show *?thesis*
unfolding *reduced_homology_group_def*
by (*rule group.trivial_group_subgroup_generated*) (*use iso* **in** ‹*auto simp*: *iso_kernel_image*›)
qed

lemma *trivial_reduced_homology_group_contractible_space*:
contractible_space X ⟹ *trivial_group* (*reduced_homology_group p X*)
apply (*simp add*: *contractible_eq_homotopy_equivalent_singleton_subtopology*)
apply (*auto simp*: *trivial_reduced_homology_group_empty*)
using *isomorphic_group_triviality*
by (*metis* (*full_types*) *group_reduced_homology_group homology_dimension_reduced homotopy_equivalent_space_imp_isomorphic_reduced_homology_groups path_connectedin_def path_connectedin_singleton topspace_subtopology_subset*)

lemma *image_reduced_homology_group*:
assumes *topspace X* ∩ *S* ≠ {}
shows *hom_induced p X* {} *X S id* ‘ *carrier* (*reduced_homology_group p X*)
= *hom_induced p X* {} *X S id* ‘ *carrier* (*homology_group p X*)
(**is** *?h* ‘ *carrier ?G* = *?h* ‘ *carrier ?H*)

```
proof –
  obtain a where a: a ∈ topspace X and a ∈ S
    using assms by blast
  have [simp]: A ∩ {x ∈ A. P x} = {x ∈ A. P x} for A P
    by blast
  interpret comm_group homology_group p X
    by (rule abelian_relative_homology_group)
  have ∗: ∃ x'. ?h y = ?h x' ∧
            x' ∈ carrier ?H ∧
            hom_induced p X {} (discrete_topology {()}) {} (λx. ()) x'
          = 1homology_group p (discrete_topology {()})
    if y ∈ carrier ?H for y
  proof –
    let ?f = hom_induced p (discrete_topology {()}) {} X {} (λx. a)
    let ?g = hom_induced p X {} (discrete_topology {()}) {} (λx. ())
    have bcarr: ?f (?g y) ∈ carrier ?H
      by (simp add: hom_induced_carrier)
    interpret gh1:
         group_hom relative_homology_group p X S relative_homology_group p
(discrete_topology {()}) {()}
                hom_induced p X S (discrete_topology {()}) {()} (λx. ())
        by (meson group_hom_axioms_def group_hom_def hom_induced_hom
group_relative_homology_group)
    interpret gh2:
        group_hom relative_homology_group p (discrete_topology {()}) {()} rela-
tive_homology_group p X S
                hom_induced p (discrete_topology {()}) {()} X S (λx. a)
        by (meson group_hom_axioms_def group_hom_def hom_induced_hom
group_relative_homology_group)
    interpret gh3:
       group_hom homology_group p X relative_homology_group p X S ?h
        by (meson group_hom_axioms_def group_hom_def hom_induced_hom
group_relative_homology_group)
    interpret gh4:
       group_hom homology_group p X homology_group p (discrete_topology {()})
                ?g
        by (meson group_hom_axioms_def group_hom_def hom_induced_hom
group_relative_homology_group)
    interpret gh5:
       group_hom homology_group p (discrete_topology {()}) homology_group p X
                ?f
        by (meson group_hom_axioms_def group_hom_def hom_induced_hom
group_relative_homology_group)
    interpret gh6:
      group_hom homology_group p (discrete_topology {()}) relative_homology_group
p (discrete_topology {()}) {()}
                 hom_induced p (discrete_topology {()}) {} (discrete_topology {()})
{()} id
        by (meson group_hom_axioms_def group_hom_def hom_induced_hom
```

```
group_relative_homology_group)
    show ?thesis
    proof (intro exI conjI)
      have (?h ∘ ?f ∘ ?g) y
          = (hom_induced p (discrete_topology {()}) {()} X S (λx. a) ∘
            hom_induced p (discrete_topology {()}) {} (discrete_topology {()}) {()}
id ∘ ?g) y
        by (simp add: a ‹a ∈ S› flip: hom_induced_compose)
      also have … = 𝟭relative_homology_group p X S
          using trivial_relative_homology_group_topspace [of p discrete_topology
{()}]
        apply simp
          by (metis (full_types) empty_iff gh1.H.one_closed gh1.H.trivial_group
gh2.hom_one hom_induced_carrier insert_iff)
      finally have ?h (?f (?g y)) = 𝟭relative_homology_group p X S
        by simp
      then show ?h y = ?h (y ⊗?H inv?H ?f (?g y))
        by (simp add: that hom_induced_carrier)
      show (y ⊗?H inv?H ?f (?g y)) ∈ carrier (homology_group p X)
        by (simp add: hom_induced_carrier that)
      have ∗: (?g ∘ hom_induced p X {} X {} (λx. a)) y = hom_induced p X {}
(discrete_topology {()}) {} (λa. ()) y
        by (simp add: a ‹a ∈ S› flip: hom_induced_compose)
      have ?g (y ⊗?H inv?H (?f ∘ ?g) y)
          = 𝟭homology_group p (discrete_topology {()})
      by (simp add: a ‹a ∈ S› that hom_induced_carrier flip: hom_induced_compose
∗ [unfolded o_def])
      then show ?g (y ⊗?H inv?H ?f (?g y))
          = 𝟭homology_group p (discrete_topology {()})
        by simp
    qed
  qed
  show ?thesis
     apply (auto simp: reduced_homology_group_def carrier_subgroup_generated
kernel_def image_iff)
    apply (metis (no_types, lifting) generate_in_carrier mem_Collect_eq subsetI)
    apply (force simp: dest: ∗ intro: generate.incl)
    done
qed


lemma homology_exactness_reduced_1:
  assumes topspace X ∩ S ≠ {}
  shows  exact_seq([reduced_homology_group(p − 1) (subtopology X S),
                    relative_homology_group p X S,
                    reduced_homology_group p X],
                   [hom_boundary p X S, hom_induced p X {} X S id])
    (is exact_seq ([?G1,?G2,?G3], [?h1,?h2]))
proof −
```

```
  have *: ?h2 ` carrier (homology_group p X)
        = kernel ?G2 (homology_group (p - 1) (subtopology X S)) ?h1
    using homology_exactness_axiom_1 [of p X S] by simp
  have gh: group_hom ?G3 ?G2 ?h2
   by (simp add: reduced_homology_group_def group_hom_def group_hom_axioms_def
     group.group_subgroup_generated group.hom_from_subgroup_generated hom_induced_hom)
  show ?thesis
   apply (simp add: hom_boundary_reduced_hom gh * image_reduced_homology_group
[OF assms])
    apply (simp add: kernel_def one_reduced_homology_group)
    done
qed


lemma homology_exactness_reduced_2:
    exact_seq([reduced_homology_group(p - 1) X,
                  reduced_homology_group(p - 1) (subtopology X S),
                  relative_homology_group p X S],
                 [hom_induced (p - 1) (subtopology X S) {} X {} id, hom_boundary
p X S])
     (is exact_seq ([?G1,?G2,?G3], [?h1,?h2]))
  using homology_exactness_axiom_2 [of p X S]
  apply (simp add: group_hom_axioms_def group_hom_def hom_boundary_reduced_hom
hom_induced_reduced_hom)
  apply (simp add: reduced_homology_group_def group_hom.subgroup_kernel group_hom_axioms_def
group_hom_def hom_induced_hom)
  using hom_boundary_reduced_hom [of p X S]
  apply (auto simp: image_def set_eq_iff)
  by (metis carrier_reduced_homology_group hom_in_carrier set_eq_iff)


lemma homology_exactness_reduced_3:
    exact_seq([relative_homology_group p X S,
                reduced_homology_group p X,
                reduced_homology_group p (subtopology X S)],
               [hom_induced p X {} X S id, hom_induced p (subtopology X S) {} X
{} id])
     (is exact_seq ([?G1,?G2,?G3], [?h1,?h2]))
proof -
  have kernel ?G2 ?G1 ?h1 =
      ?h2 ` carrier ?G3
  proof -
    obtain U where U:
      (hom_induced p (subtopology X S) {} X {} id) ` carrier ?G3 ⊆ U
      (hom_induced p (subtopology X S) {} X {} id) ` carrier ?G3
      ⊆ (hom_induced p (subtopology X S) {} X {} id) ` carrier (homology_group
p (subtopology X S))
      U ∩ kernel (homology_group p X) ?G1 (hom_induced p X {} X S id)
     = kernel ?G2 ?G1 (hom_induced p X {} X S id)
```

U ∩ (*hom_induced p* (*subtopology X S*) {} *X* {} *id*) ‘ *carrier* (*homology_group p* (*subtopology X S*))
⊆ (*hom_induced p* (*subtopology X S*) {} *X* {} *id*) ‘ *carrier ?G3*
proof
show *?h2* ‘ *carrier ?G3* ⊆ *carrier ?G2*
by (*simp add*: *hom_induced_reduced image_subset_iff*)
show *?h2* ‘ *carrier ?G3* ⊆ *?h2* ‘ *carrier* (*homology_group p* (*subtopology X S*))
by (*meson carrier_reduced_homology_group_subset image_mono*)
have *subgroup* (*kernel* (*homology_group p X*) (*homology_group p* (*discrete_topology* {()}))
(*hom_induced p X* {} (*discrete_topology* {()}) {} (λ*x*. ())))
(*homology_group p X*)
by (*simp add*: *group.normal_invE*(*1*) *group_hom.normal_kernel group_hom_axioms_def group_hom_def hom_induced_empty_hom*)
then show *carrier ?G2* ∩ *kernel* (*homology_group p X*) *?G1 ?h1* = *kernel ?G2 ?G1 ?h1*
unfolding *carrier_reduced_homology_group*
by (*auto simp*: *reduced_homology_group_def*)
show *carrier ?G2* ∩ *?h2* ‘ *carrier* (*homology_group p* (*subtopology X S*))
⊆ *?h2* ‘ *carrier ?G3*
by (*force simp*: *carrier_reduced_homology_group kernel_def hom_induced_compose'*)
qed
with *homology_exactness_axiom_3* [*of p X S*] **show** *?thesis*
by (*fastforce simp add*:)
qed
then show *?thesis*
apply (*simp add*: *group_hom_axioms_def group_hom_def hom_boundary_reduced_hom hom_induced_reduced_hom*)
apply (*simp add*: *group.hom_from_subgroup_generated hom_induced_hom reduced_homology_group_def*)
done
qed

0.3.2 More homology properties of deformations, retracts, contractible spaces

lemma *iso_relative_homology_of_contractible*:
⟦*contractible_space X*; *topspace X* ∩ *S* ≠ {}⟧
⟹ *hom_boundary p X S*
∈ *iso* (*relative_homology_group p X S*) (*reduced_homology_group*(*p* − *1*) (*subtopology X S*))
using *very_short_exact_sequence*
[*of reduced_homology_group* (*p* − *1*) *X*
reduced_homology_group (*p* − *1*) (*subtopology X S*)
relative_homology_group p X S
reduced_homology_group p X
hom_induced (*p* − *1*) (*subtopology X S*) {} *X* {} *id*
hom_boundary p X S

hom_induced p X {} X S id]
by (*meson exact_seq_cons_iff homology_exactness_reduced_1 homology_exactness_reduced_2 trivial_reduced_homology_group_contractible_space*)

lemma *isomorphic_group_relative_homology_of_contractible*:
⟦*contractible_space X*; *topspace X* ∩ *S* ≠ {}⟧
⟹ *relative_homology_group p X S* ≅
reduced_homology_group(*p* − *1*) (*subtopology X S*)
by (*meson iso_relative_homology_of_contractible is_isoI*)

lemma *isomorphic_group_reduced_homology_of_contractible*:
⟦*contractible_space X*; *topspace X* ∩ *S* ≠ {}⟧
⟹ *reduced_homology_group p* (*subtopology X S*) ≅ *relative_homology_group*(*p* + *1*) *X S*
by (*metis add.commute add_diff_cancel_left′ group.iso_sym group_relative_homology_group isomorphic_group_relative_homology_of_contractible*)

lemma *iso_reduced_homology_by_contractible*:
⟦*contractible_space*(*subtopology X S*); *topspace X* ∩ *S* ≠ {}⟧
⟹ (*hom_induced p X* {} *X S id*) ∈ *iso* (*reduced_homology_group p X*) (*relative_homology_group p X S*)
using *very_short_exact_sequence*
[*of reduced_homology_group* (*p* − *1*) (*subtopology X S*)
relative_homology_group p X S
reduced_homology_group p X
reduced_homology_group p (*subtopology X S*)
hom_boundary p X S
hom_induced p X {} *X S id*
hom_induced p (*subtopology X S*) {} *X* {} *id*]
by (*meson exact_seq_cons_iff homology_exactness_reduced_1 homology_exactness_reduced_3 trivial_reduced_homology_group_contractible_space*)

lemma *isomorphic_reduced_homology_by_contractible*:
⟦*contractible_space*(*subtopology X S*); *topspace X* ∩ *S* ≠ {}⟧
⟹ *reduced_homology_group p X* ≅ *relative_homology_group p X S*
using *is_isoI iso_reduced_homology_by_contractible* **by** *blast*

lemma *isomorphic_relative_homology_by_contractible*:
⟦*contractible_space*(*subtopology X S*); *topspace X* ∩ *S* ≠ {}⟧
⟹ *relative_homology_group p X S* ≅ *reduced_homology_group p X*
using *group.iso_sym group_reduced_homology_group isomorphic_reduced_homology_by_contractible*
by *blast*

lemma *isomorphic_reduced_homology_by_singleton*:
a ∈ *topspace X* ⟹ *reduced_homology_group p X* ≅ *relative_homology_group p X* ({*a*})
by (*simp add*: *contractible_space_subtopology_singleton isomorphic_reduced_homology_by_contractible*)

lemma *isomorphic_relative_homology_by_singleton*:

a ∈ *topspace X* ⟹ *relative_homology_group p X* ({*a*}) ≅ *reduced_homology_group p X*
by (*simp add*: *group.iso_sym isomorphic_reduced_homology_by_singleton*)

lemma *reduced_homology_group_pair*:
assumes *t1_space X* **and** *a*: *a* ∈ *topspace X* **and** *b*: *b* ∈ *topspace X* **and** *a* ≠ *b*
shows *reduced_homology_group p* (*subtopology X* {*a*,*b*}) ≅ *homology_group p* (*subtopology X* {*a*})
(**is** *?lhs* ≅ *?rhs*)
proof −
have *?lhs* ≅ *relative_homology_group p* (*subtopology X* {*a*,*b*}) {*b*}
by (*simp add*: *b isomorphic_reduced_homology_by_singleton topspace_subtopology*)
also have . . . ≅ *?rhs*
proof −
have *sub*: *subtopology X* {*a*, *b*} *closure_of* {*b*} ⊆ *subtopology X* {*a*, *b*} *interior_of* {*b*}
by (*simp add*: *assms t1_space_subtopology closure_of_singleton subtopology_eq_discrete_topology_finite discrete_topology_closure_of*)
show *?thesis*
using *homology_excision_axiom* [*OF sub*, *of* {*a*,*b*} *p*]
by (*simp add*: *assms*(*4*) *group.iso_sym is_isoI subtopology_subtopology*)
qed
finally show *?thesis* **.**
qed

lemma *deformation_retraction_relative_homology_group_isomorphisms*:
⟦*retraction_maps X Y r s*; *r* ‘ *U* ⊆ *V*; *s* ‘ *V* ⊆ *U*; *homotopic_with* (λ*h*. *h* ‘ *U* ⊆ *U*) *X X* (*s* ∘ *r*) *id*⟧
⟹ *group_isomorphisms* (*relative_homology_group p X U*) (*relative_homology_group p Y V*)
(*hom_induced p X U Y V r*) (*hom_induced p Y V X U s*)
apply (*simp add*: *retraction_maps_def*)
apply (*rule homotopy_equivalence_relative_homology_group_isomorphisms*)
apply (*auto simp*: *image_subset_iff continuous_map_compose homotopic_with_equal*)
done

lemma *deformation_retract_relative_homology_group_isomorphisms*:
⟦*retraction_maps X Y r id*; *V* ⊆ *U*; *r* ‘ *U* ⊆ *V*; *homotopic_with* (λ*h*. *h* ‘ *U* ⊆ *U*) *X X r id*⟧
⟹ *group_isomorphisms* (*relative_homology_group p X U*) (*relative_homology_group p Y V*)
(*hom_induced p X U Y V r*) (*hom_induced p Y V X U id*)
by (*simp add*: *deformation_retraction_relative_homology_group_isomorphisms*)

lemma *deformation_retract_relative_homology_group_isomorphism*:
⟦*retraction_maps X Y r id*; *V* ⊆ *U*; *r* ‘ *U* ⊆ *V*; *homotopic_with* (λ*h*. *h* ‘ *U* ⊆ *U*) *X X r id*⟧

⟹ (*hom_induced p X U Y V r*) ∈ *iso* (*relative_homology_group p X U*) (*relative_homology_group p Y V*)
by (*metis deformation_retract_relative_homology_group_isomorphisms group_isomorphisms_imp_iso*)

lemma *deformation_retract_relative_homology_group_isomorphism_id*:
⟦*retraction_maps X Y r id*; *V* ⊆ *U*; *r* ‘ *U* ⊆ *V*; *homotopic_with* (λ*h*. *h* ‘ *U* ⊆ *U*) *X X r id*⟧
⟹ (*hom_induced p Y V X U id*) ∈ *iso* (*relative_homology_group p Y V*) (*relative_homology_group p X U*)
by (*metis deformation_retract_relative_homology_group_isomorphisms group_isomorphisms_imp_iso group_isomorphisms_sym*)

lemma *deformation_retraction_imp_isomorphic_relative_homology_groups*:
⟦*retraction_maps X Y r s*; *r* ‘ *U* ⊆ *V*; *s* ‘ *V* ⊆ *U*; *homotopic_with* (λ*h*. *h* ‘ *U* ⊆ *U*) *X X* (*s* ∘ *r*) *id*⟧
⟹ *relative_homology_group p X U* ≅ *relative_homology_group p Y V*
by (*blast intro*: *is_isoI group_isomorphisms_imp_iso deformation_retraction_relative_homology_group_isomorp*

lemma *deformation_retraction_imp_isomorphic_homology_groups*:
⟦*retraction_maps X Y r s*; *homotopic_with* (λ*h*. *True*) *X X* (*s* ∘ *r*) *id*⟧
⟹ *homology_group p X* ≅ *homology_group p Y*
by (*simp add*: *deformation_retraction_imp_homotopy_equivalent_space homotopy_equivalent_space_imp_isomorphic_homology_groups*)

lemma *deformation_retract_imp_isomorphic_relative_homology_groups*:
⟦*retraction_maps X X′ r id*; *V* ⊆ *U*; *r* ‘ *U* ⊆ *V*; *homotopic_with* (λ*h*. *h* ‘ *U* ⊆ *U*) *X X r id*⟧
⟹ *relative_homology_group p X U* ≅ *relative_homology_group p X′ V*
by (*simp add*: *deformation_retraction_imp_isomorphic_relative_homology_groups*)

lemma *deformation_retract_imp_isomorphic_homology_groups*:
⟦*retraction_maps X X′ r id*; *homotopic_with* (λ*h*. *True*) *X X r id*⟧
⟹ *homology_group p X* ≅ *homology_group p X′*
by (*simp add*: *deformation_retraction_imp_isomorphic_homology_groups*)

lemma *epi_hom_induced_inclusion*:
assumes *homotopic_with* (λ*x*. *True*) *X X id f* **and** *f* ‘ (*topspace X*) ⊆ *S*
shows (*hom_induced p* (*subtopology X S*) {} *X* {} *id*)
∈ *epi* (*homology_group p* (*subtopology X S*)) (*homology_group p X*)
proof (*rule epi_right_invertible*)
show *hom_induced p* (*subtopology X S*) {} *X* {} *id*
∈ *hom* (*homology_group p* (*subtopology X S*)) (*homology_group p X*)
by (*simp add*: *hom_induced_empty_hom*)
show *hom_induced p X* {} (*subtopology X S*) {} *f*
∈ *carrier* (*homology_group p X*) → *carrier* (*homology_group p* (*subtopology X S*))
by (*simp add*: *hom_induced_carrier*)
fix *x*

assume *x* ∈ *carrier* (*homology_group p X*)
then show *hom_induced p* (*subtopology X S*) {} *X* {} *id* (*hom_induced p X* {} (*subtopology X S*) {} *f x*) = *x*
by (*metis assms continuous_map_id_subt continuous_map_in_subtopology hom_induced_compose′ hom_induced_id homology_homotopy_empty homotopic_with_imp_continuous image_empty order_refl*)
qed

lemma *trivial_homomorphism_hom_induced_relativization*:
assumes *homotopic_with* (*λx. True*) *X X id f* **and** *f* ‘ (*topspace X*) ⊆ *S*
shows *trivial_homomorphism* (*homology_group p X*) (*relative_homology_group p X S*)
(*hom_induced p X* {} *X S id*)
proof −
have (*hom_induced p* (*subtopology X S*) {} *X* {} *id*)
∈ *epi* (*homology_group p* (*subtopology X S*)) (*homology_group p X*)
by (*metis assms epi_hom_induced_inclusion*)
then show *?thesis*
using *homology_exactness_axiom_3* [*of p X S*] *homology_exactness_axiom_1* [*of p X S*]
by (*simp add*: *epi_def group.trivial_homomorphism_image group_hom.trivial_hom_iff*)
qed

lemma *mon_hom_boundary_inclusion*:
assumes *homotopic_with* (*λx. True*) *X X id f* **and** *f* ‘ (*topspace X*) ⊆ *S*
shows (*hom_boundary p X S*) ∈ *mon*
(*relative_homology_group p X S*) (*homology_group* (*p* − *1*) (*subtopology X S*))
proof −
have (*hom_induced p* (*subtopology X S*) {} *X* {} *id*)
∈ *epi* (*homology_group p* (*subtopology X S*)) (*homology_group p X*)
by (*metis assms epi_hom_induced_inclusion*)
then show *?thesis*
using *homology_exactness_axiom_3* [*of p X S*] *homology_exactness_axiom_1* [*of p X S*]
apply (*simp add*: *mon_def epi_def hom_boundary_hom*)
by (*metis* (*no_types*, *opaque_lifting*) *group_hom.trivial_hom_iff group_hom.trivial_ker_imp_inj group_hom_axioms_def group_hom_def group_relative_homology_group hom_boundary_hom*)
qed

lemma *short_exact_sequence_hom_induced_relativization*:
assumes *homotopic_with* (*λx. True*) *X X id f* **and** *f* ‘ (*topspace X*) ⊆ *S*
shows *short_exact_sequence* (*homology_group* (*p*−*1*) *X*) (*homology_group* (*p*−*1*) (*subtopology X S*)) (*relative_homology_group p X S*)
(*hom_induced* (*p*−*1*) (*subtopology X S*) {} *X* {} *id*) (*hom_boundary p X S*)
unfolding *short_exact_sequence_iff*

```
  by (intro conjI homology_exactness_axiom_2 epi_hom_induced_inclusion [OF
assms] mon_hom_boundary_inclusion [OF assms])


lemma group_isomorphisms_homology_group_prod_deformation:
  fixes p::int
  assumes homotopic_with (λx. True) X X id f and f ‘ (topspace X) ⊆ S
  obtains H K where
    subgroup H (homology_group p (subtopology X S))
    subgroup K (homology_group p (subtopology X S))
    (λ(x, y). x ⊗homology_group p (subtopology X S) y)
          ∈ Group.iso (subgroup_generated (homology_group p (subtopology X S))
H ××
                        subgroup_generated (homology_group p (subtopology X S)) K)
                         (homology_group p (subtopology X S))
    hom_boundary (p + 1) X S
     ∈ Group.iso (relative_homology_group (p + 1) X S)
         (subgroup_generated (homology_group p (subtopology X S)) H)
    hom_induced p (subtopology X S) {} X {} id
     ∈ Group.iso
         (subgroup_generated (homology_group p (subtopology X S)) K)
         (homology_group p X)
proof −
  let ?rhs = relative_homology_group (p + 1) X S
  let ?pXS = homology_group p (subtopology X S)
  let ?pX = homology_group p X
  let ?hb = hom_boundary (p + 1) X S
  let ?hi = hom_induced p (subtopology X S) {} X {} id
  have x: short_exact_sequence (?pX) ?pXS ?rhs ?hi ?hb
    using short_exact_sequence_hom_induced_relativization [OF assms, of p +
1] by simp
  have contf: continuous_map X (subtopology X S) f
   by (meson assms continuous_map_in_subtopology homotopic_with_imp_continuous_maps)
   obtain H K where HK: H ◁ ?pXS subgroup K ?pXS H ∩ K ⊆ {one ?pXS}
set_mult ?pXS H K = carrier ?pXS
   and iso: ?hb ∈ iso ?rhs (subgroup_generated ?pXS H) ?hi ∈ iso (subgroup_generated
?pXS K) ?pX
    apply (rule splitting_lemma_right [OF x, where g' = hom_induced p X {}
(subtopology X S) {} f])
      apply (simp add: hom_induced_empty_hom)
     apply (simp add: contf hom_induced_compose')
    apply (metis (full_types) assms(1) hom_induced_id homology_homotopy_empty)
    apply blast
    done
  show ?thesis
  proof
    show subgroup H ?pXS
      using HK(1) normal_imp_subgroup by blast
    then show (λ(x, y). x ⊗?pXS y)
```

∈ *Group.iso* (*subgroup_generated* (*?pXS*) *H* ×× *subgroup_generated* (*?pXS*) *K*) (*?pXS*)
by (*meson HK abelian_relative_homology_group group_disjoint_sum.iso_group_mul group_disjoint_sum_def group_relative_homology_group*)
show *subgroup K ?pXS*
by (*rule HK*)
show *hom_boundary* (*p* + *1*) *X S* ∈ *Group.iso ?rhs* (*subgroup_generated* (*?pXS*) *H*)
using *iso int_ops*(*4*) **by** *presburger*
show *hom_induced p* (*subtopology X S*) {} *X* {} *id* ∈ *Group.iso* (*subgroup_generated* (*?pXS*) *K*) (*?pX*)
by (*simp add*: *iso*(*2*))
qed
qed

lemma *iso_homology_group_prod_deformation*:
assumes *homotopic_with* (λ*x*. *True*) *X X id f* **and** *f* ' (*topspace X*) ⊆ *S*
shows *homology_group p* (*subtopology X S*)
≅ *DirProd* (*homology_group p X*) (*relative_homology_group*(*p* + *1*) *X S*)
(**is** *?G* ≅ *DirProd ?H ?R*)
proof −
obtain *H K* **where** *HK*:
(λ(*x*, *y*). $x \otimes_{?G} y$)
∈ *Group.iso* (*subgroup_generated* (*?G*) *H* ×× *subgroup_generated* (*?G*) *K*) (*?G*)
hom_boundary (*p* + *1*) *X S* ∈ *Group.iso* (*?R*) (*subgroup_generated* (*?G*) *H*)
hom_induced p (*subtopology X S*) {} *X* {} *id* ∈ *Group.iso* (*subgroup_generated* (*?G*) *K*) (*?H*)
by (*blast intro*: *group_isomorphisms_homology_group_prod_deformation* [*OF assms*])
have *?G* ≅ *DirProd* (*subgroup_generated* (*?G*) *H*) (*subgroup_generated* (*?G*) *K*)
by (*meson DirProd_group HK*(*1*) *group.group_subgroup_generated group.iso_sym group_relative_homology_group is_isoI*)
also have ... ≅ *DirProd ?R ?H*
by (*meson HK group.DirProd_iso_trans group.group_subgroup_generated group.iso_sym group_relative_homology_group is_isoI*)
also have ... ≅ *DirProd ?H ?R*
by (*simp add*: *DirProd_commute_iso*)
finally show *?thesis* **.**
qed

lemma *iso_homology_contractible_space_subtopology1*:
assumes *contractible_space X S* ⊆ *topspace X S* ≠ {}
shows *homology_group 0* (*subtopology X S*) ≅ *DirProd integer_group* (*relative_homology_group*(*1*) *X S*)
proof −

obtain *f* **where** *homotopic_with* (λ*x*. *True*) *X X id f* **and** *f* ' (*topspace X*) ⊆ *S*
using *assms contractible_space_alt* **by** *fastforce*
then have *homology_group 0* (*subtopology X S*) ≅ *homology_group 0 X* ×× *relative_homology_group 1 X S*
using *iso_homology_group_prod_deformation* [*of X* _ *S 0*] **by** *auto*
also have ... ≅ *integer_group* ×× *relative_homology_group 1 X S*
using *assms contractible_imp_path_connected_space group.DirProd_iso_trans group_relative_homology_group iso_refl isomorphic_integer_zeroth_homology_group*
by *blast*
finally show *?thesis* **.**
qed

lemma *iso_homology_contractible_space_subtopology2*:
⟦*contractible_space X*; *S* ⊆ *topspace X*; *p* ≠ *0*; *S* ≠ {}⟧
⟹ *homology_group p* (*subtopology X S*) ≅ *relative_homology_group* (*p* + *1*) *X S*
by (*metis* (*no_types*, *opaque_lifting*) *add.commute isomorphic_group_reduced_homology_of_contractible topspace_subtopology topspace_subtopology_subset un_reduced_homology_group*)

lemma *trivial_relative_homology_group_contractible_spaces*:
⟦*contractible_space X*; *contractible_space*(*subtopology X S*); *topspace X* ∩ *S* ≠ {}⟧
⟹ *trivial_group*(*relative_homology_group p X S*)
using *group_reduced_homology_group group_relative_homology_group isomorphic_group_triviality isomorphic_relative_homology_by_contractible trivial_reduced_homology_group_contractibl*
by *blast*

lemma *trivial_relative_homology_group_alt*:
assumes *contf*: *continuous_map X* (*subtopology X S*) *f* **and** *hom*: *homotopic_with* (λ*k*. *k* ' *S* ⊆ *S*) *X X f id*
shows *trivial_group* (*relative_homology_group p X S*)
proof (*rule trivial_relative_homology_group_gen* [*OF contf*])
show *homotopic_with* (λ*h*. *True*) (*subtopology X S*) (*subtopology X S*) *f id*
using *hom* **unfolding** *homotopic_with_def*
apply (*rule ex_forward*)
apply (*auto simp*: *prod_topology_subtopology continuous_map_in_subtopology continuous_map_from_subtopology image_subset_iff topspace_subtopology*)
done
show *homotopic_with* (λ*k*. *True*) *X X f id*
using *assms* **by** (*force simp*: *homotopic_with_def*)
qed

lemma *iso_hom_induced_relativization_contractible*:
assumes *contractible_space*(*subtopology X S*) *contractible_space*(*subtopology X T*) *T* ⊆ *S topspace X* ∩ *T* ≠ {}
shows (*hom_induced p X T X S id*) ∈ *iso* (*relative_homology_group p X T*) (*relative_homology_group p X S*)
proof (*rule very_short_exact_sequence*)

show *exact_seq*
([*relative_homology_group*(*p* − *1*) (*subtopology X S*) *T*, *relative_homology_group p X S*, *relative_homology_group p X T*, *relative_homology_group p* (*subtopology X S*) *T*],
[*hom_relboundary p X S T*, *hom_induced p X T X S id*, *hom_induced p* (*subtopology X S*) *T X T id*])
using *homology_exactness_triple_1* [*OF* ‹*T* ⊆ *S*›] *homology_exactness_triple_3* [*OF* ‹*T* ⊆ *S*›]
by *fastforce*
show *trivial_group* (*relative_homology_group p* (*subtopology X S*) *T*) *trivial_group* (*relative_homology_group*(*p* − *1*) (*subtopology X S*) *T*)
using *assms*
by (*force simp*: *inf.absorb_iff2 subtopology_subtopology topspace_subtopology intro*!: *trivial_relative_homology_group_contractible_spaces*)+
qed

corollary *isomorphic_relative_homology_groups_relativization_contractible*:
assumes *contractible_space*(*subtopology X S*) *contractible_space*(*subtopology X T*) *T* ⊆ *S topspace X* ∩ *T* ≠ {}
shows *relative_homology_group p X T* ≅ *relative_homology_group p X S*
by (*rule is_isoI*) (*rule iso_hom_induced_relativization_contractible* [*OF assms*])

lemma *iso_hom_induced_inclusion_contractible*:
assumes *contractible_space X contractible_space*(*subtopology X S*) *T* ⊆ *S topspace X* ∩ *S* ≠ {}
shows (*hom_induced p* (*subtopology X S*) *T X T id*)
∈ *iso* (*relative_homology_group p* (*subtopology X S*) *T*) (*relative_homology_group p X T*)
proof (*rule very_short_exact_sequence*)
show *exact_seq*
([*relative_homology_group p X S*, *relative_homology_group p X T*,
relative_homology_group p (*subtopology X S*) *T*, *relative_homology_group* (*p*+*1*) *X S*],
[*hom_induced p X T X S id*, *hom_induced p* (*subtopology X S*) *T X T id*, *hom_relboundary* (*p*+*1*) *X S T*])
using *homology_exactness_triple_2* [*OF* ‹*T* ⊆ *S*›] *homology_exactness_triple_3* [*OF* ‹*T* ⊆ *S*›]
by (*metis add_diff_cancel_left′ diff_add_cancel exact_seq_cons_iff*)
show *trivial_group* (*relative_homology_group* (*p*+*1*) *X S*) *trivial_group* (*relative_homology_group p X S*)
using *assms*
by (*auto simp*: *subtopology_subtopology topspace_subtopology intro*!: *trivial_relative_homology_group_*
qed

corollary *isomorphic_relative_homology_groups_inclusion_contractible*:
assumes *contractible_space X contractible_space*(*subtopology X S*) *T* ⊆ *S topspace X* ∩ *S* ≠ {}
shows *relative_homology_group p* (*subtopology X S*) *T* ≅ *relative_homology_group p X T*

by (*rule is_isoI*) (*rule iso_hom_induced_inclusion_contractible* [*OF assms*])

lemma *iso_hom_relboundary_contractible*:
assumes *contractible_space X contractible_space*(*subtopology X T*) *T* ⊆ *S topspace X* ∩ *T* ≠ {}
shows *hom_relboundary p X S T*
∈ *iso* (*relative_homology_group p X S*) (*relative_homology_group* (*p* − *1*) (*subtopology X S*) *T*)
proof (*rule very_short_exact_sequence*)
show *exact_seq*
([*relative_homology_group* (*p* − *1*) *X T*, *relative_homology_group* (*p* − *1*) (*subtopology X S*) *T*, *relative_homology_group p X S*, *relative_homology_group p X T*],
[*hom_induced* (*p* − *1*) (*subtopology X S*) *T X T id*, *hom_relboundary p X S T*, *hom_induced p X T X S id*])
using *homology_exactness_triple_1* [*OF* ‹*T* ⊆ *S*›] *homology_exactness_triple_2* [*OF* ‹*T* ⊆ *S*›] **by** *simp*
show *trivial_group* (*relative_homology_group p X T*) *trivial_group* (*relative_homology_group* (*p* − *1*) *X T*)
using *assms*
by (*auto simp*: *subtopology_subtopology topspace_subtopology intro*!: *trivial_relative_homology_group_contractib*
qed

corollary *isomorphic_relative_homology_groups_relboundary_contractible*:
assumes *contractible_space X contractible_space*(*subtopology X T*) *T* ⊆ *S topspace X* ∩ *T* ≠ {}
shows *relative_homology_group p X S* ≅ *relative_homology_group* (*p* − *1*) (*subtopology X S*) *T*
by (*rule is_isoI*) (*rule iso_hom_relboundary_contractible* [*OF assms*])

lemma *isomorphic_relative_contractible_space_imp_homology_groups*:
assumes *contractible_space X contractible_space Y S* ⊆ *topspace X T* ⊆ *topspace Y*
and *ST*: *S* = {} ⟷ *T* = {}
and *iso*: ⋀*p*. *relative_homology_group p X S* ≅ *relative_homology_group p Y T*
shows *homology_group p* (*subtopology X S*) ≅ *homology_group p* (*subtopology Y T*)
proof (*cases T* = {})
case *True*
have *homology_group p* (*subtopology X* {}) ≅ *homology_group p* (*subtopology Y* {})
by (*simp add*: *homeomorphic_empty_space_eq homeomorphic_space_imp_isomorphic_homology_groups*)
then show *?thesis*
using *ST True* **by** *blast*
next
case *False*
show *?thesis*
proof (*cases p* = *0*)

```
    case True
   have homology_group p (subtopology X S) ≅ integer_group ×× relative_homology_group
1 X S
      using assms True ‹T ≠ {}›
      by (simp add: iso_homology_contractible_space_subtopology1)
    also have ... ≅ integer_group ×× relative_homology_group 1 Y T
      by (simp add: assms group.DirProd_iso_trans iso_refl)
    also have ... ≅ homology_group p (subtopology Y T)
    by (simp add: True ‹T ≠ {}› assms group.iso_sym iso_homology_contractible_space_subtopology1)
    finally show ?thesis .
  next
    case False
    have homology_group p (subtopology X S) ≅ relative_homology_group (p+1)
X S
      using assms False ‹T ≠ {}›
      by (simp add: iso_homology_contractible_space_subtopology2)
    also have ... ≅ relative_homology_group (p+1) Y T
      by (simp add: assms)
    also have ... ≅ homology_group p (subtopology Y T)
    by (simp add: False ‹T ≠ {}› assms group.iso_sym iso_homology_contractible_space_subtopology2)
    finally show ?thesis .
  qed
qed
```

