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

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*) *vec* (the type (*real*, *3*) *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*
⟨*proof*⟩

lemma *zero-in-ball3*: *0* ∈ *ball3*
⟨*proof*⟩

lemma *zero-notin-sphere2*: (*0*::*real^3*) ∉ *sphere2*
⟨*proof*⟩

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* []
⟨*proof*⟩

lemma *pairwise-disjoint-singleton* [*simp*]: *pairwise-disjoint* [*X*]
⟨*proof*⟩

lemma *pairwise-disjoint-nthD*:
assumes *pairwise-disjoint Xs*
and $i < length\ Xs$ **and** $j < length\ Xs$ **and** $i \neq j$
shows $Xs\ !\ i \cap Xs\ !\ j = \{\}$
⟨*proof*⟩

lemma *indexed-union-set*:
fixes $P :: 'a\ set\ list$
shows $(\bigcup i{<}length\ P.\ P\ !\ i) = \bigcup (set\ P)$
⟨*proof*⟩

lemma *indexed-union-append*:
fixes $P\ Q :: 'a\ set\ list$
shows $(\bigcup i{<}length\ (P\ @\ Q).\ (P\ @\ Q)\ !\ i) =$
$(\bigcup i{<}length\ P.\ P\ !\ i) \cup (\bigcup i{<}length\ Q.\ Q\ !\ i)$
⟨*proof*⟩

lemma *indexed-image-union-zip*:
fixes $P :: 'a\ set\ list$ **and** $G :: ('a \Rightarrow 'b)\ list$
assumes $length\ P = length\ G$
shows $(\bigcup i{<}length\ P.\ G\ !\ i\ `\ (P\ !\ i)) =$
$(\bigcup (g,\ A){\in}set\ (zip\ G\ P).\ g\ `\ A)$
⟨*proof*⟩

lemma *indexed-image-union-append*:
fixes $P\ Q :: 'a\ set\ list$ **and** $G\ H :: ('a \Rightarrow 'b)\ list$
assumes *lenP*: $length\ P = length\ G$ **and** *lenQ*: $length\ Q = length\ H$
shows $(\bigcup i{<}length\ (P\ @\ Q).\ (G\ @\ H)\ !\ i\ `\ ((P\ @\ Q)\ !\ i)) =$
$(\bigcup i{<}length\ P.\ G\ !\ i\ `\ (P\ !\ i)) \cup$
$(\bigcup i{<}length\ Q.\ H\ !\ i\ `\ (Q\ !\ i))$
⟨*proof*⟩

lemma *indexed-union-map-fst*:
fixes $xs :: ('a\ set \times 'b)\ list$
shows $(\bigcup i{<}length\ (map\ fst\ xs).\ map\ fst\ xs\ !\ i) =$
$(\bigcup x{\in}set\ xs.\ fst\ x)$
⟨*proof*⟩

lemma *indexed-image-union-pairs*:
fixes $xs :: ('a\ set \times ('a \Rightarrow 'b))\ list$
shows $(\bigcup i{<}length\ (map\ fst\ xs).\ map\ snd\ xs\ !\ i\ `\ (map\ fst\ xs\ !\ i)) =$
$(\bigcup x{\in}set\ xs.\ snd\ x\ `\ fst\ x)$
⟨*proof*⟩

lemma *pairwise-disjoint-map-fstI*:
fixes $xs :: ('a\ set \times 'b)\ list$
assumes *distinct*: $distinct\ xs$
and *disj*: $\bigwedge x\ y.\ [\![x \in set\ xs;\ y \in set\ xs;\ x \neq y]\!]$

$\Longrightarrow$ *fst x* $\cap$ *fst y* = {}
shows *pairwise-disjoint* (*map fst xs*)
⟨*proof*⟩

lemma *pairwise-disjoint-appendI*:
assumes *xs*: *pairwise-disjoint xs*
and *ys*: *pairwise-disjoint ys*
and *cross*: $\bigwedge x\ y.\ [\![x \in set\ xs;\ y \in set\ ys]\!] \Longrightarrow x \cap y = \{\}$
shows *pairwise-disjoint* (*xs* @ *ys*)
⟨*proof*⟩

lemma *pairwise-disjoint-appendD1*:
assumes *pairwise-disjoint* (*xs* @ *ys*)
shows *pairwise-disjoint xs*
⟨*proof*⟩

lemma *pairwise-disjoint-appendD2*:
assumes *pairwise-disjoint* (*xs* @ *ys*)
shows *pairwise-disjoint ys*
⟨*proof*⟩

lemma *pairwise-disjoint-append-crossD*:
assumes *pairwise-disjoint* (*xs* @ *ys*)
and *i* < *length xs* **and** *j* < *length ys*
shows *xs* ! *i* $\cap$ *ys* ! *j* = {}
⟨*proof*⟩

lemma *pairwise-disjoint-pair-imagesI*:
fixes *xs* :: (*'a set* × (*'a* ⇒ *'b*)) *list*
assumes *distinct*: *distinct xs*
and *disj*: $\bigwedge x\ y.\ [\![x \in set\ xs;\ y \in set\ xs;\ x \neq y]\!]$
$\Longrightarrow$ *snd x ‘ fst x* $\cap$ *snd y ‘ fst y* = {}
shows *pairwise-disjoint* (*map2* (λ*g A*. *g ‘ A*) (*map snd xs*) (*map fst xs*))
⟨*proof*⟩

3 Transporting finite partitions

definition *transfer-piece* ::
'a set list ⇒ (*'a* ⇒ *'a*) *list* ⇒ *'a set list* ⇒
(*'a* ⇒ *'a*) *list* ⇒ *'a set list* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *'a set*
where
transfer-piece A e P g B k i l =
{*x* ∈ *A* ! *k*. (*e* ! *k*) *x* ∈ *P* ! *i* ∧ (*g* ! *i*) ((*e* ! *k*) *x*) ∈ *B* ! *l*}

definition *transfer-map* ::
(*'a* ⇒ *'a*) *list* ⇒ (*'a* ⇒ *'a*) *list* ⇒
(*'a* ⇒ *'a*) *list* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ (*'a* ⇒ *'a*)
where
transfer-map t g e k i l = (*t* ! *l*) ∘ (*g* ! *i*) ∘ (*e* ! *k*)

definition *transfer-pairs* ::
  'a set list ⇒ (*'a* ⇒ *'a*) *list* ⇒ *'a set list* ⇒
    (*'a* ⇒ *'a*) *list* ⇒ *'a set list* ⇒ (*'a* ⇒ *'a*) *list* ⇒
    (*'a set* × (*'a* ⇒ *'a*)) *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*) |
      k i l. *k* < *length A* ∧ *i* < *length P* ∧ *l* < *length B*}

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

lemma *transfer-source-cover*:
  assumes *A-cover*: *Y* = (⋃*k*<*length A*. *A* ! *k*)
    and *B-cover*: *X* = (⋃*k*<*length B*. *B* ! *k*)
    and *e-into-B*: ⋀*k x*. ⟦*k* < *length A*; *x* ∈ *A* ! *k*⟧ ⟹ (*e* ! *k*) *x* ∈ *B* ! *k*
    and *len-AB*: *length A* = *length B*
    and *source-cover*: *X* = (⋃*i*<*length P*. *P* ! *i*) ∪ (⋃*j*<*length Q*. *Q* ! *j*)
    and *imageP-cover*: *X* = (⋃*i*<*length P*. *gP* ! *i* ‘ (*P* ! *i*))
    and *imageQ-cover*: *X* = (⋃*j*<*length Q*. *gQ* ! *j* ‘ (*Q* ! *j*))
  shows *Y* =
    (⋃*p*∈*transfer-pairs A e P gP B t*. *fst p*) ∪
    (⋃*q*∈*transfer-pairs A e Q gQ B t*. *fst q*)
⟨*proof*⟩

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*: *length A* = *length B*
    and *e-left*: ⋀*k x*. ⟦*k* < *length A*; *x* ∈ *A* ! *k*⟧
      ⟹ (*e* ! *k*) *x* ∈ *B* ! *k* ∧ (*t* ! *k*) ((*e* ! *k*) *x*) = *x*
    and *t-left*: ⋀*k y*. ⟦*k* < *length B*; *y* ∈ *B* ! *k*⟧
      ⟹ (*t* ! *k*) *y* ∈ *A* ! *k* ∧ (*e* ! *k*) ((*t* ! *k*) *y*) = *y*
    and *P-sub*: ⋀*i*. *i* < *length P* ⟹ *P* ! *i* ⊆ *X*
    and *image-cover*: *X* = (⋃*i*<*length P*. *g* ! *i* ‘ (*P* ! *i*))
  shows *Y* = (⋃*p*∈*transfer-pairs A e P g B t*. *snd p* ‘ *fst p*)
⟨*proof*⟩

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*⟩

lemma *transfer-source-disjoint-cross*:
  assumes *A-disj*: *pairwise-disjoint A*
    and *PQ-cross*: $\bigwedge i\ j.\ [\![i < \mathit{length}\ P;\ j < \mathit{length}\ Q]\!] \Longrightarrow P\ !\ i \cap Q\ !\ j = \{\}$
    and *B-disj*: *pairwise-disjoint B*
    and *x-in*: $x \in \mathit{transfer\text{-}pairs}\ A\ e\ P\ gP\ B\ t$
    and *y-in*: $y \in \mathit{transfer\text{-}pairs}\ A\ e\ Q\ gQ\ B\ t$
  shows $\mathit{fst}\ x \cap \mathit{fst}\ y = \{\}$
⟨*proof*⟩

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* ($\lambda h\ C.\ h$ ‘ C) *g P*)
    and *len*: *length P* = *length g*
    and *g-inj*: $\bigwedge i.\ i < \mathit{length}\ P \Longrightarrow \mathit{inj}\ (g\ !\ i)$
    and *e-into-B*: $\bigwedge k\ x.\ [\![k < \mathit{length}\ A;\ x \in A\ !\ k]\!] \Longrightarrow (e\ !\ k)\ x \in B\ !\ k$
    and *t-left*: $\bigwedge l\ y.\ [\![l < \mathit{length}\ B;\ y \in B\ !\ l]\!]$
      $\Longrightarrow (t\ !\ l)\ y \in A\ !\ l \wedge (e\ !\ l)\ ((t\ !\ l)\ y) = y$
    and *x-in*: $x \in \mathit{transfer\text{-}pairs}\ A\ e\ P\ g\ B\ t$
    and *y-in*: $y \in \mathit{transfer\text{-}pairs}\ A\ e\ P\ g\ B\ t$
    and *xy*: $x \neq y$
  shows $\mathit{snd}\ x$ ‘ $\mathit{fst}\ x \cap \mathit{snd}\ y$ ‘ $\mathit{fst}\ y = \{\}$
⟨*proof*⟩

lemma *transfer-partitioned-paradox*:
  assumes *len-AB*: *length A* = *length B*
    and *A-cover*: $Y = (\bigcup k{<}\mathit{length}\ A.\ A\ !\ k)$
    and *A-disj*: *pairwise-disjoint A*
    and *B-cover*: $X = (\bigcup k{<}\mathit{length}\ B.\ B\ !\ k)$
    and *B-disj*: *pairwise-disjoint B*
    and *e-left*: $\bigwedge k\ x.\ [\![k < \mathit{length}\ A;\ x \in A\ !\ k]\!]$
      $\Longrightarrow (e\ !\ k)\ x \in B\ !\ k \wedge (t\ !\ k)\ ((e\ !\ k)\ x) = x$
    and *t-left*: $\bigwedge k\ y.\ [\![k < \mathit{length}\ B;\ y \in B\ !\ k]\!]$
      $\Longrightarrow (t\ !\ k)\ y \in A\ !\ k \wedge (e\ !\ k)\ ((t\ !\ k)\ y) = y$
    and *lenP*: *length P* = *length gP*
    and *lenQ*: *length Q* = *length gQ*
    and *source-disj*: *pairwise-disjoint* (*P* @ *Q*)
    and *source-cover*: $X = (\bigcup i{<}\mathit{length}\ P.\ P\ !\ i) \cup (\bigcup j{<}\mathit{length}\ Q.\ Q\ !\ j)$
    and *imageP-disj*: *pairwise-disjoint* (*map2* ($\lambda h\ C.\ h$ ‘ C) *gP P*)
    and *imageQ-disj*: *pairwise-disjoint* (*map2* ($\lambda h\ C.\ h$ ‘ C) *gQ Q*)
    and *imageP-cover*: $X = (\bigcup i{<}\mathit{length}\ P.\ gP\ !\ i\ `\ (P\ !\ i))$
    and *imageQ-cover*: $X = (\bigcup j{<}\mathit{length}\ Q.\ gQ\ !\ j\ `\ (Q\ !\ j))$
    and *gP-inj*: $\bigwedge i.\ i < \mathit{length}\ P \Longrightarrow \mathit{inj}\ (gP\ !\ i)$
    and *gQ-inj*: $\bigwedge j.\ j < \mathit{length}\ Q \Longrightarrow \mathit{inj}\ (gQ\ !\ j)$
    and *mapsP*: $\bigwedge k\ i\ l.\ [\![k < \mathit{length}\ A;\ i < \mathit{length}\ P;\ l < \mathit{length}\ B]\!]$
      $\Longrightarrow \mathit{transfer\text{-}map}\ t\ gP\ e\ k\ i\ l \in M$
    and *mapsQ*: $\bigwedge k\ j\ l.\ [\![k < \mathit{length}\ A;\ j < \mathit{length}\ Q;\ l < \mathit{length}\ B]\!]$
      $\Longrightarrow \mathit{transfer\text{-}map}\ t\ gQ\ e\ k\ j\ l \in M$

shows $\exists\, P'\ Q' :: 'a\ set\ list.\ \exists\, gP'\ gQ' :: ('a \Rightarrow 'a)\ list.$
$length\ P' = length\ gP' \wedge length\ Q' = length\ gQ' \wedge$
$set\ gP' \subseteq M \wedge set\ gQ' \subseteq M \wedge$
$pairwise\text{-}disjoint\ (P' @ Q') \wedge$
$pairwise\text{-}disjoint\ (map2\ (\lambda h\ C.\ h\ `\ C)\ gP'\ P') \wedge$
$pairwise\text{-}disjoint\ (map2\ (\lambda h\ C.\ h\ `\ C)\ gQ'\ Q') \wedge$
$Y = (\bigcup i{<}length\ P'.\ P' !\ i) \cup (\bigcup j{<}length\ Q'.\ Q' !\ j) \wedge$
$Y = (\bigcup i{<}length\ P'.\ gP' !\ i\ `\ (P' !\ i)) \wedge$
$Y = (\bigcup j{<}length\ Q'.\ gQ' !\ j\ `\ (Q' !\ j))$
⟨*proof*⟩

