Orbit-Stabiliser Theorem with Application to Rotational Symmetries

Jonas Rädle

March 17, 2025

Abstract

The Orbit-Stabiliser theorem is a simple result in the algebra of groups that factors the order of a group into the sizes of its orbits and stabilisers.

We formalize the notion of a group action and the related concepts of orbits and stabilisers. This allows us to prove the orbit-stabiliser theorem.

In the second part of this work, we formalize the tetrahedral group and use the orbit-stabiliser theorem to prove that there are twelve (orientation-preserving) rotations of the tetrahedron.

Contents

1 Orbit-Stabiliser Theorem

In this Theory we will prove the orbit-stabiliser theorem, a basic result in the algebra of groups.

theory *Orbit-Stabiliser*
  imports
    HOL−Algebra.Left-Coset

begin

1.1 Imports

/HOL/Algebra/Group.thy is used for the definitions of groups and subgroups

Left_Coset.thy is a copy of /HOL/Algebra/Coset.thy that includes additional theorems about left cosets.
The version of Coset.thy in the Isabelle library is missing some theorems about left cosets that are available for right cosets, so these had to be added by simply replacing the definitions of right cosets with those of left cosets.
Coset.thy is used for definitions of group order, quotient groups (operator LMod), and Lagranges theorem.

/HOL/Fun.thy is used for function composition and the identity function.

1.2 Group Actions

We begin by augmenting the existing definition of a group with a group action.
The group action was defined according to [4].

locale *orbit-stabiliser* = *group* +
  fixes *action* :: $'a \Rightarrow 'b \Rightarrow 'b$ (**infixl** ‹⊙› *51*)
  assumes *id-act* [*simp*]: $\mathbf{1} \odot x = x$
    and *compat-act*:
    $g \in carrier\ G \wedge h \in carrier\ G \longrightarrow g \odot (h \odot x) = (g \otimes h) \odot x$

1.3 Orbit and stabiliser

Next, we define orbit and stabiliser, according to the same Wikipedia article.

context *orbit-stabiliser*
begin

definition *orbit* :: $'b \Rightarrow 'b\ set$ **where**

orbit $x = \{y.\ (\exists\ g \in carrier\ G.\ y = g \odot x)\}$

definition *stabiliser* :: $'b \Rightarrow 'a\ set$
where *stabiliser* $x = \{g \in carrier\ G.\ g \odot x = x\}$

1.4 Stabiliser Theorems

We begin our proofs by showing that the stabiliser forms a subgroup.
This proof follows the template from [2].

theorem *stabiliser-subgroup*: *subgroup* (*stabiliser* x) G
proof(*rule subgroupI*)
show *stabiliser* $x \subseteq carrier\ G$ **using** *stabiliser-def* **by** *auto*
next
fix x
from *id-act* **have** $\mathbf{1} \odot x = x$ **by** *simp*
then have $\mathbf{1} \in stabiliser\ x$ **using** *stabiliser-def* **by** *auto*
then show *stabiliser* $x \neq \{\}$ **by** *auto*
next
fix $g\ x$
assume *gStab*:$g \in stabiliser\ x$
then have *g-car*:$g \in carrier\ G$ **using** *stabiliser-def* **by** *simp*
then have *invg-car*:$inv\ g \in carrier\ G$ **using** *inv-closed* **by** *simp*
have $g \odot x = x$ **using** *stabiliser-def gStab* **by** *simp*
then have $inv\ g \odot (g \odot x) = inv\ g \odot x$ **by** *simp*
then have $(inv\ g \otimes g) \odot x = inv\ g \odot x$ **using** *compat-act g-car invg-car* **by** *simp*
then have $x = (inv\ g) \odot x$ **using** *g-car l-inv* **by** *simp*
then show $inv\ g \in stabiliser\ x$ **using** *invg-car stabiliser-def* **by** *simp*
next
fix $g\ h\ x$
assume *g-stab*: $g \in stabiliser\ x$ **and** *h-stab*: $h \in stabiliser\ x$
then have *g-car*: $g \in carrier\ G$ **and** *h-car*: $h \in carrier\ G$ **using** *stabiliser-def* **by** *auto*
then have $g \odot x = x\ h \odot x = x$
using *stabiliser-def g-stab h-stab* **by** *auto*
then have $g \odot (h \odot x) = x$ **by** *simp*
then have $(g \otimes h) \odot x = x$ **using** *compat-act g-car h-car* **by** *simp*
then show $(g \otimes h) \in stabiliser\ x$
using *g-stab h-stab stabiliser-def* **by** *auto*
qed

As an intermediate step we formulate a lemma about the relationship between the group action and the stabiliser.
This proof follows the template from [3].

corollary *stabiliser-subgroup-corollary*:
assumes *g-car*: $g \in carrier\ G$ **and**
h-car: $h \in carrier\ G$
shows $(g \odot x) = (h \odot x) \longleftrightarrow ((inv\ g) \otimes h) \in stabiliser\ x$
proof
from *g-car* **have** *invg-car*: $(inv\ g) \in carrier\ G$ **by** *auto*
show $(g \odot x) = (h \odot x) \Longrightarrow inv\ g \otimes h \in stabiliser\ x$
proof −
assume *gh*: $(g \odot x) = (h \odot x)$
have $((inv\ g) \otimes h) \odot x = (inv\ g) \odot (h \odot x)$ **using** *assms compat-act* **by** *simp*
moreover have $(inv\ g) \odot (h \odot x) = (inv\ g) \odot (g \odot x)$ **using** *gh* **by** *simp*
moreover have $(inv\ g) \odot (g \odot x) = ((inv\ g) \otimes g) \odot x$ **using** *invg-car g-car compat-act* **by** *simp*
moreover have $((inv\ g) \otimes g) \odot x = x$ **using** *g-car* **by** *simp*
ultimately have $((inv\ g) \otimes h) \odot x = x$ **by** *simp*
then show *?thesis* **using** *stabiliser-def assms* **by** *simp*
qed

show $inv\ g \otimes h \in stabiliser\ x \Longrightarrow g \odot x = h \odot x$
proof −
assume *gh-stab*: $inv\ g \otimes h \in stabiliser\ x$
with *stabiliser-def* **have** $x = ((inv\ g) \otimes h) \odot x$ **by** *simp*
then have $\mathbf{1} \odot x = ((inv\ g) \otimes h) \odot x$ **by** *simp*
then have $((inv\ g) \otimes g) \odot x = ((inv\ g) \otimes h) \odot x$ **using** *invg-car g-car* **by** *simp*
then have $x = (inv\ g) \odot (h \odot x)$ **using** *compat-act g-car h-car* **by** *simp*
then have $g \odot x = (g \otimes (inv\ g)) \odot (h \odot x)$ **using** *compat-act g-car invg-car* **by** *metis*
then have $g \odot x = h \odot x$ **using** *compat-act g-car id-act invg-car r-inv* **by** *simp*
then show *?thesis* **by** *simp*
qed
qed

Using the previous lemma and our proof that the stabiliser forms a subgroup, we can now show that the elements of the orbit map to left cosets of the stabiliser.

This will later form the basis of showing a bijection between the orbit and those cosets.

lemma *stabiliser-cosets-equivalent*:
assumes *g-car*: $g \in carrier\ G$ **and**
h-car: $h \in carrier\ G$

shows $(g \odot x) = (h \odot x) \longleftrightarrow (g <\# \textit{ stabiliser } x) = (h <\# \textit{ stabiliser } x)$
proof
show $g \odot x = h \odot x \Longrightarrow g <\# \textit{ stabiliser } x = h <\# \textit{ stabiliser } x$
proof –
assume $g \odot x = h \odot x$
then have *stab-elem*: $((inv\ g) \otimes h) \in \textit{stabiliser } x$
using *assms stabiliser-subgroup-corollary* **by** *simp*
with *subgroup.lcos-module-rev*[*OF stabiliser-subgroup*] **have** $h \in g <\# (\textit{stabiliser } x)$
using *assms is-group* **by** *simp*
with *l-repr-independence* **have** $g <\# (\textit{stabiliser } x) = h <\# (\textit{stabiliser } x)$
using *assms stab-elem stabiliser-subgroup* **by** *auto*
then show *?thesis* **by** *simp*
qed
show $g <\# \textit{ stabiliser } x = h <\# \textit{ stabiliser } x \Longrightarrow g \odot x = h \odot x$
proof –
assume $g <\# \textit{ stabiliser } x = h <\# \textit{ stabiliser } x$
with *subgroup.lcos-module-rev*[*OF stabiliser-subgroup*] **have** $h \in g <\# (\textit{stabiliser } x)$
using *assms is-group l-inv stabiliser-subgroup subgroup-def* **by** *metis*
with *subgroup.lcos-module-imp*[*OF stabiliser-subgroup*] **have** $((inv\ g) \otimes h) \in \textit{stabiliser } x$
using *assms is-group* **by** *blast*
with *stabiliser-subgroup-corollary* **have** $g \odot x = h \odot x$ **using** *assms* **by** *simp*
then show *?thesis* **by** *simp*
qed
qed

1.5 Picking representatives from cosets

Before we can prove the bijection, we need a few lemmas about representatives from sets.

First we define rep to be an arbitrary element from a left coset of the stabiliser.

definition *rep* :: *'a set* ⇒ *'a* **where**
$(H \in \textit{carrier } (G\ \textit{LMod } (\textit{stabiliser } x))) \Longrightarrow \textit{rep } H = (\textit{SOME } y.\ y \in H)$

The next lemma shows that the representative is always an element of its coset.

lemma *quotient-rep-ex* : $H \in (\textit{carrier } (G\ \textit{LMod } (\textit{stabiliser } x))) \Longrightarrow \textit{rep } H \in H$

```
proof –
  fix H
  assume H:H ∈ carrier (G LMod stabiliser x)
  then obtain g where g ∈ carrier G H = g <# (stabiliser x)
    unfolding LFactGroup-def LCOSETS-def by auto
   then have (SOME x. x ∈ H) ∈ H using lcos-self stabiliser-subgroup
someI-ex by fast
  then show rep H ∈ H using H rep-def by auto
qed
```

The final lemma about representatives shows that it does not matter which element of the coset is picked, i.e. all representatives are equivalent.

```
lemma rep-equivalent:
  assumes H:H ∈ carrier (G LMod stabiliser x) and
    gH:g ∈ H
  shows H = g <# (stabiliser x)
proof –
  fix h
 from H obtain h where hG:h ∈ carrier G and H2:H = h <# (stabiliser
x)
    unfolding LFactGroup-def LCOSETS-def by auto
  with H gH have gh:g ∈ h <# (stabiliser x) by simp
  from l-repr-independence have h <# stabiliser x = g <# stabiliser x
    using hG gh stabiliser-subgroup by simp
  with H2 have H = g <# (stabiliser x) by simp
  then show ?thesis by simp
qed
```

