

The Banach–Tarski Paradox in Isabelle/HOL

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

Abstract

We formalise in Isabelle/HOL the classical Banach–Tarski paradox for the closed unit ball in three-dimensional Euclidean space. The development proves the free-group paradox, the free subgroup of $SO(3)$, the Hausdorff paradox away from a countable bad set, the absorption arguments for the sphere and the origin, and the final finite-partition theorem for the ball.

The proof follows the standard route: the paradoxical decomposition of the free group on two generators F_2 , the existence of a free subgroup $F_2 \leq SO(3)$ via the AFP `Free-Groups` entry, the Hausdorff paradox for $S^2 \setminus D$ where D is a countable set of fixed points, and absorption arguments extending the partition to S^2 , then radially to $B^3 \setminus \{0\}$, and finally to B^3 itself. The argument crucially relies on the axiom of choice.

Background

The Banach–Tarski paradox was proved by Banach and Tarski [1], building on Hausdorff’s earlier paradoxical decomposition of the sphere [4]. The route formalised here follows the standard free-group presentation from Wagon’s expositions [7, 6] and can be compared with de Rauglaudre’s Coq formalisation [3]. The Isabelle development uses Breitner’s AFP entry on free groups [2].

Formalisation Profile

The generated Isabelle session is `Banach_Tarski`. It is based on the HOL Analysis library, imports the AFP entry `Free-Groups`, and uses the `Primes` theory for the arithmetic freeness argument. The development introduces no local axiomatization or oracle. Its non-constructive steps use the standard Hilbert-choice operators of HOL, made explicit in the orbit-representative construction and inverse-map uses.

The checked target is Isabelle 2025-2 with AFP 2026-04-09.

Contents

1	The unit sphere and unit ball in \mathbb{R}^3	3
2	Disjoint finite covers	3
3	Transporting finite partitions	8
4	Group actions on a set	20
5	Paradoxical decomposition	20
6	Equidecomposability	22
7	Free actions	23
8	Choice of orbit representatives	23
9	Transport of paradoxical decompositions along free actions	24
10	Two-element generator type	24
11	The free group F_2 and its carrier	25
12	The four “starts-with” classes of F_2	27
13	The two distinguished generators a and b	28
14	The action of F_2 on itself by left multiplication	29
15	Behaviour of left multiplication by a	29
16	Image computations	30
17	The paradoxical decomposition of F_2	34
18	Rotations of three-space	43
19	Two distinguished rotations	44
20	The rotations as bijections of the sphere	47
21	The map sigma: F_2 to $\text{SO}(3)$	50
22	Arithmetic freeness invariant	58
23	The bad set of fixed points	74

24 Free action of F_2 on $S^2 \setminus D$	82
25 Transporting the free-group paradox to $S^2 \setminus D$	84
26 The Hausdorff paradox	92

```

theory BT-Prelim
  imports
    HOL-Analysis.Analysis
    Free-Groups.FreeGroups
begin

```

1 The unit sphere and unit ball in \mathbb{R}^3

We work throughout with vectors in $(real, 3) \text{ vec}$ (the type $(real, 3) \text{ vec}$ from *HOL-Analysis.Cartesian-Euclidean-Space*). These abbreviations name the principal geometric objects of interest.

```

definition sphere2 ::  $(real^3)$  set where
  sphere2 = {x. norm x = 1}

```

```

definition ball3 ::  $(real^3)$  set where
  ball3 = {x. norm x ≤ 1}

```

```

lemma sphere2-subset-ball3: sphere2 ⊆ ball3
  by (auto simp: sphere2-def ball3-def)

```

```

lemma zero-in-ball3: 0 ∈ ball3
  by (simp add: ball3-def)

```

```

lemma zero-notin-sphere2: (0:: $real^3$ ) ∉ sphere2
  by (simp add: sphere2-def)

```

2 Disjoint finite covers

A *partition* of a set into a finite list of pairwise disjoint pieces is the basic combinatorial object underlying “equidecomposability”.

```

definition pairwise-disjoint :: 'a set list ⇒ bool where
  pairwise-disjoint Xs ⇔
    (∀ i < length Xs. ∀ j < length Xs. i ≠ j ⇒ Xs ! i ∩ Xs ! j = {})

```

```

lemma pairwise-disjoint-Nil [simp]: pairwise-disjoint []
  by (simp add: pairwise-disjoint-def)

```

```

lemma pairwise-disjoint-singleton [simp]: pairwise-disjoint [X]
  by (simp add: pairwise-disjoint-def)

```

lemma *pairwise-disjoint-nthD*:
assumes *pairwise-disjoint Xs*
and $i < \text{length } Xs$ **and** $j < \text{length } Xs$ **and** $i \neq j$
shows $Xs ! i \cap Xs ! j = \{\}$
using *assms bot-set-def pairwise-disjoint-def* **by** *auto*

lemma *indexed-union-set*:
fixes $P :: 'a \text{ set list}$
shows $\bigcup_{i < \text{length } P} P ! i = \bigcup (\text{set } P)$
by (*auto simp add: set-conv-nth image-def*)

lemma *indexed-union-append*:
fixes $P Q :: 'a \text{ set list}$
shows $\bigcup_{i < \text{length } (P @ Q)} (P @ Q) ! i =$
 $\bigcup_{i < \text{length } P} P ! i \cup \bigcup_{i < \text{length } Q} Q ! i$
by (*metis Union-Un-distrib indexed-union-set set-append*)

lemma *indexed-image-union-zip*:
fixes $P :: 'a \text{ set list}$ **and** $G :: ('a \Rightarrow 'b) \text{ list}$
assumes $\text{length } P = \text{length } G$
shows $\bigcup_{i < \text{length } P} G ! i \text{ ` } (P ! i) =$
 $\bigcup_{(g, A) \in \text{set } (\text{zip } G P)} g \text{ ` } A$
using *assms* **by** (*auto simp add: set-zip*)

lemma *indexed-image-union-append*:
fixes $P Q :: 'a \text{ set list}$ **and** $G H :: ('a \Rightarrow 'b) \text{ list}$
assumes *lenP: length P = length G* **and** *lenQ: length Q = length H*
shows $\bigcup_{i < \text{length } (P @ Q)} (G @ H) ! i \text{ ` } ((P @ Q) ! i) =$
 $\bigcup_{i < \text{length } P} G ! i \text{ ` } (P ! i) \cup$
 $\bigcup_{i < \text{length } Q} H ! i \text{ ` } (Q ! i)$
proof –
have *len-append: length (P @ Q) = length (G @ H)*
using *lenP lenQ* **by** *simp*
have $\bigcup_{i < \text{length } (P @ Q)} (G @ H) ! i \text{ ` } ((P @ Q) ! i) =$
 $\bigcup_{(g, A) \in \text{set } (\text{zip } (G @ H) (P @ Q))} g \text{ ` } A$
using *indexed-image-union-zip[OF len-append]* .
also have ... =
 $\bigcup_{(g, A) \in \text{set } (\text{zip } G P)} g \text{ ` } A \cup$
 $\bigcup_{(g, A) \in \text{set } (\text{zip } H Q)} g \text{ ` } A$
using *lenP* **by** *simp*
also have ... =
 $\bigcup_{i < \text{length } P} G ! i \text{ ` } (P ! i) \cup$
 $\bigcup_{i < \text{length } Q} H ! i \text{ ` } (Q ! i)$
using *indexed-image-union-zip[OF lenP]* *indexed-image-union-zip[OF lenQ]* **by**
simp
finally show *?thesis* .
qed

lemma *indexed-union-map-fst*:

fixes $xs :: ('a \text{ set} \times 'b) \text{ list}$

shows $(\bigcup_{i < \text{length} (\text{map } \text{fst } xs)}. \text{map } \text{fst } xs ! i) =$
 $(\bigcup_{x \in \text{set } xs}. \text{fst } x)$

by (*auto simp add: set-conv-nth*)

lemma *indexed-image-union-pairs*:

fixes $xs :: ('a \text{ set} \times ('a \Rightarrow 'b)) \text{ list}$

shows $(\bigcup_{i < \text{length} (\text{map } \text{fst } xs)}. \text{map } \text{snd } xs ! i \text{ ' } (\text{map } \text{fst } xs ! i)) =$
 $(\bigcup_{x \in \text{set } xs}. \text{snd } x \text{ ' } \text{fst } x)$

by (*auto simp add: set-conv-nth*)

lemma *pairwise-disjoint-map-fstI*:

fixes $xs :: ('a \text{ set} \times 'b) \text{ list}$

assumes *distinct: distinct xs*

and *disj*: $\bigwedge x y. \llbracket x \in \text{set } xs; y \in \text{set } xs; x \neq y \rrbracket$
 $\implies \text{fst } x \cap \text{fst } y = \{\}$

shows *pairwise-disjoint* (*map* *fst* *xs*)

unfolding *pairwise-disjoint-def*

proof (*intro allI impI*)

fix $i j$

assume $i: i < \text{length} (\text{map } \text{fst } xs)$ **and** $j: j < \text{length} (\text{map } \text{fst } xs)$ **and** $ij: i \neq j$

have $xi: xs ! i \in \text{set } xs$ **and** $xj: xs ! j \in \text{set } xs$

using $i j$ **by** *auto*

have *neq*: $xs ! i \neq xs ! j$

using *distinct* $i j$ **by** (*simp add: nth-eq-iff-index-eq*)

show $\text{map } \text{fst } xs ! i \cap \text{map } \text{fst } xs ! j = \{\}$

using *disj[OF xi xj neq]* $i j$ **by** *simp*

qed

lemma *pairwise-disjoint-appendI*:

assumes $xs: \text{pairwise-disjoint } xs$

and $ys: \text{pairwise-disjoint } ys$

and *cross*: $\bigwedge x y. \llbracket x \in \text{set } xs; y \in \text{set } ys \rrbracket \implies x \cap y = \{\}$

shows *pairwise-disjoint* ($xs @ ys$)

unfolding *pairwise-disjoint-def*

proof (*intro allI impI*)

fix $i j$

assume $i: i < \text{length} (xs @ ys)$ **and** $j: j < \text{length} (xs @ ys)$ **and** $ij: i \neq j$

show $(xs @ ys) ! i \cap (xs @ ys) ! j = \{\}$

proof (*cases* $i < \text{length } xs$)

case *True*

show *?thesis*

proof (*cases* $j < \text{length } xs$)

case *True*

have $xs ! i \cap xs ! j = \{\}$

by (*rule pairwise-disjoint-nthD[OF xs <i < length xs> True ij]*)

with $\langle i < \text{length } xs \rangle$ *True* **show** *?thesis*

by (*simp add: nth-append*)

```

next
  case False
  hence  $(xs @ ys) ! j \in \text{set } ys$ 
    using j by (simp add: nth-append)
  moreover have  $(xs @ ys) ! i \in \text{set } xs$ 
    by (simp add: True nth-append-left)
  ultimately show ?thesis
    using cross by simp
qed
next
  case False
  show ?thesis
  proof (cases j < length xs)
    case True
    hence  $(xs @ ys) ! j \in \text{set } xs$ 
    proof -
      have  $(xs @ ys) ! j = xs ! j$ 
        using True by (simp add: nth-append)
      moreover have  $xs ! j \in \text{set } xs$ 
        by (rule nth-mem[OF True])
      ultimately show ?thesis
        by simp
    qed
    moreover have  $(xs @ ys) ! i \in \text{set } ys$ 
      using False i by (simp add: nth-append)
    ultimately have  $(xs @ ys) ! j \cap (xs @ ys) ! i = \{\}$ 
      using cross by blast
    thus ?thesis
      by (simp add: Int-commute)
  next
    case False
    have  $ys ! (i - \text{length } xs) \cap ys ! (j - \text{length } xs) = \{\}$ 
      by (rule pairwise-disjoint-nthD[OF ys]) (use False <¬ i < length xs> i j ij in
auto)
    with False <¬ i < length xs> i j show ?thesis
      by (simp add: nth-append)
    qed
  qed
qed

```

lemma *pairwise-disjoint-appendD1*:

```

  assumes pairwise-disjoint (xs @ ys)
  shows pairwise-disjoint xs
  unfolding pairwise-disjoint-def
  proof (intro allI impI)
    fix i j
    assume i: i < length xs and j: j < length xs and ij: i ≠ j
    have  $(xs @ ys) ! i \cap (xs @ ys) ! j = \{\}$ 
      by (rule pairwise-disjoint-nthD[OF assms]) (use i j ij in simp-all)
  
```

thus $xs ! i \cap xs ! j = \{\}$
using $i j$ **by** (*simp add: nth-append*)
qed

lemma *pairwise-disjoint-appendD2*:

assumes *pairwise-disjoint* ($xs @ ys$)
shows *pairwise-disjoint* ys
unfolding *pairwise-disjoint-def*
proof (*intro allI impI*)
fix $i j$
assume $i: i < \text{length } ys$ **and** $j: j < \text{length } ys$ **and** $ij: i \neq j$
have $\text{length } xs + i < \text{length } (xs @ ys)$ **and** $\text{length } xs + j < \text{length } (xs @ ys)$
using $i j$ **by** *simp-all*
moreover **have** $\text{length } xs + i \neq \text{length } xs + j$
using ij **by** *simp*
ultimately **have** $(xs @ ys) ! (\text{length } xs + i) \cap (xs @ ys) ! (\text{length } xs + j) = \{\}$
by (*rule pairwise-disjoint-nthD[OF assms]*)
thus $ys ! i \cap ys ! j = \{\}$
using $i j$ **by** *simp*
qed

lemma *pairwise-disjoint-append-crossD*:

assumes *pairwise-disjoint* ($xs @ ys$)
and $i < \text{length } xs$ **and** $j < \text{length } ys$
shows $xs ! i \cap ys ! j = \{\}$
proof –
have $i < \text{length } (xs @ ys)$ **and** $\text{length } xs + j < \text{length } (xs @ ys)$
using *assms* **by** *simp-all*
moreover **have** $i \neq \text{length } xs + j$
using *assms(2)* **by** *simp*
ultimately **have** $(xs @ ys) ! i \cap (xs @ ys) ! (\text{length } xs + j) = \{\}$
by (*rule pairwise-disjoint-nthD[OF assms(1)]*)
moreover **have** $(xs @ ys) ! i = xs ! i$
using *assms(2)* **by** (*simp add: nth-append*)
moreover **have** $(xs @ ys) ! (\text{length } xs + j) = ys ! j$
using *assms(3)* **by** (*simp add: nth-append*)
ultimately **show** *?thesis*
by *simp*
qed

lemma *pairwise-disjoint-pair-imagesI*:

fixes $xs :: ('a \text{ set} \times ('a \Rightarrow 'b)) \text{ list}$
assumes *distinct*: *distinct* xs
and *disj*: $\bigwedge x y. \llbracket x \in \text{set } xs; y \in \text{set } xs; x \neq y \rrbracket$
 $\implies \text{snd } x \text{ 'fst } x \cap \text{snd } y \text{ 'fst } y = \{\}$
shows *pairwise-disjoint* ($\text{map2 } (\lambda g A. g \text{ ' } A) (\text{map } \text{snd } xs) (\text{map } \text{fst } xs)$)
proof –
have *map-eq*: $\text{map2 } (\lambda g A. g \text{ ' } A) (\text{map } \text{snd } xs) (\text{map } \text{fst } xs) =$
 $\text{map } (\lambda x. \text{snd } x \text{ 'fst } x) xs$

```

  by (induction xs) simp-all
show ?thesis
  unfolding map-eq pairwise-disjoint-def
proof (intro allI impI)
  fix i j
  assume i: i < length (map (λx. snd x ' fst x) xs)
  and j: j < length (map (λx. snd x ' fst x) xs) and ij: i ≠ j
  have xi: xs ! i ∈ set xs and xj: xs ! j ∈ set xs
  using i j by auto
  have neq: xs ! i ≠ xs ! j
  using distinct i j ij by (simp add: nth-eq-iff-index-eq)
  show map (λx. snd x ' fst x) xs ! i ∩
    map (λx. snd x ' fst x) xs ! j = {}
  using disj[OF xi xj neq] i j by simp
qed
qed

```

3 Transporting finite partitions

definition *transfer-piece* ::
 $'a \text{ set list} \Rightarrow ('a \Rightarrow 'a) \text{ list} \Rightarrow 'a \text{ set list} \Rightarrow$
 $('a \Rightarrow 'a) \text{ list} \Rightarrow 'a \text{ set list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ set}$
where
transfer-piece $A e P g B k i l =$
 $\{x \in A ! k. (e ! k) x \in P ! i \wedge (g ! i) ((e ! k) x) \in B ! l\}$

definition *transfer-map* ::
 $('a \Rightarrow 'a) \text{ list} \Rightarrow ('a \Rightarrow 'a) \text{ list} \Rightarrow$
 $('a \Rightarrow 'a) \text{ list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow ('a \Rightarrow 'a)$
where
transfer-map $t g e k i l = (t ! l) \circ (g ! i) \circ (e ! k)$

definition *transfer-pairs* ::
 $'a \text{ set list} \Rightarrow ('a \Rightarrow 'a) \text{ list} \Rightarrow 'a \text{ set list} \Rightarrow$
 $('a \Rightarrow 'a) \text{ list} \Rightarrow 'a \text{ set list} \Rightarrow ('a \Rightarrow 'a) \text{ list} \Rightarrow$
 $('a \text{ set} \times ('a \Rightarrow 'a)) \text{ set}$
where
transfer-pairs $A e P g B t =$
 $\{(transfer-piece A e P g B k i l, transfer-map t g e k i l) \mid$
 $k i l. k < length A \wedge i < length P \wedge l < length B\}$

lemma *finite-transfer-pairs* [simp]: *finite* (transfer-pairs $A e P g B t$)

proof –
have *transfer-pairs* $A e P g B t \subseteq$
 $(\lambda((k, i), l). (transfer-piece A e P g B k i l, transfer-map t g e k i l)) \text{ `}$
 $((\{..<length A\} \times \{..<length P\}) \times \{..<length B\})$
proof
fix p
assume $p \in transfer-pairs A e P g B t$

then obtain $k\ i\ l$ **where** $k: k < \text{length } A$ **and** $i: i < \text{length } P$ **and** $l: l < \text{length } B$
and $p\text{-eq}: p = (\text{transfer-piece } A\ e\ P\ g\ B\ k\ i\ l, \text{transfer-map } t\ g\ e\ k\ i\ l)$
unfolding $\text{transfer-pairs-def}$ **by** blast
have $\text{tril-in}: ((k, i), l) \in (\{..<\text{length } A\} \times \{..<\text{length } P\}) \times \{..<\text{length } B\}$
using $k\ i\ l$ **by** simp
show $p \in$
 $(\lambda((k, i), l). (\text{transfer-piece } A\ e\ P\ g\ B\ k\ i\ l, \text{transfer-map } t\ g\ e\ k\ i\ l)) \text{ ‘}$
 $(\{..<\text{length } A\} \times \{..<\text{length } P\}) \times \{..<\text{length } B\}$
using $p\text{-eq}$ tril-in **by** force
qed
thus $?thesis$
by $(\text{rule finite-subset})\ \text{simp}$
qed

lemma $\text{transfer-source-cover}$:

assumes $A\text{-cover}: Y = (\bigcup_{k < \text{length } A} A\ !\ k)$
and $B\text{-cover}: X = (\bigcup_{k < \text{length } B} B\ !\ k)$
and $e\text{-into-}B: \bigwedge k\ x. \llbracket k < \text{length } A; x \in A\ !\ k \rrbracket \implies (e\ !\ k)\ x \in B\ !\ k$
and $\text{len-}AB: \text{length } A = \text{length } B$
and $\text{source-cover}: X = (\bigcup_{i < \text{length } P} P\ !\ i) \cup (\bigcup_{j < \text{length } Q} Q\ !\ j)$
and $\text{image}P\text{-cover}: X = (\bigcup_{i < \text{length } P} gP\ !\ i\ \text{ ‘ } (P\ !\ i))$
and $\text{image}Q\text{-cover}: X = (\bigcup_{j < \text{length } Q} gQ\ !\ j\ \text{ ‘ } (Q\ !\ j))$

shows $Y =$
 $(\bigcup_{p \in \text{transfer-pairs } A\ e\ P\ gP\ B\ t. \text{fst } p}) \cup$
 $(\bigcup_{q \in \text{transfer-pairs } A\ e\ Q\ gQ\ B\ t. \text{fst } q})$

proof

show $Y \subseteq$
 $(\bigcup_{p \in \text{transfer-pairs } A\ e\ P\ gP\ B\ t. \text{fst } p}) \cup$
 $(\bigcup_{q \in \text{transfer-pairs } A\ e\ Q\ gQ\ B\ t. \text{fst } q})$

proof

fix x
assume $xY: x \in Y$
then obtain k **where** $k: k < \text{length } A$ **and** $xA: x \in A\ !\ k$
using $A\text{-cover}$ **by** blast

have $ex: (e\ !\ k)\ x \in X$
using $B\text{-cover}$ $e\text{-into-}B$ k $\text{len-}AB$ xA **by** auto

from source-cover ex **have** $ex\text{-cases}$:

$(e\ !\ k)\ x \in (\bigcup_{i < \text{length } P} P\ !\ i) \vee$
 $(e\ !\ k)\ x \in (\bigcup_{j < \text{length } Q} Q\ !\ j)$

by blast

thus $x \in$
 $(\bigcup_{p \in \text{transfer-pairs } A\ e\ P\ gP\ B\ t. \text{fst } p}) \cup$
 $(\bigcup_{q \in \text{transfer-pairs } A\ e\ Q\ gQ\ B\ t. \text{fst } q})$

proof

assume $(e\ !\ k)\ x \in (\bigcup_{i < \text{length } P} P\ !\ i)$

then obtain i **where** $i: i < \text{length } P$ **and** $xP: (e\ !\ k)\ x \in P\ !\ i$
by blast

have $gxX: (gP\ !\ i)\ ((e\ !\ k)\ x) \in X$

```

    using imageP-cover i xP by blast
  then obtain l where l: l < length B and gxB: (gP ! i) ((e ! k) x) ∈ B ! l
    using B-cover by blast
  have pair-in: (transfer-piece A e P gP B k i l, transfer-map t gP e k i l)
    ∈ transfer-pairs A e P gP B t
    using k i l unfolding transfer-pairs-def by blast
  have x-piece: x ∈ transfer-piece A e P gP B k i l
    using xA xP gxB by (simp add: transfer-piece-def)
  have set-in: fst (transfer-piece A e P gP B k i l, transfer-map t gP e k i l) ∈
    ((λp. fst p) ‘ transfer-pairs A e P gP B t)
    by (rule imageI[OF pair-in])
  have x-fst: x ∈ fst (transfer-piece A e P gP B k i l, transfer-map t gP e k i l)
    using x-piece by simp
  have x ∈ ⋃ ((λp. fst p) ‘ transfer-pairs A e P gP B t)
    using set-in x-fst by (rule UnionI)
  thus ?thesis
    by simp
next
assume (e ! k) x ∈ (⋃ j < length Q. Q ! j)
then obtain j where j: j < length Q and xQ: (e ! k) x ∈ Q ! j
  by blast
have gxX: (gQ ! j) ((e ! k) x) ∈ X
  using imageQ-cover j xQ by blast
then obtain l where l: l < length B and gxB: (gQ ! j) ((e ! k) x) ∈ B ! l
  using B-cover by blast
have pair-in: (transfer-piece A e Q gQ B k j l, transfer-map t gQ e k j l)
  ∈ transfer-pairs A e Q gQ B t
  using k j l unfolding transfer-pairs-def by blast
have x-piece: x ∈ transfer-piece A e Q gQ B k j l
  using xA xQ gxB by (simp add: transfer-piece-def)
have set-in: fst (transfer-piece A e Q gQ B k j l, transfer-map t gQ e k j l) ∈
  ((λq. fst q) ‘ transfer-pairs A e Q gQ B t)
  by (rule imageI[OF pair-in])
have x-fst: x ∈ fst (transfer-piece A e Q gQ B k j l, transfer-map t gQ e k j l)
  using x-piece by simp
have x ∈ ⋃ ((λq. fst q) ‘ transfer-pairs A e Q gQ B t)
  using set-in x-fst by (rule UnionI)
thus ?thesis
  by simp
qed
qed
show (⋃ p ∈ transfer-pairs A e P gP B t. fst p) ∪
  (⋃ q ∈ transfer-pairs A e Q gQ B t. fst q) ⊆ Y
  using A-cover unfolding transfer-pairs-def transfer-piece-def by auto
qed

```

lemma *transfer-image-cover*:

```

assumes A-cover: Y = (⋃ k < length A. A ! k)
  and B-cover: X = (⋃ k < length B. B ! k)

```

and *len-AB*: $\text{length } A = \text{length } B$
and *e-left*: $\bigwedge k x. \llbracket k < \text{length } A; x \in A ! k \rrbracket$
 $\implies (e ! k) x \in B ! k \wedge (t ! k) ((e ! k) x) = x$
and *t-left*: $\bigwedge k y. \llbracket k < \text{length } B; y \in B ! k \rrbracket$
 $\implies (t ! k) y \in A ! k \wedge (e ! k) ((t ! k) y) = y$
and *P-sub*: $\bigwedge i. i < \text{length } P \implies P ! i \subseteq X$
and *image-cover*: $X = (\bigcup_{i < \text{length } P} g ! i \text{ ' } (P ! i))$
shows $Y = (\bigcup_{p \in \text{transfer-pairs } A \text{ e } P \text{ g } B \text{ t. snd } p \text{ ' fst } p}$
proof
show $Y \subseteq (\bigcup_{p \in \text{transfer-pairs } A \text{ e } P \text{ g } B \text{ t. snd } p \text{ ' fst } p}$
proof
fix y
assume $yY: y \in Y$
then obtain l **where** $l: l < \text{length } A$ **and** $yA: y \in A ! l$
using *A-cover* **by** *blast*
have $eyB: (e ! l) y \in B ! l$ **and** $tey: (t ! l) ((e ! l) y) = y$
using *e-left*[*OF l yA*] **by** *auto*
have $eyX: (e ! l) y \in X$
using *B-cover* eyB l *len-AB* **by** *auto*
then obtain i x **where** $i: i < \text{length } P$ **and** $xP: x \in P ! i$
and $ey\text{-eq}: (e ! l) y = (g ! i) x$
using *image-cover* **by** *blast*
have $lB: l < \text{length } B$
using l *len-AB* **by** *simp*
have $xX: x \in X$
using *P-sub*[*OF i*] xP **by** (*rule subsetD*)
then obtain k **where** $k: k < \text{length } B$ **and** $xB: x \in B ! k$
using *B-cover* **by** *blast*
have $txA: (t ! k) x \in A ! k$ **and** $etx: (e ! k) ((t ! k) x) = x$
using *t-left*[*OF k xB*] **by** *auto*
have $kA: k < \text{length } A$
using k *len-AB* **by** *simp*
have *piece-in*: $(t ! k) x \in \text{transfer-piece } A \text{ e } P \text{ g } B \text{ k } i \text{ l}$
using txA etx xP eyB $ey\text{-eq}$ **by** (*simp add: transfer-piece-def*)
have *pair-in*: $(\text{transfer-piece } A \text{ e } P \text{ g } B \text{ k } i \text{ l}, \text{transfer-map } t \text{ g } e \text{ k } i \text{ l})$
 $\in \text{transfer-pairs } A \text{ e } P \text{ g } B \text{ t}$
using kA i lB **unfolding** *transfer-pairs-def* **by** *blast*
have *transfer-map* $t \text{ g } e \text{ k } i \text{ l} ((t ! k) x) = y$
using etx $ey\text{-eq}$ tey **by** (*simp add: transfer-map-def*)
thus $y \in (\bigcup_{p \in \text{transfer-pairs } A \text{ e } P \text{ g } B \text{ t. snd } p \text{ ' fst } p}$
using *pair-in* *piece-in* **by** *force*
qed
show $(\bigcup_{p \in \text{transfer-pairs } A \text{ e } P \text{ g } B \text{ t. snd } p \text{ ' fst } p} \subseteq Y$
proof
fix y
assume $y \in (\bigcup_{p \in \text{transfer-pairs } A \text{ e } P \text{ g } B \text{ t. snd } p \text{ ' fst } p}$
then obtain k i l x **where** $k: k < \text{length } A$ **and** $i: i < \text{length } P$
and $l: l < \text{length } B$ **and** $x\text{-piece}: x \in \text{transfer-piece } A \text{ e } P \text{ g } B \text{ k } i \text{ l}$
and $y\text{-eq}: y = \text{transfer-map } t \text{ g } e \text{ k } i \text{ l } x$

```

    unfolding transfer-pairs-def by auto
  have gxB: (g ! i) ((e ! k) x) ∈ B ! l
    using x-piece by (simp add: transfer-piece-def)
  have tyA: (t ! l) ((g ! i) ((e ! k) x)) ∈ A ! l
    using t-left[OF l gxB] by auto
  hence y ∈ A ! l
    using y-eq by (simp add: transfer-map-def)
  moreover have A ! l ⊆ Y
    using A-cover l len-AB by auto
  ultimately show y ∈ Y
    by blast
qed
qed

lemma transfer-source-disjoint-same:
  assumes A-disj: pairwise-disjoint A
    and P-disj: pairwise-disjoint P
    and B-disj: pairwise-disjoint B
    and x-in: x ∈ transfer-pairs A e P g B t
    and y-in: y ∈ transfer-pairs A e P g B t
    and xy: x ≠ y
  shows fst x ∩ fst y = {}
proof (rule ccontr)
  assume nonempty: fst x ∩ fst y ≠ {}
  obtain k i l where k: k < length A and i: i < length P and l: l < length B
    and x-eq: x = (transfer-piece A e P g B k i l, transfer-map t g e k i l)
    using x-in unfolding transfer-pairs-def by blast
  obtain k' i' l' where k': k' < length A and i': i' < length P and l': l' < length
  B
    and y-eq: y = (transfer-piece A e P g B k' i' l', transfer-map t g e k' i' l')
    using y-in unfolding transfer-pairs-def by blast
  from nonempty obtain z where zx: z ∈ fst x and zy: z ∈ fst y
    by blast
  have zA: z ∈ A ! k and zeP: (e ! k) z ∈ P ! i
    and zgB: (g ! i) ((e ! k) z) ∈ B ! l
    using zx x-eq by (auto simp: transfer-piece-def)
  have zA': z ∈ A ! k' and zeP': (e ! k') z ∈ P ! i'
    and zgB': (g ! i') ((e ! k') z) ∈ B ! l'
    using zy y-eq by (auto simp: transfer-piece-def)
  show False
proof (cases k = k')
  case False
  have A ! k ∩ A ! k' = {}
    by (rule pairwise-disjoint-nthD[OF A-disj k k' False])
  with zA zA' show False
    by blast
next
  case k-eq: True
  show False

```

```

proof (cases i = i')
  case False
  have P ! i ∩ P ! i' = {}
    by (rule pairwise-disjoint-nthD[OF P-disj i i' False])
  moreover have (e ! k) z ∈ P ! i'
    using zeP' k-eq by simp
  ultimately show False
    using zeP by blast
next
  case i-eq: True
  show False
  proof (cases l = l')
    case False
    have B ! l ∩ B ! l' = {}
      by (rule pairwise-disjoint-nthD[OF B-disj l l' False])
    moreover have (g ! i) ((e ! k) z) ∈ B ! l'
      using zgB' k-eq i-eq by simp
    ultimately show False
      using zgB by blast
  next
  case l-eq: True
  have x = y
    using x-eq y-eq k-eq i-eq l-eq by simp
  with xy show False
    by contradiction
  qed
qed
qed
qed

```

lemma transfer-source-disjoint-cross:

```

assumes A-disj: pairwise-disjoint A
  and PQ-cross:  $\bigwedge i j. [i < \text{length } P; j < \text{length } Q] \implies P ! i \cap Q ! j = \{\}$ 
  and B-disj: pairwise-disjoint B
  and x-in:  $x \in \text{transfer-pairs } A \ e \ P \ gP \ B \ t$ 
  and y-in:  $y \in \text{transfer-pairs } A \ e \ Q \ gQ \ B \ t$ 
shows fst x ∩ fst y = {}
proof (rule ccontr)
  assume nonempty: fst x ∩ fst y ≠ {}
  obtain k i l where k: k < length A and i: i < length P and l: l < length B
    and x-eq:  $x = (\text{transfer-piece } A \ e \ P \ gP \ B \ k \ i \ l, \text{transfer-map } t \ gP \ e \ k \ i \ l)$ 
    using x-in unfolding transfer-pairs-def by blast
  obtain k' j l' where k': k' < length A and j: j < length Q and l': l' < length B
    and y-eq:  $y = (\text{transfer-piece } A \ e \ Q \ gQ \ B \ k' \ j \ l', \text{transfer-map } t \ gQ \ e \ k' \ j \ l')$ 
    using y-in unfolding transfer-pairs-def by blast
  from nonempty obtain z where zx: z ∈ fst x and zy: z ∈ fst y
    by blast
  have zA: z ∈ A ! k and zeP: (e ! k) z ∈ P ! i
    using zx x-eq by (auto simp: transfer-piece-def)

```

```

have zA': z ∈ A ! k' and zeQ: (e ! k') z ∈ Q ! j
  using zy y-eq by (auto simp: transfer-piece-def)
show False
proof (cases k = k')
  case False
  have A ! k ∩ A ! k' = {}
    by (rule pairwise-disjoint-nthD[OF A-disj k k' False])
  with zA zA' show False
    by blast
next
  case k-eq: True
  have P ! i ∩ Q ! j = {}
    by (rule PQ-cross[OF i j])
  moreover have (e ! k) z ∈ Q ! j
    using zeQ k-eq by simp
  ultimately show False
    using zeP by blast
qed
qed

lemma transfer-image-disjoint-same:
  assumes len-AB: length A = length B
    and A-disj: pairwise-disjoint A
    and B-disj: pairwise-disjoint B
    and image-disj: pairwise-disjoint (map2 (λh C. h ' C) g P)
    and len: length P = length g
    and g-inj: λi. i < length P ⇒ inj (g ! i)
    and e-into-B: λk x. [k < length A; x ∈ A ! k] ⇒ (e ! k) x ∈ B ! k
    and t-left: λl y. [l < length B; y ∈ B ! l]
      ⇒ (t ! l) y ∈ A ! l ∧ (e ! l) ((t ! l) y) = y
    and x-in: x ∈ transfer-pairs A e P g B t
    and y-in: y ∈ transfer-pairs A e P g B t
    and xy: x ≠ y
  shows snd x ' fst x ∩ snd y ' fst y = {}
proof (rule ccontr)
  assume nonempty: snd x ' fst x ∩ snd y ' fst y ≠ {}
  obtain k i l where k: k < length A and i: i < length P and l: l < length B
    and x-eq: x = (transfer-piece A e P g B k i l, transfer-map t g e k i l)
    using x-in unfolding transfer-pairs-def by blast
  obtain k' i' l' where k': k' < length A and i': i' < length P and l': l' < length
  B
    and y-eq: y = (transfer-piece A e P g B k' i' l', transfer-map t g e k' i' l')
    using y-in unfolding transfer-pairs-def by blast
  from nonempty obtain z where zx: z ∈ snd x ' fst x and zy: z ∈ snd y ' fst y
    by blast
  then obtain a b where
    a-piece: a ∈ transfer-piece A e P g B k i l
    and z-a: z = transfer-map t g e k i l a
    and b-piece: b ∈ transfer-piece A e P g B k' i' l'

```

```

    and z-b: z = transfer-map t g e k' i' l' b
    using x-eq y-eq by auto
  have aA: a ∈ A ! k and aeP: (e ! k) a ∈ P ! i
    and gaB: (g ! i) ((e ! k) a) ∈ B ! l
    using a-piece by (auto simp: transfer-piece-def)
  have bA: b ∈ A ! k' and beP: (e ! k') b ∈ P ! i'
    and gbB: (g ! i') ((e ! k') b) ∈ B ! l'
    using b-piece by (auto simp: transfer-piece-def)
  have lA: l < length A and lA': l' < length A
    using l l' len-AB by simp-all
  have zA-l: z ∈ A ! l
    using t-left[OF l gaB] z-a by (simp add: transfer-map-def)
  have zA-l': z ∈ A ! l'
    using t-left[OF l' gbB] z-b by (simp add: transfer-map-def)
  show False
  proof (cases l = l')
    case False
      have A ! l ∩ A ! l' = {}
        by (rule pairwise-disjoint-nthD[OF A-disj lA lA' False])
      with zA-l zA-l' show False
        by blast
    next
      case l-eq: True
        have ez-a: (e ! l) z = (g ! i) ((e ! k) a)
          using t-left[OF l gaB] z-a by (simp add: transfer-map-def)
        have ez-b: (e ! l) z = (g ! i') ((e ! k') b)
          using t-left[OF l' gbB] z-b l-eq by (simp add: transfer-map-def)
        have g-eq: (g ! i) ((e ! k) a) = (g ! i') ((e ! k') b)
          using ez-a ez-b by simp
        show False
        proof (cases i = i')
          case False then
            have img-i: (g ! i) ((e ! k) a) ∈ map2 (λh C. h ' C) g P ! i
              using aeP i len by simp
            have img-i': (g ! i') ((e ! k') b) ∈ map2 (λh C. h ' C) g P ! i'
              using beP i' len by simp
            have map2 (λh C. h ' C) g P ! i ∩
              map2 (λh C. h ' C) g P ! i' = {}
              by (rule pairwise-disjoint-nthD[OF image-disj]) (use i i' False len in simp-all)
            moreover have (g ! i) ((e ! k) a) ∈ map2 (λh C. h ' C) g P ! i'
              using img-i' g-eq by simp
            ultimately show False
              using img-i by auto
          next
            case i-eq: True
              have e-eq: (e ! k) a = (e ! k') b
                using g-eq i-eq g-inj[OF i] by (auto dest: injD)
              show False
              proof (cases k = k')

```

```

case False
have kB:  $k < \text{length } B$  and kB':  $k' < \text{length } B$ 
  using k k' len-AB by simp-all
have eaB:  $(e ! k) a \in B ! k$ 
  by (rule e-into-B[OF k aA])
have ebB:  $(e ! k') b \in B ! k'$ 
  by (rule e-into-B[OF k' bA])
have  $B ! k \cap B ! k' = \{\}$ 
  by (rule pairwise-disjoint-nthD[OF B-disj kB kB' False])
moreover have  $(e ! k) a \in B ! k'$ 
  using ebB e-eq by simp
ultimately show False
  using eaB by auto
next
case k-eq: True
have  $x = y$ 
  using x-eq y-eq k-eq i-eq l-eq by simp
with xy show False
  by contradiction
qed
qed
qed
qed

```

lemma *transfer-partitioned-paradox*:

```

assumes len-AB:  $\text{length } A = \text{length } B$ 
and A-cover:  $Y = (\bigcup k < \text{length } A. A ! k)$ 
and A-disj: pairwise-disjoint A
and B-cover:  $X = (\bigcup k < \text{length } B. B ! k)$ 
and B-disj: pairwise-disjoint B
and e-left:  $\bigwedge k x. \llbracket k < \text{length } A; x \in A ! k \rrbracket$ 
   $\implies (e ! k) x \in B ! k \wedge (t ! k) ((e ! k) x) = x$ 
and t-left:  $\bigwedge k y. \llbracket k < \text{length } B; y \in B ! k \rrbracket$ 
   $\implies (t ! k) y \in A ! k \wedge (e ! k) ((t ! k) y) = y$ 
and lenP:  $\text{length } P = \text{length } gP$ 
and lenQ:  $\text{length } Q = \text{length } gQ$ 
and source-disj: pairwise-disjoint  $(P @ Q)$ 
and source-cover:  $X = (\bigcup i < \text{length } P. P ! i) \cup (\bigcup j < \text{length } Q. Q ! j)$ 
and imageP-disj: pairwise-disjoint  $(\text{map2 } (\lambda h C. h ' C) gP P)$ 
and imageQ-disj: pairwise-disjoint  $(\text{map2 } (\lambda h C. h ' C) gQ Q)$ 
and imageP-cover:  $X = (\bigcup i < \text{length } P. gP ! i ' (P ! i))$ 
and imageQ-cover:  $X = (\bigcup j < \text{length } Q. gQ ! j ' (Q ! j))$ 
and gP-inj:  $\bigwedge i. i < \text{length } P \implies \text{inj } (gP ! i)$ 
and gQ-inj:  $\bigwedge j. j < \text{length } Q \implies \text{inj } (gQ ! j)$ 
and mapsP:  $\bigwedge k i l. \llbracket k < \text{length } A; i < \text{length } P; l < \text{length } B \rrbracket$ 
   $\implies \text{transfer-map } t gP e k i l \in M$ 
and mapsQ:  $\bigwedge k j l. \llbracket k < \text{length } A; j < \text{length } Q; l < \text{length } B \rrbracket$ 
   $\implies \text{transfer-map } t gQ e k j l \in M$ 
shows  $\exists P' Q' :: 'a \text{ set list. } \exists gP' gQ' :: ('a \Rightarrow 'a) \text{ list.}$ 

```

$$\begin{aligned}
& \text{length } P' = \text{length } gP' \wedge \text{length } Q' = \text{length } gQ' \wedge \\
& \text{set } gP' \subseteq M \wedge \text{set } gQ' \subseteq M \wedge \\
& \text{pairwise-disjoint } (P' @ Q') \wedge \\
& \text{pairwise-disjoint } (\text{map2 } (\lambda h C. h ' C) gP' P') \wedge \\
& \text{pairwise-disjoint } (\text{map2 } (\lambda h C. h ' C) gQ' Q') \wedge \\
& Y = (\bigcup_{i < \text{length } P'. P' ! i} \cup (\bigcup_{j < \text{length } Q'. Q' ! j}) \wedge \\
& Y = (\bigcup_{i < \text{length } P'. gP' ! i ' (P' ! i)} \wedge \\
& Y = (\bigcup_{j < \text{length } Q'. gQ' ! j ' (Q' ! j)}
\end{aligned}$$

proof –

let $?CP = \text{transfer-pairs } A \ e \ P \ gP \ B \ t$
let $?CQ = \text{transfer-pairs } A \ e \ Q \ gQ \ B \ t$
have $\text{finite-CP}: \text{finite } ?CP$
by *simp*
have $\text{finite-CQ}: \text{finite } ?CQ$
by *simp*
obtain pairsP **where** $\text{setP}: \text{set pairsP} = ?CP$ **and** $\text{distinctP}: \text{distinct pairsP}$
using $\text{finite-distinct-list}[OF \ \text{finite-CP}]$ **by** *blast*
obtain pairsQ **where** $\text{setQ}: \text{set pairsQ} = ?CQ$ **and** $\text{distinctQ}: \text{distinct pairsQ}$
using $\text{finite-distinct-list}[OF \ \text{finite-CQ}]$ **by** *blast*
let $?P' = \text{map fst pairsP}$
let $?Q' = \text{map fst pairsQ}$
let $?gP' = \text{map snd pairsP}$
let $?gQ' = \text{map snd pairsQ}$

have $P\text{-disj}: \text{pairwise-disjoint } P$
by $(\text{rule pairwise-disjoint-appendD1}[OF \ \text{source-disj}])$
have $Q\text{-disj}: \text{pairwise-disjoint } Q$
by $(\text{rule pairwise-disjoint-appendD2}[OF \ \text{source-disj}])$
have $PQ\text{-cross}: \bigwedge i \ j. \llbracket i < \text{length } P; j < \text{length } Q \rrbracket \implies P ! i \cap Q ! j = \{\}$
by $(\text{rule pairwise-disjoint-append-crossD}[OF \ \text{source-disj}])$

have $P'\text{-disj}: \text{pairwise-disjoint } ?P'$
proof $(\text{rule pairwise-disjoint-map-fstI}[OF \ \text{distinctP}])$
fix $x \ y$
assume $x \in \text{set pairsP} \ y \in \text{set pairsP} \ x \neq y$
thus $\text{fst } x \cap \text{fst } y = \{\}$
using $\text{setP transfer-source-disjoint-same}[OF \ A\text{-disj } P\text{-disj } B\text{-disj}]$ **by** *blast*

qed

have $Q'\text{-disj}: \text{pairwise-disjoint } ?Q'$
by $(\text{metis } A\text{-disj } B\text{-disj } Q\text{-disj } \text{distinctQ } \text{pairwise-disjoint-map-fstI } \text{setQ } \text{transfer-source-disjoint-same})$

have $\text{cross-disj}: \bigwedge x \ y. \llbracket x \in \text{set } ?P'; y \in \text{set } ?Q' \rrbracket \implies x \cap y = \{\}$

proof –

fix $x \ y$
assume $x: x \in \text{set } ?P' \ \text{and} \ y: y \in \text{set } ?Q'$
then obtain $px \ qy$ **where** $px: px \in \text{set pairsP} \ \text{and} \ qy: qy \in \text{set pairsQ}$
and $x\text{-eq}: x = \text{fst } px \ \text{and} \ y\text{-eq}: y = \text{fst } qy$
by *auto*
show $x \cap y = \{\}$

```

    using transfer-source-disjoint-cross[OF A-disj PQ-cross B-disj]
      setP setQ px qy x-eq y-eq by blast
  qed
  have source-disj': pairwise-disjoint (?P' @ ?Q')
    by (rule pairwise-disjoint-appendI[OF P'-disj Q'-disj cross-disj])

  have imageP-disj': pairwise-disjoint (map2 (λh C. h ' C) ?gP' ?P')
  proof (rule pairwise-disjoint-pair-imagesI[OF distinctP])
    fix x y
    assume x ∈ set pairsP y ∈ set pairsP x ≠ y
    thus snd x ' fst x ∩ snd y ' fst y = {}
      using setP transfer-image-disjoint-same[
        OF len-AB A-disj B-disj imageP-disj lenP gP-inj - t-left]
        e-left by blast
  qed
  have imageQ-disj': pairwise-disjoint (map2 (λh C. h ' C) ?gQ' ?Q')
  proof (rule pairwise-disjoint-pair-imagesI[OF distinctQ])
    fix x y
    assume x ∈ set pairsQ y ∈ set pairsQ x ≠ y
    thus snd x ' fst x ∩ snd y ' fst y = {}
      using setQ transfer-image-disjoint-same[
        OF len-AB A-disj B-disj imageQ-disj lenQ gQ-inj - t-left]
        e-left by blast
  qed

  have P-sub:  $\bigwedge i. i < \text{length } P \implies P ! i \subseteq X$ 
    using source-cover by blast
  have Q-sub:  $\bigwedge j. j < \text{length } Q \implies Q ! j \subseteq X$ 
    using source-cover by blast

  have source-cover':  $Y = (\bigcup i < \text{length } ?P'. ?P' ! i) \cup (\bigcup j < \text{length } ?Q'. ?Q' ! j)$ 
  proof -
    have  $Y = (\bigcup p \in ?CP. \text{fst } p) \cup (\bigcup q \in ?CQ. \text{fst } q)$ 
      by (rule transfer-source-cover[OF A-cover B-cover - len-AB source-cover
        imageP-cover imageQ-cover])
      (use e-left in auto)
    also have ... =  $(\bigcup i < \text{length } ?P'. ?P' ! i) \cup (\bigcup j < \text{length } ?Q'. ?Q' ! j)$ 
      using setP setQ indexed-union-map-fst[of pairsP] indexed-union-map-fst[of
        pairsQ]
      by simp
    finally show ?thesis .
  qed
  have imageP-cover':  $Y = (\bigcup i < \text{length } ?P'. ?gP' ! i ' (?P' ! i))$ 
  proof -
    have  $Y = (\bigcup p \in ?CP. \text{snd } p ' \text{fst } p)$ 
      by (rule transfer-image-cover[OF A-cover B-cover len-AB e-left t-left P-sub
        imageP-cover])
    also have ... =  $(\bigcup i < \text{length } ?P'. ?gP' ! i ' (?P' ! i))$ 
      using setP indexed-image-union-pairs[of pairsP] by simp
  
```

```

    finally show ?thesis .
  qed
  have imageQ-cover':  $Y = (\bigcup_{j < \text{length } ?Q'. ?gQ' ! j ' (?Q' ! j)}$ )
  proof -
    have  $Y = (\bigcup_{q \in ?CQ. \text{snd } q ' \text{fst } q}$ )
      by (rule transfer-image-cover[OF A-cover B-cover len-AB e-left t-left Q-sub
imageQ-cover])
    also have ... =  $(\bigcup_{j < \text{length } ?Q'. ?gQ' ! j ' (?Q' ! j)}$ )
      using setQ indexed-image-union-pairs[of pairsQ] by simp
    finally show ?thesis .
  qed

  have mapsP':  $\text{set } ?gP' \subseteq M$ 
    using setP mapsP unfolding transfer-pairs-def by auto
  have mapsQ':  $\text{set } ?gQ' \subseteq M$ 
    using setQ mapsQ unfolding transfer-pairs-def by auto

  show ?thesis
  proof (intro exI conjI)
    show  $\text{length } ?P' = \text{length } ?gP'$ 
      by simp
    show  $\text{length } ?Q' = \text{length } ?gQ'$ 
      by simp
    show  $\text{set } ?gP' \subseteq M$ 
      by (rule mapsP')
    show  $\text{set } ?gQ' \subseteq M$ 
      by (rule mapsQ')
    show pairwise-disjoint (?P' @ ?Q')
      by (rule source-disj')
    show pairwise-disjoint (map2 ( $\lambda h C. h ' C$ ) ?gP' ?P')
      by (rule imageP-disj')
    show pairwise-disjoint (map2 ( $\lambda h C. h ' C$ ) ?gQ' ?Q')
      by (rule imageQ-disj')
    show  $Y = (\bigcup_{i < \text{length } ?P'. ?P' ! i} \cup (\bigcup_{j < \text{length } ?Q'. ?Q' ! j}$ )
      by (rule source-cover')
    show  $Y = (\bigcup_{i < \text{length } ?P'. ?gP' ! i ' (?P' ! i)}$ )
      by (rule imageP-cover')
    show  $Y = (\bigcup_{j < \text{length } ?Q'. ?gQ' ! j ' (?Q' ! j)}$ )
      by (rule imageQ-cover')
  qed
  qed
end

```

```

theory Paradoxical-Decomposition
  imports BT-Prelim
begin

```

4 Group actions on a set

An action of a (carrier) group G on a set X is a function act respecting the unit and multiplication on X . We do not fix the group structure here; rather, we record the algebraic laws that an action must satisfy. Concrete instances will be given in later theories (the action of F_2 on itself by left translation; the action of $SO(3)$ on S^2).

```

locale group-action =
  fixes carrier :: 'g set
    and unit :: 'g
    and mult :: 'g  $\Rightarrow$  'g  $\Rightarrow$  'g (infixl  $\langle \cdot \rangle$  70)
    and act :: 'g  $\Rightarrow$  'a  $\Rightarrow$  'a
    and ground :: 'a set
  assumes unit-carrier [simp]: unit  $\in$  carrier
    and mult-closed [intro]:  $\llbracket g \in$  carrier;  $h \in$  carrier  $\rrbracket \Longrightarrow g \cdot h \in$  carrier
    and act-closed [intro]:  $\llbracket g \in$  carrier;  $x \in$  ground  $\rrbracket \Longrightarrow act\ g\ x \in$  ground
    and act-unit [simp]:  $x \in$  ground  $\Longrightarrow act\ unit\ x = x$ 
    and act-mult:  $\llbracket g \in$  carrier;  $h \in$  carrier;  $x \in$  ground  $\rrbracket$ 
       $\Longrightarrow act\ (g \cdot h)\ x = act\ g\ (act\ h\ x)$ 

```

```

context group-action
begin

```

```

definition orbit :: 'a  $\Rightarrow$  'a set where
  orbit  $x = \{y. \exists g \in$  carrier.  $y = act\ g\ x\}$ 

```

```

definition image-set :: 'g  $\Rightarrow$  'a set  $\Rightarrow$  'a set where
  image-set  $g\ A = act\ g\ `A$ 

```

```

lemma image-set-unit:
  assumes  $A \subseteq$  ground
  shows image-set unit  $A = A$ 
  using assms unfolding image-set-def by force

```

```

lemma orbit-self:
  assumes  $x \in$  ground
  shows  $x \in$  orbit  $x$ 
  using assms by (simp add: orbit-def) (metis act-unit unit-carrier)

```

```

end

```

5 Paradoxical decomposition

A subset X of the underlying set of a group action is *paradoxical* when it admits two finite collections of pairwise disjoint pieces, all contained in X , such that each collection can be translated (by group elements) so that the resulting *disjoint* union is the whole of X .