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** ‹·› *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**
$image\text{-}set\ g\ A = act\ g\ `\ A$

lemma *image-set-unit*:

assumes $A \subseteq$ *ground*
shows *image-set unit* $A = A$
⟨*proof*⟩

lemma *orbit-self*:
assumes $x \in$ *ground*
shows $x \in$ *orbit* x
⟨*proof*⟩

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* ⇒ *bool* **where**
paradoxical $X \longleftrightarrow$
($\exists$ *P Q* :: *'a set list*. $\exists$ *gP gQ* :: *'g list*.
length P = *length gP* ∧ *length Q* = *length gQ* ∧
set gP ⊆ *carrier* ∧ *set gQ* ⊆ *carrier* ∧
pairwise-disjoint (*P* @ *Q*) ∧
pairwise-disjoint (*map2 image-set gP P*) ∧
pairwise-disjoint (*map2 image-set gQ Q*) ∧
($\bigcup$ *i*<*length P*. *P* ! *i*) ∪ ($\bigcup$ *i*<*length Q*. *Q* ! *i*) ⊆ *X* ∧
X = ($\bigcup$ *i*<*length P*. *image-set* (*gP* ! *i*) (*P* ! *i*)) ∧
X = ($\bigcup$ *i*<*length Q*. *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* ∈ *carrier p2* ∈ *carrier q1* ∈ *carrier q2* ∈ *carrier*
and *disj-P12*: *P1* ∩ *P2* = {}
and *disj-Q12*: *Q1* ∩ *Q2* = {}
and *disj-PQ*: (*P1* ∪ *P2*) ∩ (*Q1* ∪ *Q2*) = {}
and *sub*: *P1* ∪ *P2* ∪ *Q1* ∪ *Q2* ⊆ *X*
and *disj-imgP*: *image-set p1 P1* ∩ *image-set p2 P2* = {}
and *disj-imgQ*: *image-set q1 Q1* ∩ *image-set q2 Q2* = {}
and *cover-P*: *X* = *image-set p1 P1* ∪ *image-set p2 P2*

and *cover-Q*: $X = \mathit{image\text{-}set}\ q1\ Q1 \cup \mathit{image\text{-}set}\ q2\ Q2$
shows *paradoxical X*
⟨*proof*⟩

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* {}
⟨*proof*⟩

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* ⇒ *'a set* ⇒ *bool* **where**
equidecomposable X Y ⟷
(∃ *P Q* :: *'a set list*. ∃ *gs* :: *'g list*.
length P = *length Q* ∧ *length P* = *length gs* ∧
set gs ⊆ *carrier* ∧
pairwise-disjoint P ∧ *pairwise-disjoint Q* ∧
X = (⋃ *i*<*length P*. *P* ! *i*) ∧
Y = (⋃ *i*<*length Q*. *Q* ! *i*) ∧
(∀ *i* < *length P*. *Q* ! *i* = *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 $g = \mathit{unit}$.

context *group-action*
begin

definition *free-on* :: *'a set* ⇒ *bool* **where**
  free-on X ⟷
    (∀ *g* ∈ *carrier*. ∀ *x* ∈ *X*. *act g x* = *x* ⟶ *g* = *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 $\mathit{act}\ g\ m$ for some $g \in \mathit{carrier}$ 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* ⇒ *'a* ⇒ *bool* **where**
  orbit-eq x y ⟷ (∃ *g* ∈ *carrier*. *y* = *act g x*)

lemma *orbit-eq-refl*: *x* ∈ *ground* ⟹ *orbit-eq x x*
  ⟨*proof*⟩

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 $\{\mathit{act}\ g\ m \mid g \in P_i,\ m \in M\}$, 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* ⇒ *bool* **where**
transports-paradox X ⟷
(*X* ⊆ *ground* ∧
free-on X ∧
(∀ *x* ∈ *X*. ∀ *g* ∈ *carrier*. *act g x* ∈ *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* | *B*

abbreviation *Gen2* :: *gen2 set* **where**
Gen2 ≡ {*A*, *B*}

lemma *Gen2-finite* [*simp*]: *finite Gen2*
⟨*proof*⟩

lemma *Gen2-card* [*simp*]: *card Gen2* = *2*
⟨*proof*⟩

lemma *Gen2-UNIV*: *Gen2* = (*UNIV* :: *gen2 set*)
⟨*proof*⟩

11 The free group F_2 and its carrier

abbreviation *F2* :: (*bool* × *gen2*) *list monoid* **where**
F2 ≡ $\mathcal{F}_{Gen2}$

The carrier of F_2 consists of cancelled words over the alphabet *UNIV* × *Gen2* = *UNIV* × *UNIV*.

lemma *F2-is-group*: *group F2*
⟨*proof*⟩

lemma *F2-carrier-iff*:
w ∈ *carrier F2* ⟷ *w* ∈ *lists* (*UNIV* × *Gen2*) ∧ *canceled w*
⟨*proof*⟩

lemma *F2-carrier-alt*:
carrier F2 = {*w*. *canceled w*}
⟨*proof*⟩

lemma *in-F2-canceled* [*dest*]:
assumes *w* ∈ *carrier F2*
shows *canceled w*
⟨*proof*⟩

lemma *canceled-in-F2* [*intro*]:
assumes *canceled w*
shows *w* ∈ *carrier F2*
⟨*proof*⟩

lemma *canceled-iff-no-adjacent-canceling*:
canceled w ⟷ (∀ *i*. *Suc i* < *length w* ⟶ ¬ *canceling* (*w* ! *i*) (*w* ! *Suc i*))
⟨*proof*⟩

lemma *canceled-Cons-iff*:
canceled (*p* # *w*) ⟷ *canceled w* ∧ (*w* = [] ∨ ¬ *canceling p* (*hd w*))
⟨*proof*⟩

lemma *F2-ConsD*:
assumes *p* # *w* ∈ *carrier F2*
shows *w* ∈ *carrier F2* **and** *w* = [] ∨ ¬ *canceling p* (*hd w*)
⟨*proof*⟩

lemma *F2-ConsI*:
assumes *w* ∈ *carrier F2* **and** *w* = [] ∨ ¬ *canceling p* (*hd w*)
shows *p* # *w* ∈ *carrier F2*
⟨*proof*⟩

lemma *F2-one* [*simp*]: $\mathbf{1}_{F2}$ = []
⟨*proof*⟩

lemma *F2-mult*: $x \otimes_{F2} y = \mathit{normalize}\ (x\ @\ y)$
⟨*proof*⟩

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 $F_2 \setminus\setminus \{1\}$.

definition *starts-with* :: *bool* ⇒ *gen2* ⇒ (*bool* × *gen2*) *list set* **where**
starts-with b x = {*w* ∈ *carrier F2*. *w* ≠ [] ∧ *hd w* = (*b*, *x*)}

abbreviation *S-a* :: (*bool* × *gen2*) *list set* **where** *S-a* ≡ *starts-with False A*
abbreviation *S-aI* :: (*bool* × *gen2*) *list set* **where** *S-aI* ≡ *starts-with True A*
abbreviation *S-b* :: (*bool* × *gen2*) *list set* **where** *S-b* ≡ *starts-with False B*
abbreviation *S-bI* :: (*bool* × *gen2*) *list set* **where** *S-bI* ≡ *starts-with True B*

lemma *starts-with-subset*: *starts-with b x* ⊆ *carrier F2*
⟨*proof*⟩

lemma *starts-with-disjoint*:
assumes (*b1*, *x1*) ≠ (*b2*, *x2*)
shows *starts-with b1 x1* ∩ *starts-with b2 x2* = {}
⟨*proof*⟩

lemma *starts-with-pairwise-disjoint*:
shows *S-a* ∩ *S-aI* = {} *S-a* ∩ *S-b* = {} *S-a* ∩ *S-bI* = {}
S-aI ∩ *S-b* = {} *S-aI* ∩ *S-bI* = {} *S-b* ∩ *S-bI* = {}
[] ∉ *S-a* [] ∉ *S-aI* [] ∉ *S-b* [] ∉ *S-bI*
⟨*proof*⟩

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 = {[]} ∪ *S-a* ∪ *S-aI* ∪ *S-b* ∪ *S-bI*
⟨*proof*⟩

13 The two distinguished generators a and b

abbreviation *a-elt* :: (*bool* × *gen2*) *list* **where** *a-elt* ≡ [(*False*, *A*)]
abbreviation *aI-elt* :: (*bool* × *gen2*) *list* **where** *aI-elt* ≡ [(*True*, *A*)]
abbreviation *b-elt* :: (*bool* × *gen2*) *list* **where** *b-elt* ≡ [(*False*, *B*)]
abbreviation *bI-elt* :: (*bool* × *gen2*) *list* **where** *bI-elt* ≡ [(*True*, *B*)]

lemma *single-letter-canceled*:
canceled [(*b*, *x*)]
⟨*proof*⟩

lemma *a-elt-in-F2* [*simp*]: *a-elt* ∈ *carrier F2*

⟨*proof*⟩

lemma *aI-elt-in-F2* [*simp*]: *aI-elt* ∈ *carrier F2*
⟨*proof*⟩

lemma *b-elt-in-F2* [*simp*]: *b-elt* ∈ *carrier F2*
⟨*proof*⟩

lemma *bI-elt-in-F2* [*simp*]: *bI-elt* ∈ *carrier F2*
⟨*proof*⟩

lemma *inv-a-eq-aI*: inv_{F2} *a-elt* = *aI-elt*
⟨*proof*⟩

lemma *inv-b-eq-bI*: inv_{F2} *b-elt* = *bI-elt*
⟨*proof*⟩

lemma *inv-aI-eq-a*: inv_{F2} *aI-elt* = *a-elt*
⟨*proof*⟩

lemma *inv-bI-eq-b*: inv_{F2} *bI-elt* = *b-elt*
⟨*proof*⟩

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*
⟨*proof*⟩

interpretation *F2-act*:
group-action carrier F2 [] ($\otimes_{F2}$) ($\otimes_{F2}$) *carrier F2*
⟨*proof*⟩

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 $w \in$ *S-aI*
shows *a-elt* $\otimes_{F2}$ w = *tl* w
⟨*proof*⟩

lemma *b-mult-starts-with-bI*:
assumes $w \in$ *S-bI*
shows *b-elt* $\otimes_{F2}$ w = *tl* w
⟨*proof*⟩

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*⟩

lemma *image-b-S-bI*:
shows ($\otimes_{F2}$) *b-elt* ‘ *S-bI* = *carrier F2* − *S-b*
⟨*proof*⟩

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* ∪ ($\otimes_{F2}$) *a-elt* ‘ *S-aI* = *carrier F2*
⟨*proof*⟩

lemma *S-b-un-image-b-S-bI*: *S-b* ∪ ($\otimes_{F2}$) *b-elt* ‘ *S-bI* = *carrier F2*
⟨*proof*⟩

lemma *S-a-disj-image-a-S-aI*: *S-a* ∩ ($\otimes_{F2}$) *a-elt* ‘ *S-aI* = {}
⟨*proof*⟩

lemma *S-b-disj-image-b-S-bI*: *S-b* ∩ ($\otimes_{F2}$) *b-elt* ‘ *S-bI* = {}
⟨*proof*⟩

definition *aI-ray* :: (*bool* × *gen2*) *list set* **where**
aI-ray = {*replicate n* (*True*, *A*) | *n*. *True*}

lemma *canceled-replicate-aI*: *canceled* (*replicate n* (*True*, *A*))
⟨*proof*⟩

lemma *aI-ray-subset-carrier*: *aI-ray* ⊆ *carrier F2*
⟨*proof*⟩

lemma *aI-ray-cases*:

assumes $w \in$ *aI-ray*
shows $w = [] \vee w \in$ *S-aI*
⟨*proof*⟩

lemma *a-mult-aI-ray-pos*:
$(\otimes_{F2})$ *a-elt* ‘ (*aI-ray* − {[]}) = *aI-ray*
⟨*proof*⟩