0.3.3 Homology groups of spheres

```
lemma iso_reduced_homology_group_lower_hemisphere:
  assumes k ≤ n
  shows hom_induced p (nsphere n) {} (nsphere n) {x. x k ≤ 0} id
      ∈ iso (reduced_homology_group p (nsphere n)) (relative_homology_group p
(nsphere n) {x. x k ≤ 0})
proof (rule iso_reduced_homology_by_contractible)
  show contractible_space (subtopology (nsphere n) {x. x k ≤ 0})
    by (simp add: assms contractible_space_lower_hemisphere)
  have (λi. if i = k then −1 else 0) ∈ topspace (nsphere n) ∩ {x. x k ≤ 0}
    using assms by (simp add: nsphere if_distrib [of λx. x ^ 2] cong: if_cong)
  then show topspace (nsphere n) ∩ {x. x k ≤ 0} ≠ {}
    by blast
qed
```

```
lemma topspace_nsphere_1:
  assumes x ∈ topspace (nsphere n) shows (x k)² ≤ 1
proof (cases k ≤ n)
  case True
  have (∑ i ∈ {..n} − {k}. (x i)²) = (∑ i≤n. (x i)²) − (x k)²
    using ‹k ≤ n› by (simp add: sum_diff)
  then show ?thesis
    using assms
```

```
    apply (simp add: nsphere)
    by (metis diff_ge_0_iff_ge sum_nonneg zero_le_power2)
next
  case False
  then show ?thesis
    using assms by (simp add: nsphere)
qed

lemma topspace_nsphere_1_eq_0:
  fixes x :: nat ⇒ real
  assumes x: x ∈ topspace (nsphere n) and xk: (x k)² = 1 and i ≠ k
  shows x i = 0
proof (cases i ≤ n)
  case True
  have k ≤ n
    using x
    by (simp add: nsphere) (metis not_less xk zero_neq_one zero_power2)
  have (∑ i ∈ {..n} − {k}. (x i)²) = (∑ i≤n. (x i)²) − (x k)²
    using ‹k ≤ n› by (simp add: sum_diff)
  also have ... = 0
    using assms by (simp add: nsphere)
  finally have ∀ i∈{..n} − {k}. (x i)² = 0
    by (simp add: sum_nonneg_eq_0_iff)
  then show ?thesis
    using True ‹i ≠ k› by auto
next
  case False
  with x show ?thesis
    by (simp add: nsphere)
qed


proposition iso_relative_homology_group_upper_hemisphere:
   (hom_induced p (subtopology (nsphere n) {x. x k ≥ 0}) {x. x k = 0} (nsphere
n) {x. x k ≤ 0} id)
  ∈ iso (relative_homology_group p (subtopology (nsphere n) {x. x k ≥ 0}) {x. x
k = 0})
       (relative_homology_group p (nsphere n) {x. x k ≤ 0}) (is ?h ∈ iso ?G ?H)
proof −
  have topspace (nsphere n) ∩ {x. x k < − 1 / 2} ⊆ {x ∈ topspace (nsphere n).
x k ∈ {y. y ≤ − 1 / 2}}
    by force
  moreover have closedin (nsphere n) {x ∈ topspace (nsphere n). x k ∈ {y. y ≤
− 1 / 2}}
   apply (rule closedin_continuous_map_preimage [OF continuous_map_nsphere_projection])
    using closed_Collect_le [of id λx::real. −1/2] apply simp
    done
   ultimately have nsphere n closure_of {x. x k < −1/2} ⊆ {x ∈ topspace
(nsphere n). x k ∈ {y. y ≤ −1/2}}
```

by (*metis* (*no_types*, *lifting*) *closure_of_eq closure_of_mono closure_of_restrict*)
also have ... ⊆ {*x* ∈ *topspace* (*nsphere n*). *x k* ∈ {*y*. *y* < *0*}}
by *force*
also have ... ⊆ *nsphere n interior_of* {*x*. *x k* ≤ *0*}
proof (*rule interior_of_maximal*)
show {*x* ∈ *topspace* (*nsphere n*). *x k* ∈ {*y*. *y* < *0*}} ⊆ {*x*. *x k* ≤ *0*}
by *force*
show *openin* (*nsphere n*) {*x* ∈ *topspace* (*nsphere n*). *x k* ∈ {*y*. *y* < *0*}}
apply (*rule openin_continuous_map_preimage* [*OF continuous_map_nsphere_projection*])
using *open_Collect_less* [*of id* λ*x*::*real*. *0*] **apply** *simp*
done
qed
finally have *nn*: *nsphere n closure_of* {*x*. *x k* < −*1*/*2*} ⊆ *nsphere n interior_of* {*x*. *x k* ≤ *0*} .
have [*simp*]: {*x*::*nat*⇒*real*. *x k* ≤ *0*} − {*x*. *x k* < − (*1*/*2*)} = {*x*. −*1*/*2* ≤ *x k* ∧ *x k* ≤ *0*}
UNIV − {*x*::*nat*⇒*real*. *x k* < *a*} = {*x*. *a* ≤ *x k*} **for** *a*
by *auto*
let *?T01* = *top_of_set* {*0*..*1*::*real*}
let *?X12* = *subtopology* (*nsphere n*) {*x*. −*1*/*2* ≤ *x k*}
have *1*: *hom_induced p ?X12* {*x*. −*1*/*2* ≤ *x k* ∧ *x k* ≤ *0*} (*nsphere n*) {*x*. *x k* ≤ *0*} *id*
∈ *iso* (*relative_homology_group p ?X12* {*x*. −*1*/*2* ≤ *x k* ∧ *x k* ≤ *0*})
?H
using *homology_excision_axiom* [*OF nn subset_UNIV*, *of p*] **by** *simp*
define *h* **where** *h* ≡ λ(*T*,*x*). *let y* = *max* (*x k*) (−*T*) *in*
(λ*i*. *if i* = *k then y else sqrt*(*1* − *y* ^ *2*) / *sqrt*(*1* − *x k* ^ *2*) ∗ *x i*)
have *h*: *h*(*T*,*x*) = *x* **if** *0* ≤ *T T* ≤ *1* ($\sum$ *i*≤*n*. $(x\ i)^2$) = *1* **and** *0*: ∀ *i*>*n*. *x i* = *0* −*T* ≤ *x k* **for** *T x*
using *that* **by** (*force simp*: *nsphere h_def Let_def max_def intro*!: *topspace_nsphere_1_eq_0*)
have *continuous_map* (*prod_topology ?T01 ?X12*) *euclideanreal* (λ*x*. *h x i*) **for** *i*
proof −
show *?thesis*
proof (*rule continuous_map_eq*)
show *continuous_map* (*prod_topology ?T01 ?X12*)
euclideanreal (λ(*T*, *x*). *if 0* ≤ *x k then x i else h* (*T*, *x*) *i*)
unfolding *case_prod_unfold*
proof (*rule continuous_map_cases_le*)
show *continuous_map* (*prod_topology ?T01 ?X12*) *euclideanreal* (λ*x*. *snd x k*)
apply (*subst continuous_map_of_snd* [*unfolded o_def*])
by (*simp add*: *continuous_map_from_subtopology continuous_map_nsphere_projection*)
next
show *continuous_map* (*subtopology* (*prod_topology ?T01 ?X12*) {*p* ∈ *topspace* (*prod_topology ?T01 ?X12*). *0* ≤ *snd p k*})
euclideanreal (λ*x*. *snd x i*)
apply (*rule continuous_map_from_subtopology*)

```
          apply (subst continuous_map_of_snd [unfolded o_def])
        by (simp add: continuous_map_from_subtopology continuous_map_nsphere_projection)
       next
       note fst = continuous_map_into_fulltopology [OF continuous_map_subtopology_fst]
         have snd: continuous_map (subtopology (prod_topology ?T01 (subtopology
(nsphere n) T)) S) euclideanreal (λx. snd x k) for k S T
          apply (simp add: nsphere)
          apply (rule continuous_map_from_subtopology)
          apply (subst continuous_map_of_snd [unfolded o_def])
        using continuous_map_from_subtopology continuous_map_nsphere_projection
nsphere by fastforce
       show continuous_map (subtopology (prod_topology ?T01 ?X12) {p ∈ topspace
(prod_topology ?T01 ?X12). snd p k ≤ 0})
          euclideanreal (λx. h (fst x, snd x) i)
          apply (simp add: h_def case_prod_unfold Let_def)
          apply (intro conjI impI fst snd continuous_intros)
          apply (auto simp: nsphere power2_eq_1_iff)
          done
       qed (auto simp: nsphere h)
     qed (auto simp: nsphere h)
   qed
   moreover
   have h ` ({0..1} × (topspace (nsphere n) ∩ {x. − (1/2) ≤ x k}))
      ⊆ {x. (∑ i≤n. (x i)²) = 1 ∧ (∀ i>n. x i = 0)}
   proof −
     have (∑ i≤n. (h (T,x) i)²) = 1
       if x: x ∈ topspace (nsphere n) and xk: − (1/2) ≤ x k and T: 0 ≤ T T ≤ 1
for T x
     proof (cases −T ≤ x k )
       case True
       then show ?thesis
         using that by (auto simp: nsphere h)
     next
       case False
       with x ‹0 ≤ T› have k ≤ n
         apply (simp add: nsphere)
         by (metis neg_le_0_iff_le not_le)
       have 1 − (x k)² ≥ 0
         using topspace_nsphere_1 x by auto
       with False T ‹k ≤ n›
       have (∑ i≤n. (h (T,x) i)²) = T² + (1 − T²) ∗ (∑ i∈{..n} − {k}. (x i)² /
(1 − (x k)²))
         unfolding h_def Let_def max_def
             by (simp add: not_le square_le_1 power_mult_distrib power_divide
if_distrib [of λx. x ^ 2]
                 sum.delta_remove sum_distrib_left)
       also have ... = 1
         using x False xk ‹0 ≤ T›
           by (simp add: nsphere sum_diff not_le ‹k ≤ n› power2_eq_1_iff flip:
```

sum_divide_distrib)
finally show *?thesis* **.**
qed
moreover
have *h* (*T*,*x*) *i* = *0*
if *x* ∈ *topspace* (*nsphere n*) − (*1*/*2*) ≤ *x k* **and** *n* < *i 0* ≤ *T T* ≤ *1*
for *T x i*
proof (*cases* −*T* ≤ *x k*)
case *False*
then show *?thesis*
using *that* **by** (*auto simp*: *nsphere h_def Let_def not_le max_def*)
qed (*use that* **in** ‹*auto simp*: *nsphere h*›)
ultimately show *?thesis*
by *auto*
qed
ultimately
have *cmh*: *continuous_map* (*prod_topology ?T01 ?X12*) (*nsphere n*) *h*
by (*subst* (*2*) *nsphere*) (*simp add*: *continuous_map_in_subtopology continuous_map_componentwise_UNIV*)
have *hom_induced p* (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*})
(*topspace* (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*}) ∩ {*x*. *x k* = *0*}) *?X12*
(*topspace ?X12* ∩ {*x*. − *1*/*2* ≤ *x k* ∧ *x k* ≤ *0*}) *id*
∈ *iso* (*relative_homology_group p* (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*})
(*topspace* (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*}) ∩ {*x*. *x k* = *0*}))
(*relative_homology_group p ?X12* (*topspace ?X12* ∩ {*x*. − *1*/*2* ≤ *x k* ∧ *x k* ≤ *0*}))
proof (*rule deformation_retract_relative_homology_group_isomorphism_id*)
show *retraction_maps ?X12* (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*}) (*h* ∘ (λ*x*. (*0*,*x*))) *id*
unfolding *retraction_maps_def*
proof (*intro conjI ballI*)
show *continuous_map ?X12* (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*}) (*h* ∘ *Pair 0*)
apply (*simp add*: *continuous_map_in_subtopology*)
apply (*intro conjI continuous_map_compose* [*OF _ cmh*] *continuous_intros*)
apply (*auto simp*: *h_def Let_def*)
done
show *continuous_map* (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*}) *?X12 id*
by (*simp add*: *continuous_map_in_subtopology*) (*auto simp*: *nsphere*)
qed (*simp add*: *nsphere h*)
next
have *h0*: ⋀*xa*. ⟦*xa* ∈ *topspace* (*nsphere n*); − (*1*/*2*) ≤ *xa k*; *xa k* ≤ *0*⟧ ⟹ *h* (*0*, *xa*) *k* = *0*
by (*simp add*: *h_def Let_def*)
show (*h* ∘ (λ*x*. (*0*,*x*))) ‘ (*topspace ?X12* ∩ {*x*. − *1* / *2* ≤ *x k* ∧ *x k* ≤ *0*})
⊆ *topspace* (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*}) ∩ {*x*. *x k* = *0*}
apply (*auto simp*: *h0*)
apply (*rule subsetD* [*OF continuous_map_image_subset_topspace* [*OF cmh*]])

```
      apply (force simp: nsphere)
      done
    have hin: ⋀t x. ⟦x ∈ topspace (nsphere n); − (1/2) ≤ x k; 0 ≤ t; t ≤ 1⟧ ⟹
h (t,x) ∈ topspace (nsphere n)
     apply (rule subsetD [OF continuous_map_image_subset_topspace [OF cmh]])
      apply (force simp: nsphere)
      done
    have h1: ⋀x. ⟦x ∈ topspace (nsphere n); − (1/2) ≤ x k⟧ ⟹ h (1, x) = x
      by (simp add: h nsphere)
    have continuous_map (prod_topology ?T01 ?X12) (nsphere n) h
      using cmh by force
    then show homotopic_with
            (λh. h ‘ (topspace ?X12 ∩ {x. − 1 / 2 ≤ x k ∧ x k ≤ 0}) ⊆ topspace
?X12 ∩ {x. − 1 / 2 ≤ x k ∧ x k ≤ 0})
             ?X12 ?X12 (h ∘ (λx. (0,x))) id
      apply (subst homotopic_with, force)
      apply (rule_tac x=h in exI)
      apply (auto simp: hin h1 continuous_map_in_subtopology)
        apply (auto simp: h_def Let_def max_def)
      done
  qed auto
  then have 2: hom_induced p (subtopology (nsphere n) {x. 0 ≤ x k}) {x. x k =
0}
            ?X12 {x. − 1/2 ≤ x k ∧ x k ≤ 0} id
           ∈ Group.iso
              (relative_homology_group p (subtopology (nsphere n) {x. 0 ≤ x k})
{x. x k = 0})
             (relative_homology_group p ?X12 {x. − 1/2 ≤ x k ∧ x k ≤ 0})
   by (metis hom_induced_restrict relative_homology_group_restrict topspace_subtopology)
  show ?thesis
    using iso_set_trans [OF 2 1]
   by (simp add: subset_iff continuous_map_in_subtopology flip: hom_induced_compose)
qed


corollary iso_upper_hemisphere_reduced_homology_group:
  (hom_boundary (1 + p) (subtopology (nsphere (Suc n)) {x. x(Suc n) ≥ 0}) {x.
x(Suc n) = 0})
  ∈ iso (relative_homology_group (1 + p) (subtopology (nsphere (Suc n)) {x. x(Suc
n) ≥ 0}) {x. x(Suc n) = 0})
        (reduced_homology_group p (nsphere n))
proof −
  have {x. 0 ≤ x (Suc n)} ∩ {x. x (Suc n) = 0} = {x. x (Suc n) = (0::real)}
    by auto
  then have n: nsphere n = subtopology (subtopology (nsphere (Suc n)) {x. x(Suc
n) ≥ 0}) {x. x(Suc n) = 0}
    by (simp add: subtopology_nsphere_equator subtopology_subtopology)
  have ne: (λi. if i = n then 1 else 0) ∈ topspace (subtopology (nsphere (Suc n))
{x. 0 ≤ x (Suc n)}) ∩ {x. x (Suc n) = 0}
```

by (*simp add*: *nsphere if_distrib* [*of* λ*x*. *x* ^ *2*] *cong*: *if_cong*)
show *?thesis*
unfolding *n*
apply (*rule iso_relative_homology_of_contractible* [**where** *p* = *1* + *p*, *simplified*])
using *contractible_space_upper_hemisphere ne* **apply** *blast*+
done
qed

corollary *iso_reduced_homology_group_upper_hemisphere*:
assumes $k \le n$
shows *hom_induced p* (*nsphere n*) {} (*nsphere n*) {*x*. *x k* ≥ *0*} *id*
∈ *iso* (*reduced_homology_group p* (*nsphere n*)) (*relative_homology_group p* (*nsphere n*) {*x*. *x k* ≥ *0*})
proof (*rule iso_reduced_homology_by_contractible* [*OF contractible_space_upper_hemisphere* [*OF assms*]])
have (λ*i*. *if i* = *k then 1 else 0*) ∈ *topspace* (*nsphere n*) ∩ {*x*. *0* ≤ *x k*}
using *assms* **by** (*simp add*: *nsphere if_distrib* [*of* λ*x*. *x* ^ *2*] *cong*: *if_cong*)
then show *topspace* (*nsphere n*) ∩ {*x*. *0* ≤ *x k*} ≠ {}
by *blast*
qed

lemma *iso_relative_homology_group_lower_hemisphere*:
hom_induced p (*subtopology* (*nsphere n*) {*x*. *x k* ≤ *0*}) {*x*. *x k* = *0*} (*nsphere n*) {*x*. *x k* ≥ *0*} *id*
∈ *iso* (*relative_homology_group p* (*subtopology* (*nsphere n*) {*x*. *x k* ≤ *0*}) {*x*. *x k* = *0*})
(*relative_homology_group p* (*nsphere n*) {*x*. *x k* ≥ *0*}) (**is** *?k* ∈ *iso ?G ?H*)
proof −
define *r* **where** *r* ≡ λ*x i*. *if i* = *k then* −*x i else* (*x i*::*real*)
then have [*simp*]: *r* ∘ *r* = *id*
by *force*
have *cmr*: *continuous_map* (*subtopology* (*nsphere n*) *S*) (*nsphere n*) *r* **for** *S*
using *continuous_map_nsphere_reflection* [*of n k*]
by (*simp add*: *continuous_map_from_subtopology r_def*)
let *?f* = *hom_induced p* (*subtopology* (*nsphere n*) {*x*. *x k* ≤ *0*}) {*x*. *x k* = *0*}
(*subtopology* (*nsphere n*) {*x*. *x k* ≥ *0*}) {*x*. *x k* = *0*} *r*
let *?g* = *hom_induced p* (*subtopology* (*nsphere n*) {*x*. *x k* ≥ *0*}) {*x*. *x k* = *0*} (*nsphere n*) {*x*. *x k* ≤ *0*} *id*
let *?h* = *hom_induced p* (*nsphere n*) {*x*. *x k* ≤ *0*} (*nsphere n*) {*x*. *x k* ≥ *0*} *r*
obtain *f h* **where**
f: *f* ∈ *iso ?G* (*relative_homology_group p* (*subtopology* (*nsphere n*) {*x*. *x k* ≥ *0*}) {*x*. *x k* = *0*})
and *h*: *h* ∈ *iso* (*relative_homology_group p* (*nsphere n*) {*x*. *x k* ≤ *0*}) *?H*
and *eq*: *h* ∘ *?g* ∘ *f* = *?k*
proof
have *hmr*: *homeomorphic_map* (*nsphere n*) (*nsphere n*) *r*
unfolding *homeomorphic_map_maps*

by (*metis* ‹r ∘ r = id› *cmr homeomorphic_maps_involution pointfree_idE subtopology_topspace*)
then have *hmrs*: *homeomorphic_map* (*subtopology* (*nsphere n*) {*x*. *x k* ≤ *0*}) (*subtopology* (*nsphere n*) {*x*. *x k* ≥ *0*}) *r*
by (*simp add*: *homeomorphic_map_subtopologies_alt r_def*)
have *rimeq*: *r* ‘ (*topspace* (*subtopology* (*nsphere n*) {*x*. *x k* ≤ *0*}) ∩ {*x*. *x k* = *0*})
= *topspace* (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*}) ∩ {*x*. *x k* = *0*}
using *continuous_map_eq_topcontinuous_at continuous_map_nsphere_reflection topcontinuous_at_atin*
by (*fastforce simp*: *r_def Pi_iff*)
show *?f* ∈ *iso ?G* (*relative_homology_group p* (*subtopology* (*nsphere n*) {*x*. *x k* ≥ *0*}) {*x*. *x k* = *0*})
using *homeomorphic_map_relative_homology_iso* [*OF hmrs Int_lower1 rimeq*]
by (*metis hom_induced_restrict relative_homology_group_restrict*)
have *rimeq*: *r* ‘ (*topspace* (*nsphere n*) ∩ {*x*. *x k* ≤ *0*}) = *topspace* (*nsphere n*) ∩ {*x*. *0* ≤ *x k*}
by (*metis hmrs homeomorphic_imp_surjective_map topspace_subtopology*)
show *?h* ∈ *Group.iso* (*relative_homology_group p* (*nsphere n*) {*x*. *x k* ≤ *0*}) *?H*
using *homeomorphic_map_relative_homology_iso* [*OF hmr Int_lower1 rimeq*] **by** *simp*
have [*simp*]: ⋀*x*. *x k* = *0* ⟹ *r x k* = *0*
by (*auto simp*: *r_def*)
have *?h* ∘ *?g* ∘ *?f*
= *hom_induced p* (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*}) {*x*. *x k* = *0*} (*nsphere n*) {*x*. *0* ≤ *x k*} *r* ∘
hom_induced p (*subtopology* (*nsphere n*) {*x*. *x k* ≤ *0*}) {*x*. *x k* = *0*} (*subtopology* (*nsphere n*) {*x*. *0* ≤ *x k*}) {*x*. *x k* = *0*} *r*
apply (*subst hom_induced_compose* [*symmetric*])
using *continuous_map_nsphere_reflection* **apply** (*force simp*: *r_def*)+
done
also have ... = *?k*
apply (*subst hom_induced_compose* [*symmetric*])
apply (*simp_all add*: *image_subset_iff cmr*)
using *hmrs homeomorphic_imp_continuous_map* **apply** *blast*
done
finally show *?h* ∘ *?g* ∘ *?f* = *?k* **.**
qed
with *iso_relative_homology_group_upper_hemisphere* [*of p n k*]
have *h* ∘ *hom_induced p* (*subtopology* (*nsphere n*) {*f*. *0* ≤ *f k*}) {*f*. *f k* = *0*} (*nsphere n*) {*f*. *f k* ≤ *0*} *id* ∘ *f*
∈ *Group.iso ?G* (*relative_homology_group p* (*nsphere n*) {*f*. *0* ≤ *f k*})
using *f h iso_set_trans* **by** *blast*
then show *?thesis*
by (*simp add*: *eq*)
qed

lemma *iso_lower_hemisphere_reduced_homology_group*:
hom_boundary (*1* + *p*) (*subtopology* (*nsphere* (*Suc n*)) {*x*. *x*(*Suc n*) ≤ *0*}) {*x*. *x*(*Suc n*) = *0*}
∈ *iso* (*relative_homology_group* (*1* + *p*) (*subtopology* (*nsphere* (*Suc n*)) {*x*. *x*(*Suc n*) ≤ *0*})
{*x*. *x*(*Suc n*) = *0*})
(*reduced_homology_group p* (*nsphere n*))
proof −
have $\{x.\ (\sum i{\le}n.\ (x\ i)^2) = 1 \land (\forall i{>}n.\ x\ i = 0)\} =$
$(\{x.\ (\sum i{\le}n.\ (x\ i)^2) + (x\ (Suc\ n))^2 = 1 \land (\forall i{>}Suc\ n.\ x\ i = 0)\} \cap \{x.\ x\ (Suc\ n) \le 0\} \cap$
$\{x.\ x\ (Suc\ n) = (0{::}real)\})$
by (*force simp*: *dest*: *Suc_lessI*)
then have *n*: *nsphere n* = *subtopology* (*subtopology* (*nsphere* (*Suc n*)) {*x*. *x*(*Suc n*) ≤ *0*}) {*x*. *x*(*Suc n*) = *0*}
by (*simp add*: *nsphere subtopology_subtopology*)
have *ne*: (λ*i*. *if i* = *n then 1 else 0*) ∈ *topspace* (*subtopology* (*nsphere* (*Suc n*)) {*x*. *x* (*Suc n*) ≤ *0*}) ∩ {*x*. *x* (*Suc n*) = *0*}
by (*simp add*: *nsphere if_distrib* [*of* λ*x*. *x* ^ *2*] *cong*: *if_cong*)
show *?thesis*
unfolding *n*
apply (*rule iso_relative_homology_of_contractible* [**where** *p* = *1* + *p*, *simplified*])
using *contractible_space_lower_hemisphere ne* **apply** *blast*+
done
qed

lemma *isomorphism_sym*:
⟦*f* ∈ *iso G1 G2*; ⋀*x*. *x* ∈ *carrier G1* ⟹ *r*′(*f x*) = *f*(*r x*);
⋀*x*. *x* ∈ *carrier G1* ⟹ *r x* ∈ *carrier G1*; *group G1*; *group G2*⟧
⟹ ∃*f* ∈ *iso G2 G1*. ∀*x* ∈ *carrier G2*. *r*(*f x*) = *f*(*r*′ *x*)
apply (*clarsimp simp add*: *group.iso_iff_group_isomorphisms Bex_def*)
by (*metis* (*full_types*) *group_isomorphisms_def group_isomorphisms_sym hom_in_carrier*)

lemma *isomorphism_trans*:
⟦∃*f* ∈ *iso G1 G2*. ∀*x* ∈ *carrier G1*. *r2*(*f x*) = *f*(*r1 x*); ∃*f* ∈ *iso G2 G3*. ∀*x* ∈ *carrier G2*. *r3*(*f x*) = *f*(*r2 x*)⟧
⟹ ∃*f* ∈ *iso G1 G3*. ∀*x* ∈ *carrier G1*. *r3*(*f x*) = *f*(*r1 x*)
apply *clarify*
apply (*rename_tac g f*)
apply (*rule_tac x*=*f* ∘ *g* **in** *bexI*)
apply (*metis iso_iff comp_apply hom_in_carrier*)
using *iso_set_trans* **by** *blast*

lemma *reduced_homology_group_nsphere_step*:
∃*f* ∈ *iso*(*reduced_homology_group p* (*nsphere n*))
(*reduced_homology_group* (*1* + *p*) (*nsphere* (*Suc n*))).
∀*c* ∈ *carrier*(*reduced_homology_group p* (*nsphere n*)).

```
          hom_induced (1 + p) (nsphere(Suc n)) {} (nsphere(Suc n)) {}
                (λx i. if i = 0 then −x i else x i) (f c)
          = f (hom_induced p (nsphere n) {} (nsphere n) {} (λx i. if i = 0 then
−x i else x i) c)
proof −
  define r where r ≡ λx::nat⇒real. λi. if i = 0 then −x i else x i
  have cmr: continuous_map (nsphere n) (nsphere n) r for n
    unfolding r_def by (rule continuous_map_nsphere_reflection)
  have rsub: r ‘ {x. 0 ≤ x (Suc n)} ⊆ {x. 0 ≤ x (Suc n)} r ‘ {x. x (Suc n) ≤ 0}
⊆ {x. x (Suc n) ≤ 0} r ‘ {x. x (Suc n) = 0} ⊆ {x. x (Suc n) = 0}
    by (force simp: r_def)+
  let ?sub = subtopology (nsphere (Suc n)) {x. x (Suc n) ≥ 0}
  let ?G2 = relative_homology_group (1 + p) ?sub {x. x (Suc n) = 0}
  let ?r2 = hom_induced (1 + p) ?sub {x. x (Suc n) = 0} ?sub {x. x (Suc n) =
0} r
  let ?j = λp n. hom_induced p (nsphere n) {} (nsphere n) {} r
  show ?thesis
    unfolding r_def [symmetric]
  proof (rule isomorphism_trans)
    let ?f = hom_boundary (1 + p) ?sub {x. x (Suc n) = 0}
    show ∃f∈Group.iso (reduced_homology_group p (nsphere n)) ?G2.
          ∀c∈carrier (reduced_homology_group p (nsphere n)). ?r2 (f c) = f (?j p
n c)
    proof (rule isomorphism_sym)
      show ?f ∈ Group.iso ?G2 (reduced_homology_group p (nsphere n))
        using iso_upper_hemisphere_reduced_homology_group
        by (metis add.commute)
    next
      fix c
      assume c ∈ carrier ?G2
      have cmrs: continuous_map ?sub ?sub r
       by (metis (mono_tags, lifting) IntE cmr continuous_map_from_subtopology
continuous_map_in_subtopology image_subset_iff rsub(1) topspace_subtopology)
      have hom_induced p (nsphere n) {} (nsphere n) {} r ∘ hom_boundary (1 +
p) ?sub {x. x (Suc n) = 0}
          = hom_boundary (1 + p) ?sub {x. x (Suc n) = 0} ∘
           hom_induced (1 + p) ?sub {x. x (Suc n) = 0} ?sub {x. x (Suc n) = 0}
r
           using naturality_hom_induced [OF cmrs rsub(3), symmetric, of 1+p,
simplified]
         by (simp add: subtopology_subtopology subtopology_nsphere_equator flip:
Collect_conj_eq cong: rev_conj_cong)
      then show ?j p n (?f c) = ?f (hom_induced (1 + p) ?sub {x. x (Suc n) =
0} ?sub {x. x (Suc n) = 0} r c)
        by (metis comp_def)
    next
      fix c
      assume c ∈ carrier ?G2
      show hom_induced (1 + p) ?sub {x. x (Suc n) = 0} ?sub {x. x (Suc n) =
```

0} *r c* ∈ *carrier ?G2*
using *hom_induced_carrier* by *blast*
qed *auto*
next
let *?H2* = *relative_homology_group* (*1* + *p*) (*nsphere* (*Suc n*)) {*x*. *x* (*Suc n*) ≤ *0*}
let *?s2* = *hom_induced* (*1* + *p*) (*nsphere* (*Suc n*)) {*x*. *x* (*Suc n*) ≤ *0*} (*nsphere* (*Suc n*)) {*x*. *x* (*Suc n*) ≤ *0*} *r*
show ∃*f*∈*Group.iso ?G2* (*reduced_homology_group* (*1* + *p*) (*nsphere* (*Suc n*))). ∀ *c*∈*carrier ?G2*. *?j* (*1* + *p*) (*Suc n*) (*f c*)
= *f* (*?r2 c*)
proof (*rule isomorphism_trans*)
show ∃*f*∈*Group.iso ?G2 ?H2*.
∀ *c*∈*carrier ?G2*.
?s2 (*f c*) = *f* (*hom_induced* (*1* + *p*) *?sub* {*x*. *x* (*Suc n*) = *0*} *?sub* {*x*. *x* (*Suc n*) = *0*} *r c*)
proof (*intro ballI bexI*)
fix *c*
assume *c* ∈ *carrier* (*relative_homology_group* (*1* + *p*) *?sub* {*x*. *x* (*Suc n*) = *0*})
show *?s2* (*hom_induced* (*1* + *p*) *?sub* {*x*. *x* (*Suc n*) = *0*} (*nsphere* (*Suc n*)) {*x*. *x* (*Suc n*) ≤ *0*} *id c*)
= *hom_induced* (*1* + *p*) *?sub* {*x*. *x* (*Suc n*) = *0*} (*nsphere* (*Suc n*)) {*x*. *x* (*Suc n*) ≤ *0*} *id* (*?r2 c*)
apply (*simp add*: *rsub hom_induced_compose′ Collect_mono_iff cmr*)
apply (*subst hom_induced_compose′*)
apply (*simp_all add*: *continuous_map_in_subtopology continuous_map_from_subtopology* [*OF cmr*] *rsub*)
apply (*auto simp*: *r_def*)
done
qed (*simp add*: *iso_relative_homology_group_upper_hemisphere*)
next
let *?h* = *hom_induced* (*1* + *p*) (*nsphere*(*Suc n*)) {} (*nsphere* (*Suc n*)) {*x*. *x*(*Suc n*) ≤ *0*} *id*
show ∃*f*∈*Group.iso ?H2* (*reduced_homology_group* (*1* + *p*) (*nsphere* (*Suc n*))).
∀ *c*∈*carrier ?H2*. *?j* (*1* + *p*) (*Suc n*) (*f c*) = *f* (*?s2 c*)
proof (*rule isomorphism_sym*)
show *?h* ∈ *Group.iso* (*reduced_homology_group* (*1* + *p*) (*nsphere* (*Suc n*)))
(*relative_homology_group* (*1* + *p*) (*nsphere* (*Suc n*)) {*x*. *x* (*Suc n*) ≤ *0*})
using *iso_reduced_homology_group_lower_hemisphere* by *blast*
next
fix *c*
assume *c* ∈ *carrier* (*reduced_homology_group* (*1* + *p*) (*nsphere* (*Suc n*)))
show *?s2* (*?h c*) = *?h* (*?j* (*1* + *p*) (*Suc n*) *c*)
by (*simp add*: *hom_induced_compose′ cmr rsub*)
next
fix *c*

```
      assume c ∈ carrier (reduced_homology_group (1 + p) (nsphere (Suc n)))
      then show hom_induced (1 + p) (nsphere (Suc n)) {} (nsphere (Suc n))
{} r c
      ∈ carrier (reduced_homology_group (1 + p) (nsphere (Suc n)))
        by (simp add: hom_induced_reduced)
      qed auto
    qed
  qed
qed


lemma reduced_homology_group_nsphere_aux:
  if p = int n then reduced_homology_group n (nsphere n) ≅ integer_group
                  else trivial_group(reduced_homology_group p (nsphere n))
proof (induction n arbitrary: p)
  case 0
  let ?a = λi::nat. if i = 0 then 1 else (0::real)
  let ?b = λi::nat. if i = 0 then −1 else (0::real)
  have st: subtopology (powertop_real UNIV) {?a, ?b} = nsphere 0
  proof −
    have {?a, ?b} = {x. (x 0)^2 = 1 ∧ (∀i>0. x i = 0)}
      using power2_eq_iff by fastforce
    then show ?thesis
      by (simp add: nsphere)
  qed
  have *: reduced_homology_group p (subtopology (powertop_real UNIV) {?a,
?b}) ≅
        homology_group p (subtopology (powertop_real UNIV) {?a})
    apply (rule reduced_homology_group_pair)
      apply (simp_all add: fun_eq_iff)
    apply (simp add: open_fun_def separation_t1 t1_space_def)
    done
  have reduced_homology_group 0 (nsphere 0) ≅ integer_group if p=0
  proof −
   have reduced_homology_group 0 (nsphere 0) ≅ homology_group 0 (top_of_set
{?a}) if p=0
      by (metis * euclidean_product_topology st that)
    also have ... ≅ integer_group
      by (simp add: homology_coefficients)
    finally show ?thesis
      using that by blast
  qed
  moreover have trivial_group (reduced_homology_group p (nsphere 0)) if p≠0
   using * that homology_dimension_axiom [of subtopology (powertop_real UNIV)
{?a} ?a p]
    using isomorphic_group_triviality st by force
  ultimately show ?case
    by auto
next
```

case (*Suc n*)
have *eq*: *reduced_homology_group* (*int n*) (*nsphere n*) ≅ *integer_group* **if** *p−1* = *n*
by (*simp add*: *Suc.IH*)
have *neq*: *trivial_group* (*reduced_homology_group* (*p−1*) (*nsphere n*)) **if** *p−1* ≠ *n*
by (*simp add*: *Suc.IH that*)
have *iso*: *reduced_homology_group p* (*nsphere* (*Suc n*)) ≅ *reduced_homology_group* (*p−1*) (*nsphere n*)
using *reduced_homology_group_nsphere_step* [*of p−1 n*] *group.iso_sym* [*OF _ is_isoI*] *group_reduced_homology_group*
by *fastforce*
then show *?case*
using *eq iso_trans iso isomorphic_group_triviality neq*
by (*metis* (*no_types*, *opaque_lifting*) *add.commute add_left_cancel diff_add_cancel group_reduced_homology_group of_nat_Suc*)
qed

lemma *reduced_homology_group_nsphere*:
reduced_homology_group n (*nsphere n*) ≅ *integer_group*
p ≠ *n* ⟹ *trivial_group*(*reduced_homology_group p* (*nsphere n*))
using *reduced_homology_group_nsphere_aux* **by** *auto*

lemma *cyclic_reduced_homology_group_nsphere*:
cyclic_group(*reduced_homology_group p* (*nsphere n*))
by (*metis reduced_homology_group_nsphere trivial_imp_cyclic_group cyclic_integer_group group_integer_group group_reduced_homology_group isomorphic_group_cyclicity*)

lemma *trivial_reduced_homology_group_nsphere*:
trivial_group(*reduced_homology_group p* (*nsphere n*)) ⟷ (*p* ≠ *n*)
using *group_integer_group isomorphic_group_triviality nontrivial_integer_group reduced_homology_group_nsphere*(*1*) *reduced_homology_group_nsphere*(*2*) *trivial_group_def* **by** *blast*

lemma *non_contractible_space_nsphere*: ¬ (*contractible_space*(*nsphere n*))
proof (*clarsimp simp add*: *contractible_eq_homotopy_equivalent_singleton_subtopology*)
fix *a* :: *nat* ⇒ *real*
assume *a*: *a* ∈ *topspace* (*nsphere n*)
and *he*: *nsphere n homotopy_equivalent_space subtopology* (*nsphere n*) {*a*}
have *trivial_group* (*reduced_homology_group* (*int n*) (*subtopology* (*nsphere n*) {*a*}))
by (*simp add*: *a homology_dimension_reduced* [**where** *a*=*a*])
then show *False*
using *isomorphic_group_triviality* [*OF homotopy_equivalent_space_imp_isomorphic_reduced_homo* [*OF he*, *of n*]]
by (*simp add*: *trivial_reduced_homology_group_nsphere*)
qed