This is the standard Banach-Tarski definition. We will instantiate it twice: once for F_2 acting on itself, and once for the rotation group acting on S^2 (and ultimately the ball).

context *group-action*
begin

definition *paradoxical* :: 'a set \Rightarrow bool **where**

paradoxical $X \longleftrightarrow$
 $(\exists P Q :: 'a \text{ set list. } \exists gP gQ :: 'g \text{ list.}$
 $\text{length } P = \text{length } gP \wedge \text{length } Q = \text{length } gQ \wedge$
 $\text{set } gP \subseteq \text{carrier} \wedge \text{set } gQ \subseteq \text{carrier} \wedge$
 $\text{pairwise-disjoint } (P @ Q) \wedge$
 $\text{pairwise-disjoint } (\text{map2 image-set } gP P) \wedge$
 $\text{pairwise-disjoint } (\text{map2 image-set } gQ Q) \wedge$
 $(\bigcup_{i < \text{length } P} P ! i) \cup (\bigcup_{i < \text{length } Q} Q ! i) \subseteq X \wedge$
 $X = (\bigcup_{i < \text{length } P} \text{image-set } (gP ! i) (P ! i)) \wedge$
 $X = (\bigcup_{i < \text{length } Q} \text{image-set } (gQ ! i) (Q ! i))$

A specialised constructor for the most common case: each of P and Q has exactly two pieces. This matches the F_2 decomposition (split into the a / a^{-1} classes and the b / b^{-1} classes).

lemma *paradoxical-two-two*:

assumes $p1 \in \text{carrier } p2 \in \text{carrier } q1 \in \text{carrier } q2 \in \text{carrier}$
and *disj-P12*: $P1 \cap P2 = \{\}$
and *disj-Q12*: $Q1 \cap Q2 = \{\}$
and *disj-PQ*: $(P1 \cup P2) \cap (Q1 \cup Q2) = \{\}$
and *sub*: $P1 \cup P2 \cup Q1 \cup Q2 \subseteq X$
and *disj-imgP*: $\text{image-set } p1 P1 \cap \text{image-set } p2 P2 = \{\}$
and *disj-imgQ*: $\text{image-set } q1 Q1 \cap \text{image-set } q2 Q2 = \{\}$
and *cover-P*: $X = \text{image-set } p1 P1 \cup \text{image-set } p2 P2$
and *cover-Q*: $X = \text{image-set } q1 Q1 \cup \text{image-set } q2 Q2$

shows *paradoxical* X

proof –

let $?P = [P1, P2]$ **and** $?Q = [Q1, Q2]$
let $?gP = [p1, p2]$ **and** $?gQ = [q1, q2]$
have *lens*: $\text{length } ?P = \text{length } ?gP \text{ length } ?Q = \text{length } ?gQ$ **by** *simp-all*
have *closed*: $\text{set } ?gP \subseteq \text{carrier } \text{set } ?gQ \subseteq \text{carrier}$
using *assms(1-4)* **by** *auto*
have *disj-all*: $\text{pairwise-disjoint } (?P @ ?Q)$
using *disj-P12 disj-Q12 disj-PQ*
by (*auto simp: pairwise-disjoint-def nth-Cons' nth-append Int-commute Int-Un-distrib*)
have *map2-P*: $\text{map2 image-set } ?gP ?P = [\text{image-set } p1 P1, \text{image-set } p2 P2]$ **by**
simp
have *map2-Q*: $\text{map2 image-set } ?gQ ?Q = [\text{image-set } q1 Q1, \text{image-set } q2 Q2]$
by *simp*
have *disj-map-P*: $\text{pairwise-disjoint } (\text{map2 image-set } ?gP ?P)$
using *disj-imgP unfolding map2-P*
by (*auto simp: pairwise-disjoint-def nth-Cons' Int-commute*)

```

have disj-map-Q: pairwise-disjoint (map2 image-set ?gQ ?Q)
  using disj-imgQ unfolding map2-Q
  by (auto simp: pairwise-disjoint-def nth-Cons' Int-commute)
have sub-un:  $(\bigcup_{i < \text{length } ?P} ?P ! i) \cup (\bigcup_{i < \text{length } ?Q} ?Q ! i) \subseteq X$ 
  using sub by (auto simp: lessThan-Suc)
have cov-P':  $X = (\bigcup_{i < \text{length } ?P} \text{image-set } (?gP ! i) (?P ! i))$ 
  using cover-P by (auto simp: lessThan-Suc)
have cov-Q':  $X = (\bigcup_{i < \text{length } ?Q} \text{image-set } (?gQ ! i) (?Q ! i))$ 
  using cover-Q by (auto simp: lessThan-Suc)
show ?thesis
  unfolding paradoxical-def
  using lens closed disj-all disj-map-P disj-map-Q sub-un cov-P' cov-Q'
  by blast
qed

```

With the finite-list convention used above, the empty set is paradoxical: take both families of pieces to be empty. The two translated indexed unions are then both the empty union, hence both equal to $\{\}$. This degenerate case is harmless in the later non-empty geometric applications, but recording it keeps the abstract definition honest about empty indexed unions.

```

lemma paradoxical-empty: paradoxical  $\{\}$ 
  unfolding paradoxical-def
  by (intro exI [where  $x = []$ ])
  (auto simp: pairwise-disjoint-def image-set-def)

```

end

6 Equidecomposability

Two subsets X and Y are *equidecomposable* under a group action when they admit congruent partitions: the i -th piece of X maps onto the i -th piece of Y via some group element.

```

context group-action
begin

```

```

definition equidecomposable :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  bool where
  equidecomposable  $X Y \iff$ 
    ( $\exists P Q :: 'a \text{ set list. } \exists gs :: 'g \text{ list.}$ 
       $\text{length } P = \text{length } Q \wedge \text{length } P = \text{length } gs \wedge$ 
       $\text{set } gs \subseteq \text{carrier} \wedge$ 
       $\text{pairwise-disjoint } P \wedge \text{pairwise-disjoint } Q \wedge$ 
       $X = (\bigcup_{i < \text{length } P} P ! i) \wedge$ 
       $Y = (\bigcup_{i < \text{length } Q} Q ! i) \wedge$ 
       $(\forall i < \text{length } P. Q ! i = \text{image-set } (gs ! i) (P ! i))$ )

```

The main development uses this relation only as background notation: the sphere and ball arguments below work directly with explicit finite lists of

pieces and transformations, so the needed partition data is visible at each transfer step.

end

end

theory *Free-Action-Paradox*
imports *Paradoxical-Decomposition*
begin

7 Free actions

An action is *free* on a set X if non-identity group elements have no fixed points in X : $g \cdot x = x$ forces $g = \text{unit}$.

context *group-action*
begin

definition *free-on* :: 'a set \Rightarrow bool **where**
free-on $X \longleftrightarrow$
 $(\forall g \in \text{carrier}. \forall x \in X. \text{act } g \ x = x \longrightarrow g = \text{unit})$

end

8 Choice of orbit representatives

Given a free action of G on X , the orbits partition X , and for any subset of choice representatives $M \subseteq X$ (one per orbit), every $x \in X$ can be written uniquely as $\text{act } g \ m$ for some $g \in \text{carrier}$ and $m \in M$.

This is where the axiom of choice enters: we pick one element from each orbit.

context *group-action*
begin

definition *orbit-eq* :: 'a \Rightarrow 'a \Rightarrow bool **where**
orbit-eq $x \ y \longleftrightarrow (\exists g \in \text{carrier}. y = \text{act } g \ x)$

lemma *orbit-eq-refl*: $x \in \text{ground} \Longrightarrow \text{orbit-eq } x \ x$
unfolding *orbit-eq-def* **using** *act-unit* *unit-carrier* **by** *metis*

When the action is free on X , the orbit relation is symmetric on X , provided every element has a group inverse in the carrier. We do not require *group-level* inverses here, so we state symmetry conditionally.

end

9 Transport of paradoxical decompositions along free actions

This is the abstract Banach-Tarski reduction: suppose G acts freely on X , and G itself admits a paradoxical decomposition under the regular action (left multiplication on itself). Choosing one representative per G -orbit in X via the axiom of choice transports the paradox: each piece $P_i \subseteq G$ in the decomposition lifts to the union $\{act\ g\ m \mid g \in P_i, m \in M\}$, where M is the chosen set of representatives.

The later Hausdorff theory carries out this lift explicitly for $S^2 \setminus D$: it defines orbit representatives, proves the lifted pieces are disjoint, and proves the two translated covers directly. The predicate below only packages the freeness and closure hypotheses that such an argument needs.

context *group-action*
begin

A free action is suitable for orbit-by-orbit transport when the set is contained in the ground space, the action is free on it, and the set is closed under the action of every carrier element.

definition *transports-paradox* :: 'a set \Rightarrow bool **where**
transports-paradox $X \longleftrightarrow$
 $(X \subseteq ground \wedge$
free-on $X \wedge$
 $(\forall x \in X. \forall g \in carrier. act\ g\ x \in X))$

Under the assumptions captured by *transports-paradox*, a paradoxical decomposition of the carrier transports to a paradoxical decomposition of X . The proof is the standard “orbit-by-orbit” argument and uses the axiom of choice to select one representative per orbit.

end

end

theory *Free-Group-F2*
imports
BT-Prelim
Free-Groups.FreeGroups
begin

10 Two-element generator type

We pick a concrete two-element type for the generators of F_2 .

datatype *gen2* = $A \mid B$

abbreviation $Gen2$:: *gen2 set* **where**

$Gen2 \equiv \{A, B\}$

lemma $Gen2$ -finite [*simp*]: *finite* $Gen2$

by *simp*

lemma $Gen2$ -card [*simp*]: $card\ Gen2 = 2$

by *simp*

lemma $Gen2$ -UNIV: $Gen2 = (UNIV :: gen2\ set)$

using *gen2.exhaust* **by** *auto*

11 The free group F_2 and its carrier

abbreviation $F2$:: (*bool* \times *gen2*) *list monoid* **where**

$F2 \equiv \mathcal{F}\ Gen2$

The carrier of F_2 consists of cancelled words over the alphabet $UNIV \times Gen2 = UNIV \times UNIV$.

lemma $F2$ -is-group: *group* $F2$

by (*rule free-group-is-group*)

lemma $F2$ -carrier-iff:

$w \in carrier\ F2 \longleftrightarrow w \in lists\ (UNIV \times Gen2) \wedge canceled\ w$

by (*auto simp: free-group-def*)

lemma $F2$ -carrier-alt:

$carrier\ F2 = \{w.\ canceled\ w\}$

proof –

have $lists\ (UNIV \times Gen2) = (UNIV :: (bool \times gen2)\ list\ set)$

using $Gen2$ -UNIV **by** *auto*

thus *?thesis* **by** (*auto simp: free-group-def*)

qed

lemma *in-F2-canceled* [*dest*]:

assumes $w \in carrier\ F2$

shows *canceled* w

using *assms* **by** (*simp add: F2-carrier-alt*)

lemma *canceled-in-F2* [*intro*]:

assumes *canceled* w

shows $w \in carrier\ F2$

using *assms* **by** (*simp add: F2-carrier-alt*)

lemma *canceled-iff-no-adjacent-canceling*:

$canceled\ w \longleftrightarrow (\forall i.\ Suc\ i < length\ w \longrightarrow \neg\ canceling\ (w\ !\ i)\ (w\ !\ Suc\ i))$

proof

assume c : *canceled* w

```

show  $\forall i. \text{Suc } i < \text{length } w \longrightarrow \neg \text{canceling } (w ! i) (w ! \text{Suc } i)$ 
proof (intro allI impI notI)
  fix  $i$ 
  assume  $\text{len}: \text{Suc } i < \text{length } w$  and  $\text{can}: \text{canceling } (w ! i) (w ! \text{Suc } i)$ 
  have  $\text{cancels-to-1-at } i \ w$  (cancel-at  $i \ w$ )
    using  $\text{len can}$  by (auto simp: cancels-to-1-at-def)
  hence  $\text{cancels-to-1 } w$  (cancel-at  $i \ w$ )
    unfolding  $\text{cancels-to-1-def}$  by blast
  hence  $\text{Domainp } \text{cancels-to-1 } w$ 
    by auto
  with  $c$  show  $\text{False}$ 
    by (simp add: canceled-def)
qed
next
assume  $\text{no-adj}: \forall i. \text{Suc } i < \text{length } w \longrightarrow \neg \text{canceling } (w ! i) (w ! \text{Suc } i)$ 
show  $\text{canceled } w$ 
proof (simp add: canceled-def, intro notI)
  assume  $\text{Domainp } \text{cancels-to-1 } w$ 
  then obtain  $w'$  where  $\text{cancels-to-1 } w \ w'$ 
    by auto
  then obtain  $i$  where  $\text{cancels-to-1-at } i \ w \ w'$ 
    unfolding  $\text{cancels-to-1-def}$  by auto
  hence  $\text{Suc } i < \text{length } w$  and  $\text{canceling } (w ! i) (w ! \text{Suc } i)$ 
    by (auto simp: cancels-to-1-at-def)
  with  $\text{no-adj}$  show  $\text{False}$ 
    by blast
qed
qed

lemma  $\text{canceled-Cons-iff}$ :
   $\text{canceled } (p \# w) \longleftrightarrow \text{canceled } w \wedge (w = [] \vee \neg \text{canceling } p (\text{hd } w))$ 
  unfolding  $\text{canceled-iff-no-adjacent-canceling}$ 
  by (cases  $w$ )
    (auto simp: nth-Cons split: nat.splits)

lemma  $F2\text{-ConsD}$ :
  assumes  $p \# w \in \text{carrier } F2$ 
  shows  $w \in \text{carrier } F2$  and  $w = [] \vee \neg \text{canceling } p (\text{hd } w)$ 
  using  $\text{assms}$  by (simp-all add:  $F2\text{-carrier-alt canceled-Cons-iff}$ )

lemma  $F2\text{-ConsI}$ :
  assumes  $w \in \text{carrier } F2$  and  $w = [] \vee \neg \text{canceling } p (\text{hd } w)$ 
  shows  $p \# w \in \text{carrier } F2$ 
  using  $\text{assms}$  by (simp add:  $F2\text{-carrier-alt canceled-Cons-iff}$ )

lemma  $F2\text{-one}$  [simp]:  $\mathbf{1}_{F2} = []$ 
  by (simp add: free-group-def)

lemma  $F2\text{-mult}$ :  $x \otimes_{F2} y = \text{normalize } (x @ y)$ 

```

by (*simp add: free-group-def*)

12 The four “starts-with” classes of F_2

A non-empty reduced word in F_2 begins with one of four letters: a , a^{-1} , b , or b^{-1} . The corresponding “starts-with” classes partition $F_2 \setminus \{1\}$.

definition *starts-with* :: $bool \Rightarrow gen2 \Rightarrow (bool \times gen2)$ list set **where**
starts-with $b\ x = \{w \in carrier\ F2. w \neq [] \wedge hd\ w = (b, x)\}$

abbreviation *S-a* :: $(bool \times gen2)$ list set **where** *S-a* $\equiv starts-with\ False\ A$

abbreviation *S-aI* :: $(bool \times gen2)$ list set **where** *S-aI* $\equiv starts-with\ True\ A$

abbreviation *S-b* :: $(bool \times gen2)$ list set **where** *S-b* $\equiv starts-with\ False\ B$

abbreviation *S-bI* :: $(bool \times gen2)$ list set **where** *S-bI* $\equiv starts-with\ True\ B$

lemma *starts-with-subset*: $starts-with\ b\ x \subseteq carrier\ F2$

by (*auto simp: starts-with-def*)

lemma *starts-with-disjoint*:

assumes $(b1, x1) \neq (b2, x2)$

shows $starts-with\ b1\ x1 \cap starts-with\ b2\ x2 = \{\}$

using *assms* **by** (*auto simp: starts-with-def*)

lemma *starts-with-pairwise-disjoint*:

shows $S-a \cap S-aI = \{\}$ $S-a \cap S-b = \{\}$ $S-a \cap S-bI = \{\}$

$S-aI \cap S-b = \{\}$ $S-aI \cap S-bI = \{\}$ $S-b \cap S-bI = \{\}$

$[] \notin S-a$ $[] \notin S-aI$ $[] \notin S-b$ $[] \notin S-bI$

by (*auto simp: starts-with-def*)

The four classes together with the singleton $\{1\}$ partition the carrier of F_2 .

The proof is a case analysis on the head of a reduced word.

lemma *F2-decomposition*:

$carrier\ F2 = \{[]\} \cup S-a \cup S-aI \cup S-b \cup S-bI$

proof

show $\{[]\} \cup S-a \cup S-aI \cup S-b \cup S-bI \subseteq carrier\ F2$

using *starts-with-subset*

by (*auto simp: F2-carrier-alt empty-canceled*)

next

show $carrier\ F2 \subseteq \{[]\} \cup S-a \cup S-aI \cup S-b \cup S-bI$

proof

fix w **assume** $w: w \in carrier\ F2$

show $w \in \{[]\} \cup S-a \cup S-aI \cup S-b \cup S-bI$

proof (*cases w*)

case *Nil* **thus** *?thesis* **by** *simp*

next

case (*Cons a u*)

obtain $b\ x$ **where** *head*: $a = (b, x)$ **by** (*cases a*) *auto*

from w **have** $x \in Gen2$ **using** *Cons head*

by (*auto simp: F2-carrier-iff*)

hence $x = A \vee x = B$ **by** *auto*
with *Cons head w* **show** *?thesis*
unfolding *starts-with-def* **by** *auto*
qed
qed
qed

13 The two distinguished generators a and b

abbreviation $a\text{-elt} :: (\text{bool} \times \text{gen2}) \text{ list}$ **where** $a\text{-elt} \equiv [(False, A)]$
abbreviation $aI\text{-elt} :: (\text{bool} \times \text{gen2}) \text{ list}$ **where** $aI\text{-elt} \equiv [(True, A)]$
abbreviation $b\text{-elt} :: (\text{bool} \times \text{gen2}) \text{ list}$ **where** $b\text{-elt} \equiv [(False, B)]$
abbreviation $bI\text{-elt} :: (\text{bool} \times \text{gen2}) \text{ list}$ **where** $bI\text{-elt} \equiv [(True, B)]$

lemma *single-letter-canceled*:
canceled [(b, x)]
by *simp*

lemma *a-elt-in-F2* [*simp*]: $a\text{-elt} \in \text{carrier } F2$
by (*intro canceled-in-F2 single-letter-canceled*)

lemma *aI-elt-in-F2* [*simp*]: $aI\text{-elt} \in \text{carrier } F2$
by (*intro canceled-in-F2 single-letter-canceled*)

lemma *b-elt-in-F2* [*simp*]: $b\text{-elt} \in \text{carrier } F2$
by (*intro canceled-in-F2 single-letter-canceled*)

lemma *bI-elt-in-F2* [*simp*]: $bI\text{-elt} \in \text{carrier } F2$
by (*intro canceled-in-F2 single-letter-canceled*)

lemma *inv-a-eq-aI*: $\text{inv}_{F2} a\text{-elt} = aI\text{-elt}$
by (*simp add: inv-fg-def inv1-def*)

lemma *inv-b-eq-bI*: $\text{inv}_{F2} b\text{-elt} = bI\text{-elt}$
by (*simp add: inv-fg-def inv1-def*)

lemma *inv-aI-eq-a*: $\text{inv}_{F2} aI\text{-elt} = a\text{-elt}$
by (*simp add: inv-fg-def inv1-def*)

lemma *inv-bI-eq-b*: $\text{inv}_{F2} bI\text{-elt} = b\text{-elt}$
by (*simp add: inv-fg-def inv1-def*)

end

theory *F2-Paradox*
imports
Free-Group-F2
Paradoxical-Decomposition

begin

14 The action of F_2 on itself by left multiplication

We interpret *group-action* with carrier *carrier F2* and the action being left multiplication.

lemma *F2-one-in-carrier [simp]: [] ∈ carrier F2*

by (*metis F2-is-group F2-one group.is-monoid monoid.one-closed*)

interpretation *F2-act:*

group-action carrier F2 [] (⊗_{F2}) (⊗_{F2}) carrier F2

proof *unfold-locales*

show *[] ∈ carrier F2 by simp*

next

fix *g h assume g ∈ carrier F2 h ∈ carrier F2*

thus *g ⊗_{F2} h ∈ carrier F2*

by (*meson F2-is-group group.is-monoid monoid.m-closed*)

next

fix *x assume xin: x ∈ carrier F2*

have *[] ⊗_{F2} x = 1_{F2} ⊗_{F2} x by simp*

also have *... = x using xin F2-is-group group.is-monoid monoid.l-one by metis*

finally show *[] ⊗_{F2} x = x .*

next

fix *g h x*

assume *g ∈ carrier F2 h ∈ carrier F2 x ∈ carrier F2*

thus *(g ⊗_{F2} h) ⊗_{F2} x = g ⊗_{F2} (h ⊗_{F2} x)*

by (*meson F2-is-group group.is-monoid monoid.m-assoc*)

qed

15 Behaviour of left multiplication by a

When we left-multiply a word that begins with a^{-1} by a , the leading pair cancels, and the result is exactly the tail of the original word.

lemma *a-mult-starts-with-aI:*

assumes *w ∈ S-aI*

shows *a-elt ⊗_{F2} w = tl w*

proof –

from *assms have w-in: w ∈ carrier F2*

and *w-ne: w ≠ []*

and *w-hd: hd w = (True, A)*

by (*auto simp: starts-with-def*)

from *w-ne w-hd obtain u where w-eq: w = (True, A) # u*

by (*cases w*) *auto*

from *w-in have w-canc: canceled w by auto*

with *w-eq have u-canc: canceled u using cons-canceled by metis*

have *can*: *canceling* (*False*, *A*) (*True*, *A*)
by (*simp add: canceling-def*)
have *prep*: *a-elt* @ *w* = [(*False*, *A*), (*True*, *A*)] @ *u*
using *w-eq* **by** *simp*
have *step*: *cancels-to-1-at 0* [(*False*, *A*), (*True*, *A*)] @ *u* *u*
using *can* **by** (*auto simp: cancels-to-1-at-def cancel-at-def*)
hence *cancels-to-1* (*a-elt* @ *w*) *u* **using** *prep*
by (*auto simp: cancels-to-1-def*)
hence *cancels-to* (*a-elt* @ *w*) *u*
by (*auto simp: cancels-to-def*)
hence *norm*: *normalize* (*a-elt* @ *w*) = *u*
using *u-canc* **by** (*rule normalize-discover[rotated]*)
show *?thesis* **using** *norm w-eq*
by (*simp add: F2-mult*)
qed

lemma *b-mult-starts-with-bI*:

assumes *w* ∈ *S-bI*

shows *b-elt* ⊗_{*F*2} *w* = *tl w*

proof –

from *assms* **have** *w-in*: *w* ∈ *carrier F2*

and *w-ne*: *w* ≠ []

and *w-hd*: *hd w* = (*True*, *B*)

by (*auto simp: starts-with-def*)

from *w-ne w-hd* **obtain** *u* **where** *w-eq*: *w* = (*True*, *B*) # *u*

by (*cases w*) *auto*

from *w-in* **have** *w-canc*: *canceled w* **by** *auto*

with *w-eq* **have** *u-canc*: *canceled u* **using** *cons-canceled* **by** *metis*

have *can*: *canceling* (*False*, *B*) (*True*, *B*)

by (*simp add: canceling-def*)

have *prep*: *b-elt* @ *w* = [(*False*, *B*), (*True*, *B*)] @ *u*

using *w-eq* **by** *simp*

have *step*: *cancels-to-1-at 0* [(*False*, *B*), (*True*, *B*)] @ *u* *u*

using *can* **by** (*auto simp: cancels-to-1-at-def cancel-at-def*)

hence *cancels-to-1* (*b-elt* @ *w*) *u* **using** *prep*

by (*auto simp: cancels-to-1-def*)

hence *cancels-to* (*b-elt* @ *w*) *u*

by (*auto simp: cancels-to-def*)

hence *norm*: *normalize* (*b-elt* @ *w*) = *u*

using *u-canc* **by** (*rule normalize-discover[rotated]*)

show *?thesis* **using** *norm w-eq*

by (*simp add: F2-mult*)

qed

16 Image computations

Translating S_a^{-1} on the left by a yields exactly the carrier of F_2 minus S_a .

lemma *image-a-S-aI*:
shows (\otimes_{F2}) *a-elt* ‘ *S-aI* = *carrier F2* – *S-a*
proof
show (\otimes_{F2}) *a-elt* ‘ *S-aI* \subseteq *carrier F2* – *S-a*
proof
fix *v* **assume** $v \in (\otimes_{F2})$ *a-elt* ‘ *S-aI*
then obtain *w* **where** $w \in S-aI$ **and** *v-eq*: $v = a-elt \otimes_{F2} w$ **by** *auto*
from *w* **have** *w-in*: $w \in carrier F2$ **and** *w-ne*: $w \neq []$
and *w-hd*: $hd w = (True, A)$
by (*auto simp: starts-with-def*)
from *w-ne w-hd* **obtain** *u* **where** *w-eq*: $w = (True, A) \# u$
by (*cases w*) *auto*
from *a-mult-starts-with-aI*[*OF w*] **have** *v-tl*: $v = tl w$ **using** *v-eq* **by** *simp*
hence *v-u*: $v = u$ **using** *w-eq* **by** *simp*
from *w-in* **have** *canceled w* **by** *auto*
with *w-eq* **have** *u-canc*: *canceled u* **using** *cons-canceled* **by** *metis*
have *v-canc*: *canceled v* **using** *v-u u-canc* **by** *simp*
hence *v-in*: $v \in carrier F2$ **by** (*rule canceled-in-F2*)
have *not-starts-a*: $v \notin S-a$
proof (*cases u*)
case *Nil* **with** *v-u* **show** *?thesis* **by** (*auto simp: starts-with-def*)
next
case (*Cons p ps*)
have *w-split*: $w = [(True, A), p] @ ps$ **using** *w-eq Cons* **by** *simp*
have *ncan*: $\neg canceled (True, A) p$
proof
assume *can-pair*: *canceled (True, A) p*
have *cancels-to-1-at 0 w ps*
using *w-split can-pair*
by (*auto simp: cancels-to-1-at-def cancel-at-def*)
hence *cancels-to-1 w ps*
by (*auto simp: cancels-to-1-def*)
hence $\neg canceled w$
by (*auto simp: canceled-def*)
with $\langle canceled w \rangle$ **show** *False* **by** *contradiction*
qed
hence $p \neq (False, A)$
by (*auto simp: canceling-def*)
hence *hd-neq*: $hd v \neq (False, A)$
using *Cons v-u* **by** *simp*
thus *?thesis* **using** *v-in* **by** (*auto simp: starts-with-def*)
qed
show $v \in carrier F2$ – *S-a* **using** *v-in not-starts-a* **by** *simp*
qed
next
show $carrier F2$ – *S-a* \subseteq (\otimes_{F2}) *a-elt* ‘ *S-aI*
proof
fix *v* **assume** $v \in carrier F2$ – *S-a*
hence *v-in*: $v \in carrier F2$ **and** *v-not-a*: $v \notin S-a$ **by** *auto*

```

let ?w = (True, A) # v
have v-canc: canceled v using v-in by auto
have ncan: v = [] ∨ ¬ canceling (True, A) (hd v)
proof (cases v)
  case Nil thus ?thesis by simp
next
  case (Cons p ps)
  from v-not-a Cons v-in have p ≠ (False, A)
  by (auto simp: starts-with-def)
  hence ¬ canceling (True, A) p
  by (cases p) (auto simp: canceling-def)
  with Cons show ?thesis by simp
qed
have w-canc: canceled ?w
proof (rule ccontr)
  assume ¬ canceled ?w
  then obtain w' where cancels-to-1 ?w w'
  by (auto simp: canceled-def)
  then obtain xs1 x1 x2 xs2
  where decomp: ?w = xs1 @ x1 # x2 # xs2
  and can12: canceling x1 x2
  by (rule cancels-to-1-unfold)
  show False
  proof (cases xs1)
    case Nil
    with decomp have x1-eq: x1 = (True, A) by simp
    from decomp Nil have v-eq: v = x2 # xs2 by simp
    from v-eq x1-eq can12 ncan show False by simp
  next
    case (Cons y ys)
    with decomp have v-decomp: v = ys @ x1 # x2 # xs2 by simp
    with can12 have ¬ canceled v
    unfolding canceled-def
    by (auto intro: cancels-to-1-fold)
    with v-canc show False by contradiction
  qed
  qed
  hence w-in: ?w ∈ carrier F2 by auto
  have w-starts: ?w ∈ S-aI using w-in by (auto simp: starts-with-def)
  have v-eq: v = a-elt ⊗F2 ?w
  using a-mult-starts-with-aI[OF w-starts] by simp
  show v ∈ (⊗F2) a-elt ' S-aI using v-eq w-starts by auto
  qed
qed

lemma image-b-S-bI:
  shows (⊗F2) b-elt ' S-bI = carrier F2 - S-b
proof
  show (⊗F2) b-elt ' S-bI ⊆ carrier F2 - S-b

```

proof
fix v **assume** $v \in (\otimes_{F2})$ $b\text{-elt}$ ‘ $S\text{-bI}$
then obtain w **where** $w \in S\text{-bI}$ **and** $v\text{-eq}: v = b\text{-elt} \otimes_{F2} w$ **by** *auto*
from w **have** $w\text{-in}: w \in \text{carrier } F2$ **and** $w\text{-ne}: w \neq []$
and $w\text{-hd}: \text{hd } w = (\text{True}, B)$
by (*auto simp: starts-with-def*)
from $w\text{-ne}$ $w\text{-hd}$ **obtain** u **where** $w\text{-eq}: w = (\text{True}, B) \# u$
by (*cases w*) *auto*
from $b\text{-mult-starts-with-bI}[OF w]$ **have** $v\text{-tl}: v = \text{tl } w$ **using** $v\text{-eq}$ **by** *simp*
hence $v\text{-u}: v = u$ **using** $w\text{-eq}$ **by** *simp*
from $w\text{-in}$ **have** *canceled* w **by** *auto*
with $w\text{-eq}$ **have** $u\text{-canc}: \text{canceled } u$ **using** *cons-canceled* **by** *metis*
have $v\text{-canc}: \text{canceled } v$ **using** $v\text{-u}$ $u\text{-canc}$ **by** *simp*
hence $v\text{-in}: v \in \text{carrier } F2$ **by** (*rule canceled-in-F2*)
have $\text{not-starts-b}: v \notin S\text{-b}$
proof (*cases u*)
case *Nil* **with** $v\text{-u}$ **show** *?thesis* **by** (*auto simp: starts-with-def*)
next
case (*Cons p ps*)
have $w\text{-split}: w = [(\text{True}, B), p] @ ps$ **using** $w\text{-eq}$ *Cons* **by** *simp*
have $\text{ncan}: \neg \text{canceling } (\text{True}, B) p$
proof
assume $\text{can-pair}: \text{canceling } (\text{True}, B) p$
have $\text{cancels-to-1-at } 0 w ps$
using $w\text{-split}$ *can-pair*
by (*auto simp: cancels-to-1-at-def cancel-at-def*)
hence $\text{cancels-to-1 } w ps$
by (*auto simp: cancels-to-1-def*)
hence $\neg \text{canceled } w$
by (*auto simp: canceled-def*)
with $\langle \text{canceled } w \rangle$ **show** *False* **by** *contradiction*
qed
hence $p \neq (\text{False}, B)$
by (*auto simp: canceling-def*)
hence $\text{hd-neq}: \text{hd } v \neq (\text{False}, B)$
using *Cons v-u* **by** *simp*
thus *?thesis* **using** $v\text{-in}$ **by** (*auto simp: starts-with-def*)
qed
show $v \in \text{carrier } F2 - S\text{-b}$ **using** $v\text{-in}$ not-starts-b **by** *simp*
qed
next
show $\text{carrier } F2 - S\text{-b} \subseteq (\otimes_{F2})$ $b\text{-elt}$ ‘ $S\text{-bI}$
proof
fix v **assume** $v \in \text{carrier } F2 - S\text{-b}$
hence $v\text{-in}: v \in \text{carrier } F2$ **and** $v\text{-not-b}: v \notin S\text{-b}$ **by** *auto*
let $?w = (\text{True}, B) \# v$
have $v\text{-canc}: \text{canceled } v$ **using** $v\text{-in}$ **by** *auto*
have $\text{ncan}: v = [] \vee \neg \text{canceling } (\text{True}, B) (\text{hd } v)$
proof (*cases v*)

```

    case Nil thus ?thesis by simp
next
case (Cons p ps)
from v-not-b Cons v-in have p ≠ (False, B)
  by (auto simp: starts-with-def)
hence ¬ canceling (True, B) p
  by (cases p) (auto simp: canceling-def)
with Cons show ?thesis by simp
qed
have w-canc: canceled ?w
proof (rule ccontr)
  assume ¬ canceled ?w
  then obtain w' where cancels-to-1 ?w w'
    by (auto simp: canceled-def)
  then obtain xs1 x1 x2 xs2
    where decomp: ?w = xs1 @ x1 # x2 # xs2
    and can12: canceling x1 x2
    by (rule cancels-to-1-unfold)
  show False
proof (cases xs1)
  case Nil
  with decomp have x1-eq: x1 = (True, B) by simp
  from decomp Nil have v-eq: v = x2 # xs2 by simp
  from v-eq x1-eq can12 ncan show False by simp
next
case (Cons y ys)
with decomp have v-decomp: v = ys @ x1 # x2 # xs2 by simp
with can12 have ¬ canceled v
  unfolding canceled-def
  by (auto intro: cancels-to-1-fold)
with v-canc show False by contradiction
qed
qed
hence w-in: ?w ∈ carrier F2 by auto
have w-starts: ?w ∈ S-bI using w-in by (auto simp: starts-with-def)
have v-eq: v = b-elt ⊗F2 ?w
  using b-mult-starts-with-bI[OF w-starts] by simp
show v ∈ (⊗F2) b-elt ' S-bI using v-eq w-starts by auto
qed
qed

```

17 The paradoxical decomposition of F_2

Putting everything together: S_a together with the a -translate of S_a^{-1} covers F_2 (disjointly), and symmetrically for b . Hence F_2 is paradoxical.

lemma *S-a-un-image-a-S-aI*: $S-a \cup (\otimes_{F_2}) a\text{-elt} ' S-aI = \text{carrier } F_2$
 using *image-a-S-aI starts-with-subset* by blast

lemma *S-b-un-image-b-S-bI*: $S-b \cup (\otimes_{F2}) \text{ b-elt } \text{' } S-bI = \text{carrier } F2$
using *image-b-S-bI starts-with-subset* **by** *blast*

lemma *S-a-disj-image-a-S-aI*: $S-a \cap (\otimes_{F2}) \text{ a-elt } \text{' } S-aI = \{\}$
using *image-a-S-aI* **by** *blast*

lemma *S-b-disj-image-b-S-bI*: $S-b \cap (\otimes_{F2}) \text{ b-elt } \text{' } S-bI = \{\}$
using *image-b-S-bI* **by** *blast*

definition *aI-ray* :: $(\text{bool} \times \text{gen2}) \text{ list set where}$
aI-ray = $\{\text{replicate } n \text{ (True, A) } \mid n. \text{ True}\}$

lemma *canceled-replicate-aI*: *canceled (replicate n (True, A))*
unfolding *canceled-def*
proof
assume *Domainp cancels-to-1 (replicate n (True, A))*
then obtain *w where cancels-to-1 (replicate n (True, A)) w*
by *auto*
then obtain *i where i: Suc i < length (replicate n (True, A))*
and *canceling (replicate n (True, A) ! i) (replicate n (True, A) ! Suc i)*
by *(auto simp: cancels-to-1-def cancels-to-1-at-def)*
hence *canceling (True, A) (True, A)*
by *simp*
thus *False*
by *(simp add: canceling-def)*
qed

lemma *aI-ray-subset-carrier*: $aI\text{-ray} \subseteq \text{carrier } F2$
unfolding *aI-ray-def*
by *(auto intro: canceled-in-F2 canceled-replicate-aI)*

lemma *aI-ray-cases*:
assumes $w \in aI\text{-ray}$
shows $w = [] \vee w \in S-aI$
proof –
from *assms obtain n where w: w = replicate n (True, A)*
unfolding *aI-ray-def* **by** *blast*
show *?thesis*
proof *(cases n)*
case *0*
with *w show ?thesis* **by** *simp*
next
case *(Suc m)*
hence $w \neq [] \text{ hd } w = (\text{True, A}) \text{ } w \in \text{carrier } F2$
using *w aI-ray-subset-carrier assms* **by** *auto*
thus *?thesis*
by *(auto simp: starts-with-def)*
qed
qed

lemma *a-mult-aI-ray-pos*:
 (\otimes_{F2}) *a-elt* ‘ $(aI\text{-ray} - \{\emptyset\}) = aI\text{-ray}$

proof
show (\otimes_{F2}) *a-elt* ‘ $(aI\text{-ray} - \{\emptyset\}) \subseteq aI\text{-ray}$
proof
fix y
assume $y \in (\otimes_{F2})$ *a-elt* ‘ $(aI\text{-ray} - \{\emptyset\})$
then obtain w **where** $w\text{-ray}: w \in aI\text{-ray}$ **and** $w\text{-ne}: w \neq \emptyset$
and $y: y = a\text{-elt} \otimes_{F2} w$
by *auto*
from $w\text{-ray}$ **obtain** n **where** $w: w = \text{replicate } n \text{ (True, A)}$
unfolding *aI-ray-def* **by** *blast*
from $w\text{-ne}$ w **obtain** m **where** $n: n = \text{Suc } m$
by $(\text{cases } n)$ *auto*
have $w\text{-S}: w \in S\text{-aI}$
using *aI-ray-cases*[*OF w-ray*] $w\text{-ne}$ **by** *blast*
have $y = \text{tl } w$
using *a-mult-starts-with-aI*[*OF w-S*] y **by** *simp*
also have $\dots = \text{replicate } m \text{ (True, A)}$
using w n **by** *simp*
finally show $y \in aI\text{-ray}$
unfolding *aI-ray-def* **by** *blast*

qed
show $aI\text{-ray} \subseteq (\otimes_{F2})$ *a-elt* ‘ $(aI\text{-ray} - \{\emptyset\})$
proof
fix y
assume $y \in aI\text{-ray}$
then obtain n **where** $y: y = \text{replicate } n \text{ (True, A)}$
unfolding *aI-ray-def* **by** *blast*
let $?w = \text{replicate } (\text{Suc } n) \text{ (True, A)}$
have $w\text{-ray}: ?w \in aI\text{-ray}$
unfolding *aI-ray-def* **by** *blast*
have $w\text{-ne}: ?w \neq \emptyset$
by *simp*
have $w\text{-S}: ?w \in S\text{-aI}$
using *aI-ray-cases*[*OF w-ray*] $w\text{-ne}$ **by** *blast*
have $a\text{-elt} \otimes_{F2} ?w = \text{tl } ?w$
by $(\text{rule } a\text{-mult-starts-with-aI}[OF w-S])$
also have $\dots = y$
using y **by** *simp*
finally show $y \in (\otimes_{F2})$ *a-elt* ‘ $(aI\text{-ray} - \{\emptyset\})$
using $w\text{-ray}$ $w\text{-ne}$ **by** *blast*

qed
qed

lemma *a-mult-nonray-stays-nonray*:
assumes $w \in S\text{-aI}$ **and** $w \notin aI\text{-ray}$
shows $a\text{-elt} \otimes_{F2} w \notin aI\text{-ray}$

proof
assume *ray*: $a\text{-elt} \otimes_{F2} w \in aI\text{-ray}$
from *assms(1)* **have** *w-ne*: $w \neq []$ **and** *w-hd*: $hd\ w = (True, A)$
and *w-carrier*: $w \in carrier\ F2$
by (*auto simp: starts-with-def*)
from *w-ne w-hd* **obtain** *u* **where** $w = (True, A) \# u$
by (*cases w*) *auto*
have $tl\ w \in aI\text{-ray}$
using *ray a-mult-starts-with-aI[OF assms(1)]* **by** *simp*
then obtain *n* **where** $u = replicate\ n\ (True, A)$
unfolding *aI-ray-def* **using** *w* **by** *auto*
hence $w = replicate\ (Suc\ n)\ (True, A)$
using *w* **by** *simp*
hence $w \in aI\text{-ray}$
unfolding *aI-ray-def* **by** *blast*
with *assms(2)* **show** *False*
by *contradiction*
qed

lemma *image-a-S-aI-nonray*:
 $(\otimes_{F2})\ a\text{-elt}\ ' (S\text{-aI} - aI\text{-ray}) = carrier\ F2 - S\text{-a} - aI\text{-ray}$
proof
show $(\otimes_{F2})\ a\text{-elt}\ ' (S\text{-aI} - aI\text{-ray}) \subseteq carrier\ F2 - S\text{-a} - aI\text{-ray}$
proof
fix *y*
assume $y \in (\otimes_{F2})\ a\text{-elt}\ ' (S\text{-aI} - aI\text{-ray})$
then obtain *w* **where** $w \in S\text{-aI} \wedge w \notin aI\text{-ray}$
and $y = a\text{-elt} \otimes_{F2} w$
by *auto*
have $y \in carrier\ F2 - S\text{-a}$
using *image-a-S-aI w(1) y* **by** *blast*
moreover have $y \notin aI\text{-ray}$
using *a-mult-nonray-stays-nonray[OF w] y* **by** *simp*
ultimately show $y \in carrier\ F2 - S\text{-a} - aI\text{-ray}$
by *simp*

qed
show $carrier\ F2 - S\text{-a} - aI\text{-ray} \subseteq (\otimes_{F2})\ a\text{-elt}\ ' (S\text{-aI} - aI\text{-ray})$
proof
fix *y*
assume $y \in carrier\ F2 - S\text{-a} - aI\text{-ray}$
hence $y \in carrier\ F2 - S\text{-a}$
by *simp*
then obtain *w* **where** $w \in S\text{-aI}$ **and** *y-eq*: $y = a\text{-elt} \otimes_{F2} w$
using *image-a-S-aI* **by** *blast*
have $w \notin aI\text{-ray}$
proof
assume $w\text{-ray}$: $w \in aI\text{-ray}$
with *w* **have** $w \in aI\text{-ray} - \{[]\}$
by (*auto simp: starts-with-def*)

```

    hence  $y \in aI\text{-ray}$ 
      using  $y\text{-eq } a\text{-mult-}aI\text{-ray-pos}$  by blast
    with  $y$  show  $False$ 
      by simp
  qed
  thus  $y \in (\otimes_{F2}) a\text{-elt } (S\text{-}aI - aI\text{-ray})$ 
    using  $w y\text{-eq}$  by blast
  qed
  qed

theorem  $F2\text{-paradoxical-partition}$ :
   $\exists P Q :: ((bool \times gen2) list) set list. \exists gP gQ :: ((bool \times gen2) list) list.$ 
   $length P = length gP \wedge length Q = length gQ \wedge$ 
   $set gP \subseteq carrier F2 \wedge set gQ \subseteq carrier F2 \wedge$ 
   $pairwise\text{-disjoint } (P @ Q) \wedge$ 
   $pairwise\text{-disjoint } (map2 F2\text{-act.image-set } gP P) \wedge$ 
   $pairwise\text{-disjoint } (map2 F2\text{-act.image-set } gQ Q) \wedge$ 
   $carrier F2 = (\bigcup_{i < length P} P ! i) \cup (\bigcup_{i < length Q} Q ! i) \wedge$ 
   $carrier F2 = (\bigcup_{i < length P} F2\text{-act.image-set } (gP ! i) (P ! i)) \wedge$ 
   $carrier F2 = (\bigcup_{i < length Q} F2\text{-act.image-set } (gQ ! i) (Q ! i))$ 

proof -
  let  $?A1 = S\text{-}a \cup aI\text{-ray}$ 
  let  $?A2 = S\text{-}aI - aI\text{-ray}$ 
  let  $?B1 = S\text{-}b$ 
  let  $?B2 = S\text{-}bI$ 
  let  $?P = [?A1, ?A2]$ 
  let  $?Q = [?B1, ?B2]$ 
  let  $?gP = [ [] :: (bool \times gen2) list, a\text{-elt} ]$ 
  let  $?gQ = [ [] :: (bool \times gen2) list, b\text{-elt} ]$ 

  have  $ray\text{-disj-Sa}: aI\text{-ray} \cap S\text{-}a = \{ \}$ 
    using  $aI\text{-ray-cases starts-with-pairwise-disjoint}$  by blast
  have  $ray\text{-disj-Sb}: aI\text{-ray} \cap S\text{-}b = \{ \}$ 
    using  $aI\text{-ray-cases starts-with-pairwise-disjoint}$  by blast
  have  $ray\text{-disj-SbI}: aI\text{-ray} \cap S\text{-}bI = \{ \}$ 
    using  $aI\text{-ray-cases starts-with-pairwise-disjoint}$  by blast

  have  $source\text{-cover}: carrier F2 =$ 
     $(\bigcup_{i < length ?P} ?P ! i) \cup (\bigcup_{i < length ?Q} ?Q ! i)$ 
proof
  show  $carrier F2 \subseteq (\bigcup_{i < length ?P} ?P ! i) \cup (\bigcup_{i < length ?Q} ?Q ! i)$ 
proof
  fix  $w$ 
  assume  $w: w \in carrier F2$ 
  show  $w \in (\bigcup_{i < length ?P} ?P ! i) \cup (\bigcup_{i < length ?Q} ?Q ! i)$ 
proof ( $cases w \in S\text{-}a \vee w \in S\text{-}aI \vee w \in S\text{-}b \vee w \in S\text{-}bI$ )
  case  $True$ 
  then consider  $w \in S\text{-}a \mid w \in S\text{-}aI \mid w \in S\text{-}b \mid w \in S\text{-}bI$ 
    by blast

```

```

thus ?thesis
proof cases
  case 1
  then show ?thesis by (simp add: lessThan-Suc)
next
  case 2
  then show ?thesis
  proof (cases w ∈ aI-ray)
    case True
    then show ?thesis by (simp add: lessThan-Suc)
  next
    case False
    with 2 show ?thesis by (simp add: lessThan-Suc)
  qed
next
  case 3
  then show ?thesis by (simp add: lessThan-Suc)
next
  case 4
  then show ?thesis by (simp add: lessThan-Suc)
  qed
next
  case False
  with F2-decomposition w have w = []
  by blast
  hence w ∈ aI-ray
  unfolding aI-ray-def by (auto intro!: exI[of - 0])
  thus ?thesis
  by (simp add: lessThan-Suc)
  qed
qed
show (⋃ i < length ?P. ?P ! i) ∪ (⋃ i < length ?Q. ?Q ! i) ⊆ carrier F2
  using starts-with-subset[of False A] starts-with-subset[of True A]
  starts-with-subset[of False B] starts-with-subset[of True B]
  aI-ray-subset-carrier
  by (auto simp: lessThan-Suc)
qed

have source-disj: pairwise-disjoint (?P @ ?Q)
  using starts-with-pairwise-disjoint ray-disj-Sa ray-disj-Sb ray-disj-SbI
  by (auto simp: pairwise-disjoint-def nth-Cons' nth-append Int-commute)

have im-P0: F2-act.image-set ([::(bool × gen2) list]) ?A1 = ?A1
  using starts-with-subset aI-ray-subset-carrier
  by (intro F2-act.image-set-unit) auto
have im-P1: F2-act.image-set a-elt ?A2 = carrier F2 - S-a - aI-ray
  by (simp add: F2-act.image-set-def image-a-S-aI-nonray)
have im-Q0: F2-act.image-set ([::(bool × gen2) list]) ?B1 = ?B1
  using starts-with-subset

```

by (rule *F2-act.image-set-unit*)
have *im-Q1*: *F2-act.image-set b-elt ?B2* = (\otimes_{F2}) *b-elt ‘ S-bI*
 by (simp add: *F2-act.image-set-def*)

have *map2P*: *map2 F2-act.image-set ?gP ?P* = [*?A1*, *carrier F2 – S-a – aI-ray*]
 using *im-P0 im-P1* by simp
have *map2Q*: *map2 F2-act.image-set ?gQ ?Q* = [*S-b*, (\otimes_{F2}) *b-elt ‘ S-bI*]
 using *im-Q0 im-Q1* by simp

have *imageP-disj*: *pairwise-disjoint (map2 F2-act.image-set ?gP ?P)*
 using *map2P* by (auto simp: *pairwise-disjoint-def nth-Cons' Int-commute*)
have *imageQ-disj*: *pairwise-disjoint (map2 F2-act.image-set ?gQ ?Q)*
 using *S-b-disj-image-b-S-bI map2Q*
 by (auto simp: *pairwise-disjoint-def nth-Cons' Int-commute*)

have *coverP*: *carrier F2* =
 $(\bigcup_{i < \text{length } ?P} \text{F2-act.image-set } (?gP ! i) (?P ! i))$
proof –
have $(\bigcup_{i < \text{length } ?P} \text{F2-act.image-set } (?gP ! i) (?P ! i)) =$
 $?A1 \cup (\text{carrier } F2 - S-a - aI\text{-ray})$
 using *im-P0 im-P1* by (auto simp: *lessThan-Suc nth-Cons'*)
also have ... = *carrier F2*
 using *starts-with-subset aI-ray-subset-carrier* by auto
finally show *?thesis*
 by simp
qed

have *coverQ*: *carrier F2* =
 $(\bigcup_{i < \text{length } ?Q} \text{F2-act.image-set } (?gQ ! i) (?Q ! i))$
proof –
have $(\bigcup_{i < \text{length } ?Q} \text{F2-act.image-set } (?gQ ! i) (?Q ! i)) =$
 $S-b \cup (\otimes_{F2}) \text{ b-elt ‘ S-bI}$
 using *im-Q0 im-Q1* by (auto simp: *lessThan-Suc nth-Cons'*)
also have ... = *carrier F2*
 by (rule *S-b-un-image-b-S-bI*)
finally show *?thesis*
 by simp
qed

show *?thesis*
proof (*intro exI conjI*)
show *length ?P = length ?gP*
 by simp
show *length ?Q = length ?gQ*
 by simp
show *set ?gP \subseteq carrier F2*
 by simp
show *set ?gQ \subseteq carrier F2*
 by simp

```

show pairwise-disjoint (?P @ ?Q)
  by (rule source-disj)
show pairwise-disjoint (map2 F2-act.image-set ?gP ?P)
  by (rule imageP-disj)
show pairwise-disjoint (map2 F2-act.image-set ?gQ ?Q)
  by (rule imageQ-disj)
show carrier F2 = (⋃ i<length ?P. ?P ! i) ∪ (⋃ i<length ?Q. ?Q ! i)
  by (rule source-cover)
show carrier F2 = (⋃ i<length ?P. F2-act.image-set (?gP ! i) (?P ! i))
  by (rule coverP)
show carrier F2 = (⋃ i<length ?Q. F2-act.image-set (?gQ ! i) (?Q ! i))
  by (rule coverQ)
qed
qed

```

We unfold *F2-act.paradoxical* directly to avoid locale-instantiation complications with implicit polymorphic variables.

theorem *F2-paradoxical*:

F2-act.paradoxical (carrier F2)

proof –

let ?P = [S-a, S-aI] **and** ?Q = [S-b, S-bI]

let ?gP = [::(bool × gen2) list, a-elt]

let ?gQ = [::(bool × gen2) list, b-elt]

have lensP: length ?P = length ?gP **by** simp

have lensQ: length ?Q = length ?gQ **by** simp

have closedP: set ?gP ⊆ carrier F2 **by** simp

have closedQ: set ?gQ ⊆ carrier F2 **by** simp

have disj-P: S-a ∩ S-aI = {}

by (simp add: starts-with-pairwise-disjoint)

have disj-Q: S-b ∩ S-bI = {}

by (simp add: starts-with-pairwise-disjoint)

have disj-PQ: (S-a ∪ S-aI) ∩ (S-b ∪ S-bI) = {}

using starts-with-pairwise-disjoint **by** blast

have disj-all: pairwise-disjoint (?P @ ?Q)

using disj-P disj-Q disj-PQ

by (auto simp: pairwise-disjoint-def nth-Cons' nth-append Int-commute)

have im-P0: F2-act.image-set ([::(bool × gen2) list) S-a = S-a

using starts-with-subset

by (rule F2-act.image-set-unit)

have im-Q0: F2-act.image-set ([::(bool × gen2) list) S-b = S-b

using starts-with-subset

by (rule F2-act.image-set-unit)

have im-P1: F2-act.image-set a-elt S-aI = (⊗_{F2}) a-elt ‘ S-aI

by (simp add: F2-act.image-set-def)

```

have im-Q1: F2-act.image-set b-elt S-bI = ( $\otimes_{F2}$ ) b-elt ' S-bI
  by (simp add: F2-act.image-set-def)

have map2P: map2 F2-act.image-set ?gP ?P = [S-a, ( $\otimes_{F2}$ ) a-elt ' S-aI]
  using im-P0 im-P1 by simp
have map2Q: map2 F2-act.image-set ?gQ ?Q = [S-b, ( $\otimes_{F2}$ ) b-elt ' S-bI]
  using im-Q0 im-Q1 by simp

have disj-im-P: pairwise-disjoint (map2 F2-act.image-set ?gP ?P)
  using S-a-disj-image-a-S-aI map2P
  by (auto simp: pairwise-disjoint-def nth-Cons' Int-commute)
have disj-im-Q: pairwise-disjoint (map2 F2-act.image-set ?gQ ?Q)
  using S-b-disj-image-b-S-bI map2Q
  by (auto simp: pairwise-disjoint-def nth-Cons' Int-commute)

have un-sub: ( $\bigcup_{i < \text{length } ?P} ?P ! i$ )  $\cup$  ( $\bigcup_{i < \text{length } ?Q} ?Q ! i$ )  $\subseteq$  carrier F2
  using starts-with-subset
  by (auto simp: lessThan-Suc nth-Cons')

have cov-P: carrier F2 = ( $\bigcup_{i < \text{length } ?P} F2\text{-act.image-set } (?gP ! i) (?P ! i)$ )
proof -
  have ( $\bigcup_{i < \text{length } ?P} F2\text{-act.image-set } (?gP ! i) (?P ! i)$ )
    = F2-act.image-set [] S-a  $\cup$  F2-act.image-set a-elt S-aI
    by (auto simp: lessThan-Suc nth-Cons')
  also have ... = S-a  $\cup$  ( $\otimes_{F2}$ ) a-elt ' S-aI
    using im-P0 im-P1 by simp
  also have ... = carrier F2 by (rule S-a-un-image-a-S-aI)
  finally show ?thesis by simp
qed

have cov-Q: carrier F2 = ( $\bigcup_{i < \text{length } ?Q} F2\text{-act.image-set } (?gQ ! i) (?Q ! i)$ )
proof -
  have ( $\bigcup_{i < \text{length } ?Q} F2\text{-act.image-set } (?gQ ! i) (?Q ! i)$ )
    = F2-act.image-set [] S-b  $\cup$  F2-act.image-set b-elt S-bI
    by (auto simp: lessThan-Suc nth-Cons')
  also have ... = S-b  $\cup$  ( $\otimes_{F2}$ ) b-elt ' S-bI
    using im-Q0 im-Q1 by simp
  also have ... = carrier F2 by (rule S-b-un-image-b-S-bI)
  finally show ?thesis by simp
qed

show ?thesis
  unfolding F2-act.paradoxical-def
  using lensP lensQ closedP closedQ disj-all disj-im-P disj-im-Q
    un-sub cov-P cov-Q
  by blast
qed
end