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

lemma *image-a-S-aI-nonray*:
$(\otimes_{F2})$ *a-elt* ‘ (*S-aI* − *aI-ray*) = *carrier F2* − *S-a* − *aI-ray*
⟨*proof*⟩

theorem *F2-paradoxical-partition*:
∃ *P Q* :: ((*bool* × *gen2*) *list*) *set list*. ∃ *gP gQ* :: ((*bool* × *gen2*) *list*) *list*.
length P = *length gP* ∧ *length Q* = *length gQ* ∧
set gP ⊆ *carrier F2* ∧ *set gQ* ⊆ *carrier F2* ∧
pairwise-disjoint (*P* @ *Q*) ∧
pairwise-disjoint (*map2 F2-act.image-set gP P*) ∧
pairwise-disjoint (*map2 F2-act.image-set gQ Q*) ∧
carrier F2 = (⋃ *i*<*length P*. *P* ! *i*) ∪ (⋃ *i*<*length Q*. *Q* ! *i*) ∧
carrier F2 = (⋃ *i*<*length P*. *F2-act.image-set* (*gP* ! *i*) (*P* ! *i*)) ∧
carrier F2 = (⋃ *i*<*length Q*. *F2-act.image-set* (*gQ* ! *i*) (*Q* ! *i*))
⟨*proof*⟩

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

theorem *F2-paradoxical*:
F2-act.paradoxical (*carrier F2*)
⟨*proof*⟩

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*⟩

lemma *SO3-closed-compose*:
assumes *f* ∈ *SO3* **and** *g* ∈ *SO3*
shows *f* ∘ *g* ∈ *SO3*
⟨*proof*⟩

SO(3) elements preserve the unit sphere.

lemma *rotation-preserves-sphere2*:
assumes *f* ∈ *SO3* **and** *x* ∈ *sphere2*
shows *f x* ∈ *sphere2*
⟨*proof*⟩

lemma *SO3-bij*:
assumes *f* ∈ *SO3*
shows *bij f*
⟨*proof*⟩

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* ϑ = *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**
rot-c = *1 / 3*

definition *rot-s* :: *real* **where**
rot-s = *sqrt 8 / 3*

lemma *rot-c-sq-plus-rot-s-sq*: *rot-c*^{2} + *rot-s*^{2} = *1*
⟨*proof*⟩

lemma *rot-c-mul-plus-rot-s-mul* [*simp*]: *rot-c* ∗ *rot-c* + *rot-s* ∗ *rot-s* = *1*
⟨*proof*⟩

definition *Rx-mat* :: *real^3^3* **where**
Rx-mat = *vector* [
vector [*1*, *0*, *0*],
vector [*0*, *rot-c*, − *rot-s*],
vector [*0*, *rot-s*, *rot-c*]]

definition *Rz-mat* :: *real^3^3* **where**
Rz-mat = *vector* [
vector [*rot-c*, − *rot-s*, *0*],
vector [*rot-s*, *rot-c*, *0*],
vector [*0*, *0*, *1*]]

definition *Rx-inv-mat* :: *real^3^3* **where**
Rx-inv-mat = *vector* [
vector [*1*, *0*, *0*],
vector [*0*, *rot-c*, *rot-s*],
vector [*0*, − *rot-s*, *rot-c*]]

definition *Rz-inv-mat* :: *real^3^3* **where**
Rz-inv-mat = *vector* [
vector [*rot-c*, *rot-s*, *0*],
vector [− *rot-s*, *rot-c*, *0*],
vector [*0*, *0*, *1*]]

definition *Rx* :: *real^3* ⇒ *real^3* **where**
Rx v = *Rx-mat* ∗*v v*

definition *Rz* :: *real^3* ⇒ *real^3* **where**
Rz v = *Rz-mat* ∗*v v*

definition *Rx-inv* :: *real^3* ⇒ *real^3* **where**
Rx-inv v = *Rx-inv-mat* ∗*v v*

definition *Rz-inv* :: *real^3* ⇒ *real^3* **where**
Rz-inv v = *Rz-inv-mat* ∗*v v*

lemma *Rx-mat-orthogonal*: *orthogonal-matrix Rx-mat*
⟨*proof*⟩

lemma *Rz-mat-orthogonal*: *orthogonal-matrix Rz-mat*
⟨*proof*⟩

lemma *Rx-inv-mat-orthogonal*: *orthogonal-matrix Rx-inv-mat*
⟨*proof*⟩

lemma *Rz-inv-mat-orthogonal*: *orthogonal-matrix Rz-inv-mat*
⟨*proof*⟩

lemma *det-Rx-mat* [*simp*]: *matrix-det Rx-mat* = *1*

⟨*proof*⟩

lemma *det-Rz-mat* [*simp*]: *matrix-det Rz-mat = 1*
⟨*proof*⟩

lemma *det-Rx-inv-mat* [*simp*]: *matrix-det Rx-inv-mat = 1*
⟨*proof*⟩

lemma *det-Rz-inv-mat* [*simp*]: *matrix-det Rz-inv-mat = 1*
⟨*proof*⟩

lemma *Rx-mat-inverse-left*: *Rx-inv-mat ** Rx-mat = mat 1*
⟨*proof*⟩

lemma *Rx-mat-inverse-right*: *Rx-mat ** Rx-inv-mat = mat 1*
⟨*proof*⟩

lemma *Rz-mat-inverse-left*: *Rz-inv-mat ** Rz-mat = mat 1*
⟨*proof*⟩

lemma *Rz-mat-inverse-right*: *Rz-mat ** Rz-inv-mat = mat 1*
⟨*proof*⟩

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

lemma *Rx-in-SO3*: *Rx ∈ SO3*
⟨*proof*⟩

lemma *Rz-in-SO3*: *Rz ∈ SO3*
⟨*proof*⟩

lemma *Rx-inv-in-SO3*: *Rx-inv ∈ SO3*
⟨*proof*⟩

lemma *Rz-inv-in-SO3*: *Rz-inv ∈ SO3*
⟨*proof*⟩

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

lemma *Rx-inv-left*: *Rx-inv ∘ Rx = id*
⟨*proof*⟩

lemma *Rz-inv-left*: *Rz-inv ∘ Rz = id*
⟨*proof*⟩

lemma *Rx-inv-right*: *Rx ∘ Rx-inv = id*
⟨*proof*⟩

lemma *Rz-inv-right*: *Rz ∘ Rz-inv = id*
⟨*proof*⟩

20 The rotations as bijections of the sphere

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

lemma *SO3-bij-betw-sphere2*:
assumes $f \in SO3$
shows *bij-betw f sphere2 sphere2*
⟨*proof*⟩

lemma *sphere-perm-in-Bij*:
assumes $f \in SO3$
shows *sphere-perm f* $\in$ *Bij sphere2*
⟨*proof*⟩

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*⟩

lemma *sphere-perm-id*: *sphere-perm id* $= \mathbf{1}_{BijGroup\ sphere2}$
⟨*proof*⟩

lemma *sphere-perm-Rx* [*simp*]: *sphere-perm Rx* $\in$ *Bij sphere2*
⟨*proof*⟩

lemma *sphere-perm-Rx-inv* [*simp*]: *sphere-perm Rx-inv* $\in$ *Bij sphere2*
⟨*proof*⟩

lemma *sphere-perm-Rz* [*simp*]: *sphere-perm Rz* $\in$ *Bij sphere2*
⟨*proof*⟩

lemma *sphere-perm-Rz-inv* [*simp*]: *sphere-perm Rz-inv* $\in$ *Bij sphere2*
⟨*proof*⟩

lemma *sphere-perm-Rx-inv-left*:
sphere-perm Rx-inv $\otimes_{BijGroup\ sphere2}$ *sphere-perm Rx* $= \mathbf{1}_{BijGroup\ sphere2}$
⟨*proof*⟩

lemma *sphere-perm-Rx-inv-right*:
sphere-perm Rx $\otimes_{BijGroup\ sphere2}$ *sphere-perm Rx-inv* $= \mathbf{1}_{BijGroup\ sphere2}$
⟨*proof*⟩

lemma *sphere-perm-Rz-inv-left*:
sphere-perm Rz-inv $\otimes_{BijGroup\ sphere2}$ *sphere-perm Rz* $= \mathbf{1}_{BijGroup\ sphere2}$
⟨*proof*⟩

lemma *sphere-perm-Rz-inv-right*:
sphere-perm Rz $\otimes_{BijGroup\ sphere2}$ *sphere-perm Rz-inv* $= \mathbf{1}_{BijGroup\ sphere2}$

⟨*proof*⟩

lemma *inv-sphere-perm-Rx*:
inv$_{BijGroup\ sphere2}$ (*sphere-perm Rx*) = *sphere-perm Rx-inv*
⟨*proof*⟩

lemma *inv-sphere-perm-Rz*:
inv$_{BijGroup\ sphere2}$ (*sphere-perm Rz*) = *sphere-perm Rz-inv*
⟨*proof*⟩

interpretation *sphere-Bij*: *group BijGroup sphere2*
⟨*proof*⟩

21 The map sigma: F_2 to 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* × *gen2* ⇒ (*real^3* ⇒ *real^3*) **where**
letter-to-rot p = (*case p of*
(*False*, *A*) ⇒ *Rx*
| (*True*, *A*) ⇒ *Rx-inv*
| (*False*, *B*) ⇒ *Rz*
| (*True*, *B*) ⇒ *Rz-inv*)

definition *rho* :: *gen2* ⇒ (*real^3* ⇒ *real^3*) **where**
rho x = (*case x of A* ⇒ *sphere-perm Rx* | *B* ⇒ *sphere-perm Rz*)

lemma *rho-in-BijGroup*:
rho ∈ *Gen2* → *carrier* (*BijGroup sphere2*)
⟨*proof*⟩

lemma *rho-image-subset-BijGroup*:
rho ‘ *Gen2* ⊆ *carrier* (*BijGroup sphere2*)
⟨*proof*⟩

definition *sphere-rot-group* :: (*real^3* ⇒ *real^3*) *monoid* **where**
sphere-rot-group =
(*BijGroup sphere2*)(|*carrier* := ⟨*rho* ‘ *Gen2*⟩$_{BijGroup\ sphere2}$|)

lemma *sphere-rot-subgroup*:
subgroup (*carrier sphere-rot-group*) (*BijGroup sphere2*)
⟨*proof*⟩

lemma *sphere-rot-group-is-group*: *group sphere-rot-group*
⟨*proof*⟩

interpretation *sphere-rot*: *group sphere-rot-group*
⟨*proof*⟩

lemma *rho-in-sphere-rot-group*:
rho ∈ *Gen2* → *carrier sphere-rot-group*
⟨*proof*⟩

lemma *sphere-rot-inclusion-hom*:
id ∈ *hom sphere-rot-group* (*BijGroup sphere2*)
⟨*proof*⟩

lemma *sphere-rot-generated*:
⟨*rho* ‘ *Gen2*⟩$_{sphere\text{-}rot\text{-}group}$ = *carrier sphere-rot-group*
⟨*proof*⟩

lemma *sphere-Bij-lift-gi-rho*:
assumes *snd p* ∈ *Gen2*
shows *sphere-Bij.lift-gi rho p* = *sphere-perm* (*letter-to-rot p*)
⟨*proof*⟩

lemma *sphere-rot-inv-rho-eq-sphere-Bij*:
assumes *i* ∈ *Gen2*
shows *inv*$_{sphere\text{-}rot\text{-}group}$ (*rho i*) = *inv*$_{BijGroup\ sphere2}$ (*rho i*)
⟨*proof*⟩

lemma *sphere-rot-lift-gi-rho-eq-sphere-Bij*:
assumes *snd p* ∈ *Gen2*
shows *sphere-rot.lift-gi rho p* = *sphere-Bij.lift-gi rho p*
⟨*proof*⟩

lemma *sphere-rot-lift-rho-eq-sphere-Bij*:
assumes *w* ∈ *lists* (*UNIV* × *Gen2*)
shows *sphere-rot.lift rho w* = *sphere-Bij.lift rho w*
⟨*proof*⟩

lemma *sphere-Bij-lift-rho-injective-if-ping-pong*:
fixes *Xin Xout* :: *gen2* ⇒ (*real^3*) *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*. (*inv*$_{sphere\text{-}rot\text{-}group}$ (*rho i*)) ‘ (*sphere2* − *Xin i*) ⊆ *Xout i*
shows *inj-on* (*sphere-Bij.lift rho*) (*carrier F2*)
⟨*proof*⟩