1.6 Orbit-Stabiliser Theorem

We can now establish the bijection between orbit(x) and the quotient group G/(stabiliser(x))

The idea for this bijection is from [1]

```
theorem orbit-stabiliser-bij:
  bij-betw (λH. rep H ⊙ x) (carrier (G LMod (stabiliser x))) (orbit x)
proof (rule bij-betw-imageI)

  show inj-on (λH. rep H ⊙ x) (carrier (G LMod stabiliser x))
  proof(rule inj-onI)
    fix H H′
    assume H:H ∈ carrier (G LMod (stabiliser x))
    assume H′:H′ ∈ carrier (G LMod (stabiliser x))
    obtain h h′ where  h:h = rep H and h′: h′ = rep H′ by simp
```

assume *act-equal*: (*rep H*) ⊙ *x* = (*rep H′*) ⊙ *x*
from *H h quotient-rep-ex* **have** *hH*: *h* ∈ *H* **by** *simp*
from *H′ h′ quotient-rep-ex* **have** *hH′*: *h′* ∈ *H′* **by** *simp*
from *subgroup.lcosets-carrier*[*OF stabiliser-subgroup is-group*] *H* **have** *H* ⊆ *carrier G*
unfolding *LFactGroup-def* **by** *simp*
then have *hG*: *h* ∈ *carrier G* **using** *hH* **by** *auto*
from *subgroup.lcosets-carrier*[*OF stabiliser-subgroup is-group*] *H′* **have** *H′* ⊆ *carrier G*
unfolding *LFactGroup-def* **by** *simp*
then have *h′G*: *h′* ∈ *carrier G* **using** *hH′* **by** *auto*

have *hh′-equiv*:*h* <# (*stabiliser x*) = *h′* <# (*stabiliser x*)
using *hG h′G h h′ act-equal stabiliser-cosets-equivalent* **by** *simp*

from *hh′-equiv* **have** *H2*:*H* = *h* <# (*stabiliser x*)
using *H hH rep-equivalent* **by** *blast*
moreover from *hh′-equiv* **have** *H3*:*H′* = *h* <# (*stabiliser x*)
using *H′ hH′ rep-equivalent* **by** *blast*
then show *H* = *H′* **using** *H2 H3* **by** *simp*
qed
next
show (λ*H*. *rep H* ⊙ *x*) ‘ *carrier* (*G LMod stabiliser x*) = *orbit x*
proof(*auto*)
show ⋀*H*. *H* ∈ *carrier* (*G LMod stabiliser x*) ⟹ *rep H* ⊙ *x* ∈ *orbit x*
proof −
fix *H*
assume *H*:*H* ∈ *carrier* (*G LMod* (*stabiliser x*))
obtain *h* **where** *h*:*h* = *rep H* **by** *simp*
from *H h quotient-rep-ex* **have** *hH*: *h* ∈ *H* **by** *simp*
have *stab-sub*: (*stabiliser x*) ⊆ *carrier G* **using** *stabiliser-def* **by** *auto*
from *subgroup.lcosets-carrier*[*OF stabiliser-subgroup is-group*] *H* **have** *H* ⊆ *carrier G*
unfolding *LFactGroup-def* **by** *simp*
with *hH* **have** *h* ∈ *carrier G* **by** *auto*
then show (*rep H*) ⊙ *x* ∈ *orbit x* **using** *h orbit-def mem-Collect-eq* **by** *blast*
qed
show ⋀*y*. *y* ∈ *orbit x* ⟹ *y* ∈ (λ*H*. *rep H* ⊙ *x*) ‘ *carrier* (*G LMod stabiliser x*)
proof −
fix *y*
assume *y*:*y* ∈ *orbit x*

```
      obtain g  where gG:g ∈ carrier G and y = g ⊙ x using y orbit-def
by auto
      obtain H where H:H = g <# (stabiliser x) by auto
      with gG have H-carr:H ∈ carrier (G LMod stabiliser x)
        unfolding LFactGroup-def LCOSETS-def by auto
      then have rep H ∈ H using quotient-rep-ex by auto
      then obtain h where h-stab:h ∈ stabiliser x and gh:rep H = g ⊗ h
        unfolding H l-coset-def by auto
      have hG:h ∈ carrier G using h-stab stabiliser-def by auto
      from stabiliser-def h-stab have h ⊙ x = x by auto
      with ‹y = g ⊙ x› have y = g ⊙ (h ⊙ x) by simp
      then have y = (g ⊗ h) ⊙ x using gG hG compat-act by auto
      then have y = (rep H) ⊙ x using gh by simp
      then show y ∈ (λH. rep H ⊙ x) ‘ carrier (G LMod stabiliser x)
        using H-carr by simp
    qed
  qed
qed
```

The actual orbit-stabiliser theorem is a consequence of the bijection we established in the previous theorem and of Lagrange's theorem

```
theorem orbit-stabiliser:
  assumes finite: finite (carrier G)
  shows order G = card (orbit x) ∗ card (stabiliser x)
proof −
  have card (carrier (G LMod (stabiliser x))) = card (orbit x)
    using bij-betw-same-card orbit-stabiliser-bij by auto
  moreover have card (carrier (G LMod (stabiliser x))) ∗ card (stabiliser
x)  = order G
   using finite stabiliser-subgroup l-lagrange unfolding LFactGroup-def by
simp
  ultimately show ?thesis by simp
qed
end

end
```

2 Rotational Symmetries of the Tetrahedron

```
theory Tetrahedron
imports Orbit-Stabiliser
begin
```

2.1 Definition of the Tetrahedron and its Rotations

In this section we will use the orbit-stabiliser theorem to count the number of rotational symmetries of a tetrahedron.

The tetrahedron will be defined as a set of four vertices, labelled A, B, C, and D. A rotation is defined as a function between the vertices.

datatype *Vertex* = *A* | *B* | *C* | *D*
definition *vertices* :: *Vertex set* **where**
vertices = {*A*, *B*, *C*, *D*}

type-synonym *Rotation* = (*Vertex* ⇒ *Vertex*)

We define four primitive rotations explicitly. The axis of each rotation is the line through a vertex that is perpendicular to the face opposite to the vertex. Every rotation is by 120 degrees counter clockwise.

We also define the identity as a possible rotation.

definition *rotate-A* :: *Rotation* **where**
rotate-A = (λ*v*. (*case v of A* ⇒ *A* | *B* ⇒ *C* | *C* ⇒ *D* | *D* ⇒ *B*))
definition *rotate-B* :: *Rotation* **where**
rotate-B = (λ*v*. (*case v of A* ⇒ *D* | *B* ⇒ *B* | *C* ⇒ *A* | *D* ⇒ *C*))
definition *rotate-C* :: *Rotation* **where**
rotate-C = (λ*v*. (*case v of A* ⇒ *B* | *B* ⇒ *D* | *C* ⇒ *C* | *D* ⇒ *A*))
definition *rotate-D* :: *Rotation* **where**
rotate-D = (λ*v*. (*case v of A* ⇒ *C* | *B* ⇒ *A* | *C* ⇒ *B* | *D* ⇒ *D*))

named-theorems *simple-rotations*
declare *rotate-A-def* [*simple-rotations*] *rotate-B-def* [*simple-rotations*] *rotate-C-def* [*simple-rotations*] *rotate-D-def* [*simple-rotations*]

definition *simple-rotations* :: *Rotation set* **where**
simple-rotations = {*id*, *rotate-A*, *rotate-B*, *rotate-C*, *rotate-D*}

All other rotations are combinations of the previously defined simple rotations. We define these inductively.

inductive-set *complex-rotations* :: *Rotation set* **where**
simp: *r* ∈ *simple-rotations* ⟹ *r* ∈ *complex-rotations* |
comp: *r* ∈ *simple-rotations* ⟹ *s* ∈ *complex-rotations* ⟹ (*r* ∘ *s*) ∈ *complex-rotations*

2.2 Properties of Rotations

In this section we prove some basic properties of rotations that will be useful later. We begin by showing that rotations are bijections.

lemma *simple-rotations-inj*:
  assumes *r*:*r* ∈ *simple-rotations*
  shows *inj r*
  using *r* **unfolding** *simple-rotations-def*
  by *safe*
    (*rule injI*; *rename-tac a b*; *case-tac a*; *case-tac b*;
     simp add: *simple-rotations*
)+

lemma *simple-rotations-surj*:
  assumes *r*:*r* ∈ *simple-rotations*
  shows *surj r*
  using *r* **unfolding** *simple-rotations-def*
  by *safe*
    (*rename-tac a*; *case-tac a*;
     auto simp add: *simple-rotations Vertex.split*
)+

lemma *simple-rotations-bij*:
  assumes *r*:*r* ∈ *simple-rotations*
  shows *bij r*
  by (*simp add*: *r bij-def simple-rotations-surj simple-rotations-inj*)

lemma *complex-rotations-bij*: *r* ∈ *complex-rotations* ⟹ *bij r*
proof(*induction r rule*: *complex-rotations.induct*)
  case (*simp r*) **then show** *?case* **using** *simple-rotations-bij* **by** *simp*
next
  case (*comp r s*) **then show** *?case* **using** *bij-comp simple-rotations-bij* **by** *blast*
qed

lemma *simple-rotation-bij-corollary*: *r* ∈ *simple-rotations* ⟹ *r x* ≠ *r y* ⟷ *x* ≠ *y*
  using *bij-def simple-rotations-bij inj-eq* **by** *metis*

lemma *rotation-bij-corollary*: *r* ∈ *complex-rotations* ⟹ *r x* ≠ *r y* ⟷ *x* ≠ *y*
  using *bij-def complex-rotations-bij inj-eq* **by** *metis*

lemma *complex-rotations-comp*:
   r ∈ *complex-rotations* ⟹ *s* ∈ *complex-rotations* ⟹ (*r* ∘ *s*) ∈ *complex-rotations*
apply(*induction arbitrary*: *s rule*: *complex-rotations.induct*)
apply(*auto simp add*: *comp-assoc complex-rotations.comp*)

done

Next, we show that simple rotations (except the identity) keep exactly one vertex fixed.

lemma *simple-rotations-fix*:
  assumes *r*:*r* ∈ *simple-rotations*
  shows ∃ *v*. *r v* = *v*
  using *r* **unfolding** *simple-rotations-def*
  by (*auto simp add*: *simple-rotations split*: *Vertex.split*)

lemma *simple-rotations-fix-unique*:
  assumes *r*:*r* ∈ *simple-rotations*
  shows *r* ≠ *id* ⟹ *r v* = *v* ⟹ *r w* = *w* ⟹ *v* = *w*
  using *r* **unfolding** *simple-rotations-def*
  by *safe*
    (*case-tac v*; *case-tac w*;
     auto simp add: *simple-rotations*
)+

We also show that simple rotations do not contain cycles of length 2.