0.3.4 Brouwer degree of a Map

```
definition Brouwer_degree2 :: nat ⇒ ((nat ⇒ real) ⇒ nat ⇒ real) ⇒ int
  where
 Brouwer_degree2 p f ≡
   @d::int. ∀ x ∈ carrier(reduced_homology_group p (nsphere p)).
          hom_induced p (nsphere p) {} (nsphere p) {} f x = pow (reduced_homology_group
p (nsphere p)) x d

lemma Brouwer_degree2_eq:
   (⋀x. x ∈ topspace(nsphere p) ⟹ f x = g x) ⟹ Brouwer_degree2 p f =
Brouwer_degree2 p g
  unfolding Brouwer_degree2_def Ball_def
  apply (intro Eps_cong all_cong)
  by (metis (mono_tags, lifting) hom_induced_eq)

lemma Brouwer_degree2:
  assumes x ∈ carrier(reduced_homology_group p (nsphere p))
  shows hom_induced p (nsphere p) {} (nsphere p) {} f x
     = pow (reduced_homology_group p (nsphere p)) x (Brouwer_degree2 p f)
     (is ?h x = pow ?G x _)
proof (cases continuous_map(nsphere p) (nsphere p) f)
  case True
  interpret group ?G
    by simp
  interpret group_hom ?G ?G ?h
     using hom_induced_reduced_hom group_hom_axioms_def group_hom_def
is_group by blast
  obtain a where a: a ∈ carrier ?G
    and aeq: subgroup_generated ?G {a} = ?G
   using cyclic_reduced_homology_group_nsphere [of p p] by (auto simp: cyclic_group_def)
  then have carra: carrier (subgroup_generated ?G {a}) = range (λn::int. pow
?G a n)
    using carrier_subgroup_generated_by_singleton by blast
  moreover have ?h a ∈ carrier (subgroup_generated ?G {a})
    by (simp add: a aeq hom_induced_reduced)
  ultimately obtain d::int where d: ?h a = pow ?G a d
    by auto
  have *: hom_induced (int p) (nsphere p) {} (nsphere p) {} f x = x [^]?G d
    if x: x ∈ carrier ?G for x
  proof -
    obtain n::int where xeq: x = pow ?G a n
      using carra x aeq by moura
    show ?thesis
      by (simp add: xeq a d hom_int_pow int_pow_pow mult.commute)
  qed
  show ?thesis
    unfolding Brouwer_degree2_def
    apply (rule someI2 [where a=d])
    using assms * apply blast+
```

```
    done
next
  case False
  show ?thesis
    unfolding Brouwer_degree2_def
      by (rule someI2 [where a=0]) (simp_all add: hom_induced_default False
one_reduced_homology_group assms)
qed


lemma Brouwer_degree2_iff:
  assumes f: continuous_map (nsphere p) (nsphere p) f
    and x: x ∈ carrier(reduced_homology_group p (nsphere p))
  shows (hom_induced (int p) (nsphere p) {} (nsphere p) {} f x =
         x [^]reduced_homology_group (int p) (nsphere p) d)
     ⟷ (x = 1reduced_homology_group (int p) (nsphere p) ∨ Brouwer_degree2 p f
= d)
    (is (?h x = x [^]?G d) ⟷ _)
proof -
  interpret group ?G
    by simp
  obtain a where a: a ∈ carrier ?G
    and aeq: subgroup_generated ?G {a} = ?G
   using cyclic_reduced_homology_group_nsphere [of p p] by (auto simp: cyclic_group_def)
  then obtain i::int where i: x = (a [^]?G i)
    using carrier_subgroup_generated_by_singleton x by fastforce
  then have a [^]?G i ∈ carrier ?G
    using x by blast
  have [simp]: ord a = 0
    by (simp add: a aeq iso_finite [OF reduced_homology_group_nsphere(1)] flip:
infinite_cyclic_subgroup_order)
  show ?thesis
   by (auto simp: Brouwer_degree2 int_pow_eq_id x i a int_pow_pow int_pow_eq)
qed


lemma Brouwer_degree2_unique:
  assumes f: continuous_map (nsphere p) (nsphere p) f
    and hi: ⋀x. x ∈ carrier(reduced_homology_group p (nsphere p))
                   ⟹ hom_induced p (nsphere p) {} (nsphere p) {} f x = pow
(reduced_homology_group p (nsphere p)) x d
          (is ⋀x. x ∈ carrier ?G ⟹ ?h x = _)
  shows Brouwer_degree2 p f = d
proof -
  obtain a where a: a ∈ carrier ?G
    and aeq: subgroup_generated ?G {a} = ?G
   using cyclic_reduced_homology_group_nsphere [of p p] by (auto simp: cyclic_group_def)
  show ?thesis
```

```
    using hi [OF a]
    apply (simp add: Brouwer_degree2 a)
   by (metis Brouwer_degree2_iff a aeq f group.trivial_group_subgroup_generated
group_reduced_homology_group subsetI trivial_reduced_homology_group_nsphere)
qed

lemma Brouwer_degree2_unique_generator:
  assumes f: continuous_map (nsphere p) (nsphere p) f
    and eq: subgroup_generated (reduced_homology_group p (nsphere p)) {a}
           = reduced_homology_group p (nsphere p)
   and hi: hom_induced p (nsphere p) {} (nsphere p) {} f a = pow (reduced_homology_group
p (nsphere p)) a d
          (is ?h a = pow ?G a _)
  shows Brouwer_degree2 p f = d
proof (cases a ∈ carrier ?G)
  case True
  then show ?thesis
   by (metis Brouwer_degree2_iff hi eq f group.trivial_group_subgroup_generated
group_reduced_homology_group
             subset_singleton_iff trivial_reduced_homology_group_nsphere)
next
  case False
  then show ?thesis
    using trivial_reduced_homology_group_nsphere [of p p]
     by (metis group.trivial_group_subgroup_generated_eq disjoint_insert(1) eq
group_reduced_homology_group inf_bot_right subset_singleton_iff)
qed

lemma Brouwer_degree2_homotopic:
  assumes homotopic_with (λx. True) (nsphere p) (nsphere p) f g
  shows Brouwer_degree2 p f = Brouwer_degree2 p g
proof –
  have continuous_map (nsphere p) (nsphere p) f
    using homotopic_with_imp_continuous_maps [OF assms] by auto
  show ?thesis
    using Brouwer_degree2_def assms homology_homotopy_empty by fastforce
qed

lemma Brouwer_degree2_id [simp]: Brouwer_degree2 p id = 1
proof (rule Brouwer_degree2_unique)
  fix x
  assume x: x ∈ carrier (reduced_homology_group (int p) (nsphere p))
  then have x ∈ carrier (homology_group (int p) (nsphere p))
    using carrier_reduced_homology_group_subset by blast
  then show hom_induced (int p) (nsphere p) {} (nsphere p) {} id x =
       x [^]reduced_homology_group (int p) (nsphere p) (1::int)
    by (simp add: hom_induced_id group.int_pow_1 x)
qed auto
```

lemma *Brouwer_degree2_compose*:
assumes *f*: *continuous_map* (*nsphere p*) (*nsphere p*) *f* **and** *g*: *continuous_map* (*nsphere p*) (*nsphere p*) *g*
shows *Brouwer_degree2 p* (*g* ∘ *f*) = *Brouwer_degree2 p g* ∗ *Brouwer_degree2 p f*
proof (*rule Brouwer_degree2_unique*)
show *continuous_map* (*nsphere p*) (*nsphere p*) (*g* ∘ *f*)
by (*meson continuous_map_compose f g*)
next
fix *x*
assume *x*: *x* ∈ *carrier* (*reduced_homology_group* (*int p*) (*nsphere p*))
have *hom_induced* (*int p*) (*nsphere p*) {} (*nsphere p*) {} (*g* ∘ *f*) =
hom_induced (*int p*) (*nsphere p*) {} (*nsphere p*) {} *g* ∘
hom_induced (*int p*) (*nsphere p*) {} (*nsphere p*) {} *f*
by (*blast intro*: *hom_induced_compose* [*OF f _ g*])
with *x* **show** *hom_induced* (*int p*) (*nsphere p*) {} (*nsphere p*) {} (*g* ∘ *f*) *x* =
x [^]*reduced_homology_group* (*int p*) (*nsphere p*) (*Brouwer_degree2 p g* ∗ *Brouwer_degree2 p f*)
by (*simp add*: *mult.commute hom_induced_reduced flip*: *Brouwer_degree2 group.int_pow_pow*)
qed

lemma *Brouwer_degree2_homotopy_equivalence*:
assumes *f*: *continuous_map* (*nsphere p*) (*nsphere p*) *f* **and** *g*: *continuous_map* (*nsphere p*) (*nsphere p*) *g*
and *hom*: *homotopic_with* (λ*x*. *True*) (*nsphere p*) (*nsphere p*) (*f* ∘ *g*) *id*
obtains |*Brouwer_degree2 p f*| = *1* |*Brouwer_degree2 p g*| = *1* *Brouwer_degree2 p g* = *Brouwer_degree2 p f*
using *Brouwer_degree2_homotopic* [*OF hom*] *Brouwer_degree2_compose f g zmult_eq_1_iff* **by** *auto*

lemma *Brouwer_degree2_homeomorphic_maps*:
assumes *homeomorphic_maps* (*nsphere p*) (*nsphere p*) *f g*
obtains |*Brouwer_degree2 p f*| = *1* |*Brouwer_degree2 p g*| = *1* *Brouwer_degree2 p g* = *Brouwer_degree2 p f*
using *assms*
by (*auto simp*: *homeomorphic_maps_def homotopic_with_equal continuous_map_compose intro*: *Brouwer_degree2_homotopy_equivalence*)

lemma *Brouwer_degree2_retraction_map*:
assumes *retraction_map* (*nsphere p*) (*nsphere p*) *f*
shows |*Brouwer_degree2 p f*| = *1*
proof −
obtain *g* **where** *g*: *retraction_maps* (*nsphere p*) (*nsphere p*) *f g*
using *assms* **by** (*auto simp*: *retraction_map_def*)
show *?thesis*
proof (*rule Brouwer_degree2_homotopy_equivalence*)
show *homotopic_with* (λ*x*. *True*) (*nsphere p*) (*nsphere p*) (*f* ∘ *g*) *id*

```
      using g apply (auto simp: retraction_maps_def)
      by (simp add: homotopic_with_equal continuous_map_compose)
    show continuous_map (nsphere p) (nsphere p) f continuous_map (nsphere p)
(nsphere p) g
      using g retraction_maps_def by blast+
  qed
qed

lemma Brouwer_degree2_section_map:
  assumes section_map (nsphere p) (nsphere p) f
  shows |Brouwer_degree2 p f| = 1
proof –
  obtain g where g: retraction_maps (nsphere p) (nsphere p) g f
    using assms by (auto simp: section_map_def)
  show ?thesis
  proof (rule Brouwer_degree2_homotopy_equivalence)
    show homotopic_with (λx. True) (nsphere p) (nsphere p) (g ∘ f) id
      using g apply (auto simp: retraction_maps_def)
      by (simp add: homotopic_with_equal continuous_map_compose)
    show continuous_map (nsphere p) (nsphere p) g continuous_map (nsphere p)
(nsphere p) f
      using g retraction_maps_def by blast+
  qed
qed

lemma Brouwer_degree2_homeomorphic_map:
   homeomorphic_map (nsphere p) (nsphere p) f ⟹ |Brouwer_degree2 p f| = 1
 using Brouwer_degree2_retraction_map section_and_retraction_eq_homeomorphic_map
by blast


lemma Brouwer_degree2_nullhomotopic:
  assumes homotopic_with (λx. True) (nsphere p) (nsphere p) f (λx. a)
  shows Brouwer_degree2 p f = 0
proof –
  have contf: continuous_map (nsphere p) (nsphere p) f
   and contc: continuous_map (nsphere p) (nsphere p) (λx. a)
    using homotopic_with_imp_continuous_maps [OF assms] by metis+
  have Brouwer_degree2 p f = Brouwer_degree2 p (λx. a)
    using Brouwer_degree2_homotopic [OF assms] .
  moreover
  let ?G = reduced_homology_group (int p) (nsphere p)
  interpret group ?G
    by simp
  have Brouwer_degree2 p (λx. a) = 0
  proof (rule Brouwer_degree2_unique [OF contc])
    fix c
    assume c: c ∈ carrier ?G
    have continuous_map (nsphere p) (subtopology (nsphere p) {a}) (λf. a)
```

using *contc continuous_map_in_subtopology* **by** *blast*
then have *he*: *hom_induced p* (*nsphere p*) {} (*nsphere p*) {} (λ*x*. *a*)
= *hom_induced p* (*subtopology* (*nsphere p*) {*a*}) {} (*nsphere p*) {} *id*
∘
hom_induced p (*nsphere p*) {} (*subtopology* (*nsphere p*) {*a*}) {}
(λ*x*. *a*)
by (*metis continuous_map_id_subt hom_induced_compose id_comp image_empty order_refl*)
have *1*: *hom_induced p* (*nsphere p*) {} (*subtopology* (*nsphere p*) {*a*}) {} (λ*x*. *a*) *c* =
$\mathbf{1}_{reduced\_homology\_group\ (int\ p)\ (subtopology\ (nsphere\ p)\ \{a\})}$
using *c trivial_reduced_homology_group_contractible_space* [*of subtopology* (*nsphere p*) {*a*} *p*]
by (*simp add*: *hom_induced_reduced contractible_space_subtopology_singleton trivial_group_subset group.trivial_group_subset subset_iff*)
show *hom_induced* (*int p*) (*nsphere p*) {} (*nsphere p*) {} (λ*x*. *a*) *c* =
c $[\urcorner]_{?G}$ (*0*::*int*)
apply (*simp add*: *he 1*)
using *hom_induced_reduced_hom group_hom.hom_one group_hom_axioms_def group_hom_def group_reduced_homology_group* **by** *blast*
qed
ultimately show *?thesis*
by *metis*
qed

lemma *Brouwer_degree2_const*: *Brouwer_degree2 p* (λ*x*. *a*) = *0*
proof (*cases continuous_map*(*nsphere p*) (*nsphere p*) (λ*x*. *a*))
case *True*
then show *?thesis*
by (*auto intro*: *Brouwer_degree2_nullhomotopic* [**where** *a*=*a*])
next
case *False*
let *?G* = *reduced_homology_group* (*int p*) (*nsphere p*)
let *?H* = *homology_group* (*int p*) (*nsphere p*)
interpret *group ?G*
by *simp*
have *eq1*: $\mathbf{1}_{?H} = \mathbf{1}_{?G}$
by (*simp add*: *one_reduced_homology_group*)
have ∗: ∀ *x*∈*carrier ?G*. *hom_induced* (*int p*) (*nsphere p*) {} (*nsphere p*) {} (λ*x*. *a*) *x* = $\mathbf{1}_{?H}$
by (*metis False hom_induced_default one_relative_homology_group*)
obtain *c* **where** *c*: *c* ∈ *carrier ?G* **and** *ceq*: *subgroup_generated ?G* {*c*} = *?G*
using *cyclic_reduced_homology_group_nsphere* [*of p p*] **by** (*force simp*: *cyclic_group_def*)
have [*simp*]: *ord c* = *0*
by (*simp add*: *c ceq iso_finite* [*OF reduced_homology_group_nsphere*(*1*)] *flip*: *infinite_cyclic_subgroup_order*)
show *?thesis*
unfolding *Brouwer_degree2_def*

```
  proof (rule some_equality)
    fix d :: int
     assume ∀x∈carrier ?G. hom_induced (int p) (nsphere p) {} (nsphere p) {}
(λx. a) x = x [^]?G d
    then have c [^]?G d = 1?H
      using * c by blast
    then have int (ord c) dvd d
      using c eq1 int_pow_eq_id by auto
    then show d = 0
      by (simp add: * del: one_relative_homology_group)
  qed (use * eq1 in force)
qed


corollary Brouwer_degree2_nonsurjective:
   ⟦continuous_map(nsphere p) (nsphere p) f; f ` topspace (nsphere p) ≠ topspace
(nsphere p)⟧
    ⟹ Brouwer_degree2 p f = 0
  by (meson Brouwer_degree2_nullhomotopic nullhomotopic_nonsurjective_sphere_map)


proposition Brouwer_degree2_reflection:
  Brouwer_degree2 p (λx i. if i = 0 then −x i else x i) = −1 (is Brouwer_degree2
_ ?r = −1)
proof (induction p)
  case 0
  let ?G = homology_group 0 (nsphere 0)
  let ?D = homology_group 0 (discrete_topology {()})
  interpret group ?G
    by simp
  define r where r ≡ λx::nat⇒real. λi. if i = 0 then −x i else x i
  then have [simp]: r ∘ r = id
    by force
  have cmr: continuous_map (nsphere 0) (nsphere 0) r
    by (simp add: r_def continuous_map_nsphere_reflection)
  have *: hom_induced 0 (nsphere 0) {} (nsphere 0) {} r c = inv?G c
    if c ∈ carrier(reduced_homology_group 0 (nsphere 0)) for c
  proof −
    have c: c ∈ carrier ?G
      and ceq: hom_induced 0 (nsphere 0) {} (discrete_topology {()}) {} (λx. ())
c = 1?D
      using that by (auto simp: carrier_reduced_homology_group kernel_def)
    define pp::nat⇒real where pp ≡ λi. if i = 0 then 1 else 0
    define nn::nat⇒real where nn ≡ λi. if i = 0 then −1 else 0
    have topn0: topspace(nsphere 0) = {pp,nn}
       by (auto simp: nsphere pp_def nn_def fun_eq_iff power2_eq_1_iff split:
if_split_asm)
    have t1_space (nsphere 0)
      unfolding nsphere
```

apply (*rule t1_space_subtopology*)
by (*metis* (*full_types*) *open_fun_def t1_space t1_space_def*)
then have *dtn0*: *discrete_topology* {*pp*,*nn*} = *nsphere 0*
using *finite_t1_space_imp_discrete_topology* [*OF topn0*] **by** *auto*
have *pp* ≠ *nn*
by (*auto simp*: *pp_def nn_def fun_eq_iff*)
have [*simp*]: *r pp* = *nn r nn* = *pp*
by (*auto simp*: *r_def pp_def nn_def fun_eq_iff*)
have *iso*: (λ(*a*,*b*). *hom_induced 0* (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} *id a*
⊗$_{?G}$ *hom_induced 0* (*subtopology* (*nsphere 0*) {*nn*}) {} (*nsphere 0*) {} *id b*)
∈ *iso* (*homology_group 0* (*subtopology* (*nsphere 0*) {*pp*}) ×× *homology_group 0* (*subtopology* (*nsphere 0*) {*nn*}))
?G (**is** *?f* ∈ *iso* (*?P* ×× *?N*) *?G*)
apply (*rule homology_additivity_explicit*)
using *dtn0* ‹*pp* ≠ *nn*› **by** (*auto simp*: *discrete_topology_unique*)
then have *fim*: *?f* ' *carrier*(*?P* ×× *?N*) = *carrier ?G*
by (*simp add*: *iso_def bij_betw_def*)
obtain *d d'* **where** *d*: *d* ∈ *carrier ?P* **and** *d'*: *d'* ∈ *carrier ?N* **and** *eqc*: *?f*(*d*,*d'*) = *c*
using *c* **by** (*force simp flip*: *fim*)
let *?h* = λ*xx*. *hom_induced 0* (*subtopology* (*nsphere 0*) {*xx*}) {} (*discrete_topology* {()}) {} (λ*x*. ())
have *retraction_map* (*subtopology* (*nsphere 0*) {*pp*}) (*subtopology* (*nsphere 0*) {*nn*}) *r*
apply (*simp add*: *retraction_map_def retraction_maps_def continuous_map_in_subtopology continuous_map_from_subtopology cmr image_subset_iff*)
apply (*rule_tac x*=*r* **in** *exI*)
apply (*force simp*: *retraction_map_def retraction_maps_def continuous_map_in_subtopology continuous_map_from_subtopology cmr*)
done
then have *carrier ?N* = (*hom_induced 0* (*subtopology* (*nsphere 0*) {*pp*}) {} (*subtopology* (*nsphere 0*) {*nn*}) {} *r*) ' *carrier ?P*
by (*rule surj_hom_induced_retraction_map*)
then obtain *e* **where** *e*: *e* ∈ *carrier ?P* **and** *eqd'*: *hom_induced 0* (*subtopology* (*nsphere 0*) {*pp*}) {} (*subtopology* (*nsphere 0*) {*nn*}) {} *r e* = *d'*
using *d'* **by** *auto*
have *section_map* (*subtopology* (*nsphere 0*) {*pp*}) (*discrete_topology* {()}) (λ*x*. ())
by (*force simp*: *section_map_def retraction_maps_def topn0*)
then have *?h pp* ∈ *mon ?P ?D*
by (*rule mon_hom_induced_section_map*)
then have *one*: *x* = *one ?P*
if *?h pp x* = $\mathbf{1}_{?D}$ *x* ∈ *carrier ?P* **for** *x*
using *that* **by** (*simp add*: *mon_iff_hom_one*)
interpret *hpd*: *group_hom ?P ?D ?h pp*
using *hom_induced_empty_hom* **by** (*simp add*: *hom_induced_empty_hom group_hom_axioms_def group_hom_def*)

interpret *hgd*: *group_hom ?G ?D hom_induced 0* (*nsphere 0*) {} (*discrete_topology* {()}) {} (λ*x*. ())
using *hom_induced_empty_hom* **by** (*simp add*: *hom_induced_empty_hom group_hom_axioms_def group_hom_def*)
interpret *hpg*: *group_hom ?P ?G hom_induced 0* (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} *r*
using *hom_induced_empty_hom* **by** (*simp add*: *hom_induced_empty_hom group_hom_axioms_def group_hom_def*)
interpret *hgg*: *group_hom ?G ?G hom_induced 0* (*nsphere 0*) {} (*nsphere 0*) {} *r*
using *hom_induced_empty_hom* **by** (*simp add*: *hom_induced_empty_hom group_hom_axioms_def group_hom_def*)
have *?h pp d* =
(*hom_induced 0* (*nsphere 0*) {} (*discrete_topology* {()}) {} (λ*x*. ())
∘ *hom_induced 0* (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} *id*) *d*
by (*simp flip*: *hom_induced_compose_empty*)
moreover
have *?h pp* = *?h nn* ∘ *hom_induced 0* (*subtopology* (*nsphere 0*) {*pp*}) {} (*subtopology* (*nsphere 0*) {*nn*}) {} *r*
by (*simp add*: *cmr continuous_map_from_subtopology continuous_map_in_subtopology image_subset_iff flip*: *hom_induced_compose_empty*)
then have *?h pp e* =
(*hom_induced 0* (*nsphere 0*) {} (*discrete_topology* {()}) {} (λ*x*. ())
∘ *hom_induced 0* (*subtopology* (*nsphere 0*) {*nn*}) {} (*nsphere 0*) {} *id*) *d'*
by (*simp flip*: *hom_induced_compose_empty eqd'*)
ultimately have *?h pp* ($d \otimes_{?P} e$) = *hom_induced 0* (*nsphere 0*) {} (*discrete_topology* {()}) {} (λ*x*. ()) (*?f*(*d*,*d'*))
by (*simp add*: *d e hom_induced_carrier*)
then have *?h pp* ($d \otimes_{?P} e$) = $\mathbf{1}_{?D}$
using *ceq eqc* **by** *simp*
then have *inv_p*: $inv_{?P}$ *d* = *e*
by (*metis* (*no_types*, *lifting*) *Group.group_def d e group.inv_equality group.r_inv group_relative_homology_group one monoid.m_closed*)
have *cmr_pn*: *continuous_map* (*subtopology* (*nsphere 0*) {*pp*}) (*subtopology* (*nsphere 0*) {*nn*}) *r*
by (*simp add*: *cmr continuous_map_from_subtopology continuous_map_in_subtopology image_subset_iff*)
then have *hom_induced 0* (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} (*id* ∘ *r*) =
hom_induced 0 (*subtopology* (*nsphere 0*) {*nn*}) {} (*nsphere 0*) {} *id* ∘
hom_induced 0 (*subtopology* (*nsphere 0*) {*pp*}) {} (*subtopology* (*nsphere 0*) {*nn*}) {} *r*
using *hom_induced_compose_empty continuous_map_id_subt* **by** *blast*
then have $inv_{?G}$ *hom_induced 0* (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} *r d* =
hom_induced 0 (*subtopology* (*nsphere 0*) {*nn*}) {} (*nsphere 0*) {} *id d'*
apply (*simp add*: *flip*: *inv_p eqd'*)

using *d hpg.hom_inv* **by** *auto*
then have *c*: *c* = (*hom_induced 0* (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} *id d*)
⊗$_{?G}$ *inv*$_{?G}$ (*hom_induced 0* (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} *r d*)
by (*simp flip*: *eqc*)
have *hom_induced 0* (*nsphere 0*) {} (*nsphere 0*) {} *r* ∘
hom_induced 0 (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} *id* =
hom_induced 0 (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} *r*
by (*metis cmr comp_id continuous_map_id_subt hom_induced_compose_empty*)
moreover
have *hom_induced 0* (*nsphere 0*) {} (*nsphere 0*) {} *r* ∘
hom_induced 0 (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} *r* =
hom_induced 0 (*subtopology* (*nsphere 0*) {*pp*}) {} (*nsphere 0*) {} *id*
by (*metis* ‹*r* ∘ *r* = *id*› *cmr continuous_map_from_subtopology hom_induced_compose_empty*)
ultimately show *?thesis*
by (*metis inv_p c comp_def d e hgg.hom_inv hgg.hom_mult hom_induced_carrier hpd.G.inv_inv hpg.hom_inv inv_mult_group*)
qed
show *?case*
unfolding *r_def* [*symmetric*]
using *Brouwer_degree2_unique* [*OF cmr*]
by (*auto simp*: ∗ *group.int_pow_neg group.int_pow_1 reduced_homology_group_def intro*!: *Brouwer_degree2_unique* [*OF cmr*])
next
case (*Suc p*)
let *?G* = *reduced_homology_group* (*int p*) (*nsphere p*)
let *?G1* = *reduced_homology_group* (*1* + *int p*) (*nsphere* (*Suc p*))
obtain *f g* **where** *fg*: *group_isomorphisms ?G ?G1 f g*
and ∗: ∀ *c*∈*carrier ?G*.
hom_induced (*1* + *int p*) (*nsphere* (*Suc p*)) {} (*nsphere* (*Suc p*)) {} *?r* (*f c*) =
f (*hom_induced p* (*nsphere p*) {} (*nsphere p*) {} *?r c*)
using *reduced_homology_group_nsphere_step*
by (*meson group.iso_iff_group_isomorphisms group_reduced_homology_group*)
then have *eq*: *carrier ?G1* = *f* ‘ *carrier ?G*
by (*fastforce simp add*: *iso_iff dest*: *group_isomorphisms_imp_iso*)
interpret *group_hom ?G ?G1 f*
by (*meson fg group_hom_axioms_def group_hom_def group_isomorphisms_def group_reduced_homology_group*)
have *homf*: *f* ∈ *hom ?G ?G1*
using *fg group_isomorphisms_def* **by** *blast*
have *hom_induced* (*1* + *int p*) (*nsphere* (*Suc p*)) {} (*nsphere* (*Suc p*)) {} *?r* (*f y*) = *f y* [^]$_{?G1}$ (−*1*::*int*)
if *y* ∈ *carrier ?G* **for** *y*
by (*simp add*: *that* ∗ *Brouwer_degree2 Suc hom_int_pow*)
then show *?case*
by (*fastforce simp*: *eq intro*: *Brouwer_degree2_unique* [*OF continuous_map_nsphere_reflection*])
qed

end

0.4 Invariance of Domain

theory *Invariance_of_Domain*
imports *Brouwer_Degree HOL−Analysis.Continuous_Extension HOL−Analysis.Homeomorphism*

begin

0.4.1 Degree invariance mod 2 for map between pairs

```
theorem Borsuk_odd_mapping_degree_step:
  assumes cmf: continuous_map (nsphere n) (nsphere n) f
    and f: ⋀x. x ∈ topspace(nsphere n) ⟹ (f ∘ (λx i. −x i)) x = ((λx i. −x i) ∘
f) x
    and fim: f ‘ (topspace(nsphere(n − Suc 0))) ⊆ topspace(nsphere(n − Suc 0))
  shows even (Brouwer_degree2 n f − Brouwer_degree2 (n − Suc 0) f)
proof (cases n = 0)
  case False
  define neg where neg ≡ λx::nat⇒real. λi. −x i
  define upper where upper ≡ λn. {x::nat⇒real. x n ≥ 0}
  define lower where lower ≡ λn. {x::nat⇒real. x n ≤ 0}
  define equator where equator ≡ λn. {x::nat⇒real. x n = 0}
  define usphere where usphere ≡ λn. subtopology (nsphere n) (upper n)
  define lsphere where lsphere ≡ λn. subtopology (nsphere n) (lower n)
  have [simp]: neg x i = −x i for x i
    by (force simp: neg_def)
  have equator_upper: equator n ⊆ upper n
    by (force simp: equator_def upper_def)
  have upper_usphere: subtopology (nsphere n) (upper n) = usphere n
    by (simp add: usphere_def)
  let ?rhgn = relative_homology_group n (nsphere n)
  let ?hi_ee = hom_induced n (nsphere n) (equator n) (nsphere n) (equator n)
  interpret GE: comm_group ?rhgn (equator n)
    by simp
  interpret HB: group_hom ?rhgn (equator n)
                      homology_group (int n − 1) (subtopology (nsphere n) (equator
n))
                       hom_boundary n (nsphere n) (equator n)
    by (simp add: group_hom_axioms_def group_hom_def hom_boundary_hom)
  interpret HIU: group_hom ?rhgn (equator n)
                        ?rhgn (upper n)
                         hom_induced n (nsphere n) (equator n) (nsphere n) (upper
n) id
    by (simp add: group_hom_axioms_def group_hom_def hom_induced_hom)
  have subt_eq: subtopology (nsphere n) {x. x n = 0} = nsphere (n − Suc 0)
    by (metis False Suc_pred le_zero_eq not_le subtopology_nsphere_equator)
  then have equ: subtopology (nsphere n) (equator n) = nsphere(n − Suc 0)
```

subtopology (*lsphere n*) (*equator n*) = *nsphere*(*n* − *Suc 0*)
subtopology (*usphere n*) (*equator n*) = *nsphere*(*n* − *Suc 0*)
using *False* **by** (*auto simp*: *lsphere_def usphere_def equator_def lower_def upper_def subtopology_subtopology simp flip*: *Collect_conj_eq cong*: *rev_conj_cong*)
have *cmr*: *continuous_map* (*nsphere*(*n* − *Suc 0*)) (*nsphere*(*n* − *Suc 0*)) *f*
by (*metis* (*no_types*, *lifting*) *IntE cmf fim continuous_map_from_subtopology continuous_map_in_subtopology equ*(*1*) *image_subset_iff topspace_subtopology*)

have *f x n* = *0* **if** *x* ∈ *topspace* (*nsphere n*) *x n* = *0* **for** *x*
proof −
have *x* ∈ *topspace* (*nsphere* (*n* − *Suc 0*))
by (*simp add*: *that topspace_nsphere_minus1*)
moreover have *topspace* (*nsphere n*) ∩ {*f*. *f n* = *0*} = *topspace* (*nsphere* (*n* − *Suc 0*))
by (*metis subt_eq topspace_subtopology*)
ultimately show *?thesis*
using *fim* **by** *auto*
qed
then have *fimeq*: *f* ‘ (*topspace* (*nsphere n*) ∩ *equator n*) ⊆ *topspace* (*nsphere n*) ∩ *equator n*
using *fim cmf* **by** (*auto simp*: *equator_def continuous_map_def image_subset_iff*)
have ⋀*k*. *continuous_map* (*powertop_real UNIV*) *euclideanreal* (λ*x*. − *x k*)
by (*metis UNIV_I continuous_map_product_projection continuous_map_minus*)
then have *cm_neg*: *continuous_map* (*nsphere m*) (*nsphere m*) *neg* **for** *m*
by (*force simp*: *nsphere continuous_map_in_subtopology neg_def continuous_map_componentwise_UNIV intro*: *continuous_map_from_subtopology*)
then have *cm_neg_lu*: *continuous_map* (*lsphere n*) (*usphere n*) *neg*
by (*auto simp*: *lsphere_def usphere_def lower_def upper_def continuous_map_from_subtopology continuous_map_in_subtopology*)
have *neg_in_top_iff*: *neg x* ∈ *topspace*(*nsphere m*) ⟷ *x* ∈ *topspace*(*nsphere m*) **for** *m x*
by (*simp add*: *nsphere_def neg_def topspace_Euclidean_space*)
obtain *z* **where** *zcarr*: *z* ∈ *carrier* (*reduced_homology_group* (*int n* − *1*) (*nsphere* (*n* − *Suc 0*)))
and *zeq*: *subgroup_generated* (*reduced_homology_group* (*int n* − *1*) (*nsphere* (*n* − *Suc 0*))) {*z*}
= *reduced_homology_group* (*int n* − *1*) (*nsphere* (*n* − *Suc 0*))
using *cyclic_reduced_homology_group_nsphere* [*of int n* − *1 n* − *Suc 0*] **by** (*auto simp*: *cyclic_group_def*)
have *hom_boundary n* (*subtopology* (*nsphere n*) {*x*. *x n* ≤ *0*}) {*x*. *x n* = *0*}
∈ *Group.iso* (*relative_homology_group n*
(*subtopology* (*nsphere n*) {*x*. *x n* ≤ *0*}) {*x*. *x n* = *0*})
(*reduced_homology_group* (*int n* − *1*) (*nsphere* (*n* − *Suc 0*)))
using *iso_lower_hemisphere_reduced_homology_group* [*of int n* − *1 n* − *Suc 0*] *False* **by** *simp*
then obtain *gp* **where** *g*: *group_isomorphisms*
(*relative_homology_group n* (*subtopology* (*nsphere n*) {*x*. *x n* ≤ *0*}) {*x*. *x n* = *0*})
(*reduced_homology_group* (*int n* − *1*) (*nsphere* (*n* − *Suc 0*)))

```
                  (hom_boundary n (subtopology (nsphere n) {x. x n ≤ 0}) {x.
x n = 0})
                  gp
   by (auto simp: group.iso_iff_group_isomorphisms)
  then interpret gp: group_hom reduced_homology_group (int n − 1) (nsphere
(n − Suc 0))
    relative_homology_group n (subtopology (nsphere n) {x. x n ≤ 0}) {x. x n =
0} gp
   by (simp add: group_hom_axioms_def group_hom_def group_isomorphisms_def)
  obtain zp where zpcarr: zp ∈ carrier(relative_homology_group n (lsphere n)
(equator n))
    and zp_z: hom_boundary n (lsphere n) (equator n) zp = z
   and zp_sg: subgroup_generated (relative_homology_group n (lsphere n) (equator
n)) {zp}
          = relative_homology_group n (lsphere n) (equator n)
  proof
    show gp z ∈ carrier (relative_homology_group n (lsphere n) (equator n))
      hom_boundary n (lsphere n) (equator n) (gp z) = z
    using g zcarr by (auto simp: lsphere_def equator_def lower_def group_isomorphisms_def)
    have giso: gp ∈ Group.iso (reduced_homology_group (int n − 1) (nsphere (n
− Suc 0)))
                (relative_homology_group n (subtopology (nsphere n) {x. x n ≤
0}) {x. x n = 0})
    by (metis (mono_tags, lifting) g group_isomorphisms_imp_iso group_isomorphisms_sym)
   show subgroup_generated (relative_homology_group n (lsphere n) (equator n))
{gp z} =
        relative_homology_group n (lsphere n) (equator n)
      apply (rule monoid.equality)
      using giso gp.subgroup_generated_by_image [of {z}] zcarr
      by (auto simp: lsphere_def equator_def lower_def zeq gp.iso_iff)
  qed
  have hb_iso: hom_boundary n (subtopology (nsphere n) {x. x n ≥ 0}) {x. x n
= 0}
          ∈ iso (relative_homology_group n (subtopology (nsphere n) {x. x n ≥
0}) {x. x n = 0})
                (reduced_homology_group (int n − 1) (nsphere (n − Suc 0)))
    using iso_upper_hemisphere_reduced_homology_group [of int n − 1 n − Suc
0] False by simp
  then obtain gn where g: group_isomorphisms
                  (relative_homology_group n (subtopology (nsphere n) {x. x n
≥ 0}) {x. x n = 0})
                  (reduced_homology_group (int n − 1) (nsphere (n − Suc 0)))
                  (hom_boundary n (subtopology (nsphere n) {x. x n ≥ 0}) {x.
x n = 0})
                  gn
   by (auto simp: group.iso_iff_group_isomorphisms)
  then interpret gn: group_hom reduced_homology_group (int n − 1) (nsphere
(n − Suc 0))
    relative_homology_group n (subtopology (nsphere n) {x. x n ≥ 0}) {x. x n =
```