definition *sigma* :: (*bool* × *gen2*) *list* ⇒ (*real^3* ⇒ *real^3*) **where**
sigma w = *foldr* (λ*p f*. *letter-to-rot p* ∘ *f*) *w id*

lemma *letter-to-rot-cancel-left* [*simp*]:
letter-to-rot (*True*, *A*) ∘ *letter-to-rot* (*False*, *A*) = *id*
letter-to-rot (*False*, *A*) ∘ *letter-to-rot* (*True*, *A*) = *id*
letter-to-rot (*True*, *B*) ∘ *letter-to-rot* (*False*, *B*) = *id*
letter-to-rot (*False*, *B*) ∘ *letter-to-rot* (*True*, *B*) = *id*
⟨*proof*⟩

lemma *letter-to-rot-canceling*:
assumes *canceling p q*
shows *letter-to-rot p* ∘ *letter-to-rot q* = *id*
⟨*proof*⟩

lemma *sigma-Nil* [*simp*]: *sigma* [] = *id*
⟨*proof*⟩

lemma *sigma-Cons* [*simp*]: *sigma* (*p* # *w*) = *letter-to-rot p* ∘ *sigma w*
⟨*proof*⟩

lemma *sigma-append* [*simp*]: *sigma* (*u* @ *v*) = *sigma u* ∘ *sigma v*
⟨*proof*⟩

lemma *sigma-singleton* [*simp*]: *sigma* [*p*] = *letter-to-rot p*
⟨*proof*⟩

lemma *sigma-doubleton-canceling*:
assumes *canceling p q*
shows *sigma* [*p*, *q*] = *id*
⟨*proof*⟩

lemma *sigma-cancels-to-1*:
assumes *cancels-to-1 x y*
shows *sigma x* = *sigma y*
⟨*proof*⟩

lemma *sigma-cancels-to*:
assumes *cancels-to x y*
shows *sigma x* = *sigma y*
⟨*proof*⟩

lemma *sigma-normalize* [*simp*]: *sigma* (*normalize w*) = *sigma w*
⟨*proof*⟩

lemma *letter-to-rot-in-SO3*: *letter-to-rot p* ∈ *SO3*
⟨*proof*⟩

lemma *sphere-perm-letter-to-rot-in-Bij* [*simp*]:
sphere-perm (*letter-to-rot p*) ∈ *Bij sphere2*
⟨*proof*⟩

lemma *sigma-in-SO3*: *sigma w* ∈ *SO3*
⟨*proof*⟩

lemma *sigma-preserves-sphere2*:
assumes *x* ∈ *sphere2*
shows *sigma w x* ∈ *sphere2*
⟨*proof*⟩

lemma *sigma-bij*: *bij* (*sigma w*)
⟨*proof*⟩

lemma *sphere-perm-sigma-in-Bij* [*simp*]:
sphere-perm (*sigma w*) ∈ *Bij sphere2*
⟨*proof*⟩

lemma *sphere-Bij-lift-rho-eq-sigma*:
assumes *w* ∈ *lists* (*UNIV* × *Gen2*)
shows *sphere-Bij.lift rho w* = *sphere-perm* (*sigma w*)
⟨*proof*⟩

lemma *sigma-injective-if-sphere-lift-injective*:
assumes *inj-on* (*sphere-Bij.lift rho*) (*carrier F2*)
shows *inj-on sigma* (*carrier F2*)
⟨*proof*⟩

lemma *sigma-injective-if-ping-pong*:
fixes *Xin Xout* :: *gen2* ⇒ (*real*^3) *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*. (*inv*$_{\textit{sphere-rot-group}}$ (*rho i*)) ‘ (*sphere2* − *Xin i*) ⊆ *Xout i*
shows *inj-on sigma* (*carrier F2*)
⟨*proof*⟩

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 *3*, so every word has entries in the integer quadratic ring generated by *sqrt 2*. A nonempty reduced word leaves a coefficient nonzero modulo *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* × *int*
type-synonym *zvec3* = *zsqrt2* × *zsqrt2* × *zsqrt2*

definition *zsqrt2-val* :: *zsqrt2* ⇒ *real* **where**
zsqrt2-val q = *of-int* (*fst q*) + *of-int* (*snd q*) ∗ *sqrt 2*

definition *zsqrt2-add* :: *zsqrt2* ⇒ *zsqrt2* ⇒ *zsqrt2* **where**
zsqrt2-add p q = (*fst p* + *fst q*, *snd p* + *snd q*)

definition *zsqrt2-neg* :: *zsqrt2* ⇒ *zsqrt2* **where**
zsqrt2-neg p = (− *fst p*, − *snd p*)

definition *zsqrt2-sub* :: *zsqrt2* ⇒ *zsqrt2* ⇒ *zsqrt2* **where**
zsqrt2-sub p q = *zsqrt2-add p* (*zsqrt2-neg q*)

definition *zsqrt2-scale* :: *int* ⇒ *zsqrt2* ⇒ *zsqrt2* **where**
zsqrt2-scale n p = (*n* ∗ *fst p*, *n* ∗ *snd p*)

definition *zsqrt2-mul-sqrt2* :: *zsqrt2* ⇒ *zsqrt2* **where**
zsqrt2-mul-sqrt2 p = (*2* ∗ *snd p*, *fst p*)

definition *zsqrt2-div3* :: *zsqrt2* ⇒ *bool* **where**
zsqrt2-div3 p ⟷ *3 dvd fst p* ∧ *3 dvd snd p*

definition *zvec3-div3* :: *zvec3* ⇒ *bool* **where**
zvec3-div3 v ⟷
(*case v of* (*x*, *y*, *z*) ⇒ *zsqrt2-div3 x* ∧ *zsqrt2-div3 y* ∧ *zsqrt2-div3 z*)

definition *zvec3-scale* :: *int* ⇒ *zvec3* ⇒ *zvec3* **where**
zvec3-scale n v = (*case v of* (*x*, *y*, *z*) ⇒
(*zsqrt2-scale n x*, *zsqrt2-scale n y*, *zsqrt2-scale n z*))

definition *zvec3-val* :: *zvec3* ⇒ *real^3* **where**
zvec3-val v = (*case v of* (*x*, *y*, *z*) ⇒
vector [*zsqrt2-val x*, *zsqrt2-val y*, *zsqrt2-val z*])

definition *ze-x* :: *zvec3* **where**
ze-x = ((*1*, *0*), (*0*, *0*), (*0*, *0*))

definition *ze-y* :: *zvec3* **where**
ze-y = ((*0*, *0*), (*1*, *0*), (*0*, *0*))

definition *ze-z* :: *zvec3* **where**
ze-z = ((*0*, *0*), (*0*, *0*), (*1*, *0*))

```
definition zRx-step :: zvec3 ⇒ zvec3 where
  zRx-step v = (case v of (x, y, z) ⇒
    (zsqrt2-scale 3 x,
     zsqrt2-sub y (zsqrt2-scale 2 (zsqrt2-mul-sqrt2 z)),
     zsqrt2-add (zsqrt2-scale 2 (zsqrt2-mul-sqrt2 y)) z))

definition zRx-inv-step :: zvec3 ⇒ zvec3 where
  zRx-inv-step v = (case v of (x, y, z) ⇒
    (zsqrt2-scale 3 x,
     zsqrt2-add y (zsqrt2-scale 2 (zsqrt2-mul-sqrt2 z)),
     zsqrt2-sub z (zsqrt2-scale 2 (zsqrt2-mul-sqrt2 y))))

definition zRz-step :: zvec3 ⇒ zvec3 where
  zRz-step v = (case v of (x, y, z) ⇒
    (zsqrt2-sub x (zsqrt2-scale 2 (zsqrt2-mul-sqrt2 y)),
     zsqrt2-add (zsqrt2-scale 2 (zsqrt2-mul-sqrt2 x)) y,
     zsqrt2-scale 3 z))

definition zRz-inv-step :: zvec3 ⇒ zvec3 where
  zRz-inv-step v = (case v of (x, y, z) ⇒
    (zsqrt2-add x (zsqrt2-scale 2 (zsqrt2-mul-sqrt2 y)),
     zsqrt2-sub y (zsqrt2-scale 2 (zsqrt2-mul-sqrt2 x)),
     zsqrt2-scale 3 z))

definition zletter-step :: bool × gen2 ⇒ zvec3 ⇒ zvec3 where
  zletter-step p = (case p of
      (False, A) ⇒ zRx-step
    | (True,  A) ⇒ zRx-inv-step
    | (False, B) ⇒ zRz-step
    | (True,  B) ⇒ zRz-inv-step)

definition zword-step :: (bool × gen2) list ⇒ zvec3 ⇒ zvec3 where
  zword-step w v = foldr (λp f. zletter-step p ∘ f) w id v

definition zword-witness :: (bool × gen2) list ⇒ zvec3 where
  zword-witness w = (case snd (last w) of A ⇒ ze-y | B ⇒ ze-x)

datatype r3 = R0 | R1 | R2

type-synonym rcoef = r3 × r3
type-synonym rvec3 = rcoef × rcoef × rcoef

fun r3-add :: r3 ⇒ r3 ⇒ r3 where
  r3-add R0 x = x
| r3-add x R0 = x
| r3-add R1 R1 = R2
| r3-add R1 R2 = R0
| r3-add R2 R1 = R0
| r3-add R2 R2 = R1
```

fun *r3-neg* :: *r3* ⇒ *r3* **where**
  r3-neg R0 = *R0*
| *r3-neg R1* = *R2*
| *r3-neg R2* = *R1*

definition *r3-sub* :: *r3* ⇒ *r3* ⇒ *r3* **where**
  r3-sub x y = *r3-add x* (*r3-neg y*)

fun *r3-double* :: *r3* ⇒ *r3* **where**
  r3-double R0 = *R0*
| *r3-double R1* = *R2*
| *r3-double R2* = *R1*

definition *rcoef-add* :: *rcoef* ⇒ *rcoef* ⇒ *rcoef* **where**
  rcoef-add p q = (*r3-add* (*fst p*) (*fst q*), *r3-add* (*snd p*) (*snd q*))

definition *rcoef-neg* :: *rcoef* ⇒ *rcoef* **where**
  rcoef-neg p = (*r3-neg* (*fst p*), *r3-neg* (*snd p*))

definition *rcoef-sub* :: *rcoef* ⇒ *rcoef* ⇒ *rcoef* **where**
  rcoef-sub p q = *rcoef-add p* (*rcoef-neg q*)

definition *rcoef-double* :: *rcoef* ⇒ *rcoef* **where**
  rcoef-double p = (*r3-double* (*fst p*), *r3-double* (*snd p*))

definition *rcoef-mul-sqrt2* :: *rcoef* ⇒ *rcoef* **where**
  rcoef-mul-sqrt2 p = (*r3-double* (*snd p*), *fst p*)

definition *rRx-step* :: *rvec3* ⇒ *rvec3* **where**
  rRx-step v = (*case v of* (*x*, *y*, *z*) ⇒
    ((*R0*, *R0*),
     rcoef-sub y (*rcoef-double* (*rcoef-mul-sqrt2 z*)),
     rcoef-add (*rcoef-double* (*rcoef-mul-sqrt2 y*)) *z*))

definition *rRx-inv-step* :: *rvec3* ⇒ *rvec3* **where**
  rRx-inv-step v = (*case v of* (*x*, *y*, *z*) ⇒
    ((*R0*, *R0*),
     rcoef-add y (*rcoef-double* (*rcoef-mul-sqrt2 z*)),
     rcoef-sub z (*rcoef-double* (*rcoef-mul-sqrt2 y*)))))

definition *rRz-step* :: *rvec3* ⇒ *rvec3* **where**
  rRz-step v = (*case v of* (*x*, *y*, *z*) ⇒
    (*rcoef-sub x* (*rcoef-double* (*rcoef-mul-sqrt2 y*)),
     rcoef-add (*rcoef-double* (*rcoef-mul-sqrt2 x*)) *y*,
     (*R0*, *R0*)))

definition *rRz-inv-step* :: *rvec3* ⇒ *rvec3* **where**
  rRz-inv-step v = (*case v of* (*x*, *y*, *z*) ⇒

```
    (rcoef-add x (rcoef-double (rcoef-mul-sqrt2 y)),
     rcoef-sub y (rcoef-double (rcoef-mul-sqrt2 x)),
     (R0, R0)))

definition rletter-step :: bool × gen2 ⇒ rvec3 ⇒ rvec3 where
  rletter-step p = (case p of
      (False, A) ⇒ rRx-step
    | (True,  A) ⇒ rRx-inv-step
    | (False, B) ⇒ rRz-step
    | (True,  B) ⇒ rRz-inv-step)

definition rword-step :: (bool × gen2) list ⇒ rvec3 ⇒ rvec3 where
  rword-step w v = foldr (λp f. rletter-step p ∘ f) w id v

definition re-x :: rvec3 where
  re-x = ((R1, R0), (R0, R0), (R0, R0))

definition re-y :: rvec3 where
  re-y = ((R0, R0), (R1, R0), (R0, R0))

definition rword-witness :: (bool × gen2) list ⇒ rvec3 where
  rword-witness w = (case snd (last w) of A ⇒ re-y | B ⇒ re-x)

definition rzero-vec :: rvec3 where
  rzero-vec = ((R0, R0), (R0, R0), (R0, R0))

definition rstate :: bool × gen2 ⇒ rvec3 ⇒ bool where
  rstate p v ⟷
    v ∈ (case p of
      (False, A) ⇒ {
          ((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) ⇒ {
          ((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) ⇒ {
          ((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) ⇒ {
          ((R0, R1), (R2, R0), (R0, R0)),
          ((R0, R2), (R1, R0), (R0, R0)),
          ((R1, R0), (R0, R1), (R0, R0)),
          ((R2, R0), (R0, R2), (R0, R0))})
```