lemma *simple-rotations-cycle*:
  assumes *r*:*r* ∈ *simple-rotations*
  shows *r* ≠ *id* ⟹ *r v* = *w* ⟹ *v* ≠ *w* ⟹ *r w* ≠ *v*
  using *r* **unfolding** *simple-rotations-def*
  by *safe*
    (*case-tac v*;
     auto simp add: *simple-rotations*
)+

The following lemmas are all variations on the fact that any property that holds for 4 distinct vertices holds for all vertices. This is necessary to avoid having to use Vertex.exhaust as much as possible.

lemma *distinct-vertices*: *distinct*[(*a*::*Vertex*),*b*,*c*,*d*] ⟹ (∀ *e*. *e* ∈ {*a*,*b*,*c*,*d*})
apply(*safe*)
apply(*case-tac a*)
apply(*auto simp add*: *distinct-def*)
apply(*metis Vertex.exhaust*)+
done

lemma *distinct-map*: *r* ∈ *complex-rotations* ⟹ *distinct*[*a*,*b*,*c*,*d*] ⟹ (∀ *e* ∈ {*a*,*b*,*c*}. *r e* ≠ *f*) ⟹ *r d* = *f*
proof −
  assume *r*:*r* ∈ *complex-rotations* **and** *dist*:*distinct* [*a*,*b*,*c*,*d*] **and** *notf*:∀ *e* ∈ {*a*,*b*,*c*}. *r e* ≠ *f*

then have *1*:(∀ *v*. *v* ∈ {*a*,*b*,*c*,*d*}) **using** *distinct-vertices* **by** *simp*
from *complex-rotations-bij* **have** ∃ *v*. *r v* = *f* **using** *r bij-pointE* **by** *metis*
then have ∃ *v* ∈ {*a*,*b*,*c*,*d*}. *r v* = *f* **using** *1* **by** *blast*
then show *r d* = *f* **using** *notf* **by** *simp*
qed

lemma *distinct-map′*: *r* ∈ *complex-rotations* ⟹ *distinct*[*a*,*b*,*c*,*d*] ⟹ (∀ *e* ∈ {*a*,*b*,*c*}. *r f* ≠ *e*) ⟹ *r f* = *d*
proof −
assume *r*:*r* ∈ *complex-rotations* **and** *dist*:*distinct* [*a*,*b*,*c*,*d*] **and** *notf*:∀ *e* ∈ {*a*,*b*,*c*}. *r f* ≠ *e*
then have *1*:(∀ *v*. *v* ∈ {*a*,*b*,*c*,*d*}) **using** *distinct-vertices* **by** *simp*
have ∃ *v*. *r f* = *v* **by** *simp*
then have ∃ *v* ∈ {*a*,*b*,*c*,*d*}. *r f* = *v* **using** *1* **by** *blast*
then show *r f* = *d* **using** *notf* **by** *simp*
qed

lemma *cycle-map*: *r* ∈ *complex-rotations* ⟹ *distinct*[*a*,*b*,*c*,*d*] ⟹
r a = *b* ⟹ *r b* = *a* ⟹ *r c* = *d* ⟹ *r d* = *c* ⟹ ∀ *v w*. *r v* = *w* ⟶ *r w* = *v*
using *distinct-map′ rotation-bij-corollary* **by** *fastforce*

lemma *simple-distinct-map*: *r* ∈ *simple-rotations* ⟹ *distinct*[*a*,*b*,*c*,*d*] ⟹ (∀ *e* ∈ {*a*,*b*,*c*}. *r e* ≠ *f*) ⟹ *r d* = *f*
using *complex-rotations.simp distinct-map* **by** *simp*

lemma *simple-distinct-map′*: *r* ∈ *simple-rotations* ⟹ *distinct*[*a*,*b*,*c*,*d*] ⟹ (∀ *e* ∈ {*a*,*b*,*c*}. *r f* ≠ *e*) ⟹ *r f* = *d*
using *complex-rotations.simp distinct-map′* **by** *simp*

lemma *simple-distinct-ident*: *r* ∈ *simple-rotations* ⟹ *distinct*[*a*,*b*,*c*,*d*] ⟹ (∀ *e* ∈ {*a*,*b*,*c*}. *r e* ≠ *e*) ⟹ *r d* = *d*
using *simple-rotations-fix simple-distinct-map′* **by** *metis*

lemma *id-decomp*:
assumes *distinct*:*distinct* [(*a*::*Vertex*),*b*,*c*,*d*] **and** *ident*:(∀ *x* ∈ {*a*,*b*,*c*,*d*}. *r x* = *x*)
shows *r* = *id*
proof −
from *distinct-vertices* **have** ∀ *e*. *e* ∈ *set* [*a*,*b*,*c*,*d*] **using** *distinct* **by** *simp*
then have ∀ *e*. *r e* = *e* **using** *ident* **by** *auto*
then show *r* = *id* **by** *auto*
qed

Here we show that two invariants hold for rotations. Firstly, any rotation that does not fix a vertex consists of 2-cycles. Secondly, the only rotation that fixes more than one vertex is the identity.

This proof is very long in part because both invariants have to be proved simultaneously because they depend on each other.

lemma *complex-rotations-invariants*:
$r \in$ *complex-rotations* $\Longrightarrow (((\forall\ v.\ r\ v \neq v) \longrightarrow r\ v = w \longrightarrow r\ w = v) \land (r\ v = v \longrightarrow r\ w = w \longrightarrow v \neq w \longrightarrow r = id))$
proof(*induction r arbitrary*: *v w rule*: *complex-rotations.induct*)
case (*simp r*)
assume *r*:$r \in$ *simple-rotations*
show *?case*
proof
have $\exists\ v.\ r\ v = v$ **using** *simple-rotations-fix r* **by** *simp*
then have $\neg\ (\forall\ v.\ r\ v \neq v)$ **by** *simp*
then show $(\forall\ v.\ r\ v \neq v) \longrightarrow r\ v = w \longrightarrow r\ w = v$ **by** *blast*

show $r\ v = v \longrightarrow r\ w = w \longrightarrow v \neq w \longrightarrow r = id$ **using** *simple-rotations-fix-unique simp* **by** *blast*
qed
next
case (*comp r s*)
assume *r*:$r \in$ *simple-rotations*
assume *s*:$s \in$ *complex-rotations*
have *fix-unique*:$\forall\ v\ w.\ s\ v = v \longrightarrow s\ w = w \longrightarrow v \neq w \longrightarrow s = id$ **using** *comp* **by** *blast*
show *?case*
proof
show $(\forall x.\ (r \circ s)\ x \neq x) \longrightarrow (r \circ s)\ v = w \longrightarrow (r \circ s)\ w = v$
proof (*rule impI*)+
assume *nofixrs*:$\forall\ x.(r \circ s)\ x \neq x$
assume $(r \circ s)\ v = w$
show $(r \circ s)\ w = v$
proof (*cases* $\forall\ x.\ s\ x \neq x$)
assume *nofixs*:$\forall\ x.\ s\ x \neq x$
then have *cycle*:$\forall\ x\ y.\ (s\ x = y \longrightarrow s\ y = x)$ **using** *comp.IH* **by** *blast*
then show *?thesis*
proof (*cases* $r = id$)
assume *id*:$r = id$
then have $s\ v = w$ **using** ‹$(r \circ s)\ v = w$› **by** *simp*
then have $s\ w = v$ **using** *cycle* **by** *blast*
then show $(r \circ s)\ w = v$ **using** *id* **by** *simp*

next
assume *notid*:$r \neq id$
obtain a **where** $s\ v = a$ **and** $s\ a = v$ **and** $a \neq v$ **using** *comp.IH nofixs* **by** *blast*
obtain b **where** $s\ w = b$ **and** $s\ b = w$ **and** $b \neq w$ **using** *comp.IH nofixs* **by** *blast*
have $v \neq w$ **using** ‹$(r \circ s)\ v = w$› *nofixrs* **by** *blast*
then have $a \neq b$ **using** *comp.hyps rotation-bij-corollary* ‹$s\ a = v$› ‹$s\ b = w$› **by** *auto*
have $r\ a = w$ **using** ‹$s\ v = a$› ‹$(r \circ s)\ v = w$› **by** *auto*
then show *?thesis*
proof (*cases* $a = w$)
assume $a = w$
then have $r\ a = a$ **using** ‹$r\ a = w$› **by** *simp*
then have $s\ v = w$ **using** ‹$a = w$› ‹$s\ v = a$› **by** *simp*
then have $b = v$ **using** ‹$s\ b = w$› *rotation-bij-corollary comp.hyps* **by** *blast*
then have $s\ w = v$ **using** ‹$s\ w = b$› **by** *simp*
then have $r\ v \neq v$ **using** *simple-rotations-fix-unique notid* ‹$r\ a = a$› ‹$a \neq v$›
comp.hyps(*1*) **by** *auto*
obtain $c\ d$ **where** $s\ c = d$ **and** $c \neq v$ **and** $c \neq w$
using ‹$a \neq v$› ‹$r\ a = w$› ‹$r\ v \neq v$› *comp.hyps*(*1*) *simple-rotation-bij-corollary* **by** *blast*
then have $d \neq v$ **and** $d \neq w$
using ‹$s\ w = v$› ‹$c \neq v$› ‹$s\ c = d$› ‹$s\ v = w$› *comp.hyps*(*2*) *rotation-bij-corollary* **by** *auto*
then have $s\ d = c$ **using** ‹$s\ c = d$› *comp.IH nofixs* **by** *blast*
then have $c \neq d$ **using** *nofixs* **by** *auto*
then show *?thesis*
proof(*cases* $r\ v = c$)
assume $r\ v = c$
then have $r\ c \neq v$ **using** ‹$c \neq v$› *simple-rotations-cycle comp.hyps*(*1*) *notid* **by** *simp*
have $r\ c \neq w$
using ‹$r\ a = a$› ‹$c \neq w$› ‹$r\ a = w$› *simple-rotation-bij-corollary comp.hyps*(*1*) **by** *auto*
have $r\ c \neq c$ **using** ‹$a = w$› ‹$c \neq w$› ‹$r\ a = a$›
comp.hyps(*1*) *simple-rotations-fix-unique notid* **by** *blast*
have *dist*:*distinct* $[v,w,c,d]$ **using** ‹$c \neq v$› ‹$c \neq w$› ‹$c \neq d$› ‹$d \neq v$› ‹$d \neq w$› ‹$v \neq w$› **by** *simp*
then have $\forall\ v \in \{v,w,c\}.\ r\ c \neq v$ **using** ‹$r\ c \neq c$› ‹$r\ c \neq v$› ‹$r\ c \neq w$› **by** *auto*
then have $r\ c = d$ **using** *simple-distinct-map′ comp.hyps*(*1*)