0} *gn*
by (*simp add*: *group_hom_axioms_def group_hom_def group_isomorphisms_def*)
obtain *zn* **where** *zncarr*: *zn* ∈ *carrier*(*relative_homology_group n* (*usphere n*) (*equator n*))
and *zn_z*: *hom_boundary n* (*usphere n*) (*equator n*) *zn* = *z*
and *zn_sg*: *subgroup_generated* (*relative_homology_group n* (*usphere n*) (*equator n*)) {*zn*}
= *relative_homology_group n* (*usphere n*) (*equator n*)
proof
show *gn z* ∈ *carrier* (*relative_homology_group n* (*usphere n*) (*equator n*))
hom_boundary n (*usphere n*) (*equator n*) (*gn z*) = *z*
using *g zcarr* **by** (*auto simp*: *usphere_def equator_def upper_def group_isomorphisms_def*)
have *giso*: *gn* ∈ *Group.iso* (*reduced_homology_group* (*int n* − *1*) (*nsphere* (*n* − *Suc 0*)))
(*relative_homology_group n* (*subtopology* (*nsphere n*) {*x*. *x n* ≥ *0*}) {*x*. *x n* = *0*})
by (*metis* (*mono_tags*, *lifting*) *g group_isomorphisms_imp_iso group_isomorphisms_sym*)
show *subgroup_generated* (*relative_homology_group n* (*usphere n*) (*equator n*)) {*gn z*} =
relative_homology_group n (*usphere n*) (*equator n*)
apply (*rule monoid.equality*)
using *giso gn.subgroup_generated_by_image* [*of* {*z*}] *zcarr*
by (*auto simp*: *usphere_def equator_def upper_def zeq gn.iso_iff*)
qed
let *?hi_lu* = *hom_induced n* (*lsphere n*) (*equator n*) (*nsphere n*) (*upper n*) *id*
interpret *gh_lu*: *group_hom relative_homology_group n* (*lsphere n*) (*equator n*) *?rhgn* (*upper n*) *?hi_lu*
by (*simp add*: *group_hom_axioms_def group_hom_def hom_induced_hom*)
interpret *gh_eef*: *group_hom ?rhgn* (*equator n*) *?rhgn* (*equator n*) *?hi_ee f*
by (*simp add*: *group_hom_axioms_def group_hom_def hom_induced_hom*)
define *wp* **where** *wp* ≡ *?hi_lu zp*
then have *wpcarr*: *wp* ∈ *carrier*(*?rhgn* (*upper n*))
by (*simp add*: *hom_induced_carrier*)
have *hom_induced n* (*nsphere n*) {} (*nsphere n*) {*x*. *x n* ≥ *0*} *id*
∈ *iso* (*reduced_homology_group n* (*nsphere n*))
(*?rhgn* {*x*. *x n* ≥ *0*})
using *iso_reduced_homology_group_upper_hemisphere* [*of n n n*] **by** *auto*
then have *carrier*(*?rhgn* {*x*. *x n* ≥ *0*})
⊆ (*hom_induced n* (*nsphere n*) {} (*nsphere n*) {*x*. *x n* ≥ *0*} *id*)
‘ *carrier*(*reduced_homology_group n* (*nsphere n*))
by (*simp add*: *iso_iff*)
then obtain *vp* **where** *vpcarr*: *vp* ∈ *carrier*(*reduced_homology_group n* (*nsphere n*))
and *eqwp*: *hom_induced n* (*nsphere n*) {} (*nsphere n*) (*upper n*) *id vp* = *wp*
using *wpcarr* **by** (*auto simp*: *upper_def*)
define *wn* **where** *wn* ≡ *hom_induced n* (*usphere n*) (*equator n*) (*nsphere n*) (*lower n*) *id zn*
then have *wncarr*: *wn* ∈ *carrier*(*?rhgn* (*lower n*))
by (*simp add*: *hom_induced_carrier*)

```
  have hom_induced n (nsphere n) {} (nsphere n) {x. x n ≤ 0} id
      ∈ iso (reduced_homology_group n (nsphere n))
          (?rhgn {x. x n ≤ 0})
    using iso_reduced_homology_group_lower_hemisphere [of n n n] by auto
  then have carrier(?rhgn {x. x n ≤ 0})
          ⊆ (hom_induced n (nsphere n) {} (nsphere n) {x. x n ≤ 0} id)
                       ` carrier(reduced_homology_group n (nsphere n))
    by (simp add: iso_iff)
  then obtain vn where vpcarr: vn ∈ carrier(reduced_homology_group n (nsphere
n))
    and eqwp: hom_induced n (nsphere n) {} (nsphere n) (lower n) id vn = wn
    using wncarr by (auto simp: lower_def)
   define up where up ≡ hom_induced n (lsphere n) (equator n) (nsphere n)
(equator n) id zp
  then have upcarr: up ∈ carrier(?rhgn (equator n))
    by (simp add: hom_induced_carrier)
   define un where un ≡ hom_induced n (usphere n) (equator n) (nsphere n)
(equator n) id zn
  then have uncarr: un ∈ carrier(?rhgn (equator n))
    by (simp add: hom_induced_carrier)
  have *: (λ(x, y).
          hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id x
        ⊗?rhgn (equator n)
          hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id y)
      ∈ Group.iso
          (relative_homology_group n (lsphere n) (equator n) ××
           relative_homology_group n (usphere n) (equator n))
          (?rhgn (equator n))
  proof (rule conjunct1 [OF exact_sequence_sum_lemma [OF abelian_relative_homology_group]])
    show hom_induced n (lsphere n) (equator n) (nsphere n) (upper n) id
      ∈ Group.iso (relative_homology_group n (lsphere n) (equator n))
          (?rhgn (upper n))
      apply (simp add: lsphere_def usphere_def equator_def lower_def upper_def)
      using iso_relative_homology_group_lower_hemisphere by blast
    show hom_induced n (usphere n) (equator n) (nsphere n) (lower n) id
      ∈ Group.iso (relative_homology_group n (usphere n) (equator n))
          (?rhgn (lower n))
       apply (simp_all add: lsphere_def usphere_def equator_def lower_def up-
per_def)
      using iso_relative_homology_group_upper_hemisphere by blast+
    show exact_seq
        ([?rhgn (lower n),
          ?rhgn (equator n),
          relative_homology_group n (lsphere n) (equator n)],
         [hom_induced n (nsphere n) (equator n) (nsphere n) (lower n) id,
          hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id])
      unfolding lsphere_def usphere_def equator_def lower_def upper_def
      by (rule homology_exactness_triple_3) force
    show exact_seq
```

([*?rhgn* (*upper n*),
?rhgn (*equator n*),
relative_homology_group n (*usphere n*) (*equator n*)],
[*hom_induced n* (*nsphere n*) (*equator n*) (*nsphere n*) (*upper n*) *id*,
hom_induced n (*usphere n*) (*equator n*) (*nsphere n*) (*equator n*) *id*])
unfolding *lsphere_def usphere_def equator_def lower_def upper_def*
by (*rule homology_exactness_triple_3*) *force*
next
fix *x*
assume $x \in$ *carrier* (*relative_homology_group n* (*lsphere n*) (*equator n*))
show *hom_induced n* (*nsphere n*) (*equator n*) (*nsphere n*) (*upper n*) *id*
(*hom_induced n* (*lsphere n*) (*equator n*) (*nsphere n*) (*equator n*) *id x*) =
hom_induced n (*lsphere n*) (*equator n*) (*nsphere n*) (*upper n*) *id x*
by (*simp add*: *hom_induced_compose′ subset_iff lsphere_def usphere_def equator_def lower_def upper_def*)
next
fix *x*
assume $x \in$ *carrier* (*relative_homology_group n* (*usphere n*) (*equator n*))
show *hom_induced n* (*nsphere n*) (*equator n*) (*nsphere n*) (*lower n*) *id*
(*hom_induced n* (*usphere n*) (*equator n*) (*nsphere n*) (*equator n*) *id x*) =
hom_induced n (*usphere n*) (*equator n*) (*nsphere n*) (*lower n*) *id x*
by (*simp add*: *hom_induced_compose′ subset_iff lsphere_def usphere_def equator_def lower_def upper_def*)
qed
then have *sb*: *carrier* (*?rhgn* (*equator n*))
$\subseteq$ (λ(*x*, *y*).
hom_induced n (*lsphere n*) (*equator n*) (*nsphere n*) (*equator n*) *id x*
$\otimes_{?rhgn\ (equator\ n)}$
hom_induced n (*usphere n*) (*equator n*) (*nsphere n*) (*equator n*) *id y*)
‘ *carrier* (*relative_homology_group n* (*lsphere n*) (*equator n*) ××
relative_homology_group n (*usphere n*) (*equator n*))
by (*simp add*: *iso_iff*)
obtain *a b*::*int*
where *up_ab*: *?hi_ee f up*
= *up* $[\urcorner]_{?rhgn\ (equator\ n)}$ $a \otimes_{?rhgn\ (equator\ n)}$ *un* $[\urcorner]_{?rhgn\ (equator\ n)}$ b
proof −
have *hiupcarr*: *?hi_ee f up* $\in$ *carrier*(*?rhgn* (*equator n*))
by (*simp add*: *hom_induced_carrier*)
obtain *u v* **where** *u*: $u \in$ *carrier* (*relative_homology_group n* (*lsphere n*) (*equator n*))
and *v*: $v \in$ *carrier* (*relative_homology_group n* (*usphere n*) (*equator n*))
and *eq*: *?hi_ee f up* =
hom_induced n (*lsphere n*) (*equator n*) (*nsphere n*) (*equator n*) *id u*
$\otimes_{?rhgn\ (equator\ n)}$
hom_induced n (*usphere n*) (*equator n*) (*nsphere n*) (*equator n*) *id v*
using *subsetD* [*OF sb hiupcarr*] **by** *auto*
have $u \in$ *carrier* (*subgroup_generated* (*relative_homology_group n* (*lsphere n*) (*equator n*)) {*zp*})
by (*simp_all add*: *u zp_sg*)

then obtain *a*::*int* **where** *a*: *u* = *zp* [^]_{*relative_homology_group n (lsphere n) (equator n)*} *a*
by (*metis group.carrier_subgroup_generated_by_singleton group_relative_homology_group rangeE zpcarr*)
have *ae*: *hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id (pow (relative_homology_group n (lsphere n) (equator n)) zp a) = pow (?rhgn (equator n)) (hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id zp) a*
by (*meson group_hom.hom_int_pow group_hom_axioms_def group_hom_def group_relative_homology_group hom_induced zpcarr*)
have *v* ∈ *carrier (subgroup_generated (relative_homology_group n (usphere n) (equator n)) {zn})*
by (*simp_all add*: *v zn_sg*)
then obtain *b*::*int* **where** *b*: *v* = *zn* [^]_{*relative_homology_group n (usphere n) (equator n)*} *b*
by (*metis group.carrier_subgroup_generated_by_singleton group_relative_homology_group rangeE zncarr*)
have *be*: *hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id (zn* [^]_{*relative_homology_group n (usphere n) (equator n)*} *b)*
= *hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id zn* [^]_{*relative_homology_group n (nsphere n) (equator n)*} *b*
by (*meson group_hom.hom_int_pow group_hom_axioms_def group_hom_def group_relative_homology_group hom_induced zncarr*)
show *thesis*
proof
show *?hi_ee f up*
= *up* [^]_{*?rhgn (equator n)*} *a* ⊗_{*?rhgn (equator n)*} *un* [^]_{*?rhgn (equator n)*} *b*
using *a ae b be eq local.up_def un_def* **by** *auto*
qed
qed
have (*hom_boundary n (nsphere n) (equator n)*
∘ *hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id) zp = z*
using *zp_z equ* **apply** (*simp add*: *lsphere_def naturality_hom_induced*)
by (*metis hom_boundary_carrier hom_induced_id*)
then have *up_z*: *hom_boundary n (nsphere n) (equator n) up = z*
by (*simp add*: *up_def*)
have (*hom_boundary n (nsphere n) (equator n)*
∘ *hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id) zn = z*
using *zn_z equ* **apply** (*simp add*: *usphere_def naturality_hom_induced*)
by (*metis hom_boundary_carrier hom_induced_id*)
then have *un_z*: *hom_boundary n (nsphere n) (equator n) un = z*
by (*simp add*: *un_def*)
have *Bd_ab*: *Brouwer_degree2 (n − Suc 0) f = a + b*
proof (*rule Brouwer_degree2_unique_generator*; *use False int_ops* **in** *simp_all*)
show *continuous_map (nsphere (n − Suc 0)) (nsphere (n − Suc 0)) f*
using *cmr* **by** *auto*
show *subgroup_generated (reduced_homology_group (int n − 1) (nsphere (n − Suc 0))) {z} =*
reduced_homology_group (int n − 1) (nsphere (n − Suc 0))

```
    using zeq by blast
  have (hom_induced (int n − 1) (nsphere (n − Suc 0)) {} (nsphere (n − Suc
0)) {} f
        ∘ hom_boundary n (nsphere n) (equator n)) up
      = (hom_boundary n (nsphere n) (equator n) ∘
          ?hi_ee f) up
    using naturality_hom_induced [OF cmf fimeq, of n, symmetric]
    by (simp add: subtopology_restrict equ fun_eq_iff)
  also have ... = hom_boundary n (nsphere n) (equator n)
                    (up [^]relative_homology_group n (nsphere n) (equator n)
                     a ⊗relative_homology_group n (nsphere n) (equator n)
                     un [^]relative_homology_group n (nsphere n) (equator n) b)
    by (simp add: o_def up_ab)
  also have ... = z [^]reduced_homology_group (int n − 1) (nsphere (n − Suc 0))
(a + b)
    using zcarr
  apply (simp add: HB.hom_int_pow reduced_homology_group_def group.int_pow_subgroup_generat
upcarr uncarr)
   by (metis equ(1) group.int_pow_mult group_relative_homology_group hom_boundary_carrier
un_z up_z)
 finally show hom_induced (int n − 1) (nsphere (n − Suc 0)) {} (nsphere (n
− Suc 0)) {} f z =
      z [^]reduced_homology_group (int n − 1) (nsphere (n − Suc 0)) (a + b)
    by (simp add: up_z)
 qed
 define u where u ≡ up ⊗?rhgn (equator n) inv?rhgn (equator n) un
 have ucarr: u ∈ carrier (?rhgn (equator n))
   by (simp add: u_def uncarr upcarr)
 then have u [^]?rhgn (equator n) Brouwer_degree2 n f = u [^]?rhgn (equator n)
(a − b)
          ⟷ (GE.ord u) dvd a − b − Brouwer_degree2 n f
   by (simp add: GE.int_pow_eq)
 moreover
 have GE.ord u = 0
 proof (clarsimp simp add: GE.ord_eq_0 ucarr)
   fix d :: nat
   assume 0 < d
      and u [^]?rhgn (equator n) d = singular_relboundary_set n (nsphere n)
(equator n)
   then have hom_induced n (nsphere n) (equator n) (nsphere n) (upper n) id u
[^]?rhgn (upper n) d
              = 1?rhgn (upper n)
     by (metis HIU.hom_one HIU.hom_nat_pow one_relative_homology_group
ucarr)
   moreover
   have ?hi_lu
      = hom_induced n (nsphere n) (equator n) (nsphere n) (upper n) id ∘
        hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id
```

by (*simp add*: *lsphere_def image_subset_iff equator_upper flip*: *hom_induced_compose*)
then have *p*: *wp* = *hom_induced n* (*nsphere n*) (*equator n*) (*nsphere n*) (*upper n*) *id up*
by (*simp add*: *local.up_def wp_def*)
have *n*: *hom_induced n* (*nsphere n*) (*equator n*) (*nsphere n*) (*upper n*) *id un* = $\mathbf{1}_{?rhgn\ (upper\ n)}$
using *homology_exactness_triple_3* [*OF equator_upper*, *of n nsphere n*]
using *un_def zncarr* **by** (*auto simp*: *upper_usphere kernel_def*)
have *hom_induced n* (*nsphere n*) (*equator n*) (*nsphere n*) (*upper n*) *id u* = *wp*
unfolding *u_def*
using *p n HIU.inv_one HIU.r_one uncarr upcarr* **by** *auto*
ultimately have (*wp* $[\urcorner]_{?rhgn\ (upper\ n)}$ *d*) = $\mathbf{1}_{?rhgn\ (upper\ n)}$
by *simp*
moreover have *infinite* (*carrier* (*subgroup_generated* (*?rhgn* (*upper n*)) {*wp*}))
proof −
have *?rhgn* (*upper n*) ≅ *reduced_homology_group n* (*nsphere n*)
unfolding *upper_def*
using *iso_reduced_homology_group_upper_hemisphere* [*of n n n*]
by (*blast intro*: *group.iso_sym group_reduced_homology_group is_isoI*)
also have . . . ≅ *integer_group*
by (*simp add*: *reduced_homology_group_nsphere*)
finally have *iso*: *?rhgn* (*upper n*) ≅ *integer_group* **.**
have *carrier* (*subgroup_generated* (*?rhgn* (*upper n*)) {*wp*}) = *carrier* (*?rhgn* (*upper n*))
using *gh_lu.subgroup_generated_by_image* [*of* {*zp*}] *zpcarr HIU.carrier_subgroup_generated_subset*
gh_lu.iso_iff iso_relative_homology_group_lower_hemisphere zp_sg
by (*auto simp*: *lower_def lsphere_def upper_def equator_def wp_def*)
then show *?thesis*
using *infinite_UNIV_int iso_finite* [*OF iso*] **by** *simp*
qed
ultimately show *False*
using *HIU.finite_cyclic_subgroup* ‹*0* < *d*› *wpcarr* **by** *blast*
qed
ultimately have *iff*: *u* $[\urcorner]_{?rhgn\ (equator\ n)}$ *Brouwer_degree2 n f* = *u* $[\urcorner]_{?rhgn\ (equator\ n)}$ (*a* − *b*)
⟷ *Brouwer_degree2 n f* = *a* − *b*
by *auto*
have *u* $[\urcorner]_{?rhgn\ (equator\ n)}$ *Brouwer_degree2 n f* = *?hi_ee f u*
proof −
have *ne*: *topspace* (*nsphere n*) ∩ *equator n* ≠ {}
using *False equator_def in_topspace_nsphere* **by** *fastforce*
have *eq1*: *hom_boundary n* (*nsphere n*) (*equator n*) *u*
= $\mathbf{1}_{reduced\_homology\_group\ (int\ n\ -\ 1)\ (subtopology\ (nsphere\ n)\ (equator\ n))}$
using *one_reduced_homology_group u_def un_z uncarr up_z upcarr* **by** *force*
then have *uhom*: *u* ∈ *hom_induced n* (*nsphere n*) {} (*nsphere n*) (*equator n*) *id* ‘
carrier (*reduced_homology_group* (*int n*) (*nsphere n*))
using *homology_exactness_reduced_1* [*OF ne*, *of n*] *eq1 ucarr* **by** (*auto simp*: *kernel_def*)

then obtain *v* **where** *vcarr*: *v* ∈ *carrier* (*reduced_homology_group* (*int n*) (*nsphere n*))
and *ueq*: *u* = *hom_induced n* (*nsphere n*) {} (*nsphere n*) (*equator n*) *id v*
by *blast*
interpret *GH_hi*: *group_hom homology_group n* (*nsphere n*)
?rhgn (*equator n*)
hom_induced n (*nsphere n*) {} (*nsphere n*) (*equator n*) *id*
by (*simp add*: *group_hom_axioms_def group_hom_def hom_induced_hom*)
have *poweq*: *pow* (*homology_group n* (*nsphere n*)) *x i* = *pow* (*reduced_homology_group n* (*nsphere n*)) *x i*
for *x* **and** *i*::*int*
by (*simp add*: *False un_reduced_homology_group*)
have *vcarr'*: *v* ∈ *carrier* (*homology_group n* (*nsphere n*))
using *carrier_reduced_homology_group_subset vcarr* **by** *blast*
have *u* $[\frown]_{?rhgn\ (equator\ n)}$ *Brouwer_degree2 n f*
= *hom_induced n* (*nsphere n*) {} (*nsphere n*) (*equator n*) *f v*
using *vcarr vcarr'*
by (*simp add*: *ueq poweq hom_induced_compose' cmf flip*: *GH_hi.hom_int_pow Brouwer_degree2*)
also have ... = *hom_induced n* (*nsphere n*) (*topspace*(*nsphere n*) ∩ *equator n*) (*nsphere n*) (*equator n*) *f*
(*hom_induced n* (*nsphere n*) {} (*nsphere n*) (*topspace*(*nsphere n*) ∩ *equator n*) *id v*)
using *fimeq* **by** (*simp add*: *hom_induced_compose' cmf*)
also have ... = *?hi_ee f u*
by (*metis hom_induced inf.left_idem ueq*)
finally show *?thesis* **.**
qed
moreover
interpret *gh_een*: *group_hom ?rhgn* (*equator n*) *?rhgn* (*equator n*) *?hi_ee neg*
by (*simp add*: *group_hom_axioms_def group_hom_def hom_induced_hom*)
have *hi_up_eq_un*: *?hi_ee neg up* = *un* $[\frown]_{?rhgn\ (equator\ n)}$ *Brouwer_degree2* (*n* − *Suc 0*) *neg*
proof −
have *?hi_ee neg* (*hom_induced n* (*lsphere n*) (*equator n*) (*nsphere n*) (*equator n*) *id zp*)
= *hom_induced n* (*lsphere n*) (*equator n*) (*nsphere n*) (*equator n*) (*neg* ∘ *id*) *zp*
by (*intro hom_induced_compose'*) (*auto simp*: *lsphere_def equator_def cm_neg*)
also have ... = *hom_induced n* (*usphere n*) (*equator n*) (*nsphere n*) (*equator n*) *id*
(*hom_induced n* (*lsphere n*) (*equator n*) (*usphere n*) (*equator n*) *neg zp*)
by (*subst hom_induced_compose'* [*OF cm_neg_lu*]) (*auto simp*: *usphere_def equator_def*)
also have *hom_induced n* (*lsphere n*) (*equator n*) (*usphere n*) (*equator n*) *neg zp*
= *zn* $[\frown]_{relative\_homology\_group\ n\ (usphere\ n)\ (equator\ n)}$ *Brouwer_degree2* (*n* − *Suc 0*) *neg*

proof −
let *?hb* = *hom_boundary n* (*usphere n*) (*equator n*)
have *eq*: *subtopology* (*nsphere n*) {*x*. *x n* ≥ *0*} = *usphere n* ∧ {*x*. *x n* = *0*} = *equator n*
by (*auto simp*: *usphere_def upper_def equator_def*)
with *hb_iso* **have** *inj*: *inj_on* (*?hb*) (*carrier* (*relative_homology_group n* (*usphere n*) (*equator n*)))
by (*simp add*: *iso_iff*)
interpret *hb_hom*: *group_hom relative_homology_group n* (*usphere n*) (*equator n*)
reduced_homology_group (*int n* − *1*) (*nsphere* (*n* − *Suc 0*))
?hb
using *hb_iso iso_iff eq group_hom_axioms_def group_hom_def* **by** *fastforce*
show *?thesis*
proof (*rule inj_onD* [*OF inj*])
have ∗: *hom_induced* (*int n* − *1*) (*nsphere* (*n* − *Suc 0*)) {} (*nsphere* (*n* − *Suc 0*)) {} *neg z*
= $z\ [\urcorner]_{homology\_group\ (int\ n\ -\ 1)\ (nsphere\ (n\ -\ Suc\ 0))}$ *Brouwer_degree2* (*n* − *Suc 0*) *neg*
using *Brouwer_degree2* [*of z n* − *Suc 0 neg*] *False zcarr*
by (*simp add*: *int_ops group.int_pow_subgroup_generated reduced_homology_group_def*)
have *?hb* ∘
hom_induced n (*lsphere n*) (*equator n*) (*usphere n*) (*equator n*) *neg*
= *hom_induced* (*int n* − *1*) (*nsphere* (*n* − *Suc 0*)) {} (*nsphere* (*n* − *Suc 0*)) {} *neg* ∘
hom_boundary n (*lsphere n*) (*equator n*)
apply (*subst naturality_hom_induced* [*OF cm_neg_lu*])
apply (*force simp*: *equator_def neg_def*)
by (*simp add*: *equ*)
then have *?hb*
(*hom_induced n* (*lsphere n*) (*equator n*) (*usphere n*) (*equator n*) *neg zp*)
= ($z\ [\urcorner]_{homology\_group\ (int\ n\ -\ 1)\ (nsphere\ (n\ -\ Suc\ 0))}$ *Brouwer_degree2* (*n* − *Suc 0*) *neg*)
by (*metis* ∗ *comp_apply zp_z*)
also have ... = *?hb* ($zn\ [\urcorner]_{relative\_homology\_group\ n\ (usphere\ n)\ (equator\ n)}$
Brouwer_degree2 (*n* − *Suc 0*) *neg*)
by (*metis group.int_pow_subgroup_generated group_relative_homology_group hb_hom.hom_int_pow reduced_homology_group_def zcarr zn_z zncarr*)
finally show *?hb* (*hom_induced n* (*lsphere n*) (*equator n*) (*usphere n*) (*equator n*) *neg zp*) =
?hb ($zn\ [\urcorner]_{relative\_homology\_group\ n\ (usphere\ n)\ (equator\ n)}$
Brouwer_degree2 (*n* − *Suc 0*) *neg*) **by** *simp*
qed (*auto simp*: *hom_induced_carrier group.int_pow_closed zncarr*)
qed
finally show *?thesis*
by (*metis* (*no_types*, *lifting*) *group_hom.hom_int_pow group_hom_axioms_def group_hom_def group_relative_homology_group hom_induced local.up_def un_def*

zncarr)
qed
have *continuous_map* (*nsphere* (*n* − *Suc 0*)) (*nsphere* (*n* − *Suc 0*)) *neg*
using *cm_neg* **by** *blast*
then have *homeomorphic_map* (*nsphere* (*n* − *Suc 0*)) (*nsphere* (*n* − *Suc 0*)) *neg*
apply (*auto simp*: *homeomorphic_map_maps homeomorphic_maps_def*)
apply (*rule_tac x*=*neg* **in** *exI*, *auto*)
done
then have *Brouwer_degree2_21*: *Brouwer_degree2* (*n* − *Suc 0*) *neg* ^ *2* = *1*
using *Brouwer_degree2_homeomorphic_map power2_eq_1_iff* **by** *force*
have *hi_un_eq_up*: *?hi_ee neg un* = *up* $[\urcorner]_{?rhgn\ (equator\ n)}$ *Brouwer_degree2* (*n* − *Suc 0*) *neg* (**is** *?f un* = *?y*)
proof −
have [*simp*]: *neg* ∘ *neg* = *id*
by *force*
have *?f* (*?f ?y*) = *?y*
apply (*subst hom_induced_compose′* [*OF cm_neg _ cm_neg*])
apply(*force simp*: *equator_def*)
apply (*simp add*: *upcarr hom_induced_id_gen*)
done
moreover have *?f ?y* = *un*
using *upcarr* **apply** (*simp only*: *gh_een.hom_int_pow hi_up_eq_un*)
by (*metis* (*no_types*, *lifting*) *Brouwer_degree2_21 GE.group_l_invI GE.l_inv_ex group.int_pow_1 group.int_pow_pow power2_eq_1_iff uncarr zmult_eq_1_iff*)
ultimately show *?f un* = *?y*
by *simp*
qed
have *?hi_ee f un* = *un* $[\urcorner]_{?rhgn\ (equator\ n)}$ *a* $\otimes_{?rhgn\ (equator\ n)}$ *up* $[\urcorner]_{?rhgn\ (equator\ n)}$ *b*
proof −
let *?TE* = *topspace* (*nsphere n*) ∩ *equator n*
have *fneg*: (*f* ∘ *neg*) *x* = (*neg* ∘ *f*) *x* **if** *x* ∈ *topspace* (*nsphere n*) **for** *x*
using *f* [*OF that*] **by** (*force simp*: *neg_def*)
have *neg_im*: *neg* ‘ (*topspace* (*nsphere n*) ∩ *equator n*) ⊆ *topspace* (*nsphere n*) ∩ *equator n*
by (*metis cm_neg continuous_map_image_subset_topspace equ*(*1*) *topspace_subtopology*)
have *1*: *hom_induced n* (*nsphere n*) *?TE* (*nsphere n*) *?TE f* ∘ *hom_induced n* (*nsphere n*) *?TE* (*nsphere n*) *?TE neg*
= *hom_induced n* (*nsphere n*) *?TE* (*nsphere n*) *?TE neg* ∘ *hom_induced n* (*nsphere n*) *?TE* (*nsphere n*) *?TE f*
using *neg_im fimeq cm_neg cmf*
apply (*simp add*: *flip*: *hom_induced_compose del*: *hom_induced_restrict*)
using *fneg* **by** (*auto intro*: *hom_induced_eq*)
have (*un* $[\urcorner]_{?rhgn\ (equator\ n)}$ *a*) $\otimes_{?rhgn\ (equator\ n)}$ (*up* $[\urcorner]_{?rhgn\ (equator\ n)}$ *b*)
= *un* $[\urcorner]_{?rhgn\ (equator\ n)}$ (*Brouwer_degree2* (*n* − *1*) *neg* ∗ *a* ∗ *Brouwer_degree2* (*n* − *1*) *neg*)
$\otimes_{?rhgn\ (equator\ n)}$

up [^]*?rhgn* (*equator n*) (*Brouwer_degree2* (*n* − *1*) *neg* ∗ *b* ∗ *Brouwer_degree2* (*n* − *1*) *neg*)
proof −
have *Brouwer_degree2* (*n* − *Suc 0*) *neg* = *1* ∨ *Brouwer_degree2* (*n* − *Suc 0*) *neg* = − *1*
using *Brouwer_degree2_21 power2_eq_1_iff* **by** *blast*
then show *?thesis*
by *fastforce*
qed
also have ... = ((*un* [^]*?rhgn* (*equator n*) *Brouwer_degree2* (*n* − *1*) *neg*) [^]*?rhgn* (*equator n*) *a* ⊗*?rhgn* (*equator n*)
(*up* [^]*?rhgn* (*equator n*) *Brouwer_degree2* (*n* − *1*) *neg*) [^]*?rhgn* (*equator n*) *b*) [^]*?rhgn* (*equator n*)
Brouwer_degree2 (*n* − *1*) *neg*
by (*simp add*: *GE.int_pow_distrib GE.int_pow_pow uncarr upcarr*)
also have ... = *?hi_ee neg* (*?hi_ee f up*) [^]*?rhgn* (*equator n*) *Brouwer_degree2* (*n* − *Suc 0*) *neg*
by (*simp add*: *gh_een.hom_int_pow hi_un_eq_up hi_up_eq_un uncarr up_ab upcarr*)
finally have *2*: (*un* [^]*?rhgn* (*equator n*) *a*) ⊗*?rhgn* (*equator n*) (*up* [^]*?rhgn* (*equator n*) *b*)
= *?hi_ee neg* (*?hi_ee f up*) [^]*?rhgn* (*equator n*) *Brouwer_degree2* (*n* − *Suc 0*) *neg* .
have *un* = *?hi_ee neg up* [^]*?rhgn* (*equator n*) *Brouwer_degree2* (*n* − *Suc 0*) *neg*
by (*metis* (*no_types*, *opaque_lifting*) *Brouwer_degree2_21 GE.int_pow_1 GE.int_pow_pow hi_up_eq_un power2_eq_1_iff uncarr zmult_eq_1_iff*)
moreover have *?hi_ee f* ((*?hi_ee neg up*) [^]*?rhgn* (*equator n*) (*Brouwer_degree2* (*n* − *Suc 0*) *neg*))
= *un* [^]*?rhgn* (*equator n*) *a* ⊗*?rhgn* (*equator n*) *up* [^]*?rhgn* (*equator n*) *b*
using *1 2* **by** (*simp add*: *hom_induced_carrier gh_eef.hom_int_pow fun_eq_iff*)
ultimately show *?thesis*
by *blast*
qed
then have *?hi_ee f u* = *u* [^]*?rhgn* (*equator n*) (*a* − *b*)
by (*simp add*: *u_def upcarr uncarr up_ab GE.int_pow_diff GE.m_ac GE.int_pow_distrib GE.int_pow_inv GE.inv_mult_group*)
ultimately
have *Brouwer_degree2 n f* = *a* − *b*
using *iff* **by** *blast*
with *Bd_ab* **show** *?thesis*
by *simp*
qed *simp*

0.4.2 General Jordan-Brouwer separation theorem and invariance of dimension

proposition *relative_homology_group_Euclidean_complement_step*:
assumes *closedin* (*Euclidean_space n*) *S*
shows *relative_homology_group p* (*Euclidean_space n*) (*topspace*(*Euclidean_space n*) − *S*)
$\cong$ *relative_homology_group* (*p* + *k*) (*Euclidean_space* (*n*+*k*)) (*topspace*(*Euclidean_space* (*n*+*k*)) − *S*)
proof −
have ∗: *relative_homology_group p* (*Euclidean_space n*) (*topspace*(*Euclidean_space n*) − *S*)
$\cong$ *relative_homology_group* (*p* + *1*) (*Euclidean_space* (*Suc n*)) (*topspace*(*Euclidean_space* (*Suc n*)) − {*x* ∈ *S*. *x n* = *0*})
(**is** *?lhs* $\cong$ *?rhs*)
if *clo*: *closedin* (*Euclidean_space* (*Suc n*)) *S* **and** *cong*: $\bigwedge$*x y*. ⟦*x* ∈ *S*; $\bigwedge$*i*. *i* ≠ *n* ⟹ *x i* = *y i*⟧ ⟹ *y* ∈ *S*
for *p n S*
proof −
have *Ssub*: *S* ⊆ *topspace* (*Euclidean_space* (*Suc n*))
by (*meson clo closedin_def*)
define *lo* **where** *lo* ≡ {*x* ∈ *topspace*(*Euclidean_space* (*Suc n*)). *x n* < (*if x* ∈ *S then 0 else 1*)}
define *hi* **where** *hi* = {*x* ∈ *topspace*(*Euclidean_space* (*Suc n*)). *x n* > (*if x* ∈ *S then 0 else* −*1*)}
have *lo_hi_Int*: *lo* ∩ *hi* = {*x* ∈ *topspace*(*Euclidean_space* (*Suc n*)) − *S*. *x n* ∈ {−*1*<..<*1*}}
by (*auto simp*: *hi_def lo_def*)
have *lo_hi_Un*: *lo* ∪ *hi* = *topspace*(*Euclidean_space* (*Suc n*)) − {*x* ∈ *S*. *x n* = *0*}
by (*auto simp*: *hi_def lo_def*)
define *ret* **where** *ret* ≡ λ*c*::*real*. λ*x i*. *if i* = *n then c else x i*
have *cm_ret*: *continuous_map* (*powertop_real UNIV*) (*powertop_real UNIV*) (*ret t*) **for** *t*
by (*auto simp*: *ret_def continuous_map_componentwise_UNIV intro*: *continuous_map_product_projection*)
let *?ST* = λ*t*. *subtopology* (*Euclidean_space* (*Suc n*)) {*x*. *x n* = *t*}
define *squashable* **where**
squashable ≡ λ*t S*. ∀ *x t*′. *x* ∈ *S* ∧ (*x n* ≤ *t*′ ∧ *t*′ ≤ *t* ∨ *t* ≤ *t*′ ∧ *t*′ ≤ *x n*) ⟶ *ret t*′ *x* ∈ *S*
have *squashable*: *squashable t* (*topspace*(*Euclidean_space*(*Suc n*))) **for** *t*
by (*simp add*: *squashable_def topspace_Euclidean_space ret_def*)
have *squashableD*: ⟦*squashable t S*; *x* ∈ *S*; *x n* ≤ *t*′ ∧ *t*′ ≤ *t* ∨ *t* ≤ *t*′ ∧ *t*′ ≤ *x n*⟧ ⟹ *ret t*′ *x* ∈ *S* **for** *x t*′ *t S*
by (*auto simp*: *squashable_def*)
have *squashable 1 hi*
by (*force simp*: *squashable_def hi_def ret_def topspace_Euclidean_space intro*: *cong*)
have *squashable t UNIV* **for** *t*
by (*force simp*: *squashable_def hi_def ret_def topspace_Euclidean_space*

```
intro: cong)
    have squashable_0_lohi: squashable 0 (lo ∩ hi)
      using Ssub
     by (auto simp: squashable_def hi_def lo_def ret_def topspace_Euclidean_space
intro: cong)
    have rm_ret: retraction_maps (subtopology (Euclidean_space (Suc n)) U)
                              (subtopology (Euclidean_space (Suc n)) {x. x ∈ U ∧ x
n = t})
                              (ret t) id
      if squashable t U for t U
      unfolding retraction_maps_def
    proof (intro conjI ballI)
      show continuous_map (subtopology (Euclidean_space (Suc n)) U)
             (subtopology (Euclidean_space (Suc n)) {x ∈ U. x n = t}) (ret t)
       apply (simp add: cm_ret continuous_map_in_subtopology continuous_map_from_subtopology
Euclidean_space_def)
        using that by (fastforce simp: squashable_def ret_def)
    next
      show continuous_map (subtopology (Euclidean_space (Suc n)) {x ∈ U. x n
= t})
                         (subtopology (Euclidean_space (Suc n)) U) id
        using continuous_map_in_subtopology by fastforce
      show ret t (id x) = x
        if x ∈ topspace (subtopology (Euclidean_space (Suc n)) {x ∈ U. x n = t})
for x
        using that by (simp add: topspace_Euclidean_space ret_def fun_eq_iff)
    qed
   have cm_snd: continuous_map (prod_topology (top_of_set {0..1}) (subtopology
(powertop_real UNIV) S))
                            euclideanreal (λx. snd x k) for k::nat and S
     using continuous_map_componentwise_UNIV continuous_map_into_fulltopology
continuous_map_snd by fastforce
   have cm_fstsnd: continuous_map (prod_topology (top_of_set {0..1}) (subtopology
(powertop_real UNIV) S))
                            euclideanreal (λx. fst x ∗ snd x k) for k::nat and S
      by (intro continuous_intros continuous_map_into_fulltopology [OF continu-
ous_map_fst] cm_snd)
    have hw_sub: homotopic_with (λk. k ‘ V ⊆ V) (subtopology (Euclidean_space
(Suc n)) U)
                              (subtopology (Euclidean_space (Suc n)) U) (ret t) id
      if squashable t U squashable t V for U V t
      unfolding homotopic_with_def
    proof (intro exI conjI allI ballI)
      let ?h = λ(z,x). ret ((1 − z) ∗ t + z ∗ x n) x
      show (λx. ?h (u, x)) ‘ V ⊆ V if u ∈ {0..1} for u
        using that
         by clarsimp (metis squashableD [OF ‹squashable t V›] convex_bound_le
diff_ge_0_iff_ge eq_diff_eq' le_cases less_eq_real_def segment_bound_lemma)
      have 1: ?h ‘ ({0..1} × ({x. ∀ i≥Suc n. x i = 0} ∩ U)) ⊆ U
```