lemma *rstate-single*:
rstate p (*rletter-step p* (*case snd p of A* ⇒ *re-y* | *B* ⇒ *re-x*))
⟨*proof*⟩

lemma *rstate-step-FA-FA*:
rstate (*False*, *A*) *v* ⟹ *rstate* (*False*, *A*) (*rletter-step* (*False*, *A*) *v*)
⟨*proof*⟩

lemma *rstate-step-FA-FB*:
rstate (*False*, *B*) *v* ⟹ *rstate* (*False*, *A*) (*rletter-step* (*False*, *A*) *v*)
⟨*proof*⟩

lemma *rstate-step-FA-TB*:
rstate (*True*, *B*) *v* ⟹ *rstate* (*False*, *A*) (*rletter-step* (*False*, *A*) *v*)
⟨*proof*⟩

lemma *rstate-step-TA-TA*:
rstate (*True*, *A*) *v* ⟹ *rstate* (*True*, *A*) (*rletter-step* (*True*, *A*) *v*)
⟨*proof*⟩

lemma *rstate-step-TA-FB*:
rstate (*False*, *B*) *v* ⟹ *rstate* (*True*, *A*) (*rletter-step* (*True*, *A*) *v*)
⟨*proof*⟩

lemma *rstate-step-TA-TB*:
rstate (*True*, *B*) *v* ⟹ *rstate* (*True*, *A*) (*rletter-step* (*True*, *A*) *v*)
⟨*proof*⟩

lemma *rstate-step-FB-FB*:
rstate (*False*, *B*) *v* ⟹ *rstate* (*False*, *B*) (*rletter-step* (*False*, *B*) *v*)
⟨*proof*⟩

lemma *rstate-step-FB-FA*:
rstate (*False*, *A*) *v* ⟹ *rstate* (*False*, *B*) (*rletter-step* (*False*, *B*) *v*)
⟨*proof*⟩

lemma *rstate-step-FB-TA*:
rstate (*True*, *A*) *v* ⟹ *rstate* (*False*, *B*) (*rletter-step* (*False*, *B*) *v*)
⟨*proof*⟩

lemma *rstate-step-TB-TB*:
rstate (*True*, *B*) *v* ⟹ *rstate* (*True*, *B*) (*rletter-step* (*True*, *B*) *v*)
⟨*proof*⟩

lemma *rstate-step-TB-FA*:
rstate (*False*, *A*) *v* ⟹ *rstate* (*True*, *B*) (*rletter-step* (*True*, *B*) *v*)
⟨*proof*⟩

lemma *rstate-step-TB-TA*:
rstate (*True*, *A*) *v* ⟹ *rstate* (*True*, *B*) (*rletter-step* (*True*, *B*) *v*)
⟨*proof*⟩

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** ¬ *canceling p q*
shows *rstate p* (*rletter-step p v*)
⟨*proof*⟩

lemma *rstate-nonzero*:
assumes *rstate p v*
shows *v* ≠ *rzero-vec*
⟨*proof*⟩

lemma *rword-step-Cons* [*simp*]:
rword-step (*p* # *w*) *v* = *rletter-step p* (*rword-step w v*)
⟨*proof*⟩

lemma *zword-step-Cons* [*simp*]:
zword-step (*p* # *w*) *v* = *zletter-step p* (*zword-step w v*)
⟨*proof*⟩

lemma *rword-witness-Cons-nonempty* [*simp*]:
assumes *w* ≠ []
shows *rword-witness* (*p* # *w*) = *rword-witness w*
⟨*proof*⟩

lemma *rword-step-witness-state*:
assumes *w* ∈ *carrier F2* **and** *w* ≠ []
shows *rstate* (*hd w*) (*rword-step w* (*rword-witness w*))
⟨*proof*⟩

definition *r3-of-int* :: *int* ⇒ *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* :: *zsqrt2* ⇒ *rcoef* **where**
rcoef-of-z p = (*r3-of-int* (*fst p*), *r3-of-int* (*snd p*))

definition *rvec3-of-z* :: *zvec3* ⇒ *rvec3* **where**
rvec3-of-z v = (*case v of* (*x*, *y*, *z*) ⇒ (*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*)

⟨*proof*⟩

lemma *r3-of-int-neg*:
r3-of-int (− *a*) = *r3-neg* (*r3-of-int a*)
⟨*proof*⟩

lemma *r3-of-int-sub*:
r3-of-int (*a* − *b*) = *r3-add* (*r3-of-int a*) (*r3-neg* (*r3-of-int b*))
⟨*proof*⟩

lemma *r3-of-int-double*:
r3-of-int (*2* ∗ *a*) = *r3-double* (*r3-of-int a*)
⟨*proof*⟩

lemma *r3-of-int-triple*:
r3-of-int (*3* ∗ *a*) = *R0*
⟨*proof*⟩

lemma *rcoef-of-z-simps* [*simp*]:
rcoef-of-z (*zsqrt2-add p q*) = *rcoef-add* (*rcoef-of-z p*) (*rcoef-of-z q*)
rcoef-of-z (*zsqrt2-neg p*) = *rcoef-neg* (*rcoef-of-z p*)
rcoef-of-z (*zsqrt2-sub p q*) = *rcoef-sub* (*rcoef-of-z p*) (*rcoef-of-z q*)
rcoef-of-z (*zsqrt2-scale 2 p*) = *rcoef-double* (*rcoef-of-z p*)
rcoef-of-z (*zsqrt2-scale 3 p*) = (*R0*, *R0*)
rcoef-of-z (*zsqrt2-mul-sqrt2 p*) = *rcoef-mul-sqrt2* (*rcoef-of-z p*)
⟨*proof*⟩

lemma *rvec3-of-z-basis* [*simp*]:
rvec3-of-z ze-x = *re-x*
rvec3-of-z ze-y = *re-y*
⟨*proof*⟩

lemma *rstep-of-z-simps* [*simp*]:
rvec3-of-z (*zRx-step v*) = *rRx-step* (*rvec3-of-z v*)
rvec3-of-z (*zRx-inv-step v*) = *rRx-inv-step* (*rvec3-of-z v*)
rvec3-of-z (*zRz-step v*) = *rRz-step* (*rvec3-of-z v*)
rvec3-of-z (*zRz-inv-step v*) = *rRz-inv-step* (*rvec3-of-z v*)
⟨*proof*⟩

lemma *rletter-step-of-z* [*simp*]:
rvec3-of-z (*zletter-step p v*) = *rletter-step p* (*rvec3-of-z v*)
⟨*proof*⟩

lemma *rword-step-of-z* [*simp*]:
rvec3-of-z (*zword-step w v*) = *rword-step w* (*rvec3-of-z v*)
⟨*proof*⟩

lemma *rword-witness-of-z* [*simp*]:
rvec3-of-z (*zword-witness w*) = *rword-witness w*

⟨*proof*⟩

lemma *r3-of-int-eq-R0-iff*:
r3-of-int n = *R0* ⟷ *3 dvd n*
⟨*proof*⟩

lemma *rcoef-of-z-zero-iff*:
rcoef-of-z p = (*R0*, *R0*) ⟷ *zsqrt2-div3 p*
⟨*proof*⟩

lemma *rvec3-of-z-zero-iff*:
rvec3-of-z v = *rzero-vec* ⟷ *zvec3-div3 v*
⟨*proof*⟩

lemma *zsqrt2-val-simps* [*simp*]:
zsqrt2-val (*zsqrt2-add p q*) = *zsqrt2-val p* + *zsqrt2-val q*
zsqrt2-val (*zsqrt2-neg p*) = − *zsqrt2-val p*
zsqrt2-val (*zsqrt2-sub p q*) = *zsqrt2-val p* − *zsqrt2-val q*
zsqrt2-val (*zsqrt2-scale n p*) = *of-int n* ∗ *zsqrt2-val p*
⟨*proof*⟩

lemma *zsqrt2-mul-sqrt2-val* [*simp*]:
zsqrt2-val (*zsqrt2-mul-sqrt2 p*) = *sqrt 2* ∗ *zsqrt2-val p*
⟨*proof*⟩

lemma *sqrt-prime-irrational-dev*:
fixes *p* :: *nat*
assumes *prime p*
shows *sqrt p* ∉ ℚ
⟨*proof*⟩

lemma *sqrt-2-not-rat-dev*: *sqrt 2* ∉ ℚ
⟨*proof*⟩

lemma *zsqrt2-val-eq-iff*:
zsqrt2-val p = *zsqrt2-val q* ⟷ *p* = *q*
⟨*proof*⟩

lemma *zvec3-val-eq-iff*:
zvec3-val u = *zvec3-val v* ⟷ *u* = *v*
⟨*proof*⟩

lemma *zvec3-scale-val* [*simp*]:
zvec3-val (*zvec3-scale n v*) = *scaleR* (*of-int n*) (*zvec3-val v*)
⟨*proof*⟩

lemma *sqrt8-eq-2-sqrt2*: *sqrt 8* = *2* ∗ *sqrt 2*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *zRx-step-val*:
zvec3-val (*zRx-step v*) = *scaleR 3* (*Rx* (*zvec3-val v*))
⟨*proof*⟩

lemma *zRx-inv-step-val*:
zvec3-val (*zRx-inv-step v*) = *scaleR 3* (*Rx-inv* (*zvec3-val v*))
⟨*proof*⟩

lemma *zRz-step-val*:
zvec3-val (*zRz-step v*) = *scaleR 3* (*Rz* (*zvec3-val v*))
⟨*proof*⟩

lemma *zRz-inv-step-val*:
zvec3-val (*zRz-inv-step v*) = *scaleR 3* (*Rz-inv* (*zvec3-val v*))
⟨*proof*⟩

lemma *zletter-step-val*:
zvec3-val (*zletter-step p v*) = *scaleR 3* (*letter-to-rot p* (*zvec3-val v*))
⟨*proof*⟩

lemma *letter-to-rot-linear*: *linear* (*letter-to-rot p*)
⟨*proof*⟩

lemma *zword-step-val*:
zvec3-val (*zword-step w v*) =
scaleR ((*3*::*real*) ^ *length w*) (*sigma w* (*zvec3-val v*))
⟨*proof*⟩

lemma *zvec3-scale-power3-div3*:
assumes *n* > *0*
shows *zvec3-div3* (*zvec3-scale* ((*3*::*int*) ^ *n*) *v*)
⟨*proof*⟩

lemma *zword-step-identity-div3*:
assumes *w* ≠ [] **and** *sigma w* = *id*
shows *zvec3-div3* (*zword-step w* (*zword-witness w*))
⟨*proof*⟩

lemma *zword-step-witness-not-div3-if-nonempty-F2*:
assumes *w* ∈ *carrier F2* **and** *w* ≠ []
shows ¬ *zvec3-div3* (*zword-step w* (*zword-witness w*))
⟨*proof*⟩

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

theorem *sigma-homomorphism*:
  assumes *w1* ∈ *carrier F2 w2* ∈ *carrier F2*
  shows *sigma* (*w1* $\otimes_{F2}$ *w2*) = *sigma w1* ∘ *sigma w2*
  ⟨*proof*⟩

theorem *sigma-nontrivial-word-not-id*:
  assumes *w* ∈ *carrier F2* **and** *w* ≠ []
  shows *sigma w* ≠ *id*
⟨*proof*⟩

lemma *sigma-injective-if-trivial-kernel*:
  assumes *kernel*: ⋀*w*. *w* ∈ *carrier F2* ⟹ *w* ≠ [] ⟹ *sigma w* ≠ *id*
  shows *inj-on sigma* (*carrier F2*)
⟨*proof*⟩

theorem *sigma-injective-on-F2*:
  inj-on sigma (*carrier F2*)
  ⟨*proof*⟩

theorem *sigma-image-in-SO3*:
  assumes *w* ∈ *carrier F2*
  shows *sigma w* ∈ *SO3*
  ⟨*proof*⟩

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* :: (*real^3* ⇒ *real^3*) ⇒ (*real^3*) *set* **where**
  fixed-point-set f = {*x* ∈ *sphere2*. *f x* = *x*}

definition *bad-set-D* :: (*real^3*) *set* **where**
  bad-set-D = (⋃ *w* ∈ *carrier F2* − {[]}. *fixed-point-set* (*sigma w*))

lemma *carrier-F2-countable*: *countable* (*carrier F2*)
⟨*proof*⟩

lemma *bad-set-D-index-countable*: *countable* (*carrier F2* − {[]})