dist **by** *auto*
then have $(r \circ s)\ d = d$ **using** ‹$s\ d = c$› **by** *simp*
then have *False* **using** *nofixrs* **by** *blast*
then show *?thesis* **by** *simp*
next
assume $r\ v \neq c$
then have $r\ v \neq w$
using ‹$r\ a = a$› ‹$v \neq w$› ‹$r\ a = w$› *simple-rotation-bij-corollary* *comp.hyps*(*1*) **by** *auto*
then have $r\ v \neq v$ **using** ‹$a = w$› ‹$r\ a = a$›
comp.hyps(*1*) *simple-rotations-fix-unique notid* **by** *blast*
have *dist*:*distinct* [w,c,v,d] **using** ‹$c \neq v$› ‹$c \neq w$› ‹$c \neq d$› ‹$d \neq v$› ‹$d \neq w$› ‹$v \neq w$› **by** *simp*
then have $\forall\ x \in \{w,c,v\}.\ r\ v \neq x$ **using** ‹$r\ v \neq c$› ‹$r\ v \neq v$› ‹$r\ v \neq w$› **by** *auto*
then have $r\ v = d$ **using** *simple-distinct-map′* *comp.hyps*(*1*) *dist* **by** *auto*
then have $r\ d \neq v$ **using** ‹$d \neq v$› *simple-rotations-cycle* *comp.hyps*(*1*) *notid* **by** *simp*
have $r\ d \neq w$
using ‹$r\ a = a$› ‹$d \neq w$› ‹$r\ a = w$› *simple-rotation-bij-corollary* *comp.hyps*(*1*) **by** *auto*
have $r\ d \neq d$ **using** ‹$a = w$› ‹$d \neq w$› ‹$r\ a = a$›
comp.hyps(*1*) *simple-rotations-fix-unique notid* **by** *blast*
have *dist′*:*distinct* [w,v,d,c] **using** ‹$c \neq v$› ‹$c \neq w$› ‹$c \neq d$› ‹$d \neq v$› ‹$d \neq w$› ‹$v \neq w$› **by** *simp*
then have $\forall\ v \in \{w,v,d\}.\ r\ d \neq v$ **using** ‹$r\ d \neq d$› ‹$r\ d \neq w$› ‹$r\ d \neq v$› **by** *auto*
then have $r\ d = c$ **using** *simple-distinct-map′* *comp.hyps*(*1*) *dist′* **by** *auto*
then have $(r \circ s)\ c = c$ **using** ‹$s\ c = d$› **by** *simp*
then have *False* **using** *nofixrs* **by** *blast*
then show *?thesis* **by** *simp*
qed
next
assume $a \neq w$
then have $r\ a \neq a$ **using** ‹$r\ a = w$› **by** *simp*
have $b \neq v$ **using** ‹$a \neq w$› ‹$s\ b = w$› ‹$s\ v = a$› **by** *auto*
have $r\ w \neq w$ **using** ‹$a \neq w$› ‹$r\ a = w$› *comp.hyps*(*1*) *simple-rotation-bij-corollary* **by** *auto*
from *nofixs* **have** $s\ w \neq w$ **by** *simp*
then have $r\ v \neq w$ **using** ‹$a \neq v$› ‹$r\ a = w$› *comp.hyps* *simple-rotation-bij-corollary* **by** *blast*
have $s\ v \neq w$ **using** ‹$r\ a = w$› ‹$r\ a \neq a$› ‹$s\ v = a$› **by** *blast*

then show *?thesis*
proof (*cases* $r\ b = b$)
assume $r\ b = b$
then have $r\ b \neq a$ **using** ‹$a \neq b$› **by** *simp*
have $r\ w \neq a$ **using** ‹$r\ a = w$› ‹$r\ w \neq w$› *comp.hyps*(*1*) *notid simple-rotations-cycle* **by** *blast*
have *dist*:*distinct* $[a,b,w,v]$ **using** ‹$a \neq w$› ‹$a \neq b$› ‹$a \neq v$› ‹$b \neq w$› ‹$b \neq v$› ‹$v \neq w$› **by** *simp*
then have $\forall\ x \in \{a,b,w\}.\ r\ x \neq a$ **using** ‹$r\ a \neq a$› ‹$r\ b \neq a$› ‹$r\ w \neq a$› **by** *auto*
then have $r\ v = a$ **using** *simple-distinct-map comp.hyps*(*1*) *dist* **by** *auto*
then show *?thesis* **using** ‹$s\ a = v$› *nofixrs comp-apply* **by** *metis*
next
assume $r\ b \neq b$
have *dist*:*distinct* $[w,a,b,v]$ **using** ‹$a \neq w$› ‹$a \neq b$› ‹$a \neq v$› ‹$b \neq w$› ‹$b \neq v$› ‹$v \neq w$› **by** *simp*
then have $\forall\ x \in \{w,a,b\}.\ r\ x \neq x$ **using** ‹$r\ w \neq w$› ‹$r\ a \neq a$› ‹$r\ b \neq b$› **by** *auto*
then have $r\ v = v$ **using** *simple-distinct-ident comp.hyps*(*1*) *dist* **by** *auto*
have $r\ w \neq a$ **using** ‹$a \neq w$› *simple-rotations-cycle comp.hyps*(*1*) *notid* ‹$r\ a = w$› **by** *simp*
have $r\ w \neq v$ **using** ‹$r\ v = v$› ‹$v \neq w$› *comp.hyps*(*1*) *simple-rotation-bij-corollary* **by** *blast*
have *dist'*:*distinct* $[a,v,w,b]$ **using** ‹$a \neq w$› ‹$a \neq b$› ‹$a \neq v$› ‹$b \neq w$› ‹$b \neq v$› ‹$v \neq w$› **by** *simp*
then have $\forall\ x \in \{a,v,w\}.\ r\ w \neq x$ **using** ‹$r\ w \neq a$› ‹$r\ w \neq v$› ‹$r\ w \neq w$› **by** *auto*
then have $r\ w = b$ **using** *simple-distinct-map'* *comp.hyps*(*1*) *dist'* **by** *auto*
then show *?thesis* **using** ‹$s\ b = w$› *nofixrs comp-apply* **by** *metis*
qed
qed
qed
next
assume $\neg\ (\forall\ v.\ s\ v \neq v)$
then have *fix1*:$\exists\ v.\ s\ v = v$ **by** *blast*
then obtain *a* **where** *a*:$s\ a = a$ **by** *blast*
then show *?thesis*
proof (*cases* $r = id$)
assume *id*:$r = id$
then have $(r \circ s)\ a = a$ **using** *a* **by** *simp*
then have *False* **using** *nofixrs* **by** *auto*

then show *?thesis* **by** *simp*
next
assume *notid*: *r* ≠ *id*
then have *fix1*:∃ *v*. *r v* = *v* **using** *simple-rotations-fix comp.hyps* **by** *simp*
then obtain *b* **where** *b*:*r b* = *b* **by** *blast*
then show *?thesis*
proof (*cases a* = *b*)
assume *a* = *b*
then have (*r* ∘ *s*) *a* = *a* **using** *a b* **by** *simp*
then have *False* **using** *nofixrs* **by** *blast*
then show *?thesis* **by** *simp*
next
assume *a* ≠ *b*
have *r a* ≠ *a* **using** ‹*a* ≠ *b*› *b comp.hyps*(*1*) *notid simple-rotations-fix-unique* **by** *blast*
have *r a* ≠ *b* **using** ‹*a* ≠ *b*› *b comp.hyps*(*1*) *simple-rotation-bij-corollary* **by** *auto*
then obtain *c* **where** *r a* = *c* **and** *a* ≠ *c* **and** *b* ≠ *c* **using** ‹*r a* ≠ *a*› **by** *auto*
have *s b* ≠ *a* **using** ‹*a* ≠ *b*› *a comp.hyps*(*2*) *rotation-bij-corollary* **by** *blast*
have *s b* ≠ *b* **using** *b nofixrs comp-apply* **by** *metis*
then obtain *d* **where** *s b* = *d* **and** *a* ≠ *d* **and** *b* ≠ *d* **using** ‹*s b* ≠ *a*› **by** *auto*
have *r c* ≠ *a* **using** *simple-rotations-cycle* ‹*a* ≠ *c*› ‹*r a* = *c*› *comp.hyps*(*1*) *notid* **by** *blast*
have *r c* ≠ *b* **using** ‹*b* ≠ *c*› *b comp.hyps*(*1*) *simple-rotation-bij-corollary* **by** *blast*
have *r c* ≠ *c* **using** ‹*b* ≠ *c*› *b comp.hyps*(*1*) *notid simple-rotations-fix-unique* **by** *blast*
then show *?thesis*
proof (*cases c* = *d*)
assume *c* = *d*
then have *s c* ≠ *c* **using** ‹*b* ≠ *c*› ‹*s b* = *d*› *rotation-bij-corollary s* **by** *auto*
obtain *e* **where** *r c* = *e* **and** *a* ≠ *e* **and** *b* ≠ *e* **and** *c* ≠ *e* **and** *d* ≠ *e*
using ‹*r c* ≠ *a*› ‹*r c* ≠ *b*› ‹*r c* ≠ *c*› ‹*c* = *d*› **by** *auto*
have *r e* ≠ *b* **using** ‹*b* ≠ *e*› *b r simple-rotation-bij-corollary* **by** *blast*
have *r e* ≠ *c* **using** ‹*a* ≠ *e*› ‹*r a* = *c*› *r simple-rotation-bij-corollary* **by** *blast*
have *r e* ≠ *e* **using** ‹*b* ≠ *e*› *b notid r simple-rotations-fix-unique*

by *blast*
then have *dist*:*distinct* [*b*,*c*,*e*,*a*] **using** ‹$a \neq b$› ‹$a \neq c$› ‹$a \neq e$› ‹$b \neq c$› ‹$b \neq e$› ‹$c \neq e$› **by** *simp*
then have $\forall\ x \in \{b,c,e\}.\ r\ e \neq x$ **using** ‹$r\ e \neq b$› ‹$r\ e \neq c$› ‹$r\ e \neq e$› **by** *auto*
then have $r\ e = a$ **using** *simple-distinct-map′ comp.hyps*(*1*) *dist* **by** *auto*
have *dist*:*distinct* [*a*,*b*,*c*,*e*] **using** ‹$a \neq b$› ‹$a \neq c$› ‹$a \neq e$› ‹$b \neq c$› ‹$b \neq e$› ‹$c \neq e$› **by** *simp*
then have $\forall\ x \in \{a,b,c\}.\ r\ c \neq x$ **using** ‹$r\ c \neq a$› ‹$r\ c \neq b$› ‹$r\ c \neq c$› **by** *auto*
then have $r\ c = e$ **using** *simple-distinct-map′ comp.hyps*(*1*) *dist* **by** *auto*
have $s\ e \neq a$ **using** ‹$a \neq e$› *a rotation-bij-corollary s* **by** *blast*
have $s\ e \neq c$ **using** ‹$b \neq e$› ‹$c = d$› ‹$s\ b = d$› *rotation-bij-corollary s* **by** *blast*
have $s\ e \neq e$ **using** ‹$a \neq e$› ‹$s\ b \neq b$› *a fix-unique* **by** *fastforce*