by *clarsimp* (*metis squashableD* [*OF* ‹*squashable t U*›] *convex_bound_le diff_ge_0_iff_ge eq_diff_eq′ le_cases less_eq_real_def segment_bound_lemma*)
show *continuous_map* (*prod_topology* (*top_of_set* {*0..1*}) (*subtopology* (*Euclidean_space* (*Suc n*)) *U*))
(*subtopology* (*Euclidean_space* (*Suc n*)) *U*) *?h*
apply (*simp add*: *continuous_map_in_subtopology Euclidean_space_def subtopology_subtopology 1*)
apply (*auto simp*: *case_prod_unfold ret_def continuous_map_componentwise_UNIV*)
apply (*intro continuous_map_into_fulltopology* [*OF continuous_map_fst*] *cm_snd continuous_intros*)
by (*auto simp*: *cm_snd*)
qed (*auto simp*: *ret_def*)
have *cs_hi*: *contractible_space*(*subtopology* (*Euclidean_space*(*Suc n*)) *hi*)
proof −
have *homotopic_with* (λ*x*. *True*) (*?ST 1*) (*?ST 1*) *id* (λ*x*. (λ*i*. *if i* = *n then 1 else 0*))
apply (*subst homotopic_with_sym*)
apply (*simp add*: *homotopic_with*)
apply (*rule_tac x*=(λ(*z*,*x*) *i*. *if i*=*n then 1 else z* ∗ *x i*) **in** *exI*)
apply (*auto simp*: *Euclidean_space_def subtopology_subtopology continuous_map_in_subtopology case_prod_unfold continuous_map_componentwise_UNIV cm_fstsnd*)
done
then have *contractible_space* (*?ST 1*)
unfolding *contractible_space_def* **by** *metis*
moreover have *?thesis* = *contractible_space* (*?ST 1*)
proof (*intro deformation_retract_imp_homotopy_equivalent_space homotopy_equivalent_space_contractibility*)
have {*x*. ∀ *i*≥*Suc n*. *x i* = *0*} ∩ {*x* ∈ *hi*. *x n* = *1*} = {*x*. ∀ *i*≥*Suc n*. *x i* = *0*} ∩ {*x*. *x n* = *1*}
by (*auto simp*: *hi_def topspace_Euclidean_space*)
then have *eq*: *subtopology* (*Euclidean_space* (*Suc n*)) {*x*. *x* ∈ *hi* ∧ *x n* = *1*} = *?ST 1*
by (*simp add*: *Euclidean_space_def subtopology_subtopology*)
show *homotopic_with* (λ*x*. *True*) (*subtopology* (*Euclidean_space* (*Suc n*)) *hi*) (*subtopology* (*Euclidean_space* (*Suc n*)) *hi*) (*ret 1*) *id*
using *hw_sub* [*OF* ‹*squashable 1 hi*› ‹*squashable 1 UNIV*›] *eq* **by** *simp*
show *retraction_maps* (*subtopology* (*Euclidean_space* (*Suc n*)) *hi*) (*?ST 1*) (*ret 1*) *id*
using *rm_ret* [*OF* ‹*squashable 1 hi*›] *eq* **by** *simp*
qed
ultimately show *?thesis* **by** *metis*
qed
have *?lhs* ≅ *relative_homology_group p* (*Euclidean_space* (*Suc n*)) (*lo* ∩ *hi*)
proof (*rule group.iso_sym* [*OF* _ *deformation_retract_imp_isomorphic_relative_homology_groups*])
have {*x*. ∀ *i*≥*Suc n*. *x i* = *0*} ∩ {*x*. *x n* = *0*} = {*x*. ∀ *i*≥*n*. *x i* = (*0*::*real*)}
by *auto* (*metis le_less_Suc_eq not_le*)
then have *?ST 0* = *Euclidean_space n*
by (*simp add*: *Euclidean_space_def subtopology_subtopology*)

```
      then show retraction_maps (Euclidean_space (Suc n)) (Euclidean_space n)
(ret 0) id
        using rm_ret [OF ‹squashable 0 UNIV›] by auto
      then have ret 0 x ∈ topspace (Euclidean_space n)
        if x ∈ topspace (Euclidean_space (Suc n)) −1 < x n x n < 1 for x
      using that by (metis continuous_map_image_subset_topspace image_subset_iff
retraction_maps_def)
      then show (ret 0) ‘ (lo ∩ hi) ⊆ topspace (Euclidean_space n) − S
        by (auto simp: local.cong ret_def hi_def lo_def)
       show homotopic_with (λh. h ‘ (lo ∩ hi) ⊆ lo ∩ hi) (Euclidean_space (Suc
n)) (Euclidean_space (Suc n)) (ret 0) id
        using hw_sub [OF squashable squashable_0_lohi] by simp
    qed (auto simp: lo_def hi_def Euclidean_space_def)
    also have . . . ≅ relative_homology_group p (subtopology (Euclidean_space
(Suc n)) hi) (lo ∩ hi)
   proof (rule group.iso_sym [OF _ isomorphic_relative_homology_groups_inclusion_contractible])
      show contractible_space (subtopology (Euclidean_space (Suc n)) hi)
        by (simp add: cs_hi)
      show topspace (Euclidean_space (Suc n)) ∩ hi ≠ {}
        apply (simp add: hi_def topspace_Euclidean_space set_eq_iff)
        apply (rule_tac x=λi. if i = n then 1 else 0 in exI, auto)
        done
    qed auto
    also have . . . ≅ relative_homology_group p (subtopology (Euclidean_space
(Suc n)) (lo ∪ hi)) lo
    proof −
      have oo: openin (Euclidean_space (Suc n)) {x ∈ topspace (Euclidean_space
(Suc n)). x n ∈ A}
        if open A for A
      proof (rule openin_continuous_map_preimage)
        show continuous_map (Euclidean_space (Suc n)) euclideanreal (λx. x n)
        proof −
          have ∀ n f. continuous_map (product_topology f UNIV) (f (n::nat)) (λf.
f n::real)
            by (simp add: continuous_map_product_projection)
          then show ?thesis
            using Euclidean_space_def continuous_map_from_subtopology
            by (metis (mono_tags))
        qed
      qed (auto intro: that)
      have openin (Euclidean_space(Suc n)) lo
        apply (simp add: openin_subopen [of _ lo])
        apply (simp add: lo_def, safe)
         apply (force intro: oo [of lessThan 0, simplified] open_Collect_less)
        apply (rule_tac x={x ∈ topspace(Euclidean_space(Suc n)). x n < 1}
                        ∩ (topspace(Euclidean_space(Suc n)) − S) in exI)
      using clo apply (force intro: oo [of lessThan 1, simplified] open_Collect_less)
        done
      moreover have openin (Euclidean_space(Suc n)) hi
```

```
      apply (simp add: openin_subopen [of _ hi])
      apply (simp add: hi_def, safe)
       apply (force intro: oo [of greaterThan 0, simplified] open_Collect_less)
      apply (rule_tac x={x ∈ topspace(Euclidean_space(Suc n)). x n > −1}
                             ∩ (topspace(Euclidean_space(Suc n)) − S) in exI)
    using clo apply (force intro: oo [of greaterThan (−1), simplified] open_Collect_less)
      done
    ultimately
    have ∗: subtopology (Euclidean_space (Suc n)) (lo ∪ hi) closure_of
               (topspace (subtopology (Euclidean_space (Suc n)) (lo ∪ hi)) − hi)
               ⊆ subtopology (Euclidean_space (Suc n)) (lo ∪ hi) interior_of lo
       by (metis (no_types, lifting) Diff_idemp Diff_subset_conv Un_commute
Un_upper2 closure_of_interior_of interior_of_closure_of interior_of_complement
interior_of_eq lo_hi_Un openin_Un openin_open_subtopology topspace_subtopology_subset)
    have eq: ((lo ∪ hi) ∩ (lo ∪ hi − (topspace (Euclidean_space (Suc n)) ∩ (lo
∪ hi) − hi))) = hi
      (lo − (topspace (Euclidean_space (Suc n)) ∩ (lo ∪ hi) − hi)) = lo ∩ hi
      by (auto simp: lo_def hi_def Euclidean_space_def)
    show ?thesis
      using homology_excision_axiom [OF ∗, of lo ∪ hi p]
      by (force simp: subtopology_subtopology eq is_iso_def)
  qed
  also have ... ≅ relative_homology_group (p + 1 − 1) (subtopology (Euclidean_space
(Suc n)) (lo ∪ hi)) lo
    by simp
  also have ... ≅ relative_homology_group (p + 1) (Euclidean_space (Suc n))
(lo ∪ hi)
  proof (rule group.iso_sym [OF _ isomorphic_relative_homology_groups_relboundary_contractible])
    have proj: continuous_map (powertop_real UNIV) euclideanreal (λf. f n)
      by (metis UNIV_I continuous_map_product_projection)
    have hilo: ⋀x. x ∈ hi ⟹ (λi. if i = n then − x i else x i) ∈ lo
               ⋀x. x ∈ lo ⟹ (λi. if i = n then − x i else x i) ∈ hi
      using local.cong
     by (auto simp: hi_def lo_def topspace_Euclidean_space split: if_split_asm)
   have subtopology (Euclidean_space (Suc n)) hi homeomorphic_space subtopol-
ogy (Euclidean_space (Suc n)) lo
      unfolding homeomorphic_space_def
      apply (rule_tac x=λx i. if i = n then −(x i) else x i in exI)+
      using proj
      apply (auto simp: homeomorphic_maps_def Euclidean_space_def continu-
ous_map_in_subtopology
                          hilo continuous_map_componentwise_UNIV continu-
ous_map_from_subtopology continuous_map_minus
              intro: continuous_map_from_subtopology continuous_map_product_projection)
      done
    then have contractible_space(subtopology (Euclidean_space(Suc n)) hi)
        ⟷ contractible_space (subtopology (Euclidean_space (Suc n)) lo)
      by (rule homeomorphic_space_contractibility)
    then show contractible_space (subtopology (Euclidean_space (Suc n)) lo)
```

```
        using cs_hi by auto
      show topspace (Euclidean_space (Suc n)) ∩ lo ≠ {}
        apply (simp add: lo_def Euclidean_space_def set_eq_iff)
        apply (rule_tac x=λi. if i = n then −1 else 0 in exI, auto)
        done
    qed auto
    also have . . . ≅ ?rhs
      by (simp flip: lo_hi_Un)
    finally show ?thesis .
  qed
  show ?thesis
  proof (induction k)
    case (Suc m)
    with assms obtain T where cloT: closedin (powertop_real UNIV) T
                          and SeqT: S = T ∩ {x. ∀ i≥n. x i = 0}
      by (auto simp: Euclidean_space_def closedin_subtopology)
    then have closedin (Euclidean_space (m + n)) S
      apply (simp add: Euclidean_space_def closedin_subtopology)
      apply (rule_tac x=T ∩ topspace(Euclidean_space n) in exI)
      using closedin_Euclidean_space topspace_Euclidean_space by force
      moreover have relative_homology_group p (Euclidean_space n) (topspace
(Euclidean_space n) − S)
                    ≅ relative_homology_group (p + 1) (Euclidean_space (Suc n))
(topspace (Euclidean_space (Suc n)) − S)
      if closedin (Euclidean_space n) S for p n
    proof −
      define S' where S' ≡ {x ∈ topspace(Euclidean_space(Suc n)). (λi. if i < n
then x i else 0) ∈ S}
      have Ssub_n: S ⊆ topspace (Euclidean_space n)
        by (meson that closedin_def)
     have relative_homology_group p (Euclidean_space n) (topspace(Euclidean_space
n) − S')
        ≅ relative_homology_group (p + 1) (Euclidean_space (Suc n)) (topspace(Euclidean_space
(Suc n)) − {x ∈ S'. x n = 0})
      proof (rule ∗)
         have cm: continuous_map (powertop_real UNIV) euclideanreal (λf. f u)
for u
          by (metis UNIV_I continuous_map_product_projection)
        have continuous_map (subtopology (powertop_real UNIV) {x. ∀ i>n. x i =
0}) euclideanreal
                (λx. if k ≤ n then x k else 0) for k
          by (simp add: continuous_map_from_subtopology [OF cm])
        moreover have ∀ i≥n. (if i < n then x i else 0) = 0
          if x ∈ topspace (subtopology (powertop_real UNIV) {x. ∀ i>n. x i = 0})
for x
          using that by simp
      ultimately have continuous_map (Euclidean_space (Suc n)) (Euclidean_space
n) (λx i. if i < n then x i else 0)
            by (simp add: Euclidean_space_def continuous_map_in_subtopology
```

```
continuous_map_componentwise_UNIV
                     continuous_map_from_subtopology [OF cm] image_subset_iff)
       then show closedin (Euclidean_space (Suc n)) S'
       unfolding S'_def using that by (rule closedin_continuous_map_preimage)
     next
       fix x y
       assume xy: ⋀i. i ≠ n ⟹ x i = y i x ∈ S'
       then have (λi. if i < n then x i else 0) = (λi. if i < n then y i else 0)
         by (simp add: S'_def Euclidean_space_def fun_eq_iff)
       with xy show y ∈ S'
         by (simp add: S'_def Euclidean_space_def)
     qed
     moreover
     have abs_eq: (λi. if i < n then x i else 0) = x if ⋀i. i ≥ n ⟹ x i = 0 for
x :: nat ⇒ real and n
       using that by auto
     then have topspace (Euclidean_space n) − S' = topspace (Euclidean_space
n) − S
       by (simp add: S'_def Euclidean_space_def set_eq_iff cong: conj_cong)
     moreover
      have topspace (Euclidean_space (Suc n)) − {x ∈ S'. x n = 0} = topspace
(Euclidean_space (Suc n)) − S
       using Ssub_n
      apply (auto simp: S'_def subset_iff Euclidean_space_def set_eq_iff abs_eq
cong: conj_cong)
       by (metis abs_eq le_antisym not_less_eq_eq)
     ultimately show ?thesis
       by simp
   qed
    ultimately have relative_homology_group (p + m)(Euclidean_space (m +
n))(topspace (Euclidean_space (m + n)) − S)
          ≅ relative_homology_group (p + m + 1) (Euclidean_space (Suc (m +
n))) (topspace (Euclidean_space (Suc (m + n))) − S)
     by (metis ‹closedin (Euclidean_space (m + n)) S›)
   then show ?case
     using Suc.IH iso_trans by (force simp: algebra_simps)
 qed (simp add: iso_refl)
qed

lemma iso_Euclidean_complements_lemma1:
 assumes S: closedin (Euclidean_space m) S and cmf: continuous_map(subtopology
(Euclidean_space m) S) (Euclidean_space n) f
 obtains g where continuous_map (Euclidean_space m) (Euclidean_space n) g
               ⋀x. x ∈ S ⟹ g x = f x
proof −
 have cont: continuous_on (topspace (Euclidean_space m) ∩ S) (λx. f x i) for i
   by (metis (no_types) continuous_on_product_then_coordinatewise
         cm_Euclidean_space_iff_continuous_on cmf topspace_subtopology)
 have f ` (topspace (Euclidean_space m) ∩ S) ⊆ topspace (Euclidean_space n)
```

```
    using cmf continuous_map_image_subset_topspace by fastforce
  then
  have ∃g. continuous_on (topspace (Euclidean_space m)) g ∧ (∀x ∈ S. g x = f x i) for i
    using S Tietze_unbounded [OF cont [of i]]
   by (metis closedin_Euclidean_space_iff closedin_closed_Int topspace_subtopology topspace_subtopology_subset)
 then obtain g where cmg: ⋀i. continuous_map (Euclidean_space m) euclideanreal (g i)
    and gf: ⋀i x. x ∈ S ⟹ g i x = f x i
    unfolding continuous_map_Euclidean_space_iff by metis
  let ?GG = λx i. if i < n then g i x else 0
  show thesis
  proof
    show continuous_map (Euclidean_space m) (Euclidean_space n) ?GG
      unfolding Euclidean_space_def [of n]
     by (auto simp: continuous_map_in_subtopology continuous_map_componentwise cmg)
    show ?GG x = f x if x ∈ S for x
    proof -
      have S ⊆ topspace (Euclidean_space m)
        by (meson S closedin_def)
      then have f x ∈ topspace (Euclidean_space n)
         using cmf that unfolding continuous_map_def topspace_subtopology by blast
      then show ?thesis
        by (force simp: topspace_Euclidean_space gf that)
    qed
  qed
qed


lemma iso_Euclidean_complements_lemma2:
  assumes S: closedin (Euclidean_space m) S
      and T: closedin (Euclidean_space n) T
     and hom: homeomorphic_map (subtopology (Euclidean_space m) S) (subtopology (Euclidean_space n) T) f
 obtains g where homeomorphic_map (prod_topology (Euclidean_space m) (Euclidean_space n))
                                  (prod_topology (Euclidean_space n) (Euclidean_space m)) g
                 ⋀x. x ∈ S ⟹ g(x,(λi. 0)) = (f x,(λi. 0))
proof -
  obtain g where cmf: continuous_map (subtopology (Euclidean_space m) S) (subtopology (Euclidean_space n) T) f
        and cmg: continuous_map (subtopology (Euclidean_space n) T) (subtopology (Euclidean_space m) S) g
           and gf: ⋀x. x ∈ S ⟹ g (f x) = x
           and fg: ⋀y. y ∈ T ⟹ f (g y) = y
```

using *hom S T closedin_subset* **unfolding** *homeomorphic_map_maps homeomorphic_maps_def*
by *fastforce*
obtain f' **where** *cmf'*: *continuous_map* (*Euclidean_space m*) (*Euclidean_space n*) f'
and $f'f$: $\bigwedge x.\ x \in S \Longrightarrow f'\ x = f\ x$
using *iso_Euclidean_complements_lemma1 S cmf continuous_map_into_fulltopology* **by** *metis*
obtain g' **where** *cmg'*: *continuous_map* (*Euclidean_space n*) (*Euclidean_space m*) g'
and $g'g$: $\bigwedge x.\ x \in T \Longrightarrow g'\ x = g\ x$
using *iso_Euclidean_complements_lemma1 T cmg continuous_map_into_fulltopology* **by** *metis*
define p **where** $p \equiv \lambda(x,y).\ (x,(\lambda i.\ y\ i + f'\ x\ i))$
define p' **where** $p' \equiv \lambda(x,y).\ (x,(\lambda i.\ y\ i - f'\ x\ i))$
define q **where** $q \equiv \lambda(x,y).\ (x,(\lambda i.\ y\ i + g'\ x\ i))$
define q' **where** $q' \equiv \lambda(x,y).\ (x,(\lambda i.\ y\ i - g'\ x\ i))$
have *homeomorphic_maps* (*prod_topology* (*Euclidean_space m*) (*Euclidean_space n*))
(*prod_topology* (*Euclidean_space m*) (*Euclidean_space n*))
$p\ p'$
homeomorphic_maps (*prod_topology* (*Euclidean_space n*) (*Euclidean_space m*))
(*prod_topology* (*Euclidean_space n*) (*Euclidean_space m*))
$q\ q'$
homeomorphic_maps (*prod_topology* (*Euclidean_space m*) (*Euclidean_space n*))
(*prod_topology* (*Euclidean_space n*) (*Euclidean_space m*)) $(\lambda(x,y).\ (y,x))\ (\lambda(x,y).\ (y,x))$
apply (*simp_all add*: *p_def p'_def q_def q'_def homeomorphic_maps_def continuous_map_pairwise*)
apply (*force simp*: *case_prod_unfold continuous_map_of_fst* [*unfolded o_def*] *cmf' cmg' intro*: *continuous_intros*)+
done
then have *homeomorphic_maps* (*prod_topology* (*Euclidean_space m*) (*Euclidean_space n*))
(*prod_topology* (*Euclidean_space n*) (*Euclidean_space m*))
$(q' \circ (\lambda(x,y).\ (y,x)) \circ p)\ (p' \circ ((\lambda(x,y).\ (y,x)) \circ q))$
using *homeomorphic_maps_compose homeomorphic_maps_sym* **by** (*metis* (*no_types*, *lifting*))
moreover
have $\bigwedge x.\ x \in S \Longrightarrow (q' \circ (\lambda(x,y).\ (y,x)) \circ p)\ (x,\ \lambda i.\ 0) = (f\ x,\ \lambda i.\ 0)$
apply (*simp add*: *q'_def p_def f'f*)
apply (*simp add*: *fun_eq_iff*)
by (*metis S T closedin_subset g'g gf hom homeomorphic_imp_surjective_map image_eqI topspace_subtopology_subset*)
ultimately
show *thesis*
using *homeomorphic_map_maps that* **by** *blast*

qed

proposition *isomorphic_relative_homology_groups_Euclidean_complements*:
assumes *S*: *closedin* (*Euclidean_space n*) *S* **and** *T*: *closedin* (*Euclidean_space n*) *T*
and *hom*: (*subtopology* (*Euclidean_space n*) *S*) *homeomorphic_space* (*subtopology* (*Euclidean_space n*) *T*)
shows *relative_homology_group p* (*Euclidean_space n*) (*topspace*(*Euclidean_space n*) − *S*)
≅ *relative_homology_group p* (*Euclidean_space n*) (*topspace*(*Euclidean_space n*) − *T*)
proof −
have *subST*: *S* ⊆ *topspace*(*Euclidean_space n*) *T* ⊆ *topspace*(*Euclidean_space n*)
by (*meson S T closedin_def*)+
have *relative_homology_group p* (*Euclidean_space n*) (*topspace* (*Euclidean_space n*) − *S*)
≅ *relative_homology_group* (*p* + *int n*) (*Euclidean_space* (*n* + *n*)) (*topspace* (*Euclidean_space* (*n* + *n*)) − *S*)
using *relative_homology_group_Euclidean_complement_step* [*OF S*] **by** *blast*
moreover have *relative_homology_group p* (*Euclidean_space n*) (*topspace* (*Euclidean_space n*) − *T*)
≅ *relative_homology_group* (*p* + *int n*) (*Euclidean_space* (*n* + *n*)) (*topspace* (*Euclidean_space* (*n* + *n*)) − *T*)
using *relative_homology_group_Euclidean_complement_step* [*OF T*] **by** *blast*
moreover have *relative_homology_group* (*p* + *int n*) (*Euclidean_space* (*n* + *n*)) (*topspace* (*Euclidean_space* (*n* + *n*)) − *S*)
≅ *relative_homology_group* (*p* + *int n*) (*Euclidean_space* (*n* + *n*)) (*topspace* (*Euclidean_space* (*n* + *n*)) − *T*)
proof −
obtain *f* **where** *f*: *homeomorphic_map* (*subtopology* (*Euclidean_space n*) *S*)
(*subtopology* (*Euclidean_space n*) *T*) *f*
using *hom* **unfolding** *homeomorphic_space* **by** *blast*
obtain *g* **where** *g*: *homeomorphic_map* (*prod_topology* (*Euclidean_space n*) (*Euclidean_space n*))
(*prod_topology* (*Euclidean_space n*) (*Euclidean_space n*)) *g*
and *gf*: ⋀*x*. *x* ∈ *S* ⟹ *g*(*x*,(λ*i*. *0*)) = (*f x*,(λ*i*. *0*))
using *S T f iso_Euclidean_complements_lemma2* **by** *blast*
define *h* **where** *h* ≡ λ*x*::*nat* ⇒*real*. ((λ*i*. *if i* < *n then x i else 0*), (λ*j*. *if j* < *n then x*(*n* + *j*) *else 0*))
define *k* **where** *k* ≡ λ(*x*,*y*) *i*. *if i* < *2* ∗ *n then if i* < *n then x i else y*(*i* − *n*) *else* (*0*::*real*)
have *hk*: *homeomorphic_maps* (*Euclidean_space*(*2* ∗ *n*)) (*prod_topology* (*Euclidean_space n*) (*Euclidean_space n*)) *h k*
unfolding *homeomorphic_maps_def*
proof *safe*
show *continuous_map* (*Euclidean_space* (*2* ∗ *n*))

```
                         (prod_topology (Euclidean_space n) (Euclidean_space n)) h
      apply (simp add: h_def continuous_map_pairwise o_def continuous_map_componentwise_Euclide
        unfolding Euclidean_space_def
         by (metis (mono_tags) UNIV_I continuous_map_from_subtopology con-
tinuous_map_product_projection)
      have continuous_map (prod_topology (Euclidean_space n) (Euclidean_space
n)) euclideanreal (λp. fst p i) for i
          using Euclidean_space_def continuous_map_into_fulltopology continu-
ous_map_fst by fastforce
       moreover
      have continuous_map (prod_topology (Euclidean_space n) (Euclidean_space
n)) euclideanreal (λp. snd p (i − n)) for i
          using Euclidean_space_def continuous_map_into_fulltopology continu-
ous_map_snd by fastforce
       ultimately
      show continuous_map (prod_topology (Euclidean_space n) (Euclidean_space
n))
                         (Euclidean_space (2 ∗ n)) k
      by (simp add: k_def continuous_map_pairwise o_def continuous_map_componentwise_Euclidean_
case_prod_unfold)
    qed (auto simp: k_def h_def fun_eq_iff topspace_Euclidean_space)
    define kgh where kgh ≡ k ∘ g ∘ h
   let ?i = hom_induced (p + n) (Euclidean_space(2 ∗ n)) (topspace(Euclidean_space(2
∗ n)) − S)
                              (Euclidean_space(2 ∗ n)) (topspace(Euclidean_space(2
∗ n)) − T) kgh
    have ?i ∈ iso (relative_homology_group (p + int n) (Euclidean_space (2 ∗ n))
                 (topspace (Euclidean_space (2 ∗ n)) − S))
                (relative_homology_group (p + int n) (Euclidean_space (2 ∗ n))
                 (topspace (Euclidean_space (2 ∗ n)) − T))
    proof (rule homeomorphic_map_relative_homology_iso)
      show hm: homeomorphic_map (Euclidean_space (2 ∗ n)) (Euclidean_space
(2 ∗ n)) kgh
        unfolding kgh_def by (meson hk g homeomorphic_map_maps homeomor-
phic_maps_compose homeomorphic_maps_sym)
      have Teq: T = f ` S
      using f homeomorphic_imp_surjective_map subST(1) subST(2) topspace_subtopology_subset
by blast
      have khf: ⋀x. x ∈ S ⟹ k(h(f x)) = f x
      by (metis (no_types, lifting) Teq hk homeomorphic_maps_def image_subset_iff
le_add1 mult_2 subST(2) subsetD subset_Euclidean_space)
      have gh: g(h x) = h(f x) if x ∈ S for x
      proof −
        have [simp]: (λi. if i < n then x i else 0) = x
        using subST(1) that topspace_Euclidean_space by (auto simp: fun_eq_iff)
        have f x ∈ topspace(Euclidean_space n)
          using Teq subST(2) that by blast
        moreover have (λj. if j < n then x (n + j) else 0) = (λj. 0::real)
          using Euclidean_space_def subST(1) that by force
```

```
        ultimately show ?thesis
          by (simp add: topspace_Euclidean_space h_def gf ‹x ∈ S› fun_eq_iff)
      qed
      have *: ⟦S ⊆ U; T ⊆ U; kgh ‘ U = U; inj_on kgh U; kgh ‘ S = T⟧ ⟹ kgh
‘ (U − S) = U − T for U
        unfolding inj_on_def set_eq_iff by blast
    show kgh ‘ (topspace (Euclidean_space (2 ∗ n)) − S) = topspace (Euclidean_space
(2 ∗ n)) − T
      proof (rule ∗)
        show kgh ‘ topspace (Euclidean_space (2 ∗ n)) = topspace (Euclidean_space
(2 ∗ n))
          by (simp add: hm homeomorphic_imp_surjective_map)
        show inj_on kgh (topspace (Euclidean_space (2 ∗ n)))
          using hm homeomorphic_map_def by auto
        show kgh ‘ S = T
          by (simp add: Teq kgh_def gh khf)
      qed (use subST topspace_Euclidean_space in ‹fastforce+›)
    qed auto
    then show ?thesis
      by (simp add: is_isoI mult_2)
  qed
  ultimately show ?thesis
    by (meson group.iso_sym iso_trans group_relative_homology_group)
qed

lemma lemma_iod:
  assumes S ⊆ T S ≠ {} and Tsub: T ⊆ topspace(Euclidean_space n)
      and S: ⋀a b u. ⟦a ∈ S; b ∈ T; 0 < u; u < 1⟧ ⟹ (λi. (1 − u) ∗ a i + u ∗
b i) ∈ S
    shows path_connectedin (Euclidean_space n) T
proof −
  obtain a where a ∈ S
    using assms by blast
  have path_component_of (subtopology (Euclidean_space n) T) a b if b ∈ T for
b
    unfolding path_component_of_def
  proof (intro exI conjI)
    have [simp]: ∀ i≥n. a i = 0
      using Tsub ‹a ∈ S› assms(1) topspace_Euclidean_space by auto
    have [simp]: ∀ i≥n. b i = 0
      using Tsub that topspace_Euclidean_space by auto
    have inT: (λi. (1 − x) ∗ a i + x ∗ b i) ∈ T if 0 ≤ x x ≤ 1 for x
    proof (cases x = 0 ∨ x = 1)
      case True
      with ‹a ∈ S› ‹b ∈ T› ‹S ⊆ T› show ?thesis
        by force
    next
      case False
      then show ?thesis
```

using *subsetD* [*OF* ‹*S* ⊆ *T*› *S*] ‹*a* ∈ *S*› ‹*b* ∈ *T*› *that* **by** *auto*
qed
have *continuous_on* {*0..1*} (λ*x*. (*1* − *x*) ∗ *a k* + *x* ∗ *b k*) **for** *k*
by (*intro continuous_intros*)
then show *pathin* (*subtopology* (*Euclidean_space n*) *T*) (λ*t i*. (*1* − *t*) ∗ *a i* + *t* ∗ *b i*)
apply (*simp add*: *Euclidean_space_def subtopology_subtopology pathin_subtopology*)
apply (*simp add*: *pathin_def continuous_map_componentwise_UNIV inT*)
done
qed *auto*
then have *path_connected_space* (*subtopology* (*Euclidean_space n*) *T*)
by (*metis Tsub path_component_of_equiv path_connected_space_iff_path_component topspace_subtopology_subset*)
then show *?thesis*
by (*simp add*: *Tsub path_connectedin_def*)
qed

lemma *invariance_of_dimension_closedin_Euclidean_space*:
assumes *closedin* (*Euclidean_space n*) *S*
shows *subtopology* (*Euclidean_space n*) *S homeomorphic_space Euclidean_space n*
⟷ *S* = *topspace*(*Euclidean_space n*)
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
have *Ssub*: *S* ⊆ *topspace* (*Euclidean_space n*)
by (*meson assms closedin_def*)
moreover have *False* **if** *a* ∉ *S* **and** *a* ∈ *topspace* (*Euclidean_space n*) **for** *a*
proof −
have *cl_n*: *closedin* (*Euclidean_space* (*Suc n*)) (*topspace*(*Euclidean_space n*))
using *Euclidean_space_def closedin_Euclidean_space closedin_subtopology* **by** *fastforce*
then have *sub*: *subtopology* (*Euclidean_space*(*Suc n*)) (*topspace*(*Euclidean_space n*)) = *Euclidean_space n*
by (*metis* (*no_types*, *lifting*) *Euclidean_space_def closedin_subset subtopology_subtopology topspace_Euclidean_space topspace_subtopology topspace_subtopology_subset*)
then have *cl_S*: *closedin* (*Euclidean_space*(*Suc n*)) *S*
using *cl_n assms closedin_closed_subtopology* **by** *fastforce*
have *sub_SucS*: *subtopology* (*Euclidean_space* (*Suc n*)) *S* = *subtopology* (*Euclidean_space n*) *S*
by (*metis Ssub sub subtopology_subtopology topspace_subtopology topspace_subtopology_subset*)
have *non0*: {*y*. ∃ *x*::*nat*⇒*real*. (∀ *i*≥*Suc n*. *x i* = *0*) ∧ (∃ *i*≥*n*. *x i* ≠ *0*) ∧ *y* = *x n*} = −{*0*}
proof *safe*
show *False* **if** ∀ *i*≥*Suc n*. *f i* = *0 0* = *f n n* ≤ *i f i* ≠ *0* **for** *f*::*nat*⇒*real* **and** *i*
by (*metis that le_antisym not_less_eq_eq*)
show ∃ *f*::*nat*⇒*real*. (∀ *i*≥*Suc n*. *f i* = *0*) ∧ (∃ *i*≥*n*. *f i* ≠ *0*) ∧ *a* = *f n* **if** *a* ≠ *0* **for** *a*

by (*rule_tac x*=(λ*i*. *0*)(*n*:= *a*) **in** *exI*) (*force simp*: *that*)
qed
have *homology_group 0* (*subtopology* (*Euclidean_space* (*Suc n*)) (*topspace* (*Euclidean_space* (*Suc n*)) − *S*))
≅ *homology_group 0* (*subtopology* (*Euclidean_space* (*Suc n*)) (*topspace* (*Euclidean_space* (*Suc n*)) − *topspace* (*Euclidean_space n*)))
proof (*rule isomorphic_relative_contractible_space_imp_homology_groups*)
show (*topspace* (*Euclidean_space* (*Suc n*)) − *S* = {}) =
(*topspace* (*Euclidean_space* (*Suc n*)) − *topspace* (*Euclidean_space n*) = {})
using *cl_n closedin_subset that* **by** *auto*
next
fix *p*
show *relative_homology_group p* (*Euclidean_space* (*Suc n*))
(*topspace* (*Euclidean_space* (*Suc n*)) − *S*) ≅
relative_homology_group p (*Euclidean_space* (*Suc n*))
(*topspace* (*Euclidean_space* (*Suc n*)) − *topspace* (*Euclidean_space n*))
by (*simp add*: *L sub_SucS cl_S cl_n isomorphic_relative_homology_groups_Euclidean_complements sub*)
qed (*auto simp*: *L*)
moreover
have *continuous_map* (*powertop_real UNIV*) *euclideanreal* (λ*x*. *x n*)
by (*metis* (*no_types*) *UNIV_I continuous_map_product_projection*)
then have *cm*: *continuous_map* (*subtopology* (*Euclidean_space* (*Suc n*)) (*topspace* (*Euclidean_space* (*Suc n*)) − *topspace* (*Euclidean_space n*)))
euclideanreal (λ*x*. *x n*)
by (*simp add*: *Euclidean_space_def continuous_map_from_subtopology*)
have *False* **if** *path_connected_space*
(*subtopology* (*Euclidean_space* (*Suc n*))
(*topspace* (*Euclidean_space* (*Suc n*)) − *topspace* (*Euclidean_space n*)))
using *path_connectedin_continuous_map_image* [*OF cm that* [*unfolded path_connectedin_topspace* [*symmetric*]]]
bounded_path_connected_Compl_real [*of* {*0*}]
by (*simp add*: *topspace_Euclidean_space image_def Bex_def non0 flip*: *path_connectedin_topspace*)
moreover
have *eq*: *T* = *T* ∩ {*x*. *x n* ≤ *0*} ∪ *T* ∩ {*x*. *x n* ≥ *0*} **for** *T* :: (*nat* ⇒ *real*) *set*
by *auto*
have *path_connectedin* (*Euclidean_space* (*Suc n*)) (*topspace* (*Euclidean_space* (*Suc n*)) − *S*)
proof (*subst eq*, *rule path_connectedin_Un*)
have *topspace*(*Euclidean_space*(*Suc n*)) ∩ {*x*. *x n* = *0*} = *topspace*(*Euclidean_space n*)
apply (*auto simp*: *topspace_Euclidean_space*)
by (*metis Suc_leI inf.absorb_iff2 inf.orderE leI*)
let *?S* = *topspace*(*Euclidean_space*(*Suc n*)) ∩ {*x*. *x n* < *0*}
show *path_connectedin* (*Euclidean_space* (*Suc n*))
((*topspace* (*Euclidean_space* (*Suc n*)) − *S*) ∩ {*x*. *x n* ≤ *0*})

```
      proof (rule lemma_iod)
        show ?S ⊆ (topspace (Euclidean_space (Suc n)) − S) ∩ {x. x n ≤ 0}
          using Ssub topspace_Euclidean_space by auto
        show ?S ≠ {}
          apply (simp add: topspace_Euclidean_space set_eq_iff)
          apply (rule_tac x=(λi. 0)(n:= −1) in exI)
          apply auto
          done
        fix a b and u::real
        assume
          a ∈ ?S 0 < u u < 1
          b ∈ (topspace (Euclidean_space (Suc n)) − S) ∩ {x. x n ≤ 0}
        then show (λi. (1 − u) ∗ a i + u ∗ b i) ∈ ?S
       by (simp add: topspace_Euclidean_space add_neg_nonpos less_eq_real_def
mult_less_0_iff)
      qed (simp add: topspace_Euclidean_space subset_iff)
      let ?T = topspace(Euclidean_space(Suc n)) ∩ {x. x n > 0}
      show path_connectedin (Euclidean_space (Suc n))
              ((topspace (Euclidean_space (Suc n)) − S) ∩ {x. 0 ≤ x n})
      proof (rule lemma_iod)
        show ?T ⊆ (topspace (Euclidean_space (Suc n)) − S) ∩ {x. 0 ≤ x n}
          using Ssub topspace_Euclidean_space by auto
        show ?T ≠ {}
          apply (simp add: topspace_Euclidean_space set_eq_iff)
          apply (rule_tac x=(λi. 0)(n:= 1) in exI)
          apply auto
          done
        fix a b and u::real
        assume  a ∈ ?T 0 < u u < 1 b ∈ (topspace (Euclidean_space (Suc n)) −
S) ∩ {x. 0 ≤ x n}
        then show (λi. (1 − u) ∗ a i + u ∗ b i) ∈ ?T
          by (simp add: topspace_Euclidean_space add_pos_nonneg)
      qed (simp add: topspace_Euclidean_space subset_iff)
      show (topspace (Euclidean_space (Suc n)) − S) ∩ {x. x n ≤ 0} ∩
            ((topspace (Euclidean_space (Suc n)) − S) ∩ {x. 0 ≤ x n}) ≠ {}
        using that
        apply (auto simp: Set.set_eq_iff topspace_Euclidean_space)
        by (metis Suc_leD order_refl)
    qed
    then have path_connected_space (subtopology (Euclidean_space (Suc n))
                                     (topspace (Euclidean_space (Suc n)) − S))
     apply (simp add: path_connectedin_subtopology flip: path_connectedin_topspace)
      by (metis Int_Diff inf_idem)
    ultimately
    show ?thesis
      using isomorphic_homology_imp_path_connectedness by blast
  qed
  ultimately show ?rhs
    by blast
```

qed (*simp add*: *homeomorphic_space_refl*)

lemma *isomorphic_homology_groups_Euclidean_complements*:
assumes *closedin* (*Euclidean_space n*) *S closedin* (*Euclidean_space n*) *T*
(*subtopology* (*Euclidean_space n*) *S*) *homeomorphic_space* (*subtopology* (*Euclidean_space n*) *T*)
shows *homology_group p* (*subtopology* (*Euclidean_space n*) (*topspace*(*Euclidean_space n*) − *S*))
≅ *homology_group p* (*subtopology* (*Euclidean_space n*) (*topspace*(*Euclidean_space n*) − *T*))
proof (*rule isomorphic_relative_contractible_space_imp_homology_groups*)
show *topspace* (*Euclidean_space n*) − *S* ⊆ *topspace* (*Euclidean_space n*)
using *assms homeomorphic_space_sym invariance_of_dimension_closedin_Euclidean_space subtopology_superset* **by** *fastforce*
show *topspace* (*Euclidean_space n*) − *T* ⊆ *topspace* (*Euclidean_space n*)
using *assms invariance_of_dimension_closedin_Euclidean_space subtopology_superset* **by** *force*
show (*topspace* (*Euclidean_space n*) − *S* = {}) = (*topspace* (*Euclidean_space n*) − *T* = {})
by (*metis Diff_eq_empty_iff assms closedin_subset homeomorphic_space_sym invariance_of_dimension_closedin_Euclidean_space subset_antisym subtopology_topspace*)
show *relative_homology_group p* (*Euclidean_space n*) (*topspace* (*Euclidean_space n*) − *S*) ≅
relative_homology_group p (*Euclidean_space n*) (*topspace* (*Euclidean_space n*) − *T*) **for** *p*
using *assms isomorphic_relative_homology_groups_Euclidean_complements* **by** *blast*
qed *auto*

lemma *eqpoll_path_components_Euclidean_complements*:
assumes *closedin* (*Euclidean_space n*) *S closedin* (*Euclidean_space n*) *T*
(*subtopology* (*Euclidean_space n*) *S*) *homeomorphic_space* (*subtopology* (*Euclidean_space n*) *T*)
shows *path_components_of*
(*subtopology* (*Euclidean_space n*)
(*topspace*(*Euclidean_space n*) − *S*))
≈ *path_components_of*
(*subtopology* (*Euclidean_space n*)
(*topspace*(*Euclidean_space n*) − *T*))
by (*simp add*: *assms isomorphic_homology_groups_Euclidean_complements isomorphic_homology_imp_path_components*)

lemma *path_connectedin_Euclidean_complements*:
assumes *closedin* (*Euclidean_space n*) *S closedin* (*Euclidean_space n*) *T*
(*subtopology* (*Euclidean_space n*) *S*) *homeomorphic_space* (*subtopology* (*Euclidean_space n*) *T*)
shows *path_connectedin* (*Euclidean_space n*) (*topspace*(*Euclidean_space n*) −