⟨*proof*⟩

lemma *fixed-point-set-subset-sphere2*: *fixed-point-set f* ⊆ *sphere2*
⟨*proof*⟩

lemma *SO3-fixed-cross*:
assumes *f* ∈ *SO3* **and** *f x* = *x* **and** *f y* = *y*
shows *f* (*cross3 x y*) = *cross3 x y*
⟨*proof*⟩

lemma *SO3-fixed-cross-nonzero-imp-id*:
assumes *f* ∈ *SO3* **and** *f x* = *x* **and** *f y* = *y* **and** *cross3 x y* ≠ *0*
shows *f* = *id*
⟨*proof*⟩

lemma *nonidentity-SO3-fixed-points-cross-zero*:
assumes *f* ∈ *SO3* **and** *f* ≠ *id* **and** *x* ∈ *fixed-point-set f* **and** *y* ∈ *fixed-point-set f*
shows *cross3 x y* = *0*
⟨*proof*⟩

lemma *collinear-sphere2-two-points*:
assumes *a* ∈ *sphere2* **and** *y* ∈ *sphere2* **and** *cross3 a y* = *0*
shows *y* = *a* ∨ *y* = − *a*
⟨*proof*⟩

lemma *bad-set-D-countable-if-fixed-point-sets-countable*:
assumes ⋀*w*. *w* ∈ *carrier F2* − {[]} ⟹ *countable* (*fixed-point-set* (*sigma w*))
shows *countable bad-set-D*
⟨*proof*⟩

lemma *nonidentity-SO3-fixed-point-set-countable*:
assumes *f* ∈ *SO3* **and** *f* ≠ *id*
shows *countable* (*fixed-point-set f*)
⟨*proof*⟩

lemma *sigma-fixed-point-set-countable*:
assumes *w* ∈ *carrier F2* − {[]}
shows *countable* (*fixed-point-set* (*sigma w*))
⟨*proof*⟩

lemma *bad-set-D-countable*: *countable bad-set-D*
⟨*proof*⟩

lemma *bad-set-D-subset*: *bad-set-D* ⊆ *sphere2*
⟨*proof*⟩

lemma *sphere2-minus-bad-subset*: *sphere2* − *bad-set-D* ⊆ *sphere2*
⟨*proof*⟩

interpretation *sigma-sphere-action*:
group-action carrier F2 [] (λ*w1 w2*. *w1* $\otimes_{F2}$ *w2*) λ*w x*. *sigma w x sphere2*
⟨*proof*⟩

lemma *F2-conjugate-nontrivial*:
assumes *w* ∈ *carrier F2 v* ∈ *carrier F2 v* ≠ []
shows (*inv*$_{F2}$ *w* $\otimes_{F2}$ *v*) $\otimes_{F2}$ *w* ≠ []
⟨*proof*⟩

lemma *sigma-inverse-left*:
assumes *w* ∈ *carrier F2*
shows *sigma* (*inv*$_{F2}$ *w*) (*sigma w x*) = *x*
⟨*proof*⟩

lemma *sigma-conjugate-fixed-pullback*:
assumes *w* ∈ *carrier F2*
and *v* ∈ *carrier F2*
and *sigma v* (*sigma w x*) = *sigma w x*
shows *sigma* ((*inv*$_{F2}$ *w* $\otimes_{F2}$ *v*) $\otimes_{F2}$ *w*) *x* = *x*
⟨*proof*⟩

lemma *sigma-preimage-bad-set*:
assumes *w* ∈ *carrier F2*
and *x* ∈ *sphere2*
and *sigma w x* ∈ *bad-set-D*
shows *x* ∈ *bad-set-D*
⟨*proof*⟩

lemma *sigma-preserves-sphere2-minus-bad-set*:
assumes *w* ∈ *carrier F2*
and *x* ∈ *sphere2* − *bad-set-D*
shows *sigma w x* ∈ *sphere2* − *bad-set-D*
⟨*proof*⟩

interpretation *sigma-sphere-minus-bad-action*:
group-action carrier F2 [] (λ*w1 w2*. *w1* $\otimes_{F2}$ *w2*) λ*w x*. *sigma w x*
sphere2 − *bad-set-D*
⟨*proof*⟩

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* ∈ *sphere2* − *bad-set-D*

and $w \in$ *carrier F2*
and *sigma* $w\ x = x$
shows $w = []$
⟨*proof*⟩

lemma *sigma-sphere-minus-bad-action-free*:
sigma-sphere-minus-bad-action.free-on (*sphere2* − *bad-set-D*)
⟨*proof*⟩

lemma *sigma-orbit-eqI*:
assumes x: $x \in$ *sphere2* − *bad-set-D*
and y: $y \in$ *sphere2* − *bad-set-D*
and g: $g \in$ *carrier F2*
and *y-eq*: $y =$ *sigma* $g\ x$
shows *sigma-sphere-minus-bad-action.orbit* $y =$
sigma-sphere-minus-bad-action.orbit x
⟨*proof*⟩

lemma *sigma-orbit-eq-if-common-point*:
assumes x: $x \in$ *sphere2* − *bad-set-D*
and y: $y \in$ *sphere2* − *bad-set-D*
and z: $z \in$ *sigma-sphere-minus-bad-action.orbit* x
and z': $z \in$ *sigma-sphere-minus-bad-action.orbit* y
shows *sigma-sphere-minus-bad-action.orbit* $x =$
sigma-sphere-minus-bad-action.orbit y
⟨*proof*⟩

25 Transporting the free-group paradox to $S^2 \setminus D$

definition *hausdorff-rep* :: *real^3* ⇒ *real^3* **where**
hausdorff-rep x = (*SOME* y. $y \in$ *sigma-sphere-minus-bad-action.orbit* x)

definition *hausdorff-lift* :: ((*bool* × *gen2*) *list*) *set* ⇒ (*real^3*) *set* **where**
hausdorff-lift u =
{*sigma* g (*hausdorff-rep* x) | $g\ x$. $g \in u \wedge x \in$ *sphere2* − *bad-set-D*}

definition *hausdorff-sigma-image-set* ::
(*bool* × *gen2*) *list* ⇒ (*real^3*) *set* ⇒ (*real^3*) *set* **where**
hausdorff-sigma-image-set $g\ u$ = *sigma* g ‘ u

lemma *map2-hausdorff-sigma-image-set-eq*:
map2 sigma-sphere-minus-bad-action.image-set gs As =
map2 hausdorff-sigma-image-set gs As
⟨*proof*⟩

lemma *hausdorff-rep-in-orbit*:
assumes $x \in$ *sphere2* − *bad-set-D*
shows *hausdorff-rep* $x \in$ *sigma-sphere-minus-bad-action.orbit* x
⟨*proof*⟩

lemma *hausdorff-rep-in-X*:
assumes $x \in$ *sphere2* $-$ *bad-set-D*
shows *hausdorff-rep* $x \in$ *sphere2* $-$ *bad-set-D*
⟨*proof*⟩

lemma *hausdorff-rep-eq-if-orbit-eq*:
assumes *sigma-sphere-minus-bad-action.orbit* $x =$
sigma-sphere-minus-bad-action.orbit y
shows *hausdorff-rep* $x =$ *hausdorff-rep* y
⟨*proof*⟩

lemma *hausdorff-rep-idem*:
assumes x: $x \in$ *sphere2* $-$ *bad-set-D*
shows *hausdorff-rep* (*hausdorff-rep* x) = *hausdorff-rep* x
⟨*proof*⟩

lemma *hausdorff-lift-subset-X*:
assumes $u \subseteq$ *carrier F2*
shows *hausdorff-lift* $u \subseteq$ *sphere2* $-$ *bad-set-D*
⟨*proof*⟩

lemma *F2-act-image-set-subset-carrier*:
assumes $g \in$ *carrier F2* **and** $u \subseteq$ *carrier F2*
shows *F2-act.image-set* g $u \subseteq$ *carrier F2*
⟨*proof*⟩

lemma *sigma-free-same-point-eq*:
assumes x: $x \in$ *sphere2* $-$ *bad-set-D*
and p: $p \in$ *carrier F2*
and q: $q \in$ *carrier F2*
and *eq*: *sigma* p x = *sigma* q x
shows $p = q$
⟨*proof*⟩

lemma *hausdorff-lift-disjoint*:
assumes *disj*: $u \cap v = \{\}$
and u: $u \subseteq$ *carrier F2*
and v: $v \subseteq$ *carrier F2*
shows *hausdorff-lift* $u \cap$ *hausdorff-lift* $v = \{\}$
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *hausdorff-sigma-image-lift*:

assumes *g*: $g \in$ *carrier F2*
and *u*: $u \subseteq$ *carrier F2*
shows *hausdorff-sigma-image-set g* (*hausdorff-lift u*) =
hausdorff-lift (*F2-act.image-set g u*)
⟨*proof*⟩

lemma *hausdorff-translated-lift-pairwise-disjoint*:
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 *disj*: *pairwise-disjoint* (*map2 F2-act.image-set gs As*)
shows *pairwise-disjoint* (*map2 hausdorff-sigma-image-set gs* (*map hausdorff-lift As*))
⟨*proof*⟩

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* =
($\bigcup i$<*length As*. *F2-act.image-set* (*gs* ! *i*) (*As* ! *i*))
shows *sphere2* − *bad-set-D* =
($\bigcup i$<*length As*. *hausdorff-sigma-image-set* (*gs* ! *i*) (*hausdorff-lift* (*As* ! *i*)))
⟨*proof*⟩

26 The Hausdorff paradox

Define the rotation-action of F_2 on the sphere by evaluation: *sigma w* · *x* = *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 SO(3).

theorem *hausdorff-paradox-strong*:
$\exists\, P\ Q ::$ (*real*^3) *set list*. $\exists\, gP\ gQ ::$ ((*bool* × *gen2*) *list*) *list*.
length P = *length gP* ∧ *length Q* = *length gQ* ∧
set gP $\subseteq$ *carrier F2* ∧ *set gQ* $\subseteq$ *carrier F2* ∧
set (*map sigma gP*) $\subseteq$ *SO3* ∧ *set* (*map sigma gQ*) $\subseteq$ *SO3* ∧
pairwise-disjoint (*P* @ *Q*) ∧
pairwise-disjoint (*map2 hausdorff-sigma-image-set gP P*) ∧
pairwise-disjoint (*map2 hausdorff-sigma-image-set gQ Q*) ∧
($\forall\, i <$ *length P*. *P* ! *i* $\subseteq$ *sphere2* − *bad-set-D*) ∧
($\forall\, i <$ *length Q*. *Q* ! *i* $\subseteq$ *sphere2* − *bad-set-D*) ∧
(*sphere2* − *bad-set-D*) =
($\bigcup i$<*length P*. *P* ! *i*) ∪ ($\bigcup i$<*length Q*. *Q* ! *i*) ∧
(*sphere2* − *bad-set-D*) =
($\bigcup i$<*length P*. *hausdorff-sigma-image-set* (*gP* ! *i*) (*P* ! *i*)) ∧
(*sphere2* − *bad-set-D*) =
($\bigcup i$<*length Q*. *hausdorff-sigma-image-set* (*gQ* ! *i*) (*Q* ! *i*))

⟨*proof*⟩

theorem *hausdorff-paradoxical*:
  sigma-sphere-minus-bad-action.paradoxical (*sphere2* − *bad-set-D*)
⟨*proof*⟩

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* (λ*g A*. *g* ‘ *A*) *gP P*) ∧
      pairwise-disjoint (*map2* (λ*g 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*⟩

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*⟩

end

theory *Sphere-Decomposition*
  imports *Hausdorff-Paradox*
begin

Equidecomposability of $S^2 \setminus D$ and S^2.

definition *Rz-angle-mat* :: *real* ⇒ *real^3^3* **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* ⇒ *real^3* ⇒ *real^3* **where**
Rz-angle t v = *Rz-angle-mat t* ∗*v v*

definition *e1-3* :: *real^3* **where**
e1-3 = *vector* [*1*, *0*, *0*]

definition *e3-3* :: *real^3* **where**
e3-3 = *vector* [*0*, *0*, *1*]

lemma *Rz-angle-mat-orthogonal*: *orthogonal-matrix* (*Rz-angle-mat t*)
⟨*proof*⟩

lemma *det-Rz-angle-mat* [*simp*]: *matrix-det* (*Rz-angle-mat t*) = *1*
⟨*proof*⟩

lemma *Rz-angle-in-SO3*: *Rz-angle t* ∈ *SO3*
⟨*proof*⟩

lemma *Rz-angle-mat-add*:
Rz-angle-mat a ∗∗ *Rz-angle-mat b* = *Rz-angle-mat* (*a* + *b*)
⟨*proof*⟩

lemma *Rz-angle-compose*:
Rz-angle a ∘ *Rz-angle b* = *Rz-angle* (*a* + *b*)
⟨*proof*⟩

lemma *Rz-angle-funpow*:
(*Rz-angle t* ^^ *n*) = *Rz-angle* (*real n* ∗ *t*)
⟨*proof*⟩

lemma *Rz-angle-components* [*simp*]:
(*Rz-angle t x*) $ *1* = *cos t* ∗ *x* $ *1* − *sin t* ∗ *x* $ *2*
(*Rz-angle t x*) $ *2* = *sin t* ∗ *x* $ *1* + *cos t* ∗ *x* $ *2*
(*Rz-angle t x*) $ *3* = *x* $ *3*
⟨*proof*⟩

lemma *Rz-angle-eq-non-axis-sin-cos*:
assumes *Rz-angle a x* = *Rz-angle b x* **and** *x* $ *1* ≠ *0* ∨ *x* $ *2* ≠ *0*
shows *sin a* = *sin b* ∧ *cos a* = *cos b*
⟨*proof*⟩

lemma *Rz-angle-collision-angles-countable*:
assumes *nonaxis*: *x* $ *1* ≠ *0* ∨ *x* $ *2* ≠ *0*
shows *countable* {*t*. *Rz-angle t x* = *y*}
⟨*proof*⟩

lemma *Rz-angle-scaled-collision-angles-countable*:
assumes *k* > *0* **and** *nonaxis*: *x* $ *1* ≠ *0* ∨ *x* $ *2* ≠ *0*
shows *countable* {*t*. *Rz-angle* (*real k* ∗ *t*) *x* = *y*}