then have *dist*:*distinct* [*a*,*c*,*e*,*b*] **using** ‹$a \neq b$› ‹$a \neq c$› ‹$a \neq e$› ‹$b \neq c$› ‹$b \neq e$› ‹$c \neq e$› **by** *simp*
then have $\forall\ x \in \{a,c,e\}.\ s\ e \neq x$ **using** ‹$s\ e \neq a$› ‹$s\ e \neq c$› ‹$s\ e \neq e$› **by** *auto*
then have $s\ e = b$ **using** *distinct-map′ comp.hyps*(*2*) *dist* **by** *auto*
have $s\ c \neq a$ **using** ‹$a \neq c$› *a rotation-bij-corollary s* **by** *blast*
have $s\ c \neq b$ **using** ‹$c \neq e$› ‹$s\ e = b$› *rotation-bij-corollary s* **by** *blast*
then have *dist*:*distinct* [*a*,*b*,*c*,*e*] **using** ‹$a \neq b$› ‹$a \neq c$› ‹$a \neq e$› ‹$b \neq c$› ‹$b \neq e$› ‹$c \neq e$› **by** *simp*
then have $\forall\ x \in \{a,b,c\}.\ s\ c \neq x$ **using** ‹$s\ c \neq a$› ‹$s\ c \neq b$› ‹$s\ c \neq c$› **by** *auto*
then have $s\ c = e$ **using** *distinct-map′ comp.hyps*(*2*) *dist* **by** *auto*

have *rsa*:$(r \circ s)\ a = c$ **using** ‹$r\ a = c$› *a* **by** *simp*
have *rsb*:$(r \circ s)\ b = e$ **using** ‹$c = d$› ‹$r\ c = e$› ‹$s\ b = d$› **by** *auto*

have *rsc*:$(r \circ s)\ c = a$ **using** ‹$r\ e = a$› ‹$s\ c = e$› **by** *auto*
have *rse*:$(r \circ s)\ e = b$ **using** ‹$s\ e = b$› *b* **by** *simp*
then have *dist*:*distinct* [*a*,*c*,*b*,*e*] **using** ‹$a \neq b$› ‹$a \neq c$› ‹$a \neq e$› ‹$b \neq c$› ‹$b \neq e$› ‹$c \neq e$› **by** *simp*
have *comprs*:$r \circ s \in$ *complex-rotations* **using** *complex-rotations.comp r s* **by** *simp*
show *?thesis* **using** *cycle-map*[*OF comprs dist rsa rsc rsb rse*] ‹$(r \circ s)\ v = w$› **by** *blast*

next
assume $c \neq d$
then have *dist*:*distinct* $[a,b,c,d]$ **using** ‹$a \neq b$› ‹$a \neq c$› ‹$a \neq d$› ‹$b \neq c$› ‹$b \neq d$› ‹$c \neq d$› **by** *simp*
then have $\forall\ x \in \{a,b,c\}.\ r\ c \neq x$ **using** ‹$r\ c \neq a$› ‹$r\ c \neq b$› ‹$r\ c \neq c$› **by** *auto*
then have $r\ c = d$ **using** *simple-distinct-map′ comp.hyps*(*1*) *dist* **by** *auto*
have $r\ d \neq b$ **using** ‹$b \neq d$› *b comp.hyps*(*1*) *simple-rotation-bij-corollary* **by** *blast*
have $r\ d \neq c$ **using** ‹$c \neq d$› ‹$r\ c = d$› *comp.hyps*(*1*) *notid simple-rotations-cycle* **by** *blast*
have $r\ d \neq d$ **using** ‹$c \neq d$› ‹$r\ c = d$› *comp.hyps*(*1*) *simple-rotation-bij-corollary* **by** *auto*
have *dist*:*distinct* $[b,c,d,a]$ **using** ‹$a \neq b$› ‹$a \neq c$› ‹$a \neq d$› ‹$b \neq c$› ‹$b \neq d$› ‹$c \neq d$› **by** *simp*
then have $\forall\ x \in \{b,c,d\}.\ r\ d \neq x$ **using** ‹$r\ d \neq b$› ‹$r\ d \neq c$› ‹$r\ d \neq d$› **by** *auto*
then have $r\ d = a$ **using** *simple-distinct-map′ comp.hyps*(*1*) *dist* **by** *auto*
have $s\ d \neq a$ **using** ‹$a \neq d$› *a comp.hyps*(*2*) *rotation-bij-corollary* **by** *blast*
have $s\ d \neq c$ **using** *nofixrs* ‹$r\ c = d$› ‹$c \neq d$› *comp-apply* **by** *metis*
have $s\ d \neq d$ **using** ‹$b \neq d$› ‹$s\ b = d$› *comp.hyps*(*2*) *rotation-bij-corollary* **by** *auto*
have *dist*:*distinct* $[a,c,d,b]$ **using** ‹$a \neq b$› ‹$a \neq c$› ‹$a \neq d$› ‹$b \neq c$› ‹$b \neq d$› ‹$c \neq d$› **by** *simp*
then have $\forall\ x \in \{a,c,d\}.\ s\ d \neq x$ **using** ‹$s\ d \neq a$› ‹$s\ d \neq c$› ‹$s\ d \neq d$› **by** *auto*
then have $s\ d = b$ **using** *distinct-map′ comp.hyps*(*2*) *dist* **by** *auto*
have $s\ c \neq a$ **using** ‹$a \neq c$› *a comp.hyps*(*2*) *rotation-bij-corollary* **by** *blast*
have $s\ c \neq b$ **using** ‹$c \neq d$› ‹$s\ d = b$› *comp.hyps*(*2*) *rotation-bij-corollary* **by** *blast*
have $s\ c \neq d$ **using** ‹$b \neq c$› ‹$s\ b = d$› *comp.hyps*(*2*) *rotation-bij-corollary* **by** *blast*
have *dist*:*distinct* $[a,b,d,c]$ **using** ‹$a \neq b$› ‹$a \neq c$› ‹$a \neq d$› ‹$b \neq c$› ‹$b \neq d$› ‹$c \neq d$› **by** *simp*
then have $\forall\ x \in \{a,b,d\}.\ s\ c \neq x$ **using** ‹$s\ c \neq a$› ‹$s\ c \neq b$› ‹$s\ c \neq d$› **by** *auto*
then have $s\ c = c$ **using** *distinct-map′ comp.hyps*(*2*) *dist* **by** *auto*

then have *False* **using** *fix-unique* ‹$s\ d \neq d$› ‹$a \neq c$› *a* **by** *fastforce*

then show *?thesis* **by** *simp*
qed
qed
qed
qed
qed
next
show $(r \circ s)\ v = v \longrightarrow (r \circ s)\ w = w \longrightarrow v \neq w \longrightarrow r \circ s = id$
proof(*rule impI*)+
assume *rsv*:$(r \circ s)\ v = v$ **and** *rsw*:$(r \circ s)\ w = w$ **and** $v \neq w$
show $r \circ s = id$
proof($cases\ s = id$)
assume *sid*:$s = id$
then have $s\ v = v$ **and** $s\ w = w$ **by** *auto*
then have $r = id$ **using** *simple-rotations-fix-unique rsv rsw* ‹$v \neq w$› *r* **by** *auto*
with *sid* **show** *?thesis* **by** *simp*
next
assume *snotid*:$s \neq id$
then show *?thesis*
proof($cases\ r = id$)
assume *rid*:$r = id$
then have $s\ v = v$ **and** $s\ w = w$ **using** *rsv rsw* **by** *auto*
then have $s = id$ **using** ‹$v \neq w$› *fix-unique* **by** *blast*
with *rid* **show** *?thesis* **by** *simp*
next
assume *rnotid*:$r \neq id$
from *simple-rotations-fix-unique*[*OF comp.hyps*(*1*) *rnotid*] **have**
r-fix-forall:$\forall\ v\ w.\ r\ v = v \wedge r\ w = w \longrightarrow v = w$ **by** *blast*
from *comp.IH snotid* **have**
s-fix-forall:$\forall\ v\ w.\ s\ v = v \wedge s\ w = w \longrightarrow v = w$ **by** *blast*
have *fixes-two*: $\exists\, a\ b.\ (r \circ s)\ a = a \wedge (r \circ s)\ b = b \wedge a \neq b$ **using** ‹$v \neq w$› *rsv rsw* **by** *blast*
then show *?thesis*
proof ($cases\ \forall\ x.\ s\ x \neq x$)
assume *sfix'*:$\forall\ x.\ s\ x \neq x$
from *simple-rotations-fix* **obtain** *a* **where** *a*:$r\ a = a$ **using** *r* **by** *blast*
from *sfix'* **have** $s\ a \neq a$ **by** *blast*
then have $(r \circ s)\ a \neq a$ **using** *a simple-rotation-bij-corollary r* **by** *fastforce*
with *fixes-two* **obtain** *b* **where** $(r \circ s)\ b = b$ **and** $b \neq a$ **by** *blast*

with *fixes-two* **obtain** *c* **where** (*r* ∘ *s*) *c* = *c* **and** *c* ≠ *a* **and** *c* ≠ *b*
using ‹(*r* ∘ *s*) *a* ≠ *a*› **by** *blast*

have *s b* ≠ *a* **using** *a* ‹(*r* ∘ *s*) *b* = *b*› *sfix′* **by** *force*
have *s c* ≠ *a* **using** *a* ‹(*r* ∘ *s*) *c* = *c*› *sfix′* **by** *force*

then obtain *d* **where** *s d* = *a* **and** *d* ≠ *a* **and** *d* ≠ *b* **and** *d* ≠ *c*
using ‹*s a* ≠ *a*› ‹*s b* ≠ *a*› ‹*s c* ≠ *a*› *complex-rotations-bij s bij-pointE* **by** *metis*
have (*r* ∘ *s*) *d* = *a* **using** *a* ‹*s d* = *a*› **by** *simp*

have *r b* ≠ *a* **using** *a r simple-rotation-bij-corollary* ‹*b* ≠ *a*› **by** *auto*
have *r c* ≠ *a* **using** *a r simple-rotation-bij-corollary* ‹*c* ≠ *a*› **by** *auto*
have *r d* ≠ *a* **using** *a r simple-rotation-bij-corollary* ‹*d* ≠ *a*› **by** *auto*
have *r b* ≠ *b* **using** *a r simple-rotations-fix-unique rnotid* ‹*b* ≠ *a*› **by** *blast*
have *r c* ≠ *c* **using** *a r simple-rotations-fix-unique rnotid* ‹*c* ≠ *a*› **by** *blast*
have *r d* ≠ *d* **using** *a r simple-rotations-fix-unique rnotid* ‹*d* ≠ *a*› **by** *blast*