```

```

theory Free-Rotations-SO3
  imports
    Banach-Tarski.F2-Paradox
    Banach-Tarski.Free-Action-Paradox
    HOL-Computational-Algebra.Primes
    Free-Groups.PingPongLemma
begin

```

18 Rotations of three-space

A rotation is an orthogonal transformation of determinant 1.

```

definition rotation :: (real^3 ⇒ real^3) ⇒ bool where
  rotation f ⟷ orthogonal-transformation f ∧ matrix-det (matrix f) = 1

```

```

definition SO3 :: (real^3 ⇒ real^3) set where
  SO3 = {f. rotation f}

```

SO(3) contains the identity and is closed under composition.

```

lemma id-in-SO3: id ∈ SO3

```

```

proof –
  have orthogonal-transformation (id::real^3 ⇒ real^3)
    using orthogonal-transformation-id by (simp add: id-def)
  moreover have matrix-det (matrix (id::real^3 ⇒ real^3)) = 1
    by simp
  ultimately show ?thesis
    unfolding SO3-def rotation-def by simp
qed

```

```

lemma SO3-closed-compose:

```

```

  assumes f ∈ SO3 and g ∈ SO3
  shows f ∘ g ∈ SO3

```

```

proof –
  from assms have f-orth: orthogonal-transformation f
    and g-orth: orthogonal-transformation g
    and f-det: matrix-det (matrix f) = 1
    and g-det: matrix-det (matrix g) = 1
    unfolding SO3-def rotation-def by auto
  have f-lin: linear f using f-orth by (rule orthogonal-transformation-linear)
  have g-lin: linear g using g-orth by (rule orthogonal-transformation-linear)

  have orth-comp: orthogonal-transformation (f ∘ g)
    using f-orth g-orth by (rule orthogonal-transformation-compose)

  have matrix (f ∘ g) = matrix f ** matrix g
    using g-lin f-lin by (rule matrix-compose)

```

hence $\text{matrix-det (matrix (f \circ g))} = \text{matrix-det (matrix f)} * \text{matrix-det (matrix g)}$
by (*simp add: det-mul*)
also have $\dots = 1$ **using** *f-det g-det* **by** *simp*
finally have $\text{matrix-det (matrix (f \circ g))} = 1$.

with *orth-comp* **show** *?thesis*
unfolding *SO3-def rotation-def* **by** *simp*
qed

SO(3) elements preserve the unit sphere.

lemma *rotation-preserves-sphere2*:
assumes $f \in \text{SO3}$ **and** $x \in \text{sphere2}$
shows $f x \in \text{sphere2}$
proof –
from *assms(1)* **have** *orthogonal-transformation f*
unfolding *SO3-def rotation-def* **by** *auto*
hence $\text{norm (f x)} = \text{norm x}$
by (*rule orthogonal-transformation-norm*)
with *assms(2)* **show** *?thesis*
by (*simp add: sphere2-def*)
qed

lemma *SO3-bij*:
assumes $f \in \text{SO3}$
shows *bij f*
using *assms orthogonal-transformation-bij*
unfolding *SO3-def rotation-def* **by** *auto*

19 Two distinguished rotations

We use the rotations R_x (about the x-axis) and R_z (about the z-axis) by the same angle ϑ with $\cos \vartheta = 1/3$. The exact-form witnesses are unimportant for the paradoxical decomposition; what matters is the abstract conclusion that these generate a free F_2 inside SO(3).

definition *rot-c* :: *real* **where**
 $\text{rot-c} = 1 / 3$

definition *rot-s* :: *real* **where**
 $\text{rot-s} = \text{sqrt } 8 / 3$

lemma *rot-c-sq-plus-rot-s-sq*: $\text{rot-c}^2 + \text{rot-s}^2 = 1$
unfolding *rot-c-def rot-s-def*
by (*simp add: power-divide*)

lemma *rot-c-mul-plus-rot-s-mul* [*simp*]: $\text{rot-c} * \text{rot-c} + \text{rot-s} * \text{rot-s} = 1$
using *rot-c-sq-plus-rot-s-sq* **by** (*simp add: power2-eq-square*)

definition $Rx\text{-mat} :: \text{real}^3 \times \text{real}^3$ **where**

$Rx\text{-mat} = \text{vector}$ [
 vector [1, 0, 0],
 vector [0, rot-c, - rot-s],
 vector [0, rot-s, rot-c]]

definition $Rz\text{-mat} :: \text{real}^3 \times \text{real}^3$ **where**

$Rz\text{-mat} = \text{vector}$ [
 vector [rot-c, - rot-s, 0],
 vector [rot-s, rot-c, 0],
 vector [0, 0, 1]]

definition $Rx\text{-inv-mat} :: \text{real}^3 \times \text{real}^3$ **where**

$Rx\text{-inv-mat} = \text{vector}$ [
 vector [1, 0, 0],
 vector [0, rot-c, rot-s],
 vector [0, - rot-s, rot-c]]

definition $Rz\text{-inv-mat} :: \text{real}^3 \times \text{real}^3$ **where**

$Rz\text{-inv-mat} = \text{vector}$ [
 vector [rot-c, rot-s, 0],
 vector [- rot-s, rot-c, 0],
 vector [0, 0, 1]]

definition $Rx :: \text{real}^3 \Rightarrow \text{real}^3$ **where**

$Rx\ v = Rx\text{-mat} * v$

definition $Rz :: \text{real}^3 \Rightarrow \text{real}^3$ **where**

$Rz\ v = Rz\text{-mat} * v$

definition $Rx\text{-inv} :: \text{real}^3 \Rightarrow \text{real}^3$ **where**

$Rx\text{-inv}\ v = Rx\text{-inv-mat} * v$

definition $Rz\text{-inv} :: \text{real}^3 \Rightarrow \text{real}^3$ **where**

$Rz\text{-inv}\ v = Rz\text{-inv-mat} * v$

lemma $Rx\text{-mat-orthogonal}$: orthogonal-matrix $Rx\text{-mat}$

unfolding orthogonal-matrix $Rx\text{-mat-def}$

by (simp add: matrix-matrix-mult-def transpose-def mat-def vec-eq-iff vector-def sum-3 forall-3

rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps)

lemma $Rz\text{-mat-orthogonal}$: orthogonal-matrix $Rz\text{-mat}$

unfolding orthogonal-matrix $Rz\text{-mat-def}$

by (simp add: matrix-matrix-mult-def transpose-def mat-def vec-eq-iff vector-def sum-3 forall-3

rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps)

lemma *Rx-inv-mat-orthogonal*: orthogonal-matrix *Rx-inv-mat*
unfolding *orthogonal-matrix Rx-inv-mat-def*
by (*simp add: matrix-matrix-mult-def transpose-def mat-def vec-eq-iff vector-def sum-3 forall-3 rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps*)

lemma *Rz-inv-mat-orthogonal*: orthogonal-matrix *Rz-inv-mat*
unfolding *orthogonal-matrix Rz-inv-mat-def*
by (*simp add: matrix-matrix-mult-def transpose-def mat-def vec-eq-iff vector-def sum-3 forall-3 rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps*)

lemma *det-Rx-mat [simp]*: matrix-det *Rx-mat* = 1
unfolding *Rx-mat-def*
by (*simp add: det-3 vector-def rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps*)

lemma *det-Rz-mat [simp]*: matrix-det *Rz-mat* = 1
unfolding *Rz-mat-def*
by (*simp add: det-3 vector-def rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps*)

lemma *det-Rx-inv-mat [simp]*: matrix-det *Rx-inv-mat* = 1
unfolding *Rx-inv-mat-def*
by (*simp add: det-3 vector-def rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps*)

lemma *det-Rz-inv-mat [simp]*: matrix-det *Rz-inv-mat* = 1
unfolding *Rz-inv-mat-def*
by (*simp add: det-3 vector-def rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps*)

lemma *Rx-mat-inverse-left*: *Rx-inv-mat ** Rx-mat* = mat 1
unfolding *Rx-inv-mat-def Rx-mat-def*
by (*simp add: matrix-matrix-mult-def mat-def vec-eq-iff vector-def sum-3 forall-3 rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps*)

lemma *Rx-mat-inverse-right*: *Rx-mat ** Rx-inv-mat* = mat 1
unfolding *Rx-inv-mat-def Rx-mat-def*
by (*simp add: matrix-matrix-mult-def mat-def vec-eq-iff vector-def sum-3 forall-3 rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps*)

lemma *Rz-mat-inverse-left*: *Rz-inv-mat ** Rz-mat* = mat 1
unfolding *Rz-inv-mat-def Rz-mat-def*
by (*simp add: matrix-matrix-mult-def mat-def vec-eq-iff vector-def sum-3 forall-3 rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps*)

lemma *Rz-mat-inverse-right*: *Rz-mat ** Rz-inv-mat* = mat 1
unfolding *Rz-inv-mat-def Rz-mat-def*
by (*simp add: matrix-matrix-mult-def mat-def vec-eq-iff vector-def sum-3 forall-3 rot-c-sq-plus-rot-s-sq power2-eq-square algebra-simps*)

The concrete generators and their inverses lie in $SO(3)$.

lemma *Rx-in-SO3*: $Rx \in SO3$
unfolding *SO3-def rotation-def Rx-def*
by (*simp add: orthogonal-transformation-matrix Rx-mat-orthogonal matrix-vector-mul-linear*)

lemma *Rz-in-SO3*: $Rz \in SO3$
unfolding *SO3-def rotation-def Rz-def*
by (*simp add: orthogonal-transformation-matrix Rz-mat-orthogonal matrix-vector-mul-linear*)

lemma *Rx-inv-in-SO3*: $Rx\text{-inv} \in SO3$
unfolding *SO3-def rotation-def Rx-inv-def*
by (*simp add: orthogonal-transformation-matrix Rx-inv-mat-orthogonal matrix-vector-mul-linear*)

lemma *Rz-inv-in-SO3*: $Rz\text{-inv} \in SO3$
unfolding *SO3-def rotation-def Rz-inv-def*
by (*simp add: orthogonal-transformation-matrix Rz-inv-mat-orthogonal matrix-vector-mul-linear*)

The named inverse rotations are two-sided inverses of the generators.

lemma *Rx-inv-left*: $Rx\text{-inv} \circ Rx = id$
by (*rule ext*) (*simp add: Rx-inv-def Rx-def matrix-vector-mul-assoc Rx-mat-inverse-left*)

lemma *Rz-inv-left*: $Rz\text{-inv} \circ Rz = id$
by (*rule ext*) (*simp add: Rz-inv-def Rz-def matrix-vector-mul-assoc Rz-mat-inverse-left*)

lemma *Rx-inv-right*: $Rx \circ Rx\text{-inv} = id$
by (*rule ext*) (*simp add: Rx-inv-def Rx-def matrix-vector-mul-assoc Rx-mat-inverse-right*)

lemma *Rz-inv-right*: $Rz \circ Rz\text{-inv} = id$
by (*rule ext*) (*simp add: Rz-inv-def Rz-def matrix-vector-mul-assoc Rz-mat-inverse-right*)

20 The rotations as bijections of the sphere

definition *sphere-perm* :: $(\text{real}^3 \Rightarrow \text{real}^3) \Rightarrow (\text{real}^3 \Rightarrow \text{real}^3)$ **where**
sphere-perm $f = \text{restrict } f \text{ sphere2}$

lemma *SO3-bij-betw-sphere2*:
assumes $f \in SO3$
shows *bij-betw* $f \text{ sphere2 sphere2}$
proof –
have $f \text{ ' sphere2} \subseteq \text{ sphere2}$
using *rotation-preserves-sphere2* [*OF assms*] **by** *auto*
moreover **have** $\text{ sphere2} \subseteq f \text{ ' sphere2}$
proof
fix y **assume** $y: y \in \text{ sphere2}$
from *SO3-bij* [*OF assms*] **obtain** x **where** $x: y = f x$
by (*meson bij-pointE*)
have $x \in \text{ sphere2}$
proof –
from *assms* **have** *orth*: *orthogonal-transformation* f
unfolding *SO3-def rotation-def* **by** *auto*

```

from  $x\ y$  have  $\text{norm } (f\ x) = 1$ 
  by (simp add: sphere2-def)
moreover from orth have  $\text{norm } (f\ x) = \text{norm } x$ 
  by (rule orthogonal-transformation-norm)
ultimately show ?thesis
  by (simp add: sphere2-def)
qed
with  $x$  show  $y \in f^{-1} \text{ sphere2}$ 
  by blast
qed
moreover have inj-on f sphere2
proof –
  have inj f
    using SO3-bij[OF assms] by (rule bij-is-inj)
  thus ?thesis
    by (rule inj-on-subset simp)
qed
ultimately show ?thesis
  unfolding bij-betw-def by auto
qed

```

```

lemma sphere-perm-in-Bij:
  assumes  $f \in \text{SO3}$ 
  shows sphere-perm f \in Bij sphere2
  using SO3-bij-betw-sphere2[OF assms]
  by (simp add: sphere-perm-def Bij-def)

```

```

lemma sphere-perm-mult:
  assumes sphere-perm f \in Bij sphere2 sphere-perm g \in Bij sphere2
  shows sphere-perm f \otimes_{BijGroup sphere2} sphere-perm g = sphere-perm (f \circ g)
proof (rule ext)
  fix  $x$ 
  show (sphere-perm f \otimes_{BijGroup sphere2} sphere-perm g)  $x = \text{sphere-perm } (f \circ g)$ 
 $x$ 
  proof (cases x \in sphere2)
    case True
      from assms True have sphere-perm g x \in sphere2
        by (auto simp: Bij-def bij-betw-def)
      with assms True show ?thesis
        by (simp add: sphere-perm-def BijGroup-def compose-def restrict-def)
    next
      case False
      with assms show ?thesis
        by (simp add: sphere-perm-def BijGroup-def compose-def restrict-def)
  qed
qed

```

```

lemma sphere-perm-id: sphere-perm id = 1_{BijGroup sphere2}
  by (simp add: sphere-perm-def BijGroup-def id-def)

```

lemma *sphere-perm-Rx* [simp]: *sphere-perm Rx* \in *Bij sphere2*
 by (rule *sphere-perm-in-Bij*) (rule *Rx-in-SO3*)

lemma *sphere-perm-Rx-inv* [simp]: *sphere-perm Rx-inv* \in *Bij sphere2*
 by (rule *sphere-perm-in-Bij*) (rule *Rx-inv-in-SO3*)

lemma *sphere-perm-Rz* [simp]: *sphere-perm Rz* \in *Bij sphere2*
 by (rule *sphere-perm-in-Bij*) (rule *Rz-in-SO3*)

lemma *sphere-perm-Rz-inv* [simp]: *sphere-perm Rz-inv* \in *Bij sphere2*
 by (rule *sphere-perm-in-Bij*) (rule *Rz-inv-in-SO3*)

lemma *sphere-perm-Rx-inv-left*:
sphere-perm Rx-inv $\otimes_{\text{BijGroup sphere2}}$ *sphere-perm Rx* = $\mathbf{1}_{\text{BijGroup sphere2}}$
 by (simp add: *sphere-perm-mult Rx-inv-left sphere-perm-id*)

lemma *sphere-perm-Rx-inv-right*:
sphere-perm Rx $\otimes_{\text{BijGroup sphere2}}$ *sphere-perm Rx-inv* = $\mathbf{1}_{\text{BijGroup sphere2}}$
 by (simp add: *sphere-perm-mult Rx-inv-right sphere-perm-id*)

lemma *sphere-perm-Rz-inv-left*:
sphere-perm Rz-inv $\otimes_{\text{BijGroup sphere2}}$ *sphere-perm Rz* = $\mathbf{1}_{\text{BijGroup sphere2}}$
 by (simp add: *sphere-perm-mult Rz-inv-left sphere-perm-id*)

lemma *sphere-perm-Rz-inv-right*:
sphere-perm Rz $\otimes_{\text{BijGroup sphere2}}$ *sphere-perm Rz-inv* = $\mathbf{1}_{\text{BijGroup sphere2}}$
 by (simp add: *sphere-perm-mult Rz-inv-right sphere-perm-id*)

lemma *inv-sphere-perm-Rx*:
*inv*_{BijGroup sphere2} (*sphere-perm Rx*) = *sphere-perm Rx-inv*
proof –
 interpret *B*: group *BijGroup sphere2*
 by (rule *group-BijGroup*)
 show ?thesis
proof (rule *B.inv-equality*)
 show *sphere-perm Rx-inv* $\otimes_{\text{BijGroup sphere2}}$ *sphere-perm Rx* = $\mathbf{1}_{\text{BijGroup sphere2}}$
 by (rule *sphere-perm-Rx-inv-left*)
 show *sphere-perm Rx* \in carrier (*BijGroup sphere2*)
 by (simp add: *BijGroup-def*)
 show *sphere-perm Rx-inv* \in carrier (*BijGroup sphere2*)
 by (simp add: *BijGroup-def*)
qed
qed

lemma *inv-sphere-perm-Rz*:
*inv*_{BijGroup sphere2} (*sphere-perm Rz*) = *sphere-perm Rz-inv*
proof –
 interpret *B*: group *BijGroup sphere2*

by (rule group-BijGroup)
 show ?thesis
 proof (rule B.inv-equality)
 show sphere-perm Rz-inv \otimes BijGroup sphere2 sphere-perm Rz = $\mathbf{1}_{\text{BijGroup sphere2}}$
 by (rule sphere-perm-Rz-inv-left)
 show sphere-perm Rz \in carrier (BijGroup sphere2)
 by (simp add: BijGroup-def)
 show sphere-perm Rz-inv \in carrier (BijGroup sphere2)
 by (simp add: BijGroup-def)
 qed
 qed

interpretation sphere-Bij: group BijGroup sphere2
 by (rule group-BijGroup)

21 The map sigma: F_2 to $\text{SO}(3)$

The map *sigma* sends each reduced word in F_2 to the corresponding composition of R_x , R_z , and their inverses.

definition letter-to-rot :: bool \times gen2 \Rightarrow (real³ \Rightarrow real³) **where**
 letter-to-rot p = (case p of
 (False, A) \Rightarrow Rx
 | (True, A) \Rightarrow Rx-inv
 | (False, B) \Rightarrow Rz
 | (True, B) \Rightarrow Rz-inv)

definition rho :: gen2 \Rightarrow (real³ \Rightarrow real³) **where**
 rho x = (case x of A \Rightarrow sphere-perm Rx | B \Rightarrow sphere-perm Rz)

lemma rho-in-BijGroup:
 rho \in Gen2 \rightarrow carrier (BijGroup sphere2)
 by (auto simp: rho-def BijGroup-def)

lemma rho-image-subset-BijGroup:
 rho ‘ Gen2 \subseteq carrier (BijGroup sphere2)
 using rho-in-BijGroup by auto

definition sphere-rot-group :: (real³ \Rightarrow real³) monoid **where**
 sphere-rot-group =
 (BijGroup sphere2)(carrier := \langle rho ‘ Gen2 \rangle BijGroup sphere2)

lemma sphere-rot-subgroup:
 subgroup (carrier sphere-rot-group) (BijGroup sphere2)
 unfolding sphere-rot-group-def
 apply simp
 by (rule sphere-Bij.gen-subgroup-is-subgroup) (auto simp: rho-def BijGroup-def)

lemma sphere-rot-group-is-group: group sphere-rot-group

unfolding *sphere-rot-group-def*
by (*rule sphere-Bij.subgroup-imp-group*)
(*rule sphere-Bij.gen-subgroup-is-subgroup, rule rho-image-subset-BijGroup*)

interpretation *sphere-rot: group sphere-rot-group*
by (*rule sphere-rot-group-is-group*)

lemma *rho-in-sphere-rot-group*:
 $\rho \in \text{Gen2} \rightarrow \text{carrier sphere-rot-group}$
by (*auto simp: sphere-rot-group-def intro: gen-span.gen-gens*)

lemma *sphere-rot-inclusion-hom*:
 $id \in \text{hom sphere-rot-group (BijGroup sphere2)}$
using *sphere-Bij.gen-span-closed[OF rho-image-subset-BijGroup]*
by (*auto simp: hom-def sphere-rot-group-def*)

lemma *sphere-rot-generated*:
 $\langle \rho \text{ ' Gen2} \rangle_{\text{sphere-rot-group}} = \text{carrier sphere-rot-group}$

proof –
let $?H = \langle \rho \text{ ' Gen2} \rangle_{\text{BijGroup sphere2}}$
let $?G = (\text{BijGroup sphere2}) \{ \text{carrier} := ?H \}$
have *sub: subgroup ?H (BijGroup sphere2)*
by (*rule sphere-Bij.gen-subgroup-is-subgroup*) (*rule rho-image-subset-BijGroup*)
have *left: $\langle \rho \text{ ' Gen2} \rangle_{\text{sphere-rot-group}} \subseteq \text{carrier sphere-rot-group}$*
by (*rule sphere-rot.gen-span-closed*) (*use rho-in-sphere-rot-group in auto*)
have *right: $?H \subseteq \langle \rho \text{ ' Gen2} \rangle_{?G}$*
proof
fix x
assume $x \in ?H$
thus $x \in \langle \rho \text{ ' Gen2} \rangle_{?G}$
proof (*induct rule: gen-span.induct*)
case *gen-one*
have $1_{?G} \in \langle \rho \text{ ' Gen2} \rangle_{?G}$
by (*rule gen-span.gen-one*)
thus *?case by simp*
next
case (*gen-gens x*)
thus *?case*
by (*rule gen-span.gen-gens*)
next
case (*gen-inv x*)
have *inv-eq: $\text{inv}_{?G} x = \text{inv}_{\text{BijGroup sphere2}} x$*
using *sphere-Bij.m-inv-consistent[OF sub gen-inv.hyps(1)]* .
have $\text{inv}_{?G} x \in \langle \rho \text{ ' Gen2} \rangle_{?G}$
by (*rule gen-span.gen-inv*) (*rule gen-inv.hyps(2)*)
thus *?case*
using *inv-eq by metis*
next
case (*gen-mult x y*)

```

    have  $x \otimes_{?G} y \in \langle \text{rho} \text{ ' Gen2} \rangle_{?G}$ 
      by (rule gen-span.gen-mult) (use gen-mult.hyps in auto)
    thus ?case by simp
  qed
qed
show ?thesis
  using left right by (auto simp: sphere-rot-group-def)
qed

lemma sphere-Bij-lift-gi-rho:
  assumes  $snd\ p \in \text{Gen2}$ 
  shows  $\text{sphere-Bij.lift-gi rho } p = \text{sphere-perm (letter-to-rot } p)$ 
  using assms
  by (cases p)
    (auto simp: rho-def letter-to-rot-def sphere-Bij.lift-gi-def
      inv-sphere-perm-Rx inv-sphere-perm-Rz split: bool.splits gen2.splits)

lemma sphere-rot-inv-rho-eq-sphere-Bij:
  assumes  $i \in \text{Gen2}$ 
  shows  $\text{inv}_{\text{sphere-rot-group}} (\text{rho } i) = \text{inv}_{\text{BijGroup sphere2}} (\text{rho } i)$ 
  proof -
    have rho-i:  $\text{rho } i \in \text{carrier sphere-rot-group}$ 
      using rho-in-sphere-rot-group assms by auto
    have  $\text{inv}_{(\text{BijGroup sphere2})(\text{carrier} := \text{carrier sphere-rot-group})} (\text{rho } i) =$ 
       $\text{inv}_{\text{BijGroup sphere2}} (\text{rho } i)$ 
      using sphere-Bij.m-inv-consistent[OF sphere-rot-subgroup rho-i] .
    thus ?thesis
      by (simp add: sphere-rot-group-def)
  qed

lemma sphere-rot-lift-gi-rho-eq-sphere-Bij:
  assumes  $snd\ p \in \text{Gen2}$ 
  shows  $\text{sphere-rot.lift-gi rho } p = \text{sphere-Bij.lift-gi rho } p$ 
  using assms
  by (cases p)
    (auto simp: sphere-rot.lift-gi-def sphere-Bij.lift-gi-def
      sphere-rot-inv-rho-eq-sphere-Bij)

lemma sphere-rot-lift-rho-eq-sphere-Bij:
  assumes  $w \in \text{lists } (\text{UNIV} \times \text{Gen2})$ 
  shows  $\text{sphere-rot.lift rho } w = \text{sphere-Bij.lift rho } w$ 
  using assms
  proof (induct w)
    case Nil
    show ?case
  proof -
    have  $\text{sphere-rot.lift rho } [] = \mathbf{1}_{\text{sphere-rot-group}}$ 
      by simp
    also have  $\dots = \mathbf{1}_{\text{BijGroup sphere2}}$ 

```

```

    by (simp add: sphere-rot-group-def)
  also have ... = sphere-Bij.lift rho []
    by simp
  finally show ?case .
qed
next
case (Cons p w)
have p-gen: snd p ∈ Gen2
  by (simp add: Gen2-UNIV)
have w-lists: w ∈ lists (UNIV × Gen2)
  using Cons.prem1 by (auto simp: Gen2-UNIV)
have IH: sphere-rot.lift rho w = sphere-Bij.lift rho w
  using Cons.hyps w-lists by simp
have gi-rot-closed: sphere-rot.lift-gi rho p ∈ carrier sphere-rot-group
  using rho-in-sphere-rot-group p-gen by (rule sphere-rot.lift-gi-closed)
have tail-rot-closed: sphere-rot.lift rho w ∈ carrier sphere-rot-group
  using rho-in-sphere-rot-group w-lists by (rule sphere-rot.lift-closed)
have gi-Bij-closed: sphere-Bij.lift-gi rho p ∈ carrier (BijGroup sphere2)
  using rho-in-BijGroup p-gen by (rule sphere-Bij.lift-gi-closed)
have tail-Bij-closed: sphere-Bij.lift rho w ∈ carrier (BijGroup sphere2)
  using rho-in-BijGroup w-lists by (rule sphere-Bij.lift-closed)
have sphere-rot.lift rho (p # w) =
  sphere-rot.lift-gi rho p ⊗sphere-rot-group sphere-rot.lift rho w
  using gi-rot-closed tail-rot-closed
  by (simp add: sphere-rot.lift-def)
also have ... =
  sphere-Bij.lift-gi rho p ⊗BijGroup sphere2 sphere-Bij.lift rho w
  using IH sphere-rot.lift-gi-rho-eq-sphere-Bij[OF p-gen]
  by (simp add: sphere-rot-group-def)
also have ... = sphere-Bij.lift rho (p # w)
  using gi-Bij-closed tail-Bij-closed
  by (simp add: sphere-Bij.lift-def)
finally show ?case .
qed

lemma sphere-Bij.lift-rho-injective-if-ping-pong:
  fixes Xin Xout :: gen2 ⇒ (real3) set
  assumes sub-out: ∀ i ∈ Gen2. Xout i ⊆ sphere2
    and sub-in: ∀ i ∈ Gen2. Xin i ⊆ sphere2
    and disj-out: ∀ i ∈ Gen2. ∀ j ∈ Gen2. i ≠ j ⟶ Xout i ∩ Xout j = {}
    and disj-in: ∀ i ∈ Gen2. ∀ j ∈ Gen2. i ≠ j ⟶ Xin i ∩ Xin j = {}
    and disj-cross: ∀ i ∈ Gen2. ∀ j ∈ Gen2. Xin i ∩ Xout j = {}
    and x-sphere: x ∈ sphere2
    and ping: ∀ i ∈ Gen2. rho i ‘ (sphere2 - Xout i) ⊆ Xin i
    and pong: ∀ i ∈ Gen2. (invsphere-rot-group (rho i)) ‘ (sphere2 - Xin i) ⊆ Xout i
  shows inj-on (sphere-Bij.lift rho) (carrier F2)
proof -
  have iso: sphere-rot.lift rho ∈ iso F2 sphere-rot-group
  proof (rule ping-pong-lemma[

```

where $X = \text{sphere2}$ **and** $I = \text{Gen2}$ **and** $\text{act} = \text{id}$ **and** $g = \text{rho}$
and $X_{\text{out}} = X_{\text{out}}$ **and** $X_{\text{in}} = X_{\text{in}}$ **and** $x = x$]

show $\text{group sphere-rot-group}$
by (*rule sphere-rot-group-is-group*)

show $\text{id} \in \text{hom sphere-rot-group (BijGroup sphere2)}$
by (*rule sphere-rot-inclusion-hom*)

show $\text{rho} \in \text{Gen2} \rightarrow \text{carrier sphere-rot-group}$
by (*rule rho-in-sphere-rot-group*)

show $\langle \text{rho} \text{ ' Gen2} \rangle_{\text{sphere-rot-group}} = \text{carrier sphere-rot-group}$
by (*rule sphere-rot-generated*)

show $\forall i \in \text{Gen2}. X_{\text{out}} i \subseteq \text{sphere2}$
by (*rule sub-out*)

show $\forall i \in \text{Gen2}. X_{\text{in}} i \subseteq \text{sphere2}$
by (*rule sub-in*)

show $\forall i \in \text{Gen2}. \forall j \in \text{Gen2}. i \neq j \rightarrow X_{\text{out}} i \cap X_{\text{out}} j = \{\}$
by (*rule disj-out*)

show $\forall i \in \text{Gen2}. \forall j \in \text{Gen2}. i \neq j \rightarrow X_{\text{in}} i \cap X_{\text{in}} j = \{\}$
by (*rule disj-in*)

show $\forall i \in \text{Gen2}. \forall j \in \text{Gen2}. X_{\text{in}} i \cap X_{\text{out}} j = \{\}$
by (*rule disj-cross*)

show $x \in \text{sphere2}$
by (*rule x-sphere*)

show $\forall i. \text{Gen2} = \{i\} \rightarrow x \notin X_{\text{out}} i \wedge x \notin X_{\text{in}} i$
by *auto*

show $\forall i \in \text{Gen2}. \text{id} (\text{rho } i) \text{ ' } (\text{sphere2} - X_{\text{out}} i) \subseteq X_{\text{in}} i$
using *ping* **by** *simp*

show $\forall i \in \text{Gen2}. \text{id} (\text{inv}_{\text{sphere-rot-group}} (\text{rho } i)) \text{ ' } (\text{sphere2} - X_{\text{in}} i) \subseteq X_{\text{out}} i$
using *pong* **by** *simp*

qed

have $\text{inj-rot: inj-on (sphere-rot.lift rho) (carrier F2)}$
using *iso* **by** (*auto simp: iso-def bij-betw-def*)

show *?thesis*

proof (*rule inj-onI*)

fix $w v$

assume $w: w \in \text{carrier F2}$ **and** $v: v \in \text{carrier F2}$
and $\text{eq: sphere-Bij.lift rho } w = \text{sphere-Bij.lift rho } v$

have $w\text{-lists: } w \in \text{lists (UNIV} \times \text{Gen2)}$
using w **by** (*simp add: F2-carrier-iff*)

have $v\text{-lists: } v \in \text{lists (UNIV} \times \text{Gen2)}$
using v **by** (*simp add: F2-carrier-iff*)

have $\text{sphere-rot.lift rho } w = \text{sphere-Bij.lift rho } w$
using $w\text{-lists}$ **by** (*rule sphere-rot-lift-rho-eq-sphere-Bij*)

also have $\dots = \text{sphere-Bij.lift rho } v$
by (*rule eq*)

also have $\dots = \text{sphere-rot.lift rho } v$
using $v\text{-lists}$ **by** (*rule sphere-rot-lift-rho-eq-sphere-Bij[symmetric]*)

finally have $\text{sphere-rot.lift rho } w = \text{sphere-rot.lift rho } v$.

with $\text{inj-rot } w v$ **show** $w = v$
by (*auto simp: inj-on-def*)

qed
qed

definition $\sigma :: (\text{bool} \times \text{gen2}) \text{ list} \Rightarrow (\text{real}^3 \Rightarrow \text{real}^3)$ **where**
 $\sigma w = \text{foldr } (\lambda p f. \text{letter-to-rot } p \circ f) w \text{ id}$

lemma *letter-to-rot-cancel-left* [simp]:

$\text{letter-to-rot } (\text{True}, A) \circ \text{letter-to-rot } (\text{False}, A) = \text{id}$

$\text{letter-to-rot } (\text{False}, A) \circ \text{letter-to-rot } (\text{True}, A) = \text{id}$

$\text{letter-to-rot } (\text{True}, B) \circ \text{letter-to-rot } (\text{False}, B) = \text{id}$

$\text{letter-to-rot } (\text{False}, B) \circ \text{letter-to-rot } (\text{True}, B) = \text{id}$

by (*simp-all add: letter-to-rot-def Rx-inv-left Rx-inv-right Rz-inv-left Rz-inv-right*)

lemma *letter-to-rot-canceling*:

assumes *canceling* $p \ q$

shows $\text{letter-to-rot } p \circ \text{letter-to-rot } q = \text{id}$

using *assms*

by (*cases* p ; *cases* q)

(*auto split: bool.splits gen2.splits*)

simp: canceling-def letter-to-rot-def Rx-inv-left Rx-inv-right Rz-inv-left Rz-inv-right)

lemma *sigma-Nil* [simp]: $\sigma [] = \text{id}$

by (*simp add: sigma-def*)

lemma *sigma-Cons* [simp]: $\sigma (p \# w) = \text{letter-to-rot } p \circ \sigma w$

by (*simp add: sigma-def*)

lemma *sigma-append* [simp]: $\sigma (u @ v) = \sigma u \circ \sigma v$

by (*induct* u) (*simp-all add: o-assoc*)

lemma *sigma-singleton* [simp]: $\sigma [p] = \text{letter-to-rot } p$

by (*simp add: fun-eq-iff*)

lemma *sigma-doubleton-canceling*:

assumes *canceling* $p \ q$

shows $\sigma [p, q] = \text{id}$

using *letter-to-rot-canceling[OF assms]*

by (*simp add: o-def fun-eq-iff*)

lemma *sigma-cancels-to-1*:

assumes *cancels-to-1* $x \ y$

shows $\sigma x = \sigma y$

proof –

from *assms* **obtain** $xs1 \ x1 \ x2 \ xs2$

where *x-eq*: $x = xs1 @ x1 \# x2 \# xs2$

and *y-eq*: $y = xs1 @ xs2$

and *cancel*: *canceling* $x1 \ x2$

by (*rule cancels-to-1-unfold*)

have $\sigma x = \sigma xs1 \circ \sigma [x1, x2] \circ \sigma xs2$

```

    by (simp add: x-eq o-assoc)
  also have ... = sigma xs1 ∘ sigma xs2
    using letter-to-rot-canceling[OF cancel]
    by (auto simp: fun-eq-iff o-def)
  also have ... = sigma y
    by (simp add: y-eq)
  finally show ?thesis .
qed

```

```

lemma sigma-cancels-to:
  assumes cancels-to x y
  shows sigma x = sigma y
  using assms unfolding cancels-to-def
proof (induct rule: rtranclp-induct)
  case base
  show ?case by simp
next
  case (step y z)
  then show ?case
    using sigma-cancels-to-1 by simp
qed

```

```

lemma sigma-normalize [simp]: sigma (normalize w) = sigma w
  using sigma-cancels-to[OF normalized-cancels-to] by simp

```

```

lemma letter-to-rot-in-SO3: letter-to-rot p ∈ SO3
  by (cases p)
    (auto split: bool.splits gen2.splits
      simp: letter-to-rot-def Rx-in-SO3 Rz-in-SO3 Rx-inv-in-SO3 Rz-inv-in-SO3)

```

```

lemma sphere-perm-letter-to-rot-in-Bij [simp]:
  sphere-perm (letter-to-rot p) ∈ Bij sphere2
  by (rule sphere-perm-in-Bij) (rule letter-to-rot-in-SO3)

```

```

lemma sigma-in-SO3: sigma w ∈ SO3
proof (induct w)
  case Nil
  show ?case
    using id-in-SO3 by (simp add: id-def)
next
  case (Cons p w)
  have letter-to-rot p ∘ sigma w ∈ SO3
    using letter-to-rot-in-SO3 Cons.hyps by (rule SO3-closed-compose)
  then show ?case by (simp add: o-def)
qed

```

```

lemma sigma-preserves-sphere2:
  assumes x ∈ sphere2
  shows sigma w x ∈ sphere2

```

using *rotation-preserves-sphere2*[*OF sigma-in-SO3 assms*] .

lemma *sigma-bij*: *bij (sigma w)*

using *SO3-bij sigma-in-SO3* **by** *blast*

lemma *sphere-perm-sigma-in-Bij* [*simp*]:

sphere-perm (sigma w) ∈ Bij sphere2

by (*rule sphere-perm-in-Bij*) (*rule sigma-in-SO3*)

lemma *sphere-Bij-lift-rho-eq-sigma*:

assumes *w ∈ lists (UNIV × Gen2)*

shows *sphere-Bij.lift rho w = sphere-perm (sigma w)*

using *assms*

proof (*induct w*)

case *Nil*

show *?case*

by (*simp add: sphere-perm-id[symmetric] id-def*)

next

case (*Cons p w*)

have *p-gen: snd p ∈ Gen2*

by (*simp add: Gen2-UNIV*)

have *w-lists: w ∈ lists (UNIV × Gen2)*

using *Cons.prem* **by** (*auto simp: Gen2-UNIV*)

have *IH: sphere-Bij.lift rho w = sphere-perm (sigma w)*

using *Cons.hyps w-lists* **by** *simp*

have *singleton-lists: [p] ∈ lists (UNIV × Gen2)*

by (*simp add: Gen2-UNIV*)

have *sphere-Bij.lift rho (p # w) = sphere-Bij.lift rho ([p] @ w)*

by *simp*

also have $\dots = \text{sphere-Bij.lift rho } [p] \otimes_{\text{BijGroup sphere2}} \text{sphere-Bij.lift rho } w$

using *sphere-Bij.lift-append*[*OF rho-in-BijGroup singleton-lists w-lists*] .

also have *sphere-Bij.lift rho [p] = sphere-perm (letter-to-rot p)*

proof –

have *gi-closed: sphere-Bij.lift-gi rho p ∈ carrier (BijGroup sphere2)*

using *rho-in-BijGroup p-gen* **by** (*rule sphere-Bij.lift-gi-closed*)

have *sphere-Bij.lift rho [p] = sphere-Bij.lift-gi rho p*

using *rho-in-BijGroup p-gen*

by (*simp add: sphere-Bij.lift-def gi-closed*)

also have $\dots = \text{sphere-perm (letter-to-rot p)}$

using *p-gen* **by** (*rule sphere-Bij.lift-gi-rho*)

finally show *?thesis* .

qed

also have $\text{sphere-perm (letter-to-rot p)} \otimes_{\text{BijGroup sphere2}} \text{sphere-Bij.lift rho } w =$

$\text{sphere-perm (letter-to-rot p)} \otimes_{\text{BijGroup sphere2}} \text{sphere-perm (sigma w)}$

using *IH* **by** *simp*

also have $\dots = \text{sphere-perm (letter-to-rot p} \circ \text{sigma w)}$

by (*rule sphere-perm-mult*) *simp-all*

also have $\dots = \text{sphere-perm (sigma (p \# w))}$

by *simp*

finally show *?case* .
qed

lemma *sigma-injective-if-sphere-lift-injective*:

assumes *inj-on* (*sphere-Bij.lift rho*) (*carrier F2*)

shows *inj-on sigma* (*carrier F2*)

proof (*rule inj-onI*)

fix *w v*

assume *w*: $w \in \text{carrier } F2$ **and** *v*: $v \in \text{carrier } F2$ **and** *eq*: $\text{sigma } w = \text{sigma } v$

have *w-lists*: $w \in \text{lists } (UNIV \times Gen2)$

using *w* **by** (*simp add: F2-carrier-iff*)

have *v-lists*: $v \in \text{lists } (UNIV \times Gen2)$

using *v* **by** (*simp add: F2-carrier-iff*)

have *sphere-Bij.lift rho w = sphere-perm (sigma w)*

using *w-lists* **by** (*rule sphere-Bij-lift-rho-eq-sigma*)

also have $\dots = \text{sphere-perm } (\text{sigma } v)$

using *eq* **by** *simp*

also have $\dots = \text{sphere-Bij.lift rho } v$

using *v-lists* **by** (*rule sphere-Bij-lift-rho-eq-sigma[symmetric]*)

finally have *lift-eq*: $\text{sphere-Bij.lift rho } w = \text{sphere-Bij.lift rho } v$.

show $w = v$

using *assms w v lift-eq* **by** (*auto simp: inj-on-def*)

qed

lemma *sigma-injective-if-ping-pong*:

fixes *Xin Xout* :: $gen2 \Rightarrow (\text{real } ^3) \text{ set}$

assumes *sub-out*: $\forall i \in Gen2. Xout\ i \subseteq \text{sphere2}$

and *sub-in*: $\forall i \in Gen2. Xin\ i \subseteq \text{sphere2}$

and *disj-out*: $\forall i \in Gen2. \forall j \in Gen2. i \neq j \longrightarrow Xout\ i \cap Xout\ j = \{\}$

and *disj-in*: $\forall i \in Gen2. \forall j \in Gen2. i \neq j \longrightarrow Xin\ i \cap Xin\ j = \{\}$

and *disj-cross*: $\forall i \in Gen2. \forall j \in Gen2. Xin\ i \cap Xout\ j = \{\}$

and *x-sphere*: $x \in \text{sphere2}$

and *ping*: $\forall i \in Gen2. \text{rho } i \text{ ' } (\text{sphere2} - Xout\ i) \subseteq Xin\ i$

and *pong*: $\forall i \in Gen2. (\text{inv}_{\text{sphere-rot-group}} (\text{rho } i)) \text{ ' } (\text{sphere2} - Xin\ i) \subseteq Xout\ i$

shows *inj-on sigma* (*carrier F2*)

by (*rule sigma-injective-if-sphere-lift-injective*)

(*rule sphere-Bij-lift-rho-injective-if-ping-pong* [

OF sub-out sub-in disj-out disj-in disj-cross x-sphere ping pong])

The ping-pong criterion gives a reusable injectivity criterion for *sigma*.

22 Arithmetic freeness invariant

The classical proof that the above rotations generate a free subgroup is arithmetic rather than geometric: scale each letter by $\mathfrak{3}$, so every word has entries in the integer quadratic ring generated by $\text{sqrt } 2$. A nonempty reduced word leaves a coefficient nonzero modulo $\mathfrak{3}$ after applying it to a suitable basis vector, whereas the identity matrix would have all scaled off-axis coefficients

divisible by 3.

The following definitions make that invariant explicit. They are deliberately separated from real-vector semantics so that the finite modulo-3 induction can be discharged by simplification and case analysis before being connected back to *sigma*.

type-synonym $zsqrt2 = int \times int$

type-synonym $zvec3 = zsqrt2 \times zsqrt2 \times zsqrt2$

definition $zsqrt2\text{-val} :: zsqrt2 \Rightarrow real$ **where**

$zsqrt2\text{-val } q = of\text{-int } (fst\ q) + of\text{-int } (snd\ q) * sqrt\ 2$

definition $zsqrt2\text{-add} :: zsqrt2 \Rightarrow zsqrt2 \Rightarrow zsqrt2$ **where**

$zsqrt2\text{-add } p\ q = (fst\ p + fst\ q, snd\ p + snd\ q)$

definition $zsqrt2\text{-neg} :: zsqrt2 \Rightarrow zsqrt2$ **where**

$zsqrt2\text{-neg } p = (-\ fst\ p, -\ snd\ p)$

definition $zsqrt2\text{-sub} :: zsqrt2 \Rightarrow zsqrt2 \Rightarrow zsqrt2$ **where**

$zsqrt2\text{-sub } p\ q = zsqrt2\text{-add } p\ (zsqrt2\text{-neg } q)$

definition $zsqrt2\text{-scale} :: int \Rightarrow zsqrt2 \Rightarrow zsqrt2$ **where**

$zsqrt2\text{-scale } n\ p = (n * fst\ p, n * snd\ p)$

definition $zsqrt2\text{-mul-sqrt2} :: zsqrt2 \Rightarrow zsqrt2$ **where**

$zsqrt2\text{-mul-sqrt2 } p = (2 * snd\ p, fst\ p)$

definition $zsqrt2\text{-div3} :: zsqrt2 \Rightarrow bool$ **where**

$zsqrt2\text{-div3 } p \longleftrightarrow 3\ dvd\ fst\ p \wedge 3\ dvd\ snd\ p$

definition $zvec3\text{-div3} :: zvec3 \Rightarrow bool$ **where**

$zvec3\text{-div3 } v \longleftrightarrow$

$(case\ v\ of\ (x, y, z) \Rightarrow zsqrt2\text{-div3 } x \wedge zsqrt2\text{-div3 } y \wedge zsqrt2\text{-div3 } z)$

definition $zvec3\text{-scale} :: int \Rightarrow zvec3 \Rightarrow zvec3$ **where**

$zvec3\text{-scale } n\ v = (case\ v\ of\ (x, y, z) \Rightarrow$

$(zsqrt2\text{-scale } n\ x, zsqrt2\text{-scale } n\ y, zsqrt2\text{-scale } n\ z))$

definition $zvec3\text{-val} :: zvec3 \Rightarrow real^3$ **where**

$zvec3\text{-val } v = (case\ v\ of\ (x, y, z) \Rightarrow$

$vector\ [zsqrt2\text{-val } x, zsqrt2\text{-val } y, zsqrt2\text{-val } z])$

definition $ze\text{-x} :: zvec3$ **where**

$ze\text{-x} = ((1, 0), (0, 0), (0, 0))$

definition $ze\text{-y} :: zvec3$ **where**

$ze\text{-y} = ((0, 0), (1, 0), (0, 0))$

definition $ze\text{-z} :: zvec3$ **where**

$$ze-z = ((0, 0), (0, 0), (1, 0))$$

definition $zRx\text{-step} :: \text{zvec3} \Rightarrow \text{zvec3}$ **where**

$$\begin{aligned} zRx\text{-step } v = & (\text{case } v \text{ of } (x, y, z) \Rightarrow \\ & (\text{zsqr2-scale } 3 \ x, \\ & \text{zsqr2-sub } y \ (\text{zsqr2-scale } 2 \ (\text{zsqr2-mul-sqr2 } z)), \\ & \text{zsqr2-add } (\text{zsqr2-scale } 2 \ (\text{zsqr2-mul-sqr2 } y)) \ z)) \end{aligned}$$

definition $zRx\text{-inv-step} :: \text{zvec3} \Rightarrow \text{zvec3}$ **where**

$$\begin{aligned} zRx\text{-inv-step } v = & (\text{case } v \text{ of } (x, y, z) \Rightarrow \\ & (\text{zsqr2-scale } 3 \ x, \\ & \text{zsqr2-add } y \ (\text{zsqr2-scale } 2 \ (\text{zsqr2-mul-sqr2 } z)), \\ & \text{zsqr2-sub } z \ (\text{zsqr2-scale } 2 \ (\text{zsqr2-mul-sqr2 } y)))) \end{aligned}$$

definition $zRz\text{-step} :: \text{zvec3} \Rightarrow \text{zvec3}$ **where**

$$\begin{aligned} zRz\text{-step } v = & (\text{case } v \text{ of } (x, y, z) \Rightarrow \\ & (\text{zsqr2-sub } x \ (\text{zsqr2-scale } 2 \ (\text{zsqr2-mul-sqr2 } y)), \\ & \text{zsqr2-add } (\text{zsqr2-scale } 2 \ (\text{zsqr2-mul-sqr2 } x)) \ y, \\ & \text{zsqr2-scale } 3 \ z)) \end{aligned}$$

definition $zRz\text{-inv-step} :: \text{zvec3} \Rightarrow \text{zvec3}$ **where**

$$\begin{aligned} zRz\text{-inv-step } v = & (\text{case } v \text{ of } (x, y, z) \Rightarrow \\ & (\text{zsqr2-add } x \ (\text{zsqr2-scale } 2 \ (\text{zsqr2-mul-sqr2 } y)), \\ & \text{zsqr2-sub } y \ (\text{zsqr2-scale } 2 \ (\text{zsqr2-mul-sqr2 } x)), \\ & \text{zsqr2-scale } 3 \ z)) \end{aligned}$$

definition $zletter\text{-step} :: \text{bool} \times \text{gen2} \Rightarrow \text{zvec3} \Rightarrow \text{zvec3}$ **where**

$$\begin{aligned} zletter\text{-step } p = & (\text{case } p \text{ of} \\ & (\text{False}, A) \Rightarrow zRx\text{-step} \\ & | (\text{True}, A) \Rightarrow zRx\text{-inv-step} \\ & | (\text{False}, B) \Rightarrow zRz\text{-step} \\ & | (\text{True}, B) \Rightarrow zRz\text{-inv-step}) \end{aligned}$$

definition $zword\text{-step} :: (\text{bool} \times \text{gen2}) \text{ list} \Rightarrow \text{zvec3} \Rightarrow \text{zvec3}$ **where**

$$zword\text{-step } w \ v = \text{foldr } (\lambda p \ f. \ zletter\text{-step } p \circ f) \ w \ \text{id } v$$

definition $zword\text{-witness} :: (\text{bool} \times \text{gen2}) \text{ list} \Rightarrow \text{zvec3}$ **where**

$$zword\text{-witness } w = (\text{case } \text{snd } (\text{last } w) \text{ of } A \Rightarrow ze-y \ | \ B \Rightarrow ze-x)$$

datatype $r3 = R0 \ | \ R1 \ | \ R2$

type-synonym $rcoef = r3 \times r3$

type-synonym $rvec3 = rcoef \times rcoef \times rcoef$

fun $r3\text{-add} :: r3 \Rightarrow r3 \Rightarrow r3$ **where**

$$\begin{aligned} r3\text{-add } R0 \ x = & x \\ | \ r3\text{-add } x \ R0 = & x \\ | \ r3\text{-add } R1 \ R1 = & R2 \\ | \ r3\text{-add } R1 \ R2 = & R0 \end{aligned}$$

| $r3\text{-add } R2 \ R1 = R0$
| $r3\text{-add } R2 \ R2 = R1$

fun $r3\text{-neg} :: r3 \Rightarrow r3$ **where**
 $r3\text{-neg } R0 = R0$
| $r3\text{-neg } R1 = R2$
| $r3\text{-neg } R2 = R1$

definition $r3\text{-sub} :: r3 \Rightarrow r3 \Rightarrow r3$ **where**
 $r3\text{-sub } x \ y = r3\text{-add } x \ (r3\text{-neg } y)$

fun $r3\text{-double} :: r3 \Rightarrow r3$ **where**
 $r3\text{-double } R0 = R0$
| $r3\text{-double } R1 = R2$
| $r3\text{-double } R2 = R1$

definition $rcoef\text{-add} :: rcoef \Rightarrow rcoef \Rightarrow rcoef$ **where**
 $rcoef\text{-add } p \ q = (r3\text{-add } (fst \ p) \ (fst \ q), r3\text{-add } (snd \ p) \ (snd \ q))$

definition $rcoef\text{-neg} :: rcoef \Rightarrow rcoef$ **where**
 $rcoef\text{-neg } p = (r3\text{-neg } (fst \ p), r3\text{-neg } (snd \ p))$

definition $rcoef\text{-sub} :: rcoef \Rightarrow rcoef \Rightarrow rcoef$ **where**
 $rcoef\text{-sub } p \ q = rcoef\text{-add } p \ (rcoef\text{-neg } q)$

definition $rcoef\text{-double} :: rcoef \Rightarrow rcoef$ **where**
 $rcoef\text{-double } p = (r3\text{-double } (fst \ p), r3\text{-double } (snd \ p))$

definition $rcoef\text{-mul-sqrt2} :: rcoef \Rightarrow rcoef$ **where**
 $rcoef\text{-mul-sqrt2 } p = (r3\text{-double } (snd \ p), fst \ p)$

definition $rRx\text{-step} :: rvec3 \Rightarrow rvec3$ **where**
 $rRx\text{-step } v = (case \ v \ of \ (x, y, z) \Rightarrow$
 $((R0, R0),$
 $rcoef\text{-sub } y \ (rcoef\text{-double } (rcoef\text{-mul-sqrt2 } z)),$
 $rcoef\text{-add } (rcoef\text{-double } (rcoef\text{-mul-sqrt2 } y)) \ z))$

definition $rRx\text{-inv-step} :: rvec3 \Rightarrow rvec3$ **where**
 $rRx\text{-inv-step } v = (case \ v \ of \ (x, y, z) \Rightarrow$
 $((R0, R0),$
 $rcoef\text{-add } y \ (rcoef\text{-double } (rcoef\text{-mul-sqrt2 } z)),$
 $rcoef\text{-sub } z \ (rcoef\text{-double } (rcoef\text{-mul-sqrt2 } y))))$

definition $rRz\text{-step} :: rvec3 \Rightarrow rvec3$ **where**
 $rRz\text{-step } v = (case \ v \ of \ (x, y, z) \Rightarrow$
 $(rcoef\text{-sub } x \ (rcoef\text{-double } (rcoef\text{-mul-sqrt2 } y)),$
 $rcoef\text{-add } (rcoef\text{-double } (rcoef\text{-mul-sqrt2 } x)) \ y,$
 $(R0, R0)))$

definition $rRz\text{-inv}\text{-step} :: rvec3 \Rightarrow rvec3$ **where**

$$rRz\text{-inv}\text{-step } v = (\text{case } v \text{ of } (x, y, z) \Rightarrow \\ (rcoef\text{-add } x (rcoef\text{-double } (rcoef\text{-mul}\text{-sqrt2 } y)), \\ rcoef\text{-sub } y (rcoef\text{-double } (rcoef\text{-mul}\text{-sqrt2 } x)), \\ (R0, R0)))$$

definition $rletter\text{-step} :: bool \times gen2 \Rightarrow rvec3 \Rightarrow rvec3$ **where**

$$rletter\text{-step } p = (\text{case } p \text{ of} \\ (False, A) \Rightarrow rRx\text{-step} \\ | (True, A) \Rightarrow rRx\text{-inv}\text{-step} \\ | (False, B) \Rightarrow rRz\text{-step} \\ | (True, B) \Rightarrow rRz\text{-inv}\text{-step})$$

definition $rword\text{-step} :: (bool \times gen2) \text{ list} \Rightarrow rvec3 \Rightarrow rvec3$ **where**

$$rword\text{-step } w v = \text{foldr } (\lambda p f. rletter\text{-step } p \circ f) w \text{ id } v$$

definition $re\text{-x} :: rvec3$ **where**

$$re\text{-x} = ((R1, R0), (R0, R0), (R0, R0))$$

definition $re\text{-y} :: rvec3$ **where**

$$re\text{-y} = ((R0, R0), (R1, R0), (R0, R0))$$

definition $rword\text{-witness} :: (bool \times gen2) \text{ list} \Rightarrow rvec3$ **where**

$$rword\text{-witness } w = (\text{case } \text{snd } (\text{last } w) \text{ of } A \Rightarrow re\text{-y} \mid B \Rightarrow re\text{-x})$$

definition $rzero\text{-vec} :: rvec3$ **where**

$$rzero\text{-vec} = ((R0, R0), (R0, R0), (R0, R0))$$

definition $rstate :: bool \times gen2 \Rightarrow rvec3 \Rightarrow bool$ **where**

$$rstate p v \longleftrightarrow \\ v \in (\text{case } p \text{ of} \\ (False, A) \Rightarrow \{ \\ ((R0, R0), (R0, R1), (R1, R0)), \\ ((R0, R0), (R0, R2), (R2, R0)), \\ ((R0, R0), (R1, R0), (R0, R2)), \\ ((R0, R0), (R2, R0), (R0, R1))\} \\ | (True, A) \Rightarrow \{ \\ ((R0, R0), (R0, R1), (R2, R0)), \\ ((R0, R0), (R0, R2), (R1, R0)), \\ ((R0, R0), (R1, R0), (R0, R1)), \\ ((R0, R0), (R2, R0), (R0, R2))\} \\ | (False, B) \Rightarrow \{ \\ ((R0, R1), (R1, R0), (R0, R0)), \\ ((R0, R2), (R2, R0), (R0, R0)), \\ ((R1, R0), (R0, R2), (R0, R0)), \\ ((R2, R0), (R0, R1), (R0, R0))\} \\ | (True, B) \Rightarrow \{ \\ ((R0, R1), (R2, R0), (R0, R0)), \\ ((R0, R2), (R1, R0), (R0, R0)),$$

$$\begin{aligned} & ((R1, R0), (R0, R1), (R0, R0)), \\ & ((R2, R0), (R0, R2), (R0, R0)) \end{aligned}$$

lemma *rstate-single*:

rstate p (*rletter-step* p (*case snd* p of $A \Rightarrow re-y \mid B \Rightarrow re-x$))

by (*cases* p)