⟨*proof*⟩

lemma *e1-3-in-sphere2*: *e1-3* ∈ *sphere2*
⟨*proof*⟩

lemma *e1-3-nonaxis*: *e1-3* $ *1* ≠ *0* ∨ *e1-3* $ *2* ≠ *0*
⟨*proof*⟩

lemma *e3-3-in-sphere2*: *e3-3* ∈ *sphere2*
⟨*proof*⟩

lemma *z-axis-sphere2-cases*:
assumes *x* ∈ *sphere2* **and** *x* $ *1* = *0* **and** *x* $ *2* = *0*
shows *x* = *e3-3* ∨ *x* = − *e3-3*
⟨*proof*⟩

lemma *SO3-funpow*:
assumes *R* ∈ *SO3*
shows (*R* ^^ *n*) ∈ *SO3*
⟨*proof*⟩

lemma *SO3-inj*:
assumes *f* ∈ *SO3*
shows *inj f*
⟨*proof*⟩

lemma *SO3-surj*:
assumes *f* ∈ *SO3*
shows *surj f*
⟨*proof*⟩

lemma *SO3-linear*:
assumes *f* ∈ *SO3*
shows *linear f*
⟨*proof*⟩

lemma *SO3-inverse*:
assumes *f* ∈ *SO3*
shows *Hilbert-Choice.inv f* ∈ *SO3*
⟨*proof*⟩

lemma *conjugate-funpow*:
assumes *bijS*: *bij S*
shows ((*S* ∘ *R* ∘ *Hilbert-Choice.inv S*) ^^ *n*) =
S ∘ (*R* ^^ *n*) ∘ *Hilbert-Choice.inv S*
⟨*proof*⟩

lemma *conjugate-funpow-image*:
assumes *bijS*: *bij S*

shows $((S \circ R \circ \textit{Hilbert-Choice.inv}\ S) \mathbin{\frown\!\frown} n)\ `\ D =$
$S\ `\ ((R \mathbin{\frown\!\frown} n)\ `\ (\textit{Hilbert-Choice.inv}\ S\ `\ D))$
⟨*proof*⟩

lemma *exists-antipodal-axis-avoiding-countable*:
assumes *count-D*: *countable D*
shows $\exists\, a \in \textit{sphere2}.\ a \notin D \wedge -\ a \notin D$
⟨*proof*⟩

lemma *exists-Rz-angle-absorbing*:
assumes *count-D*: *countable D*
and *nonaxis*: $\bigwedge x.\ x \in D \Longrightarrow x\ \$\ 1 \neq 0 \vee x\ \$\ 2 \neq 0$
shows $\exists\, t.\ \forall\, n\ m.\ n \neq m \longrightarrow ((\textit{Rz-angle}\ t \mathbin{\frown\!\frown} n)\ `\ D) \cap ((\textit{Rz-angle}\ t \mathbin{\frown\!\frown} m)\ `\ D) = \{\}$
⟨*proof*⟩

lemma *exists-absorbing-rotation*:
$\exists\, R \in \textit{SO3}.\ \forall\, n\ m.\ n \neq m \longrightarrow ((R \mathbin{\frown\!\frown} n)\ `\ \textit{bad-set-D}) \cap ((R \mathbin{\frown\!\frown} m)\ `\ \textit{bad-set-D}) = \{\}$
⟨*proof*⟩

lemma *SO3-funpow-preserves-sphere2*:
assumes $R \in \textit{SO3}$ **and** $x \in \textit{sphere2}$
shows $(R \mathbin{\frown\!\frown} n)\ x \in \textit{sphere2}$
⟨*proof*⟩

lemma *funpow-Suc-image*:
$((R \mathbin{\frown\!\frown} \textit{Suc}\ n)\ `\ S) = R\ `\ ((R \mathbin{\frown\!\frown} n)\ `\ S)$
⟨*proof*⟩

lemma *absorbing-rotation-shift*:
assumes *disj*: $\forall\, n\ m.\ n \neq m \longrightarrow ((R \mathbin{\frown\!\frown} n)\ `\ D) \cap ((R \mathbin{\frown\!\frown} m)\ `\ D) = \{\}$
defines $E \equiv (\bigcup n.\ (R \mathbin{\frown\!\frown} n)\ `\ D)$
shows $R\ `\ E = E - D$
⟨*proof*⟩

lemma *sphere2-absorb-bad-set*:
$\exists\, P\ Q :: (\textit{real\^{}3})\ \textit{set list}.\ \exists\, gs :: ((\textit{real\^{}3}) \Rightarrow (\textit{real\^{}3}))\ \textit{list}.$
$\textit{length}\ P = \textit{length}\ Q \wedge \textit{length}\ P = \textit{length}\ gs \wedge$
$\textit{set}\ gs \subseteq \textit{SO3} \wedge$
$\textit{sphere2} = (\bigcup i{<}\textit{length}\ P.\ P\ !\ i) \wedge$
$(\textit{sphere2} - \textit{bad-set-D}) = (\bigcup i{<}\textit{length}\ Q.\ Q\ !\ i) \wedge$
$(\forall\, i < \textit{length}\ P.\ Q\ !\ i = (gs\ !\ i)\ `\ (P\ !\ i))$
⟨*proof*⟩

Combining the Hausdorff paradox with the absorption argument yields the paradoxical decomposition of the full sphere.

lemma *sphere-indexed-union-append-singleton*:
fixes $P :: 'a\ \textit{set list}$ **and** $S :: 'a\ \textit{set}$

shows $(\bigcup i{<}\mathit{length}\ (P\ @\ [S]).\ (P\ @\ [S])\ !\ i) =$
$(\bigcup i{<}\mathit{length}\ P.\ P\ !\ i) \cup S$
⟨*proof*⟩

lemma *sphere-indexed-image-union-append-singleton*:
fixes $P :: 'a\ set\ list$ **and** $G :: ('a \Rightarrow 'b)\ list$ **and** $S :: 'a\ set$
and $g :: 'a \Rightarrow 'b$
assumes $\mathit{length}\ P = \mathit{length}\ G$
shows $(\bigcup i{<}\mathit{length}\ (P\ @\ [S]).\ (G\ @\ [g])\ !\ i\ `\ ((P\ @\ [S])\ !\ i)) =$
$(\bigcup i{<}\mathit{length}\ P.\ G\ !\ i\ `\ (P\ !\ i)) \cup g\ `\ S$
⟨*proof*⟩

lemma *sphere2-absorb-cover*:
assumes *R-SO3*: $R \in \mathit{SO3}$
and *disj*: $\forall\, n\ m.\ n \neq m \longrightarrow ((R\ \hat{}\hat{}\ n)\ `\ \mathit{bad\text{-}set\text{-}D}) \cap ((R\ \hat{}\hat{}\ m)\ `\ \mathit{bad\text{-}set\text{-}D}) = \{\}$
and *len*: $\mathit{length}\ P = \mathit{length}\ g$
and *g-SO3*: $\mathit{set}\ g \subseteq \mathit{SO3}$
and *P-sub*: $\forall\, i < \mathit{length}\ P.\ P\ !\ i \subseteq \mathit{sphere2} - \mathit{bad\text{-}set\text{-}D}$
and *X-cover*: $\mathit{sphere2} - \mathit{bad\text{-}set\text{-}D} = (\bigcup i{<}\mathit{length}\ P.\ (g\ !\ i)\ `\ (P\ !\ i))$
shows $\exists\, P' :: (\mathit{real\hat{}3})\ \mathit{set\ list}.\ \exists\, g' :: ((\mathit{real\hat{}3}) \Rightarrow (\mathit{real\hat{}3}))\ \mathit{list}.$
$\mathit{length}\ P' = \mathit{length}\ g' \wedge$
$\mathit{set}\ g' \subseteq \mathit{SO3} \wedge$
$(\forall\, i < \mathit{length}\ P'.\ P'\ !\ i \subseteq \mathit{sphere2}) \wedge$
$\mathit{sphere2} = (\bigcup i{<}\mathit{length}\ P'.\ P'\ !\ i) \wedge$
$\mathit{sphere2} = (\bigcup i{<}\mathit{length}\ P'.\ (g'\ !\ i)\ `\ (P'\ !\ i))$
⟨*proof*⟩

theorem *sphere2-paradoxical-strong*:
$\exists\, P\ Q :: (\mathit{real\hat{}3})\ \mathit{set\ list}.\ \exists\, gP\ gQ :: ((\mathit{real\hat{}3}) \Rightarrow (\mathit{real\hat{}3}))\ \mathit{list}.$
$\mathit{length}\ P = \mathit{length}\ gP \wedge \mathit{length}\ Q = \mathit{length}\ gQ \wedge$
$\mathit{set}\ gP \subseteq \mathit{SO3} \wedge \mathit{set}\ gQ \subseteq \mathit{SO3} \wedge$
$\mathit{pairwise\text{-}disjoint}\ (P\ @\ Q) \wedge$
$\mathit{pairwise\text{-}disjoint}\ (\mathit{map2}\ (\lambda g\ A.\ g\ `\ A)\ gP\ P) \wedge$
$\mathit{pairwise\text{-}disjoint}\ (\mathit{map2}\ (\lambda g\ A.\ g\ `\ A)\ gQ\ Q) \wedge$
$(\forall\, i < \mathit{length}\ P.\ P\ !\ i \subseteq \mathit{sphere2}) \wedge$
$(\forall\, i < \mathit{length}\ Q.\ Q\ !\ i \subseteq \mathit{sphere2}) \wedge$
$\mathit{sphere2} = (\bigcup i{<}\mathit{length}\ P.\ P\ !\ i) \cup (\bigcup i{<}\mathit{length}\ Q.\ Q\ !\ i) \wedge$
$\mathit{sphere2} = (\bigcup i{<}\mathit{length}\ P.\ (gP\ !\ i)\ `\ (P\ !\ i)) \wedge$
$\mathit{sphere2} = (\bigcup i{<}\mathit{length}\ Q.\ (gQ\ !\ i)\ `\ (Q\ !\ i))$
⟨*proof*⟩

theorem *sphere2-paradoxical*:
$\exists\, P\ Q :: (\mathit{real\hat{}3})\ \mathit{set\ list}.\ \exists\, gP\ gQ :: ((\mathit{real\hat{}3}) \Rightarrow (\mathit{real\hat{}3}))\ \mathit{list}.$
$\mathit{length}\ P = \mathit{length}\ gP \wedge \mathit{length}\ Q = \mathit{length}\ gQ \wedge$
$\mathit{set}\ gP \subseteq \mathit{SO3} \wedge \mathit{set}\ gQ \subseteq \mathit{SO3} \wedge$
$(\forall\, i < \mathit{length}\ P.\ P\ !\ i \subseteq \mathit{sphere2}) \wedge$
$(\forall\, i < \mathit{length}\ Q.\ Q\ !\ i \subseteq \mathit{sphere2}) \wedge$
$\mathit{sphere2} = (\bigcup i{<}\mathit{length}\ P.\ P\ !\ i) \wedge$

$sphere2 = (\bigcup i{<}length\ Q.\ Q\ !\ i) \wedge$
$sphere2 = (\bigcup i{<}length\ P.\ (gP\ !\ i)\ `\ (P\ !\ i)) \wedge$
$sphere2 = (\bigcup i{<}length\ Q.\ (gQ\ !\ i)\ `\ (Q\ !\ i))$
⟨*proof*⟩