then have *False* **using** *sfix′*
proof (*cases r b* = *c*)
assume *r b* = *c*
then have *r c* ≠ *c* **using** *r simple-rotation-bij-corollary* ‹*c* ≠ *b*› **by** *auto*
then have *r c* ≠ *b* **using** *r rnotid simple-rotations-cycle* ‹*r b* = *c*› **by** *auto*
have *dist*:*distinct* [*a*,*b*,*c*,*d*] **using** ‹*c* ≠ *a*› ‹*d* ≠ *a*› ‹*d* ≠ *c*› ‹*d* ≠ *b*› ‹*c* ≠ *b*› ‹*b* ≠ *a*› **by** *simp*
then have ∀ *v* ∈ {*a*,*b*,*c*}. *r c* ≠ *v* **using** ‹*r c* ≠ *c*› ‹*r c* ≠ *a*› ‹*r c* ≠ *b*› **by** *auto*
then have *r c* = *d* **using** *simple-distinct-map′ r dist* **by** *auto*

have *r d* ≠ *c* **using** *r simple-rotation-bij-corollary* ‹*d* ≠ *b*› ‹*r b* = *c*› **by** *auto*
have *r d* ≠ *d* **using** *r a* ‹*d* ≠ *a*› ‹*r d* ≠ *d*› **by** *simp*
have *dist′*:*distinct* [*a*,*c*,*d*,*b*] **using** ‹*c* ≠ *a*› ‹*d* ≠ *a*› ‹*d* ≠ *c*› ‹*d* ≠ *b*› ‹*c* ≠ *b*› ‹*b* ≠ *a*› **by** *simp*
then have ∀ *v* ∈ {*a*,*c*,*d*}. *r d* ≠ *v* **using** ‹*r d* ≠ *c*› ‹*r d* ≠ *a*› ‹*r*

$d \neq d$› **by** *auto*

then have $r\ d = b$ **using** *simple-distinct-map′ r dist′* **by** *auto*

then have $s\ b = d$ **using** ‹$(r \circ s)\ b = b$› *r simple-rotation-bij-corollary* **by** *auto*

have $s\ c = b$ **using** ‹$(r \circ s)\ c = c$› ‹$r\ b = c$› *r simple-rotation-bij-corollary* **by** *auto*

then have $s\ b \neq c$ **using** ‹$s\ b = d$› ‹$d \neq c$› **by** *simp*
then show *False* **using** *s sfix′* ‹$s\ c = b$› *comp(3)* **by** *blast*
next
assume $r\ b \neq c$
have *dist′*:*distinct* $[a,b,c,d]$ **using** ‹$c \neq a$› ‹$d \neq a$› ‹$d \neq c$› ‹$d \neq b$› ‹$c \neq b$› ‹$b \neq a$› **by** *simp*
then have $\forall\ v \in \{a,b,c\}.\ r\ b \neq v$ **using** ‹$r\ b \neq a$› ‹$r\ b \neq b$› ‹$r\ b \neq c$› **by** *auto*
then have $r\ b = d$ **using** *simple-distinct-map′ r dist′* **by** *auto*

then have $r\ c \neq d$ **using** *r simple-rotation-bij-corollary* ‹$c \neq b$› **by** *auto*
have *dist″*:*distinct* $[a,c,d,b]$ **using** ‹$c \neq a$› ‹$d \neq a$› ‹$d \neq c$› ‹$d \neq b$› ‹$c \neq b$› ‹$b \neq a$› **by** *simp*
then have $\forall\ v \in \{a,c,d\}.\ r\ c \neq v$ **using** ‹$r\ c \neq a$› ‹$r\ c \neq c$› ‹$r\ c \neq d$› **by** *auto*
then have $r\ c = b$ **using** *simple-distinct-map′ r dist″* **by** *auto*

then have $r\ d \neq b$ **using** *r simple-rotation-bij-corollary* ‹$d \neq c$› **by** *auto*
have *dist‴*:*distinct* $[a,b,d,c]$ **using** ‹$c \neq a$› ‹$d \neq a$› ‹$d \neq c$› ‹$d \neq b$› ‹$c \neq b$› ‹$b \neq a$› **by** *simp*
then have $\forall\ v \in \{a,b,d\}.\ r\ d \neq v$ **using** ‹$r\ d \neq a$› ‹$r\ d \neq b$› ‹$r\ d \neq d$› **by** *auto*
then have $r\ d = c$ **using** *simple-distinct-map′ r dist‴* **by** *auto*

then have $s\ b = c$ **using** ‹$r\ c = b$› ‹$(r \circ s)\ b = b$› *r simple-rotation-bij-corollary* **by** *auto*
have $s\ c = d$ **using** ‹$(r \circ s)\ c = c$› ‹$r\ d = c$› *r simple-rotation-bij-corollary* **by** *auto*

then have $s\ c \neq b$ **using** ‹$d \neq b$› **by** *simp*
then have *False* **using** *comp(3) s sfix′* ‹$s\ b = c$› **by** *blast*
then show *?thesis* **by** *simp*
qed
then show *?thesis* **by** *simp*

next
assume ¬ (∀ *x*. *s x* ≠ *x*)
then have ∃ *x*. *s x* = *x* **by** *simp*
then obtain *a* **where** *a*:*s a* = *a* **by** *blast*
from *simple-rotations-fix* **obtain** *b* **where** *b*:*r b* = *b* **using** *r* **by** *blast*
then show *?thesis*
proof (*cases a* = *b*)
assume *a* ≠ *b*
with *a b* **have** *r a* ≠ *a* **using** *r rnotid simple-rotations-fix-unique* **by** *blast*
then have (*r* ∘ *s*) *a* ≠ *a* **using** *a* **by** *simp*
have *s b* ≠ *b* **using** *a* ‹*a* ≠ *b*› *s-fix-forall* **by** *blast*
then have (*r* ∘ *s*) *b* ≠ *b* **using** *b simple-rotations-inj r*
complex-rotations.simp rotation-bij-corollary **by** *fastforce*
with *fixes-two* **obtain** *c* **where** (*r* ∘ *s*) *c* = *c* **and** *c* ≠ *a* **and** *c* ≠ *b* **using** ‹(*r* ∘ *s*) *a* ≠ *a*› **by** *blast*
from *fixes-two* **obtain** *d* **where** (*r* ∘ *s*) *d* = *d* **and** *d* ≠ *a* **and** *d* ≠ *b* **and** *d* ≠ *c*
using ‹(*r* ∘ *s*) *a* ≠ *a*› ‹(*r* ∘ *s*) *b* ≠ *b*› **by** *blast*

have *s c* ≠ *a* **using** *a* ‹*c* ≠ *a*› *rotation-bij-corollary s* **by** *force*
have *s d* ≠ *a* **using** *a* ‹*d* ≠ *a*› *rotation-bij-corollary s* **by** *force*

have *r a* ≠ *c* **using** ‹*s c* ≠ *a*› ‹(*r* ∘ *s*) *c* = *c*› ‹*c* ≠ *a*› *r simple-rotation-bij-corollary* **by** *auto*
have *r a* ≠ *d* **using** ‹*s d* ≠ *a*› ‹(*r* ∘ *s*) *d* = *d*› ‹*d* ≠ *a*› *r simple-rotation-bij-corollary* **by** *auto*
have *r a* ≠ *b* **using** *b simple-rotation-bij-corollary* ‹*a* ≠ *b*› *r* **by** *auto*

have *dist*:*distinct* [*b*,*c*,*d*,*a*] **using** ‹*c* ≠ *a*› ‹*d* ≠ *a*› ‹*c* ≠ *b*› ‹*a* ≠ *b*› ‹*d* ≠ *c*› ‹*d* ≠ *b*› **by** *simp*
then have ∀ *v* ∈ {*b*,*c*,*d*}. *r a* ≠ *v* **using** ‹*r a* ≠ *b*› ‹*r a* ≠ *c*› ‹*r a* ≠ *d*› **by** *auto*
then have *r a* = *a* **using** *simple-distinct-map′ r dist* **by** *simp*
then have *False* **using** ‹*r a* ≠ *a*› **by** *simp*
then show *?thesis* **by** *simp*
next
assume *a* = *b*
with *a b* **have** (*r* ∘ *s*) *a* = *a* **by** *simp*
from *fixes-two* **obtain** *c* **where** *rsc*:(*r* ∘ *s*) *c* = *c* **and** *c* ≠ *a* **by** *blast*
then have *r c* ≠ *c* **using** *b* ‹*a* = *b*›*r rnotid simple-rotations-fix-unique*

by *blast*
then have *s c* ≠ *c* **using** *rsc* **by** *auto*
then obtain *d* **where** *s c* = *d* **and** *d* ≠ *c* **by** *blast*
then have *d* ≠ *a* **using** *a s rotation-bij-corollary* **by** *blast*
have *s d* ≠ *d* **using** *a* **using** ‹*d* ≠ *a*› *s-fix-forall* **by** *blast*
have *r d* = *c* **using** *rsc* ‹*s c* = *d*› **by** *simp*
then have *r c* ≠ *d* **using** ‹*d* ≠ *c*› *simple-rotations-cycle r rnotid* **by** *auto*
then obtain *e* **where** *r c* = *e* **and** *e* ≠ *d* **by** *blast*
then have *e* ≠ *a* **using** *b* ‹*a* = *b*› *simple-rotation-bij-corollary* ‹*c* ≠ *a*› *r* **by** *auto*
then have *e* ≠ *c* **using** *b* ‹*a* = *b*› ‹*r c* = *e*› ‹*r c* ≠ *c*› **by** *blast*
then have *r e* ≠ *c* **using** ‹*r c* = *e*› *simple-rotations-cycle r rnotid* **by** *auto*
have *r e* ≠ *a* **using** *b* ‹*a* = *b*› ‹*e* ≠ *a*› *simple-rotation-bij-corollary r* **by** *auto*
then have *r e* ≠ *e* **using** ‹*e* ≠ *c*› ‹*r c* = *e*› *r simple-rotation-bij-corollary* **by** *blast*
have *dist*:*distinct* [*a*,*c*,*d*,*e*] **using** ‹*c* ≠ *a*› ‹*d* ≠ *a*› ‹*d* ≠ *c*› ‹*e* ≠ *a*› ‹*e* ≠ *c*› ‹*e* ≠ *d*› **by** *simp*
then have ∀ *v* ∈ {*a*,*c*,*d*}. *r v* ≠ *d* **using** ‹*r b* = *b*› ‹*a* = *b*› ‹*r d* = *c*› ‹*r c* = *e*› **by** *auto*
then have *r e* = *d* **using** *simple-distinct-map r dist* **by** *auto*

have *dist′*:*distinct* [*a*,*c*,*e*,*d*] **using** *dist* **by** *auto*
have *s e* ≠ *e* **using** ‹*e* ≠ *a*› *a s-fix-forall* **by** *blast*
then have ∀ *v* ∈ {*a*,*c*,*e*}. *s v* ≠ *e* **using** ‹*s a* = *a*› ‹*s c* = *d*› *dist* **by** *auto*
then have *s d* = *e* **using** *distinct-map s dist′* **by** *auto*
then have ∀ *v* ∈ {*a*,*c*,*d*}. *s v* ≠ *c* **using** ‹*s a* = *a* › ‹*s c* = *d*› *dist* **by** *auto*
then have *s e* = *c* **using** *distinct-map s dist* **by** *auto*
then have (*r* ∘ *s*) *d* = *d* **using** ‹*s d* = *e*› ‹*r e* = *d*› **by** *auto*
then have (*r* ∘ *s*) *e* = *e* **using** ‹*s e* = *c*› ‹*r c* = *e*› **by** *auto*
then show (*r* ∘ *s*) = *id* **using** ‹(*r* ∘ *s*) *d* = *d*› ‹(*r* ∘ *s*) *a* = *a*› ‹(*r* ∘ *s*) *c* = *c*› *dist id-decomp* **by** *auto*
qed
qed
qed
qed
qed
qed
qed