(*auto split*: *bool.splits gen2.splits*)

simp: *rstate-def rletter-step-def rRx-step-def rRx-inv-step-def*

rRz-step-def rRz-inv-step-def re-x-def re-y-def rcoef-add-def

rcoef-sub-def rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def)

lemma *rstate-step-FA-FA*:

rstate ($False, A$) $v \Longrightarrow rstate$ ($False, A$) (*rletter-step* ($False, A$) v)

by (*simp add*: *rstate-def rletter-step-def rRx-step-def rcoef-add-def rcoef-sub-def*

rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; *elim disjE*; *simp*)

lemma *rstate-step-FA-FB*:

rstate ($False, B$) $v \Longrightarrow rstate$ ($False, A$) (*rletter-step* ($False, A$) v)

by (*simp add*: *rstate-def rletter-step-def rRx-step-def rcoef-add-def rcoef-sub-def*

rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; *elim disjE*; *simp*)

lemma *rstate-step-FA-TB*:

rstate ($True, B$) $v \Longrightarrow rstate$ ($False, A$) (*rletter-step* ($False, A$) v)

by (*simp add*: *rstate-def rletter-step-def rRx-step-def rcoef-add-def rcoef-sub-def*

rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; *elim disjE*; *simp*)

lemma *rstate-step-TA-TA*:

rstate ($True, A$) $v \Longrightarrow rstate$ ($True, A$) (*rletter-step* ($True, A$) v)

by (*simp add*: *rstate-def rletter-step-def rRx-inv-step-def rcoef-add-def rcoef-sub-def*

rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; *elim disjE*; *simp*)

lemma *rstate-step-TA-FB*:

rstate ($False, B$) $v \Longrightarrow rstate$ ($True, A$) (*rletter-step* ($True, A$) v)

by (*simp add*: *rstate-def rletter-step-def rRx-inv-step-def rcoef-add-def rcoef-sub-def*

rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; *elim disjE*; *simp*)

lemma *rstate-step-TA-TB*:

rstate ($True, B$) $v \Longrightarrow rstate$ ($True, A$) (*rletter-step* ($True, A$) v)

by (*simp add*: *rstate-def rletter-step-def rRx-inv-step-def rcoef-add-def rcoef-sub-def*

rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; *elim disjE*; *simp*)

lemma *rstate-step-FB-FB*:

rstate ($False, B$) $v \Longrightarrow rstate$ ($False, B$) (*rletter-step* ($False, B$) v)

by (*simp add*: *rstate-def rletter-step-def rRz-step-def rcoef-add-def rcoef-sub-def*

rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; *elim disjE*; *simp*)

lemma *rstate-step-FB-FA*:

rstate ($False, A$) $v \Longrightarrow rstate$ ($False, B$) (*rletter-step* ($False, B$) v)

by (*simp add*: *rstate-def rletter-step-def rRz-step-def rcoef-add-def rcoef-sub-def*

rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; elim disjE; simp)

lemma *rstate-step-FB-TA:*

rstate (True, A) v \implies rstate (False, B) (rletter-step (False, B) v)
by (*simp add: rstate-def rletter-step-def rRz-step-def rcoef-add-def rcoef-sub-def rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; elim disjE; simp*)

lemma *rstate-step-TB-TB:*

rstate (True, B) v \implies rstate (True, B) (rletter-step (True, B) v)
by (*simp add: rstate-def rletter-step-def rRz-inv-step-def rcoef-add-def rcoef-sub-def rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; elim disjE; simp*)

lemma *rstate-step-TB-FA:*

rstate (False, A) v \implies rstate (True, B) (rletter-step (True, B) v)
by (*simp add: rstate-def rletter-step-def rRz-inv-step-def rcoef-add-def rcoef-sub-def rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; elim disjE; simp*)

lemma *rstate-step-TB-TA:*

rstate (True, A) v \implies rstate (True, B) (rletter-step (True, B) v)
by (*simp add: rstate-def rletter-step-def rRz-inv-step-def rcoef-add-def rcoef-sub-def rcoef-neg-def rcoef-double-def rcoef-mul-sqrt2-def r3-sub-def; elim disjE; simp*)

lemmas *rstate-step-cases =*

rstate-step-FA-FA rstate-step-FA-FB rstate-step-FA-TB
rstate-step-TA-TA rstate-step-TA-FB rstate-step-TA-TB
rstate-step-FB-FB rstate-step-FB-FA rstate-step-FB-TA
rstate-step-TB-TB rstate-step-TB-FA rstate-step-TB-TA

lemma *rstate-step:*

assumes *rstate q v and \neg canceling p q*
shows *rstate p (rletter-step p v)*
using *assms*
by (*cases p; cases q; cases fst p; cases snd p; cases fst q; cases snd q*)
(auto simp: canceling-def rstate-step-cases)

lemma *rstate-nonzero:*

assumes *rstate p v*
shows *v \neq rzero-vec*
using *assms*
by (*cases p*)
(auto split: bool.splits gen2.splits simp: rstate-def rzero-vec-def)

lemma *rword-step-Cons [simp]:*

rword-step (p # w) v = rletter-step p (rword-step w v)
by (*simp add: rword-step-def*)

lemma *zword-step-Cons [simp]:*

zword-step (p # w) v = zletter-step p (zword-step w v)
by (*simp add: zword-step-def*)

```

lemma rword-witness-Cons-nonempty [simp]:
  assumes  $w \neq []$ 
  shows  $rword-witness (p \# w) = rword-witness w$ 
  using assms by (cases w) (simp-all add: rword-witness-def)

lemma rword-step-witness-state:
  assumes  $w \in carrier F2$  and  $w \neq []$ 
  shows  $rstate (hd w) (rword-step w (rword-witness w))$ 
  using assms
proof (induct w)
  case Nil
  then show ?case
    by simp
next
  case (Cons p w)
  have tail:  $w \in carrier F2$  and ncan-tail:  $w = [] \vee \neg canceling p (hd w)$ 
    using Cons.prem1 by (auto dest: F2-ConsD)
  show ?case
  proof (cases w)
    case Nil
    then show ?thesis
      using rstate-single[of p] by (simp add: rword-step-def rword-witness-def)
    next
    case (Cons q ws)
    have  $rstate q (rword-step (q \# ws) (rword-witness (q \# ws)))$ 
      using Cons.hyps[OF tail] Cons by simp
    moreover have  $\neg canceling p q$ 
      using ncan-tail Cons by simp
    ultimately have  $rstate p (rletter-step p (rword-step (q \# ws) (rword-witness (q \# ws))))$ 
      by (rule rstate-step)
    then show ?thesis
      using Cons by simp
  qed
qed

definition r3-of-int ::  $int \Rightarrow r3$  where
  r3-of-int n = (if n mod 3 = 0 then R0 else if n mod 3 = 1 then R1 else R2)

definition rcoef-of-z ::  $zsqr2 \Rightarrow rcoef$  where
  rcoef-of-z p = (r3-of-int (fst p), r3-of-int (snd p))

definition rvec3-of-z ::  $zvec3 \Rightarrow rvec3$  where
  rvec3-of-z v = (case v of (x, y, z) \Rightarrow (rcoef-of-z x, rcoef-of-z y, rcoef-of-z z))

lemma r3-of-int-add:
   $r3-of-int (a + b) = r3-add (r3-of-int a) (r3-of-int b)$ 
  by (auto simp: r3-of-int-def split: if-splits; presburger)

```

lemma *r3-of-int-neg*:

$$r3\text{-of-int } (- a) = r3\text{-neg } (r3\text{-of-int } a)$$

by (*auto simp: r3-of-int-def split: if-splits; presburger*)

lemma *r3-of-int-sub*:

$$r3\text{-of-int } (a - b) = r3\text{-add } (r3\text{-of-int } a) (r3\text{-neg } (r3\text{-of-int } b))$$

by (*auto simp: r3-of-int-def split: if-splits; presburger*)

lemma *r3-of-int-double*:

$$r3\text{-of-int } (2 * a) = r3\text{-double } (r3\text{-of-int } a)$$

by (*auto simp: r3-of-int-def split: if-splits; presburger*)

lemma *r3-of-int-triple*:

$$r3\text{-of-int } (3 * a) = R0$$

by (*simp add: r3-of-int-def*)

lemma *rcoef-of-z-simps [simp]*:

$$rcoef\text{-of-z } (zsqrt2\text{-add } p q) = rcoef\text{-add } (rcoef\text{-of-z } p) (rcoef\text{-of-z } q)$$

$$rcoef\text{-of-z } (zsqrt2\text{-neg } p) = rcoef\text{-neg } (rcoef\text{-of-z } p)$$

$$rcoef\text{-of-z } (zsqrt2\text{-sub } p q) = rcoef\text{-sub } (rcoef\text{-of-z } p) (rcoef\text{-of-z } q)$$

$$rcoef\text{-of-z } (zsqrt2\text{-scale } 2 p) = rcoef\text{-double } (rcoef\text{-of-z } p)$$

$$rcoef\text{-of-z } (zsqrt2\text{-scale } 3 p) = (R0, R0)$$

$$rcoef\text{-of-z } (zsqrt2\text{-mul-sqrt2 } p) = rcoef\text{-mul-sqrt2 } (rcoef\text{-of-z } p)$$

by (*simp-all add: rcoef-of-z-def zsqrt2-add-def zsqrt2-neg-def zsqrt2-sub-def zsqrt2-scale-def zsqrt2-mul-sqrt2-def rcoef-add-def rcoef-neg-def rcoef-sub-def rcoef-double-def rcoef-mul-sqrt2-def r3-of-int-add r3-of-int-neg r3-of-int-sub r3-of-int-double r3-of-int-triple*)

lemma *rvec3-of-z-basis [simp]*:

$$rvec3\text{-of-z } ze\text{-x} = re\text{-x}$$

$$rvec3\text{-of-z } ze\text{-y} = re\text{-y}$$

by (*simp-all add: rvec3-of-z-def rcoef-of-z-def ze-x-def ze-y-def re-x-def re-y-def r3-of-int-def*)

lemma *rstep-of-z-simps [simp]*:

$$rvec3\text{-of-z } (zRx\text{-step } v) = rRx\text{-step } (rvec3\text{-of-z } v)$$

$$rvec3\text{-of-z } (zRx\text{-inv-step } v) = rRx\text{-inv-step } (rvec3\text{-of-z } v)$$

$$rvec3\text{-of-z } (zRz\text{-step } v) = rRz\text{-step } (rvec3\text{-of-z } v)$$

$$rvec3\text{-of-z } (zRz\text{-inv-step } v) = rRz\text{-inv-step } (rvec3\text{-of-z } v)$$

by (*cases v; simp add: rvec3-of-z-def zRx-step-def zRx-inv-step-def zRz-step-def zRz-inv-step-def rRx-step-def rRx-inv-step-def rRz-step-def rRz-inv-step-def*)⁺

lemma *rletter-step-of-z [simp]*:

$$rvec3\text{-of-z } (zletter\text{-step } p v) = rletter\text{-step } p (rvec3\text{-of-z } v)$$

by (*cases p*)

(*auto split: bool.splits gen2.splits simp: zletter-step-def rletter-step-def*)

lemma *rword-step-of-z [simp]*:

$rvec3\text{-of-z } (zword\text{-step } w \ v) = rword\text{-step } w \ (rvec3\text{-of-z } v)$
proof (*induct w arbitrary: v*)
case *Nil*
then show *?case*
by (*simp add: zword-step-def rword-step-def*)
next
case (*Cons p w*)
then show *?case*
by *simp*
qed

lemma *rword-witness-of-z [simp]:*
 $rvec3\text{-of-z } (zword\text{-witness } w) = rword\text{-witness } w$
by (*cases snd (last w)*)
(simp-all add: zword-witness-def rword-witness-def)

lemma *r3-of-int-eq-R0-iff:*
 $r3\text{-of-int } n = R0 \iff \exists \text{ dvd } n$
by (*simp add: r3-of-int-def dvd-eq-mod-eq-0*)

lemma *rcoef-of-z-zero-iff:*
 $rcoef\text{-of-z } p = (R0, R0) \iff zsqrt2\text{-div3 } p$
by (*cases p*) (*simp add: rcoef-of-z-def zsqrt2-div3-def r3-of-int-eq-R0-iff*)

lemma *rvec3-of-z-zero-iff:*
 $rvec3\text{-of-z } v = rzero\text{-vec} \iff zvec3\text{-div3 } v$
by (*cases v*) (*simp add: rvec3-of-z-def rzero-vec-def zvec3-div3-def rcoef-of-z-zero-iff*)

lemma *zsqrt2-val-simps [simp]:*
 $zsqrt2\text{-val } (zsqrt2\text{-add } p \ q) = zsqrt2\text{-val } p + zsqrt2\text{-val } q$
 $zsqrt2\text{-val } (zsqrt2\text{-neg } p) = - zsqrt2\text{-val } p$
 $zsqrt2\text{-val } (zsqrt2\text{-sub } p \ q) = zsqrt2\text{-val } p - zsqrt2\text{-val } q$
 $zsqrt2\text{-val } (zsqrt2\text{-scale } n \ p) = of\text{-int } n * zsqrt2\text{-val } p$
unfolding *zsqrt2-val-def zsqrt2-add-def zsqrt2-neg-def zsqrt2-sub-def zsqrt2-scale-def*
by (*simp-all add: algebra-simps*)

lemma *zsqrt2-mul-sqrt2-val [simp]:*
 $zsqrt2\text{-val } (zsqrt2\text{-mul-sqrt2 } p) = \text{sqrt } 2 * zsqrt2\text{-val } p$
unfolding *zsqrt2-val-def zsqrt2-mul-sqrt2-def*
by (*simp add: algebra-simps*)

lemma *sqrt-prime-irrational-dev:*
fixes *p :: nat*
assumes *prime p*
shows $\text{sqrt } p \notin \mathbb{Q}$
proof
from *assms have p: p > 1*
by (*rule prime-gt-1-nat*)
assume $\text{sqrt } p \in \mathbb{Q}$

```

then obtain m n :: nat
  where n: n ≠ 0
    and sqrt-rat: |sqrt p| = m / n
    and coprime m n
  by (rule Rats-abs-nat-div-natE)
have eq: m2 = p * n2
proof -
  from n sqrt-rat have m = |sqrt p| * n
    by simp
  then have m2 = (sqrt p)2 * n2
    by (simp add: power-mult-distrib)
  also have (sqrt p)2 = p
    by simp
  also have ... * n2 = p * n2
    by simp
  finally show ?thesis
    by linarith
qed
have p dvd m ∧ p dvd n
proof
  from eq have p dvd m2 ..
  with assms show p dvd m
    by (rule prime-dvd-power)
  then obtain k where m = p * k ..
  with eq have p * n2 = p2 * k2
    by algebra
  with p have n2 = p * k2
    by (simp add: power2-eq-square)
  then have p dvd n2 ..
  with assms show p dvd n
    by (rule prime-dvd-power)
qed
then have p dvd gcd m n
  by simp
with ⟨coprime m n⟩ have p = 1
  by simp
with p show False
  by simp
qed

lemma sqrt-2-not-rat-dev: sqrt 2 ∉ ℚ
  using sqrt-prime-irrational-dev[of 2] by simp

lemma zsqr2-val-eq-iff:
  zsqr2-val p = zsqr2-val q ⟷ p = q
proof
  assume eq: zsqr2-val p = zsqr2-val q
  obtain a b c d where defs: p = (a, b) q = (c, d)
    by (cases p; cases q) auto

```

```

let ?A = a - c
let ?B = b - d
from eq defs have eq0: of-int ?A + of-int ?B * sqrt 2 = (0::real)
  by (simp add: zsqrt2-val-def algebra-simps)
have ?B = 0
proof (rule ccontr)
  assume ?B ≠ 0
  hence sqrt 2 = - of-int ?A / of-int ?B
    using eq0 by (simp add: field-simps)
  moreover have - of-int ?A / of-int ?B ∈ ℚ
    by simp
  ultimately have sqrt 2 ∈ ℚ
    by simp
  with sqrt-2-not-rat-dev show False
    by simp
qed
moreover with eq0 have ?A = 0
  by simp
ultimately show p = q
  using defs by simp
qed simp

```

```

lemma zvec3-val-eq-iff:
  zvec3-val u = zvec3-val v ↔ u = v
  by (cases u; cases v)
  (auto simp: zvec3-val-def vector-def vec-eq-iff forall-3 zsqrt2-val-eq-iff)

```

```

lemma zvec3-scale-val [simp]:
  zvec3-val (zvec3-scale n v) = scaleR (of-int n) (zvec3-val v)
  by (cases v)
  (simp add: zvec3-scale-def zvec3-val-def vector-def vec-eq-iff forall-3)

```

```

lemma sqrt8-eq-2-sqrt2: sqrt 8 = 2 * sqrt 2
proof -
  have sqrt (8::real) = sqrt ((4::real) * 2)
    by simp
  also have ... = sqrt (4::real) * sqrt 2
    by (subst real-sqrt-mult) simp
  also have ... = 2 * sqrt 2
    by simp
  finally show ?thesis .
qed

```

```

lemma zletter-step-single-mod3:
  ¬ zvec3-div3 (zletter-step (False, A) ze-y)
  ¬ zvec3-div3 (zletter-step (True, A) ze-y)
  ¬ zvec3-div3 (zletter-step (False, B) ze-x)
  ¬ zvec3-div3 (zletter-step (True, B) ze-x)
by (simp-all add: zletter-step-def zRx-step-def zRx-inv-step-def zRz-step-def zRz-inv-step-def)

```

*ze-x-def ze-y-def zvec3-div3-def zsqr2-div3-def zsqr2-add-def zsqr2-sub-def
zsqr2-neg-def zsqr2-scale-def zsqr2-mul-sqr2-def)*

lemma *zRx-step-val:*

zvec3-val (zRx-step v) = scaleR 3 (Rx (zvec3-val v))

by *(cases v)*

(simp add: zvec3-val-def zRx-step-def Rx-def Rx-mat-def rot-c-def rot-s-def

*sqr2-eq-2-sqr2 matrix-vector-mult-def vector-def vec-eq-iff sum-3 forall-3 al-
gebra-simps)*

lemma *zRx-inv-step-val:*

zvec3-val (zRx-inv-step v) = scaleR 3 (Rx-inv (zvec3-val v))

by *(cases v)*

*(simp add: zvec3-val-def zRx-inv-step-def Rx-inv-def Rx-inv-mat-def rot-c-def
rot-s-def*

*sqr2-eq-2-sqr2 matrix-vector-mult-def vector-def vec-eq-iff sum-3 forall-3 al-
gebra-simps)*

lemma *zRz-step-val:*

zvec3-val (zRz-step v) = scaleR 3 (Rz (zvec3-val v))

by *(cases v)*

(simp add: zvec3-val-def zRz-step-def Rz-def Rz-mat-def rot-c-def rot-s-def

*sqr2-eq-2-sqr2 matrix-vector-mult-def vector-def vec-eq-iff sum-3 forall-3 al-
gebra-simps)*

lemma *zRz-inv-step-val:*

zvec3-val (zRz-inv-step v) = scaleR 3 (Rz-inv (zvec3-val v))

by *(cases v)*

*(simp add: zvec3-val-def zRz-inv-step-def Rz-inv-def Rz-inv-mat-def rot-c-def
rot-s-def*

*sqr2-eq-2-sqr2 matrix-vector-mult-def vector-def vec-eq-iff sum-3 forall-3 al-
gebra-simps)*

lemma *zletter-step-val:*

zvec3-val (zletter-step p v) = scaleR 3 (letter-to-rot p (zvec3-val v))

by *(cases p)*

(auto split: bool.splits gen2.splits

simp: zletter-step-def letter-to-rot-def zRx-step-val zRx-inv-step-val

zRz-step-val zRz-inv-step-val)

lemma *letter-to-rot-linear: linear (letter-to-rot p)*

unfolding *letter-to-rot-def Rx-def Rx-inv-def Rz-def Rz-inv-def*

by *(cases p)*

(simp split: bool.splits gen2.splits add: matrix-vector-mul-linear)

lemma *zword-step-val:*

zvec3-val (zword-step w v) =

scaleR ((3::real) ^ length w) (sigma w (zvec3-val v))

proof *(induct w arbitrary: v)*

```

case Nil
show ?case
  by (simp add: zword-step-def)
next
case (Cons p w)
have lin: linear (letter-to-rot p)
  by (rule letter-to-rot-linear)
have zvec3-val (zword-step (p # w) v) =
  zvec3-val (zletter-step p (zword-step w v))
  by (simp add: zword-step-def)
also have ... = scaleR 3 (letter-to-rot p (zvec3-val (zword-step w v)))
  by (rule zletter-step-val)
also have ... =
  scaleR 3 (letter-to-rot p
    (scaleR ((3::real) ^ length w) (sigma w (zvec3-val v))))
  using Cons.hyps by simp
also have ... =
  scaleR ((3::real) ^ Suc (length w)) ((letter-to-rot p ∘ sigma w) (zvec3-val v))
  using linear.scaleR[OF lin, of ((3::real) ^ length w) sigma w (zvec3-val v)]
  by (simp add: scaleR-scaleR mult-ac)
also have ... = scaleR ((3::real) ^ length (p # w)) (sigma (p # w) (zvec3-val
v))
  by simp
  finally show ?case .
qed

```

```

lemma zvec3-scale-power3-div3:
assumes n > 0
shows zvec3-div3 (zvec3-scale ((3::int) ^ n) v)
using assms
by (cases v)
  (auto simp: zvec3-div3-def zsqrt2-div3-def zvec3-scale-def zsqrt2-scale-def
  intro!: dvd-mult2)

```

```

lemma zword-step-identity-div3:
assumes w ≠ [] and sigma w = id
shows zvec3-div3 (zword-step w (zword-witness w))
proof –
have len-pos: length w > 0
  using assms(1) by (cases w) auto
have zvec3-val (zword-step w (zword-witness w)) =
  scaleR ((3::real) ^ length w) (zvec3-val (zword-witness w))
  using zword-step-val[of w zword-witness w] assms(2) by simp
also have ... = zvec3-val (zvec3-scale ((3::int) ^ length w) (zword-witness w))
  by simp
finally have zword-step w (zword-witness w) =
  zvec3-scale ((3::int) ^ length w) (zword-witness w)
  using zvec3-val-eq-iff by blast
thus ?thesis

```

using $\text{zvec3-scale-power3-div3}[OF \text{ len-pos, of zword-witness } w]$ by *simp*
qed

lemma *zword-step-witness-not-div3-if-nonempty-F2*:

assumes $w \in \text{carrier } F2$ **and** $w \neq []$

shows $\neg \text{zvec3-div3} (\text{zword-step } w (\text{zword-witness } w))$

proof

assume $\text{div3: zvec3-div3} (\text{zword-step } w (\text{zword-witness } w))$

have $\text{rstate} (\text{hd } w) (\text{rword-step } w (\text{rword-witness } w))$

using *assms* by (rule *rword-step-witness-state*)

hence $\text{rword-step } w (\text{rword-witness } w) \neq \text{rzero-vec}$

by (rule *rstate-nonzero*)

moreover from *div3* **have** $\text{rword-step } w (\text{rword-witness } w) = \text{rzero-vec}$

by (*simp add: rvec3-of-z-zero-iff[symmetric]*)

ultimately show *False*

by *contradiction*

qed

The word interpretation is an injective homomorphism into $SO(3)$.

theorem *sigma-homomorphism*:

assumes $w1 \in \text{carrier } F2$ $w2 \in \text{carrier } F2$

shows $\text{sigma} (w1 \otimes_{F2} w2) = \text{sigma } w1 \circ \text{sigma } w2$

by (*simp add: F2-mult*)

theorem *sigma-nontrivial-word-not-id*:

assumes $w \in \text{carrier } F2$ **and** $w \neq []$

shows $\text{sigma } w \neq \text{id}$

proof

assume $\text{sigma } w = \text{id}$

hence $\text{zvec3-div3} (\text{zword-step } w (\text{zword-witness } w))$

using *assms*(2) by (*intro zword-step-identity-div3*)

moreover have $\neg \text{zvec3-div3} (\text{zword-step } w (\text{zword-witness } w))$

using *assms* by (rule *zword-step-witness-not-div3-if-nonempty-F2*)

ultimately show *False*

by *simp*

qed

lemma *sigma-injective-if-trivial-kernel*:

assumes *kernel*: $\bigwedge w. w \in \text{carrier } F2 \implies w \neq [] \implies \text{sigma } w \neq \text{id}$

shows *inj-on* $\text{sigma} (\text{carrier } F2)$

proof (rule *inj-onI*)

interpret *F*: *group* $F2$

by (rule *F2-is-group*)

fix $w \ v$

assume $w: w \in \text{carrier } F2$ **and** $v: v \in \text{carrier } F2$ **and** *eq*: $\text{sigma } w = \text{sigma } v$

let $?iv = \text{inv}_{F2} \ v$

have *iv*: $?iv \in \text{carrier } F2$

using *v* by (rule *F.inv-closed*)

have *prod*: $w \otimes_{F2} ?iv \in \text{carrier } F2$

```

    using w iv by (rule F.m-closed)
  have sigma (w ⊗F2 ?iv) = sigma w ∘ sigma ?iv
    using w iv by (rule sigma-homomorphism)
  also have ... = sigma v ∘ sigma ?iv
    using eq by simp
  also have ... = sigma (v ⊗F2 ?iv)
    using v iv by (rule sigma-homomorphism[symmetric])
  also have ... = sigma []
    using F.r-inv[OF v] by simp
  also have ... = id
    by simp
  finally have prod-id: sigma (w ⊗F2 ?iv) = id .
  have prod-one: w ⊗F2 ?iv = []
  proof (rule ccontr)
    assume w ⊗F2 ?iv ≠ []
    with kernel[OF prod] prod-id show False
      by simp
  qed
  have w = w ⊗F2 1F2
    using w by simp
  moreover have ?iv ⊗F2 v = 1F2
    using v by (rule F.l-inv)
  ultimately have w = w ⊗F2 (?iv ⊗F2 v)
    by simp
  also have ... = (w ⊗F2 ?iv) ⊗F2 v
    using w iv v by (simp add: F.m-assoc)
  also have ... = v
    using prod-one v by simp
  finally show w = v .
qed

theorem sigma-injective-on-F2:
  inj-on sigma (carrier F2)
  by (rule sigma-injective-if-trivial-kernel) (rule sigma-nontrivial-word-not-id)

theorem sigma-image-in-SO3:
  assumes w ∈ carrier F2
  shows sigma w ∈ SO3
  by (rule sigma-in-SO3)

```

These three theorems together establish that *sigma* is a group monomorphism from F_2 to $SO(3)$ – i.e., the image is a copy of F_2 inside $SO(3)$. This supplies the free subgroup used in the geometric part of the Banach-Tarski theorem.

end

```

theory Hausdorff-Paradox
  imports Free-Rotations-SO3

```

begin

23 The bad set of fixed points

definition *fixed-point-set* :: $(\text{real}^3 \Rightarrow \text{real}^3) \Rightarrow (\text{real}^3)$ set **where**
fixed-point-set $f = \{x \in \text{sphere2}. f\ x = x\}$

definition *bad-set-D* :: (real^3) set **where**
bad-set-D = $(\bigcup w \in \text{carrier } F2 - \{\emptyset\}. \text{fixed-point-set } (\text{sigma } w))$

lemma *carrier-F2-countable*: countable (carrier F2)

proof (rule countable-subset)

show $\text{carrier } F2 \subseteq (\text{UNIV} :: ((\text{bool} \times \text{gen2}) \text{ list}) \text{ set})$

by simp

show countable (UNIV :: $((\text{bool} \times \text{gen2}) \text{ list}) \text{ set}$)

proof –

have *gen2-finite*: finite (UNIV :: gen2 set)

using *Gen2-finite Gen2-UNIV* **by** simp

have *alphabet-finite*: finite $((\text{UNIV} :: \text{bool set}) \times (\text{UNIV} :: \text{gen2 set}))$

by (rule finite-cartesian-product) (simp-all add: *gen2-finite*)

have *alphabet-countable*: countable (UNIV :: $(\text{bool} \times \text{gen2}) \text{ set}$)

by (rule countable-finite) (use *alphabet-finite* **in** $\langle \text{simp only: UNIV-Times-UNIV} \rangle$)

have countable (lists (UNIV :: $(\text{bool} \times \text{gen2}) \text{ set}$))

by (rule countable-lists) (rule *alphabet-countable*)

then show *?thesis* **by** simp

qed

qed

lemma *bad-set-D-index-countable*: countable (carrier F2 - $\{\emptyset\}$)
using *carrier-F2-countable* **by** simp

lemma *fixed-point-set-subset-sphere2*: *fixed-point-set* $f \subseteq \text{sphere2}$
by (simp add: *fixed-point-set-def*)

lemma *SO3-fixed-cross*:

assumes $f \in \text{SO3}$ **and** $f\ x = x$ **and** $f\ y = y$

shows $f\ (\text{cross3 } x\ y) = \text{cross3 } x\ y$

proof –

have *f-orth*: orthogonal-transformation f

using *assms(1)* **unfolding** *SO3-def rotation-def* **by** auto

have *f-det*: $\text{matrix-det } (\text{matrix } f) = 1$

using *assms(1)* **unfolding** *SO3-def rotation-def* **by** auto

have $\text{cross3 } (f\ x)\ (f\ y) = \text{matrix-det } (\text{matrix } f) *_R f\ (\text{cross3 } x\ y)$

using *cross-orthogonal-transformation*[*OF f-orth, of x y*] .

with *assms(2,3)* *f-det* **show** *?thesis*

by simp

qed

lemma *SO3-fixed-cross-nonzero-imp-id*:

assumes $f \in SO3$ and $f x = x$ and $f y = y$ and $cross3 x y \neq 0$
shows $f = id$

proof –

let $?n = cross3 x y$
let $?u = y - ((x \cdot y) / (x \cdot x)) *R x$
let $?S = \{x, ?u, ?n\}$
have $f-orth$: orthogonal-transformation f
 using $assms(1)$ **unfolding** $SO3-def$ $rotation-def$ **by** $auto$
have $f-lin$: linear f
 by (rule $orthogonal-transformation-linear[OF f-orth]$)
have $x-ne$: $x \neq 0$
 using $assms(4)$ **by** $auto$
have $xx-ne$: $x \cdot x \neq 0$
 using $x-ne$ **by** $simp$
have $n-fix$: $f ?n = ?n$
 by (rule $SO3-fixed-cross[OF assms(1,2,3)]$)
have $u-fix$: $f ?u = ?u$
 using $assms(2,3)$ $f-lin$
 by ($simp$ add: $linear-diff$ $linear-scale$)
have $cross-x-u$: $cross3 x ?u = ?n$
 by ($simp$ add: $Cross3.right-diff-distrib$ $cross-mult-right$)
have $u-ne$: $?u \neq 0$

proof

 assume $?u = 0$
 hence $cross3 x ?u = 0$
 by $simp$
 with $cross-x-u$ $assms(4)$ **show** $False$
 by $simp$

qed

have $x-u-orth$: orthogonal $x ?u$
 using $xx-ne$ **by** ($simp$ add: $orthogonal-def$ $inner-diff-right$ $inner-commute$)
have $n-x-orth$: orthogonal $?n x$
 by (rule $orthogonal-cross$)
have $n-y-orth$: orthogonal $?n y$
 by (rule $orthogonal-cross$)
have $n-u-orth$: orthogonal $?n ?u$
 using $n-x-orth$ $n-y-orth$
 by ($simp$ add: $orthogonal-def$ $inner-diff-right$)
have $pairwise-S$: pairwise orthogonal $?S$
 using $x-u-orth$ $n-x-orth$ $n-u-orth$
 by ($auto$ $simp$: $pairwise-def$ $orthogonal-commute$)
have $zero-notin-S$: $0 \notin ?S$
 using $assms(4)$ $x-ne$ $u-ne$ **by** $auto$
have $ind-S$: independent $?S$
 by (rule $pairwise-orthogonal-independent[OF pairwise-S zero-notin-S]$)
have $x-neq-u$: $x \neq ?u$

proof

 assume $x = ?u$
 hence $orthogonal x x$

```

    using x-u-orth by simp
  with x-ne show False
    by (simp add: orthogonal-def)
qed
have x-neq-n:  $x \neq ?n$ 
proof
  assume  $x = ?n$ 
  hence orthogonal x x
    using n-x-orth by (simp add: orthogonal-commute)
  with x-ne show False
    by (simp add: orthogonal-def)
qed
have u-neq-n:  $?u \neq ?n$ 
proof
  assume  $?u = ?n$ 
  hence orthogonal ?u ?u
    using n-u-orth by (simp add: orthogonal-commute)
  with u-ne show False
    by (simp add: orthogonal-def)
qed
have card-S:  $\text{card } ?S = \text{CARD}(3)$ 
  using x-neq-u x-neq-n u-neq-n by simp
have dim-S:  $\text{dim } ?S = \text{CARD}(3)$ 
  using ind-S card-S by (simp add: dim-eq-card-independent)
have span-S:  $\text{span } ?S = \text{UNIV}$ 
proof (rule iffD1[OF dim-eq-full])
  show  $\text{dim } ?S = \text{DIM}(\text{real}^3)$ 
    using dim-S by simp
qed
have fixed-on-S:  $\bigwedge z. z \in ?S \implies fz = \text{id } z$ 
  using assms(2) u-fix n-fix by auto
have fixed-on-span:  $\bigwedge z. z \in \text{span } ?S \implies fz = \text{id } z$ 
  by (rule linear-eq-on-span[OF f-lin linear-id fixed-on-S])
show ?thesis
proof
  fix  $z$ 
  have  $z \in \text{span } ?S$ 
    using span-S by simp
  thus  $fz = \text{id } z$ 
    by (rule fixed-on-span)
qed
qed

lemma nonidentity-SO3-fixed-points-cross-zero:
  assumes  $f \in \text{SO}3$  and  $f \neq \text{id}$  and  $x \in \text{fixed-point-set } f$  and  $y \in \text{fixed-point-set } f$ 
  shows  $\text{cross}3 \ x \ y = 0$ 
proof (rule ccontr)
  assume cross-ne:  $\text{cross}3 \ x \ y \neq 0$ 

```

```

have fx: f x = x and fy: f y = y
  using assms(3,4) by (simp-all add: fixed-point-set-def)
have f = id
  by (rule SO3-fixed-cross-nonzero-imp-id[OF assms(1) fx fy cross-ne])
with assms(2) show False
  by simp
qed

```

```

lemma collinear-sphere2-two-points:
  assumes a ∈ sphere2 and y ∈ sphere2 and cross3 a y = 0
  shows y = a ∨ y = - a
proof -
  have cross-col: cross3 a y = 0 ⟷ collinear {0, a, y}
    by (rule cross-eq-0)
  have norm-a: norm a = 1
    using assms(1) by (simp add: sphere2-def)
  have norm-y: norm y = 1
    using assms(2) by (simp add: sphere2-def)
  have a-ne: a ≠ 0
  proof
    assume a = 0
    with norm-a show False
      by simp
  qed
  have y-ne: y ≠ 0
  proof
    assume y = 0
    with norm-y show False
      by simp
  qed
  have col: collinear {0, a, y}
    using assms(3) cross-col by blast
  have cases: a = 0 ∨ y = 0 ∨ (∃ c. y = c *R a)
    using iffD1[OF collinear-lemma[of a y] col] .
  then obtain c where y-eq: y = c *R a
    using a-ne y-ne by blast
  have |c| = 1
    using norm-y by (simp add: y-eq norm-a)
  hence c-cases: c = 1 ∨ c = -1
  proof (cases c ≥ 0)
    case True
      with ⟨|c| = 1⟩ have c = 1
        by simp
      thus ?thesis ..
    next
      case False
      with ⟨|c| = 1⟩ have c = -1
        by simp
      thus ?thesis ..
  qed

```

qed
show *?thesis*
using *c-cases y-eq* **by** *auto*
qed

lemma *bad-set-D-countable-if-fixed-point-sets-countable:*
assumes $\bigwedge w. w \in \text{carrier } F2 - \{\emptyset\} \implies \text{countable } (\text{fixed-point-set } (\text{sigma } w))$
shows *countable bad-set-D*
proof $-$
have *countable* $(\bigcup w \in \text{carrier } F2 - \{\emptyset\}. \text{fixed-point-set } (\text{sigma } w))$
by *(rule countable-UN) (use bad-set-D-index-countable assms in auto)*
thus *?thesis*
by *(simp add: bad-set-D-def)*
qed

lemma *nonidentity-SO3-fixed-point-set-countable:*
assumes $f \in \text{SO3}$ **and** $f \neq \text{id}$
shows *countable (fixed-point-set f)*
proof *(cases fixed-point-set f = {})*
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then obtain *a* **where** $a \in \text{fixed-point-set } f$
by *auto*
hence *a-sphere: a ∈ sphere2*
by *(simp add: fixed-point-set-def)*
have *subset-two: fixed-point-set f ⊆ {a, - a}*
proof
fix *y*
assume $y \in \text{fixed-point-set } f$
hence *y-sphere: y ∈ sphere2*
by *(simp add: fixed-point-set-def)*
have *cross-zero: cross3 a y = 0*
by *(rule nonidentity-SO3-fixed-points-cross-zero[OF assms a y])*
have $y = a \vee y = - a$
by *(rule collinear-sphere2-two-points[OF a-sphere y-sphere cross-zero])*
thus $y \in \{a, - a\}$
by *simp*
qed
have *finite (fixed-point-set f)*
by *(rule finite-subset[OF subset-two]) simp*
thus *?thesis*
by *(rule countable-finite)*
qed

lemma *sigma-fixed-point-set-countable:*
assumes $w \in \text{carrier } F2 - \{\emptyset\}$
shows *countable (fixed-point-set (sigma w))*

proof –

have $\sigma w \in SO3$
using *assms sigma-image-in-SO3* **by** *auto*
moreover have $\sigma w \neq id$
using *assms sigma-nontrivial-word-not-id* **by** *auto*
ultimately show *?thesis*
by (*rule nonidentity-SO3-fixed-point-set-countable*)
qed

lemma *bad-set-D-countable: countable bad-set-D*
by (*rule bad-set-D-countable-if-fixed-point-sets-countable*)
(*rule sigma-fixed-point-set-countable*)

lemma *bad-set-D-subset: bad-set-D \subseteq sphere2*
unfolding *bad-set-D-def fixed-point-set-def* **by** *auto*

lemma *sphere2-minus-bad-subset: sphere2 – bad-set-D \subseteq sphere2*
by *auto*

interpretation *sigma-sphere-action:*

group-action carrier F2 \square ($\lambda w1 w2. w1 \otimes_{F2} w2$) $\lambda w x. \sigma w x$ sphere2

proof

show $\square \in \text{carrier } F2$

by *simp*

show $\bigwedge g h. g \in \text{carrier } F2 \implies h \in \text{carrier } F2 \implies g \otimes_{F2} h \in \text{carrier } F2$

using *group.subgroup-self group-def monoid.m-closed free-group-is-group* **by**

blast

show $\bigwedge g x. g \in \text{carrier } F2 \implies x \in \text{sphere2} \implies \sigma g x \in \text{sphere2}$

using *sigma-preserves-sphere2* **by** *blast*

show $\bigwedge x. x \in \text{sphere2} \implies \sigma \square x = x$

by *simp*

show $\bigwedge g h x. g \in \text{carrier } F2 \implies h \in \text{carrier } F2 \implies x \in \text{sphere2} \implies$

$\sigma (g \otimes_{F2} h) x = \sigma g (\sigma h x)$

using *sigma-homomorphism* **by** (*simp add: o-def*)

qed

lemma *F2-conjugate-nontrivial:*

assumes $w \in \text{carrier } F2 \ v \in \text{carrier } F2 \ v \neq \square$

shows $(\text{inv}_{F2} w \otimes_{F2} v) \otimes_{F2} w \neq \square$

proof

interpret $G: \text{group } F2$

by (*rule free-group-is-group*)

assume *conj-eq: ($\text{inv}_{F2} w \otimes_{F2} v$) $\otimes_{F2} w = \square$*

have *inv-closed: $\text{inv}_{F2} w \in \text{carrier } F2$*

using *assms(1)* **by** (*rule G.inv-closed*)

have *vw-closed: $v \otimes_{F2} w \in \text{carrier } F2$*

by (*rule G.m-closed*) (*use assms in simp-all*)

have *assoc-eq:*

$\text{inv}_{F2} w \otimes_{F2} (v \otimes_{F2} w) =$

$(\text{inv}_{F_2} w \otimes_{F_2} v) \otimes_{F_2} w$
using *inv-closed assms* **by** (*simp add: G.m-assoc*)
have *eq1*: $\text{inv}_{F_2} w \otimes_{F_2} (v \otimes_{F_2} w) = []$
using *conj-eq assoc-eq* **by** *simp*
have $v \otimes_{F_2} w = w \otimes_{F_2} []$
using *G.inv-solve-left* [*OF G.one-closed assms(1) vw-closed*] *eq1* **by** *simp*
hence $v \otimes_{F_2} w = w$
using *assms* **by** *simp*
hence $v = []$
using *assms* **by** *simp*
with *assms(3)* **show** *False* **by** *simp*
qed

lemma *sigma-inverse-left*:
assumes $w \in \text{carrier } F_2$
shows $\text{sigma} (\text{inv}_{F_2} w) (\text{sigma } w x) = x$
proof –
interpret *G*: *group F2*
by (*rule free-group-is-group*)
have *inv-closed*: $\text{inv}_{F_2} w \in \text{carrier } F_2$
using *assms* **by** (*rule G.inv-closed*)
have $\text{sigma} (\text{inv}_{F_2} w \otimes_{F_2} w) x = \text{sigma} (\text{inv}_{F_2} w) (\text{sigma } w x)$
using *sigma-homomorphism* [*OF inv-closed assms*]
by (*simp add: o-def*)
moreover **have** $\text{inv}_{F_2} w \otimes_{F_2} w = []$
using *G.l-inv* [*OF assms*] **by** *simp*
ultimately show *?thesis*
by *simp*
qed

lemma *sigma-conjugate-fixed-pullback*:
assumes $w \in \text{carrier } F_2$
and $v \in \text{carrier } F_2$
and $\text{sigma } v (\text{sigma } w x) = \text{sigma } w x$
shows $\text{sigma} ((\text{inv}_{F_2} w \otimes_{F_2} v) \otimes_{F_2} w) x = x$
proof –
interpret *G*: *group F2*
by (*rule free-group-is-group*)
have *inv-closed*: $\text{inv}_{F_2} w \in \text{carrier } F_2$
using *assms(1)* **by** (*rule G.inv-closed*)
have *iv-closed*: $\text{inv}_{F_2} w \otimes_{F_2} v \in \text{carrier } F_2$
using *inv-closed assms(2)* **by** (*rule G.m-closed*)
have $\text{sigma} ((\text{inv}_{F_2} w \otimes_{F_2} v) \otimes_{F_2} w) x =$
 $\text{sigma} (\text{inv}_{F_2} w \otimes_{F_2} v) (\text{sigma } w x)$
using *sigma-homomorphism* [*OF iv-closed assms(1)*] **by** (*simp add: o-def*)
also have $\dots = \text{sigma} (\text{inv}_{F_2} w) (\text{sigma } v (\text{sigma } w x))$
using *sigma-homomorphism* [*OF inv-closed assms(2)*] **by** (*simp add: o-def*)
also have $\dots = \text{sigma} (\text{inv}_{F_2} w) (\text{sigma } w x)$
using *assms(3)* **by** *simp*

also have $\dots = x$
using *sigma-inverse-left*[*OF assms(1)*] **by** *simp*
finally show *?thesis* .
qed

lemma *sigma-preimage-bad-set*:
assumes $w \in \text{carrier } F2$
and $x \in \text{sphere2}$
and $\text{sigma } w \ x \in \text{bad-set-}D$
shows $x \in \text{bad-set-}D$
proof –
from *assms(3)* **obtain** v **where** $v\text{-carrier}: v \in \text{carrier } F2$
and $v\text{-ne}: v \neq []$
and *fixed*: $\text{sigma } v \ (\text{sigma } w \ x) = \text{sigma } w \ x$
unfolding *bad-set-D-def fixed-point-set-def* **by** *auto*
define u **where** $u = (\text{inv}_{F2} \ w \ \otimes_{F2} \ v) \ \otimes_{F2} \ w$
have $u\text{-carrier}: u \in \text{carrier } F2$
proof –
interpret G : *group* $F2$
by (*rule free-group-is-group*)
have $\text{inv-closed}: \text{inv}_{F2} \ w \in \text{carrier } F2$
using *assms(1)* **by** (*rule G.inv-closed*)
have $\text{inv}_{F2} \ w \ \otimes_{F2} \ v \in \text{carrier } F2$
using *inv-closed v-carrier* **by** (*rule G.m-closed*)
thus *?thesis*
unfolding *u-def* **using** *assms(1)* **by** (*rule G.m-closed*)
qed
have $u\text{-ne}: u \neq []$
using *F2-conjugate-nontrivial*[*OF assms(1) v-carrier v-ne*] **by** (*simp add: u-def*)
have $\text{sigma } u \ x = x$
using *sigma-conjugate-fixed-pullback*[*OF assms(1) v-carrier fixed*] **by** (*simp add: u-def*)
hence $x \in \text{fixed-point-set } (\text{sigma } u)$
using *assms(2)* **by** (*simp add: fixed-point-set-def*)
with $u\text{-carrier } u\text{-ne}$ **show** *?thesis*
unfolding *bad-set-D-def* **by** *blast*
qed

lemma *sigma-preserves-sphere2-minus-bad-set*:
assumes $w \in \text{carrier } F2$
and $x \in \text{sphere2} - \text{bad-set-}D$
shows $\text{sigma } w \ x \in \text{sphere2} - \text{bad-set-}D$
proof
show $\text{sigma } w \ x \in \text{sphere2}$
using *assms sigma-preserves-sphere2* **by** *blast*
show $\text{sigma } w \ x \notin \text{bad-set-}D$
proof
assume $\text{sigma } w \ x \in \text{bad-set-}D$
hence $x \in \text{bad-set-}D$

using *assms sigma-preimage-bad-set* **by** *auto*
with *assms(2)* **show** *False* **by** *simp*
qed
qed

interpretation *sigma-sphere-minus-bad-action:*

group-action carrier F2 \square $(\lambda w1 w2. w1 \otimes_{F2} w2) \lambda w x. \text{sigma } w x$
sphere2 - bad-set-D

proof

show $\square \in \text{carrier } F2$

by *simp*

show $\bigwedge g h. g \in \text{carrier } F2 \implies h \in \text{carrier } F2 \implies g \otimes_{F2} h \in \text{carrier } F2$

using *group.subgroup-self group-def monoid.m-closed free-group-is-group* **by**

blast

show $\bigwedge g x. g \in \text{carrier } F2 \implies x \in \text{sphere2} - \text{bad-set-D} \implies$

sigma g x $\in \text{sphere2} - \text{bad-set-D}$

by *(rule sigma-preserves-sphere2-minus-bad-set)*

show $\bigwedge x. x \in \text{sphere2} - \text{bad-set-D} \implies \text{sigma } \square x = x$

by *simp*

show $\bigwedge g h x. g \in \text{carrier } F2 \implies h \in \text{carrier } F2 \implies x \in \text{sphere2} - \text{bad-set-D}$

\implies

sigma (g \otimes_{F2} h) x = sigma g (sigma h x)

using *sigma-homomorphism* **by** *(simp add: o-def)*

qed

24 Free action of F_2 on $S^2 \setminus D$

By construction, every point of $S^2 \setminus D$ avoids the fixed-point set of every non-trivial group element, hence is moved by every non-trivial *sigma w*.