S)
⟷ path_connectedin (Euclidean_space n) (topspace(Euclidean_space n)
− T)
by (meson Diff_subset assms isomorphic_homology_groups_Euclidean_complements
isomorphic_homology_imp_path_connectedness path_connectedin_def)

lemma eqpoll_connected_components_Euclidean_complements:
assumes S: closedin (Euclidean_space n) S **and** T: closedin (Euclidean_space
n) T
and ST: (subtopology (Euclidean_space n) S) homeomorphic_space (subtopology
(Euclidean_space n) T)
shows connected_components_of
(subtopology (Euclidean_space n)
(topspace(Euclidean_space n) − S))
≈ connected_components_of
(subtopology (Euclidean_space n)
(topspace(Euclidean_space n) − T))
using eqpoll_path_components_Euclidean_complements [OF assms]
by (metis S T closedin_def locally_path_connected_Euclidean_space locally_path_connected_space_op
path_components_eq_connected_components_of)

lemma connected_in_Euclidean_complements:
assumes closedin (Euclidean_space n) S closedin (Euclidean_space n) T
(subtopology (Euclidean_space n) S) homeomorphic_space (subtopology
(Euclidean_space n) T)
shows connectedin (Euclidean_space n) (topspace(Euclidean_space n) − S)
⟷ connectedin (Euclidean_space n) (topspace(Euclidean_space n) − T)
apply (simp add: connectedin_def connected_space_iff_components_subset_singleton
subset_singleton_iff_lepoll)
using eqpoll_connected_components_Euclidean_complements [OF assms]
by (meson eqpoll_sym lepoll_trans1)

theorem invariance_of_dimension_Euclidean_space:
Euclidean_space m homeomorphic_space Euclidean_space n ⟷ m = n
proof (cases m n rule: linorder_cases)
case less
then have ∗: topspace (Euclidean_space m) ⊆ topspace (Euclidean_space n)
by (meson le_cases not_le subset_Euclidean_space)
then have Euclidean_space m = subtopology (Euclidean_space n) (topspace(Euclidean_space
m))
by (simp add: Euclidean_space_def inf.absorb_iff2 subtopology_subtopology)
then show ?thesis
by (metis (no_types, lifting) ∗ Euclidean_space_def closedin_Euclidean_space
closedin_closed_subtopology eq_iff invariance_of_dimension_closedin_Euclidean_space
subset_Euclidean_space topspace_Euclidean_space)
next
case equal
then show ?thesis

```
    by (simp add: homeomorphic_space_refl)
next
  case greater
  then have *: topspace (Euclidean_space n) ⊆ topspace (Euclidean_space m)
    by (meson le_cases not_le subset_Euclidean_space)
  then have Euclidean_space n = subtopology (Euclidean_space m) (topspace(Euclidean_space
n))
    by (simp add: Euclidean_space_def inf.absorb_iff2 subtopology_subtopology)
  then show ?thesis
    by (metis (no_types, lifting) * Euclidean_space_def closedin_Euclidean_space
closedin_closed_subtopology eq_iff homeomorphic_space_sym invariance_of_dimension_closedin_Euclidean_spac
subset_Euclidean_space topspace_Euclidean_space)
qed


lemma biglemma:
  assumes n ≠ 0 and S: compactin (Euclidean_space n) S
    and cmh: continuous_map (subtopology (Euclidean_space n) S) (Euclidean_space
n) h
      and inj_on h S
    shows path_connectedin (Euclidean_space n) (topspace(Euclidean_space n) −
h ‘ S)
      ⟷ path_connectedin (Euclidean_space n) (topspace(Euclidean_space n) −
S)
proof (rule path_connectedin_Euclidean_complements)
  have hS_sub: h ‘ S ⊆ topspace(Euclidean_space n)
   by (metis (no_types) S cmh compactin_subspace continuous_map_image_subset_topspace
topspace_subtopology_subset)
  show clo_S: closedin (Euclidean_space n) S
   using assms by (simp add: continuous_map_in_subtopology Hausdorff_Euclidean_space
compactin_imp_closedin)
  show clo_hS: closedin (Euclidean_space n) (h ‘ S)
   using Hausdorff_Euclidean_space S cmh compactin_absolute compactin_imp_closedin
image_compactin by blast
  have homeomorphic_map (subtopology (Euclidean_space n) S) (subtopology (Euclidean_space
n) (h ‘ S)) h
  proof (rule continuous_imp_homeomorphic_map)
    show compact_space (subtopology (Euclidean_space n) S)
      by (simp add: S compact_space_subtopology)
    show Hausdorff_space (subtopology (Euclidean_space n) (h ‘ S))
      using hS_sub
      by (simp add: Hausdorff_Euclidean_space Hausdorff_space_subtopology)
   show continuous_map (subtopology (Euclidean_space n) S) (subtopology (Euclidean_space
n) (h ‘ S)) h
      using cmh continuous_map_in_subtopology by fastforce
    show h ‘ topspace (subtopology (Euclidean_space n) S) = topspace (subtopology
(Euclidean_space n) (h ‘ S))
      using clo_hS clo_S closedin_subset by auto
```

show *inj_on h* (*topspace* (*subtopology* (*Euclidean_space n*) *S*))
by (*metis* ‹*inj_on h S*› *clo_S closedin_def topspace_subtopology_subset*)
qed
then show *subtopology* (*Euclidean_space n*) (*h* ‘ *S*) *homeomorphic_space subtopology* (*Euclidean_space n*) *S*
using *homeomorphic_space homeomorphic_space_sym* **by** *blast*
qed

lemma *lemmaIOD*:
assumes
$\exists T.\ T \in U \wedge c \subseteq T$ $\exists T.\ T \in U \wedge d \subseteq T$ $\bigcup U = c \cup d$ $\bigwedge T.\ T \in U \Longrightarrow T \neq \{\}$
pairwise disjnt U ~($\exists T.\ U \subseteq \{T\}$)
shows $c \in U$
using *assms*
apply *safe*
subgoal for C' D'
proof (*cases* $C'=D'$)
show $c \in U$
if *UU*: $\bigcup U = c \cup d$
and *U*: $\bigwedge T.\ T \in U \Longrightarrow T \neq \{\}$ *disjoint U* **and** $\nexists T.\ U \subseteq \{T\}$ $c \subseteq C'$ $D' \in U$ $d \subseteq D'$ $C' = D'$
proof −
have $c \cup d = D'$
using *Union_upper sup_mono UU that*(*5*) *that*(*6*) *that*(*7*) *that*(*8*) **by** *auto*
then have $\bigcup U = D'$
by (*simp add*: *UU*)
with *U* **have** $U = \{D'\}$
by (*metis* (*no_types*, *lifting*) *disjnt_Union1 disjnt_self_iff_empty insertCI pairwiseD subset_iff that*(*4*) *that*(*6*))
then show *?thesis*
using *that*(*4*) **by** *auto*
qed
show $c \in U$
if $\bigcup U = c \cup d$*disjoint U* $C' \in U$ $c \subseteq C'$$D' \in U$ $d \subseteq D'$ $C' \neq D'$
proof −
have $C' \cap D' = \{\}$
using ‹*disjoint U*› ‹$C' \in U$› ‹$D' \in U$› ‹$C' \neq D'$›**unfolding** *disjnt_iff pairwise_def*
by *blast*
then show *?thesis*
using *subset_antisym that*(*1*) ‹$C' \in U$› ‹$c \subseteq C'$› ‹$d \subseteq D'$› **by** *fastforce*
qed
qed
done

```
theorem invariance_of_domain_Euclidean_space:
  assumes U: openin (Euclidean_space n) U
   and cmf: continuous_map (subtopology (Euclidean_space n) U) (Euclidean_space
n) f
    and inj_on f U
  shows openin (Euclidean_space n) (f ‘ U)   (is openin ?E (f ‘ U))
proof (cases n = 0)
  case True
  have [simp]: Euclidean_space 0 = discrete_topology {λi. 0}
   by (auto simp: subtopology_eq_discrete_topology_sing topspace_Euclidean_space)
  show ?thesis
    using cmf True U by auto
next
  case False
  define enorm where enorm ≡ λx. sqrt(∑ i<n. x i ^ 2)
   have enorm_if [simp]: enorm (λi. if i = k then d else 0) = (if k < n then |d|
else 0) for k d
    using ‹n ≠ 0› by (auto simp: enorm_def power2_eq_square if_distrib [of λx.
x ∗ _] cong: if_cong)
  define zero::nat⇒real where zero ≡ λi. 0
  have zero_in [simp]: zero ∈ topspace ?E
    using False by (simp add: zero_def topspace_Euclidean_space)
  have enorm_eq_0 [simp]: enorm x = 0 ⟷ x = zero
    if x ∈ topspace(Euclidean_space n) for x
    using that unfolding zero_def enorm_def
   apply (simp add: sum_nonneg_eq_0_iff fun_eq_iff topspace_Euclidean_space)
    using le_less_linear by blast
  have [simp]: enorm zero = 0
    by (simp add: zero_def enorm_def)
  have cm_enorm: continuous_map ?E euclideanreal enorm
    unfolding enorm_def
  proof (intro continuous_intros)
    show continuous_map ?E euclideanreal (λx. x i)
      if i ∈ {..<n} for i
     using that by (auto simp: Euclidean_space_def intro: continuous_map_product_projection
continuous_map_from_subtopology)
  qed auto
  have enorm_ge0: 0 ≤ enorm x for x
    by (auto simp: enorm_def sum_nonneg)
  have le_enorm: |x i| ≤ enorm x if i < n for i x
  proof −
    have |x i| ≤ sqrt (∑ k∈{i}. (x k)²)
      by auto
    also have ... ≤ sqrt (∑ k<n. (x k)²)
      by (rule real_sqrt_le_mono [OF sum_mono2]) (use that in auto)
    finally show ?thesis
      by (simp add: enorm_def)
  qed
```

define *B* **where** $B \equiv \lambda r.\ \{x \in topspace\ ?E.\ enorm\ x < r\}$
define *C* **where** $C \equiv \lambda r.\ \{x \in topspace\ ?E.\ enorm\ x \le r\}$
define *S* **where** $S \equiv \lambda r.\ \{x \in topspace\ ?E.\ enorm\ x = r\}$
have *BC*: $B\ r \subseteq C\ r$ **and** *SC*: $S\ r \subseteq C\ r$ **and** *disjSB*: *disjnt* (*S r*) (*B r*) **and** *eqC*: $B\ r \cup S\ r = C\ r$ **for** *r*
by (*auto simp*: *B_def C_def S_def disjnt_def*)
consider $n = 1 \mid n \ge 2$
using *False* **by** *linarith*
then have ∗∗: *openin ?E* (*h* ‘ (*B r*))
if $r > 0$ **and** *cmh*: *continuous_map*(*subtopology ?E* (*C r*)) *?E h* **and** *injh*: *inj_on h* (*C r*) **for** *r h*
proof *cases*
case *1*
define *e* :: [*real*,*nat*]⇒*real* **where** $e \equiv \lambda x\ i.$ *if* $i = 0$ *then x else 0*
define *e′* :: (*nat*⇒*real*)⇒*real* **where** $e' \equiv \lambda x.\ x\ 0$
have *continuous_map euclidean euclideanreal* (λ*f*. *f* (*0*::*nat*))
by *auto*
then have *continuous_map* (*subtopology* (*powertop_real UNIV*) {*f*. ∀ *n*≥*Suc 0*. *f n* = *0*}) *euclideanreal* (λ*f*. *f 0*)
by (*metis* (*mono_tags*) *continuous_map_from_subtopology euclidean_product_topology*)
then have *hom_ee′*: *homeomorphic_maps euclideanreal* (*Euclidean_space 1*) *e e′*
by (*auto simp*: *homeomorphic_maps_def e_def e′_def continuous_map_in_subtopology Euclidean_space_def*)
have *eBr*: *e* ‘ {−*r*<..<*r*} = *B r*
unfolding *B_def e_def C_def*
by(*force simp*: *1 topspace_Euclidean_space enorm_def power2_eq_square if_distrib* [*of* λ*x*. *x* ∗ _] *cong*: *if_cong*)
have *in_Cr*: ⋀*x*. ⟦−*r* < *x*; *x* < *r*⟧ ⟹ (λ*i*. *if i* = *0 then x else 0*) ∈ *C r*
using ‹*n* ≠ *0*› **by** (*auto simp*: *C_def topspace_Euclidean_space*)
have *inj*: *inj_on* (*e′* ∘ *h* ∘ *e*) {− *r*<..<*r*}
proof (*clarsimp simp*: *inj_on_def e_def e′_def*)
show (*x*::*real*) = *y*
if *f*: *h* (λ*i*. *if i* = *0 then x else 0*) *0* = *h* (λ*i*. *if i* = *0 then y else 0*) *0*
and −*r* < *x x* < *r* −*r* < *y y* < *r*
for *x y* :: *real*
proof −
have *x*: (λ*i*. *if i* = *0 then x else 0*) ∈ *C r* **and** *y*: (λ*i*. *if i* = *0 then y else 0*) ∈ *C r*
by (*blast intro*: *inj_onD* [*OF* ‹*inj_on h* (*C r*)›] *that in_Cr*)+
have *continuous_map* (*subtopology* (*Euclidean_space* (*Suc 0*)) (*C r*)) (*Euclidean_space* (*Suc 0*)) *h*
using *cmh* **by** (*simp add*: *1*)
then have *h* ‘ ({*x*. ∀ *i*≥*Suc 0*. *x i* = *0*} ∩ *C r*) ⊆ {*x*. ∀ *i*≥*Suc 0*. *x i* = *0*}
by (*force simp*: *Euclidean_space_def subtopology_subtopology continuous_map_def*)
have *h* (λ*i*. *if i* = *0 then x else 0*) *j* = *h* (λ*i*. *if i* = *0 then y else 0*) *j* **for** *j*
proof (*cases j*)
case (*Suc j′*)

```
          have h ` ({x. ∀ i≥Suc 0. x i = 0} ∩ C r) ⊆ {x. ∀ i≥Suc 0. x i = 0}
            using continuous_map_image_subset_topspace [OF cmh]
            by (simp add: 1 Euclidean_space_def subtopology_subtopology)
          with Suc f x y show ?thesis
            by (simp add: 1 image_subset_iff)
        qed (use f in blast)
        then have (λi. if i = 0 then x else 0) = (λi::nat. if i = 0 then y else 0)
          by (blast intro: inj_onD [OF ‹inj_on h (C r)›] that in_Cr)
        then show ?thesis
          by (simp add: fun_eq_iff) presburger
      qed
    qed
    have hom_e': homeomorphic_map (Euclidean_space 1) euclideanreal e'
      using hom_ee' homeomorphic_maps_map by blast
    have openin (Euclidean_space n) (h ` e ` {− r<..<r})
      unfolding 1
    proof (subst homeomorphic_map_openness [OF hom_e', symmetric])
      show hesub: h ` e ` {− r<..<r} ⊆ topspace (Euclidean_space 1)
      using 1 C_def ‹⋀r. B r ⊆ C r› cmh continuous_map_image_subset_topspace
eBr by fastforce
      have cont: continuous_on {− r<..<r} (e' ∘ h ∘ e)
      proof (intro continuous_on_compose)
        have ⋀i. continuous_on {− r<..<r} (λx. if i = 0 then x else 0)
          by (auto simp: continuous_on_topological)
        then show continuous_on {− r<..<r} e
         by (force simp: e_def intro: continuous_on_coordinatewise_then_product)
        have subCr: e ` {− r<..<r} ⊆ topspace (subtopology ?E (C r))
          by (auto simp: eBr ‹⋀r. B r ⊆ C r›) (auto simp: B_def)
        with cmh show continuous_on (e ` {− r<..<r}) h
       by (meson cm_Euclidean_space_iff_continuous_on continuous_on_subset)
         have continuous_on (topspace ?E) e'
           by (metis 1 continuous_map_Euclidean_space_iff hom_ee' homeomor-
phic_maps_def)
        then show continuous_on (h ` e ` {− r<..<r}) e'
          using hesub by (simp add: 1 e'_def continuous_on_subset)
      qed
      show openin euclideanreal (e' ` h ` e ` {− r<..<r})
          using injective_eq_1d_open_map_UNIV [OF cont] inj by (simp add:
image_image is_interval_1)
    qed
    then show ?thesis
      by (simp flip: eBr)
  next
    case 2
    have cloC: ⋀r. closedin (Euclidean_space n) (C r)
      unfolding C_def
     by (rule closedin_continuous_map_preimage [OF cm_enorm, of concl: {.._},
simplified])
    have cloS: ⋀r. closedin (Euclidean_space n) (S r)
```

unfolding *S_def*
by (*rule closedin_continuous_map_preimage* [*OF cm_enorm*, *of concl*: {_}, *simplified*])
have *C_subset*: $C\ r \subseteq UNIV \rightarrow_E \{-|r|..|r|\}$
using *le_enorm* ‹$r > 0$›
apply (*auto simp*: *C_def topspace_Euclidean_space abs_le_iff*)
apply (*metis add.inverse_neutral le_cases less_minus_iff not_le order_trans*)
by (*metis enorm_ge0 not_le order.trans*)
have *compactinC*: *compactin* (*Euclidean_space n*) (*C r*)
unfolding *Euclidean_space_def compactin_subtopology*
proof
show *compactin* (*powertop_real UNIV*) (*C r*)
proof (*rule closed_compactin* [*OF _ C_subset*])
show *closedin* (*powertop_real UNIV*) (*C r*)
by (*metis Euclidean_space_def cloC closedin_Euclidean_space closedin_closed_subtopology topspace_Euclidean_space*)
qed (*simp add*: *compactin_PiE*)
qed (*auto simp*: *C_def topspace_Euclidean_space*)
have *compactinS*: *compactin* (*Euclidean_space n*) (*S r*)
unfolding *Euclidean_space_def compactin_subtopology*
proof
show *compactin* (*powertop_real UNIV*) (*S r*)
proof (*rule closed_compactin*)
show $S\ r \subseteq UNIV \rightarrow_E \{-|r|..|r|\}$
using *C_subset* ‹$\bigwedge r.\ S\ r \subseteq C\ r$› **by** *blast*
show *closedin* (*powertop_real UNIV*) (*S r*)
by (*metis Euclidean_space_def cloS closedin_Euclidean_space closedin_closed_subtopology topspace_Euclidean_space*)
qed (*simp add*: *compactin_PiE*)
qed (*auto simp*: *S_def topspace_Euclidean_space*)
have *h_if_B*: $\bigwedge y.\ y \in B\ r \Longrightarrow h\ y \in$ *topspace ?E*
using *B_def* ‹$\bigwedge r.\ B\ r \cup S\ r = C\ r$› *cmh continuous_map_image_subset_topspace* **by** *fastforce*
have *com_hSr*: *compactin* (*Euclidean_space n*) (*h ‘ S r*)
by (*meson* ‹$\bigwedge r.\ S\ r \subseteq C\ r$› *cmh compactinS compactin_subtopology image_compactin*)
have *ope_comp_hSr*: *openin* (*Euclidean_space n*) (*topspace* (*Euclidean_space n*) − *h ‘ S r*)
proof (*rule openin_diff*)
show *closedin* (*Euclidean_space n*) (*h ‘ S r*)
using *Hausdorff_Euclidean_space com_hSr compactin_imp_closedin* **by** *blast*
qed *auto*
have *h_pcs*: *h ‘* (*B r*) ∈ *path_components_of* (*subtopology ?E* (*topspace ?E* − *h ‘* (*S r*)))
proof (*rule lemmaIOD*)
have *pc_interval*: *path_connectedin* (*Euclidean_space n*) {$x \in$ *topspace*(*Euclidean_space n*). *enorm* $x \in T$}
if *T*: *is_interval T* **for** *T*

```
    proof -
      define mul :: [real, nat ⇒ real, nat] ⇒ real where mul ≡ λa x i. a * x i
      let ?neg = mul (-1)
      have neg_neg [simp]: ?neg (?neg x) = x for x
        by (simp add: mul_def)
      have enorm_mul [simp]: enorm(mul a x) = abs a * enorm x for a x
     by (simp add: enorm_def mul_def power_mult_distrib) (metis real_sqrt_abs
real_sqrt_mult sum_distrib_left)
      have mul_in_top: mul a x ∈ topspace ?E
          if x ∈ topspace ?E for a x
        using mul_def that topspace_Euclidean_space by auto
      have neg_in_S: ?neg x ∈ S r
          if x ∈ S r for x r
       using that topspace_Euclidean_space S_def by simp (simp add: mul_def)
      have *: path_connectedin ?E (S d)
        if d ≥ 0 for d
      proof (cases d = 0)
        let ?ES = subtopology ?E (S d)
        case False
        then have d > 0
          using that by linarith
        moreover have path_connected_space ?ES
          unfolding path_connected_space_iff_path_component
        proof clarify
          have **: path_component_of ?ES x y
            if x: x ∈ topspace ?ES and y: y ∈ topspace ?ES x ≠ ?neg y for x y
          proof -
            show ?thesis
              unfolding path_component_of_def pathin_def S_def
            proof (intro exI conjI)
              let ?g = (λx. mul (d / enorm x) x) ∘ (λt i. (1 - t) * x i + t * y i)
              show continuous_map (top_of_set {0::real..1}) (subtopology ?E {x
∈ topspace ?E. enorm x = d}) ?g
              proof (rule continuous_map_compose)
                let ?Y = subtopology ?E (- {zero})
                have **: False
                  if eq0: ⋀j. (1 - r) * x j + r * y j = 0
                    and ne: x i ≠ - y i
                    and d: enorm x = d enorm y = d
                    and r: 0 ≤ r r ≤ 1
                  for i r
                proof -
                  have mul (1-r) x = ?neg (mul r y)
                    using eq0 by (simp add: mul_def fun_eq_iff algebra_simps)
                  then have enorm (mul (1-r) x) = enorm (?neg (mul r y))
                    by metis
                  with r have (1-r) * enorm x = r * enorm y
                    by simp
                  then have r12: r = 1/2
```

using ‹*d* ≠ *0*› *d* **by** *auto*
show *?thesis*
using *ne eq0* [*of i*] **unfolding** *r12* **by** (*simp add*: *algebra_simps*)
qed
show *continuous_map* (*top_of_set* {*0..1*}) *?Y* (λ*t i*. (*1* − *t*) ∗ *x i* + *t* ∗ *y i*)
using *x y*
unfolding *continuous_map_componentwise_UNIV Euclidean_space_def continuous_map_in_subtopology*
apply (*intro conjI allI continuous_intros*)
apply (*auto simp*: *zero_def mul_def S_def Euclidean_space_def fun_eq_iff*)
using ∗∗ **by** *blast*
have *cm_enorm'*: *continuous_map* (*subtopology* (*powertop_real UNIV*) *A*) *euclideanreal enorm* **for** *A*
unfolding *enorm_def* **by** (*intro continuous_intros*) *auto*
have *continuous_map ?Y* (*subtopology ?E* {*x*. *enorm x* = *d*}) (λ*x*. *mul* (*d* / *enorm x*) *x*)
unfolding *continuous_map_in_subtopology*
proof (*intro conjI*)
show *continuous_map ?Y* (*Euclidean_space n*) (λ*x*. *mul* (*d* / *enorm x*) *x*)
unfolding *continuous_map_in_subtopology Euclidean_space_def mul_def zero_def subtopology_subtopology continuous_map_componentwise_UNIV*
proof (*intro conjI allI cm_enorm' continuous_intros*)
show *enorm x* ≠ *0*
if *x* ∈ *topspace* (*subtopology* (*powertop_real UNIV*) ({*x*. ∀ *i*≥*n*. *x i* = *0*} ∩ − {λ*i*. *0*})) **for** *x*
using *that* **by** *simp* (*metis abs_le_zero_iff le_enorm not_less*)
qed *auto*
qed (*use* ‹*d* > *0*› *enorm_ge0* **in** *auto*)
moreover have *subtopology ?E* {*x* ∈ *topspace ?E*. *enorm x* = *d*} = *subtopology ?E* {*x*. *enorm x* = *d*}
by (*simp add*: *subtopology_restrict Collect_conj_eq*)
ultimately show *continuous_map ?Y* (*subtopology* (*Euclidean_space n*) {*x* ∈ *topspace* (*Euclidean_space n*). *enorm x* = *d*}) (λ*x*. *mul* (*d* / *enorm x*) *x*)
by *metis*
qed
show *?g* (*0*::*real*) = *x ?g* (*1*::*real*) = *y*
using *that* **by** (*auto simp*: *S_def zero_def mul_def fun_eq_iff*)
qed
qed
obtain *a b* **where** *a*: *a* ∈ *topspace ?ES* **and** *b*: *b* ∈ *topspace ?ES*
and *a* ≠ *b* **and** *negab*: *?neg a* ≠ *b*
proof
let *?v* = λ*j i*::*nat*. *if i* = *j then d else 0*
show *?v 0* ∈ *topspace* (*subtopology ?E* (*S d*)) *?v 1* ∈ *topspace* (*subtopology ?E* (*S d*))
using ‹*n* ≥ *2*› ‹*d* ≥ *0*› **by** (*auto simp*: *S_def topspace_Euclidean_space*)

```
      show ?v 0 ≠ ?v 1 ?neg (?v 0) ≠ (?v 1)
        using ‹d > 0› by (auto simp: mul_def fun_eq_iff)
    qed
    show path_component_of ?ES x y
      if x: x ∈ topspace ?ES and y: y ∈ topspace ?ES
      for x y
    proof -
      have path_component_of ?ES x (?neg x)
      proof -
        have path_component_of ?ES x a
              by (metis (no_types, opaque_lifting) ∗∗ a b ‹a ≠ b› negab
path_component_of_trans path_component_of_sym x)
        moreover
        have pa_ab: path_component_of ?ES a b using ∗∗ a b negab neg_neg
by blast
        then have path_component_of ?ES a (?neg x)
       by (metis ∗∗ ‹a ≠ b› cloS closedin_def neg_in_S path_component_of_equiv
topspace_subtopology_subset x)
        ultimately show ?thesis
          by (meson path_component_of_trans)
      qed
      then show ?thesis
        using ∗∗ x y by force
    qed
  qed
  ultimately show ?thesis
    by (simp add: cloS closedin_subset path_connectedin_def)
qed (simp add: S_def cong: conj_cong)
 have path_component_of (subtopology ?E {x ∈ topspace ?E. enorm x ∈
T}) x y
  if enorm x = a x ∈ topspace ?E enorm x ∈ T enorm y = b y ∈ topspace
?E enorm y ∈ T
  for x y a b
  using that
  proof (induction a b arbitrary: x y rule: linorder_less_wlog)
    case (less a b)
    then have a ≥ 0
      using enorm_ge0 by blast
    with less.hyps have b > 0
      by linarith
    show ?case
    proof (rule path_component_of_trans)
      have y'_ts: mul (a / b) y ∈ topspace ?E
        using ‹y ∈ topspace ?E› mul_in_top by blast
      moreover have enorm (mul (a / b) y) = a
        unfolding enorm_mul using ‹0 < b› ‹0 ≤ a› less.prems by simp
      ultimately have y'_S: mul (a / b) y ∈ S a
        using S_def by blast
      have x ∈ S a
```

using *S_def less.prems* **by** *blast*
with ‹$x \in$ *topspace ?E*› *y′_ts y′_S*
have *path_component_of* (*subtopology ?E* (*S a*)) *x* (*mul* (*a* / *b*) *y*)
by (*metis* ∗ [*OF* ‹$a \geq 0$›] *path_connected_space_iff_path_component path_connectedin_def topspace_subtopology_subset*)
moreover
have {$f \in$ *topspace ?E*. *enorm f* = *a*} ⊆ {$f \in$ *topspace ?E*. *enorm f* ∈ *T*}
using ‹*enorm x* = *a*› ‹*enorm x* ∈ *T*› **by** *force*
ultimately
show *path_component_of* (*subtopology ?E* {*x*. *x* ∈ *topspace ?E* ∧ *enorm x* ∈ *T*}) *x* (*mul* (*a* / *b*) *y*)
by (*simp add*: *S_def path_component_of_mono*)
have *pathin ?E* (λ*t*. *mul* (((*1* − *t*) ∗ *b* + *t* ∗ *a*) / *b*) *y*)
using ‹*b* > *0*› ‹*y* ∈ *topspace ?E*›
unfolding *pathin_def Euclidean_space_def mul_def continuous_map_in_subtopology continuous_map_componentwise_UNIV*
by (*intro allI conjI continuous_intros*) *auto*
moreover have *mul* (((*1* − *t*) ∗ *b* + *t* ∗ *a*) / *b*) *y* ∈ *topspace ?E*
if *t* ∈ {*0..1*} **for** *t*
using ‹*y* ∈ *topspace ?E*› *mul_in_top* **by** *blast*
moreover have *enorm* (*mul* (((*1* − *t*) ∗ *b* + *t* ∗ *a*) / *b*) *y*) ∈ *T*
if *t* ∈ {*0..1*} **for** *t*
proof −
have *a* ∈ *T* *b* ∈ *T*
using *less.prems* **by** *auto*
then have |(*1* − *t*) ∗ *b* + *t* ∗ *a*| ∈ *T*
proof (*rule mem_is_interval_1_I* [*OF T*])
show *a* ≤ |(*1* − *t*) ∗ *b* + *t* ∗ *a*|
using *that* ‹*a* ≥ *0*› *less.hyps segment_bound_lemma* **by** *auto*
show |(*1* − *t*) ∗ *b* + *t* ∗ *a*| ≤ *b*
using *that* ‹*a* ≥ *0*› *less.hyps* **by** (*auto intro*: *convex_bound_le*)
qed
then show *?thesis*
unfolding *enorm_mul* ‹*enorm y* = *b*› **using** *that* ‹*b* > *0*› **by** *simp*
qed
ultimately have *pa*: *pathin* (*subtopology ?E* {*x* ∈ *topspace ?E*. *enorm x* ∈ *T*})
(λ*t*. *mul* (((*1* − *t*) ∗ *b* + *t* ∗ *a*) / *b*) *y*)
by (*auto simp*: *pathin_subtopology*)
have *ex_pathin*: ∃ *g*. *pathin* (*subtopology ?E* {*x* ∈ *topspace ?E*. *enorm x* ∈ *T*}) *g* ∧
g 0 = *y* ∧ *g 1* = *mul* (*a* / *b*) *y*
apply (*rule_tac x*=λ*t*. *mul* (((*1* − *t*) ∗ *b* + *t* ∗ *a*) / *b*) *y* **in** *exI*)
using ‹*b* > *0*› *pa* **by** (*auto simp*: *mul_def*)
show *path_component_of* (*subtopology ?E* {*x*. *x* ∈ *topspace ?E* ∧ *enorm x* ∈ *T*}) (*mul* (*a* / *b*) *y*) *y*
by (*rule path_component_of_sym*) (*simp add*: *path_component_of_def ex_pathin*)

```
        qed
      next
        case (refl a)
        then have pc: path_component_of (subtopology ?E (S (enorm u))) u v
          if u ∈ topspace ?E ∩ S (enorm x) v ∈ topspace ?E ∩ S (enorm u) for
u v
          using * [of a] enorm_ge0 that
      by (auto simp: path_connectedin_def path_connected_space_iff_path_component
S_def)
        have sub: {u ∈ topspace ?E. enorm u = enorm x} ⊆ {u ∈ topspace ?E.
enorm u ∈ T}
          using ‹enorm x ∈ T› by auto
        show ?case
          using pc [of x y] refl by (auto simp: S_def path_component_of_mono
[OF _ sub])
      next
        case (sym a b)
        then show ?case
          by (blast intro: path_component_of_sym)
      qed
    then show ?thesis
     by (simp add: path_connectedin_def path_connected_space_iff_path_component)
  qed
  have h ‘ S r ⊆ topspace ?E
    by (meson SC cmh compact_imp_compactin_subtopology compactinS com-
pactin_subset_topspace image_compactin)
  moreover
  have ¬ compact_space ?E
    by (metis compact_Euclidean_space ‹n ≠ 0›)
  then have ¬ compactin ?E (topspace ?E)
    by (simp add: compact_space_def topspace_Euclidean_space)
  then have h ‘ S r ≠ topspace ?E
    using com_hSr by auto
  ultimately have top_hSr_ne: topspace (subtopology ?E (topspace ?E − h ‘
S r)) ≠ {}
    by auto
  show pc1: ∃ T. T ∈ path_components_of (subtopology ?E (topspace ?E − h
‘ S r)) ∧ h ‘ B r ⊆ T
  proof (rule exists_path_component_of_superset [OF _ top_hSr_ne])
    have path_connectedin ?E (h ‘ B r)
    proof (rule path_connectedin_continuous_map_image)
      show continuous_map (subtopology ?E (C r)) ?E h
        by (simp add: cmh)
      have path_connectedin ?E (B r)
          using pc_interval[of {..<r}] is_interval_convex_1 unfolding B_def
by auto
        then show path_connectedin (subtopology ?E (C r)) (B r)
          by (simp add: path_connectedin_subtopology BC)
      qed
```