This lemma is a simple corollary of the previous result. It is the main result necessary to count stabilisers.

corollary *complex-rotations-fix*: $r \in$ *complex-rotations* $\Longrightarrow r\ a = a \Longrightarrow r\ b = b \Longrightarrow a \neq b \Longrightarrow r = id$
using *complex-rotations-invariants* **by** *blast*

2.3 Inversions

In this section we show that inverses exist for each rotation, which we will need to show that the rotations we defined indeed form a group.

lemma *simple-rotations-rotate-id*:
assumes r:$r \in$ *simple-rotations*
shows $r \circ r \circ r = id$
using r **unfolding** *simple-rotations-def*
by *safe*
(*rule ext*; *rename-tac a*; *case-tac a*;
simp add: *simple-rotations*
)+

lemma *simple-rotations-inverses*:
assumes r:$r \in$ *simple-rotations*
shows $\exists\, y \in$*complex-rotations*. $y \circ r = id$
proof
let *?y* $= r \circ r$
from r **show** y:*?y* $\in$ *complex-rotations* **using** *complex-rotations.intros* **by** *simp*
from *simple-rotations-rotate-id* **show** *?y* $\circ\ r = id$ **using** r **by** *auto*
qed

lemma *complex-rotations-inverses*:
$r \in$ *complex-rotations* $\Longrightarrow \exists\, y \in$*complex-rotations*. $y \circ r = id$
proof (*induction r rule*: *complex-rotations.induct*)
case (*simp r*) **then show** *?case* **using** *simple-rotations-inverses* **by** *blast*
next
case (*comp r s*)
obtain r' **where** r'-*comp*:r'∈*complex-rotations* **and** r'-*inv*:$r' \circ r = id$
using *simple-rotations-inverses comp.hyps* **by** *auto*
obtain y **where** y-*comp*:y∈*complex-rotations* **and** y-*inv*:$y \circ s = id$ **using** *comp.IH* **by** *blast*
from *complex-rotations-comp* **have** yr':$y \circ r' \in$ *complex-rotations* **using** r'-*comp y-comp* **by** *simp*
have $r' \circ (r \circ s) = r' \circ r \circ s$ **using** *comp-assoc* **by** *metis*
then have $r' \circ (r \circ s) = s$ **using** r'-*inv* **by** *simp*

then have $y \circ r' \circ (r \circ s) = id$ **using** *y-inv comp-assoc* **by** *metis*
then show *?case* **using** yr' **by** *metis*
qed

2.4 The Tetrahedral Group

We can now define the group of rotational symmetries of a tetrahedron. Since we modeled rotations as functions, the group operation is functional composition and the identity element of the group is the identity function

definition *tetrahedral-group* :: *Rotation monoid* **where**
tetrahedral-group = (|*carrier* = *complex-rotations*, *mult* = (∘), *one* = *id*|)

We now prove that this indeed forms a group. Most of the subgoals are trivial, the last goal uses our results from the previous section about inverses.

lemma *is-tetrahedral-group*: *group tetrahedral-group*
proof(*rule groupI*)
show $\mathbf{1}_{tetrahedral\text{-}group} \in$ *carrier tetrahedral-group*
by (*simp add*: *complex-rotations.intros*(*1*) *simple-rotations-def tetrahedral-group-def*)
next
fix x
assume $x \in$ *carrier tetrahedral-group*
show $\mathbf{1}_{tetrahedral\text{-}group} \otimes_{tetrahedral\text{-}group} x = x$
unfolding *id-comp tetrahedral-group-def monoid.select-convs*(*1*) *monoid.select-convs*(*2*)
..
next
fix $x\ y\ z$
assume $x \in$ *carrier tetrahedral-group* **and**
$y \in$ *carrier tetrahedral-group* **and**
$z \in$ *carrier tetrahedral-group*
then show $x \otimes_{tetrahedral\text{-}group} y \otimes_{tetrahedral\text{-}group} z =$
$x \otimes_{tetrahedral\text{-}group} (y \otimes_{tetrahedral\text{-}group} z)$
unfolding *tetrahedral-group-def monoid.select-convs*(*1*) **by** *auto*
next
fix $x\ y$
assume $x \in$ *carrier tetrahedral-group* **and**
$y \in$ *carrier tetrahedral-group*
then show $x \otimes_{tetrahedral\text{-}group} y \in$ *carrier tetrahedral-group*
by (*simp add*: *complex-rotations.intros*(*2*) *tetrahedral-group-def complex-rotations-comp*)
next
fix x
assume $x \in$ *carrier tetrahedral-group*
then show $\exists\, y \in$ *carrier tetrahedral-group*.

y $\otimes_{tetrahedral\text{-}group}$ *x* = $\mathbf{1}_{tetrahedral\text{-}group}$
using *complex-rotations-inverses* **by** (*simp add*: *tetrahedral-group-def*)
qed

Having proved that our definition forms a group we can now instantiate our orbit-stabiliser locale. The group action is the application of a rotation.

fun *apply-rotation* :: *Rotation* ⇒ *Vertex* ⇒ *Vertex* **where** *apply-rotation r v* = *r v*

interpretation *tetrahedral*: *orbit-stabiliser tetrahedral-group apply-rotation* :: *Rotation* ⇒ *Vertex* ⇒ *Vertex*
proof *intro-locales*
show *Group.monoid tetrahedral-group* **using** *is-tetrahedral-group* **by** (*simp add*: *group.is-monoid*)
show *group-axioms tetrahedral-group* **using** *is-tetrahedral-group* **by** (*simp add*: *group-def*)
show *orbit-stabiliser-axioms tetrahedral-group apply-rotation*
proof
fix *x*
show *apply-rotation* $\mathbf{1}_{tetrahedral\text{-}group}$ *x* = *x* **by** (*simp add*: *tetrahedral-group-def*)
next
fix *g h x*
show *g* ∈ *carrier tetrahedral-group* ∧ *h* ∈ *carrier tetrahedral-group* ⟶ *apply-rotation g* (*apply-rotation h x*) = *apply-rotation* (*g* $\otimes_{tetrahedral\text{-}group}$ *h*) *x*
by (*simp add*: *tetrahedral-group-def*)
qed
qed

2.5 Counting Orbits

We now prove that there is an orbit for each vertex. That is, the group action is transitive.

lemma *orbit-is-transitive*: *tetrahedral.orbit A* = *vertices*
proof
show *tetrahedral.orbit A* ⊆ *vertices* **unfolding** *vertices-def* **using** *Vertex.exhaust* **by** *blast*
have *id* ∈ *complex-rotations* **using** *complex-rotations.intros simple-rotations-def* **by** *auto*
then have *id* ∈ *carrier tetrahedral-group*
unfolding *tetrahedral-group-def partial-object.select-convs*(*1*).
moreover have *apply-rotation id A* = *A* **by** *simp*
ultimately have *A*:*A* ∈ (*tetrahedral.orbit A*)

using *tetrahedral.orbit-def mem-Collect-eq* **by** *fastforce*

have *rotate-C* $\in$ *simple-rotations*
using *simple-rotations-def insert-subset subset-insertI* **by** *blast*
then have *rotate-C* $\in$ *complex-rotations* **using** *complex-rotations.intros*(*1*) **by** *simp*
then have *rotate-C* $\in$ *carrier tetrahedral-group*
unfolding *tetrahedral-group-def partial-object.select-convs*(*1*)**.**
moreover have *apply-rotation rotate-C A* = *B* **by** (*simp add*: *rotate-C-def*)
ultimately have *B*:*B* $\in$ (*tetrahedral.orbit A*)
using *tetrahedral.orbit-def mem-Collect-eq* **by** *fastforce*

have *rotate-D* $\in$ *simple-rotations*
using *simple-rotations-def insert-subset subset-insertI* **by** *blast*
then have *rotate-D* $\in$ *complex-rotations* **using** *complex-rotations.intros*(*1*) **by** *simp*
then have *rotate-D* $\in$ *carrier tetrahedral-group*
unfolding *tetrahedral-group-def partial-object.select-convs*(*1*)**.**
moreover have *apply-rotation rotate-D A* = *C* **by** (*simp add*: *rotate-D-def*)
ultimately have *C*:*C* $\in$ (*tetrahedral.orbit A*)
using *tetrahedral.orbit-def mem-Collect-eq* **by** *fastforce*

have *rotate-B* $\in$ *simple-rotations*
using *simple-rotations-def insert-subset subset-insertI* **by** *blast*
then have *rotate-B* $\in$ *complex-rotations* **using** *complex-rotations.intros*(*1*) **by** *simp*
then have *rotate-B* $\in$ *carrier tetrahedral-group*
unfolding *tetrahedral-group-def partial-object.select-convs*(*1*)**.**
moreover have *apply-rotation rotate-B A* = *D* **by** (*simp add*: *rotate-B-def*)
ultimately have *D*:*D* $\in$ (*tetrahedral.orbit A*)
using *tetrahedral.orbit-def mem-Collect-eq* **by** *fastforce*

from *A B C D* **show** *vertices* $\subseteq$ *tetrahedral.orbit A* **by** (*simp add*: *vertices-def subsetI*)
qed

It follows from the previous lemma, that the cardinality of the set of orbits for a particular vertex is 4.

lemma *card-orbit*: *card* (*tetrahedral.orbit A*) = *4*
proof −
from *card.empty card-insert-if* **have** *card vertices* = *4* **unfolding** *vertices-def* **by** *auto*
with *orbit-is-transitive* **show** *card* (*tetrahedral.orbit A*) = *4* **by** *simp*
qed