The action is free away from the fixed-point bad set.

lemma *sigma-action-free-off-D:*

assumes $x \in \text{sphere2} - \text{bad-set-D}$

and $w \in \text{carrier } F2$

and $\text{sigma } w x = x$

shows $w = \square$

proof *(rule ccontr)*

assume *w-ne: w \neq \square*

from *assms(1)* **have** *x-sphere: x \in sphere2* **and** *x-notbad: x \notin bad-set-D*

by *auto*

from *x-sphere assms(3)* **have** $x \in \text{fixed-point-set } (\text{sigma } w)$

by *(simp add: fixed-point-set-def)*

with *assms(2) w-ne* **have** $x \in \text{bad-set-D}$

unfolding *bad-set-D-def* **by** *blast*

with *x-notbad* **show** *False ..*

qed

lemma *sigma-sphere-minus-bad-action-free:*

sigma-sphere-minus-bad-action.free-on (*sphere2* - *bad-set-D*)
unfolding *sigma-sphere-minus-bad-action.free-on-def*
using *sigma-action-free-off-D* **by** *auto*

lemma *sigma-orbit-eqI*:

assumes *x*: $x \in \textit{sphere2} - \textit{bad-set-D}$
and *y*: $y \in \textit{sphere2} - \textit{bad-set-D}$
and *g*: $g \in \textit{carrier } F2$
and *y-eq*: $y = \textit{sigma } g x$
shows *sigma-sphere-minus-bad-action.orbit y* =
sigma-sphere-minus-bad-action.orbit x

proof

interpret *G*: *group F2*

by (*rule free-group-is-group*)

show *sigma-sphere-minus-bad-action.orbit y* \subseteq
sigma-sphere-minus-bad-action.orbit x

proof

fix *z*

assume $z \in \textit{sigma-sphere-minus-bad-action.orbit } y$

then obtain *h* **where** $h: h \in \textit{carrier } F2$ **and** $z: z = \textit{sigma } h y$

unfolding *sigma-sphere-minus-bad-action.orbit-def* **by** *blast*

have *hg*: $h \otimes_{F2} g \in \textit{carrier } F2$

by (*rule G.m-closed[OF h g]*)

have $z = \textit{sigma } (h \otimes_{F2} g) x$

using *z y-eq sigma-homomorphism[OF h g]* **by** (*simp add: o-def*)

thus $z \in \textit{sigma-sphere-minus-bad-action.orbit } x$

unfolding *sigma-sphere-minus-bad-action.orbit-def* **using** *hg* **by** *blast*

qed

show *sigma-sphere-minus-bad-action.orbit x* \subseteq
sigma-sphere-minus-bad-action.orbit y

proof

fix *z*

assume $z \in \textit{sigma-sphere-minus-bad-action.orbit } x$

then obtain *h* **where** $h: h \in \textit{carrier } F2$ **and** $z: z = \textit{sigma } h x$

unfolding *sigma-sphere-minus-bad-action.orbit-def* **by** *blast*

have *invg*: $\textit{inv}_{F2} g \in \textit{carrier } F2$

by (*rule G.inv-closed[OF g]*)

have *h-invg*: $h \otimes_{F2} \textit{inv}_{F2} g \in \textit{carrier } F2$

by (*rule G.m-closed[OF h invg]*)

have *x-eq*: $x = \textit{sigma } (\textit{inv}_{F2} g) y$

using *y-eq sigma-inverse-left[OF g]* **by** *simp*

have $z = \textit{sigma } (h \otimes_{F2} \textit{inv}_{F2} g) y$

using *z x-eq sigma-homomorphism[OF h invg]* **by** (*simp add: o-def*)

thus $z \in \textit{sigma-sphere-minus-bad-action.orbit } y$

unfolding *sigma-sphere-minus-bad-action.orbit-def* **using** *h-invg* **by** *blast*

qed

qed

lemma *sigma-orbit-eq-if-common-point*:

assumes $x: x \in \text{sphere2} - \text{bad-set-D}$
and $y: y \in \text{sphere2} - \text{bad-set-D}$
and $z: z \in \text{sigma-sphere-minus-bad-action.orbit } x$
and $z': z \in \text{sigma-sphere-minus-bad-action.orbit } y$
shows $\text{sigma-sphere-minus-bad-action.orbit } x =$
 $\text{sigma-sphere-minus-bad-action.orbit } y$
proof –
from z **obtain** g **where** $g: g \in \text{carrier } F2$ **and** $z\text{-eq}: z = \text{sigma } g \ x$
unfolding $\text{sigma-sphere-minus-bad-action.orbit-def}$ **by** blast
have $zX: z \in \text{sphere2} - \text{bad-set-D}$
using $\text{sigma-preserves-sphere2-minus-bad-set}[OF \ g \ x] \ z\text{-eq}$ **by** simp
have $\text{orbit-z-x}: \text{sigma-sphere-minus-bad-action.orbit } z =$
 $\text{sigma-sphere-minus-bad-action.orbit } x$
using $\text{sigma-orbit-eqI}[OF \ x \ zX \ g \ z\text{-eq}]$.
from z' **obtain** h **where** $h: h \in \text{carrier } F2$ **and** $z\text{-eq-y}: z = \text{sigma } h \ y$
unfolding $\text{sigma-sphere-minus-bad-action.orbit-def}$ **by** blast
have $\text{orbit-z-y}: \text{sigma-sphere-minus-bad-action.orbit } z =$
 $\text{sigma-sphere-minus-bad-action.orbit } y$
using $\text{sigma-orbit-eqI}[OF \ y \ zX \ h \ z\text{-eq-y}]$.
show $?thesis$
using $\text{orbit-z-x orbit-z-y}$ **by** simp
qed

25 Transporting the free-group paradox to $S^2 \setminus D$

definition $\text{hausdorff-rep} :: \text{real}^3 \Rightarrow \text{real}^3$ **where**
 $\text{hausdorff-rep } x = (\text{SOME } y. y \in \text{sigma-sphere-minus-bad-action.orbit } x)$

definition $\text{hausdorff-lift} :: ((\text{bool} \times \text{gen2}) \text{ list}) \text{ set} \Rightarrow (\text{real}^3) \text{ set}$ **where**
 $\text{hausdorff-lift } u =$
 $\{\text{sigma } g \ (\text{hausdorff-rep } x) \mid g \ x. g \in u \wedge x \in \text{sphere2} - \text{bad-set-D}\}$

definition $\text{hausdorff-sigma-image-set} ::$
 $(\text{bool} \times \text{gen2}) \text{ list} \Rightarrow (\text{real}^3) \text{ set} \Rightarrow (\text{real}^3) \text{ set}$ **where**
 $\text{hausdorff-sigma-image-set } g \ u = \text{sigma } g \ ' u$

lemma $\text{map2-hausdorff-sigma-image-set-eq}:$
 $\text{map2 } \text{sigma-sphere-minus-bad-action.image-set } g \ s \ A \ s =$
 $\text{map2 } \text{hausdorff-sigma-image-set } g \ s \ A \ s$
by $(\text{induction } g \ s \ \text{arbitrary}: A \ s)$
 $(\text{auto simp: hausdorff-sigma-image-set-def sigma-sphere-minus-bad-action.image-set-def}$
 $\text{split: list.splits})$

lemma $\text{hausdorff-rep-in-orbit}:$
assumes $x \in \text{sphere2} - \text{bad-set-D}$
shows $\text{hausdorff-rep } x \in \text{sigma-sphere-minus-bad-action.orbit } x$
unfolding hausdorff-rep-def
by (rule someI-ex) $(\text{use sigma-sphere-minus-bad-action.orbit-self}[OF \ \text{assms}] \text{ in}$
 $\text{blast})$

lemma *hausdorff-rep-in-X*:
assumes $x \in \text{sphere2} - \text{bad-set-D}$
shows $\text{hausdorff-rep } x \in \text{sphere2} - \text{bad-set-D}$
proof –
from *hausdorff-rep-in-orbit*[*OF assms*] **obtain** g
where $g: g \in \text{carrier } F2$ **and** *rep-eq*: $\text{hausdorff-rep } x = \text{sigma } g \ x$
unfolding *sigma-sphere-minus-bad-action.orbit-def* **by** *blast*
show *?thesis*
using *sigma-preserves-sphere2-minus-bad-set*[*OF g assms*] *rep-eq* **by** *simp*
qed

lemma *hausdorff-rep-eq-if-orbit-eq*:
assumes $\text{sigma-sphere-minus-bad-action.orbit } x =$
 $\text{sigma-sphere-minus-bad-action.orbit } y$
shows $\text{hausdorff-rep } x = \text{hausdorff-rep } y$
using *assms* **by** (*simp add: hausdorff-rep-def*)

lemma *hausdorff-rep-idem*:
assumes $x: x \in \text{sphere2} - \text{bad-set-D}$
shows $\text{hausdorff-rep } (\text{hausdorff-rep } x) = \text{hausdorff-rep } x$
proof –
let $?r = \text{hausdorff-rep } x$
have $rX: ?r \in \text{sphere2} - \text{bad-set-D}$
by (*rule hausdorff-rep-in-X*[*OF x*])
have $r\text{-orbit-x}: ?r \in \text{sigma-sphere-minus-bad-action.orbit } x$
by (*rule hausdorff-rep-in-orbit*[*OF x*])
have $r\text{-orbit-r}: ?r \in \text{sigma-sphere-minus-bad-action.orbit } ?r$
by (*rule sigma-sphere-minus-bad-action.orbit-self*[*OF rX*])
have $\text{sigma-sphere-minus-bad-action.orbit } ?r =$
 $\text{sigma-sphere-minus-bad-action.orbit } x$
by (*rule sigma-orbit-eq-if-common-point*[*OF rX x r-orbit-r r-orbit-x*])
hence $\text{hausdorff-rep } ?r = \text{hausdorff-rep } x$
by (*rule hausdorff-rep-eq-if-orbit-eq*)
thus *?thesis*
by *simp*
qed

lemma *hausdorff-lift-subset-X*:
assumes $u \subseteq \text{carrier } F2$
shows $\text{hausdorff-lift } u \subseteq \text{sphere2} - \text{bad-set-D}$
proof
fix y
assume $y \in \text{hausdorff-lift } u$
then obtain $g \ x$ **where** $g: g \in u$ **and** $x: x \in \text{sphere2} - \text{bad-set-D}$
and $y\text{-eq}: y = \text{sigma } g \ (\text{hausdorff-rep } x)$
unfolding *hausdorff-lift-def* **by** *blast*
have $g: g \in \text{carrier } F2$
using *assms gu* **by** *auto*

have $\text{rep}X$: $\text{hausdorff-rep } x \in \text{sphere2} - \text{bad-set-}D$
by (rule $\text{hausdorff-rep-in-}X[OF\ x]$)
show $y \in \text{sphere2} - \text{bad-set-}D$
using $\text{sigma-preserves-sphere2-minus-bad-set}[OF\ g\ \text{rep}X]$ $y\text{-eq}$ **by** simp
qed

lemma $F2\text{-act-image-set-subset-carrier}$:
assumes $g \in \text{carrier } F2$ **and** $u \subseteq \text{carrier } F2$
shows $F2\text{-act.image-set } g\ u \subseteq \text{carrier } F2$
unfolding $F2\text{-act.image-set-def}$
using assms **by** auto

lemma $\text{sigma-free-same-point-eq}$:
assumes $x: x \in \text{sphere2} - \text{bad-set-}D$
and $p: p \in \text{carrier } F2$
and $q: q \in \text{carrier } F2$
and $\text{eq}: \text{sigma } p\ x = \text{sigma } q\ x$
shows $p = q$
proof –
interpret G : $\text{group } F2$
by (rule $\text{free-group-is-group}$)
have $\text{inv}p: \text{inv}_{F2}\ p \in \text{carrier } F2$
by (rule $G.\text{inv-closed}[OF\ p]$)
have $\text{inv}pq: \text{inv}_{F2}\ p \otimes_{F2}\ q \in \text{carrier } F2$
by (rule $G.\text{m-closed}[OF\ \text{inv}p\ q]$)
have $\text{fixed}: \text{sigma } (\text{inv}_{F2}\ p \otimes_{F2}\ q)\ x = x$
proof –
have $\text{sigma } (\text{inv}_{F2}\ p \otimes_{F2}\ q)\ x =$
 $\text{sigma } (\text{inv}_{F2}\ p)\ (\text{sigma } q\ x)$
using $\text{sigma-homomorphism}[OF\ \text{inv}p\ q]$ **by** (simp add: o-def)
also have $\dots = \text{sigma } (\text{inv}_{F2}\ p)\ (\text{sigma } p\ x)$
using eq **by** simp
also have $\dots = x$
using $\text{sigma-inverse-left}[OF\ p]$ **by** simp
finally show $?thesis$.
qed
have $\text{inv}pq\text{-unit}: \text{inv}_{F2}\ p \otimes_{F2}\ q = []$
by (rule $\text{sigma-action-free-off-}D[OF\ x\ \text{inv}pq\ \text{fixed}]$)
have $q = p \otimes_{F2}\ []$
using $G.\text{inv-solve-left}'[OF\ G.\text{one-closed } p\ q]$ $\text{inv}pq\text{-unit}$ **by** simp
thus $?thesis$
using p **by** simp
qed

lemma $\text{hausdorff-lift-disjoint}$:
assumes $\text{disj}: u \cap v = \{\}$
and $u: u \subseteq \text{carrier } F2$
and $v: v \subseteq \text{carrier } F2$
shows $\text{hausdorff-lift } u \cap \text{hausdorff-lift } v = \{\}$

```

proof
  show hausdorff-lift u  $\cap$  hausdorff-lift v  $\subseteq$  {}
  proof
    fix z
    assume z: z  $\in$  hausdorff-lift u  $\cap$  hausdorff-lift v
    then obtain p x where pu: p  $\in$  u and x: x  $\in$  sphere2 - bad-set-D
      and z-p: z = sigma p (hausdorff-rep x)
      unfolding hausdorff-lift-def by blast
    from z obtain q y where qv: q  $\in$  v and y: y  $\in$  sphere2 - bad-set-D
      and z-q: z = sigma q (hausdorff-rep y)
      unfolding hausdorff-lift-def by blast
    let ?rx = hausdorff-rep x
    let ?ry = hausdorff-rep y
    have p: p  $\in$  carrier F2
      using u pu by auto
    have q: q  $\in$  carrier F2
      using v qv by auto
    have rxX: ?rx  $\in$  sphere2 - bad-set-D
      by (rule hausdorff-rep-in-X[OF x])
    have ryX: ?ry  $\in$  sphere2 - bad-set-D
      by (rule hausdorff-rep-in-X[OF y])
    have z-orbit-rx: z  $\in$  sigma-sphere-minus-bad-action.orbit ?rx
      unfolding sigma-sphere-minus-bad-action.orbit-def
      using p z-p by blast
    have z-orbit-ry: z  $\in$  sigma-sphere-minus-bad-action.orbit ?ry
      unfolding sigma-sphere-minus-bad-action.orbit-def
      using q z-q by blast
    have orbit-eq: sigma-sphere-minus-bad-action.orbit ?rx =
      sigma-sphere-minus-bad-action.orbit ?ry
      by (rule sigma-orbit-eq-if-common-point[OF rxX ryX z-orbit-rx z-orbit-ry])
    have rep-rx: hausdorff-rep ?rx = ?rx
      by (rule hausdorff-rep-idem[OF x])
    have rep-ry: hausdorff-rep ?ry = ?ry
      by (rule hausdorff-rep-idem[OF y])
    have rx-eq-ry: ?rx = ?ry
      using hausdorff-rep-eq-if-orbit-eq[OF orbit-eq] rep-rx rep-ry by simp
    have p-eq-q: p = q
      by (rule sigma-free-same-point-eq[OF rxX p q]) (use z-p z-q rx-eq-ry in simp)
    have p  $\in$  u  $\cap$  v
      using pu qv p-eq-q by auto
    with disj show z  $\in$  {}
      by simp
  qed
qed simp

```

```

lemma hausdorff-lift-pairwise-disjoint:
  assumes disj: pairwise-disjoint As
  and sub:  $\forall u \in$  set As. u  $\subseteq$  carrier F2
  shows pairwise-disjoint (map hausdorff-lift As)

```

unfolding *pairwise-disjoint-def*
proof (*intro allI impI*)
fix $i\ j$
assume $i: i < \text{length} (\text{map } \text{hausdorff-lift } As)$
and $j: j < \text{length} (\text{map } \text{hausdorff-lift } As)$
and $ij: i \neq j$
have $iA: i < \text{length } As$
using i **by** *simp*
have $jA: j < \text{length } As$
using j **by** *simp*
have $Ai: As ! i \subseteq \text{carrier } F2$
using *sub iA* **by** *simp*
have $Aj: As ! j \subseteq \text{carrier } F2$
using *sub jA* **by** *simp*
have $As ! i \cap As ! j = \{\}$
using *disj iA jA ij* **by** (*simp add: pairwise-disjoint-def*)
hence $\text{hausdorff-lift } (As ! i) \cap \text{hausdorff-lift } (As ! j) = \{\}$
by (*rule hausdorff-lift-disjoint[OF - Ai Aj]*)
thus $(\text{map } \text{hausdorff-lift } As) ! i \cap (\text{map } \text{hausdorff-lift } As) ! j = \{\}$
using $iA\ jA$ **by** *simp*
qed

lemma *hausdorff-sigma-image-lift*:
assumes $g: g \in \text{carrier } F2$
and $u: u \subseteq \text{carrier } F2$
shows $\text{hausdorff-sigma-image-set } g (\text{hausdorff-lift } u) =$
 $\text{hausdorff-lift } (F2\text{-act.image-set } g\ u)$
proof
show $\text{hausdorff-sigma-image-set } g (\text{hausdorff-lift } u) \subseteq$
 $\text{hausdorff-lift } (F2\text{-act.image-set } g\ u)$
proof
fix z
assume $z \in \text{hausdorff-sigma-image-set } g (\text{hausdorff-lift } u)$
then obtain y **where** $y: y \in \text{hausdorff-lift } u$ **and** $z\text{-eq}: z = \text{sigma } g\ y$
unfolding *hausdorff-sigma-image-set-def* **by** *blast*
from y **obtain** $a\ x$ **where** $au: a \in u$ **and** $x: x \in \text{sphere2} - \text{bad-set-}D$
and $y\text{-eq}: y = \text{sigma } a (\text{hausdorff-rep } x)$
unfolding *hausdorff-lift-def* **by** *blast*
have $a: a \in \text{carrier } F2$
using $u\ au$ **by** *auto*
have $ga\text{-image}: g \otimes_{F2} a \in F2\text{-act.image-set } g\ u$
unfolding *F2-act.image-set-def* **using** au **by** *auto*
have $z = \text{sigma } (g \otimes_{F2} a) (\text{hausdorff-rep } x)$
using *sigma-homomorphism[OF g a]* $z\text{-eq } y\text{-eq}$ **by** (*simp add: o-def*)
thus $z \in \text{hausdorff-lift } (F2\text{-act.image-set } g\ u)$
unfolding *hausdorff-lift-def* **using** $ga\text{-image } x$ **by** *blast*
qed
show $\text{hausdorff-lift } (F2\text{-act.image-set } g\ u) \subseteq$
 $\text{hausdorff-sigma-image-set } g (\text{hausdorff-lift } u)$

```

proof
  fix  $z$ 
  assume  $z \in \text{hausdorff-lift } (F2\text{-act.image-set } g \ u)$ 
  then obtain  $h \ x$  where  $h: h \in F2\text{-act.image-set } g \ u$ 
    and  $x: x \in \text{sphere2} - \text{bad-set-}D$ 
    and  $z\text{-eq}: z = \text{sigma } h \ (\text{hausdorff-rep } x)$ 
    unfolding  $\text{hausdorff-lift-def}$  by  $\text{blast}$ 
  from  $h$  obtain  $a$  where  $au: a \in u$  and  $h\text{-eq}: h = g \otimes_{F2} a$ 
    unfolding  $F2\text{-act.image-set-def}$  by  $\text{auto}$ 
  have  $a: a \in \text{carrier } F2$ 
    using  $u \ au$  by  $\text{auto}$ 
  have  $y\text{-lift}: \text{sigma } a \ (\text{hausdorff-rep } x) \in \text{hausdorff-lift } u$ 
    unfolding  $\text{hausdorff-lift-def}$  using  $au \ x$  by  $\text{blast}$ 
  have  $z = \text{sigma } g \ (\text{sigma } a \ (\text{hausdorff-rep } x))$ 
    using  $\text{sigma-homomorphism}[OF \ g \ a] \ z\text{-eq} \ h\text{-eq}$  by  $(\text{simp add: } o\text{-def})$ 
  thus  $z \in \text{hausdorff-sigma-image-set } g \ (\text{hausdorff-lift } u)$ 
    unfolding  $\text{hausdorff-sigma-image-set-def}$  using  $y\text{-lift}$  by  $\text{blast}$ 
qed
qed

lemma  $\text{hausdorff-translated-lift-pairwise-disjoint}$ :
  assumes  $len: \text{length } As = \text{length } gs$ 
    and  $gs: \text{set } gs \subseteq \text{carrier } F2$ 
    and  $sub: \forall u \in \text{set } As. \ u \subseteq \text{carrier } F2$ 
    and  $disj: \text{pairwise-disjoint } (\text{map2 } F2\text{-act.image-set } gs \ As)$ 
  shows  $\text{pairwise-disjoint } (\text{map2 } \text{hausdorff-sigma-image-set } gs \ (\text{map } \text{hausdorff-lift } As))$ 
    unfolding  $\text{pairwise-disjoint-def}$ 
proof  $(\text{intro allI impI})$ 
  fix  $i \ j$ 
  assume  $i: i < \text{length } (\text{map2 } \text{hausdorff-sigma-image-set } gs \ (\text{map } \text{hausdorff-lift } As))$ 
    and  $j: j < \text{length } (\text{map2 } \text{hausdorff-sigma-image-set } gs \ (\text{map } \text{hausdorff-lift } As))$ 
    and  $ij: i \neq j$ 
  have  $iA: i < \text{length } As$ 
    using  $i \ len$  by  $\text{simp}$ 
  have  $jA: j < \text{length } As$ 
    using  $j \ len$  by  $\text{simp}$ 
  have  $ig: gs \ ! \ i \in \text{carrier } F2$ 
proof  $-$ 
  have  $i < \text{length } gs$ 
    using  $iA \ len$  by  $\text{simp}$ 
  thus  $?thesis$ 
    using  $gs \ nth\text{-mem}$  by  $\text{blast}$ 
qed
have  $ig: gs \ ! \ j \in \text{carrier } F2$ 
proof  $-$ 
  have  $j < \text{length } gs$ 
    using  $jA \ len$  by  $\text{simp}$ 

```

```

thus ?thesis
  using gs nth-mem by blast
qed
have Ai: As ! i  $\subseteq$  carrier F2
  using sub iA by simp
have Aj: As ! j  $\subseteq$  carrier F2
  using sub jA by simp
have img-i: F2-act.image-set (gs ! i) (As ! i)  $\subseteq$  carrier F2
  by (rule F2-act.image-set-subset-carrier[OF ig Ai])
have img-j: F2-act.image-set (gs ! j) (As ! j)  $\subseteq$  carrier F2
  by (rule F2-act.image-set-subset-carrier[OF jg Aj])
have img-disj: F2-act.image-set (gs ! i) (As ! i)  $\cap$ 
  F2-act.image-set (gs ! j) (As ! j) = {}
  using disj iA jA ij len by (simp add: pairwise-disjoint-def)
have lift-disj: hausdorff-lift (F2-act.image-set (gs ! i) (As ! i))  $\cap$ 
  hausdorff-lift (F2-act.image-set (gs ! j) (As ! j)) = {}
  by (rule hausdorff-lift-disjoint[OF img-disj img-i img-j])
have trans-i: hausdorff-sigma-image-set (gs ! i) (hausdorff-lift (As ! i)) =
  hausdorff-lift (F2-act.image-set (gs ! i) (As ! i))
  by (rule hausdorff-sigma-image-lift[OF ig Ai])
have trans-j: hausdorff-sigma-image-set (gs ! j) (hausdorff-lift (As ! j)) =
  hausdorff-lift (F2-act.image-set (gs ! j) (As ! j))
  by (rule hausdorff-sigma-image-lift[OF jg Aj])
show (map2 hausdorff-sigma-image-set gs (map hausdorff-lift As)) ! i  $\cap$ 
  (map2 hausdorff-sigma-image-set gs (map hausdorff-lift As)) ! j = {}
  using lift-disj trans-i trans-j iA jA len by simp
qed

```

```

lemma hausdorff-lift-cover:
  assumes len: length As = length gs
  and gs: set gs  $\subseteq$  carrier F2
  and sub:  $\forall u \in$  set As. u  $\subseteq$  carrier F2
  and cover: carrier F2 =
    ( $\bigcup_{i < \text{length } As}$  F2-act.image-set (gs ! i) (As ! i))
  shows sphere2 - bad-set-D =
    ( $\bigcup_{i < \text{length } As}$  hausdorff-sigma-image-set (gs ! i) (hausdorff-lift (As ! i)))
proof -
  let ?X = sphere2 - bad-set-D
  let ?U = ( $\bigcup_{i < \text{length } As}$  hausdorff-sigma-image-set (gs ! i) (hausdorff-lift (As !
  i)))
  show ?X = ?U
  proof (rule subset-antisym)
  show ?X  $\subseteq$  ?U
  proof
  fix x
  assume x: x  $\in$  ?X
  interpret G: group F2
  by (rule free-group-is-group)
  from hausdorff-rep-in-orbit[OF x] obtain r

```

where $r: r \in \text{carrier } F2$ and $\text{rep-eq: hausdorff-rep } x = \text{sigma } r \ x$
 unfolding $\text{sigma-sphere-minus-bad-action.orbit-def}$ by blast
 define h where $h = \text{inv}_{F2} \ r$
 have $h: h \in \text{carrier } F2$
 unfolding $h\text{-def}$ by $(\text{rule } G.\text{inv-closed}[OF \ r])$
 have $x\text{-eq: } x = \text{sigma } h \ (\text{hausdorff-rep } x)$
 unfolding $h\text{-def}$ using $\text{rep-eq sigma-inverse-left}[OF \ r]$ by simp
 from $\text{cover } h$ obtain i where $i: i < \text{length } As$
 and $h\text{-img: } h \in F2.\text{act.image-set } (gs \ ! \ i) \ (As \ ! \ i)$
 by blast
 from $h\text{-img}$ obtain a where $aA: a \in As \ ! \ i$ and $h\text{-eq: } h = gs \ ! \ i \otimes_{F2} \ a$
 unfolding $F2.\text{act.image-set-def}$ by auto
 have $gi: gs \ ! \ i \in \text{carrier } F2$
 proof –
 have $i < \text{length } gs$
 using $i \text{ len}$ by simp
 thus $?thesis$
 using $gs \ \text{nth-mem}$ by blast
 qed
 have $Ai: As \ ! \ i \subseteq \text{carrier } F2$
 using $\text{sub } i$ by simp
 have $a: a \in \text{carrier } F2$
 using $Ai \ aA$ by auto
 have $y\text{-lift: } \text{sigma } a \ (\text{hausdorff-rep } x) \in \text{hausdorff-lift } (As \ ! \ i)$
 unfolding $\text{hausdorff-lift-def}$ using $aA \ x$ by blast
 have $\text{sigma } (gs \ ! \ i) \ (\text{sigma } a \ (\text{hausdorff-rep } x)) = \text{sigma } h \ (\text{hausdorff-rep } x)$
 using $\text{sigma-homomorphism}[OF \ gi \ a] \ h\text{-eq}$ by $(\text{simp add: } o\text{-def})$
 hence $x = \text{sigma } (gs \ ! \ i) \ (\text{sigma } a \ (\text{hausdorff-rep } x))$
 using $x\text{-eq}$ by simp
 hence $x \in \text{hausdorff-sigma-image-set } (gs \ ! \ i) \ (\text{hausdorff-lift } (As \ ! \ i))$
 unfolding $\text{hausdorff-sigma-image-set-def}$ using $y\text{-lift}$ by blast
 thus $x \in ?U$
 using i by blast
 qed
 show $?U \subseteq ?X$
 proof
 fix y
 assume $y \in ?U$
 then obtain i where $i: i < \text{length } As$
 and $y\text{-img: } y \in \text{hausdorff-sigma-image-set } (gs \ ! \ i) \ (\text{hausdorff-lift } (As \ ! \ i))$
 by blast
 from $y\text{-img}$ obtain z where $z\text{-lift: } z \in \text{hausdorff-lift } (As \ ! \ i)$
 and $y\text{-eq: } y = \text{sigma } (gs \ ! \ i) \ z$
 unfolding $\text{hausdorff-sigma-image-set-def}$ by blast
 have $gi: gs \ ! \ i \in \text{carrier } F2$
 proof –
 have $i < \text{length } gs$
 using $i \text{ len}$ by simp
 thus $?thesis$

```

    using gs nth-mem by blast
qed
have Ai: As ! i ⊆ carrier F2
    using sub i by simp
have zX: z ∈ ?X
    using hausdorff-lift-subset-X[OF Ai] z-lift by auto
show y ∈ ?X
    using sigma-preserves-sphere2-minus-bad-set[OF gi zX] y-eq by simp
qed
qed
qed

```

26 The Hausdorff paradox

Define the rotation-action of F_2 on the sphere by evaluation: $\text{sigma } w \cdot x = \text{sigma } w x$. Off the bad set, this is a free action, and the paradoxical decomposition of F_2 transports along orbits.

Hausdorff's theorem: $S^2 \setminus D$ is paradoxical under the action of $\text{SO}(3)$.

theorem *hausdorff-paradox-strong*:

```

∃ P Q :: (real^3) set list. ∃ gP gQ :: ((bool × gen2) list) list.
  length P = length gP ∧ length Q = length gQ ∧
  set gP ⊆ carrier F2 ∧ set gQ ⊆ carrier F2 ∧
  set (map sigma gP) ⊆ SO3 ∧ set (map sigma gQ) ⊆ SO3 ∧
  pairwise-disjoint (P @ Q) ∧
  pairwise-disjoint (map2 hausdorff-sigma-image-set gP P) ∧
  pairwise-disjoint (map2 hausdorff-sigma-image-set gQ Q) ∧
  (∀ i < length P. P ! i ⊆ sphere2 - bad-set-D) ∧
  (∀ i < length Q. Q ! i ⊆ sphere2 - bad-set-D) ∧
  (sphere2 - bad-set-D) =
    (⋃ i < length P. P ! i) ∪ (⋃ i < length Q. Q ! i) ∧
  (sphere2 - bad-set-D) =
    (⋃ i < length P. hausdorff-sigma-image-set (gP ! i) (P ! i)) ∧
  (sphere2 - bad-set-D) =
    (⋃ i < length Q. hausdorff-sigma-image-set (gQ ! i) (Q ! i))

```

proof –

```

from F2-paradoxical-partition obtain P0 Q0 :: ((bool × gen2) list) set list
and gP gQ :: ((bool × gen2) list) list
where lenP: length P0 = length gP
and lenQ: length Q0 = length gQ
and gP: set gP ⊆ carrier F2
and gQ: set gQ ⊆ carrier F2
and disj-all: pairwise-disjoint (P0 @ Q0)
and disj-imP: pairwise-disjoint (map2 F2-act.image-set gP P0)
and disj-imQ: pairwise-disjoint (map2 F2-act.image-set gQ Q0)
and source-cover0: carrier F2 =
  (⋃ i < length P0. P0 ! i) ∪ (⋃ i < length Q0. Q0 ! i)
and coverP: carrier F2 = (⋃ i < length P0. F2-act.image-set (gP ! i) (P0 !

```

i))
and *coverQ*: carrier *F2* = ($\bigcup_{i < \text{length } Q0}. F2\text{-act.image-set } (gQ ! i) (Q0 ! i)$)
i))
by *auto*
let *?P* = *map hausdorff-lift P0*
let *?Q* = *map hausdorff-lift Q0*
have *P0-sub*: $\forall u \in \text{set } P0. u \subseteq \text{carrier } F2$
proof
fix *u*
assume $u \in \text{set } P0$
then obtain *i* **where** $i: i < \text{length } P0$ **and** *u-eq*: $u = P0 ! i$
by (*auto simp: in-set-conv-nth*)
show $u \subseteq \text{carrier } F2$
using *source-cover0 i u-eq* **by** *auto*
qed
have *Q0-sub*: $\forall u \in \text{set } Q0. u \subseteq \text{carrier } F2$
proof
fix *u*
assume $u \in \text{set } Q0$
then obtain *i* **where** $i: i < \text{length } Q0$ **and** *u-eq*: $u = Q0 ! i$
by (*auto simp: in-set-conv-nth*)
show $u \subseteq \text{carrier } F2$
using *source-cover0 i u-eq* **by** *auto*
qed
have *all-sub*: $\forall u \in \text{set } (P0 @ Q0). u \subseteq \text{carrier } F2$
using *P0-sub Q0-sub* **by** *auto*
have *lenP'*: $\text{length } ?P = \text{length } gP$
using *lenP* **by** *simp*
have *lenQ'*: $\text{length } ?Q = \text{length } gQ$
using *lenQ* **by** *simp*
have *gP-SO3*: $\text{set } (\text{map } \text{sigma } gP) \subseteq \text{SO3}$
using *gP sigma-image-in-SO3* **by** *auto*
have *gQ-SO3*: $\text{set } (\text{map } \text{sigma } gQ) \subseteq \text{SO3}$
using *gQ sigma-image-in-SO3* **by** *auto*
have *source-disj*: *pairwise-disjoint* (*?P @ ?Q*)
using *hausdorff-lift-pairwise-disjoint[OF disj-all all-sub]* **by** *simp*
have *trans-disjP*: *pairwise-disjoint* (*map2 hausdorff-sigma-image-set gP ?P*)
by (*rule hausdorff-translated-lift-pairwise-disjoint[OF lenP gP P0-sub disj-imP]*)
have *trans-disjQ*: *pairwise-disjoint* (*map2 hausdorff-sigma-image-set gQ ?Q*)
by (*rule hausdorff-translated-lift-pairwise-disjoint[OF lenQ gQ Q0-sub disj-imQ]*)
have *P-sub-X*: $\forall i < \text{length } ?P. ?P ! i \subseteq \text{sphere2} - \text{bad-set-D}$
proof (*intro allI impI*)
fix *i*
assume $i: i < \text{length } ?P$
have $P0 ! i \subseteq \text{carrier } F2$
using *P0-sub i* **by** *simp*
thus $?P ! i \subseteq \text{sphere2} - \text{bad-set-D}$
using *i hausdorff-lift-subset-X* **by** *simp*
qed

```

have Q-sub-X:  $\forall i < \text{length } ?Q. ?Q ! i \subseteq \text{sphere2} - \text{bad-set-D}$ 
proof (intro allI impI)
  fix i
  assume i:  $i < \text{length } ?Q$ 
  have Q0 ! i  $\subseteq \text{carrier } F2$ 
  using Q0-sub i by simp
  thus ?Q ! i  $\subseteq \text{sphere2} - \text{bad-set-D}$ 
  using i hausdorff-lift-subset-X by simp
qed
have source-cover':  $\text{sphere2} - \text{bad-set-D} =$ 
   $(\bigcup i < \text{length } ?P. ?P ! i) \cup (\bigcup i < \text{length } ?Q. ?Q ! i)$ 
proof (rule subset-antisym)
  show  $\text{sphere2} - \text{bad-set-D} \subseteq$ 
     $(\bigcup i < \text{length } ?P. ?P ! i) \cup (\bigcup i < \text{length } ?Q. ?Q ! i)$ 
  proof
    fix x
    assume x:  $x \in \text{sphere2} - \text{bad-set-D}$ 
    interpret G: group F2
    by (rule free-group-is-group)
    from hausdorff-rep-in-orbit[OF x] obtain r
      where r:  $r \in \text{carrier } F2$  and rep-eq:  $\text{hausdorff-rep } x = \text{sigma } r x$ 
    unfolding sigma-sphere-minus-bad-action.orbit-def by blast
    define h where  $h = \text{inv}_{F2} r$ 
    have h:  $h \in \text{carrier } F2$ 
    unfolding h-def by (rule G.inv-closed[OF r])
    have x-eq:  $x = \text{sigma } h (\text{hausdorff-rep } x)$ 
    unfolding h-def using rep-eq sigma-inverse-left[OF r] by simp
    from source-cover0 h have h-cases:
       $h \in (\bigcup i < \text{length } P0. P0 ! i) \vee h \in (\bigcup i < \text{length } Q0. Q0 ! i)$ 
    by blast
    thus x  $\in (\bigcup i < \text{length } ?P. ?P ! i) \cup (\bigcup i < \text{length } ?Q. ?Q ! i)$ 
  proof
    assume h  $\in (\bigcup i < \text{length } P0. P0 ! i)$ 
    then obtain i where i:  $i < \text{length } P0$  and hP:  $h \in P0 ! i$ 
    by blast
    have x  $\in \text{hausdorff-lift } (P0 ! i)$ 
    unfolding hausdorff-lift-def using hP x x-eq by blast
    hence x  $\in ?P ! i$ 
    using i by simp
    hence x  $\in (\bigcup i < \text{length } ?P. ?P ! i)$ 
    using i by auto
  thus ?thesis
    by blast
  next
    assume h  $\in (\bigcup i < \text{length } Q0. Q0 ! i)$ 
    then obtain i where i:  $i < \text{length } Q0$  and hQ:  $h \in Q0 ! i$ 
    by blast
    have x  $\in \text{hausdorff-lift } (Q0 ! i)$ 
    unfolding hausdorff-lift-def using hQ x x-eq by blast
  
```

```

    hence  $x \in ?Q ! i$ 
      using  $i$  by simp
    hence  $x \in (\bigcup i < \text{length } ?Q. ?Q ! i)$ 
      using  $i$  by auto
    thus ?thesis
      by blast
  qed
qed
show  $(\bigcup i < \text{length } ?P. ?P ! i) \cup (\bigcup i < \text{length } ?Q. ?Q ! i)$ 
   $\subseteq \text{sphere2} - \text{bad-set-D}$ 
  using P-sub-X Q-sub-X by auto
qed
have coverP':  $\text{sphere2} - \text{bad-set-D} =$ 
   $(\bigcup i < \text{length } ?P. \text{hausdorff-sigma-image-set } (gP ! i) (?P ! i))$ 
  using hausdorff-lift-cover[OF lenP gP P0-sub coverP] by simp
have coverQ':  $\text{sphere2} - \text{bad-set-D} =$ 
   $(\bigcup i < \text{length } ?Q. \text{hausdorff-sigma-image-set } (gQ ! i) (?Q ! i))$ 
  using hausdorff-lift-cover[OF lenQ gQ Q0-sub coverQ] by simp
show ?thesis
  apply (rule exI[of - ?P])
  apply (rule exI[of - ?Q])
  apply (rule exI[of - gP])
  apply (rule exI[of - gQ])
  using lenP' lenQ' gP gQ gP-SO3 gQ-SO3 source-disj trans-disjP trans-disjQ
    P-sub-X Q-sub-X source-cover' coverP' coverQ'
  by blast
qed

theorem hausdorff-paradoxical:
  sigma-sphere-minus-bad-action.paradoxical ( $\text{sphere2} - \text{bad-set-D}$ )
proof -
  from hausdorff-paradox-strong obtain  $P Q :: (\text{real}^3) \text{ set list}$ 
  and  $gP gQ :: ((\text{bool} \times \text{gen2}) \text{ list}) \text{ list}$ 
  where  $\text{length } P = \text{length } gP$  and  $\text{length } Q = \text{length } gQ$ 
  and  $\text{set } gP \subseteq \text{carrier } F2$  and  $\text{set } gQ \subseteq \text{carrier } F2$ 
  and pairwise-disjoint ( $P @ Q$ )
  and pairwise-disjoint ( $\text{map2 } \text{hausdorff-sigma-image-set } gP P$ )
  and pairwise-disjoint ( $\text{map2 } \text{hausdorff-sigma-image-set } gQ Q$ )
  and P-sub:  $\forall i < \text{length } P. P ! i \subseteq \text{sphere2} - \text{bad-set-D}$ 
  and Q-sub:  $\forall i < \text{length } Q. Q ! i \subseteq \text{sphere2} - \text{bad-set-D}$ 
  and coverP:  $\text{sphere2} - \text{bad-set-D} =$ 
     $(\bigcup i < \text{length } P. \text{hausdorff-sigma-image-set } (gP ! i) (P ! i))$ 
  and coverQ:  $\text{sphere2} - \text{bad-set-D} =$ 
     $(\bigcup i < \text{length } Q. \text{hausdorff-sigma-image-set } (gQ ! i) (Q ! i))$ 
  by auto
have pieces-sub:  $(\bigcup i < \text{length } P. P ! i) \cup (\bigcup i < \text{length } Q. Q ! i)$ 
   $\subseteq \text{sphere2} - \text{bad-set-D}$ 
  using P-sub Q-sub by auto
have imageP-eq:  $\text{map2 } \text{sigma-sphere-minus-bad-action.image-set } gP P =$ 

```

```

    map2 hausdorff-sigma-image-set gP P
  by (rule map2-hausdorff-sigma-image-set-eq)
  have imageQ-eq: map2 sigma-sphere-minus-bad-action.image-set gQ Q =
    map2 hausdorff-sigma-image-set gQ Q
  by (rule map2-hausdorff-sigma-image-set-eq)
  have coverP-action: sphere2 - bad-set-D =
    (⋃ i <length P. sigma-sphere-minus-bad-action.image-set (gP ! i) (P ! i))
  using coverP
  by (simp add: hausdorff-sigma-image-set-def sigma-sphere-minus-bad-action.image-set-def)
  have coverQ-action: sphere2 - bad-set-D =
    (⋃ i <length Q. sigma-sphere-minus-bad-action.image-set (gQ ! i) (Q ! i))
  using coverQ
  by (simp add: hausdorff-sigma-image-set-def sigma-sphere-minus-bad-action.image-set-def)
  have imageP-disj-action: pairwise-disjoint (map2 sigma-sphere-minus-bad-action.image-set
gP P)
  using <pairwise-disjoint (map2 hausdorff-sigma-image-set gP P)> imageP-eq
  by metis
  have imageQ-disj-action: pairwise-disjoint (map2 sigma-sphere-minus-bad-action.image-set
gQ Q)
  using <pairwise-disjoint (map2 hausdorff-sigma-image-set gQ Q)> imageQ-eq
  by metis
  show ?thesis
  unfolding sigma-sphere-minus-bad-action.paradoxical-def
  apply (rule exI[of - P])
  apply (rule exI[of - Q])
  apply (rule exI[of - gP])
  apply (rule exI[of - gQ])
  using <length P = length gP> <length Q = length gQ>
    <set gP ⊆ carrier F2> <set gQ ⊆ carrier F2>
    <pairwise-disjoint (P @ Q)>
    imageP-disj-action imageQ-disj-action pieces-sub coverP-action coverQ-action
  by blast
qed

```

theorem hausdorff-paradox-rot-strong:

```

∃ P Q :: (real^3) set list. ∃ gP gQ :: ((real^3) ⇒ (real^3)) list.
  length P = length gP ∧ length Q = length gQ ∧
  set gP ⊆ SO3 ∧ set gQ ⊆ SO3 ∧
  pairwise-disjoint (P @ Q) ∧
  pairwise-disjoint (map2 (λg A. g ' A) gP P) ∧
  pairwise-disjoint (map2 (λg A. g ' A) gQ Q) ∧
  (∀ i < length P. P ! i ⊆ sphere2 - bad-set-D) ∧
  (∀ i < length Q. Q ! i ⊆ sphere2 - bad-set-D) ∧
  (sphere2 - bad-set-D) =
    (⋃ i <length P. P ! i) ∪ (⋃ i <length Q. Q ! i) ∧
  (sphere2 - bad-set-D) =
    (⋃ i <length P. (gP ! i) ' (P ! i)) ∧
  (sphere2 - bad-set-D) =
    (⋃ i <length Q. (gQ ! i) ' (Q ! i))

```

```

proof –
from hausdorff-paradox-strong obtain P Q :: (real3) set list
  and wP wQ :: ((bool × gen2) list) list
  where lenP: length P = length wP
    and lenQ: length Q = length wQ
    and wP-SO3: set (map sigma wP) ⊆ SO3
    and wQ-SO3: set (map sigma wQ) ⊆ SO3
    and disj: pairwise-disjoint (P @ Q)
    and disjP: pairwise-disjoint (map2 hausdorff-sigma-image-set wP P)
    and disjQ: pairwise-disjoint (map2 hausdorff-sigma-image-set wQ Q)
    and P-sub: ∀ i < length P. P ! i ⊆ sphere2 – bad-set-D
    and Q-sub: ∀ i < length Q. Q ! i ⊆ sphere2 – bad-set-D
    and source-cover: sphere2 – bad-set-D =
      (⋃ i < length P. P ! i) ∪ (⋃ i < length Q. Q ! i)
    and coverP: sphere2 – bad-set-D =
      (⋃ i < length P. hausdorff-sigma-image-set (wP ! i) (P ! i))
    and coverQ: sphere2 – bad-set-D =
      (⋃ i < length Q. hausdorff-sigma-image-set (wQ ! i) (Q ! i))
  by auto
let ?gP = map sigma wP
let ?gQ = map sigma wQ
have map2P-eq:
  map2 (λg A. g ‘ A) ?gP P = map2 hausdorff-sigma-image-set wP P
  using lenP
proof (induction wP arbitrary: P)
  case Nil
  then show ?case by simp
next
  case (Cons w ws)
  then obtain A Ps where P: P = A # Ps
  by (cases P) auto
  with Cons show ?case
  by (simp add: hausdorff-sigma-image-set-def)
qed
have map2Q-eq:
  map2 (λg A. g ‘ A) ?gQ Q = map2 hausdorff-sigma-image-set wQ Q
  using lenQ
proof (induction wQ arbitrary: Q)
  case Nil
  then show ?case by simp
next
  case (Cons w ws)
  then obtain A Qs where Q: Q = A # Qs
  by (cases Q) auto
  with Cons show ?case
  by (simp add: hausdorff-sigma-image-set-def)
qed
have disjP-rot: pairwise-disjoint (map2 (λg A. g ‘ A) ?gP P)
  using disjP map2P-eq by simp

```

```

have disjQ-rot: pairwise-disjoint (map2 (λg A. g ‘ A) ?gQ Q)
  using disjQ map2Q-eq by simp
have coverP-rot: sphere2 - bad-set-D =
  (⋃ i < length P. (?gP ! i) ‘ (P ! i))
  using coverP lenP by (simp add: hausdorff-sigma-image-set-def)
have coverQ-rot: sphere2 - bad-set-D =
  (⋃ i < length Q. (?gQ ! i) ‘ (Q ! i))
  using coverQ lenQ by (simp add: hausdorff-sigma-image-set-def)
show ?thesis
  apply (rule exI[of - P])
  apply (rule exI[of - Q])
  apply (rule exI[of - ?gP])
  apply (rule exI[of - ?gQ])
  using lenP lenQ wP-SO3 wQ-SO3 disj disjP-rot disjQ-rot
    P-sub Q-sub source-cover coverP-rot coverQ-rot
  by simp
qed

```

theorem *hausdorff-paradox*:

```

∃ P Q :: (real^3) set list. ∃ gP gQ :: ((real^3) ⇒ (real^3)) list.
  length P = length gP ∧ length Q = length gQ ∧
  set gP ⊆ SO3 ∧ set gQ ⊆ SO3 ∧
  (∀ i < length P. P ! i ⊆ sphere2 - bad-set-D) ∧
  (∀ i < length Q. Q ! i ⊆ sphere2 - bad-set-D) ∧
  (sphere2 - bad-set-D) =
    (⋃ i < length P. (gP ! i) ‘ (P ! i)) ∧
  (sphere2 - bad-set-D) =
    (⋃ i < length Q. (gQ ! i) ‘ (Q ! i))

```

proof –

```

from hausdorff-paradox-strong obtain P Q :: (real^3) set list
and wP wQ :: ((bool × gen2) list) list
where lenP: length P = length wP
and lenQ: length Q = length wQ
and wP-SO3: set (map sigma wP) ⊆ SO3
and wQ-SO3: set (map sigma wQ) ⊆ SO3
and P-sub: ∀ i < length P. P ! i ⊆ sphere2 - bad-set-D
and Q-sub: ∀ i < length Q. Q ! i ⊆ sphere2 - bad-set-D
and coverP: sphere2 - bad-set-D =
  (⋃ i < length P. hausdorff-sigma-image-set (wP ! i) (P ! i))
and coverQ: sphere2 - bad-set-D =
  (⋃ i < length Q. hausdorff-sigma-image-set (wQ ! i) (Q ! i))
by auto
let ?gP = map sigma wP
let ?gQ = map sigma wQ
have coverP-rot: sphere2 - bad-set-D =
  (⋃ i < length P. (?gP ! i) ‘ (P ! i))
  using coverP lenP by (simp add: hausdorff-sigma-image-set-def)
have coverQ-rot: sphere2 - bad-set-D =
  (⋃ i < length Q. (?gQ ! i) ‘ (Q ! i))

```

```

    using coverQ lenQ by (simp add: hausdorff-sigma-image-set-def)
  show ?thesis
    apply (rule exI[of - P])
    apply (rule exI[of - Q])
    apply (rule exI[of - ?gP])
    apply (rule exI[of - ?gQ])
    using lenP lenQ wP-SO3 wQ-SO3 P-sub Q-sub coverP-rot coverQ-rot
    by simp
qed

end

```

```

theory Sphere-Decomposition
  imports Hausdorff-Paradox
begin

```

Equidecomposability of $S^2 \setminus D$ and S^2 .

```

definition Rz-angle-mat :: real  $\Rightarrow$  real3 where
  Rz-angle-mat t = vector [
    vector [cos t, - sin t, 0],
    vector [sin t, cos t, 0],
    vector [0, 0, 1]]

```

```

definition Rz-angle :: real  $\Rightarrow$  real3  $\Rightarrow$  real3 where
  Rz-angle t v = Rz-angle-mat t * v v

```

```

definition e1-3 :: real3 where
  e1-3 = vector [1, 0, 0]

```

```

definition e3-3 :: real3 where
  e3-3 = vector [0, 0, 1]

```

```

lemma Rz-angle-mat-orthogonal: orthogonal-matrix (Rz-angle-mat t)
  unfolding orthogonal-matrix Rz-angle-mat-def
  by (simp add: matrix-matrix-mult-def transpose-def mat-def vec-eq-iff vector-def
  sum-3 forall-3
  power2-eq-square algebra-simps)

```

```

lemma det-Rz-angle-mat [simp]: matrix-det (Rz-angle-mat t) = 1
  unfolding Rz-angle-mat-def
  by (simp add: det-3 vector-def power2-eq-square algebra-simps)

```

```

lemma Rz-angle-in-SO3: Rz-angle t  $\in$  SO3
  unfolding SO3-def rotation-def Rz-angle-def
  by (simp add: orthogonal-transformation-matrix Rz-angle-mat-orthogonal ma-
  trix-vector-mul-linear)

```

```

lemma Rz-angle-mat-add:

```

$Rz\text{-angle-mat } a ** Rz\text{-angle-mat } b = Rz\text{-angle-mat } (a + b)$
unfolding $Rz\text{-angle-mat-def}$
by (*simp add: matrix-matrix-mult-def vec-eq-iff vector-def sum-3 forall-3*
cos-add sin-add algebra-simps)

lemma $Rz\text{-angle-compose}$:

$Rz\text{-angle } a \circ Rz\text{-angle } b = Rz\text{-angle } (a + b)$
by (*rule ext*) (*simp add: Rz-angle-def matrix-vector-mul-assoc Rz-angle-mat-add*)

lemma $Rz\text{-angle-funpow}$:

$(Rz\text{-angle } t \overset{\sim}{\sim} n) = Rz\text{-angle } (real\ n * t)$

proof (*induction n*)

case 0

show *?case*

by (*rule ext*) (*simp add: Rz-angle-def Rz-angle-mat-def mat-def matrix-vector-mult-def*
vec-eq-iff vector-def forall-3 sum-3)

next

case ($Suc\ n$)

have $(Rz\text{-angle } t \overset{\sim}{\sim} Suc\ n) = Rz\text{-angle } t \circ Rz\text{-angle } (real\ n * t)$

using $Suc.IH$ **by** *simp*

also have $\dots = Rz\text{-angle } (t + real\ n * t)$

by (*rule Rz-angle-compose*)

also have $t + real\ n * t = real\ (Suc\ n) * t$

by (*simp add: algebra-simps*)

finally show *?case* .

qed

lemma $Rz\text{-angle-components}$ [*simp*]:

$(Rz\text{-angle } t\ x)\ \$\ 1 = \cos\ t * x\ \$\ 1 - \sin\ t * x\ \$\ 2$

$(Rz\text{-angle } t\ x)\ \$\ 2 = \sin\ t * x\ \$\ 1 + \cos\ t * x\ \$\ 2$

$(Rz\text{-angle } t\ x)\ \$\ 3 = x\ \$\ 3$

by (*simp-all add: Rz-angle-def Rz-angle-mat-def matrix-vector-mult-def*
vector-def sum-3)

lemma $Rz\text{-angle-eq-non-axis-sin-cos}$:

assumes $Rz\text{-angle } a\ x = Rz\text{-angle } b\ x$ **and** $x\ \$\ 1 \neq 0 \vee x\ \$\ 2 \neq 0$

shows $\sin\ a = \sin\ b \wedge \cos\ a = \cos\ b$

proof –

let $?X = x\ \$\ 1$

let $?Y = x\ \$\ 2$

let $?C = \cos\ a - \cos\ b$

let $?S = \sin\ a - \sin\ b$

have $comp1: (Rz\text{-angle } a\ x)\ \$\ 1 = (Rz\text{-angle } b\ x)\ \$\ 1$

using $arg\text{-cong}[OF\ assms(1),\ of\ \lambda v. v\ \$\ 1]$.