moreover have *h ‘ B r* ⊆ *topspace ?E* − *h ‘ S r*
apply (*auto simp*: *h_if_B*)
by (*metis BC SC disjSB disjnt_iff inj_onD* [*OF injh*] *subsetD*)
ultimately show *path_connectedin* (*subtopology ?E* (*topspace ?E* − *h ‘ S r*)) (*h ‘ B r*)
by (*simp add*: *path_connectedin_subtopology*)
qed *metis*
show ∃ *T*. *T* ∈ *path_components_of* (*subtopology ?E* (*topspace ?E* − *h ‘ S r*)) ∧ *topspace ?E* − *h ‘* (*C r*) ⊆ *T*
proof (*rule exists_path_component_of_superset* [*OF _ top_hSr_ne*])
have *eq*: *topspace ?E* − {*x* ∈ *topspace ?E*. *enorm x* ≤ *r*} = {*x* ∈ *topspace ?E*. *r* < *enorm x*}
by *auto*
have *path_connectedin ?E* (*topspace ?E* − *C r*)
using *pc_interval*[*of* {*r*<..}] *is_interval_convex_1* **unfolding** *C_def eq* **by** *auto*
then have *path_connectedin ?E* (*topspace ?E* − *h ‘ C r*)
by (*metis biglemma* [*OF* ‹*n* ≠ *0*› *compactinC cmh injh*])
then show *path_connectedin* (*subtopology ?E* (*topspace ?E* − *h ‘ S r*)) (*topspace ?E* − *h ‘ C r*)
by (*simp add*: *Diff_mono SC image_mono path_connectedin_subtopology*)
qed *metis*
have *topspace ?E* ∩ (*topspace ?E* − *h ‘ S r*) = *h ‘ B r* ∪ (*topspace ?E* − *h ‘ C r*) (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
using ‹⋀*r*. *B r* ∪ *S r* = *C r*› **by** *auto*
have *h ‘ B r* ∩ *h ‘ S r* = {}
by (*metis Diff_triv* ‹⋀*r*. *B r* ∪ *S r* = *C r*› ‹⋀*r*. *disjnt* (*S r*) (*B r*)› *disjnt_def inf_commute inj_on_Un injh*)
then show *?rhs* ⊆ *?lhs*
using *path_components_of_subset pc1* ‹⋀*r*. *B r* ∪ *S r* = *C r*›
by (*fastforce simp add*: *h_if_B*)
qed
then show ⋃ (*path_components_of* (*subtopology ?E* (*topspace ?E* − *h ‘ S r*))) = *h ‘ B r* ∪ (*topspace ?E* − *h ‘* (*C r*))
by (*simp add*: *Union_path_components_of*)
show *T* ≠ {}
if *T* ∈ *path_components_of* (*subtopology ?E* (*topspace ?E* − *h ‘ S r*)) **for** *T*
using *that* **by** (*simp add*: *nonempty_path_components_of*)
show *disjoint* (*path_components_of* (*subtopology ?E* (*topspace ?E* − *h ‘ S r*)))
by (*simp add*: *pairwise_disjoint_path_components_of*)
have ¬ *path_connectedin ?E* (*topspace ?E* − *h ‘ S r*)
proof (*subst biglemma* [*OF* ‹*n* ≠ *0*› *compactinS*])
show *continuous_map* (*subtopology ?E* (*S r*)) *?E h*
by (*metis Un_commute Un_upper1 cmh continuous_map_from_subtopology_mono eqC*)
show *inj_on h* (*S r*)

```
          using SC inj_on_subset injh by blast
        show ¬ path_connectedin ?E (topspace ?E − S r)
        proof
          have topspace ?E − S r = {x ∈ topspace ?E. enorm x ≠ r}
            by (auto simp: S_def)
          moreover have enorm ‘ {x ∈ topspace ?E. enorm x ≠ r} = {0..} − {r}
          proof
            have ∃x. x ∈ topspace ?E ∧ enorm x ≠ r ∧ d = enorm x
              if d ≠ r d ≥ 0 for d
            proof (intro exI conjI)
              show (λi. if i = 0 then d else 0) ∈ topspace ?E
                using ‹n ≠ 0› by (auto simp: Euclidean_space_def)
              show enorm (λi. if i = 0 then d else 0) ≠ r  d = enorm (λi. if i = 0
then d else 0)
                using ‹n ≠ 0› that by simp_all
            qed
            then show {0..} − {r} ⊆ enorm ‘ {x ∈ topspace ?E. enorm x ≠ r}
              by (auto simp: image_def)
          qed (auto simp: enorm_ge0)
          ultimately have non_r: enorm ‘ (topspace ?E − S r) = {0..} − {r}
            by simp
          have ∃x≥0. x ≠ r ∧ r ≤ x
            by (metis gt_ex le_cases not_le order_trans)
          then have ¬ is_interval ({0..} − {r})
            unfolding is_interval_1
            using  ‹r > 0› by (auto simp: Bex_def)
          then show False
            if path_connectedin ?E (topspace ?E − S r)
             using path_connectedin_continuous_map_image [OF cm_enorm that]
by (simp add: is_interval_path_connected_1 non_r)
        qed
      qed
      then have ¬ path_connected_space (subtopology ?E (topspace ?E − h ‘ S r))
        by (simp add: path_connectedin_def)
      then show ∄T. path_components_of (subtopology ?E (topspace ?E − h ‘ S
r)) ⊆ {T}
        by (simp add: path_components_of_subset_singleton)
    qed
    moreover have openin ?E A
      if A ∈ path_components_of (subtopology ?E (topspace ?E − h ‘ (S r))) for
A
      using locally_path_connected_Euclidean_space [of n] that ope_comp_hSr
      by (simp add: locally_path_connected_space_open_path_components)
    ultimately show ?thesis by metis
  qed
  have ∃T. openin ?E T ∧ f x ∈ T ∧ T ⊆ f ‘ U
    if x ∈ U for x
  proof −
    have x: x ∈ topspace ?E
```

```
    by (meson U in_mono openin_subset that)
  obtain V where V: openin (powertop_real UNIV) V and Ueq: U = V ∩ {x.
∀i≥n. x i = 0}
    using U by (auto simp: openin_subtopology Euclidean_space_def)
  with ‹x ∈ U› have x ∈ V by blast
  then obtain T where Tfin: finite {i. T i ≠ UNIV} and Topen: ⋀i. open (T
i)
    and Tx: x ∈ Pi_E UNIV T and TV: Pi_E UNIV T ⊆ V
    using V by (force simp: openin_product_topology_alt)
  have ∃e>0. ∀x'. |x' − x i| < e ⟶ x' ∈ T i for i
    using Topen [of i] Tx by (auto simp: open_real)
  then obtain β where B0: ⋀i. β i > 0 and BT: ⋀i x'. |x' − x i| < β i ⟹
x' ∈ T i
    by metis
  define r where r ≡ Min (insert 1 (β ‘ {i. T i ≠ UNIV}))
  have r > 0
    by (simp add: B0 Tfin r_def)
  have inU: y ∈ U
    if y: y ∈ topspace ?E and yxr: ⋀i. i<n ⟹ |y i − x i| < r for y
  proof −
    have y i ∈ T i for i
    proof (cases T i = UNIV)
      show y i ∈ T i if T i ≠ UNIV
      proof (cases i < n)
        case True
        then show ?thesis
          using yxr [OF True] that by (simp add: r_def BT Tfin)
      next
        case False
        then show ?thesis
         using B0 Ueq ‹x ∈ U› topspace_Euclidean_space y by (force intro: BT)
      qed
    qed auto
    with TV have y ∈ V by auto
    then show ?thesis
      using that by (auto simp: Ueq topspace_Euclidean_space)
  qed
  have xinU: (λi. x i + y i) ∈ U if y ∈ C(r/2) for y
  proof (rule inU)
    have y: y ∈ topspace ?E
      using C_def that by blast
    show (λi. x i + y i) ∈ topspace ?E
      using x y by (simp add: topspace_Euclidean_space)
    have enorm y ≤ r/2
      using that by (simp add: C_def)
    then show |x i + y i − x i| < r if i < n for i
      using le_enorm enorm_ge0 that ‹0 < r› leI order_trans by fastforce
  qed
  show ?thesis
```

```
    proof (intro exI conjI)
      show openin ?E ((f ∘ (λy i. x i + y i)) ` B (r/2))
      proof (rule **)
         have continuous_map (subtopology ?E (C(r/2))) (subtopology ?E U) (λy
i. x i + y i)
          by (auto simp: xinU continuous_map_in_subtopology
              intro!: continuous_intros continuous_map_Euclidean_space_add x)
        then show continuous_map (subtopology ?E (C(r/2))) ?E (f ∘ (λy i. x i
+ y i))
          by (rule continuous_map_compose) (simp add: cmf)
        show inj_on (f ∘ (λy i. x i + y i)) (C(r/2))
         proof (clarsimp simp add: inj_on_def C_def topspace_Euclidean_space
simp del: divide_const_simps)
          show y' = y
            if ey: enorm y ≤ r / 2 and ey': enorm y' ≤ r / 2
              and y0: ∀i≥n. y i = 0 and y'0: ∀i≥n. y' i = 0
              and feq: f (λi. x i + y' i) = f (λi. x i + y i)
            for y' y :: nat ⇒ real
          proof -
            have (λi. x i + y i) ∈ U
            proof (rule inU)
              show (λi. x i + y i) ∈ topspace ?E
                using topspace_Euclidean_space x y0 by auto
              show |x i + y i - x i| < r if i < n for i
                using ey le_enorm [of _ y] ‹r > 0› that by fastforce
            qed
            moreover have (λi. x i + y' i) ∈ U
            proof (rule inU)
              show (λi. x i + y' i) ∈ topspace ?E
                using topspace_Euclidean_space x y'0 by auto
              show |x i + y' i - x i| < r if i < n for i
                using ey' le_enorm [of _ y'] ‹r > 0› that by fastforce
            qed
            ultimately have (λi. x i + y' i) = (λi. x i + y i)
              using feq by (meson ‹inj_on f U› inj_on_def)
            then show ?thesis
              by (auto simp: fun_eq_iff)
          qed
        qed
      qed (simp add: ‹0 < r›)
      have x ∈ (λy i. x i + y i) ` B (r / 2)
      proof
        show x = (λi. x i + zero i)
          by (simp add: zero_def)
      qed (auto simp: B_def ‹r > 0›)
      then show f x ∈ (f ∘ (λy i. x i + y i)) ` B (r/2)
        by (metis image_comp image_eqI)
      show (f ∘ (λy i. x i + y i)) ` B (r/2) ⊆ f ` U
        using ‹⋀r. B r ⊆ C r› xinU by fastforce
```

```
    qed
  qed
  then show ?thesis
    using openin_subopen by force
qed


corollary invariance_of_domain_Euclidean_space_embedding_map:
  assumes openin (Euclidean_space n) U
   and cmf: continuous_map(subtopology (Euclidean_space n) U) (Euclidean_space
n) f
    and inj_on f U
   shows embedding_map(subtopology (Euclidean_space n) U) (Euclidean_space
n) f
proof (rule injective_open_imp_embedding_map [OF cmf])
  show open_map (subtopology (Euclidean_space n) U) (Euclidean_space n) f
    unfolding open_map_def
      by (meson assms continuous_map_from_subtopology_mono inj_on_subset
invariance_of_domain_Euclidean_space openin_imp_subset openin_trans_full)
  show inj_on f (topspace (subtopology (Euclidean_space n) U))
    using assms openin_subset topspace_subtopology_subset by fastforce
qed

corollary invariance_of_domain_Euclidean_space_gen:
  assumes n ≤ m and U: openin (Euclidean_space m) U
   and cmf: continuous_map(subtopology (Euclidean_space m) U) (Euclidean_space
n) f
    and inj_on f U
  shows openin (Euclidean_space n) (f ` U)
proof –
 have *: Euclidean_space n = subtopology (Euclidean_space m) (topspace(Euclidean_space
n))
   by (metis Euclidean_space_def ‹n ≤ m› inf.absorb_iff2 subset_Euclidean_space
subtopology_subtopology topspace_Euclidean_space)
  moreover have U ⊆ topspace (subtopology (Euclidean_space m) U)
   by (metis U inf.absorb_iff2 openin_subset openin_subtopology openin_topspace)
  ultimately show ?thesis
     by (metis (no_types) U ‹inj_on f U› cmf continuous_map_in_subtopology
inf.absorb_iff2
          inf.orderE invariance_of_domain_Euclidean_space openin_imp_subset
openin_subtopology openin_topspace)
qed

corollary invariance_of_domain_Euclidean_space_embedding_map_gen:
  assumes n ≤ m and U: openin (Euclidean_space m) U
   and cmf: continuous_map(subtopology (Euclidean_space m) U) (Euclidean_space
n) f
    and inj_on f U
   shows embedding_map(subtopology (Euclidean_space m) U) (Euclidean_space
```

n) f
proof (*rule injective_open_imp_embedding_map* [*OF cmf*])
show *open_map* (*subtopology* (*Euclidean_space m*) *U*) (*Euclidean_space n*) *f*
by (*meson U* ‹*n* ≤ *m*› ‹*inj_on f U*› *cmf continuous_map_from_subtopology_mono invariance_of_domain_Euclidean_space_gen open_map_def openin_open_subtopology subset_inj_on*)
show *inj_on f* (*topspace* (*subtopology* (*Euclidean_space m*) *U*))
using *assms openin_subset topspace_subtopology_subset* **by** *fastforce*
qed

0.4.3 Relating two variants of Euclidean space, one within product topology.

proposition *homeomorphic_maps_Euclidean_space_euclidean_gen_OLD*:
fixes *B* :: *'n*::*euclidean_space set*
assumes *finite B independent B* **and** *orth*: *pairwise orthogonal B* **and** *n*: *card B* = *n*
obtains *f g* **where** *homeomorphic_maps* (*Euclidean_space n*) (*top_of_set* (*span B*)) *f g*
proof −
note *representation_basis* [*OF* ‹*independent B*›, *simp*]
obtain *b* **where** *injb*: *inj_on b* {..<*n*} **and** *beq*: *b* ‘ {..<*n*} = *B*
using *finite_imp_nat_seg_image_inj_on* [*OF* ‹*finite B*›]
by (*metis n card_Collect_less_nat card_image lessThan_def*)
then have *biB*: ⋀*i*. *i* < *n* ⟹ *b i* ∈ *B*
by *force*
have *repr*: ⋀*v*. *v* ∈ *span B* ⟹ (∑*i*<*n*. *representation B v* (*b i*) $*_R$ *b i*) = *v*
using *real_vector.sum_representation_eq* [*OF* ‹*independent B*› _ ‹*finite B*›]
by (*metis* (*no_types*, *lifting*) *injb beq order_refl sum.reindex_cong*)
let *?f* = λ*x*. ∑*i*<*n*. *x i* $*_R$ *b i*
let *?g* = λ*v i*. *if i* < *n then representation B v* (*b i*) *else 0*
show *thesis*
proof
show *homeomorphic_maps* (*Euclidean_space n*) (*top_of_set* (*span B*)) *?f ?g*
unfolding *homeomorphic_maps_def*
proof (*intro conjI*)
have ∗: *continuous_map euclidean* (*top_of_set* (*span B*)) *?f*
by (*metis* (*mono_tags*) *biB continuous_map_span_sum lessThan_iff*)
show *continuous_map* (*Euclidean_space n*) (*top_of_set* (*span B*)) *?f*
unfolding *Euclidean_space_def*
by (*rule continuous_map_from_subtopology*) (*simp add*: *euclidean_product_topology* ∗)
show *continuous_map* (*top_of_set* (*span B*)) (*Euclidean_space n*) *?g*
unfolding *Euclidean_space_def*
by (*auto simp*: *continuous_map_in_subtopology continuous_map_componentwise_UNIV continuous_on_representation* ‹*independent B*› *biB orth pairwise_orthogonal_imp_finite*)
have [*simp*]: ⋀*x i*. *i*<*n* ⟹ *x i* $*_R$ *b i* ∈ *span B*
by (*simp add*: *biB span_base span_scale*)
have *representation B* (*?f x*) (*b j*) = *x j*

```
      if 0: ∀i≥n. x i = (0::real) and j < n for x j
    proof -
      have representation B (?f x) (b j) = (∑i<n. representation B (x i *R b i) (b j))
        by (subst real_vector.representation_sum) (auto simp add: ‹independent B›)
      also have ... = (∑i<n. x i * representation B (b i) (b j))
        by (simp add: assms(2) biB representation_scale span_base)
      also have ... = (∑i<n. if b j = b i then x i else 0)
        by (simp add: biB if_distrib cong: if_cong)
      also have ... = x j
        using that inj_on_eq_iff [OF injb] by auto
      finally show ?thesis .
    qed
    then show ∀x∈topspace (Euclidean_space n). ?g (?f x) = x
      by (auto simp: Euclidean_space_def)
    show ∀y∈topspace (top_of_set (span B)). ?f (?g y) = y
      using repr by (auto simp: Euclidean_space_def)
  qed
 qed
qed

proposition homeomorphic_maps_Euclidean_space_euclidean_gen:
  fixes B :: 'n::euclidean_space set
  assumes independent B and orth: pairwise orthogonal B and n: card B = n
    and 1: ⋀u. u ∈ B ⟹ norm u = 1
  obtains f g where homeomorphic_maps (Euclidean_space n) (top_of_set (span B)) f g
     and ⋀x. x ∈ topspace (Euclidean_space n) ⟹ (norm (f x))^2 = (∑i<n. (x i)^2)
proof -
  note representation_basis [OF ‹independent B›, simp]
  have finite B
    using ‹independent B› finiteI_independent by metis
  obtain b where injb: inj_on b {..<n} and beq: b ` {..<n} = B
    using finite_imp_nat_seg_image_inj_on [OF ‹finite B›]
    by (metis n card_Collect_less_nat card_image lessThan_def)
  then have biB: ⋀i. i < n ⟹ b i ∈ B
    by force
  have 0 ∉ B
    using ‹independent B› dependent_zero by blast
  have [simp]: b i · b j = (if j = i then 1 else 0)
    if i < n j < n for i j
  proof (cases i = j)
    case True
    with 1 that show ?thesis
      by (auto simp: norm_eq_sqrt_inner biB)
  next
    case False
```

```
    then have b i ≠ b j
      by (meson inj_onD injb lessThan_iff that)
    then show ?thesis
      using orth by (auto simp: orthogonal_def pairwise_def norm_eq_sqrt_inner
that biB)
  qed
  have [simp]: ⋀x i. i<n ⟹ x i *R b i ∈ span B
    by (simp add: biB span_base span_scale)
  have repr: ⋀v. v ∈ span B ⟹ (∑i<n. representation B v (b i) *R b i) = v
    using real_vector.sum_representation_eq [OF ‹independent B› _ ‹finite B›]
    by (metis (no_types, lifting) injb beq order_refl sum.reindex_cong)
    define f where f ≡ λx. ∑i<n. x i *R b i
    define g where g ≡ λv i. if i < n then representation B v (b i) else 0
  show thesis
  proof
    show homeomorphic_maps (Euclidean_space n) (top_of_set (span B)) f g
      unfolding homeomorphic_maps_def
    proof (intro conjI)
      have *: continuous_map euclidean (top_of_set (span B)) f
        unfolding f_def
        by (rule continuous_map_span_sum) (use biB ‹0 ∉ B› in auto)
      show continuous_map (Euclidean_space n) (top_of_set (span B)) f
        unfolding Euclidean_space_def
      by (rule continuous_map_from_subtopology) (simp add: euclidean_product_topology
*)
      show continuous_map (top_of_set (span B)) (Euclidean_space n) g
        unfolding Euclidean_space_def g_def
      by (auto simp: continuous_map_in_subtopology continuous_map_componentwise_UNIV
continuous_on_representation ‹independent B› biB orth pairwise_orthogonal_imp_finite)
      have representation B (f x) (b j) = x j
        if 0: ∀i≥n. x i = (0::real) and j < n for x j
      proof -
        have representation B (f x) (b j) = (∑i<n. representation B (x i *R b i)
(b j))
          unfolding f_def
          by (subst real_vector.representation_sum) (auto simp add: ‹independent
B›)
        also have ... = (∑i<n. x i * representation B (b i) (b j))
          by (simp add: ‹independent B› biB representation_scale span_base)
        also have ... = (∑i<n. if b j = b i then x i else 0)
          by (simp add: biB if_distrib cong: if_cong)
        also have ... = x j
          using that inj_on_eq_iff [OF injb] by auto
        finally show ?thesis .
      qed
      then show ∀x∈topspace (Euclidean_space n). g (f x) = x
        by (auto simp: Euclidean_space_def f_def g_def)
      show ∀y∈topspace (top_of_set (span B)). f (g y) = y
        using repr by (auto simp: Euclidean_space_def f_def g_def)
```

qed
show *normeq*: (*norm* (*f x*))^2 = ($\sum$ *i*<*n*. (*x i*)^2) **if** *x* ∈ *topspace* (*Euclidean_space n*) **for** *x*
unfolding *f_def dot_square_norm* [*symmetric*]
by (*simp add*: *power2_eq_square inner_sum_left inner_sum_right if_distrib biB cong*: *if_cong*)
qed
qed

corollary *homeomorphic_maps_Euclidean_space_euclidean*:
obtains *f* :: (*nat* ⇒ *real*) ⇒ *'n*::*euclidean_space* **and** *g*
where *homeomorphic_maps* (*Euclidean_space* (*DIM*(*'n*))) *euclidean f g*
by (*force intro*: *homeomorphic_maps_Euclidean_space_euclidean_gen* [*OF independent_Basis orthogonal_Basis refl norm_Basis*])

lemma *homeomorphic_maps_nsphere_euclidean_sphere*:
fixes *B* :: *'n*::*euclidean_space set*
assumes *B*: *independent B* **and** *orth*: *pairwise orthogonal B* **and** *n*: *card B* = *n* **and** *n* ≠ *0*
and *1*: ⋀*u*. *u* ∈ *B* ⟹ *norm u* = *1*
obtains *f* :: (*nat* ⇒ *real*) ⇒ *'n*::*euclidean_space* **and** *g*
where *homeomorphic_maps* (*nsphere*(*n* − *1*)) (*top_of_set* (*sphere 0 1* ∩ *span B*)) *f g*
proof −
have *finite B*
using ‹*independent B*› *finiteI_independent* **by** *metis*
obtain *f g* **where** *fg*: *homeomorphic_maps* (*Euclidean_space n*) (*top_of_set* (*span B*)) *f g*
and *normf*: ⋀*x*. *x* ∈ *topspace* (*Euclidean_space n*) ⟹ (*norm* (*f x*))^2 = ($\sum$ *i*<*n*. (*x i*)^2)
using *homeomorphic_maps_Euclidean_space_euclidean_gen* [*OF B orth n 1*]
by *blast*
obtain *b* **where** *injb*: *inj_on b* {..<*n*} **and** *beq*: *b* ‘ {..<*n*} = *B*
using *finite_imp_nat_seg_image_inj_on* [*OF* ‹*finite B*›]
by (*metis n card_Collect_less_nat card_image lessThan_def*)
then have *biB*: ⋀*i*. *i* < *n* ⟹ *b i* ∈ *B*
by *force*
have [*simp*]: ⋀*i*. *i* < *n* ⟹ *b i* ≠ *0*
using ‹*independent B*› *biB dependent_zero* **by** *fastforce*
have [*simp*]: *b i* · *b j* = (*if j* = *i then* (*norm* (*b i*))^2 *else 0*)
if *i* < *n j* < *n* **for** *i j*
proof (*cases i* = *j*)
case *False*
then have *b i* ≠ *b j*
by (*meson inj_onD injb lessThan_iff that*)
then show *?thesis*
using *orth* **by** (*auto simp*: *orthogonal_def pairwise_def norm_eq_sqrt_inner that biB*)
qed (*auto simp*: *norm_eq_sqrt_inner*)

```
  have [simp]: Suc (n − Suc 0) = n
    using Suc_pred ‹n ≠ 0› by blast
  then have [simp]: {..card B − Suc 0} = {..<card B}
    using n by fastforce
  show thesis
  proof
    have 1: norm (f x) = 1
      if (∑ i<card B. (x i)^2) = (1::real) x ∈ topspace (Euclidean_space n) for x
    proof −
      have norm (f x)^2 = 1
        using normf that by (simp add: n)
      with that show ?thesis
        by (simp add: power2_eq_imp_eq)
    qed
    have homeomorphic_maps (nsphere (n − 1)) (top_of_set (span B ∩ sphere 0
1)) f g
      unfolding nsphere_def subtopology_subtopology [symmetric]
      proof (rule homeomorphic_maps_subtopologies_alt)
  show homeomorphic_maps (Euclidean_space (Suc (n − 1))) (top_of_set (span
B)) f g
      using fg by (force simp add: )
    show f ‘ (topspace (Euclidean_space (Suc (n − 1))) ∩ {x. (∑ i≤n − 1. (x i)^2)
= 1}) ⊆ sphere 0 1
      using n by (auto simp: image_subset_iff Euclidean_space_def 1)
    have (∑ i≤n − Suc 0. (g u i)^2) = 1
      if u ∈ span B and norm (u::'n) = 1 for u
    proof −
      obtain v where [simp]: u = f v v ∈ topspace (Euclidean_space n)
        using fg unfolding homeomorphic_maps_map subset_iff
          by (metis ‹u ∈ span B› homeomorphic_imp_surjective_map image_eqI
topspace_euclidean_subtopology)
      then have [simp]: g (f v) = v
        by (meson fg homeomorphic_maps_map)
      have fv21: norm (f v) ^ 2 = 1
        using that by simp
      show ?thesis
        using that normf fv21 ‹v ∈ topspace (Euclidean_space n)› n by force
    qed
    then show g ‘ (topspace (top_of_set (span B)) ∩ sphere 0 1) ⊆ {x. (∑ i≤n
− 1. (x i)^2) = 1}
      by auto
  qed
  then show homeomorphic_maps (nsphere(n − 1)) (top_of_set (sphere 0 1 ∩
span B)) f g
    by (simp add: inf_commute)
  qed
qed
```

0.4.4 Invariance of dimension and domain

lemma homeomorphic_maps_iff_homeomorphism [simp]:
homeomorphic_maps (top_of_set S) (top_of_set T) f g ⟷ homeomorphism S T f g
unfolding homeomorphic_maps_def homeomorphism_def **by** force

lemma homeomorphic_space_iff_homeomorphic [simp]:
(top_of_set S) homeomorphic_space (top_of_set T) ⟷ S homeomorphic T
by (simp add: homeomorphic_def homeomorphic_space_def)

lemma homeomorphic_subspace_Euclidean_space:
fixes S :: 'a::euclidean_space set
assumes subspace S
shows top_of_set S homeomorphic_space Euclidean_space n ⟷ dim S = n
proof −
obtain B **where** B: B ⊆ S independent B span B = S card B = dim S
and orth: pairwise orthogonal B **and** 1: ⋀x. x ∈ B ⟹ norm x = 1
by (metis assms orthonormal_basis_subspace)
then have finite B
by (simp add: pairwise_orthogonal_imp_finite)
have top_of_set S homeomorphic_space top_of_set (span B)
unfolding homeomorphic_space_iff_homeomorphic
by (auto simp: assms B intro: homeomorphic_subspaces)
also have . . . homeomorphic_space Euclidean_space (dim S)
unfolding homeomorphic_space_def
using homeomorphic_maps_Euclidean_space_euclidean_gen [OF ‹independent B› orth] homeomorphic_maps_sym 1 B
by metis
finally have top_of_set S homeomorphic_space Euclidean_space (dim S) **.**
then show ?thesis
using homeomorphic_space_sym homeomorphic_space_trans invariance_of_dimension_Euclidean_
by blast
qed

lemma homeomorphic_subspace_Euclidean_space_dim:
fixes S :: 'a::euclidean_space set
assumes subspace S
shows top_of_set S homeomorphic_space Euclidean_space (dim S)
by (simp add: homeomorphic_subspace_Euclidean_space assms)

lemma homeomorphic_subspaces_eq:
fixes S T:: 'a::euclidean_space set
assumes subspace S subspace T
shows S homeomorphic T ⟷ dim S = dim T
proof
show dim S = dim T
if S homeomorphic T
proof −
have Euclidean_space (dim S) homeomorphic_space top_of_set S

```
    using ‹subspace S› homeomorphic_space_sym homeomorphic_subspace_Euclidean_space_dim
by blast
    also have ... homeomorphic_space top_of_set T
      by (simp add: that)
    also have ... homeomorphic_space Euclidean_space (dim T)
      by (simp add: homeomorphic_subspace_Euclidean_space assms)
    finally have Euclidean_space (dim S) homeomorphic_space Euclidean_space
(dim T) .
    then show ?thesis
      by (simp add: invariance_of_dimension_Euclidean_space)
  qed
next
  show S homeomorphic T
    if dim S = dim T
    by (metis that assms homeomorphic_subspaces)
qed

lemma homeomorphic_affine_Euclidean_space:
  assumes affine S
  shows top_of_set S homeomorphic_space Euclidean_space n ⟷ aff_dim S =
n
   (is ?X homeomorphic_space ?E ⟷ aff_dim S = n)
proof (cases S = {})
  case True
  with assms show ?thesis
    using homeomorphic_empty_space nontrivial_Euclidean_space by fastforce
next
  case False
  then obtain a where a ∈ S
    by force
  have (?X homeomorphic_space ?E)
       = (top_of_set (image (λx. −a + x) S) homeomorphic_space ?E)
  proof
    show top_of_set ((+) (− a) ‘ S) homeomorphic_space ?E
      if ?X homeomorphic_space ?E
      using that
     by (meson homeomorphic_space_iff_homeomorphic homeomorphic_space_sym
homeomorphic_space_trans homeomorphic_translation)
    show ?X homeomorphic_space ?E
      if top_of_set ((+) (− a) ‘ S) homeomorphic_space ?E
      using that
     by (meson homeomorphic_space_iff_homeomorphic homeomorphic_space_trans
homeomorphic_translation)
  qed
  also have ... ⟷ aff_dim S = n
   by (metis ‹a ∈ S› aff_dim_eq_dim affine_diffs_subspace affine_hull_eq assms
homeomorphic_subspace_Euclidean_space of_nat_eq_iff)
  finally show ?thesis .
qed
```

corollary *invariance_of_domain_subspaces*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *ope*: *openin* (*top_of_set U*) *S*
and *subspace U subspace V* **and** *VU*: *dim V* ≤ *dim U*
and *contf*: *continuous_on S f* **and** *fim*: *f* ' *S* ⊆ *V*
and *injf*: *inj_on f S*
shows *openin* (*top_of_set V*) (*f* ' *S*)
proof −
have *S* ⊆ *U*
using *openin_imp_subset* [*OF ope*] **.**
have *Uhom*: *top_of_set U homeomorphic_space Euclidean_space* (*dim U*)
and *Vhom*: *top_of_set V homeomorphic_space Euclidean_space* (*dim V*)
by (*simp_all add*: *assms homeomorphic_subspace_Euclidean_space_dim*)
then obtain φ φ' **where** *hom*: *homeomorphic_maps* (*top_of_set U*) (*Euclidean_space* (*dim U*)) φ φ'
by (*auto simp*: *homeomorphic_space_def*)
obtain ψ ψ' **where** ψ: *homeomorphic_map* (*top_of_set V*) (*Euclidean_space* (*dim V*)) ψ
and ψ'ψ: ∀ *x*∈*V*. ψ' (ψ *x*) = *x*
using *Vhom* **by** (*auto simp*: *homeomorphic_space_def homeomorphic_maps_map*)
have ((ψ ∘ *f* ∘ φ') *o* φ) ' *S* = (ψ *o f*) ' *S*
proof (*rule image_cong* [*OF refl*])
show (ψ ∘ *f* ∘ φ' ∘ φ) *x* = (ψ ∘ *f*) *x* **if** *x* ∈ *S* **for** *x*
using *that* **unfolding** *o_def*
by (*metis* ‹*S* ⊆ *U*› *hom homeomorphic_maps_map in_mono topspace_euclidean_subtopology*)
qed
moreover
have *openin* (*Euclidean_space* (*dim V*)) ((ψ ∘ *f* ∘ φ') ' φ ' *S*)
proof (*rule invariance_of_domain_Euclidean_space_gen* [*OF VU*])
show *openin* (*Euclidean_space* (*dim U*)) (φ ' *S*)
using *homeomorphic_map_openness_eq hom homeomorphic_maps_map ope* **by** *blast*
show *continuous_map* (*subtopology* (*Euclidean_space* (*dim U*)) (φ ' *S*)) (*Euclidean_space* (*dim V*)) (ψ ∘ *f* ∘ φ')
proof (*intro continuous_map_compose*)
have *continuous_on* ({*x*. ∀ *i*≥*dim U*. *x i* = *0*} ∩ φ ' *S*) φ'
if *continuous_on* {*x*. ∀ *i*≥*dim U*. *x i* = *0*} φ'
using *that* **by** (*force elim*: *continuous_on_subset*)
moreover have φ' ' ({*x*. ∀ *i*≥*dim U*. *x i* = *0*} ∩ φ ' *S*) ⊆ *S*
if ∀ *x*∈*U*. φ' (φ *x*) = *x*
using *that* ‹*S* ⊆ *U*› **by** *fastforce*
ultimately show *continuous_map* (*subtopology* (*Euclidean_space* (*dim U*)) (φ ' *S*)) (*top_of_set S*) φ'
using *hom* **unfolding** *homeomorphic_maps_def*
by (*simp add*: *Euclidean_space_def subtopology_subtopology euclidean_product_topology*)
show *continuous_map* (*top_of_set S*) (*top_of_set V*) *f*
by (*simp add*: *contf fim*)

show *continuous_map* (*top_of_set V*) (*Euclidean_space* (*dim V*)) *ψ*
by (*simp add*: *ψ homeomorphic_imp_continuous_map*)
qed
show *inj_on* (*ψ* ∘ *f* ∘ *φ'*) (*φ* ' *S*)
using *injf hom*
unfolding *inj_on_def homeomorphic_maps_map*
by *simp* (*metis* ‹*S* ⊆ *U*› *ψ'ψ fim imageI subsetD*)
qed
ultimately have *openin* (*Euclidean_space* (*dim V*)) (*ψ* ' *f* ' *S*)
by (*simp add*: *image_comp*)
then show *?thesis*
by (*simp add*: *fim homeomorphic_map_openness_eq* [*OF ψ*])
qed

lemma *invariance_of_domain*:
fixes *f* :: *'a* ⇒ *'a*::*euclidean_space*
assumes *continuous_on S f open S inj_on f S* **shows** *open*(*f* ' *S*)
using *invariance_of_domain_subspaces* [*of UNIV S UNIV*] *assms* **by** (*force simp add*:)