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* :: $(real\hat{}3)\ set \Rightarrow (real\hat{}3)\ set$ **where**
$cone\text{-}of\ S = \{t *_R x \mid t\ x.\ 0 < t \wedge t \leq 1 \wedge x \in S\}$

lemma *cone-of-in-ball3*:
assumes $S \subseteq sphere2$
shows $cone\text{-}of\ S \subseteq ball3$
⟨*proof*⟩

lemma *cone-of-in-ball3-minus-origin*:
assumes $S \subseteq sphere2$
shows $cone\text{-}of\ S \subseteq ball3 - \{0\}$
⟨*proof*⟩

lemma *ball3-minus-origin-eq-cone-sphere2*:
$ball3 - \{0{::}real\hat{}3\} = cone\text{-}of\ sphere2$
⟨*proof*⟩

lemma *cone-of-UN*:
$cone\text{-}of\ (\bigcup i{\in}I.\ S\ i) = (\bigcup i{\in}I.\ cone\text{-}of\ (S\ i))$
⟨*proof*⟩

lemma *SO3-image-cone-of*:
assumes $g \in SO3$
shows $g\ `\ cone\text{-}of\ S = cone\text{-}of\ (g\ `\ S)$
⟨*proof*⟩

lemma *cone-of-disjoint*:
assumes *S-sub*: $S \subseteq sphere2$
and *T-sub*: $T \subseteq sphere2$
and *disj*: $S \cap T = \{\}$
shows $cone\text{-}of\ S \cap cone\text{-}of\ T = \{\}$
⟨*proof*⟩

lemma *cone-of-pairwise-disjoint*:
assumes *disj*: *pairwise-disjoint P*

and *sub*: $\forall\, i < \mathit{length}\ P.\ P\ !\ i \subseteq \mathit{sphere2}$
shows *pairwise-disjoint* (*map cone-of P*)
⟨*proof*⟩

lemma *SO3-cone-images-pairwise-disjoint*:
assumes *len*: *length P* = *length g*
and *g-SO3*: *set g* ⊆ *SO3*
and *P-sub*: $\forall\, i < \mathit{length}\ P.\ P\ !\ i \subseteq \mathit{sphere2}$
and *disj*: *pairwise-disjoint* (*map2* (λ*h A*. *h* ‘ *A*) *g P*)
shows *pairwise-disjoint* (*map2* (λ*h A*. *h* ‘ *A*) *g* (*map cone-of P*))
⟨*proof*⟩

lemma *zero-in-ball3* [*simp*]: (*0*::*real^3*) ∈ *ball3*
⟨*proof*⟩

lemma *indexed-image-union-append-singleton*:
fixes *P* :: *'a set list* **and** *G* :: (*'a* ⇒ *'b*) *list* **and** *S* :: *'a set*
and *g* :: *'a* ⇒ *'b*
assumes *length P* = *length G*
shows $(\bigcup i{<}\mathit{length}\ (P\ @\ [S]).\ (G\ @\ [g])\ !\ i\ \text{`}\ ((P\ @\ [S])\ !\ i)) =$
$(\bigcup i{<}\mathit{length}\ P.\ G\ !\ i\ \text{`}\ (P\ !\ i)) \cup g\ \text{`}\ S$
⟨*proof*⟩

lemma *indexed-union-append-singleton*:
fixes *P* :: *'a set list* **and** *S* :: *'a set*
shows $(\bigcup i{<}\mathit{length}\ (P\ @\ [S]).\ (P\ @\ [S])\ !\ i) =$
$(\bigcup i{<}\mathit{length}\ P.\ P\ !\ i) \cup S$
⟨*proof*⟩

The punctured ball is paradoxical via radial cones over the sphere pieces.

theorem *ball3-minus-origin-paradoxical*:
$\exists\, P\ Q :: (\mathit{real^3})\ \mathit{set\ list}.\ \exists\, gP\ gQ :: ((\mathit{real^3}) \Rightarrow (\mathit{real^3}))\ \mathit{list}.$
$\mathit{length}\ P = \mathit{length}\ gP \wedge \mathit{length}\ Q = \mathit{length}\ gQ \wedge$
$\mathit{set}\ gP \subseteq \mathit{SO3} \wedge \mathit{set}\ gQ \subseteq \mathit{SO3} \wedge$
$(\forall\, i < \mathit{length}\ P.\ P\ !\ i \subseteq \mathit{ball3} - \{0\}) \wedge$
$(\forall\, i < \mathit{length}\ Q.\ Q\ !\ i \subseteq \mathit{ball3} - \{0\}) \wedge$
$\mathit{ball3} - \{0\} = (\bigcup i{<}\mathit{length}\ P.\ P\ !\ i) \wedge$
$\mathit{ball3} - \{0\} = (\bigcup i{<}\mathit{length}\ Q.\ Q\ !\ i) \wedge$
$\mathit{ball3} - \{0\} = (\bigcup i{<}\mathit{length}\ P.\ (gP\ !\ i)\ \text{`}\ (P\ !\ i)) \wedge$
$\mathit{ball3} - \{0\} = (\bigcup i{<}\mathit{length}\ Q.\ (gQ\ !\ i)\ \text{`}\ (Q\ !\ i))$
⟨*proof*⟩

theorem *ball3-minus-origin-paradoxical-strong*:
$\exists\, P\ Q :: (\mathit{real^3})\ \mathit{set\ list}.\ \exists\, gP\ gQ :: ((\mathit{real^3}) \Rightarrow (\mathit{real^3}))\ \mathit{list}.$
$\mathit{length}\ P = \mathit{length}\ gP \wedge \mathit{length}\ Q = \mathit{length}\ gQ \wedge$
$\mathit{set}\ gP \subseteq \mathit{SO3} \wedge \mathit{set}\ gQ \subseteq \mathit{SO3} \wedge$
$\mathit{pairwise\text{-}disjoint}\ (P\ @\ Q) \wedge$
$\mathit{pairwise\text{-}disjoint}\ (\mathit{map2}\ (\lambda g\ A.\ g\ \text{`}\ A)\ gP\ P) \wedge$
$\mathit{pairwise\text{-}disjoint}\ (\mathit{map2}\ (\lambda g\ A.\ g\ \text{`}\ A)\ gQ\ Q) \wedge$

$(\forall\, i < \mathit{length}\ P.\ P\ !\ i \subseteq \mathit{ball3} - \{0\}) \wedge$
$(\forall\, i < \mathit{length}\ Q.\ Q\ !\ i \subseteq \mathit{ball3} - \{0\}) \wedge$
$\mathit{ball3} - \{0\} = (\bigcup i{<}\mathit{length}\ P.\ P\ !\ i) \cup (\bigcup i{<}\mathit{length}\ Q.\ Q\ !\ i) \wedge$
$\mathit{ball3} - \{0\} = (\bigcup i{<}\mathit{length}\ P.\ (gP\ !\ i)\ `\ (P\ !\ i)) \wedge$
$\mathit{ball3} - \{0\} = (\bigcup i{<}\mathit{length}\ Q.\ (gQ\ !\ i)\ `\ (Q\ !\ i))$
⟨*proof*⟩

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*:
$\exists\, P\ Q :: (\mathit{real\hat{}3})\ \mathit{set}\ \mathit{list}.\ \exists\, gP\ gQ :: ((\mathit{real\hat{}3}) \Rightarrow (\mathit{real\hat{}3}))\ \mathit{list}.$
$\mathit{length}\ P = \mathit{length}\ gP \wedge \mathit{length}\ Q = \mathit{length}\ gQ \wedge$
$\mathit{set}\ gP \subseteq \mathit{SO3} \wedge \mathit{set}\ gQ \subseteq \mathit{SO3} \wedge$
$(\forall\, i < \mathit{length}\ P.\ P\ !\ i \subseteq \mathit{ball3}) \wedge$
$(\forall\, i < \mathit{length}\ Q.\ Q\ !\ i \subseteq \mathit{ball3}) \wedge$
$\mathit{ball3} = (\bigcup i{<}\mathit{length}\ P.\ P\ !\ i) \wedge$
$\mathit{ball3} = (\bigcup i{<}\mathit{length}\ Q.\ Q\ !\ i) \wedge$
$\mathit{ball3} = (\bigcup i{<}\mathit{length}\ P.\ (gP\ !\ i)\ `\ (P\ !\ i)) \wedge$
$\mathit{ball3} = (\bigcup i{<}\mathit{length}\ Q.\ (gQ\ !\ i)\ `\ (Q\ !\ i))$
⟨*proof*⟩

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-copy-left = *ball3*

definition *ball3-copy-right* :: (*real^3*) *set* **where**
ball3-copy-right = (+) *shift3* ` *ball3*

lemma *norm-shift3* [*simp*]: *norm shift3* = *3*
⟨*proof*⟩

lemma *ball3-copies-disjoint*:
ball3-copy-left ∩ *ball3-copy-right* = {}
⟨*proof*⟩

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* ⇒ *real^3*) ⇒ *bool* **where**

isometry f $\longleftrightarrow$ $(\forall\, x\ y.\ dist\ (f\ x)\ (f\ y) = dist\ x\ y)$

lemma *id-isometry* [*simp*]: *isometry id*
⟨*proof*⟩

lemma *translation-isometry* [*simp*]: *isometry* $((+)\ a)$
⟨*proof*⟩

lemma *SO3-isometry*:
assumes $g \in SO3$
shows *isometry g*
⟨*proof*⟩

lemma *isometry-compose*:
assumes *isometry f isometry g*
shows *isometry* $(f \circ g)$
⟨*proof*⟩

lemma *translation-compose-SO3-isometry* [*simp*]:
assumes $g \in SO3$
shows *isometry* $((+)\ a \circ g)$
⟨*proof*⟩

lemma *offcenter-rotation-isometry*:
assumes $R \in SO3$
shows *isometry* $(\lambda x.\ c + R\ (x - c))$
⟨*proof*⟩

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 $U\ (T\ x) = x$
⟨*proof*⟩

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 $T\ (U\ x) = x$
⟨*proof*⟩

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 $(T\ \widehat{\ }\widehat{\ }\ n)\ 0 = c + (R\ \widehat{\ }\widehat{\ }\ n)\ (-\ c)$
⟨*proof*⟩

lemma *sigma-replicate-b*:
sigma (*replicate n* (*False*, *B*)) = (*Rz* ^^ *n*)
⟨*proof*⟩

lemma *canceled-replicate-b*: *canceled* (*replicate n* (*False*, *B*))
⟨*proof*⟩

lemma *replicate-b-in-F2*: *replicate n* (*False*, *B*) ∈ *carrier F2*
⟨*proof*⟩

theorem *ball3-paradoxical-strong*:
∃ *P Q* :: (*real^3*) *set list*. ∃ *gP gQ* :: ((*real^3*) ⇒ (*real^3*)) *list*.
length P = *length gP* ∧ *length Q* = *length gQ* ∧
(∀ *g* ∈ *set gP*. *isometry g*) ∧ (∀ *g* ∈ *set gQ*. *isometry g*) ∧
pairwise-disjoint (*P* @ *Q*) ∧
pairwise-disjoint (*map2* (λ*g A*. *g* ‘ *A*) *gP P*) ∧
pairwise-disjoint (*map2* (λ*g A*. *g* ‘ *A*) *gQ Q*) ∧
(∀ *i* < *length P*. *P* ! *i* ⊆ *ball3*) ∧
(∀ *i* < *length Q*. *Q* ! *i* ⊆ *ball3*) ∧
ball3 = (⋃ *i*<*length P*. *P* ! *i*) ∪ (⋃ *i*<*length Q*. *Q* ! *i*) ∧
ball3 = (⋃ *i*<*length P*. (*gP* ! *i*) ‘ (*P* ! *i*)) ∧
ball3 = (⋃ *i*<*length Q*. (*gQ* ! *i*) ‘ (*Q* ! *i*))
⟨*proof*⟩

An auxiliary finite-cover theorem for the closed unit ball.

theorem *banach-tarski-finite-cover*:
∃ *P* :: (*real^3*) *set list*. ∃ *gs* :: ((*real^3*) ⇒ (*real^3*)) *list*.
length P = *length gs* ∧
(∀ *g* ∈ *set gs*. *isometry g*) ∧
(∀ *i* < *length P*. *P* ! *i* ⊆ *ball3*) ∧
ball3 = (⋃ *i*<*length P*. *P* ! *i*) ∧
(*ball3-copy-left* ∪ *ball3-copy-right*) =
(⋃ *i*<*length P*. (*gs* ! *i*) ‘ (*P* ! *i*)) ∧
ball3-copy-left ∩ *ball3-copy-right* = {}
⟨*proof*⟩

The Banach–Tarski theorem for the closed unit ball, as a partition theorem.

theorem *banach-tarski*:
∃ *P* :: (*real^3*) *set list*. ∃ *gs* :: ((*real^3*) ⇒ (*real^3*)) *list*.
length P = *length gs* ∧
(∀ *g* ∈ *set gs*. *isometry g*) ∧
pairwise-disjoint P ∧
pairwise-disjoint (*map2* (λ*g A*. *g* ‘ *A*) *gs P*) ∧
(∀ *i* < *length P*. *P* ! *i* ⊆ *ball3*) ∧
ball3 = (⋃ *i*<*length P*. *P* ! *i*) ∧
(*ball3-copy-left* ∪ *ball3-copy-right*) =
(⋃ *i*<*length P*. (*gs* ! *i*) ‘ (*P* ! *i*)) ∧
ball3-copy-left ∩ *ball3-copy-right* = {}

⟨*proof*⟩

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. Tomkowicz 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.