have $comp2: (Rz\text{-angle } a\ x)\ \$\ 2 = (Rz\text{-angle } b\ x)\ \$\ 2$

using $arg\text{-cong}[OF\ assms(1),\ of\ \lambda v. v\ \$\ 2]$.

have $eq1: ?C * ?X - ?S * ?Y = 0$

using $comp1$ **by** (*simp add: algebra-simps*)

have $eq2: ?S * ?X + ?C * ?Y = 0$

```

    using comp2 by (simp add: algebra-simps)
  have nonzero: ?X2 + ?Y2 ≠ 0
    using assms(2) by (simp add: power2-eq-square sum-squares-eq-zero-iff)
  have C-eq: ?C * (?X2 + ?Y2) =
    (?C * ?X - ?S * ?Y) * ?X + (?S * ?X + ?C * ?Y) * ?Y
    by (simp add: power2-eq-square algebra-simps)
  have Cprod0: ?C * (?X2 + ?Y2) = 0
    using C-eq eq1 eq2 by simp
  hence C-or: ?C = 0 ∨ ?X2 + ?Y2 = 0
    by (simp add: mult-eq-0-iff)
  have C0: ?C = 0
    using C-or nonzero by blast
  have S-eq: ?S * (?X2 + ?Y2) =
    (?S * ?X + ?C * ?Y) * ?X - (?C * ?X - ?S * ?Y) * ?Y
    by (simp add: power2-eq-square algebra-simps)
  have Sprod0: ?S * (?X2 + ?Y2) = 0
    using S-eq eq1 eq2 by simp
  hence S-or: ?S = 0 ∨ ?X2 + ?Y2 = 0
    by (simp add: mult-eq-0-iff)
  have S0: ?S = 0
    using S-or nonzero by blast
  from S0 C0 show ?thesis
    by simp
qed

```

lemma *Rz-angle-collision-angles-countable*:

```

  assumes nonaxis: x $ 1 ≠ 0 ∨ x $ 2 ≠ 0
  shows countable {t. Rz-angle t x = y}
  proof (cases {t. Rz-angle t x = y} = {})
  case True
    then show ?thesis by simp
  next
  case False
    then obtain a where a: Rz-angle a x = y
      by auto
    have subset: {t. Rz-angle t x = y} ⊆ (λn::int. a + 2 * pi * n) ‘ UNIV
    proof
      fix t
      assume t: t ∈ {t. Rz-angle t x = y}
      hence eq: Rz-angle t x = Rz-angle a x
        using a by simp
      have trig: sin t = sin a ∧ cos t = cos a
        by (rule Rz-angle-eq-non-axis-sin-cos[OF eq nonaxis])
      have ∃n::int. t = a + 2 * pi * n
        using iffD1[OF sin-cos-eq-iff[of t a] trig] .
      then obtain n :: int where n: t = a + 2 * pi * n
        by blast
      show t ∈ (λn::int. a + 2 * pi * n) ‘ UNIV
        using n by simp
    end
  end

```

qed
have *countable* (($\lambda n::int. a + 2 * pi * n$) ‘ *UNIV*)
by *simp*
thus *?thesis*
by (*rule countable-subset[OF subset]*)
qed

lemma *Rz-angle-scaled-collision-angles-countable*:
assumes $k > 0$ **and** *nonaxis*: $x \$ 1 \neq 0 \vee x \$ 2 \neq 0$
shows *countable* { $t. Rz\text{-angle } (real\ k * t) x = y$ }
proof –
let $?f = \lambda t::real. real\ k * t$
let $?A = \{u. Rz\text{-angle } u\ x = y\}$
have *count-A*: *countable* $?A$
by (*rule Rz-angle-collision-angles-countable[OF nonaxis]*)
have *inj-f*: *inj-on* $?f$ *UNIV*
using *assms(1)* **by** (*auto simp: inj-on-def*)
have *count-pre*: *countable* { $t \in UNIV. ?f\ t \in ?A$ }
by (*rule countable-image-inj-gen[OF inj-f count-A]*)
thus *?thesis*
by *simp*
qed

lemma *e1-3-in-sphere2*: $e1-3 \in sphere2$
by (*simp add: e1-3-def sphere2-def norm-eq-1 inner-vec-def sum-3 vector-def*)

lemma *e1-3-nonaxis*: $e1-3 \$ 1 \neq 0 \vee e1-3 \$ 2 \neq 0$
by (*simp add: e1-3-def*)

lemma *e3-3-in-sphere2*: $e3-3 \in sphere2$
by (*simp add: e3-3-def sphere2-def norm-eq-1 inner-vec-def sum-3 vector-def*)

lemma *z-axis-sphere2-cases*:
assumes $x \in sphere2$ **and** $x \$ 1 = 0$ **and** $x \$ 2 = 0$
shows $x = e3-3 \vee x = - e3-3$
proof –
have $x3\text{-sq}: (x \$ 3)^2 = 1$
using *assms unfolding sphere2-def*
by (*simp add: norm-eq-1 inner-vec-def sum-3 power2-eq-square*)
hence $x \$ 3 = 1 \vee x \$ 3 = -1$
by (*simp add: power2-eq-1-iff*)
thus *?thesis*
using *assms(2,3)*
by (*auto simp: e3-3-def vec-eq-iff vector-def forall-3*)
qed

lemma *SO3-funpow*:
assumes $R \in SO3$
shows $(R \wedge n) \in SO3$

```

proof (induction n)
  case 0
  show ?case
    using id-in-SO3 by (simp add: id-def)
next
  case (Suc n)
  have  $R \circ (R \sim n) \in SO3$ 
    by (rule SO3-closed-compose[OF assms Suc.IH])
  thus ?case
    by (simp add: o-def)
qed

lemma SO3-inj:
  assumes  $f \in SO3$ 
  shows inj f
proof –
  have orthogonal-transformation f
    using assms unfolding SO3-def rotation-def by auto
  thus ?thesis
    by (rule orthogonal-transformation-inj)
qed

lemma SO3-surj:
  assumes  $f \in SO3$ 
  shows surj f
proof –
  have orthogonal-transformation f
    using assms unfolding SO3-def rotation-def by auto
  thus ?thesis
    by (rule orthogonal-transformation-surj)
qed

lemma SO3-linear:
  assumes  $f \in SO3$ 
  shows linear f
proof –
  have orthogonal-transformation f
    using assms unfolding SO3-def rotation-def by auto
  thus ?thesis
    by (rule orthogonal-transformation-linear)
qed

lemma SO3-inverse:
  assumes  $f \in SO3$ 
  shows Hilbert-Choice.inv f  $\in SO3$ 
proof –
  have f-orth: orthogonal-transformation f
    and f-det: matrix-det (matrix f) = 1
    using assms unfolding SO3-def rotation-def by auto

```

have *inv-orth*: orthogonal-transformation (Hilbert-Choice.inv f)
by (rule orthogonal-transformation-inv[OF f-orth])
have *f-lin*: linear f
by (rule orthogonal-transformation-linear[OF f-orth])
have *inv-lin*: linear (Hilbert-Choice.inv f)
by (rule orthogonal-transformation-linear[OF inv-orth])
have *surj-f*: surj f
by (rule orthogonal-transformation-surj[OF f-orth])
have *comp-id*: f ∘ Hilbert-Choice.inv f = id
using *surj-f* **by** (auto simp: fun-eq-iff surj-f-inv-f)
have *matrix-det* (matrix (f ∘ Hilbert-Choice.inv f)) =
matrix-det (matrix f) * matrix-det (matrix (Hilbert-Choice.inv f))
using *matrix-compose*[OF inv-lin f-lin] **by** (simp add: det-mul)
also have ... = matrix-det (matrix (Hilbert-Choice.inv f))
using *f-det* **by** simp
finally have matrix-det (matrix (Hilbert-Choice.inv f)) = 1
using *comp-id* **by** simp
with *inv-orth* **show** ?thesis
unfolding SO3-def rotation-def **by** simp
qed

lemma *conjugate-funpow*:
assumes *bijS*: bij S
shows ((S ∘ R ∘ Hilbert-Choice.inv S) $\overset{\sim}{\sim}$ n) =
S ∘ (R $\overset{\sim}{\sim}$ n) ∘ Hilbert-Choice.inv S
proof (induction n)
case 0
show ?case
using *bijS* **by** (auto simp: fun-eq-iff bij-is-surj surj-f-inv-f)
next
case (Suc n)
have *injS*: inj S
using *bijS* **by** (simp add: bij-is-inj)
show ?case
proof (rule ext)
fix x
have *n-eq*: ((S ∘ R ∘ Hilbert-Choice.inv S) $\overset{\sim}{\sim}$ n) x =
S ((R $\overset{\sim}{\sim}$ n) (Hilbert-Choice.inv S x))
using *fun-cong*[OF Suc.IH, of x] **by** (simp add: o-def)
have ((S ∘ R ∘ Hilbert-Choice.inv S) $\overset{\sim}{\sim}$ Suc n) x =
S (R (Hilbert-Choice.inv S (((S ∘ R ∘ Hilbert-Choice.inv S) $\overset{\sim}{\sim}$ n) x)))
by simp
also have ... = S (R (Hilbert-Choice.inv S
(S ((R $\overset{\sim}{\sim}$ n) (Hilbert-Choice.inv S x))))))
using *n-eq* **by** simp
also have ... = S ((R $\overset{\sim}{\sim}$ Suc n) (Hilbert-Choice.inv S x))
using *injS* **by** (simp add: inv-f-f)
finally show ((S ∘ R ∘ Hilbert-Choice.inv S) $\overset{\sim}{\sim}$ Suc n) x =
(S ∘ (R $\overset{\sim}{\sim}$ Suc n) ∘ Hilbert-Choice.inv S) x

by *simp*
 qed
 qed

lemma *conjugate-funpow-image*:
 assumes *bijS*: *bij S*
 shows $((S \circ R \circ \text{Hilbert-Choice.inv } S) \overset{\sim}{\sim} n) \text{ ' } D =$
 $S \text{ ' } ((R \overset{\sim}{\sim} n) \text{ ' } (\text{Hilbert-Choice.inv } S \text{ ' } D))$
 using *conjugate-funpow*[*OF bijS, of n R*]
 by *auto*

lemma *exists-antipodal-axis-avoiding-countable*:

assumes *count-D*: *countable D*
 shows $\exists a \in \text{sphere2}. a \notin D \wedge -a \notin D$

proof –

let $?A = D \cup \text{uminus ' } D$
 have *count-A*: *countable ?A*

using *count-D* by *simp*

let $?bad = (\bigcup y \in ?A. \{t. \text{Rz-angle } t \text{ e1-3} = y\})$

have *count-bad*: *countable ?bad*

proof (*intro countable-UN count-A*)

fix *y*

assume $y \in ?A$

show *countable* $\{t. \text{Rz-angle } t \text{ e1-3} = y\}$

by (*rule Rz-angle-collision-angles-countable*[*OF e1-3-nonaxis*])

qed

obtain *t* **where** *t-not-bad*: $t \notin ?bad$

using *real-interval-avoid-countable-set*[*OF zero-less-one count-bad*] by *blast*

define *a* **where** $a = \text{Rz-angle } t \text{ e1-3}$

have *a-sphere*: $a \in \text{sphere2}$

unfolding *a-def*

by (*rule rotation-preserves-sphere2*[*OF Rz-angle-in-SO3 e1-3-in-sphere2*])

have *a-not-A*: $a \notin ?A$

using *t-not-bad* **unfolding** *a-def* by *blast*

hence *a-not-D*: $a \notin D$

by *simp*

have *neg-a-not-D*: $-a \notin D$

proof

assume $-a \in D$

hence *uminus* $(-a) \in \text{uminus ' } D$

by *blast*

hence $a \in \text{uminus ' } D$

by *simp*

with *a-not-A* **show** *False*

by *simp*

qed

from *a-sphere a-not-D neg-a-not-D* **show** *?thesis*

by *blast*

qed

lemma *exists-Rz-angle-absorbing*:
assumes *count-D*: countable D
and *nonaxis*: $\bigwedge x. x \in D \implies x \$ 1 \neq 0 \vee x \$ 2 \neq 0$
shows $\exists t. \forall n m. n \neq m \longrightarrow ((Rz\text{-angle } t \overset{\sim}{\sim} n) ' D) \cap ((Rz\text{-angle } t \overset{\sim}{\sim} m) ' D) = \{\}$
proof –
let $?bad = (\bigcup k \in \{k::nat. k > 0\}. \bigcup x \in D. \bigcup y \in D. \{t. Rz\text{-angle } (real\ k * t) x = y\})$
have *count-bad*: countable $?bad$
proof (*intro countable-UN countable-Collect countableI-type count-D*)
fix $k\ x\ y$
assume $k \in \{k::nat. 0 < k\}$ **and** $x \in D$ **and** $y \in D$
thus countable $\{t. Rz\text{-angle } (real\ k * t) x = y\}$
by (*intro Rz-angle-scaled-collision-angles-countable nonaxis*) *auto*
qed
obtain t **where** *t-not-bad*: $t \notin ?bad$
using *real-interval-avoid-countable-set[OF zero-less-one count-bad]* **by** *blast*
have *disj*: $((Rz\text{-angle } t \overset{\sim}{\sim} n) ' D) \cap ((Rz\text{-angle } t \overset{\sim}{\sim} m) ' D) = \{\}$ **if** $n \neq m$ **for**
 $n\ m$
proof (*rule ccontr*)
assume $((Rz\text{-angle } t \overset{\sim}{\sim} n) ' D) \cap ((Rz\text{-angle } t \overset{\sim}{\sim} m) ' D) \neq \{\}$
then obtain $x\ y\ z$ **where** $x: x \in D$ **and** $y: y \in D$
and $z: z = (Rz\text{-angle } t \overset{\sim}{\sim} n) x z = (Rz\text{-angle } t \overset{\sim}{\sim} m) y$
by *blast*
have *False* **if** *nm*: $n < m$
proof –
let $?k = m - n$
have *k-pos*: $?k > 0$
using *nm* **by** *simp*
have *m-eq*: $m = n + ?k$
using *nm* **by** *simp*
have *eq*: $(Rz\text{-angle } t \overset{\sim}{\sim} n) x = (Rz\text{-angle } t \overset{\sim}{\sim} m) y$
using *z* **by** *simp*
have *tail*: $(Rz\text{-angle } t \overset{\sim}{\sim} m) y = (Rz\text{-angle } t \overset{\sim}{\sim} n) ((Rz\text{-angle } t \overset{\sim}{\sim} ?k) y)$
using *m-eq* **by** (*metis comp-apply funpow-add*)
have $(Rz\text{-angle } t \overset{\sim}{\sim} n) x = (Rz\text{-angle } t \overset{\sim}{\sim} n) ((Rz\text{-angle } t \overset{\sim}{\sim} ?k) y)$
using *eq tail* **by** *simp*
moreover have *inj* $(Rz\text{-angle } t \overset{\sim}{\sim} n)$
by (*rule SO3-inj[OF SO3-funpow[OF Rz-angle-in-SO3]]*)
ultimately have *x-eq*: $x = (Rz\text{-angle } t \overset{\sim}{\sim} ?k) y$
by (*meson injD*)
have $Rz\text{-angle } (real\ ?k * t) y = x$
using *x-eq Rz-angle-funpow[of ?k t]* **by** (*simp add: fun-eq-iff*)
hence $t \in \{u. Rz\text{-angle } (real\ ?k * u) y = x\}$
by *simp*
hence $t \in ?bad$
using *k-pos x y* **by** *blast*

with *t-not-bad* **show** *False*
by *contradiction*
qed
moreover **have** *False* **if** *mn: m < n*
proof –
let *?k = n - m*
have *k-pos: ?k > 0*
using *mn* **by** *simp*
have *n-eq: n = m + ?k*
using *mn* **by** *simp*
have *eq: (Rz-angle t $\widehat{\sim}$ m) y = (Rz-angle t $\widehat{\sim}$ n) x*
using *z* **by** *simp*
have *tail: (Rz-angle t $\widehat{\sim}$ n) x =*
(Rz-angle t $\widehat{\sim}$ m) ((Rz-angle t $\widehat{\sim}$?k) x)
using *n-eq* **by** *(metis comp-apply funpow-add)*
have *(Rz-angle t $\widehat{\sim}$ m) y = (Rz-angle t $\widehat{\sim}$ m) ((Rz-angle t $\widehat{\sim}$?k) x)*
using *eq tail* **by** *simp*
moreover **have** *inj (Rz-angle t $\widehat{\sim}$ m)*
by *(rule SO3-inj[OF SO3-funpow[OF Rz-angle-in-SO3]])*
ultimately **have** *y-eq: y = (Rz-angle t $\widehat{\sim}$?k) x*
by *(meson injD)*
have *Rz-angle (real ?k * t) x = y*
using *y-eq Rz-angle-funpow[of ?k t]* **by** *(simp add: fun-eq-iff)*
hence *t ∈ {u. Rz-angle (real ?k * u) x = y}*
by *simp*
hence *t ∈ ?bad*
using *k-pos x y* **by** *blast*
with *t-not-bad* **show** *False*
by *contradiction*
qed
ultimately **show** *False*
using *that* **by** *linarith*
qed
show *?thesis*
using *disj* **by** *blast*
qed

lemma *exists-absorbing-rotation:*
 $\exists R \in SO3. \forall n m. n \neq m \longrightarrow ((R \widehat{\sim} n) \text{ ‘ bad-set-D}) \cap ((R \widehat{\sim} m) \text{ ‘ bad-set-D}) = \{\}$
proof –
obtain *a* **where** *a-sphere: a ∈ sphere2* **and** *a-not-D: a ∉ bad-set-D*
and *neg-a-not-D: – a ∉ bad-set-D*
using *exists-antipodal-axis-avoiding-countable[OF bad-set-D-countable]* **by** *blast*
have *norm-eq: norm e3-3 = norm a*
using *e3-3-in-sphere2 a-sphere unfolding sphere2-def* **by** *simp*
have *card3: 2 ≤ CARD(3)*
by *simp*
obtain *S* **where** *S-orth: orthogonal-transformation S*

```

    and S-det: matrix-det (matrix S) = 1
    and S-e3: S e3-3 = a
    using rotation-exists[OF card3 norm-eq] by blast
  have S-SO3: S ∈ SO3
    using S-orth S-det unfolding SO3-def rotation-def by simp
  have invS-SO3: Hilbert-Choice.inv S ∈ SO3
    by (rule SO3-inverse[OF S-SO3])
  have S-bij: bij S
    by (rule orthogonal-transformation-bij[OF S-orth])
  have S-inj: inj S
    using S-bij by (simp add: bij-is-inj)
  have S-surj: surj S
    using S-bij by (simp add: bij-is-surj)
  have S-lin: linear S
    by (rule orthogonal-transformation-linear[OF S-orth])
  let ?D' = Hilbert-Choice.inv S ' bad-set-D
  have count-D': countable ?D'
    using bad-set-D-countable by simp
  have nonaxis-D': x $ 1 ≠ 0 ∨ x $ 2 ≠ 0 if xD: x ∈ ?D' for x
  proof (rule ccontr)
    assume ¬ (x $ 1 ≠ 0 ∨ x $ 2 ≠ 0)
    hence x1: x $ 1 = 0 and x2: x $ 2 = 0
      by auto
    from xD obtain d where dD: d ∈ bad-set-D and x-def: x = Hilbert-Choice.inv
  S d
    by auto
  have d-sphere: d ∈ sphere2
    using dD bad-set-D-subset by auto
  have x-sphere: x ∈ sphere2
    unfolding x-def
    by (rule rotation-preserves-sphere2[OF invS-SO3 d-sphere])
  have axis-cases: x = e3-3 ∨ x = - e3-3
    by (rule z-axis-sphere2-cases[OF x-sphere x1 x2])
  have Sx-d: S x = d
    using x-def S-surj by (simp add: surj-f-inv-f)
  from axis-cases show False
  proof
    assume x = e3-3
    hence d = a
      using Sx-d S-e3 by simp
    with dD a-not-D show False
      by simp
  next
    assume x = - e3-3
    hence S x = - a
      using S-lin S-e3 by (simp add: linear-neg)
    hence d = - a
      using Sx-d by simp
    with dD neg-a-not-D show False

```

by *simp*
 qed
 qed
obtain t **where** t -*disj*:
 $\forall n m. n \neq m \longrightarrow$
 $((Rz\text{-angle } t \overset{\sim}{\sim} n) \text{ ' } ?D') \cap ((Rz\text{-angle } t \overset{\sim}{\sim} m) \text{ ' } ?D') = \{\}$
using *exists-Rz-angle-absorbing*[*OF count-D' nonaxis-D'*] **by** *blast*
let $?R = S \circ Rz\text{-angle } t \circ Hilbert\text{-Choice.inv } S$
have *inner-SO3*: $Rz\text{-angle } t \circ Hilbert\text{-Choice.inv } S \in SO3$
by (*rule SO3-closed-compose*[*OF Rz-angle-in-SO3 invS-SO3*])
have *R-SO3*: $?R \in SO3$
proof –
have $S \circ (Rz\text{-angle } t \circ Hilbert\text{-Choice.inv } S) \in SO3$
by (*rule SO3-closed-compose*[*OF S-SO3 inner-SO3*])
thus *?thesis*
by (*simp add: o-assoc*)
 qed
have *R-disj*: $((?R \overset{\sim}{\sim} n) \text{ ' } bad\text{-set-}D) \cap ((?R \overset{\sim}{\sim} m) \text{ ' } bad\text{-set-}D) = \{\}$
if $nm: n \neq m$ **for** $n m$
proof –
have *n-img*: $((?R \overset{\sim}{\sim} n) \text{ ' } bad\text{-set-}D) =$
 $S \text{ ' } ((Rz\text{-angle } t \overset{\sim}{\sim} n) \text{ ' } ?D')$
by (*rule conjugate-funpow-image*[*OF S-bij*])
have *m-img*: $((?R \overset{\sim}{\sim} m) \text{ ' } bad\text{-set-}D) =$
 $S \text{ ' } ((Rz\text{-angle } t \overset{\sim}{\sim} m) \text{ ' } ?D')$
by (*rule conjugate-funpow-image*[*OF S-bij*])
have *z-disj*: $((Rz\text{-angle } t \overset{\sim}{\sim} n) \text{ ' } ?D') \cap ((Rz\text{-angle } t \overset{\sim}{\sim} m) \text{ ' } ?D') = \{\}$
using t -*disj* nm **by** *blast*
show *?thesis*
unfolding *n-img m-img*
using z -*disj S-inj* **by** (*auto dest: injD*)
 qed
from *R-SO3 R-disj* **show** *?thesis*
by *blast*
 qed

lemma *SO3-funpow-preserves-sphere2*:
assumes $R \in SO3$ **and** $x \in sphere2$
shows $(R \overset{\sim}{\sim} n) x \in sphere2$
by (*rule rotation-preserves-sphere2*[*OF SO3-funpow*[*OF assms(1)*] *assms(2)*])

lemma *funpow-Suc-image*:
 $((R \overset{\sim}{\sim} Suc\ n) \text{ ' } S) = R \text{ ' } ((R \overset{\sim}{\sim} n) \text{ ' } S)$
by *auto*

lemma *absorbing-rotation-shift*:
assumes *disj*: $\forall n m. n \neq m \longrightarrow ((R \overset{\sim}{\sim} n) \text{ ' } D) \cap ((R \overset{\sim}{\sim} m) \text{ ' } D) = \{\}$
defines $E \equiv (\bigcup n. (R \overset{\sim}{\sim} n) \text{ ' } D)$
shows $R \text{ ' } E = E - D$

```

proof
  show  $R \text{ ' } E \subseteq E - D$ 
  proof
    fix  $y$ 
    assume  $y \in R \text{ ' } E$ 
    then obtain  $x \ n \ d$  where  $x: x = (R \ \sim n) \ d$  and  $d: d \in D$  and  $y: y = R \ x$ 
      unfolding  $E\text{-def}$  by blast
    have  $y\text{-pow}: y \in (R \ \sim \text{Suc } n) \text{ ' } D$ 
      using  $x \ d \ y$  by auto
    hence  $y \in E$ 
      unfolding  $E\text{-def}$  by blast
    moreover have  $y \notin D$ 
  proof
    assume  $y \in D$ 
    hence  $y \in (R \ \sim 0) \text{ ' } D$ 
      by simp
    with  $y\text{-pow}$  have  $((R \ \sim \text{Suc } n) \text{ ' } D) \cap ((R \ \sim 0) \text{ ' } D) \neq \{\}$ 
      by blast
    moreover have  $\text{Suc } n \neq 0$ 
      by simp
    ultimately show False
      using disj by blast
  qed
  ultimately show  $y \in E - D$ 
    by simp
  qed
next
  show  $E - D \subseteq R \text{ ' } E$ 
  proof
    fix  $y$ 
    assume  $y: y \in E - D$ 
    then obtain  $n \ d$  where  $y\text{-pow}: y = (R \ \sim n) \ d$  and  $d: d \in D$ 
      unfolding  $E\text{-def}$  by blast
    from  $y$  have  $y\text{-not-}D: y \notin D$ 
      by simp
    show  $y \in R \text{ ' } E$ 
  proof (cases n)
    case 0
      with  $y\text{-pow}$   $d \ y\text{-not-}D$  show ?thesis
        by simp
    next
      case ( $\text{Suc } k$ )
        have  $(R \ \sim k) \ d \in E$ 
          unfolding  $E\text{-def}$  using  $d$  by blast
        moreover have  $y = R \ ((R \ \sim k) \ d)$ 
          using  $y\text{-pow}$   $\text{Suc}$  by simp
        ultimately show ?thesis
          by blast
  qed

```

qed
qed

lemma *sphere2-absorb-bad-set*:

$\exists P Q :: (\text{real}^3) \text{ set list. } \exists gs :: ((\text{real}^3) \Rightarrow (\text{real}^3)) \text{ list.}$
 $\text{length } P = \text{length } Q \wedge \text{length } P = \text{length } gs \wedge$
 $\text{set } gs \subseteq SO3 \wedge$
 $\text{sphere2} = (\bigcup_{i < \text{length } P. P ! i) \wedge$
 $(\text{sphere2} - \text{bad-set-D}) = (\bigcup_{i < \text{length } Q. Q ! i) \wedge$
 $(\forall i < \text{length } P. Q ! i = (gs ! i) ' (P ! i))$

proof –

obtain *R* where *R-SO3*: $R \in SO3$

and *disj*: $\forall n m. n \neq m \longrightarrow ((R \sim n) ' \text{bad-set-D}) \cap ((R \sim m) ' \text{bad-set-D}) = \{\}$

using *exists-absorbing-rotation by auto*

define *E* where $E = (\bigcup n. (R \sim n) ' \text{bad-set-D})$

have *R-E*: $R ' E = E - \text{bad-set-D}$

unfolding *E-def* **by** (*rule absorbing-rotation-shift[OF disj]*)

have *D-subset-E*: $\text{bad-set-D} \subseteq E$

proof

fix *x*

assume *x*: $x \in \text{bad-set-D}$

have $x \in (R \sim 0) ' \text{bad-set-D}$

using *x* **by** (*auto simp add: image-def*)

thus $x \in E$

unfolding *E-def* **by** *blast*

qed

have *E-subset-sphere2*: $E \subseteq \text{sphere2}$

proof

fix *x*

assume $x \in E$

then obtain *n d* where $x = (R \sim n) d$ **and** *d*: $d \in \text{bad-set-D}$

unfolding *E-def* **by** *blast*

from *d* **have** $d \in \text{sphere2}$

using *bad-set-D-subset* **by** *blast*

with *R-SO3* *x* **show** $x \in \text{sphere2}$

using *SO3-funpow-preserves-sphere2* **by** *blast*

qed

let *?P* = $[E, \text{sphere2} - E]$

let *?Q* = $[R ' E, \text{sphere2} - E]$

let *?gs* = $[R, \text{id}]$

have *P-union*: $\text{sphere2} = (\bigcup_{i < \text{length } ?P. ?P ! i)$

proof

show $\text{sphere2} \subseteq (\bigcup_{i < \text{length } ?P. ?P ! i)$

proof

fix *x*

assume *x*: $x \in \text{sphere2}$

show $x \in (\bigcup_{i < \text{length } ?P. ?P ! i)$

proof (*cases* $x \in E$)

```

    case True
    then show ?thesis
      by force
  next
  case False
  with x show ?thesis
    by force
qed
qed
next
show  $(\bigcup_{i < \text{length } ?P}. ?P ! i) \subseteq \text{sphere2}$ 
proof
  fix x
  assume x:  $x \in (\bigcup_{i < \text{length } ?P}. ?P ! i)$ 
  then obtain i where  $i < \text{length } ?P$  and xi:  $x \in ?P ! i$ 
    by blast
  have  $i = 0 \vee i = 1$ 
    using i by auto
  thus  $x \in \text{sphere2}$ 
  proof
    assume  $i = 0$ 
    with xi E-subset-sphere2 show ?thesis
      by auto
  next
    assume  $i = 1$ 
    with xi show ?thesis
      by auto
  qed
qed
qed
have Q-union:  $\text{sphere2} - \text{bad-set-D} = (\bigcup_{i < \text{length } ?Q}. ?Q ! i)$ 
proof
  show  $\text{sphere2} - \text{bad-set-D} \subseteq (\bigcup_{i < \text{length } ?Q}. ?Q ! i)$ 
  proof
    fix x
    assume x:  $x \in \text{sphere2} - \text{bad-set-D}$ 
    show  $x \in (\bigcup_{i < \text{length } ?Q}. ?Q ! i)$ 
    proof (cases  $x \in E$ )
      case True
      with x R-E show ?thesis
        by force
    next
      case False
      with x show ?thesis
        by force
    qed
  qed
qed
next
show  $(\bigcup_{i < \text{length } ?Q}. ?Q ! i) \subseteq \text{sphere2} - \text{bad-set-D}$ 

```

```

proof
  fix  $x$ 
  assume  $x: x \in (\bigcup_{i < \text{length } ?Q} ?Q ! i)$ 
  then obtain  $i$  where  $i: i < \text{length } ?Q$  and  $xi: x \in ?Q ! i$ 
    by blast
  have  $i = 0 \vee i = 1$ 
    using  $i$  by auto
  thus  $x \in \text{sphere2} - \text{bad-set-D}$ 
  proof
    assume  $i = 0$ 
    with  $xi$  R-E E-subset-sphere2 show  $?thesis$ 
      by auto
    next
    assume  $i = 1$ 
    with  $xi$  D-subset-E show  $?thesis$ 
      by auto
  qed
qed
qed
have image-eq:  $\forall i < \text{length } ?P. ?Q ! i = (?gs ! i) \text{ ' } (?P ! i)$ 
proof (intro allI impI)
  fix  $i$ 
  assume  $i: i < \text{length } ?P$ 
  have  $i = 0 \vee i = 1$ 
    using  $i$  by auto
  thus  $?Q ! i = (?gs ! i) \text{ ' } (?P ! i)$ 
    by auto
qed
show  $?thesis$ 
  apply (rule exI[of - ?P])
  apply (rule exI[of - ?Q])
  apply (rule exI[of - ?gs])
  using R-SO3 id-in-SO3 P-union Q-union image-eq
  by auto
qed

```

Combining the Hausdorff paradox with the absorption argument yields the paradoxical decomposition of the full sphere.

lemma *sphere-indexed-union-append-singleton*:

fixes $P :: 'a \text{ set list}$ **and** $S :: 'a \text{ set}$

shows $(\bigcup_{i < \text{length } (P @ [S])} (P @ [S]) ! i) =$
 $(\bigcup_{i < \text{length } P} P ! i) \cup S$

proof –

have $(\bigcup_{i < \text{length } (P @ [S])} (P @ [S]) ! i) =$
 $(\bigcup_{i < \text{Suc } (\text{length } P)} (P @ [S]) ! i)$

by *simp*

also have $\dots =$

$(\bigcup_{i < \text{length } P} (P @ [S]) ! i) \cup (P @ [S]) ! \text{length } P$

by (*simp add: lessThan-Suc Un-commute*)

also have $\dots = (\bigcup_{i < \text{length } P} P ! i) \cup S$
proof –
have $(\bigcup_{i < \text{length } P} (P @ [S]) ! i) = (\bigcup_{i < \text{length } P} P ! i)$
by (rule *SUP-cong*) (simp-all add: *nth-append*)
thus ?thesis **by** simp
qed
finally show ?thesis .
qed

lemma *sphere-indexed-image-union-append-singleton*:
fixes $P :: 'a \text{ set list}$ **and** $G :: ('a \Rightarrow 'b) \text{ list}$ **and** $S :: 'a \text{ set}$
and $g :: 'a \Rightarrow 'b$
assumes $\text{length } P = \text{length } G$
shows $(\bigcup_{i < \text{length } P} (P @ [S]). (G @ [g]) ! i \text{ ' } ((P @ [S]) ! i)) =$
 $(\bigcup_{i < \text{length } P} G ! i \text{ ' } (P ! i)) \cup g \text{ ' } S$
proof –
have $(\bigcup_{i < \text{length } P} (P @ [S]). (G @ [g]) ! i \text{ ' } ((P @ [S]) ! i)) =$
 $(\bigcup_{i < \text{Suc } (\text{length } P)}. (G @ [g]) ! i \text{ ' } ((P @ [S]) ! i))$
by simp
also have $\dots =$
 $(\bigcup_{i < \text{length } P} (G @ [g]) ! i \text{ ' } ((P @ [S]) ! i)) \cup$
 $(G @ [g]) ! \text{length } P \text{ ' } ((P @ [S]) ! \text{length } P)$
by (simp add: *lessThan-Suc Un-commute*)
also have $\dots = (\bigcup_{i < \text{length } P} G ! i \text{ ' } (P ! i)) \cup g \text{ ' } S$
using *assms* **by** (simp add: *nth-append*)
finally show ?thesis .
qed

lemma *sphere2-absorb-cover*:
assumes $R\text{-}SO3: R \in SO3$
and *disj*: $\forall n \ m. n \neq m \longrightarrow ((R \sim n) \text{ ' } \text{bad-set-}D) \cap ((R \sim m) \text{ ' } \text{bad-set-}D) =$
 $\{\}$
and *len*: $\text{length } P = \text{length } g$
and *g-SO3*: $\text{set } g \subseteq SO3$
and *P-sub*: $\forall i < \text{length } P. P ! i \subseteq \text{sphere2} - \text{bad-set-}D$
and *X-cover*: $\text{sphere2} - \text{bad-set-}D = (\bigcup_{i < \text{length } P} (g ! i) \text{ ' } (P ! i))$
shows $\exists P' :: (\text{real}^3) \text{ set list}. \exists g' :: ((\text{real}^3) \Rightarrow (\text{real}^3)) \text{ list}.$
 $\text{length } P' = \text{length } g' \wedge$
 $\text{set } g' \subseteq SO3 \wedge$
 $(\forall i < \text{length } P'. P' ! i \subseteq \text{sphere2}) \wedge$
 $\text{sphere2} = (\bigcup_{i < \text{length } P'. P' ! i) \wedge$
 $\text{sphere2} = (\bigcup_{i < \text{length } P'. (g' ! i) \text{ ' } (P' ! i))$
proof –
define E **where** $E = (\bigcup n. (R \sim n) \text{ ' } \text{bad-set-}D)$
have $R\text{-}E: R \text{ ' } E = E - \text{bad-set-}D$
unfolding *E-def* **by** (rule *absorbing-rotation-shift[OF disj]*)
have $D\text{-subset-}E: \text{bad-set-}D \subseteq E$
proof
fix x

```

assume  $x: x \in \text{bad-set-}D$ 
have  $x \in (R \overset{\sim}{\sim} 0)$  ‘bad-set-}D
  using  $x$  by (auto simp add: image-def)
thus  $x \in E$ 
  unfolding E-def by blast
qed
have E-subset-sphere2:  $E \subseteq \text{sphere2}$ 
proof
  fix  $x$ 
  assume  $x \in E$ 
  then obtain  $n\ d$  where  $x: x = (R \overset{\sim}{\sim} n)\ d$  and  $d: d \in \text{bad-set-}D$ 
    unfolding E-def by blast
  from  $d$  have  $d \in \text{sphere2}$ 
    using bad-set-}D-subset by blast
  with R-SO3  $x$  show  $x \in \text{sphere2}$ 
    using SO3-funpow-preserves-sphere2 by blast
qed
have invR-SO3: Hilbert-Choice.inv  $R \in \text{SO3}$ 
  by (rule SO3-inverse[OF R-SO3])
have R-inj: inj  $R$ 
  by (rule SO3-inj[OF R-SO3])
define  $P0$  where  $P0 = P @ P$ 
define  $g0$  where  $g0 = \text{map } (\lambda h. \text{Hilbert-Choice.inv } R \circ h)\ g @ g$ 
define  $\text{Rest}$  where  $\text{Rest} = \text{sphere2} - (\bigcup_{i < \text{length } P0}. P0 ! i)$ 
define  $P'$  where  $P' = P0 @ [\text{Rest}]$ 
define  $g'$  where  $g' = g0 @ [\text{id}]$ 
have  $\text{len0}$ :  $\text{length } P0 = \text{length } g0$ 
  using  $\text{len}$  by (simp add: P0-def g0-def)
have  $\text{len}'$ :  $\text{length } P' = \text{length } g'$ 
  using  $\text{len0}$  by (simp add: P'-def g'-def)
have  $g0\text{-SO3}$ :  $\text{set } g0 \subseteq \text{SO3}$ 
proof
  fix  $h$ 
  assume  $h \in \text{set } g0$ 
  hence  $h \in (\lambda k. \text{Hilbert-Choice.inv } R \circ k)$  ‘set }g \vee h \in \text{set } g
    by (auto simp: g0-def)
  thus  $h \in \text{SO3}$ 
proof
  assume  $h \in (\lambda k. \text{Hilbert-Choice.inv } R \circ k)$  ‘set }g
  then obtain  $k$  where  $k: k \in \text{set } g$  and  $h: h = \text{Hilbert-Choice.inv } R \circ k$ 
    by blast
  have  $k\text{-SO3}$ :  $k \in \text{SO3}$ 
    using  $g\text{-SO3 } k$  by auto
  show ?thesis
    unfolding  $h$  by (rule SO3-closed-compose[OF invR-SO3 k-SO3])
next
  assume  $h \in \text{set } g$ 
  thus ?thesis
    using  $g\text{-SO3}$  by auto

```

```

qed
qed
have g'-SO3: set g' ⊆ SO3
  using g0-SO3 id-in-SO3 by (auto simp: g'-def)
have P0-sub: ∀ i < length P0. P0 ! i ⊆ sphere2
proof (intro allI impI)
  fix i
  assume i: i < length P0
  show P0 ! i ⊆ sphere2
  proof (cases i < length P)
    case True
    have P0-i: P0 ! i = P ! i
      using True by (simp add: P0-def nth-append)
    have P ! i ⊆ sphere2
      using P-sub True by blast
    thus ?thesis
      by (simp add: P0-i)
  next
  case False
  hence i2: i - length P < length P
    using i by (simp add: P0-def)
  have P0 ! i = P ! (i - length P)
    using False i by (simp add: P0-def nth-append)
  with P-sub i2 show ?thesis
    by auto
  qed
qed
qed
have P'-sub: ∀ i < length P'. P' ! i ⊆ sphere2
proof (intro allI impI)
  fix i
  assume i: i < length P'
  show P' ! i ⊆ sphere2
  proof (cases i < length P0)
    case True
    have P'-i: P' ! i = P0 ! i
      using True by (simp add: P'-def nth-append)
    have P0 ! i ⊆ sphere2
      using P0-sub True by blast
    thus ?thesis
      by (simp add: P'-i)
  next
  case False
  with i have i = length P0
    by (simp add: P'-def)
  thus ?thesis
    by (auto simp: P'-def Rest-def)
  qed
qed
have P'-cover: sphere2 = (⋃ i < length P'. P' ! i)

```

proof –
have $(\bigcup_{i < \text{length } P'} P' ! i) =$
 $(\bigcup_{i < \text{length } P0} P0 ! i) \cup \text{Rest}$
unfolding P' -def **by** (rule sphere-indexed-union-append-singleton)
also have $\dots = \text{sphere2}$
proof
show $(\bigcup_{i < \text{length } P0} P0 ! i) \cup \text{Rest} \subseteq \text{sphere2}$
using $P0$ -sub **by** (auto simp: Rest-def)
show $\text{sphere2} \subseteq (\bigcup_{i < \text{length } P0} P0 ! i) \cup \text{Rest}$
by (auto simp: Rest-def)
qed
finally show ?thesis
by simp
qed
have image0-cover: $\text{sphere2} \subseteq (\bigcup_{i < \text{length } P0} (g0 ! i) \text{ ‘ } (P0 ! i))$
proof
fix y
assume y -sphere: $y \in \text{sphere2}$
show $y \in (\bigcup_{i < \text{length } P0} (g0 ! i) \text{ ‘ } (P0 ! i))$
proof (cases $y \in E$)
case True
hence Ry -in-RE: $R y \in R \text{ ‘ } E$
by blast
have Ry -E-minus-D: $R y \in E - \text{bad-set-D}$
using Ry -in-RE R -E **by** simp
have Ry -X: $R y \in \text{sphere2} - \text{bad-set-D}$
proof
show $R y \in \text{sphere2}$
using Ry -E-minus-D E -subset-sphere2 **by** blast
show $R y \notin \text{bad-set-D}$
using Ry -E-minus-D **by** simp
qed
then obtain $i x$ **where** $i: i < \text{length } P$ **and** $x: x \in P ! i$
and $Ry: R y = (g ! i) x$
using X-cover **by** blast
have $i0: i < \text{length } P0$
using i **by** (simp add: P0-def)
have $g0$ - $i: g0 ! i = \text{Hilbert-Choice.inv } R \circ (g ! i)$
using i len **by** (simp add: g0-def nth-append)
have $P0$ - $i: P0 ! i = P ! i$
using i **by** (simp add: P0-def nth-append)
have $(g0 ! i) x = y$
proof –
have inv- $Ry: \text{Hilbert-Choice.inv } R (R y) = y$
by (rule inv-f-f[OF R-inj])
have $(g0 ! i) x = \text{Hilbert-Choice.inv } R ((g ! i) x)$
by (simp add: g0-i)
also have $\dots = \text{Hilbert-Choice.inv } R (R y)$
using Ry **by** simp

```

    also have ... = y
      by (rule inv-Ry)
    finally show ?thesis .
  qed
  moreover have (g0 ! i) x ∈ (g0 ! i) ‘ (P0 ! i)
    using x P0-i by blast
  ultimately show ?thesis
    using i0 by blast
next
case False
have y-X: y ∈ sphere2 – bad-set-D
proof
  show y ∈ sphere2
    by (rule y-sphere)
  show y ∉ bad-set-D
    using False D-subset-E by auto
qed
then obtain i x where i: i < length P and x: x ∈ P ! i
  and y-eq: y = (g ! i) x
  using X-cover by blast
let ?j = length P + i
have j0: ?j < length P0
  using i by (simp add: P0-def)
have g0-j: g0 ! ?j = g ! i
  using i len by (simp add: g0-def nth-append)
have P0-j: P0 ! ?j = P ! i
  using i by (simp add: P0-def nth-append)
show ?thesis
proof –
  have (g0 ! ?j) x ∈ (g0 ! ?j) ‘ (P0 ! ?j)
    using x P0-j by blast
  moreover have (g0 ! ?j) x = y
    using y-eq g0-j by simp
  ultimately show ?thesis
    using j0 by blast
qed
qed
qed
have image0-subset: (⋃ i < length P0. (g0 ! i) ‘ (P0 ! i)) ⊆ sphere2
proof
  fix y
  assume y ∈ (⋃ i < length P0. (g0 ! i) ‘ (P0 ! i))
  then obtain i x where i: i < length P0 and x: x ∈ P0 ! i
    and y: y = (g0 ! i) x
    by blast
  have g0-i: g0 ! i ∈ SO3
  proof –
    have g0 ! i ∈ set g0
      using i len0 by (simp add: nth-mem)

```

thus *?thesis*
using *g0-SO3* **by** *blast*
qed
have *x-sphere: x ∈ sphere2*
using *P0-sub i x* **by** *auto*
show *y ∈ sphere2*
using *rotation-preserves-sphere2[OF g0-i x-sphere]* **by** *simp*
qed
have *image0-eq: sphere2 = (⋃ i < length P0. (g0 ! i) ‘ (P0 ! i))*
proof
show *sphere2 ⊆ (⋃ i < length P0. (g0 ! i) ‘ (P0 ! i))*
by *(rule image0-cover)*
show *(⋃ i < length P0. (g0 ! i) ‘ (P0 ! i)) ⊆ sphere2*
by *(rule image0-subset)*
qed
have *image-cover: sphere2 = (⋃ i < length P'. (g' ! i) ‘ (P' ! i))*
proof –
have *(⋃ i < length P'. (g' ! i) ‘ (P' ! i)) =*
(⋃ i < length P0. (g0 ! i) ‘ (P0 ! i)) ∪ id ‘ Rest
unfolding *P'-def g'-def*
by *(rule sphere-indexed-image-union-append-singleton[OF len0])*
also have *... = sphere2*
proof –
have *id ‘ Rest ⊆ sphere2*
by *(auto simp: Rest-def)*
with *image0-eq* **show** *?thesis*
by *blast*
qed
finally show *?thesis*
by *simp*
qed
show *?thesis*
proof *(intro exI conjI)*
show *length P' = length g'*
by *(rule len')*
show *set g' ⊆ SO3*
by *(rule g'-SO3)*
show *∀ i < length P'. P' ! i ⊆ sphere2*
by *(rule P'-sub)*
show *sphere2 = (⋃ i < length P'. P' ! i)*
by *(rule P'-cover)*
show *sphere2 = (⋃ i < length P'. (g' ! i) ‘ (P' ! i))*
by *(rule image-cover)*
qed
qed

theorem *sphere2-paradoxical-strong:*

$\exists P Q :: (\text{real}^3) \text{ set list. } \exists gP gQ :: ((\text{real}^3) \Rightarrow (\text{real}^3)) \text{ list.}$
 $\text{length } P = \text{length } gP \wedge \text{length } Q = \text{length } gQ \wedge$

```

    set gP ⊆ SO3 ∧ set gQ ⊆ SO3 ∧
    pairwise-disjoint (P @ Q) ∧
    pairwise-disjoint (map2 (λg A. g ' A) gP P) ∧
    pairwise-disjoint (map2 (λg A. g ' A) gQ Q) ∧
    (∀ i < length P. P ! i ⊆ sphere2) ∧
    (∀ i < length Q. Q ! i ⊆ sphere2) ∧
    sphere2 = (⋃ i < length P. P ! i) ∪ (⋃ i < length Q. Q ! i) ∧
    sphere2 = (⋃ i < length P. (gP ! i) ' (P ! i)) ∧
    sphere2 = (⋃ i < length Q. (gQ ! i) ' (Q ! i))
proof -
  obtain R where R-SO3: R ∈ SO3
  and disj: ∀ n m. n ≠ m → ((R ~ n) ' bad-set-D) ∩ ((R ~ m) ' bad-set-D) =
  {}
  using exists-absorbing-rotation by blast
  define E where E = (⋃ n. (R ~ n) ' bad-set-D)
  have R-E: R ' E = E - bad-set-D
  unfolding E-def by (rule absorbing-rotation-shift[OF disj])
  have D-subset-E: bad-set-D ⊆ E
proof
  fix x
  assume x: x ∈ bad-set-D
  have x ∈ (R ~ 0) ' bad-set-D
  using x by (auto simp add: image-def)
  thus x ∈ E
  unfolding E-def by blast
qed
have E-subset-sphere2: E ⊆ sphere2
proof
  fix x
  assume x ∈ E
  then obtain n d where x: x = (R ~ n) d and d: d ∈ bad-set-D
  unfolding E-def by blast
  from d have d ∈ sphere2
  using bad-set-D-subset by blast
  with R-SO3 x show x ∈ sphere2
  using SO3-funpow-preserves-sphere2 by blast
qed
have invR-SO3: Hilbert-Choice.inv R ∈ SO3
  by (rule SO3-inverse[OF R-SO3])
have R-inj: inj R
  by (rule SO3-inj[OF R-SO3])
let ?X = sphere2 - bad-set-D
let ?Y = sphere2
let ?A = [E, sphere2 - E]
let ?B = [R ' E, sphere2 - E]
let ?e = [R, id]
let ?t = [Hilbert-Choice.inv R, id]
have A-cover: ?Y = (⋃ k < length ?A. ?A ! k)
  using E-subset-sphere2 by (auto simp: lessThan-Suc)

```

```

have A-disj: pairwise-disjoint ?A
  unfolding pairwise-disjoint-def
proof (intro allI impI)
  fix i j
  assume i: i < length ?A and j: j < length ?A and ij: i ≠ j
  have  $(i = 0 \wedge j = 1) \vee (i = 1 \wedge j = 0)$ 
    using i j ij by auto
  thus  $?A ! i \cap ?A ! j = \{\}$ 
    by auto
qed
have B-cover: ?X = (∪ k < length ?B. ?B ! k)
proof
  show  $?X \subseteq (\bigcup k < \text{length } ?B. ?B ! k)$ 
  proof
    fix x
    assume x: x ∈ ?X
    show  $x \in (\bigcup k < \text{length } ?B. ?B ! k)$ 
    proof (cases x ∈ E)
      case True
      with x R-E show ?thesis
        by force
      next
      case False
      with x show ?thesis
        by force
    qed
  qed
  show  $(\bigcup k < \text{length } ?B. ?B ! k) \subseteq ?X$ 
  proof
    fix x
    assume x: x ∈ (∪ k < length ?B. ?B ! k)
    then obtain k where k: k < length ?B and xk: x ∈ ?B ! k
      by blast
    have  $k = 0 \vee k = 1$ 
      using k by auto
    thus  $x \in ?X$ 
  proof
    assume k = 0
    with xk R-E E-subset-sphere2 show ?thesis
      by auto
    next
    assume k = 1
    with xk D-subset-E show ?thesis
      by auto
  qed
qed
qed
have B-disj: pairwise-disjoint ?B
  unfolding pairwise-disjoint-def

```

```

proof (intro allI impI)
  fix  $i\ j$ 
  assume  $i: i < \text{length } ?B$  and  $j: j < \text{length } ?B$  and  $ij: i \neq j$ 
  have  $(i = 0 \wedge j = 1) \vee (i = 1 \wedge j = 0)$ 
    using  $i\ j\ ij$  by auto
  thus  $?B ! i \cap ?B ! j = \{\}$ 
    using R-E by auto
qed
have e-left:  $\bigwedge k\ x. \llbracket k < \text{length } ?A; x \in ?A ! k \rrbracket$ 
   $\implies (?e ! k)\ x \in ?B ! k \wedge (?t ! k)\ ((?e ! k)\ x) = x$ 
proof -
  fix  $k\ x$ 
  assume  $k: k < \text{length } ?A$  and  $x: x \in ?A ! k$ 
  have  $k = 0 \vee k = 1$ 
    using  $k$  by auto
  thus  $(?e ! k)\ x \in ?B ! k \wedge (?t ! k)\ ((?e ! k)\ x) = x$ 
  proof
    assume  $k = 0$ 
    with  $x$  show ?thesis
      using inv-f-f[OF R-inj, of x] by auto
  next
    assume  $k = 1$ 
    with  $x$  show ?thesis
      by auto
  qed
qed
have t-left:  $\bigwedge k\ y. \llbracket k < \text{length } ?B; y \in ?B ! k \rrbracket$ 
   $\implies (?t ! k)\ y \in ?A ! k \wedge (?e ! k)\ ((?t ! k)\ y) = y$ 
proof -
  fix  $k\ y$ 
  assume  $k: k < \text{length } ?B$  and  $y: y \in ?B ! k$ 
  have  $k = 0 \vee k = 1$ 
    using  $k$  by auto
  thus  $(?t ! k)\ y \in ?A ! k \wedge (?e ! k)\ ((?t ! k)\ y) = y$ 
  proof
    assume  $k = 0$ 
    then obtain  $x$  where  $x: x \in E$  and  $y\text{-eq}: y = R\ x$ 
      using  $y$  by auto
    have inv-eq: Hilbert-Choice.inv R y = x
      using  $y\text{-eq}$  inv-f-f[OF R-inj, of x] by simp
    with  $x\ y\text{-eq}$   $\langle k = 0 \rangle$  show ?thesis
      by simp
  next
    assume  $k = 1$ 
    with  $y$  show ?thesis
      by auto
  qed
qed