corollary *invariance_of_dimension_subspaces*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *ope*: *openin* (*top_of_set U*) *S*
and *subspace U subspace V*
and *contf*: *continuous_on S f* **and** *fim*: *f* ' *S* ⊆ *V*
and *injf*: *inj_on f S* **and** *S* ≠ {}
shows *dim U* ≤ *dim V*
proof −
have *False* **if** *dim V* < *dim U*
proof −
obtain *T* **where** *subspace T T* ⊆ *U dim T* = *dim V*
using *choose_subspace_of_subspace* [*of dim V U*]
by (*metis* ‹*dim V* < *dim U*› *assms*(*2*) *order.strict_implies_order span_eq_iff*)
then have *V homeomorphic T*
by (*simp add*: ‹*subspace V*› *homeomorphic_subspaces*)
then obtain *h k* **where** *homhk*: *homeomorphism V T h k*
using *homeomorphic_def* **by** *blast*
have *continuous_on S* (*h* ∘ *f*)
by (*meson contf continuous_on_compose continuous_on_subset fim homeomorphism_cont1 homhk*)
moreover have (*h* ∘ *f*) ' *S* ⊆ *U*
using ‹*T* ⊆ *U*› *fim homeomorphism_image1 homhk* **by** *fastforce*
moreover have *inj_on* (*h* ∘ *f*) *S*
apply (*clarsimp simp*: *inj_on_def*)
by (*metis fim homeomorphism_apply1 homhk image_subset_iff inj_onD injf*)
ultimately have *ope_hf*: *openin* (*top_of_set U*) ((*h* ∘ *f*) ' *S*)
using *invariance_of_domain_subspaces* [*OF ope* ‹*subspace U*› ‹*subspace U*›] **by** *blast*
have (*h* ∘ *f*) ' *S* ⊆ *T*

```
      using fim homeomorphism_image1 homhk by fastforce
    then have dim ((h ∘ f) ‘ S) ≤ dim T
      by (rule dim_subset)
    also have dim ((h ∘ f) ‘ S) = dim U
      using ‹S ≠ {}› ‹subspace U›
      by (blast intro: dim_openin ope_hf)
    finally show False
      using ‹dim V < dim U› ‹dim T = dim V› by simp
  qed
  then show ?thesis
    using not_less by blast
qed

corollary invariance_of_domain_affine_sets:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
      and aff: affine U affine V aff_dim V ≤ aff_dim U
      and contf: continuous_on S f and fim: f ‘ S ⊆ V
      and injf: inj_on f S
    shows openin (top_of_set V) (f ‘ S)
proof (cases S = {})
  case False
  obtain a b where a ∈ S a ∈ U b ∈ V
    using False fim ope openin_contains_cball by fastforce
  have openin (top_of_set ((+) (− b) ‘ V)) ((((+) (− b) ∘ f ∘ (+) a) ‘ (+) (− a)
‘ S)
  proof (rule invariance_of_domain_subspaces)
    show openin (top_of_set ((+) (− a) ‘ U)) ((+) (− a) ‘ S)
      by (metis ope homeomorphism_imp_open_map homeomorphism_translation
translation_galois)
    show subspace ((+) (− a) ‘ U)
        by (simp add: ‹a ∈ U› affine_diffs_subspace_subtract ‹affine U› cong:
image_cong_simp)
    show subspace ((+) (− b) ‘ V)
        by (simp add: ‹b ∈ V› affine_diffs_subspace_subtract ‹affine V› cong:
image_cong_simp)
    show dim ((+) (− b) ‘ V) ≤ dim ((+) (− a) ‘ U)
     by (metis ‹a ∈ U› ‹b ∈ V› aff_dim_eq_dim affine_hull_eq aff of_nat_le_iff)
    show continuous_on ((+) (− a) ‘ S) ((+) (− b) ∘ f ∘ (+) a)
      by (metis contf continuous_on_compose homeomorphism_cont2 homeomor-
phism_translation translation_galois)
    show ((+) (− b) ∘ f ∘ (+) a) ‘ (+) (− a) ‘ S ⊆ (+) (− b) ‘ V
      using fim by auto
    show inj_on ((+) (− b) ∘ f ∘ (+) a) ((+) (− a) ‘ S)
      by (auto simp: inj_on_def) (meson inj_onD injf)
  qed
  then show ?thesis
      by (metis (no_types, lifting) homeomorphism_imp_open_map homeomor-
phism_translation image_comp translation_galois)
```

```
qed auto

corollary invariance_of_dimension_affine_sets:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
      and aff: affine U affine V
      and contf: continuous_on S f and fim: f ` S ⊆ V
      and injf: inj_on f S and S ≠ {}
    shows aff_dim U ≤ aff_dim V
proof –
  obtain a b where a ∈ S a ∈ U b ∈ V
    using ‹S ≠ {}› fim ope openin_contains_cball by fastforce
  have dim ((+) (– a) ` U) ≤ dim ((+) (– b) ` V)
  proof (rule invariance_of_dimension_subspaces)
    show openin (top_of_set ((+) (– a) ` U)) ((+) (– a) ` S)
      by (metis ope homeomorphism_imp_open_map homeomorphism_translation
translation_galois)
    show subspace ((+) (– a) ` U)
        by (simp add: ‹a ∈ U› affine_diffs_subspace_subtract ‹affine U› cong:
image_cong_simp)
    show subspace ((+) (– b) ` V)
        by (simp add: ‹b ∈ V› affine_diffs_subspace_subtract ‹affine V› cong:
image_cong_simp)
    show continuous_on ((+) (– a) ` S) ((+) (– b) ∘ f ∘ (+) a)
      by (metis contf continuous_on_compose homeomorphism_cont2 homeomor-
phism_translation translation_galois)
    show ((+) (– b) ∘ f ∘ (+) a) ` (+) (– a) ` S ⊆ (+) (– b) ` V
      using fim by auto
    show inj_on ((+) (– b) ∘ f ∘ (+) a) ((+) (– a) ` S)
      by (auto simp: inj_on_def) (meson inj_onD injf)
  qed (use ‹S ≠ {}› in auto)
  then show ?thesis
    by (metis ‹a ∈ U› ‹b ∈ V› aff_dim_eq_dim affine_hull_eq aff of_nat_le_iff)
qed

corollary invariance_of_dimension:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes contf: continuous_on S f and open S
      and injf: inj_on f S and S ≠ {}
    shows DIM('a) ≤ DIM('b)
  using invariance_of_dimension_subspaces [of UNIV S UNIV f] assms
  by auto

corollary continuous_injective_image_subspace_dim_le:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes subspace S subspace T
      and contf: continuous_on S f and fim: f ` S ⊆ T
      and injf: inj_on f S
    shows dim S ≤ dim T
```

```
  apply (rule invariance_of_dimension_subspaces [of S S _ f])
  using assms by (auto simp: subspace_affine)

lemma invariance_of_dimension_convex_domain:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes convex S
      and contf: continuous_on S f and fim: f ‘ S ⊆ affine hull T
      and injf: inj_on f S
    shows aff_dim S ≤ aff_dim T
proof (cases S = {})
  case True
  then show ?thesis by (simp add: aff_dim_geq)
next
  case False
  have aff_dim (affine hull S) ≤ aff_dim (affine hull T)
  proof (rule invariance_of_dimension_affine_sets)
    show openin (top_of_set (affine hull S)) (rel_interior S)
      by (simp add: openin_rel_interior)
    show continuous_on (rel_interior S) f
      using contf continuous_on_subset rel_interior_subset by blast
    show f ‘ rel_interior S ⊆ affine hull T
      using fim rel_interior_subset by blast
    show inj_on f (rel_interior S)
      using inj_on_subset injf rel_interior_subset by blast
    show rel_interior S ≠ {}
      by (simp add: False ‹convex S› rel_interior_eq_empty)
  qed auto
  then show ?thesis
    by simp
qed

lemma homeomorphic_convex_sets_le:
  assumes convex S S homeomorphic T
  shows aff_dim S ≤ aff_dim T
proof −
  obtain h k where homhk: homeomorphism S T h k
    using homeomorphic_def assms  by blast
  show ?thesis
  proof (rule invariance_of_dimension_convex_domain [OF ‹convex S›])
    show continuous_on S h
      using homeomorphism_def homhk by blast
    show h ‘ S ⊆ affine hull T
      by (metis homeomorphism_def homhk hull_subset)
    show inj_on h S
      by (meson homeomorphism_apply1 homhk inj_on_inverseI)
  qed
qed

lemma homeomorphic_convex_sets:
```

```
  assumes convex S convex T S homeomorphic T
  shows aff_dim S = aff_dim T
 by (meson assms dual_order.antisym homeomorphic_convex_sets_le homeomor-
phic_sym)

lemma homeomorphic_convex_compact_sets_eq:
  assumes convex S compact S convex T compact T
  shows S homeomorphic T ⟷ aff_dim S = aff_dim T
 by (meson assms homeomorphic_convex_compact_sets homeomorphic_convex_sets)

lemma invariance_of_domain_gen:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes open S continuous_on S f inj_on f S DIM('b) ≤ DIM('a)
    shows open(f ` S)
  using invariance_of_domain_subspaces [of UNIV S UNIV f] assms by auto

lemma injective_into_1d_imp_open_map_UNIV:
  fixes f :: 'a::euclidean_space ⇒ real
  assumes open T continuous_on S f inj_on f S T ⊆ S
    shows open (f ` T)
  apply (rule invariance_of_domain_gen [OF ‹open T›])
  using assms apply (auto simp: elim: continuous_on_subset subset_inj_on)
  done

lemma continuous_on_inverse_open:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes open S continuous_on S f DIM('b) ≤ DIM('a) and gf: ⋀x. x ∈ S ⟹
g(f x) = x
    shows continuous_on (f ` S) g
proof (clarsimp simp add: continuous_openin_preimage_eq)
  fix T :: 'a set
  assume open T
  have eq: f ` S ∩ g -` T = f ` (S ∩ T)
    by (auto simp: gf)
  have openin (top_of_set (f ` S)) (f ` (S ∩ T))
  proof (rule open_openin_trans [OF invariance_of_domain_gen])
    show inj_on f S
      using inj_on_inverseI gf by auto
    show open (f ` (S ∩ T))
      by (meson ‹inj_on f S› ‹open T› assms(1-3) continuous_on_subset inf_le1
inj_on_subset invariance_of_domain_gen open_Int)
  qed (use assms in auto)
  then show openin (top_of_set (f ` S)) (f ` S ∩ g -` T)
    by (simp add: eq)
qed

lemma invariance_of_domain_homeomorphism:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes open S continuous_on S f DIM('b) ≤ DIM('a) inj_on f S
```

obtains *g* **where** *homeomorphism S (f ‘ S) f g*
proof
show *homeomorphism S (f ‘ S) f (inv_into S f)*
by *(simp add: assms continuous_on_inverse_open homeomorphism_def)*
qed

corollary *invariance_of_domain_homeomorphic*:
fixes *f* :: *'a::euclidean_space ⇒ 'b::euclidean_space*
assumes *open S continuous_on S f DIM('b) ≤ DIM('a) inj_on f S*
shows *S homeomorphic (f ‘ S)*
using *invariance_of_domain_homeomorphism [OF assms]*
by *(meson homeomorphic_def)*

lemma *continuous_image_subset_interior*:
fixes *f* :: *'a::euclidean_space ⇒ 'b::euclidean_space*
assumes *continuous_on S f inj_on f S DIM('b) ≤ DIM('a)*
shows *f ‘ (interior S) ⊆ interior(f ‘ S)*
proof *(rule interior_maximal)*
show *f ‘ interior S ⊆ f ‘ S*
by *(simp add: image_mono interior_subset)*
show *open (f ‘ interior S)*
using *assms*
by *(auto simp: subset_inj_on interior_subset continuous_on_subset invariance_of_domain_gen)*
qed

lemma *homeomorphic_interiors_same_dimension*:
fixes *S* :: *'a::euclidean_space set* **and** *T* :: *'b::euclidean_space set*
assumes *S homeomorphic T* **and** *dimeq: DIM('a) = DIM('b)*
shows *(interior S) homeomorphic (interior T)*
using *assms [unfolded homeomorphic_minimal]*
unfolding *homeomorphic_def*
proof *(clarify elim!: ex_forward)*
fix *f g*
assume *S: ∀ x∈S. f x ∈ T ∧ g (f x) = x* **and** *T: ∀ y∈T. g y ∈ S ∧ f (g y) = y*
and *contf: continuous_on S f* **and** *contg: continuous_on T g*
then have *fST: f ‘ S = T* **and** *gTS: g ‘ T = S* **and** *inj_on f S inj_on g T*
by *(auto simp: inj_on_def intro: rev_image_eqI) metis+*
have *fim: f ‘ interior S ⊆ interior T*
using *continuous_image_subset_interior [OF contf ‹inj_on f S›] dimeq fST*
by *simp*
have *gim: g ‘ interior T ⊆ interior S*
using *continuous_image_subset_interior [OF contg ‹inj_on g T›] dimeq gTS*
by *simp*
show *homeomorphism (interior S) (interior T) f g*
unfolding *homeomorphism_def*
proof *(intro conjI ballI)*
show *⋀x. x ∈ interior S ⟹ g (f x) = x*
by *(meson ‹∀ x∈S. f x ∈ T ∧ g (f x) = x› subsetD interior_subset)*

```
    have interior T ⊆ f ` interior S
    proof
      fix x assume x ∈ interior T
      then have g x ∈ interior S
        using gim by blast
      then show x ∈ f ` interior S
        by (metis T ‹x ∈ interior T› image_iff interior_subset subsetCE)
    qed
    then show f ` interior S = interior T
      using fim by blast
    show continuous_on (interior S) f
      by (metis interior_subset continuous_on_subset contf)
    show ⋀y. y ∈ interior T ⟹ f (g y) = y
      by (meson T subsetD interior_subset)
    have interior S ⊆ g ` interior T
    proof
      fix x assume x ∈ interior S
      then have f x ∈ interior T
        using fim by blast
      then show x ∈ g ` interior T
        by (metis S ‹x ∈ interior S› image_iff interior_subset subsetCE)
    qed
    then show g ` interior T = interior S
      using gim by blast
    show continuous_on (interior T) g
      by (metis interior_subset continuous_on_subset contg)
  qed
qed

lemma homeomorphic_open_imp_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T open S S ≠ {} open T T ≠ {}
  shows DIM('a) = DIM('b)
    using assms
    apply (simp add: homeomorphic_minimal)
    apply (rule order_antisym; metis inj_onI invariance_of_dimension)
    done

proposition homeomorphic_interiors:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T interior S = {} ⟷ interior T = {}
    shows (interior S) homeomorphic (interior T)
proof (cases interior T = {})
  case True
  with assms show ?thesis by auto
next
  case False
  then have DIM('a) = DIM('b)
    using assms
```

```
    apply (simp add: homeomorphic_minimal)
   apply (rule order_antisym; metis continuous_on_subset inj_onI inj_on_subset
interior_subset invariance_of_dimension open_interior)
    done
  then show ?thesis
   by (rule homeomorphic_interiors_same_dimension [OF ‹S homeomorphic T›])
qed

lemma homeomorphic_frontiers_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T closed S closed T and dimeq: DIM('a) = DIM('b)
  shows (frontier S) homeomorphic (frontier T)
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix f g
  assume S: ∀x∈S. f x ∈ T ∧ g (f x) = x and T: ∀y∈T. g y ∈ S ∧ f (g y) = y
     and contf: continuous_on S f and contg: continuous_on T g
  then have fST: f ` S = T and gTS: g ` T = S and inj_on f S inj_on g T
    by (auto simp: inj_on_def intro: rev_image_eqI) metis+
  have g ` interior T ⊆ interior S
    using continuous_image_subset_interior [OF contg ‹inj_on g T›] dimeq gTS
by simp
  then have fim: f ` frontier S ⊆ frontier T
    apply (simp add: frontier_def)
    using continuous_image_subset_interior assms(2) assms(3) S by auto
  have f ` interior S ⊆ interior T
     using continuous_image_subset_interior [OF contf ‹inj_on f S›] dimeq fST
by simp
  then have gim: g ` frontier T ⊆ frontier S
    apply (simp add: frontier_def)
    using continuous_image_subset_interior T assms(2) assms(3) by auto
  show homeomorphism (frontier S) (frontier T) f g
    unfolding homeomorphism_def
  proof (intro conjI ballI)
    show gf: ⋀x. x ∈ frontier S ⟹ g (f x) = x
      by (simp add: S assms(2) frontier_def)
    show fg: ⋀y. y ∈ frontier T ⟹ f (g y) = y
      by (simp add: T assms(3) frontier_def)
    have frontier T ⊆ f ` frontier S
    proof
      fix x assume x ∈ frontier T
      then have g x ∈ frontier S
        using gim by blast
      then show x ∈ f ` frontier S
        by (metis fg ‹x ∈ frontier T› imageI)
    qed
    then show f ` frontier S = frontier T
      using fim by blast
```

```
    show continuous_on (frontier S) f
      by (metis Diff_subset assms(2) closure_eq contf continuous_on_subset frontier_def)
    have frontier S ⊆ g ‘ frontier T
    proof
      fix x assume x ∈ frontier S
      then have f x ∈ frontier T
        using fim by blast
      then show x ∈ g ‘ frontier T
        by (metis gf ‹x ∈ frontier S› imageI)
    qed
    then show g ‘ frontier T = frontier S
      using gim by blast
    show continuous_on (frontier T) g
      by (metis Diff_subset assms(3) closure_closed contg continuous_on_subset frontier_def)
  qed
qed

lemma homeomorphic_frontiers:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T closed S closed T
          interior S = {} ⟷ interior T = {}
    shows (frontier S) homeomorphic (frontier T)
proof (cases interior T = {})
  case True
  then show ?thesis
    by (metis Diff_empty assms closure_eq frontier_def)
next
  case False
  show ?thesis
    apply (rule homeomorphic_frontiers_same_dimension)
       apply (simp_all add: assms)
   using False assms homeomorphic_interiors homeomorphic_open_imp_same_dimension
by blast
qed

lemma continuous_image_subset_rel_interior:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes contf: continuous_on S f and injf: inj_on f S and fim: f ‘ S ⊆ T
      and TS: aff_dim T ≤ aff_dim S
  shows f ‘ (rel_interior S) ⊆ rel_interior(f ‘ S)
proof (rule rel_interior_maximal)
  show f ‘ rel_interior S ⊆ f ‘ S
    by(simp add: image_mono rel_interior_subset)
  show openin (top_of_set (affine hull f ‘ S)) (f ‘ rel_interior S)
  proof (rule invariance_of_domain_affine_sets)
    show openin (top_of_set (affine hull S)) (rel_interior S)
      by (simp add: openin_rel_interior)
```

```
    show aff_dim (affine hull f ` S) ≤ aff_dim (affine hull S)
      by (metis aff_dim_affine_hull aff_dim_subset fim TS order_trans)
    show f ` rel_interior S ⊆ affine hull f ` S
      by (meson ‹f ` rel_interior S ⊆ f ` S› hull_subset order_trans)
    show continuous_on (rel_interior S) f
      using contf continuous_on_subset rel_interior_subset by blast
    show inj_on f (rel_interior S)
      using inj_on_subset injf rel_interior_subset by blast
  qed auto
qed

lemma homeomorphic_rel_interiors_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T and aff: aff_dim S = aff_dim T
  shows (rel_interior S) homeomorphic (rel_interior T)
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix f g
  assume S: ∀x∈S. f x ∈ T ∧ g (f x) = x and T: ∀y∈T. g y ∈ S ∧ f (g y) = y
    and contf: continuous_on S f and contg: continuous_on T g
  then have fST: f ` S = T and gTS: g ` T = S and inj_on f S inj_on g T
    by (auto simp: inj_on_def intro: rev_image_eqI) metis+
  have fim: f ` rel_interior S ⊆ rel_interior T
    by (metis ‹inj_on f S› aff contf continuous_image_subset_rel_interior fST
order_refl)
  have gim: g ` rel_interior T ⊆ rel_interior S
    by (metis ‹inj_on g T› aff contg continuous_image_subset_rel_interior gTS
order_refl)
  show homeomorphism (rel_interior S) (rel_interior T) f g
    unfolding homeomorphism_def
  proof (intro conjI ballI)
    show gf: ⋀x. x ∈ rel_interior S ⟹ g (f x) = x
      using S rel_interior_subset by blast
    show fg: ⋀y. y ∈ rel_interior T ⟹ f (g y) = y
      using T mem_rel_interior_ball by blast
    have rel_interior T ⊆ f ` rel_interior S
    proof
      fix x assume x ∈ rel_interior T
      then have g x ∈ rel_interior S
        using gim by blast
      then show x ∈ f ` rel_interior S
        by (metis fg ‹x ∈ rel_interior T› imageI)
    qed
    moreover have f ` rel_interior S ⊆ rel_interior T
      by (metis ‹inj_on f S› aff contf continuous_image_subset_rel_interior fST
order_refl)
    ultimately show f ` rel_interior S = rel_interior T
      by blast
```

```
    show continuous_on (rel_interior S) f
      using contf continuous_on_subset rel_interior_subset by blast
    have rel_interior S ⊆ g ` rel_interior T
    proof
      fix x assume x ∈ rel_interior S
      then have f x ∈ rel_interior T
        using fim by blast
      then show x ∈ g ` rel_interior T
        by (metis gf ‹x ∈ rel_interior S› imageI)
    qed
    then show g ` rel_interior T = rel_interior S
      using gim by blast
    show continuous_on (rel_interior T) g
      using contg continuous_on_subset rel_interior_subset by blast
  qed
qed

lemma homeomorphic_rel_interiors:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T rel_interior S = {} ⟷ rel_interior T = {}
    shows (rel_interior S) homeomorphic (rel_interior T)
proof (cases rel_interior T = {})
  case True
  with assms show ?thesis by auto
next
  case False
  obtain f g
    where S: ∀x∈S. f x ∈ T ∧ g (f x) = x and T: ∀y∈T. g y ∈ S ∧ f (g y) = y
      and contf: continuous_on S f and contg: continuous_on T g
    using  assms [unfolded homeomorphic_minimal] by auto
  have aff_dim (affine hull S) ≤ aff_dim (affine hull T)
    apply (rule invariance_of_dimension_affine_sets [of _ rel_interior S _ f])
          apply (simp_all add: openin_rel_interior False assms)
    using contf continuous_on_subset rel_interior_subset apply blast
     apply (meson S hull_subset image_subsetI rel_interior_subset rev_subsetD)
    apply (metis S inj_on_inverseI inj_on_subset rel_interior_subset)
    done
  moreover have aff_dim (affine hull T) ≤ aff_dim (affine hull S)
    apply (rule invariance_of_dimension_affine_sets [of _ rel_interior T _ g])
          apply (simp_all add: openin_rel_interior False assms)
    using contg continuous_on_subset rel_interior_subset apply blast
     apply (meson T hull_subset image_subsetI rel_interior_subset rev_subsetD)
    apply (metis T inj_on_inverseI inj_on_subset rel_interior_subset)
    done
  ultimately have aff_dim S = aff_dim T by force
  then show ?thesis
   by (rule homeomorphic_rel_interiors_same_dimension [OF ‹S homeomorphic
T›])
qed
```

lemma *homeomorphic_rel_boundaries_same_dimension*:
  fixes *S* :: *'a::euclidean_space set* **and** *T* :: *'b::euclidean_space set*
  assumes *S homeomorphic T* **and** *aff*: *aff_dim S* = *aff_dim T*
  shows (*S* − *rel_interior S*) *homeomorphic* (*T* − *rel_interior T*)
  using *assms* [*unfolded homeomorphic_minimal*]
  unfolding *homeomorphic_def*
proof (*clarify elim*!: *ex_forward*)
  fix *f g*
  assume *S*: ∀ *x*∈*S*. *f x* ∈ *T* ∧ *g* (*f x*) = *x* **and** *T*: ∀ *y*∈*T*. *g y* ∈ *S* ∧ *f* (*g y*) = *y*
    and *contf*: *continuous_on S f* **and** *contg*: *continuous_on T g*
  then have *fST*: *f* ‘ *S* = *T* **and** *gTS*: *g* ‘ *T* = *S* **and** *inj_on f S inj_on g T*
   by (*auto simp*: *inj_on_def intro*: *rev_image_eqI*) *metis*+
  have *fim*: *f* ‘ *rel_interior S* ⊆ *rel_interior T*
    by (*metis* ‹*inj_on f S*› *aff contf continuous_image_subset_rel_interior fST order_refl*)
  have *gim*: *g* ‘ *rel_interior T* ⊆ *rel_interior S*
   by (*metis* ‹*inj_on g T*› *aff contg continuous_image_subset_rel_interior gTS order_refl*)
  show *homeomorphism* (*S* − *rel_interior S*) (*T* − *rel_interior T*) *f g*
   unfolding *homeomorphism_def*
  proof (*intro conjI ballI*)
   show *gf*: ⋀*x*. *x* ∈ *S* − *rel_interior S* ⟹ *g* (*f x*) = *x*
    using *S rel_interior_subset* **by** *blast*
   show *fg*: ⋀*y*. *y* ∈ *T* − *rel_interior T* ⟹ *f* (*g y*) = *y*
    using *T mem_rel_interior_ball* **by** *blast*
   show *f* ‘ (*S* − *rel_interior S*) = *T* − *rel_interior T*
    using *S fST fim gim* **by** *auto*
   show *continuous_on* (*S* − *rel_interior S*) *f*
    using *contf continuous_on_subset rel_interior_subset* **by** *blast*
   show *g* ‘ (*T* − *rel_interior T*) = *S* − *rel_interior S*
    using *T gTS gim fim* **by** *auto*
   show *continuous_on* (*T* − *rel_interior T*) *g*
    using *contg continuous_on_subset rel_interior_subset* **by** *blast*
  qed
qed

lemma *homeomorphic_rel_boundaries*:
  fixes *S* :: *'a::euclidean_space set* **and** *T* :: *'b::euclidean_space set*
  assumes *S homeomorphic T rel_interior S* = {} ⟷ *rel_interior T* = {}
   shows (*S* − *rel_interior S*) *homeomorphic* (*T* − *rel_interior T*)
proof (*cases rel_interior T* = {})
  case *True*
  with *assms* **show** *?thesis* **by** *auto*
next
  case *False*
  obtain *f g*
   where *S*: ∀ *x*∈*S*. *f x* ∈ *T* ∧ *g* (*f x*) = *x* **and** *T*: ∀ *y*∈*T*. *g y* ∈ *S* ∧ *f* (*g y*) = *y*

```
      and contf: continuous_on S f and contg: continuous_on T g
    using  assms [unfolded homeomorphic_minimal] by auto
  have aff_dim (affine hull S) ≤ aff_dim (affine hull T)
    apply (rule invariance_of_dimension_affine_sets [of _ rel_interior S _ f])
          apply (simp_all add: openin_rel_interior False assms)
    using contf continuous_on_subset rel_interior_subset apply blast
     apply (meson S hull_subset image_subsetI rel_interior_subset rev_subsetD)
    apply (metis S inj_on_inverseI inj_on_subset rel_interior_subset)
    done
  moreover have aff_dim (affine hull T) ≤ aff_dim (affine hull S)
    apply (rule invariance_of_dimension_affine_sets [of _ rel_interior T _ g])
          apply (simp_all add: openin_rel_interior False assms)
    using contg continuous_on_subset rel_interior_subset apply blast
     apply (meson T hull_subset image_subsetI rel_interior_subset rev_subsetD)
    apply (metis T inj_on_inverseI inj_on_subset rel_interior_subset)
    done
  ultimately have aff_dim S = aff_dim T by force
  then show ?thesis
     by (rule homeomorphic_rel_boundaries_same_dimension [OF ‹S homeomor-
phic T›])
qed

proposition uniformly_continuous_homeomorphism_UNIV_trivial:
  fixes f :: 'a::euclidean_space ⇒ 'a
   assumes contf: uniformly_continuous_on S f and hom: homeomorphism S
UNIV f g
  shows S = UNIV
proof (cases S = {})
  case True
  then show ?thesis
    by (metis UNIV_I hom empty_iff homeomorphism_def image_eqI)
next
  case False
  have inj g
    by (metis UNIV_I hom homeomorphism_apply2 injI)
  then have open (g ` UNIV)
    by (blast intro: invariance_of_domain hom homeomorphism_cont2)
  then have open S
    using hom homeomorphism_image2 by blast
  moreover have complete S
    unfolding complete_def
  proof clarify
    fix σ
    assume σ: ∀ n. σ n ∈ S and Cauchy σ
    have Cauchy (f o σ)
      using uniformly_continuous_imp_Cauchy_continuous ‹Cauchy σ› σ contf
      unfolding Cauchy_continuous_on_def by blast
    then obtain l where (f ∘ σ) ⟶ l
      by (auto simp: convergent_eq_Cauchy [symmetric])
```

show ∃ *l*∈*S*. *σ* ⟶ *l*
proof
show *g l* ∈ *S*
using *hom homeomorphism_image2* **by** *blast*
have (*g* ∘ (*f* ∘ *σ*)) ⟶ *g l*
by (*meson UNIV_I* ‹(*f* ∘ *σ*) ⟶ *l*› *continuous_on_sequentially hom homeomorphism_cont2*)
then show *σ* ⟶ *g l*
proof −
have ∀ *n*. *σ n* = (*g* ∘ (*f* ∘ *σ*)) *n*
by (*metis* (*no_types*) *σ comp_eq_dest_lhs hom homeomorphism_apply1*)
then show *?thesis*
by (*metis* (*no_types*) *LIMSEQ_iff* ‹(*g* ∘ (*f* ∘ *σ*)) ⟶ *g l*›)
qed
qed
qed
then have *closed S*
by (*simp add*: *complete_eq_closed*)
ultimately show *?thesis*
using *clopen* [*of S*] *False* **by** *simp*
qed

proposition *invariance_of_domain_sphere_affine_set_gen*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *contf*: *continuous_on S f* **and** *injf*: *inj_on f S* **and** *fim*: *f* ‘ *S* ⊆ *T*
and *U*: *bounded U convex U*
and *affine T* **and** *affTU*: *aff_dim T* < *aff_dim U*
and *ope*: *openin* (*top_of_set* (*rel_frontier U*)) *S*
shows *openin* (*top_of_set T*) (*f* ‘ *S*)
proof (*cases rel_frontier U* = {})
case *True*
then show *?thesis*
using *ope openin_subset* **by** *force*
next
case *False*
obtain *b c* **where** *b*: *b* ∈ *rel_frontier U* **and** *c*: *c* ∈ *rel_frontier U* **and** *b* ≠ *c*
using ‹*bounded U*› *rel_frontier_not_sing* [*of U*] *subset_singletonD False* **by** *fastforce*
obtain *V* :: *'a set* **where** *affine V* **and** *affV*: *aff_dim V* = *aff_dim U* − *1*
proof (*rule choose_affine_subset* [*OF affine_UNIV*])
show − *1* ≤ *aff_dim U* − *1*
by (*metis aff_dim_empty aff_dim_geq aff_dim_negative_iff affTU diff_0 diff_right_mono not_le*)
show *aff_dim U* − *1* ≤ *aff_dim* (*UNIV*::*'a set*)
by (*metis aff_dim_UNIV aff_dim_le_DIM le_cases not_le zle_diff1_eq*)
qed *auto*
have *SU*: *S* ⊆ *rel_frontier U*
using *ope openin_imp_subset* **by** *auto*
have *homb*: *rel_frontier U* − {*b*} *homeomorphic V*

and *homc*: *rel_frontier U* − {*c*} *homeomorphic V*
using *homeomorphic_punctured_sphere_affine_gen* [*of U* _ *V*]
by (*simp_all add*: ‹*affine V*› *affV U b c*)
then obtain *g h j k*
where *gh*: *homeomorphism* (*rel_frontier U* − {*b*}) *V g h*
and *jk*: *homeomorphism* (*rel_frontier U* − {*c*}) *V j k*
by (*auto simp*: *homeomorphic_def*)
with *SU* **have** *hgsub*: (*h* ‘ *g* ‘ (*S* − {*b*})) ⊆ *S* **and** *kjsub*: (*k* ‘ *j* ‘ (*S* − {*c*})) ⊆ *S*
by (*simp_all add*: *homeomorphism_def subset_eq*)
have [*simp*]: *aff_dim T* ≤ *aff_dim V*
by (*simp add*: *affTU affV*)
have *openin* (*top_of_set T*) ((*f* ∘ *h*) ‘ *g* ‘ (*S* − {*b*}))
proof (*rule invariance_of_domain_affine_sets* [*OF* _ ‹*affine V*›])
show *openin* (*top_of_set V*) (*g* ‘ (*S* − {*b*}))
apply (*rule homeomorphism_imp_open_map* [*OF gh*])
by (*meson Diff_mono Diff_subset SU ope openin_delete openin_subset_trans order_refl*)
show *continuous_on* (*g* ‘ (*S* − {*b*})) (*f* ∘ *h*)
apply (*rule continuous_on_compose*)
apply (*meson Diff_mono SU homeomorphism_def homeomorphism_of_subsets gh set_eq_subset*)
using *contf continuous_on_subset hgsub* **by** *blast*
show *inj_on* (*f* ∘ *h*) (*g* ‘ (*S* − {*b*}))
using *kjsub*
apply (*clarsimp simp add*: *inj_on_def*)
by (*metis SU b homeomorphism_def inj_onD injf insert_Diff insert_iff gh rev_subsetD*)
show (*f* ∘ *h*) ‘ *g* ‘ (*S* − {*b*}) ⊆ *T*
by (*metis fim image_comp image_mono hgsub subset_trans*)
qed (*auto simp*: *assms*)
moreover
have *openin* (*top_of_set T*) ((*f* ∘ *k*) ‘ *j* ‘ (*S* − {*c*}))
proof (*rule invariance_of_domain_affine_sets* [*OF* _ ‹*affine V*›])
show *openin* (*top_of_set V*) (*j* ‘ (*S* − {*c*}))
apply (*rule homeomorphism_imp_open_map* [*OF jk*])
by (*meson Diff_mono Diff_subset SU ope openin_delete openin_subset_trans order_refl*)
show *continuous_on* (*j* ‘ (*S* − {*c*})) (*f* ∘ *k*)
apply (*rule continuous_on_compose*)
apply (*meson Diff_mono SU homeomorphism_def homeomorphism_of_subsets jk set_eq_subset*)
using *contf continuous_on_subset kjsub* **by** *blast*
show *inj_on* (*f* ∘ *k*) (*j* ‘ (*S* − {*c*}))
using *kjsub*
apply (*clarsimp simp add*: *inj_on_def*)
by (*metis SU c homeomorphism_def inj_onD injf insert_Diff insert_iff jk rev_subsetD*)
show (*f* ∘ *k*) ‘ *j* ‘ (*S* − {*c*}) ⊆ *T*
by (*metis fim image_comp image_mono kjsub subset_trans*)

qed (*auto simp*: *assms*)
ultimately have *openin* (*top_of_set T*) ((*f* ∘ *h*) ‘ *g* ‘ (*S* − {*b*}) ∪ ((*f* ∘ *k*) ‘ *j* ‘ (*S* − {*c*})))
by (*rule openin_Un*)
moreover have (*f* ∘ *h*) ‘ *g* ‘ (*S* − {*b*}) = *f* ‘ (*S* − {*b*})
proof −
have *h* ‘ *g* ‘ (*S* − {*b*}) = (*S* − {*b*})
proof
show *h* ‘ *g* ‘ (*S* − {*b*}) ⊆ *S* − {*b*}
using *homeomorphism_apply1* [*OF gh*] *SU*
by (*fastforce simp add*: *image_iff image_subset_iff*)
show *S* − {*b*} ⊆ *h* ‘ *g* ‘ (*S* − {*b*})
apply *clarify*
by (*metis SU subsetD homeomorphism_apply1* [*OF gh*] *image_iff member_remove remove_def*)
qed
then show *?thesis*
by (*metis image_comp*)
qed
moreover have (*f* ∘ *k*) ‘ *j* ‘ (*S* − {*c*}) = *f* ‘ (*S* − {*c*})
proof −
have *k* ‘ *j* ‘ (*S* − {*c*}) = (*S* − {*c*})
proof
show *k* ‘ *j* ‘ (*S* − {*c*}) ⊆ *S* − {*c*}
using *homeomorphism_apply1* [*OF jk*] *SU*
by (*fastforce simp add*: *image_iff image_subset_iff*)
show *S* − {*c*} ⊆ *k* ‘ *j* ‘ (*S* − {*c*})
apply *clarify*
by (*metis SU subsetD homeomorphism_apply1* [*OF jk*] *image_iff member_remove remove_def*)
qed
then show *?thesis*
by (*metis image_comp*)
qed
moreover have *f* ‘ (*S* − {*b*}) ∪ *f* ‘ (*S* − {*c*}) = *f* ‘ (*S*)
using ‹*b* ≠ *c*› **by** *blast*
ultimately show *?thesis*
by *simp*
qed

lemma *invariance_of_domain_sphere_affine_set*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *contf*: *continuous_on S f* **and** *injf*: *inj_on f S* **and** *fim*: *f* ‘ *S* ⊆ *T*
and *r* ≠ *0 affine T* **and** *affTU*: *aff_dim T* < *DIM*(*'a*)
and *ope*: *openin* (*top_of_set* (*sphere a r*)) *S*
shows *openin* (*top_of_set T*) (*f* ‘ *S*)
proof (*cases sphere a r* = {})
case *True*
then show *?thesis*

```
    using ope openin_subset by force
next
  case False
  show ?thesis
   proof (rule invariance_of_domain_sphere_affine_set_gen [OF contf injf fim
bounded_cball convex_cball ‹affine T›])
    show aff_dim T < aff_dim (cball a r)
     by (metis False affTU aff_dim_cball assms(4) linorder_cases sphere_empty)
    show openin (top_of_set (rel_frontier (cball a r))) S
      by (simp add: ‹r ≠ 0› ope)
  qed
qed

lemma no_embedding_sphere_lowdim:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes contf: continuous_on (sphere a r) f and injf: inj_on f (sphere a r)
and r > 0
   shows DIM('a) ≤ DIM('b)
proof -
  have False if DIM('a) > DIM('b)
  proof -
    have compact (f ` sphere a r)
      using compact_continuous_image
      by (simp add: compact_continuous_image contf)
    then have ¬ open (f ` sphere a r)
      using compact_open
     by (metis assms(3) image_is_empty not_less_iff_gr_or_eq sphere_eq_empty)
    then show False
     using invariance_of_domain_sphere_affine_set [OF contf injf subset_UNIV]
‹r > 0›
     by (metis aff_dim_UNIV affine_UNIV less_irrefl of_nat_less_iff open_openin
openin_subtopology_self subtopology_UNIV that)
  qed
  then show ?thesis
    using not_less by blast
qed

lemma empty_interior_lowdim_gen:
  fixes S :: 'N::euclidean_space set and T :: 'M::euclidean_space set
  assumes dim: DIM('M) < DIM('N) and ST: S homeomorphic T
  shows interior S = {}
proof -
  obtain h :: 'M ⇒ 'N where linear h ⋀x. norm(h x) = norm x
   by (rule isometry_subset_subspace [OF subspace_UNIV subspace_UNIV, where
?'a = 'M and ?'b = 'N])
       (use dim in auto)
  then have inj h
    by (metis linear_inj_iff_eq_0 norm_eq_zero)
  then have h ` T homeomorphic T
```

using ‹*linear h*› *homeomorphic_sym linear_homeomorphic_image* **by** *blast*
then have *interior* (*h* ‘ *T*) *homeomorphic interior S*
using *homeomorphic_interiors_same_dimension*
by (*metis ST homeomorphic_sym homeomorphic_trans*)
moreover
have *interior* (*range h*) = {}
by (*simp add*: ‹*inj h*› ‹*linear h*› *dim dim_image_eq empty_interior_lowdim*)
then have *interior* (*h* ‘ *T*) = {}
by (*metis image_mono interior_mono subset_empty top_greatest*)
ultimately show *?thesis*
by *simp*
qed

lemma *empty_interior_lowdim_gen_le*:
fixes *S* :: *'N*::*euclidean_space set* **and** *T* :: *'M*::*euclidean_space set*
assumes *DIM*(*'M*) ≤ *DIM*(*'N*) *interior T* = {} *S homeomorphic T*
shows *interior S* = {}
by (*metis assms empty_interior_lowdim_gen homeomorphic_empty*(*1*) *homeomorphic_interiors_same_dimension less_le*)

lemma *homeomorphic_affine_sets_eq*:
fixes *S* :: *'a*::*euclidean_space set* **and** *T* :: *'b*::*euclidean_space set*
assumes *affine S affine T*
shows *S homeomorphic T* ⟷ *aff_dim S* = *aff_dim T*
proof (*cases S* = {} ∨ *T* = {})
case *True*
then show *?thesis*
using *assms homeomorphic_affine_sets* **by** *force*
next
case *False*
then obtain *a b* **where** *a* ∈ *S b* ∈ *T*
by *blast*
then have *subspace* ((+) (− *a*) ‘ *S*) *subspace* ((+) (− *b*) ‘ *T*)
using *affine_diffs_subspace assms* **by** *blast+*
then show *?thesis*
by (*metis affine_imp_convex assms homeomorphic_affine_sets homeomorphic_convex_sets*)
qed

lemma *homeomorphic_hyperplanes_eq*:
fixes *a* :: *'M*::*euclidean_space* **and** *c* :: *'N*::*euclidean_space*
assumes *a* ≠ *0 c* ≠ *0*
shows ({*x*. *a* · *x* = *b*} *homeomorphic* {*x*. *c* · *x* = *d*} ⟷ *DIM*(*'M*) = *DIM*(*'N*)) (**is** *?lhs* = *?rhs*)
proof −
have (*DIM*(*'M*) − *Suc 0* = *DIM*(*'N*) − *Suc 0*) ⟷ (*DIM*(*'M*) = *DIM*(*'N*))
by *auto* (*metis DIM_positive Suc_pred*)
then show *?thesis*
using *assms* **by** (*simp add*: *homeomorphic_affine_sets_eq affine_hyperplane*)

qed

end
theory *Homology*
  imports *Invariance_of_Domain*
begin

end