2.6 Counting Stabilisers

Each vertex has three elements in its stabiliser - the identity, a rotation around its axis by 120 degrees, and a rotation around its axis by 240 degrees. We will prove this next.

definition *stabiliser-A* :: *Rotation set* **where**
stabiliser-A = {*id*, *rotate-A*, *rotate-A* ∘ *rotate-A*}

This lemma shows that our conjectured stabiliser is correct.

```
lemma is-stabiliser: tetrahedral.stabiliser A = stabiliser-A
proof
  show stabiliser-A ⊆ tetrahedral.stabiliser A
  proof −
   have id ∈ complex-rotations using complex-rotations.intros simple-rotations-def
by auto
    then have id ∈ carrier tetrahedral-group
      unfolding tetrahedral-group-def partial-object.select-convs(1) by simp
    moreover have apply-rotation id A = A by simp
    ultimately have id:id ∈ (tetrahedral.stabiliser A)
      using tetrahedral.stabiliser-def mem-Collect-eq by fastforce

    have rotate-A ∈ simple-rotations
      using simple-rotations-def insert-subset subset-insertI by blast
   then have rotate-A ∈ complex-rotations using complex-rotations.intros(1)
by simp
    then have rotate-A ∈ carrier tetrahedral-group
      unfolding tetrahedral-group-def partial-object.select-convs(1) by simp
     moreover have apply-rotation rotate-A A = A by (simp add: ro-
tate-A-def)
    ultimately have A:rotate-A ∈ (tetrahedral.stabiliser A)
      using tetrahedral.stabiliser-def mem-Collect-eq by fastforce

    have rotate-A ∈ simple-rotations
      using simple-rotations-def insert-subset subset-insertI by blast
   then have rotate-A ∘ rotate-A ∈ complex-rotations using complex-rotations.intros
by simp
    then have rotate-A ∘ rotate-A ∈ carrier tetrahedral-group
      unfolding tetrahedral-group-def partial-object.select-convs(1) by simp
     moreover have apply-rotation (rotate-A ∘ rotate-A) A = A by (simp
add: rotate-A-def)
    ultimately have AA:(rotate-A ∘ rotate-A) ∈ (tetrahedral.stabiliser A)
      using tetrahedral.stabiliser-def mem-Collect-eq by fastforce

    from id A AA show stabiliser-A ⊆ tetrahedral.stabiliser A
```

by (*simp add*: *stabiliser-A-def subsetI*)
qed
show *tetrahedral.stabiliser A* ⊆ *stabiliser-A*
proof
fix *x*
assume *a*:*x* ∈ *tetrahedral.stabiliser A*
with *tetrahedral.stabiliser-def* **have** *apply-rotation x A* = *A* **by** *simp*
with *apply-rotation.simps* **have** *xA*:*x A* = *A* **by** *simp*
from *a* **have** *x* ∈ *carrier tetrahedral-group*
using *subgroup.mem-carrier*[*of tetrahedral.stabiliser A*] *tetrahedral.stabiliser-subgroup*
by *auto*
then have *xC*:*x* ∈ *complex-rotations*
unfolding *tetrahedral-group-def partial-object.select-convs*(*1*) **by** *simp*
have *x B* ≠ *A* **using** *xA xC rotation-bij-corollary* **by** *fastforce*
then have *x* ∈ *complex-rotations* ⟹ *x A* = *A* ⟹ *x* ∈ *stabiliser-A*
proof (*cases x B*, *simp*)
assume *x B* = *B*
then have *x* = *id* **using** *complex-rotations-fix xC xA* **by** *simp*
then show *?thesis* **using** *stabiliser-A-def* **by** *auto*
next
assume *x B* = *C*
then have *x* ≠ *id* **by** *auto*
then have *x D* ≠ *D* **using** *complex-rotations-fix xC xA* **by** *blast*
have *x D* ≠ *C* **using** *xC* ‹*x B* = *C*› *rotation-bij-corollary* **by** *fastforce*

have *x D* ≠ *A* **using** *xC xA rotation-bij-corollary* **by** *fastforce*
then have *x D* = *B* **using** ‹*x D* ≠ *C*› ‹*x D* ≠ *D*› *Vertex.exhaust* **by**
blast

have *x C* ≠ *A* **using** *xC xA rotation-bij-corollary* **by** *fastforce*
have *x C* ≠ *B* **using** *xC* ‹*x D* = *B*› *rotation-bij-corollary* **by** *fastforce*

have *x C* ≠ *C* **using** *complex-rotations-fix xC xA* ‹*x* ≠ *id*› **by** *blast*
then have *x C* = *D* **using** ‹*x C* ≠ *A*› ‹*x C* ≠ *B*› *Vertex.exhaust* **by**
blast

have ∀ *v*. *x v* = *rotate-A v*
using *xA* ‹*x B* = *C*› ‹*x D* = *B*› ‹*x C* = *D*› *Vertex.exhaust rotate-A-def*
Vertex.simps **by** *metis*
then have *x* = *rotate-A* **by** *auto*
then show *?thesis* **using** *stabiliser-A-def* **by** *auto*
next
assume *x B* = *D*
then have *x* ≠ *id* **by** *auto*

then have *x C* ≠ *C* **using** *complex-rotations-fix xC xA* **by** *blast*
have *x C* ≠ *D* **using** *xC* ‹*x B* = *D*› *rotation-bij-corollary* **by** *fastforce*

have *x C* ≠ *A* **using** *xC xA rotation-bij-corollary* **by** *fastforce*
then have *x C* = *B* **using** ‹*x C* ≠ *D*› ‹*x C* ≠ *C*› *Vertex.exhaust* **by** *blast*

have *x D* ≠ *A* **using** *xC xA rotation-bij-corollary* **by** *fastforce*
have *x D* ≠ *B* **using** *xC* ‹*x C* = *B*› *rotation-bij-corollary* **by** *fastforce*

have *x D* ≠ *D* **using** *complex-rotations-fix xC xA* ‹*x* ≠ *id*› **by** *blast*
then have *x D* = *C* **using** ‹*x D* ≠ *A*› ‹*x D* ≠ *B*› *Vertex.exhaust* **by** *blast*

have ∀ *v*. *x v* = (*rotate-A* ∘ *rotate-A*) *v*
using *xA* ‹*x B* = *D*› ‹*x C* = *B*› ‹*x D* = *C*› *Vertex.exhaust rotate-A-def Vertex.simps comp-apply* **by** *metis*
then have *x* = *rotate-A* ∘ *rotate-A* **by** *auto*
then show *?thesis* **using** *stabiliser-A-def* **by** *auto*
qed
then show *x* ∈ *stabiliser-A* **using** *xA xC* **by** *simp*
qed
qed

Using the previous result, we can now show that the cardinality of the stabiliser is 3.

lemma *card-stabiliser-help*: *card stabiliser-A* = *3*
proof −
have *idA*:*id* ≠ *rotate-A*
proof −
have *id B* = *B* **by** *simp*
moreover have *rotate-A B* = *C* **by** (*simp add*: *rotate-A-def*)
ultimately show *id* ≠ *rotate-A* **by** *force*
qed
have *idAA*:*id* ≠ *rotate-A* ∘ *rotate-A*
proof −
have *id B* = *B* **by** *simp*
moreover have (*rotate-A* ∘ *rotate-A*) *B* = *D* **by** (*simp add*: *rotate-A-def*)
ultimately show *id* ≠ *rotate-A* ∘ *rotate-A* **by** *force*
qed
have *AAA*:*rotate-A* ≠ *rotate-A* ∘ *rotate-A*
proof −
have *rotate-A B* = *C* **by** (*simp add*: *rotate-A-def*)
moreover have (*rotate-A* ∘ *rotate-A*) *B* = *D* **by** (*simp add*: *rotate-A-def*)

ultimately show *rotate-A* $\neq$ *rotate-A* $\circ$ *rotate-A* **by** *force*
qed
from *idA idAA AAA card.empty card-insert-if* **show**
(*card stabiliser-A*) = *3* **unfolding** *stabiliser-A-def* **by** *auto*
qed

lemma *card-stabiliser*: *card* (*tetrahedral.stabiliser A*) = *3*
using *is-stabiliser card-stabiliser-help* **by** *simp*

2.7 Proving Finiteness

In order to apply the orbit-stabiliser theorem, we need to prove that the set of rotations is finite. We first prove that the set of vertices is finite.

lemma *vertex-set*: (*UNIV*::*Vertex set*) = {*A*, *B*, *C*, *D*}
by(*auto*, *metis Vertex.exhaust*)

lemma *vertex-finite*: *finite* (*UNIV* :: *Vertex set*)
by (*simp add*: *vertex-set*)

Next we need instantiate Vertex as an element of the type class of finite sets in HOL/Finite_Set.thy. This will allow us to use the lemma that functions between finite sets are finite themselves.

instantiation *Vertex* :: *finite*
begin
instance proof
show *finite* (*UNIV* :: *Vertex set*) **by** (*simp add*: *vertex-set*)
qed

Now we can show that the set of rotations is finite.

lemma *finite-carrier*: *finite* (*carrier tetrahedral-group*)
proof −

have *finite* (*UNIV* :: (*Vertex* $\Rightarrow$ *Vertex*) *set*) **by** *simp*
moreover have *complex-rotations* $\subseteq$ (*UNIV* :: (*Vertex* $\Rightarrow$ *Vertex*) *set*) **by** *simp*
ultimately show *finite* (*carrier tetrahedral-group*) **using** *finite-subset top-greatest* **by** *blast*
qed

2.8 Order of the Group

We can now finally apply the orbit-stabiliser theorem. Since we have orbits of cardinality 4 and stabilisers of cardinality 3, the order of the tetrahedral group, and with it the number of rotational symmetries of the tetrahedron, is 12.

theorem *order tetrahedral-group* = *12*
proof –
  have *card* (*tetrahedral.orbit A*) ∗ *card* (*tetrahedral.stabiliser A*) = *12*
    using *card-stabiliser card-orbit* **by** *simp*
  with *tetrahedral.orbit-stabiliser*[*OF finite-carrier*]
  show *order tetrahedral-group* = *12* **by** *simp*
qed

end

end

References

[1] Proofwiki. Orbit-stabilizer theorem. https://proofwiki.org/wiki/Orbit-Stabilizer_Theorem, 2017. [Online; accessed 18-July-2017].

[2] Proofwiki. Stabilizer is subgroup. https://proofwiki.org/wiki/Stabilizer_is_Subgroup, 2017. [Online; accessed 18-July-2017].

[3] Proofwiki. Stabilizer is subgroup corollary 2. https://proofwiki.org/wiki/Stabilizer_is_Subgroup/Corollary_2, 2017. [Online; accessed 18-July-2017].

[4] Wikipedia. Group action. https://en.wikipedia.org/wiki/Group_action, 2017. [Online; accessed 18-July-2017].