```

```

from hausdorff-paradox-rot-strong obtain P Q :: (real3) set list
and gP gQ :: ((real3) ⇒ (real3)) list
where lenP: length P = length gP
and lenQ: length Q = length gQ
and gP-SO3: set gP ⊆ SO3
and gQ-SO3: set gQ ⊆ SO3
and source-disj: pairwise-disjoint (P @ Q)
and imageP-disj: pairwise-disjoint (map2 (λg A. g ‘ A) gP P)
and imageQ-disj: pairwise-disjoint (map2 (λg A. g ‘ A) gQ Q)
and P-sub: ∀ i < length P. P ! i ⊆ ?X
and Q-sub: ∀ i < length Q. Q ! i ⊆ ?X
and source-cover: ?X = (⋃ i < length P. P ! i) ∪ (⋃ i < length Q. Q ! i)
and imageP-cover: ?X = (⋃ i < length P. (gP ! i) ‘ (P ! i))
and imageQ-cover: ?X = (⋃ i < length Q. (gQ ! i) ‘ (Q ! i))
by auto
have gP-inj: ∧ i. i < length P ⇒ inj (gP ! i)
proof –
  fix i assume i: i < length P
  have gP ! i ∈ SO3
  proof –
    have i < length gP
    using lenP i by simp
    hence gP ! i ∈ set gP
    by (rule nth-mem)
    thus ?thesis
    using gP-SO3 by blast
  qed
  thus inj (gP ! i)
  by (rule SO3-inj)
qed
have gQ-inj: ∧ i. i < length Q ⇒ inj (gQ ! i)
proof –
  fix i assume i: i < length Q
  have gQ ! i ∈ SO3
  proof –
    have i < length gQ
    using lenQ i by simp
    hence gQ ! i ∈ set gQ
    by (rule nth-mem)
    thus ?thesis
    using gQ-SO3 by blast
  qed
  thus inj (gQ ! i)
  by (rule SO3-inj)
qed
have mapsP: ∧ k i l. [k < length ?A; i < length P; l < length ?B]
  ⇒ transfer-map ?t gP ?e k i l ∈ SO3
proof –
  fix k i l

```

```

assume  $k: k < \text{length } ?A$  and  $i: i < \text{length } P$  and  $l: l < \text{length } ?B$ 
have  $gi: gP ! i \in SO3$ 
proof –
  have  $i < \text{length } gP$ 
    using  $\text{lenP } i$  by  $\text{simp}$ 
  hence  $gP ! i \in \text{set } gP$ 
    by  $(\text{rule } \text{nth-mem})$ 
  thus  $?thesis$ 
    using  $gP\text{-}SO3$  by  $\text{blast}$ 
qed
have  $ek: ?e ! k \in SO3$ 
proof –
  have  $k = 0 \vee k = 1$ 
    using  $k$  by  $\text{auto}$ 
  thus  $?thesis$ 
    using  $R\text{-}SO3 \text{ id-in-}SO3$  by  $\text{auto}$ 
qed
have  $tl: ?t ! l \in SO3$ 
proof –
  have  $l = 0 \vee l = 1$ 
    using  $l$  by  $\text{auto}$ 
  thus  $?thesis$ 
    using  $\text{invR-}SO3 \text{ id-in-}SO3$  by  $\text{auto}$ 
qed
have  $\text{left}: (?t ! l) \circ (gP ! i) \in SO3$ 
  by  $(\text{rule } SO3\text{-closed-compose}[OF \text{ tl } gi])$ 
show  $\text{transfer-map } ?t \ gP \ ?e \ k \ i \ l \in SO3$ 
  unfolding  $\text{transfer-map-def}$ 
  by  $(\text{rule } SO3\text{-closed-compose}[OF \text{ left } ek])$ 
qed
have  $\text{mapsQ}: \bigwedge k \ i \ l. \llbracket k < \text{length } ?A; i < \text{length } Q; l < \text{length } ?B \rrbracket$ 
   $\implies \text{transfer-map } ?t \ gQ \ ?e \ k \ i \ l \in SO3$ 
proof –
  fix  $k \ i \ l$ 
assume  $k: k < \text{length } ?A$  and  $i: i < \text{length } Q$  and  $l: l < \text{length } ?B$ 
have  $gi: gQ ! i \in SO3$ 
proof –
  have  $i < \text{length } gQ$ 
    using  $\text{lenQ } i$  by  $\text{simp}$ 
  hence  $gQ ! i \in \text{set } gQ$ 
    by  $(\text{rule } \text{nth-mem})$ 
  thus  $?thesis$ 
    using  $gQ\text{-}SO3$  by  $\text{blast}$ 
qed
have  $ek: ?e ! k \in SO3$ 
proof –
  have  $k = 0 \vee k = 1$ 
    using  $k$  by  $\text{auto}$ 
  thus  $?thesis$ 

```

```

    using R-SO3 id-in-SO3 by auto
  qed
  have tl: ?t ! l ∈ SO3
  proof -
    have l = 0 ∨ l = 1
      using l by auto
    thus ?thesis
      using invR-SO3 id-in-SO3 by auto
  qed
  have left: (?t ! l) ◦ (gQ ! i) ∈ SO3
    by (rule SO3-closed-compose[OF tl gi])
  show transfer-map ?t gQ ?e k i l ∈ SO3
    unfolding transfer-map-def
    by (rule SO3-closed-compose[OF left ek])
  qed
  have lenAB: length ?A = length ?B
    by simp
  from transfer-partitioned-paradox[
    OF lenAB A-cover A-disj B-cover B-disj e-left t-left lenP lenQ source-disj
    source-cover imageP-disj imageQ-disj imageP-cover imageQ-cover gP-inj
  gQ-inj
  mapsP mapsQ]
  obtain P' Q' :: (real^3) set list and hP hQ :: ((real^3) ⇒ (real^3)) list
  where lenP': length P' = length hP
    and lenQ': length Q' = length hQ
    and hP-SO3: set hP ⊆ SO3
    and hQ-SO3: set hQ ⊆ SO3
    and source-disj': pairwise-disjoint (P' @ Q')
    and imageP-disj': pairwise-disjoint (map2 (λg A. g ' A) hP P')
    and imageQ-disj': pairwise-disjoint (map2 (λg A. g ' A) hQ Q')
    and source-cover': ?Y = (⋃ i < length P'. P' ! i) ∪ (⋃ i < length Q'. Q' ! i)
    and imageP-cover': ?Y = (⋃ i < length P'. (hP ! i) ' (P' ! i))
    and imageQ-cover': ?Y = (⋃ i < length Q'. (hQ ! i) ' (Q' ! i))
  by auto
  have P'-sub: ∀ i < length P'. P' ! i ⊆ sphere2
    using source-cover' by blast
  have Q'-sub: ∀ i < length Q'. Q' ! i ⊆ sphere2
    using source-cover' by blast
  show ?thesis
  proof (intro exI conjI)
    show length P' = length hP
      by (rule lenP')
    show length Q' = length hQ
      by (rule lenQ')
    show set hP ⊆ SO3
      by (rule hP-SO3)
    show set hQ ⊆ SO3
      by (rule hQ-SO3)
    show pairwise-disjoint (P' @ Q')

```

by (rule source-disj')
 show pairwise-disjoint (map2 (λg A. g ' A) hP P')
 by (rule imageP-disj')
 show pairwise-disjoint (map2 (λg A. g ' A) hQ Q')
 by (rule imageQ-disj')
 show $\forall i < \text{length } P'. P' ! i \subseteq \text{sphere2}$
 by (rule P'-sub)
 show $\forall i < \text{length } Q'. Q' ! i \subseteq \text{sphere2}$
 by (rule Q'-sub)
 show $\text{sphere2} = (\bigcup i < \text{length } P'. P' ! i) \cup (\bigcup i < \text{length } Q'. Q' ! i)$
 by (rule source-cover')
 show $\text{sphere2} = (\bigcup i < \text{length } P'. (hP ! i) ' (P' ! i))$
 by (rule imageP-cover')
 show $\text{sphere2} = (\bigcup i < \text{length } Q'. (hQ ! i) ' (Q' ! i))$
 by (rule imageQ-cover')
 qed
 qed

theorem sphere2-paradoxical:

$\exists P Q :: (\text{real}^3) \text{ set list. } \exists gP gQ :: ((\text{real}^3) \Rightarrow (\text{real}^3)) \text{ list.}$
 $\text{length } P = \text{length } gP \wedge \text{length } Q = \text{length } gQ \wedge$
 $\text{set } gP \subseteq SO3 \wedge \text{set } gQ \subseteq SO3 \wedge$
 $(\forall i < \text{length } P. P ! i \subseteq \text{sphere2}) \wedge$
 $(\forall i < \text{length } Q. Q ! i \subseteq \text{sphere2}) \wedge$
 $\text{sphere2} = (\bigcup i < \text{length } P. P ! i) \wedge$
 $\text{sphere2} = (\bigcup i < \text{length } Q. Q ! i) \wedge$
 $\text{sphere2} = (\bigcup i < \text{length } P. (gP ! i) ' (P ! i)) \wedge$
 $\text{sphere2} = (\bigcup i < \text{length } Q. (gQ ! i) ' (Q ! i))$

proof –

obtain R where R-SO3: $R \in SO3$

and disj: $\forall n m. n \neq m \longrightarrow ((R \sim n) ' \text{bad-set-D}) \cap ((R \sim m) ' \text{bad-set-D}) = \{\}$

using exists-absorbing-rotation by blast

from hausdorff-paradox **obtain** P Q :: $(\text{real}^3) \text{ set list}$

and hP hQ :: $(\text{real}^3) \Rightarrow (\text{real}^3) \text{ list}$

where hd:

$\text{length } P = \text{length } hP \wedge \text{length } Q = \text{length } hQ \wedge$
 $\text{set } hP \subseteq SO3 \wedge \text{set } hQ \subseteq SO3 \wedge$
 $(\forall i < \text{length } P. P ! i \subseteq \text{sphere2} - \text{bad-set-D}) \wedge$
 $(\forall i < \text{length } Q. Q ! i \subseteq \text{sphere2} - \text{bad-set-D}) \wedge$
 $\text{sphere2} - \text{bad-set-D} = (\bigcup i < \text{length } P. (hP ! i) ' (P ! i)) \wedge$
 $\text{sphere2} - \text{bad-set-D} = (\bigcup i < \text{length } Q. (hQ ! i) ' (Q ! i))$

by (elim exE)

from hd **have** lenP: $\text{length } P = \text{length } hP$

by (elim conjE)

from hd **have** lenQ: $\text{length } Q = \text{length } hQ$

by (elim conjE)

from hd **have** hP-SO3: $\text{set } hP \subseteq SO3$

by (elim conjE)

from hd **have** $hQ\text{-}SO3$: $set\ hQ \subseteq SO3$
by $(elim\ conjE)$
from hd **have** $P\text{-}sub$: $\forall i < length\ P. P\ !\ i \subseteq sphere2 - bad\text{-}set\text{-}D$
by $(elim\ conjE)$
from hd **have** $Q\text{-}sub$: $\forall i < length\ Q. Q\ !\ i \subseteq sphere2 - bad\text{-}set\text{-}D$
by $(elim\ conjE)$
from hd **have** $coverP$: $sphere2 - bad\text{-}set\text{-}D = (\bigcup_{i < length\ P} (hP\ !\ i) \text{ ' } (P\ !\ i))$
by $(elim\ conjE)$
from hd **have** $coverQ$: $sphere2 - bad\text{-}set\text{-}D = (\bigcup_{i < length\ Q} (hQ\ !\ i) \text{ ' } (Q\ !\ i))$
by $(elim\ conjE)$
from $sphere2\text{-}absorb\text{-}cover[OF\ R\text{-}SO3\ disj\ lenP\ hP\text{-}SO3\ P\text{-}sub\ coverP]$
obtain $P'\ hP'$ **where** $P'\text{-}all$:
 $length\ P' = length\ hP' \wedge$
 $set\ hP' \subseteq SO3 \wedge$
 $(\forall i < length\ P'. P'\ !\ i \subseteq sphere2) \wedge$
 $sphere2 = (\bigcup_{i < length\ P'} P'\ !\ i) \wedge$
 $sphere2 = (\bigcup_{i < length\ P'} (hP'\ !\ i) \text{ ' } (P'\ !\ i))$
by $(elim\ exE)$
from $P'\text{-}all$ **have** $P'\text{-}len$: $length\ P' = length\ hP'$
by $(elim\ conjE)$
from $P'\text{-}all$ **have** $P'\text{-}SO3$: $set\ hP' \subseteq SO3$
by $(elim\ conjE)$
from $P'\text{-}all$ **have** $P'\text{-}sub$: $\forall i < length\ P'. P'\ !\ i \subseteq sphere2$
by $(elim\ conjE)$
from $P'\text{-}all$ **have** $P'\text{-}src$: $sphere2 = (\bigcup_{i < length\ P'} P'\ !\ i)$
by $(elim\ conjE)$
from $P'\text{-}all$ **have** $P'\text{-}img$: $sphere2 = (\bigcup_{i < length\ P'} (hP'\ !\ i) \text{ ' } (P'\ !\ i))$
by $(elim\ conjE)$
from $sphere2\text{-}absorb\text{-}cover[OF\ R\text{-}SO3\ disj\ lenQ\ hQ\text{-}SO3\ Q\text{-}sub\ coverQ]$
obtain $Q'\ hQ'$ **where** $Q'\text{-}all$:
 $length\ Q' = length\ hQ' \wedge$
 $set\ hQ' \subseteq SO3 \wedge$
 $(\forall i < length\ Q'. Q'\ !\ i \subseteq sphere2) \wedge$
 $sphere2 = (\bigcup_{i < length\ Q'} Q'\ !\ i) \wedge$
 $sphere2 = (\bigcup_{i < length\ Q'} (hQ'\ !\ i) \text{ ' } (Q'\ !\ i))$
by $(elim\ exE)$
from $Q'\text{-}all$ **have** $Q'\text{-}len$: $length\ Q' = length\ hQ'$
by $(elim\ conjE)$
from $Q'\text{-}all$ **have** $Q'\text{-}SO3$: $set\ hQ' \subseteq SO3$
by $(elim\ conjE)$
from $Q'\text{-}all$ **have** $Q'\text{-}sub$: $\forall i < length\ Q'. Q'\ !\ i \subseteq sphere2$
by $(elim\ conjE)$
from $Q'\text{-}all$ **have** $Q'\text{-}src$: $sphere2 = (\bigcup_{i < length\ Q'} Q'\ !\ i)$
by $(elim\ conjE)$
from $Q'\text{-}all$ **have** $Q'\text{-}img$: $sphere2 = (\bigcup_{i < length\ Q'} (hQ'\ !\ i) \text{ ' } (Q'\ !\ i))$
by $(elim\ conjE)$
show $?thesis$
proof $(intro\ exI\ conjI)$
show $length\ P' = length\ hP'$

```

    by (rule P'-len)
  show length Q' = length hQ'
    by (rule Q'-len)
  show set hP' ⊆ SO3
    by (rule P'-SO3)
  show set hQ' ⊆ SO3
    by (rule Q'-SO3)
  show ∀ i < length P'. P' ! i ⊆ sphere2
    by (rule P'-sub)
  show ∀ i < length Q'. Q' ! i ⊆ sphere2
    by (rule Q'-sub)
  show sphere2 = (⋃ i < length P'. P' ! i)
    by (rule P'-src)
  show sphere2 = (⋃ i < length Q'. Q' ! i)
    by (rule Q'-src)
  show sphere2 = (⋃ i < length P'. (hP' ! i) ' (P' ! i))
    by (rule P'-img)
  show sphere2 = (⋃ i < length Q'. (hQ' ! i) ' (Q' ! i))
    by (rule Q'-img)
qed
qed
end

```

```

theory Ball-Decomposition
  imports Sphere-Decomposition
begin

```

Radial cone construction: take a sphere subset and form the open radial cone in the ball.

definition *cone-of* :: (real^3) set \Rightarrow (real^3) set **where**
cone-of S = {t *_R x | t x. 0 < t ∧ t ≤ 1 ∧ x ∈ S}

lemma *cone-of-in-ball3*:

```

  assumes S ⊆ sphere2
  shows cone-of S ⊆ ball3

```

proof

```

  fix y assume y ∈ cone-of S

```

```

  then obtain t x where y-eq: y = t *R x and t-pos: 0 < t and t-le: t ≤ 1
    and x-in: x ∈ S

```

```

    unfolding cone-of-def by auto

```

```

  from x-in assms have x-sphere: x ∈ sphere2 by auto

```

```

  hence x-norm: norm x = 1 by (simp add: sphere2-def)

```

```

  have norm y = norm (t *R x) by (simp add: y-eq)

```

```

  also have ... = |t| * norm x by simp

```

```

  also have ... = t using t-pos x-norm by simp

```

```

  also have ... ≤ 1 using t-le .

```

```

  finally have norm y ≤ 1 .

```

thus $y \in \text{ball3}$ by (simp add: ball3-def)
 qed

lemma *cone-of-in-ball3-minus-origin:*

assumes $S \subseteq \text{sphere2}$

shows $\text{cone-of } S \subseteq \text{ball3} - \{0\}$

proof

fix y assume $y\text{-in: } y \in \text{cone-of } S$

then obtain $t x$ where $y\text{-eq: } y = t *_{\mathbb{R}} x$ and $t\text{-pos: } 0 < t$ and $t\text{-le: } t \leq 1$

and $x\text{-in: } x \in S$

unfolding *cone-of-def* by *auto*

from *cone-of-in-ball3[OF assms]* $y\text{-in}$ have $y\text{-ball: } y \in \text{ball3}$

by *blast*

from $x\text{-in}$ *assms* have $x\text{-sphere: } x \in \text{sphere2}$

by *auto*

hence $x\text{-norm: } \text{norm } x = 1$

by (simp add: *sphere2-def*)

have $\text{norm } y = \text{norm } (t *_{\mathbb{R}} x)$

by (simp add: *y-eq*)

also have $\dots = t$

using $t\text{-pos}$ $x\text{-norm}$ by *simp*

finally have $\text{norm } y > 0$

using $t\text{-pos}$ by *simp*

hence $y \neq 0$

by *auto*

with $y\text{-ball}$ show $y \in \text{ball3} - \{0\}$

by *simp*

qed

lemma *ball3-minus-origin-eq-cone-sphere2:*

$\text{ball3} - \{0::\text{real}^3\} = \text{cone-of } \text{sphere2}$

proof

show $\text{ball3} - \{0::\text{real}^3\} \subseteq \text{cone-of } \text{sphere2}$

proof

fix y assume $y\text{-in: } y \in \text{ball3} - \{0::\text{real}^3\}$

hence $y\text{-ball: } y \in \text{ball3}$ and $y\text{-ne: } y \neq 0$

by *auto*

define t where $t = \text{norm } y$

define x where $x = (1 / t) *_{\mathbb{R}} y$

have $t\text{-pos: } 0 < t$

using $y\text{-ne}$ by (simp add: *t-def*)

have $t\text{-le: } t \leq 1$

using $y\text{-ball}$ by (simp add: *ball3-def t-def*)

have $x\text{-sphere: } x \in \text{sphere2}$

proof -

have $\text{norm } x = \text{norm } ((1 / t) *_{\mathbb{R}} y)$

by (simp add: *x-def*)

also have $\dots = 1$

using $t\text{-pos}$ by (simp add: *t-def*)

```

    finally show ?thesis
      by (simp add: sphere2-def)
  qed
  have y-eq:  $y = t *_R x$ 
    using t-pos by (simp add: x-def)
  show  $y \in \text{cone-of sphere2}$ 
    unfolding cone-of-def
    using y-eq t-pos t-le x-sphere by blast
  qed
  show  $\text{cone-of sphere2} \subseteq \text{ball3} - \{0::\text{real}^3\}$ 
    by (rule cone-of-in-ball3-minus-origin) simp
  qed

```

```

lemma cone-of-UN:
  cone-of ( $\bigcup i \in I. S i$ ) = ( $\bigcup i \in I. \text{cone-of } (S i)$ )
  unfolding cone-of-def by blast

```

```

lemma SO3-image-cone-of:

```

```

  assumes  $g \in \text{SO3}$ 
  shows  $g \text{ ' cone-of } S = \text{cone-of } (g \text{ ' } S)$ 

```

```

proof

```

```

  show  $g \text{ ' cone-of } S \subseteq \text{cone-of } (g \text{ ' } S)$ 

```

```

  proof

```

```

    fix y assume  $y \in g \text{ ' cone-of } S$ 

```

```

    then obtain x where y-eq:  $y = g x$  and x-in:  $x \in \text{cone-of } S$ 

```

```

    by blast

```

```

    then obtain t z where x-eq:  $x = t *_R z$  and t-pos:  $0 < t$  and t-le:  $t \leq 1$ 

```

```

    and z-in:  $z \in S$ 

```

```

    unfolding cone-of-def by blast

```

```

    have  $y = t *_R g z$ 

```

```

    using assms by (simp add: y-eq x-eq SO3-def rotation-def orthogonal-transformation-scaleR)

```

```

    thus  $y \in \text{cone-of } (g \text{ ' } S)$ 

```

```

    unfolding cone-of-def using t-pos t-le z-in by blast

```

```

  qed

```

```

  show  $\text{cone-of } (g \text{ ' } S) \subseteq g \text{ ' cone-of } S$ 

```

```

  proof

```

```

    fix y assume  $y \in \text{cone-of } (g \text{ ' } S)$ 

```

```

    then obtain t z where y-eq:  $y = t *_R z$  and t-pos:  $0 < t$  and t-le:  $t \leq 1$ 

```

```

    and z-in:  $z \in g \text{ ' } S$ 

```

```

    unfolding cone-of-def by blast

```

```

    then obtain x where x-in:  $x \in S$  and z-eq:  $z = g x$ 

```

```

    by blast

```

```

    have  $y = g (t *_R x)$ 

```

```

    using assms by (simp add: y-eq z-eq SO3-def rotation-def orthogonal-transformation-scaleR)

```

```

    moreover have  $t *_R x \in \text{cone-of } S$ 

```

```

    unfolding cone-of-def using t-pos t-le x-in by blast

```

```

    ultimately show  $y \in g \text{ ' cone-of } S$ 

```

```

    by blast

```

```

  qed

```

qed

lemma *cone-of-disjoint*:

assumes *S-sub*: $S \subseteq \text{sphere2}$

and *T-sub*: $T \subseteq \text{sphere2}$

and *disj*: $S \cap T = \{\}$

shows *cone-of* $S \cap \text{cone-of } T = \{\}$

proof

show *cone-of* $S \cap \text{cone-of } T \subseteq \{\}$

proof

fix *y*

assume *y*: $y \in \text{cone-of } S \cap \text{cone-of } T$

then obtain *t x* **where** *y-tx*: $y = t *_R x$ **and** *t-pos*: $0 < t$

and *xS*: $x \in S$

unfolding *cone-of-def* **by** *blast*

from *y* **obtain** *s z* **where** *y-sz*: $y = s *_R z$ **and** *s-pos*: $0 < s$

and *zT*: $z \in T$

unfolding *cone-of-def* **by** *blast*

have *x-norm*: $\text{norm } x = 1$

using *S-sub sphere2-def xS* **by** *auto*

have *z-norm*: $\text{norm } z = 1$

using *T-sub sphere2-def zT* **by** *auto*

have *t-eq*: $t = s$

proof –

have *t = norm (t *_R x)*

using *t-pos x-norm* **by** *simp*

also have $\dots = \text{norm } (s *_R z)$

using *y-tx y-sz* **by** *simp*

also have $\dots = s$

using *s-pos z-norm* **by** *simp*

finally show *?thesis* .

qed

have $x = z$

using *y-tx y-sz t-eq t-pos* **by** (*simp add: scaleR-cancel-left*)

hence $x \in S \cap T$

using *xS zT* **by** *simp*

with *disj* **show** $y \in \{\}$

by *simp*

qed

show $\{\} \subseteq \text{cone-of } S \cap \text{cone-of } T$

by *simp*

qed

lemma *cone-of-pairwise-disjoint*:

assumes *disj*: *pairwise-disjoint* *P*

and *sub*: $\forall i < \text{length } P. P ! i \subseteq \text{sphere2}$

shows *pairwise-disjoint* (*map cone-of* *P*)

unfolding *pairwise-disjoint-def*

proof (*intro allI impI*)

fix $i\ j$
assume $i: i < \text{length} (\text{map cone-of } P)$ **and** $j: j < \text{length} (\text{map cone-of } P)$ **and**
 $ij: i \neq j$
have $P ! i \cap P ! j = \{\}$
using $\text{disj } i\ j\ ij$ **by** $(\text{simp add: pairwise-disjoint-def})$
moreover have $P ! i \subseteq \text{sphere2 } P ! j \subseteq \text{sphere2}$
using $\text{sub } i\ j$ **by** auto
ultimately show $\text{map cone-of } P ! i \cap \text{map cone-of } P ! j = \{\}$
using $i\ j$ **by** $(\text{simp add: cone-of-disjoint})$
qed

lemma $SO3\text{-cone-images-pairwise-disjoint}$:

assumes $\text{len: length } P = \text{length } g$
and $g\text{-}SO3: \text{set } g \subseteq SO3$
and $P\text{-sub: } \forall i < \text{length } P. P ! i \subseteq \text{sphere2}$
and $\text{disj: pairwise-disjoint } (\text{map2 } (\lambda h\ A. h \text{ ' } A) g\ P)$
shows $\text{pairwise-disjoint } (\text{map2 } (\lambda h\ A. h \text{ ' } A) g (\text{map cone-of } P))$
unfolding $\text{pairwise-disjoint-def}$
proof (intro allI impI)

fix $i\ j$
assume $i: i < \text{length} (\text{map2 } (\lambda h\ A. h \text{ ' } A) g (\text{map cone-of } P))$
and $j: j < \text{length} (\text{map2 } (\lambda h\ A. h \text{ ' } A) g (\text{map cone-of } P))$
and $ij: i \neq j$
have $iP: i < \text{length } P$ **and** $jP: j < \text{length } P$
using $i\ j\ \text{len}$ **by** auto
have $gi: g ! i \in SO3$ **and** $gj: g ! j \in SO3$

proof –
have $i < \text{length } g$
using $iP\ \text{len}$ **by** simp
hence $g ! i \in \text{set } g$
by (rule nth-mem)
thus $g ! i \in SO3$
using $g\text{-}SO3$ **by** blast

next
have $j < \text{length } g$
using $jP\ \text{len}$ **by** simp
hence $g ! j \in \text{set } g$
by (rule nth-mem)
thus $g ! j \in SO3$
using $g\text{-}SO3$ **by** blast

qed
have $\text{img-disj: } (g ! i \text{ ' } (P ! i)) \cap (g ! j \text{ ' } (P ! j)) = \{\}$
using $\text{disj len } iP\ jP\ ij$ **by** $(\text{simp add: pairwise-disjoint-def})$
have $\text{img-i-sub: } g ! i \text{ ' } (P ! i) \subseteq \text{sphere2}$
using $P\text{-sub } iP\ \text{rotation-preserves-sphere2}[OF\ gi]$ **by** auto
have $\text{img-j-sub: } g ! j \text{ ' } (P ! j) \subseteq \text{sphere2}$
using $P\text{-sub } jP\ \text{rotation-preserves-sphere2}[OF\ gj]$ **by** auto
have $\text{cone-disj: cone-of } (g ! i \text{ ' } (P ! i)) \cap \text{cone-of } (g ! j \text{ ' } (P ! j)) = \{\}$
by $(\text{rule cone-of-disjoint}[OF\ \text{img-i-sub } \text{img-j-sub } \text{img-disj}])$

have $\text{map2 } (\lambda h A. h \text{ ' } A) g (\text{map cone-of } P) ! i =$
 $\text{cone-of } (g ! i \text{ ' } (P ! i))$
using $\text{len } iP \text{ SO3-image-cone-of}[OF gi, of P ! i]$ **by** *simp*
moreover have $\text{map2 } (\lambda h A. h \text{ ' } A) g (\text{map cone-of } P) ! j =$
 $\text{cone-of } (g ! j \text{ ' } (P ! j))$
using $\text{len } jP \text{ SO3-image-cone-of}[OF gj, of P ! j]$ **by** *simp*
ultimately show $\text{map2 } (\lambda h A. h \text{ ' } A) g (\text{map cone-of } P) ! i \cap$
 $\text{map2 } (\lambda h A. h \text{ ' } A) g (\text{map cone-of } P) ! j = \{\}$
using *cone-disj* **by** *simp*
qed

lemma *zero-in-ball3* [*simp*]: $(0::\text{real}^3) \in \text{ball3}$
by (*simp add: ball3-def*)

lemma *indexed-image-union-append-singleton*:
fixes $P :: 'a \text{ set list}$ **and** $G :: ('a \Rightarrow 'b) \text{ list}$ **and** $S :: 'a \text{ set}$
and $g :: 'a \Rightarrow 'b$
assumes $\text{length } P = \text{length } G$
shows $(\bigcup_{i < \text{length } (P @ [S])}. (G @ [g]) ! i \text{ ' } ((P @ [S]) ! i)) =$
 $(\bigcup_{i < \text{length } P}. G ! i \text{ ' } (P ! i)) \cup g \text{ ' } S$
proof –
have $(\bigcup_{i < \text{length } (P @ [S])}. (G @ [g]) ! i \text{ ' } ((P @ [S]) ! i)) =$
 $(\bigcup_{i < \text{Suc } (\text{length } P)}. (G @ [g]) ! i \text{ ' } ((P @ [S]) ! i))$
by *simp*
also have $\dots =$
 $(\bigcup_{i < \text{length } P}. (G @ [g]) ! i \text{ ' } ((P @ [S]) ! i)) \cup$
 $(G @ [g]) ! \text{length } P \text{ ' } ((P @ [S]) ! \text{length } P)$
by (*simp add: lessThan-Suc Un-commute*)
also have $\dots = (\bigcup_{i < \text{length } P}. G ! i \text{ ' } (P ! i)) \cup g \text{ ' } S$
using *assms* **by** (*simp add: nth-append*)
finally show *?thesis* .
qed

lemma *indexed-union-append-singleton*:
fixes $P :: 'a \text{ set list}$ **and** $S :: 'a \text{ set}$
shows $(\bigcup_{i < \text{length } (P @ [S])}. (P @ [S]) ! i) =$
 $(\bigcup_{i < \text{length } P}. P ! i) \cup S$
proof –
have $(\bigcup_{i < \text{length } (P @ [S])}. (P @ [S]) ! i) =$
 $(\bigcup_{i < \text{Suc } (\text{length } P)}. (P @ [S]) ! i)$
by *simp*
also have $\dots =$
 $(\bigcup_{i < \text{length } P}. (P @ [S]) ! i) \cup (P @ [S]) ! \text{length } P$
by (*simp add: lessThan-Suc Un-commute*)
finally show *?thesis*
by (*metis sphere-indexed-union-append-singleton*)
qed

The punctured ball is paradoxical via radial cones over the sphere pieces.

theorem *ball3-minus-origin-paradoxical*:

$\exists P Q :: (\text{real}^3) \text{ set list. } \exists gP gQ :: ((\text{real}^3) \Rightarrow (\text{real}^3)) \text{ list.}$
 $\text{length } P = \text{length } gP \wedge \text{length } Q = \text{length } gQ \wedge$
 $\text{set } gP \subseteq SO3 \wedge \text{set } gQ \subseteq SO3 \wedge$
 $(\forall i < \text{length } P. P ! i \subseteq \text{ball3} - \{0\}) \wedge$
 $(\forall i < \text{length } Q. Q ! i \subseteq \text{ball3} - \{0\}) \wedge$
 $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } P} P ! i) \wedge$
 $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } Q} Q ! i) \wedge$
 $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } P} (gP ! i) \text{ ‘ } (P ! i)) \wedge$
 $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } Q} (gQ ! i) \text{ ‘ } (Q ! i))$

proof –

from *sphere2-paradoxical* **obtain** $P Q :: (\text{real}^3) \text{ set list}$

and $gP gQ :: ((\text{real}^3 \Rightarrow \text{real}^3)) \text{ list}$

where *decomp*:

$\text{length } P = \text{length } gP \wedge \text{length } Q = \text{length } gQ \wedge$
 $\text{set } gP \subseteq SO3 \wedge \text{set } gQ \subseteq SO3 \wedge$
 $(\forall i < \text{length } P. P ! i \subseteq \text{sphere2}) \wedge$
 $(\forall i < \text{length } Q. Q ! i \subseteq \text{sphere2}) \wedge$
 $\text{sphere2} = (\bigcup_{i < \text{length } P} P ! i) \wedge$
 $\text{sphere2} = (\bigcup_{i < \text{length } Q} Q ! i) \wedge$
 $\text{sphere2} = (\bigcup_{i < \text{length } P} (gP ! i) \text{ ‘ } (P ! i)) \wedge$
 $\text{sphere2} = (\bigcup_{i < \text{length } Q} (gQ ! i) \text{ ‘ } (Q ! i))$

by (*elim exE*)

from *decomp* **have** *lenP*: $\text{length } P = \text{length } gP$

by (*elim conjE*)

from *decomp* **have** *lenQ*: $\text{length } Q = \text{length } gQ$

by (*elim conjE*)

from *decomp* **have** *gP-SO3*: $\text{set } gP \subseteq SO3$

by (*elim conjE*)

from *decomp* **have** *gQ-SO3*: $\text{set } gQ \subseteq SO3$

by (*elim conjE*)

from *decomp* **have** *P-sub-sphere*: $\forall i < \text{length } P. P ! i \subseteq \text{sphere2}$

by (*elim conjE*)

from *decomp* **have** *Q-sub-sphere*: $\forall i < \text{length } Q. Q ! i \subseteq \text{sphere2}$

by (*elim conjE*)

from *decomp* **have** *P-src-sphere*: $\text{sphere2} = (\bigcup_{i < \text{length } P} P ! i)$

by (*elim conjE*)

from *decomp* **have** *Q-src-sphere*: $\text{sphere2} = (\bigcup_{i < \text{length } Q} Q ! i)$

by (*elim conjE*)

from *decomp* **have** *P-cov-sphere*: $\text{sphere2} = (\bigcup_{i < \text{length } P} (gP ! i) \text{ ‘ } (P ! i))$

by (*elim conjE*)

from *decomp* **have** *Q-cov-sphere*: $\text{sphere2} = (\bigcup_{i < \text{length } Q} (gQ ! i) \text{ ‘ } (Q ! i))$

by (*elim conjE*)

define *P'* **where** $P' = \text{map cone-of } P$

define *Q'* **where** $Q' = \text{map cone-of } Q$

have *lenP'*: $\text{length } P' = \text{length } gP$

by (*simp add: P'-def lenP*)

have *lenQ'*: $\text{length } Q' = \text{length } gQ$

by (simp add: Q'-def lenQ)
 have P'-sub: $\forall i < \text{length } P'. P' ! i \subseteq \text{ball3} - \{0\}$
 using P-sub-sphere by (simp add: P'-def cone-of-in-ball3-minus-origin)
 have Q'-sub: $\forall i < \text{length } Q'. Q' ! i \subseteq \text{ball3} - \{0\}$
 using Q-sub-sphere by (simp add: Q'-def cone-of-in-ball3-minus-origin)
 have P'-src: $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } P'. P' ! i}$
 proof -
 have $(\bigcup_{i < \text{length } P'. P' ! i) = (\bigcup_{i < \text{length } P. \text{cone-of } (P ! i))$
 by (simp add: P'-def)
 also have ... = cone-of $(\bigcup_{i < \text{length } P. P ! i)$
 by (simp add: cone-of-UN)
 also have ... = cone-of sphere2
 using P-src-sphere by simp
 also have ... = ball3 - {0}
 using ball3-minus-origin-eq-cone-sphere2 by simp
 finally show ?thesis by simp
 qed
 have Q'-src: $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } Q'. Q' ! i}$
 proof -
 have $(\bigcup_{i < \text{length } Q'. Q' ! i) = (\bigcup_{i < \text{length } Q. \text{cone-of } (Q ! i))$
 by (simp add: Q'-def)
 also have ... = cone-of $(\bigcup_{i < \text{length } Q. Q ! i)$
 by (simp add: cone-of-UN)
 also have ... = cone-of sphere2
 using Q-src-sphere by simp
 also have ... = ball3 - {0}
 using ball3-minus-origin-eq-cone-sphere2 by simp
 finally show ?thesis by simp
 qed
 have P'-cov: $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } P'. (gP ! i) \text{ ' } (P' ! i))$
 proof -
 have $(\bigcup_{i < \text{length } P'. (gP ! i) \text{ ' } (P' ! i)) =$
 $(\bigcup_{i < \text{length } P. \text{cone-of } ((gP ! i) \text{ ' } (P ! i)))$
 proof -
 have $\bigwedge i. i < \text{length } P \implies (gP ! i) \text{ ' } \text{cone-of } (P ! i) =$
 $\text{cone-of } ((gP ! i) \text{ ' } (P ! i))$
 by (metis SO3-image-cone-of gP-SO3 lenP nth-mem subset-eq)
 thus ?thesis
 by (simp add: P'-def)
 qed
 also have ... = cone-of $(\bigcup_{i < \text{length } P. (gP ! i) \text{ ' } (P ! i))$
 by (simp add: cone-of-UN)
 also have ... = cone-of sphere2
 using P-cov-sphere by simp
 also have ... = ball3 - {0}
 using ball3-minus-origin-eq-cone-sphere2 by simp
 finally show ?thesis by simp
 qed
 have Q'-cov: $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } Q'. (gQ ! i) \text{ ' } (Q' ! i))$

proof –
have $(\bigcup_{i < \text{length } Q'} (gQ ! i) \text{ ' } (Q' ! i)) =$
 $(\bigcup_{i < \text{length } Q} \text{cone-of } ((gQ ! i) \text{ ' } (Q ! i)))$
proof –
have $\bigwedge i. i < \text{length } Q \implies (gQ ! i) \text{ ' cone-of } (Q ! i) =$
 $\text{cone-of } ((gQ ! i) \text{ ' } (Q ! i))$
proof –
fix i **assume** $i\text{-lt}: i < \text{length } Q$
have $gQ ! i \in \text{set } gQ$
using $i\text{-lt len}Q$ **by** $(\text{simp add: nth-mem})$
hence $gQ ! i \in SO3$
using $gQ\text{-}SO3$ **by** blast
thus $(gQ ! i) \text{ ' cone-of } (Q ! i) = \text{cone-of } ((gQ ! i) \text{ ' } (Q ! i))$
by $(\text{rule } SO3\text{-image-cone-of})$
qed
thus $?thesis$
by $(\text{simp add: } Q'\text{-def})$
qed
also have $\dots = \text{cone-of } (\bigcup_{i < \text{length } Q} (gQ ! i) \text{ ' } (Q ! i))$
by $(\text{simp add: cone-of-UN})$
also have $\dots = \text{cone-of sphere2}$
using $Q\text{-cov-sphere}$ **by** simp
also have $\dots = \text{ball3} - \{0\}$
using $\text{ball3-minus-origin-eq-cone-sphere2}$ **by** simp
finally show $?thesis$ **by** simp
qed
show $?thesis$
apply $(\text{rule exI[of - } P'])$
apply $(\text{rule exI[of - } Q'])$
apply $(\text{rule exI[of - } gP])$
apply $(\text{rule exI[of - } gQ])$
using $\text{len}P' \text{ len}Q' gP\text{-}SO3 gQ\text{-}SO3 P'\text{-sub } Q'\text{-sub } P'\text{-src } Q'\text{-src } P'\text{-cov } Q'\text{-cov}$
by simp
qed

theorem $\text{ball3-minus-origin-paradoxical-strong}$:
 $\exists P Q :: (\text{real}^3) \text{ set list. } \exists gP gQ :: ((\text{real}^3) \Rightarrow (\text{real}^3)) \text{ list.}$
 $\text{length } P = \text{length } gP \wedge \text{length } Q = \text{length } gQ \wedge$
 $\text{set } gP \subseteq SO3 \wedge \text{set } gQ \subseteq SO3 \wedge$
 $\text{pairwise-disjoint } (P @ Q) \wedge$
 $\text{pairwise-disjoint } (\text{map2 } (\lambda g A. g \text{ ' } A) gP P) \wedge$
 $\text{pairwise-disjoint } (\text{map2 } (\lambda g A. g \text{ ' } A) gQ Q) \wedge$
 $(\forall i < \text{length } P. P ! i \subseteq \text{ball3} - \{0\}) \wedge$
 $(\forall i < \text{length } Q. Q ! i \subseteq \text{ball3} - \{0\}) \wedge$
 $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } P} P ! i) \cup (\bigcup_{i < \text{length } Q} Q ! i) \wedge$
 $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } P} (gP ! i) \text{ ' } (P ! i)) \wedge$
 $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } Q} (gQ ! i) \text{ ' } (Q ! i))$

proof –
from $\text{sphere2-paradoxical-strong}$ **obtain** $P Q :: (\text{real}^3) \text{ set list}$

```

and  $gP\ gQ :: ((\mathbb{R}^3 \Rightarrow \mathbb{R}^3))\ list$ 
where  $lenP: length\ P = length\ gP$ 
and  $lenQ: length\ Q = length\ gQ$ 
and  $gP\text{-}SO3: set\ gP \subseteq SO3$ 
and  $gQ\text{-}SO3: set\ gQ \subseteq SO3$ 
and  $source\text{-}disj: pairwise\text{-}disjoint\ (P\ @\ Q)$ 
and  $imageP\text{-}disj: pairwise\text{-}disjoint\ (map2\ (\lambda g\ A.\ g\ 'A)\ gP\ P)$ 
and  $imageQ\text{-}disj: pairwise\text{-}disjoint\ (map2\ (\lambda g\ A.\ g\ 'A)\ gQ\ Q)$ 
and  $P\text{-}sub\text{-}sphere: \forall i < length\ P.\ P\ !\ i \subseteq sphere2$ 
and  $Q\text{-}sub\text{-}sphere: \forall i < length\ Q.\ Q\ !\ i \subseteq sphere2$ 
and  $source\text{-}cover: sphere2 = (\bigcup_{i < length\ P} P\ !\ i) \cup (\bigcup_{i < length\ Q} Q\ !\ i)$ 
and  $P\text{-}cov\text{-}sphere: sphere2 = (\bigcup_{i < length\ P} (gP\ !\ i)\ ' (P\ !\ i))$ 
and  $Q\text{-}cov\text{-}sphere: sphere2 = (\bigcup_{i < length\ Q} (gQ\ !\ i)\ '(Q\ !\ i))$ 
by auto

define  $P'$  where  $P' = map\ cone\text{-}of\ P$ 
define  $Q'$  where  $Q' = map\ cone\text{-}of\ Q$ 
have  $lenP': length\ P' = length\ gP$ 
by (simp add: P'-def lenP)
have  $lenQ': length\ Q' = length\ gQ$ 
by (simp add: Q'-def lenQ)
have  $P'\text{-}sub: \forall i < length\ P'. P'\ !\ i \subseteq ball3 - \{0\}$ 
using  $P\text{-}sub\text{-}sphere$  by (simp add: P'-def cone-of-in-ball3-minus-origin)
have  $Q'\text{-}sub: \forall i < length\ Q'. Q'\ !\ i \subseteq ball3 - \{0\}$ 
using  $Q\text{-}sub\text{-}sphere$  by (simp add: Q'-def cone-of-in-ball3-minus-origin)
have  $all\text{-}sub: \forall i < length\ (P\ @\ Q). (P\ @\ Q)\ !\ i \subseteq sphere2$ 
using  $P\text{-}sub\text{-}sphere\ Q\text{-}sub\text{-}sphere$  by (auto simp: nth-append)
have  $source\text{-}disj': pairwise\text{-}disjoint\ (P'\ @\ Q')$ 
unfolding  $P'\text{-}def\ Q'\text{-}def$ 
using  $cone\text{-}of\text{-}pairwise\text{-}disjoint[OF\ source\text{-}disj\ all\text{-}sub]$  by simp
have  $imageP\text{-}disj': pairwise\text{-}disjoint\ (map2\ (\lambda g\ A.\ g\ 'A)\ gP\ P')$ 
unfolding  $P'\text{-}def$ 
by (rule  $SO3\text{-}cone\text{-}images\text{-}pairwise\text{-}disjoint[OF\ lenP\ gP\text{-}SO3\ P\text{-}sub\text{-}sphere\ imageP\text{-}disj]$ )
have  $imageQ\text{-}disj': pairwise\text{-}disjoint\ (map2\ (\lambda g\ A.\ g\ 'A)\ gQ\ Q')$ 
unfolding  $Q'\text{-}def$ 
by (rule  $SO3\text{-}cone\text{-}images\text{-}pairwise\text{-}disjoint[OF\ lenQ\ gQ\text{-}SO3\ Q\text{-}sub\text{-}sphere\ imageQ\text{-}disj]$ )

have  $P'\text{-}src: (\bigcup_{i < length\ P'} P'\ !\ i) = cone\text{-}of\ (\bigcup_{i < length\ P} P\ !\ i)$ 
proof –
have  $(\bigcup_{i < length\ P'} P'\ !\ i) = (\bigcup_{i < length\ P} cone\text{-}of\ (P\ !\ i))$ 
by (simp add: P'-def)
also have  $\dots = cone\text{-}of\ (\bigcup_{i < length\ P} P\ !\ i)$ 
by (simp add: cone-of-UN)
finally show ?thesis .
qed
have  $Q'\text{-}src: (\bigcup_{i < length\ Q'} Q'\ !\ i) = cone\text{-}of\ (\bigcup_{i < length\ Q} Q\ !\ i)$ 
using  $Q'\text{-}def\ cone\text{-}of\text{-}UN$  by fastforce

```

have $\text{source-cover}' : \text{ball3} - \{0\} = (\bigcup_{i < \text{length } P'} P' ! i) \cup (\bigcup_{i < \text{length } Q'} Q' ! i)$
proof –
have $(\bigcup_{i < \text{length } P'} P' ! i) \cup (\bigcup_{i < \text{length } Q'} Q' ! i) =$
 $\text{cone-of } ((\bigcup_{i < \text{length } P} P ! i) \cup (\bigcup_{i < \text{length } Q} Q ! i))$
using $P'\text{-src } Q'\text{-src}$ **by** $(\text{auto simp: cone-of-def})$
also have $\dots = \text{cone-of sphere2}$
using source-cover **by** simp
also have $\dots = \text{ball3} - \{0\}$
using $\text{ball3-minus-origin-eq-cone-sphere2}$ **by** simp
finally show $?thesis$ **by** simp
qed

have $P'\text{-cov} : \text{ball3} - \{0\} = (\bigcup_{i < \text{length } P'} (gP ! i) \text{ ' } (P' ! i))$
proof –
have $(\bigcup_{i < \text{length } P'} (gP ! i) \text{ ' } (P' ! i)) =$
 $(\bigcup_{i < \text{length } P} \text{cone-of } ((gP ! i) \text{ ' } (P ! i)))$
proof –
have $\bigwedge i. i < \text{length } P \implies (gP ! i) \text{ ' } \text{cone-of } (P ! i) =$
 $\text{cone-of } ((gP ! i) \text{ ' } (P ! i))$
proof –
fix i **assume** $i\text{-lt} : i < \text{length } P$
have $gP ! i \in \text{SO3}$
using $gP\text{-SO3 } i\text{-lt } \text{len}P$ **by** auto
thus $(gP ! i) \text{ ' } \text{cone-of } (P ! i) = \text{cone-of } ((gP ! i) \text{ ' } (P ! i))$
by $(\text{rule } \text{SO3-image-cone-of})$
qed
thus $?thesis$
by $(\text{simp add: } P'\text{-def})$
qed

also have $\dots = \text{cone-of } (\bigcup_{i < \text{length } P} (gP ! i) \text{ ' } (P ! i))$
by $(\text{simp add: cone-of-UN})$
also have $\dots = \text{cone-of sphere2}$
using $P\text{-cov-sphere}$ **by** simp
also have $\dots = \text{ball3} - \{0\}$
using $\text{ball3-minus-origin-eq-cone-sphere2}$ **by** simp
finally show $?thesis$ **by** simp
qed

have $Q'\text{-cov} : \text{ball3} - \{0\} = (\bigcup_{i < \text{length } Q'} (gQ ! i) \text{ ' } (Q' ! i))$
proof –
have $(\bigcup_{i < \text{length } Q'} (gQ ! i) \text{ ' } (Q' ! i)) =$
 $(\bigcup_{i < \text{length } Q} \text{cone-of } ((gQ ! i) \text{ ' } (Q ! i)))$
proof –
have $\bigwedge i. i < \text{length } Q \implies (gQ ! i) \text{ ' } \text{cone-of } (Q ! i) =$
 $\text{cone-of } ((gQ ! i) \text{ ' } (Q ! i))$
proof –
fix i **assume** $i\text{-lt} : i < \text{length } Q$
have $gQ ! i \in \text{SO3}$
using $gQ\text{-SO3 } i\text{-lt } \text{len}Q$ **by** auto

```

    thus (gQ ! i) ‘ cone-of (Q ! i) = cone-of ((gQ ! i) ‘ (Q ! i))
      by (rule SO3-image-cone-of)
  qed
  thus ?thesis
    by (simp add: Q'-def)
  qed
  also have ... = cone-of (⋃ i<length Q. (gQ ! i) ‘ (Q ! i))
    by (simp add: cone-of-UN)
  also have ... = cone-of sphere2
    using Q-cov-sphere by simp
  also have ... = ball3 - {0}
    using ball3-minus-origin-eq-cone-sphere2 by simp
  finally show ?thesis by simp
  qed
  show ?thesis
  proof (intro exI conjI)
    show length P' = length gP
      by (rule lenP')
    show length Q' = length gQ
      by (rule lenQ')
    show set gP ⊆ SO3
      by (rule gP-SO3)
    show set gQ ⊆ SO3
      by (rule gQ-SO3)
    show pairwise-disjoint (P' @ Q')
      by (rule source-disj')
    show pairwise-disjoint (map2 (λg A. g ‘ A) gP P')
      by (rule imageP-disj')
    show pairwise-disjoint (map2 (λg A. g ‘ A) gQ Q')
      by (rule imageQ-disj')
    show ∀ i<length P'. P' ! i ⊆ ball3 - {0}
      by (rule P'-sub)
    show ∀ i<length Q'. Q' ! i ⊆ ball3 - {0}
      by (rule Q'-sub)
    show ball3 - {0} = (⋃ i<length P'. P' ! i) ∪ (⋃ i<length Q'. Q' ! i)
      by (rule source-cover')
    show ball3 - {0} = (⋃ i<length P'. (gP ! i) ‘ (P' ! i))
      by (rule P'-cov)
    show ball3 - {0} = (⋃ i<length Q'. (gQ ! i) ‘ (Q' ! i))
      by (rule Q'-cov)
  qed
  qed

```

Adjoining the origin gives a useful SO(3)-only finite-cover decomposition of the full ball. The partition-level absorption is completed in the final theory, where off-centre isometries are available.

theorem *ball3-paradoxical*:

$$\exists P Q :: (\text{real}^3) \text{ set list. } \exists gP gQ :: ((\text{real}^3) \Rightarrow (\text{real}^3)) \text{ list.}$$

$$\text{length } P = \text{length } gP \wedge \text{length } Q = \text{length } gQ \wedge$$

$$\begin{aligned}
& \text{set } gP \subseteq SO3 \wedge \text{set } gQ \subseteq SO3 \wedge \\
& (\forall i < \text{length } P. P ! i \subseteq \text{ball3}) \wedge \\
& (\forall i < \text{length } Q. Q ! i \subseteq \text{ball3}) \wedge \\
& \text{ball3} = (\bigcup_{i < \text{length } P} P ! i) \wedge \\
& \text{ball3} = (\bigcup_{i < \text{length } Q} Q ! i) \wedge \\
& \text{ball3} = (\bigcup_{i < \text{length } P} (gP ! i) \text{ ' } (P ! i)) \wedge \\
& \text{ball3} = (\bigcup_{i < \text{length } Q} (gQ ! i) \text{ ' } (Q ! i))
\end{aligned}$$

proof –

from *ball3-minus-origin-paradoxical* **obtain** $P \ Q :: (\text{real}^3) \text{ set list}$

and $gP \ gQ :: ((\text{real}^3 \Rightarrow \text{real}^3)) \text{ list}$

where *decomp*:

$\text{length } P = \text{length } gP \wedge \text{length } Q = \text{length } gQ \wedge$

$\text{set } gP \subseteq SO3 \wedge \text{set } gQ \subseteq SO3 \wedge$

$(\forall i < \text{length } P. P ! i \subseteq \text{ball3} - \{0\}) \wedge$

$(\forall i < \text{length } Q. Q ! i \subseteq \text{ball3} - \{0\}) \wedge$

$\text{ball3} - \{0\} = (\bigcup_{i < \text{length } P} P ! i) \wedge$

$\text{ball3} - \{0\} = (\bigcup_{i < \text{length } Q} Q ! i) \wedge$

$\text{ball3} - \{0\} = (\bigcup_{i < \text{length } P} (gP ! i) \text{ ' } (P ! i)) \wedge$

$\text{ball3} - \{0\} = (\bigcup_{i < \text{length } Q} (gQ ! i) \text{ ' } (Q ! i))$

by (*elim exE*)

from *decomp* **have** *lenP*: $\text{length } P = \text{length } gP$

by (*elim conjE*)

from *decomp* **have** *lenQ*: $\text{length } Q = \text{length } gQ$

by (*elim conjE*)

from *decomp* **have** *gP-SO3*: $\text{set } gP \subseteq SO3$

by (*elim conjE*)

from *decomp* **have** *gQ-SO3*: $\text{set } gQ \subseteq SO3$

by (*elim conjE*)

from *decomp* **have** *P-sub*: $\forall i < \text{length } P. P ! i \subseteq \text{ball3} - \{0\}$

by (*elim conjE*)

from *decomp* **have** *Q-sub*: $\forall i < \text{length } Q. Q ! i \subseteq \text{ball3} - \{0\}$

by (*elim conjE*)

from *decomp* **have** *P-src*: $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } P} P ! i)$

by (*elim conjE*)

from *decomp* **have** *Q-src*: $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } Q} Q ! i)$

by (*elim conjE*)

from *decomp* **have** *P-cov*: $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } P} (gP ! i) \text{ ' } (P ! i))$

by (*elim conjE*)

from *decomp* **have** *Q-cov*: $\text{ball3} - \{0\} = (\bigcup_{i < \text{length } Q} (gQ ! i) \text{ ' } (Q ! i))$

by (*elim conjE*)

define *P'* **where** $P' = P @ [\{0::\text{real}^3\}]$

define *Q'* **where** $Q' = Q @ [\{0::\text{real}^3\}]$

define *gP'* **where** $gP' = gP @ [id]$

define *gQ'* **where** $gQ' = gQ @ [id]$

have *lenP'*: $\text{length } P' = \text{length } gP'$

by (*simp add: P'-def gP'-def lenP*)

have *lenQ'*: $\text{length } Q' = \text{length } gQ'$

by (*simp add: Q'-def gQ'-def lenQ*)

have *P'-sub*: $\forall i < \text{length } P'. P' ! i \subseteq \text{ball3}$

```

    using P-sub by (auto simp add: P'-def nth-append)
  have Q'-sub:  $\forall i < \text{length } Q'. Q' ! i \subseteq \text{ball3}$ 
    using Q-sub by (auto simp add: Q'-def nth-append)
  have gP'-SO3:  $\text{set } gP' \subseteq \text{SO3}$ 
    using gP-SO3 by (simp add: gP'-def id-in-SO3)
  have gQ'-SO3:  $\text{set } gQ' \subseteq \text{SO3}$ 
    using gQ-SO3 by (simp add: gQ'-def id-in-SO3)
  have P'-src:  $\text{ball3} = (\bigcup i < \text{length } P'. P' ! i)$ 
proof -
  have append-union:
     $(\bigcup i < \text{length } (P @ \{0::\text{real}^3\})). (P @ \{0\}) ! i =$ 
     $(\bigcup i < \text{length } P. P' ! i) \cup \{0\}$ 
    using indexed-union-append-singleton[where P = P and S = {0::real^3}] .
  have  $(\bigcup i < \text{length } P'. P' ! i) = (\text{ball3} - \{0\}) \cup \{0\}$ 
    unfolding P'-def
    using append-union P-src by simp
  also have ... = ball3
    by auto
  finally show ?thesis by simp
qed
have Q'-src:  $\text{ball3} = (\bigcup i < \text{length } Q'. Q' ! i)$ 
proof -
  have append-union:
     $(\bigcup i < \text{length } (Q @ \{0::\text{real}^3\})). (Q @ \{0\}) ! i =$ 
     $(\bigcup i < \text{length } Q. Q' ! i) \cup \{0\}$ 
    using indexed-union-append-singleton[where P = Q and S = {0::real^3}] .
  have  $(\bigcup i < \text{length } Q'. Q' ! i) = (\text{ball3} - \{0\}) \cup \{0\}$ 
    unfolding Q'-def
    using append-union Q-src by simp
  also have ... = ball3
    by auto
  finally show ?thesis by simp
qed
have P'-cov:  $\text{ball3} = (\bigcup i < \text{length } P'. (gP' ! i) ' (P' ! i))$ 
proof -
  have append-union:
     $(\bigcup i < \text{length } (P @ \{0::\text{real}^3\})). (gP @ [id]) ! i ' ((P @ \{0\}) ! i) =$ 
     $(\bigcup i < \text{length } P. gP ! i ' (P ! i)) \cup \{0\}$ 
    using indexed-image-union-append-singleton[
      where P = P and G = gP and S = {0::real^3} and g = id, OF lenP]
    by simp
  have union-eq:  $(\bigcup i < \text{length } P'. (gP' ! i) ' (P' ! i)) = \text{ball3}$ 
proof -
  have  $(\bigcup i < \text{length } P'. (gP' ! i) ' (P' ! i)) =$ 
     $(\bigcup i < \text{length } P. gP ! i ' (P ! i)) \cup \{0\}$ 
    unfolding P'-def gP'-def
    using append-union .
  also have ... =  $(\text{ball3} - \{0\}) \cup \{0\}$ 
    using P-cov by simp

```

```

    also have ... = ball3
      by auto
    finally show ?thesis .
  qed
  thus ?thesis by simp
qed
have Q'-cov: ball3 = (⋃ i < length Q'. (gQ' ! i) ' (Q' ! i))
proof -
  have append-union:
    (⋃ i < length (Q @ [{0::real^3}]). (gQ @ [id]) ! i ' ((Q @ [{0}]) ! i)) =
      (⋃ i < length Q. gQ ! i ' (Q ! i)) ∪ {0}
    using indexed-image-union-append-singleton[
      where P = Q and G = gQ and S = {0::real^3} and g = id, OF lenQ]
    by simp
  have union-eq: (⋃ i < length Q'. (gQ' ! i) ' (Q' ! i)) = ball3
proof -
  have (⋃ i < length Q'. (gQ' ! i) ' (Q' ! i)) =
    (⋃ i < length Q. gQ ! i ' (Q ! i)) ∪ {0}
    unfolding Q'-def gQ'-def
    using append-union .
  also have ... = (ball3 - {0}) ∪ {0}
    using Q-cov by simp
  also have ... = ball3
    by auto
  finally show ?thesis .
qed
thus ?thesis by simp
qed
show ?thesis
  apply (rule exI[of - P'])
  apply (rule exI[of - Q'])
  apply (rule exI[of - gP'])
  apply (rule exI[of - gQ'])
  using lenP' lenQ' gP'-SO3 gQ'-SO3 P'-sub Q'-sub P'-src Q'-src P'-cov Q'-cov
  by simp
qed
end

```

```

theory Banach-Tarski-Theorem
  imports Ball-Decomposition
begin

```

Two copies of the ball, embedded in disjoint regions of \mathbb{R}^3 .

```

definition shift3 :: real^3 where
  shift3 = vector [3, 0, 0]

```

```

definition ball3-copy-left :: (real^3) set where

```

$ball3\text{-copy-left} = ball3$

definition $ball3\text{-copy-right} :: (real^3) \text{ set}$ **where**
 $ball3\text{-copy-right} = (+) \text{ shift3 } ' ball3$

lemma $norm\text{-shift3}$ [*simp*]: $norm \text{ shift3} = 3$
unfolding $shift3\text{-def}$
by (*simp add: norm-eq-sqrt-inner inner-vec-def sum-3*)

lemma $ball3\text{-copies-disjoint}$:

$ball3\text{-copy-left} \cap ball3\text{-copy-right} = \{\}$

proof

show $ball3\text{-copy-left} \cap ball3\text{-copy-right} \subseteq \{\}$

proof

fix z

assume $z \in ball3\text{-copy-left} \cap ball3\text{-copy-right}$

then obtain y **where** $z\text{-ball}: z \in ball3$ **and** $y\text{-ball}: y \in ball3$ **and** $z\text{-eq}: z =$
 $shift3 + y$

unfolding $ball3\text{-copy-left-def}$ $ball3\text{-copy-right-def}$ **by** *auto*

have $z\text{-norm}: norm\ z \leq 1$ **and** $y\text{-norm}: norm\ y \leq 1$

using $z\text{-ball}$ $y\text{-ball}$ **by** (*simp-all add: ball3-def*)

have $tri: norm\ (z - y) \leq norm\ z + norm\ y$

using $norm\text{-triangle-ineq4}$ **by** *blast*

have $3 = norm\ (z - y)$

using $z\text{-eq}$ **by** *simp*

also have $\dots \leq norm\ z + norm\ y$

by (*rule tri*)

also have $\dots \leq 2$

using $z\text{-norm}$ $y\text{-norm}$ **by** *simp*

finally show $z \in \{\}$

by *simp*

qed

qed *simp*

Isometries of \mathbb{R}^3 are the motions used in the final assembly. Rotations from $SO(3)$, translations, and their compositions are isometries.

definition $isometry :: (real^3 \Rightarrow real^3) \Rightarrow bool$ **where**
 $isometry\ f \longleftrightarrow (\forall x\ y. dist\ (f\ x)\ (f\ y) = dist\ x\ y)$

lemma $id\text{-isometry}$ [*simp*]: $isometry\ id$
by (*simp add: isometry-def*)

lemma $translation\text{-isometry}$ [*simp*]: $isometry\ ((+) a)$
by (*simp add: isometry-def dist-norm*)

lemma $SO3\text{-isometry}$:

assumes $g \in SO3$

shows $isometry\ g$

using *assms*

by (auto simp add: SO3-def rotation-def isometry-def orthogonal-transformation-isometry)

lemma *isometry-compose*:

assumes *isometry f isometry g*
shows *isometry (f o g)*
using *assms* by (simp add: *isometry-def*)

lemma *translation-compose-SO3-isometry* [simp]:

assumes $g \in SO3$
shows *isometry ((+) a o g)*
using *assms* by (intro *isometry-compose translation-isometry SO3-isometry*)

lemma *offcenter-rotation-isometry*:

assumes $R \in SO3$
shows *isometry ($\lambda x. c + R (x - c)$)*

proof –

have $(\lambda x. c + R (x - c)) = (+) c \circ R \circ (+) (- c)$
by (rule *ext*) simp
thus ?thesis
using *assms* by (simp add: *isometry-compose SO3-isometry*)

qed

lemma *offcenter-rotation-inverse-left*:

fixes $c :: real^3$
assumes *inv-left: $Rinv \circ R = id$*
defines $T \equiv \lambda x. c + R (x - c)$
and $U \equiv \lambda x. c + Rinv (x - c)$
shows $U (T x) = x$
using *fun-cong[OF inv-left, of $x - c$]*
by (simp add: *T-def U-def*)

lemma *offcenter-rotation-inverse-right*:

fixes $c :: real^3$
assumes *inv-right: $R \circ Rinv = id$*
defines $T \equiv \lambda x. c + R (x - c)$
and $U \equiv \lambda x. c + Rinv (x - c)$
shows $T (U x) = x$
using *fun-cong[OF inv-right, of $x - c$]*
by (simp add: *T-def U-def*)

lemma *offcenter-rotation-funpow-zero*:

fixes $c :: real^3$ and $R :: real^3 \Rightarrow real^3$
defines $T \equiv \lambda x. c + R (x - c)$
shows $(T \hat{\sim} n) 0 = c + (R \hat{\sim} n) (- c)$
proof (*induction n*)
case 0
show ?case
by simp
next

case (*Suc n*)
have $(T \overset{\sim}{\sim} \text{Suc } n) \ 0 = T ((T \overset{\sim}{\sim} n) \ 0)$
by *simp*
also have $\dots = T (c + (R \overset{\sim}{\sim} n) (-c))$
using *Suc.IH* **by** *simp*
also have $\dots = c + R ((R \overset{\sim}{\sim} n) (-c))$
by (*simp add: T-def*)
also have $\dots = c + (R \overset{\sim}{\sim} \text{Suc } n) (-c)$
by *simp*
finally show *?case* .
qed

lemma *sigma-replicate-b*:
 $\text{sigma } (\text{replicate } n \ (\text{False}, B)) = (Rz \overset{\sim}{\sim} n)$
by (*induction n*) (*simp-all add: letter-to-rot-def*)

lemma *canceled-replicate-b*: *canceled* (*replicate n (False, B)*)
unfolding *canceled-def*

proof
assume *Domainp cancels-to-1* (*replicate n (False, B)*)
then obtain *w* **where** *cancels-to-1* (*replicate n (False, B)*) *w*
by *auto*
then obtain *i* **where** $\text{Suc } i < \text{length } (\text{replicate } n \ (\text{False}, B))$
and *canceling* (*replicate n (False, B) ! i*) (*replicate n (False, B) ! Suc i*)
by (*auto simp: cancels-to-1-def cancels-to-1-at-def*)
hence *canceling* (*False, B*) (*False, B*)
by *simp*
thus *False*
by (*simp add: canceling-def*)
qed

lemma *replicate-b-in-F2*: *replicate n (False, B) ∈ carrier F2*
by (*intro canceled-in-F2 canceled-replicate-b*)

theorem *ball3-paradoxical-strong*:
 $\exists P \ Q :: (\text{real}^3) \ \text{set list. } \exists gP \ gQ :: ((\text{real}^3) \Rightarrow (\text{real}^3)) \ \text{list.}$
 $\text{length } P = \text{length } gP \wedge \text{length } Q = \text{length } gQ \wedge$
 $(\forall g \in \text{set } gP. \ \text{isometry } g) \wedge (\forall g \in \text{set } gQ. \ \text{isometry } g) \wedge$
 $\text{pairwise-disjoint } (P \ @ \ Q) \wedge$
 $\text{pairwise-disjoint } (\text{map2 } (\lambda g \ A. \ g \ ' \ A) \ gP \ P) \wedge$
 $\text{pairwise-disjoint } (\text{map2 } (\lambda g \ A. \ g \ ' \ A) \ gQ \ Q) \wedge$
 $(\forall i < \text{length } P. \ P \ ! \ i \subseteq \text{ball3}) \wedge$
 $(\forall i < \text{length } Q. \ Q \ ! \ i \subseteq \text{ball3}) \wedge$
 $\text{ball3} = (\bigcup i < \text{length } P. \ P \ ! \ i) \cup (\bigcup i < \text{length } Q. \ Q \ ! \ i) \wedge$
 $\text{ball3} = (\bigcup i < \text{length } P. \ (gP \ ! \ i) \ ' \ (P \ ! \ i)) \wedge$
 $\text{ball3} = (\bigcup i < \text{length } Q. \ (gQ \ ! \ i) \ ' \ (Q \ ! \ i))$

proof –
obtain *a* **where** *a-sphere*: $a \in \text{sphere2}$ **and** *a-not-D*: $a \notin \text{bad-set-D}$
using *exists-antipodal-axis-avoiding-countable[OF bad-set-D-countable]* **by** *blast*

```

let ?c = - (1 / 2) *R a
define T where T = (λx. ?c + Rz (x - ?c))
define U where U = (λx. ?c + Rz-inv (x - ?c))
have T-iso: isometry T
  unfolding T-def by (rule offcenter-rotation-isometry[OF Rz-in-SO3])
have U-iso: isometry U
  unfolding U-def by (rule offcenter-rotation-isometry[OF Rz-inv-in-SO3])
have U-T: ∧x. U (T x) = x
  unfolding T-def U-def
  by (rule offcenter-rotation-inverse-left[OF Rz-inv-left])
have T-U: ∧x. T (U x) = x
  unfolding T-def U-def
  by (rule offcenter-rotation-inverse-right[OF Rz-inv-right])

have no-Rz-power-fix: ∧n. n > 0 ⇒ (Rz ~ n) a ≠ a
proof
  fix n :: nat
  assume n-pos: n > 0 and fixed: (Rz ~ n) a = a
  let ?w = replicate n (False, B)
  have w-carrier: ?w ∈ carrier F2
    by (rule replicate-b-in-F2)
  have w-ne: ?w ≠ []
    using n-pos by auto
  have sigma ?w a = a
    using fixed by (simp add: sigma-replicate-b)
  hence a ∈ fixed-point-set (sigma ?w)
    using a-sphere by (simp add: fixed-point-set-def)
  hence a ∈ bad-set-D
    unfolding bad-set-D-def using w-carrier w-ne by blast
  with a-not-D show False
    by contradiction
qed

have Rz-power-a-inj: ∧n m. (Rz ~ n) a = (Rz ~ m) a ⇒ n = m
proof -
  fix n m :: nat
  assume eq: (Rz ~ n) a = (Rz ~ m) a
  show n = m
  proof (cases n m rule: linorder-cases)
  case less
    let ?d = m - n
    have d-pos: ?d > 0
      using less by simp
    have m-eq: m = n + ?d
      using less by simp
    have (Rz ~ n) a = (Rz ~ n) ((Rz ~ ?d) a)
      using eq m-eq by (metis comp-apply funpow-add)
    hence a = (Rz ~ ?d) a
      using SO3-inj[OF SO3-funpow[OF Rz-in-SO3, of n]] by (auto simp: inj-def)
  end
end

```

```

with no-Rz-power-fix[OF d-pos] show ?thesis
  by simp
next
case equal
then show ?thesis .
next
case greater
let ?d = n - m
have d-pos: ?d > 0
  using greater by simp
have n-eq: n = m + ?d
  using greater by simp
have (Rz  $\widehat{\sim}$  m) a = (Rz  $\widehat{\sim}$  m) ((Rz  $\widehat{\sim}$  ?d) a)
  using eq n-eq by (metis comp-apply funpow-add)
hence a = (Rz  $\widehat{\sim}$  ?d) a
  using SO3-inj[OF SO3-funpow[OF Rz-in-SO3, of m]] by (auto simp: inj-def)
with no-Rz-power-fix[OF d-pos] show ?thesis
  by simp
qed
qed

```

```

have T-pow-zero:  $\bigwedge n. (T \widehat{\sim} n) 0 = ?c + (Rz \widehat{\sim} n) (- ?c)$ 
  unfolding T-def by (rule offcenter-rotation-funpow-zero)
have T-pow-zero-diff:  $\bigwedge n. (T \widehat{\sim} n) 0 = (1 / 2) *_R ((Rz \widehat{\sim} n) a - a)$ 
proof -
  fix n
  have lin: linear (Rz  $\widehat{\sim}$  n)
    by (rule SO3-linear[OF SO3-funpow[OF Rz-in-SO3]])
  have (T  $\widehat{\sim}$  n) 0 = ?c + (Rz  $\widehat{\sim}$  n) (- ?c)
    by (rule T-pow-zero)
  also have ... = (- (1 / 2) *R a) + (1 / 2) *R ((Rz  $\widehat{\sim}$  n) a)
    using linear-scale[OF lin, of 1 / 2 a] by simp
  also have ... = (1 / 2) *R ((Rz  $\widehat{\sim}$  n) a - a)
    by (simp add: scaleR-right-diff-distrib)
  finally show (T  $\widehat{\sim}$  n) 0 = (1 / 2) *R ((Rz  $\widehat{\sim}$  n) a - a) .
qed

```

```

have T-orbit-inj:  $\bigwedge n m. (T \widehat{\sim} n) 0 = (T \widehat{\sim} m) 0 \implies n = m$ 
proof -
  fix n m :: nat
  assume eq: (T  $\widehat{\sim}$  n) 0 = (T  $\widehat{\sim}$  m) 0
  have (Rz  $\widehat{\sim}$  n) a = (Rz  $\widehat{\sim}$  m) a
    using eq T-pow-zero-diff by simp
  thus n = m
    by (rule Rz-power-a-inj)
qed

```

```

have orbit-disj:
 $\forall n m. n \neq m \longrightarrow ((T \widehat{\sim} n) \text{ ` } \{0\}) \cap ((T \widehat{\sim} m) \text{ ` } \{0\}) = \{\}$ 
  using T-orbit-inj by auto

```

```

define E where E = (⋃ n. (T ~ n) ‘ {0::real^3})
have T-E: T ‘ E = E – {0}
  unfolding E-def by (rule absorbing-rotation-shift[OF orbit-disj])
have zero-subset-E: {0::real^3} ⊆ E
proof
  fix x :: real^3
  assume x ∈ {0}
  hence x ∈ (T ~ 0) ‘ {0}
  by simp
  thus x ∈ E
  unfolding E-def by blast
qed
have E-subset-ball3: E ⊆ ball3
proof
  fix x
  assume x ∈ E
  then obtain n where x-eq: x = (T ~ n) 0
  unfolding E-def by blast
  have Rza-sphere: (Rz ~ n) a ∈ sphere2
  using SO3-funpow-preserves-sphere2[OF Rz-in-SO3 a-sphere] .
  have norm x = norm ((1 / 2) *R ((Rz ~ n) a – a))
  using x-eq T-pow-zero-diff by simp
  also have ... = (1 / 2) * norm ((Rz ~ n) a – a)
  by simp
  also have ... ≤ (1 / 2) * (norm ((Rz ~ n) a) + norm a)
  by (intro mult-left-mono norm-triangle-ineq4) simp
  also have ... = 1
  using Rza-sphere a-sphere by (simp add: sphere2-def)
  finally show x ∈ ball3
  by (simp add: ball3-def)
qed

let ?X = ball3 – {0::real^3}
let ?Y = ball3
let ?A = [E, ball3 – E]
let ?B = [T ‘ E, ball3 – E]
let ?e = [T, id]
let ?t = [U, id]
have A-cover: ?Y = (⋃ k < length ?A. ?A ! k)
  using E-subset-ball3 by (auto simp: lessThan-Suc)
have A-disj: pairwise-disjoint ?A
  unfolding pairwise-disjoint-def
proof (intro allI impI)
  fix i j
  assume i: i < length ?A and j: j < length ?A and ij: i ≠ j
  have (i = 0 ∧ j = 1) ∨ (i = 1 ∧ j = 0)
  using i j ij by auto
  thus ?A ! i ∩ ?A ! j = {}

```

```

    by auto
  qed
  have B-cover: ?X = (⋃ k < length ?B. ?B ! k)
  proof
    show ?X ⊆ (⋃ k < length ?B. ?B ! k)
    proof
      fix x
      assume x: x ∈ ?X
      show x ∈ (⋃ k < length ?B. ?B ! k)
      proof (cases x ∈ E)
        case True
          with x T-E show ?thesis
          by force
        next
          case False
          with x show ?thesis
          by force
      qed
    qed
  show (⋃ k < length ?B. ?B ! k) ⊆ ?X
  proof
    fix x
    assume x: x ∈ (⋃ k < length ?B. ?B ! k)
    then obtain k where k: k < length ?B and xk: x ∈ ?B ! k
    by blast
    have k = 0 ∨ k = 1
    using k by auto
    thus x ∈ ?X
    proof
      assume k = 0
      with xk T-E E-subset-ball3 show ?thesis
      by auto
    next
      assume k = 1
      with xk zero-subset-E show ?thesis
      by auto
    qed
  qed
  qed
  have B-disj: pairwise-disjoint ?B
  unfolding pairwise-disjoint-def
  proof (intro allI impI)
    fix i j
    assume i: i < length ?B and j: j < length ?B and ij: i ≠ j
    have (i = 0 ∧ j = 1) ∨ (i = 1 ∧ j = 0)
    using i j ij by auto
    thus ?B ! i ∩ ?B ! j = {}
    using T-E by auto
  qed

```

```

have e-left:  $\bigwedge k x. \llbracket k < \text{length } ?A; x \in ?A ! k \rrbracket$ 
   $\implies (?e ! k) x \in ?B ! k \wedge (?t ! k) ((?e ! k) x) = x$ 
proof -
  fix k x
  assume k:  $k < \text{length } ?A$  and x:  $x \in ?A ! k$ 
  have  $k = 0 \vee k = 1$ 
  using k by auto
  thus  $(?e ! k) x \in ?B ! k \wedge (?t ! k) ((?e ! k) x) = x$ 
  proof
    assume  $k = 0$ 
    with x U-T show ?thesis
      by auto
    next
    assume  $k = 1$ 
    with x show ?thesis
      by auto
  qed
qed
have t-left:  $\bigwedge k y. \llbracket k < \text{length } ?B; y \in ?B ! k \rrbracket$ 
   $\implies (?t ! k) y \in ?A ! k \wedge (?e ! k) ((?t ! k) y) = y$ 
proof -
  fix k y
  assume k:  $k < \text{length } ?B$  and y:  $y \in ?B ! k$ 
  have  $k = 0 \vee k = 1$ 
  using k by auto
  thus  $(?t ! k) y \in ?A ! k \wedge (?e ! k) ((?t ! k) y) = y$ 
  proof
    assume  $k = 0$ 
    then obtain x where  $x \in E$  and y-eq:  $y = T x$ 
    using y by auto
    hence  $U y = x$ 
    using U-T by simp
    with x y-eq T-U  $\langle k = 0 \rangle$  show ?thesis
      by simp
    next
    assume  $k = 1$ 
    with y show ?thesis
      by auto
  qed
qed

```

```

from ball3-minus-origin-paradoxical-strong obtain P Q ::  $(\text{real}^3)$  set list
and gP gQ ::  $(\text{real}^3 \Rightarrow \text{real}^3)$  list
where lenP:  $\text{length } P = \text{length } gP$ 
and lenQ:  $\text{length } Q = \text{length } gQ$ 
and gP-SO3:  $\text{set } gP \subseteq \text{SO3}$ 
and gQ-SO3:  $\text{set } gQ \subseteq \text{SO3}$ 
and source-disj:  $\text{pairwise-disjoint } (P @ Q)$ 
and imageP-disj:  $\text{pairwise-disjoint } (\text{map2 } (\lambda g A. g ' A) gP P)$ 

```

```

and imageQ-disj: pairwise-disjoint (map2 (λg A. g ‘ A) gQ Q)
and P-sub: ∀ i < length P. P ! i ⊆ ?X
and Q-sub: ∀ i < length Q. Q ! i ⊆ ?X
and source-cover: ?X = (∪ i < length P. P ! i) ∪ (∪ i < length Q. Q ! i)
and imageP-cover: ?X = (∪ i < length P. (gP ! i) ‘ (P ! i))
and imageQ-cover: ?X = (∪ i < length Q. (gQ ! i) ‘ (Q ! i))
by auto
have gP-inj: ∧ i. i < length P ⇒ inj (gP ! i)
  using SO3-inj gP-SO3 lenP nth-mem by auto
have gQ-inj: ∧ i. i < length Q ⇒ inj (gQ ! i)
  using SO3-inj gQ-SO3 lenQ nth-mem by auto
have mapsP: ∧ k i l. [k < length ?A; i < length P; l < length ?B]
  ⇒ transfer-map ?t gP ?e k i l ∈ {f. isometry f}
proof –
  fix k i l
  assume k: k < length ?A and i: i < length P and l: l < length ?B
  have gi-SO3: gP ! i ∈ SO3
    using gP-SO3 i lenP by auto
  have gi-iso: isometry (gP ! i)
    by (rule SO3-isometry[OF gi-SO3])
  have ek-iso: isometry (?e ! k)
    using T-iso k less-Suc-eq by fastforce
  have tl-iso: isometry (?t ! l)
    using U-iso l less-Suc-eq by fastforce
  show transfer-map ?t gP ?e k i l ∈ {f. isometry f}
    unfolding transfer-map-def
    by (simp add: isometry-compose tl-iso gi-iso ek-iso)
qed
have mapsQ: ∧ k i l. [k < length ?A; i < length Q; l < length ?B]
  ⇒ transfer-map ?t gQ ?e k i l ∈ {f. isometry f}
proof –
  fix k i l
  assume k: k < length ?A and i: i < length Q and l: l < length ?B
  have gi-SO3: gQ ! i ∈ SO3
    using gQ-SO3 i lenQ by auto
  have gi-iso: isometry (gQ ! i)
    by (rule SO3-isometry[OF gi-SO3])
  have ek-iso: isometry (?e ! k)
    using T-iso k less-Suc-eq by fastforce
  have tl-iso: isometry (?t ! l)
    using U-iso l less-Suc-eq by fastforce
  show transfer-map ?t gQ ?e k i l ∈ {f. isometry f}
    unfolding transfer-map-def
    by (simp add: isometry-compose tl-iso gi-iso ek-iso)
qed
have lenAB: length ?A = length ?B
  by simp
from transfer-partitioned-paradox[
  OF lenAB A-cover A-disj B-cover B-disj e-left t-left lenP lenQ source-disj

```

source-cover imageP-disj imageQ-disj imageP-cover imageQ-cover gP-inj

gQ-inj

```

  mapsP mapsQ]
obtain P' Q' :: (real^3) set list and hP hQ :: ((real^3) => (real^3)) list
where lenP': length P' = length hP
and lenQ': length Q' = length hQ
and hP-iso: set hP ⊆ {f. isometry f}
and hQ-iso: set hQ ⊆ {f. isometry f}
and source-disj': pairwise-disjoint (P' @ Q')
and imageP-disj': pairwise-disjoint (map2 (λg A. g ' A) hP P')
and imageQ-disj': pairwise-disjoint (map2 (λg A. g ' A) hQ Q')
and source-cover': ?Y = (⋃ i<length P'. P' ! i) ∪ (⋃ i<length Q'. Q' ! i)
and imageP-cover': ?Y = (⋃ i<length P'. (hP ! i) ' (P' ! i))
and imageQ-cover': ?Y = (⋃ i<length Q'. (hQ ! i) ' (Q' ! i))
by auto
have P'-sub: ∀ i < length P'. P' ! i ⊆ ball3
using source-cover' by blast
have Q'-sub: ∀ i < length Q'. Q' ! i ⊆ ball3
using source-cover' by blast
show ?thesis
proof (intro exI conjI)
show length P' = length hP
by (rule lenP')
show length Q' = length hQ
by (rule lenQ')
show ∀ g ∈ set hP. isometry g
using hP-iso by blast
show ∀ g ∈ set hQ. isometry g
using hQ-iso by blast
show pairwise-disjoint (P' @ Q')
by (rule source-disj')
show pairwise-disjoint (map2 (λg A. g ' A) hP P')
by (rule imageP-disj')
show pairwise-disjoint (map2 (λg A. g ' A) hQ Q')
by (rule imageQ-disj')
show ∀ i < length P'. P' ! i ⊆ ball3
by (rule P'-sub)
show ∀ i < length Q'. Q' ! i ⊆ ball3
by (rule Q'-sub)
show ball3 = (⋃ i < length P'. P' ! i) ∪ (⋃ i < length Q'. Q' ! i)
by (rule source-cover')
show ball3 = (⋃ i < length P'. (hP ! i) ' (P' ! i))
by (rule imageP-cover')
show ball3 = (⋃ i < length Q'. (hQ ! i) ' (Q' ! i))
by (rule imageQ-cover')
qed
qed

```

An auxiliary finite-cover theorem for the closed unit ball.

theorem *banach-tarski-finite-cover*:

$\exists P :: (\text{real}^3) \text{ set list. } \exists gs :: ((\text{real}^3) \Rightarrow (\text{real}^3)) \text{ list.}$
 $\text{length } P = \text{length } gs \wedge$
 $(\forall g \in \text{set } gs. \text{isometry } g) \wedge$
 $(\forall i < \text{length } P. P ! i \subseteq \text{ball3}) \wedge$
 $\text{ball3} = (\bigcup_{i < \text{length } P. P ! i) \wedge$
 $(\text{ball3-copy-left} \cup \text{ball3-copy-right}) =$
 $(\bigcup_{i < \text{length } P. (gs ! i) ' (P ! i)) \wedge$
 $\text{ball3-copy-left} \cap \text{ball3-copy-right} = \{\}$

proof –

from *ball3-paradoxical* **obtain** $P \ Q :: (\text{real}^3) \text{ set list}$
and $gP \ gQ :: ((\text{real}^3 \Rightarrow \text{real}^3)) \text{ list}$

where *decomp*:

$\text{length } P = \text{length } gP \wedge \text{length } Q = \text{length } gQ \wedge$
 $\text{set } gP \subseteq \text{SO3} \wedge \text{set } gQ \subseteq \text{SO3} \wedge$
 $(\forall i < \text{length } P. P ! i \subseteq \text{ball3}) \wedge$
 $(\forall i < \text{length } Q. Q ! i \subseteq \text{ball3}) \wedge$
 $\text{ball3} = (\bigcup_{i < \text{length } P. P ! i) \wedge$
 $\text{ball3} = (\bigcup_{i < \text{length } Q. Q ! i) \wedge$
 $\text{ball3} = (\bigcup_{i < \text{length } P. (gP ! i) ' (P ! i)) \wedge$
 $\text{ball3} = (\bigcup_{i < \text{length } Q. (gQ ! i) ' (Q ! i))$

by (*elim exE*)

from *decomp* **have** *lenP*: $\text{length } P = \text{length } gP$

by (*elim conjE*)

from *decomp* **have** *lenQ*: $\text{length } Q = \text{length } gQ$

by (*elim conjE*)

from *decomp* **have** *gP-SO3*: $\text{set } gP \subseteq \text{SO3}$

by (*elim conjE*)

from *decomp* **have** *gQ-SO3*: $\text{set } gQ \subseteq \text{SO3}$

by (*elim conjE*)

from *decomp* **have** *P-sub*: $\forall i < \text{length } P. P ! i \subseteq \text{ball3}$

by (*elim conjE*)

from *decomp* **have** *Q-sub*: $\forall i < \text{length } Q. Q ! i \subseteq \text{ball3}$

by (*elim conjE*)

from *decomp* **have** *P-src*: $\text{ball3} = (\bigcup_{i < \text{length } P. P ! i)$

by (*elim conjE*)

from *decomp* **have** *Q-src*: $\text{ball3} = (\bigcup_{i < \text{length } Q. Q ! i)$

by (*elim conjE*)

from *decomp* **have** *P-cov*: $\text{ball3} = (\bigcup_{i < \text{length } P. (gP ! i) ' (P ! i))$

by (*elim conjE*)

from *decomp* **have** *Q-cov*: $\text{ball3} = (\bigcup_{i < \text{length } Q. (gQ ! i) ' (Q ! i))$

by (*elim conjE*)

define *pieces* **where** $\text{pieces} = P @ Q$

define *hQ* **where** $hQ = \text{map } (\lambda g. (+) \text{shift3} \circ g) \ gQ$

define *gs* **where** $gs = gP @ hQ$

have *len-hQ*: $\text{length } Q = \text{length } hQ$

by (*simp add: hQ-def lenQ*)

have *len*: $\text{length } \text{pieces} = \text{length } gs$

by (*simp add: pieces-def gs-def lenP len-hQ*)
have *iso*: $\forall g \in \text{set } gs. \text{isometry } g$
 using *gP-SO3 gQ-SO3*
 by (*auto simp add: gs-def hQ-def SO3-isometry*)
have *sub*: $\forall i < \text{length } pieces. pieces ! i \subseteq ball3$
 using *P-sub Q-sub* by (*auto simp add: pieces-def nth-append*)
have *src*: $ball3 = (\bigcup_{i < \text{length } pieces. pieces ! i)$
proof –
have *append-union*:
 $(\bigcup_{i < \text{length } (P @ Q). (P @ Q) ! i) =$
 $(\bigcup_{i < \text{length } P. P ! i) \cup (\bigcup_{i < \text{length } Q. Q ! i)$
 using *indexed-union-append[of P Q]* .
have $(\bigcup_{i < \text{length } pieces. pieces ! i) =$
 $(\bigcup_{i < \text{length } P. P ! i) \cup (\bigcup_{i < \text{length } Q. Q ! i)$
 unfolding *pieces-def*
 using *append-union* .
also have $\dots = ball3$
 using *P-src Q-src* by *simp*
finally show *?thesis* by *simp*
qed
have *right-cov*:
 $ball3\text{-copy-right} = (\bigcup_{i < \text{length } Q. (hQ ! i) ' (Q ! i))$
proof –
have *right-union*:
 $(\bigcup_{i < \text{length } Q. (hQ ! i) ' (Q ! i)) =$
 $(+) \text{shift3} ' (\bigcup_{i < \text{length } Q. (gQ ! i) ' (Q ! i))$
proof (*rule equalityI*)
show $(\bigcup_{i < \text{length } Q. (hQ ! i) ' (Q ! i)) \subseteq$
 $(+) \text{shift3} ' (\bigcup_{i < \text{length } Q. (gQ ! i) ' (Q ! i))$
proof
fix *x* **assume** $x \in (\bigcup_{i < \text{length } Q. (hQ ! i) ' (Q ! i))$
then obtain *i y* **where** *i-lt*: $i < \text{length } Q$ **and** *y-in*: $y \in Q ! i$
and *x-eq*: $x = (hQ ! i) y$
by *blast*
have *h-eq*: $hQ ! i = (+) \text{shift3} \circ (gQ ! i)$
using *i-lt lenQ* **by** (*simp add: hQ-def*)
have $(gQ ! i) y \in (\bigcup_{i < \text{length } Q. (gQ ! i) ' (Q ! i))$
using *i-lt y-in* **by** *blast*
moreover have $x = \text{shift3} + (gQ ! i) y$
using *x-eq h-eq* **by** *simp*
ultimately show $x \in (+) \text{shift3} ' (\bigcup_{i < \text{length } Q. (gQ ! i) ' (Q ! i))$
by *blast*
qed
show $(+) \text{shift3} ' (\bigcup_{i < \text{length } Q. (gQ ! i) ' (Q ! i)) \subseteq$
 $(\bigcup_{i < \text{length } Q. (hQ ! i) ' (Q ! i))$
proof
fix *x* **assume** $x \in (+) \text{shift3} ' (\bigcup_{i < \text{length } Q. (gQ ! i) ' (Q ! i))$
then obtain *z i y* **where** *x-eq*: $x = \text{shift3} + z$
and *i-lt*: $i < \text{length } Q$ **and** *y-in*: $y \in Q ! i$ **and** *z-eq*: $z = (gQ ! i) y$

```

    by blast
  have h-eq: hQ ! i = (+) shift3 ∘ (gQ ! i)
    using i-lt lenQ by (simp add: hQ-def)
  have x = (hQ ! i) y
    using x-eq z-eq h-eq by simp
  thus x ∈ (⋃ i < length Q. (hQ ! i) ‘ (Q ! i))
    using i-lt y-in by blast
qed
qed
show ?thesis
  unfolding ball3-copy-right-def
  using right-union Q-cov by simp
qed
have images:
  ball3-copy-left ∪ ball3-copy-right =
  (⋃ i < length pieces. (gs ! i) ‘ (pieces ! i))
proof -
  have append-images:
    (⋃ i < length (P @ Q). (gP @ hQ) ! i ‘ ((P @ Q) ! i)) =
    (⋃ i < length P. (gP ! i) ‘ (P ! i)) ∪
    (⋃ i < length Q. (hQ ! i) ‘ (Q ! i))
    using indexed-image-union-append[OF lenP len-hQ] .
  have (⋃ i < length pieces. (gs ! i) ‘ (pieces ! i)) =
    (⋃ i < length P. (gP ! i) ‘ (P ! i)) ∪
    (⋃ i < length Q. (hQ ! i) ‘ (Q ! i))
    unfolding pieces-def gs-def
    using append-images .
  also have ... = ball3-copy-left ∪ ball3-copy-right
    using P-cov right-cov by (simp add: ball3-copy-left-def)
  finally show ?thesis by simp
qed
show ?thesis
  apply (rule exI[of - pieces])
  apply (rule exI[of - gs])
  using len iso sub src images ball3-copies-disjoint
  by simp
qed

```

The Banach–Tarski theorem for the closed unit ball, as a partition theorem.

theorem *banach-tarski*:

$$\begin{aligned}
& \exists P :: (\mathbb{R}^3) \text{ set list. } \exists gs :: ((\mathbb{R}^3) \Rightarrow (\mathbb{R}^3)) \text{ list.} \\
& \text{length } P = \text{length } gs \wedge \\
& (\forall g \in \text{set } gs. \text{isometry } g) \wedge \\
& \text{pairwise-disjoint } P \wedge \\
& \text{pairwise-disjoint } (\text{map2 } (\lambda g A. g ‘ A) gs P) \wedge \\
& (\forall i < \text{length } P. P ! i \subseteq \text{ball3}) \wedge \\
& \text{ball3} = (\bigcup i < \text{length } P. P ! i) \wedge \\
& (\text{ball3-copy-left} \cup \text{ball3-copy-right}) = \\
& (\bigcup i < \text{length } P. (gs ! i) ‘ (P ! i)) \wedge
\end{aligned}$$

$$\text{ball3-copy-left} \cap \text{ball3-copy-right} = \{\}$$

proof –

from *ball3-paradoxical-strong* **obtain** $P\ Q :: (\text{real}^3)$ *set list*
and $gP\ gQ :: ((\text{real}^3 \Rightarrow \text{real}^3))$ *list*
where $\text{len}P$: $\text{length } P = \text{length } gP$
and $\text{len}Q$: $\text{length } Q = \text{length } gQ$
and $gP\text{-iso}$: $\forall g \in \text{set } gP. \text{isometry } g$
and $gQ\text{-iso}$: $\forall g \in \text{set } gQ. \text{isometry } g$
and source-disj : $\text{pairwise-disjoint } (P \text{ @ } Q)$
and $\text{image}P\text{-disj}$: $\text{pairwise-disjoint } (\text{map2 } (\lambda g\ A. g \text{ ' } A) gP\ P)$
and $\text{image}Q\text{-disj}$: $\text{pairwise-disjoint } (\text{map2 } (\lambda g\ A. g \text{ ' } A) gQ\ Q)$
and $P\text{-sub}$: $\forall i < \text{length } P. P ! i \subseteq \text{ball3}$
and $Q\text{-sub}$: $\forall i < \text{length } Q. Q ! i \subseteq \text{ball3}$
and source-cover : $\text{ball3} = (\bigcup i < \text{length } P. P ! i) \cup (\bigcup i < \text{length } Q. Q ! i)$
and $P\text{-cov}$: $\text{ball3} = (\bigcup i < \text{length } P. (gP ! i) \text{ ' } (P ! i))$
and $Q\text{-cov}$: $\text{ball3} = (\bigcup i < \text{length } Q. (gQ ! i) \text{ ' } (Q ! i))$

by *auto*

define pieces **where** $\text{pieces} = P \text{ @ } Q$
define hQ **where** $hQ = \text{map } (\lambda g. (+) \text{ shift3 } \circ g) gQ$
define gs **where** $gs = gP \text{ @ } hQ$
have $\text{len-h}Q$: $\text{length } Q = \text{length } hQ$
by (*simp add: hQ-def lenQ*)
have len : $\text{length } \text{pieces} = \text{length } gs$
by (*simp add: pieces-def gs-def lenP len-hQ*)
have iso : $\forall g \in \text{set } gs. \text{isometry } g$
using $gP\text{-iso } gQ\text{-iso}$
by (*auto simp add: gs-def hQ-def isometry-compose*)
have sub : $\forall i < \text{length } \text{pieces}. \text{pieces} ! i \subseteq \text{ball3}$
using $P\text{-sub } Q\text{-sub}$ **by** (*auto simp add: pieces-def nth-append*)
have src : $\text{ball3} = (\bigcup i < \text{length } \text{pieces}. \text{pieces} ! i)$
by (*metis indexed-union-append pieces-def source-cover*)
have $\text{source-disj-pieces}$: $\text{pairwise-disjoint } \text{pieces}$
using source-disj **by** (*simp add: pieces-def*)

have right-cov :

$$\text{ball3-copy-right} = (\bigcup i < \text{length } Q. (hQ ! i) \text{ ' } (Q ! i))$$

proof –

have right-union :

$$\begin{aligned} & (\bigcup i < \text{length } Q. (hQ ! i) \text{ ' } (Q ! i)) = \\ & (+) \text{ shift3 } \text{ ' } (\bigcup i < \text{length } Q. (gQ ! i) \text{ ' } (Q ! i)) \end{aligned}$$

proof (*rule equalityI*)

$$\begin{aligned} & \text{show } (\bigcup i < \text{length } Q. (hQ ! i) \text{ ' } (Q ! i)) \subseteq \\ & (+) \text{ shift3 } \text{ ' } (\bigcup i < \text{length } Q. (gQ ! i) \text{ ' } (Q ! i)) \end{aligned}$$

proof

fix x **assume** $x \in (\bigcup i < \text{length } Q. (hQ ! i) \text{ ' } (Q ! i))$
then obtain $i\ y$ **where** $i\text{-lt}$: $i < \text{length } Q$ **and** $y\text{-in}$: $y \in Q ! i$
and $x\text{-eq}$: $x = (hQ ! i) y$
by *blast*
have $h\text{-eq}$: $hQ ! i = (+) \text{ shift3 } \circ (gQ ! i)$

```

    using i-lt lenQ by (simp add: hQ-def)
  have  $(gQ ! i) y \in (\bigcup_{i < \text{length } Q}. (gQ ! i) \text{ ' } (Q ! i))$ 
    using i-lt y-in by blast
  moreover have  $x = \text{shift3} + (gQ ! i) y$ 
    using x-eq h-eq by simp
  ultimately show  $x \in (+) \text{shift3} \text{ ' } (\bigcup_{i < \text{length } Q}. (gQ ! i) \text{ ' } (Q ! i))$ 
    by blast
qed
show  $(+) \text{shift3} \text{ ' } (\bigcup_{i < \text{length } Q}. (gQ ! i) \text{ ' } (Q ! i)) \subseteq$ 
   $(\bigcup_{i < \text{length } Q}. (hQ ! i) \text{ ' } (Q ! i))$ 
proof
  fix x assume  $x \in (+) \text{shift3} \text{ ' } (\bigcup_{i < \text{length } Q}. (gQ ! i) \text{ ' } (Q ! i))$ 
  then obtain z i y where x-eq:  $x = \text{shift3} + z$ 
    and i-lt:  $i < \text{length } Q$  and y-in:  $y \in Q ! i$  and z-eq:  $z = (gQ ! i) y$ 
    by blast
  have h-eq:  $hQ ! i = (+) \text{shift3} \circ (gQ ! i)$ 
    using i-lt lenQ by (simp add: hQ-def)
  have  $x = (hQ ! i) y$ 
    using x-eq z-eq h-eq by simp
  thus  $x \in (\bigcup_{i < \text{length } Q}. (hQ ! i) \text{ ' } (Q ! i))$ 
    using i-lt y-in by blast
qed
qed
show ?thesis
  unfolding ball3-copy-right-def
  using right-union Q-cov by simp
qed
have images:
   $\text{ball3-copy-left} \cup \text{ball3-copy-right} =$ 
   $(\bigcup_{i < \text{length } \text{pieces}}. (gs ! i) \text{ ' } (\text{pieces} ! i))$ 
proof -
  have  $(\bigcup_{i < \text{length } \text{pieces}}. (gs ! i) \text{ ' } (\text{pieces} ! i)) =$ 
   $(\bigcup_{i < \text{length } P}. (gP ! i) \text{ ' } (P ! i)) \cup$ 
   $(\bigcup_{i < \text{length } Q}. (hQ ! i) \text{ ' } (Q ! i))$ 
  unfolding pieces-def gs-def
  by (rule indexed-image-union-append[OF lenP len-hQ])
  also have  $\dots = \text{ball3-copy-left} \cup \text{ball3-copy-right}$ 
    using P-cov right-cov by (simp add: ball3-copy-left-def)
  finally show ?thesis by simp
qed

have shift-inj: inj  $((+) \text{shift3})$ 
  by (simp add: inj-def)
have right-image-disj: pairwise-disjoint  $(\text{map2 } (\lambda g A. g \text{ ' } A) hQ Q)$ 
  unfolding pairwise-disjoint-def
proof (intro allI impI)
  fix i j
  assume i:  $i < \text{length } (\text{map2 } (\lambda g A. g \text{ ' } A) hQ Q)$ 
    and j:  $j < \text{length } (\text{map2 } (\lambda g A. g \text{ ' } A) hQ Q)$ 

```

```

    and ij: i ≠ j
  have iQ: i < length Q and jQ: j < length Q
    using i j len-hQ by auto
  have base-disj: (gQ ! i ' (Q ! i)) ∩ (gQ ! j ' (Q ! j)) = {}
    using imageQ-disj lenQ iQ jQ ij by (simp add: pairwise-disjoint-def)
  have hi: hQ ! i = (+) shift3 ∘ (gQ ! i)
    using iQ lenQ by (simp add: hQ-def)
  have hj: hQ ! j = (+) shift3 ∘ (gQ ! j)
    using jQ lenQ by (simp add: hQ-def)
  show map2 (λg A. g ' A) hQ Q ! i ∩ map2 (λg A. g ' A) hQ Q ! j = {}
    using base-disj shift-inj iQ jQ len-hQ
    by (auto simp: hi hj inj-def)
qed
have left-sets-sub: ⋀A. A ∈ set (map2 (λg A. g ' A) gP P) ⇒ A ⊆ ball3-copy-left
proof -
  fix A
  assume A: A ∈ set (map2 (λg A. g ' A) gP P)
  then obtain i where i: i < length P and A-eq: A = (gP ! i) ' (P ! i)
    using lenP by (auto simp: in-set-conv-nth)
  have A ⊆ (⋃ i < length P. (gP ! i) ' (P ! i))
    using i A-eq by blast
  thus A ⊆ ball3-copy-left
    using P-cov by (simp add: ball3-copy-left-def)
qed
have right-sets-sub: ⋀A. A ∈ set (map2 (λg A. g ' A) hQ Q) ⇒ A ⊆ ball3-copy-right
proof -
  fix A
  assume A: A ∈ set (map2 (λg A. g ' A) hQ Q)
  then obtain i where i: i < length Q and A-eq: A = (hQ ! i) ' (Q ! i)
    using len-hQ by (auto simp: in-set-conv-nth)
  have A ⊆ (⋃ i < length Q. (hQ ! i) ' (Q ! i))
    using i A-eq by blast
  thus A ⊆ ball3-copy-right
    using right-cov by simp
qed
have image-cross-disj:
  ⋀A B. [A ∈ set (map2 (λg A. g ' A) gP P);
  B ∈ set (map2 (λg A. g ' A) hQ Q)] ⇒ A ∩ B = {}
proof -
  fix A B
  assume A: A ∈ set (map2 (λg A. g ' A) gP P)
  and B: B ∈ set (map2 (λg A. g ' A) hQ Q)
  have A ∩ B ⊆ ball3-copy-left ∩ ball3-copy-right
    using left-sets-sub[OF A] right-sets-sub[OF B] by blast
  thus A ∩ B = {}
    using ball3-copies-disjoint by blast
qed
have image-disj: pairwise-disjoint (map2 (λg A. g ' A) gs pieces)
  unfolding gs-def pieces-def

```

```

using pairwise-disjoint-appendI[OF imageP-disj right-image-disj image-cross-disj]
using lenP by fastforce
show ?thesis
proof (intro exI conjI)
  show length pieces = length gs
    by (rule len)
  show  $\forall g \in \text{set } gs. \text{ isometry } g$ 
    by (rule iso)
  show pairwise-disjoint pieces
    by (rule source-disj-pieces)
  show pairwise-disjoint (map2 ( $\lambda g A. g \text{ ` } A$ ) gs pieces)
    by (rule image-disj)
  show  $\forall i < \text{length pieces}. \text{ pieces } ! i \subseteq \text{ball3}$ 
    by (rule sub)
  show  $\text{ball3} = (\bigcup i < \text{length pieces}. \text{ pieces } ! i)$ 
    by (rule src)
  show  $\text{ball3-copy-left} \cup \text{ball3-copy-right} =$ 
     $(\bigcup i < \text{length pieces}. (gs ! i) \text{ ` } (\text{pieces } ! i))$ 
    by (rule images)
  show  $\text{ball3-copy-left} \cap \text{ball3-copy-right} = \{\}$ 
    by (rule ball3-copies-disjoint)
qed
qed

end

```

References

- [1] S. Banach and A. Tarski. Sur la décomposition des ensembles de points en parties respectivement congruentes. *Fundamenta Mathematicae*, 6(1):244–277, 1924. DOI: <https://doi.org/10.4064/fm-6-1-244-277>. Also available at <https://eudml.org/doc/214280>.
- [2] J. Breitner. Free groups. *Archive of Formal Proofs*, June 2010. <https://isa-afp.org/entries/Free-Groups.html>, Formal proof development.
- [3] D. de Rauglaudre. Formal proof of Banach–Tarski paradox. *Journal of Formalized Reasoning*, 10(1):37–49, 2017. DOI: <https://doi.org/10.6092/issn.1972-5787/6927>.
- [4] F. Hausdorff. Bemerkung über den Inhalt von Punktmengen. *Mathematische Annalen*, 75:428–433, 1914. DOI: <https://doi.org/10.1007/BF01563735>. Also available at <https://eudml.org/doc/158671>.
- [5] T. Tao. *An Introduction to Measure Theory*, volume 126 of *Graduate Studies in Mathematics*. American Mathematical Society, 2011. DOI: <https://doi.org/10.1090/gsm/126>.

- [6] G. Tomkiewicz and S. Wagon. *The Banach–Tarski Paradox*, volume 163 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2nd edition, 2016. DOI: <https://doi.org/10.1017/CBO9781107337145>.
- [7] S. Wagon. *The Banach–Tarski Paradox*, volume 24 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1993. First paperback edition. DOI: <https://doi.org/10.1017/CBO9780511609596>.