Morley's Theorem*

Benjamin Puyobro

Université Paris Saclay, IRT SystemX, LMF, CNRS

April 2, 2025

Contents

*This work has been supported by the French government under the "France 2030" program, as part of the SystemX Technological Research Institute within the CVH project.

theory *Complex-Angles*

imports *Complex-Geometry.Elementary-Complex-Geometry*

begin

1 Introduction

Morley's trisector theorem states that in any triangle, the three points of intersection of the adjacent angle trisectors form an equilateral triangle, called the first Morley triangle or simply the Morley triangle. This theorem is listed in the 100 theorem list [3].

In this theory, we define some basics elements on complex geometry such as axial symmetry, rotations. We also define basic property of congruent triangles in the complex field following the model already presented in the afp [1] In addition we demonstrate sines law in the complex context.

Finally we prove the Morley theorem using Alain Connes's proof [2].

2 Angles between three complex

definition *angle-c-def*:‹*angle-c a b c* ≡ ∠ (*a*−*b*) (*c*−*b*)›

lemma *angle-c-commute-pi*:
  assumes ‹*angle-c a b c* = *pi*›
  shows *angle-c a b c* = *angle-c c b a*
  using *angle-c-def assms ang-vec-sym-pi* **by** *presburger*

lemma *angle-c-commute*:
  assumes ‹*angle-c a b c* ≠ *pi*›
  shows *angle-c a b c* = −*angle-c c b a*
  using *angle-c-def assms ang-vec-sym* **by** *presburger*

lemma *angle-c-sum*:
  assumes *1*:‹*a* − *b*≠*0*› **and** *2*:‹*c* − *b*≠ *0*› **and** *3*:‹*b*−*d* ≠ *0*›
  shows‹⌊*angle-c a b c* + *angle-c c b d*⌋ = *angle-c a b d*›
  using *angle-c-def canon-ang-sum* **by** *force*

lemma *collinear-angle*:‹*collinear a b c* ⟹ *a*≠*b* ⟹ *b*≠*c* ⟹ *c*≠*a* ⟹ *angle-c a b c* = *0* ∨ *angle-c a b c* = *pi*›
  unfolding *collinear-def* **unfolding** *angle-c-def*
  by (*metis ang-vec-def arg-div collinear-def collinear-sym2′ is-real-arg2 right-minus-eq*)

lemma *cdist-commute*:‹*cdist a b* = *cdist b a*›
  by (*simp add*: *norm-minus-commute*)

```
lemma angle-sum-triangle:
  assumes h:a ≠ b ∧ b ≠ c ∧ a ≠ c
  shows   ⌊∠ (c−a) (b−a) + ∠ (a−b) (c−b)  + ∠ (b−c) (a−c)⌋ = pi
proof −
  have f0:‹b−a ≠ 0› ‹c−a≠0› ‹b−c≠0› using h by auto
  then have ‹∠c (c−a) (b−a) = abs (Arg ((b−a)/(c−a)))›
    unfolding ang-vec-def ang-vec-c-def
    apply(subst arg-div[OF f0(1) f0(2), symmetric])
    by(simp)
  have ‹(b−a)/(c−a) = rcis (cmod ((b−a)/(c−a))) (Arg ((b−a)/(c−a)))›
    by (simp add: rcis-cmod-Arg)
  have ‹(b−a)/(c−a) ∗ (c−b)/(a−b) ∗ (a−c)/(b−c) = −1›
    using h
   by (smt (z3) divide-eq-0-iff divide-eq-minus-1-iff f0(1,2,3) minus-diff-eq mult-minus-right
        nonzero-eq-divide-eq times-divide-eq-left)
  then have ‹Arg ((b−a)/(c−a) ∗ (c−b)/(a−b) ∗ (a−c)/(b−c)) = pi›
    by (metis arg-cis cis-pi pi-canonical)
  then have ‹Arg ((b−a)/(c−a) ∗ (c−b)/(a−b) ∗ (a−c)/(b−c))
 = ⌊Arg ((b−a)/(c−a) ∗ (c−b)/(a−b)) + Arg ((a−c)/(b−c))⌋›
    using arg-mult f0
    by (metis (no-types, lifting) Arg-zero arg-mult divide-divide-eq-left′ pi-neq-zero
times-divide-eq-left
        times-divide-eq-right)
  then have ‹⌊Arg ((b−a)/(c−a) ∗ (c−b)/(a−b)) + Arg ((a−c)/(b−c))⌋
=⌊Arg ((b−a)/(c−a)) + Arg ((c−b)/(a−b)) + Arg ((a−c)/(b−c))⌋›
   by (metis (no-types, lifting) arg-mult canon-ang-arg canon-ang-sum divide-eq-0-iff
eq-iff-diff-eq-0 h
        no-zero-divisors times-divide-eq-right)
  then show ?thesis
    using
      ‹Arg ((b − a) / (c − a) ∗ (c − b) / (a − b) ∗ (a − c) / (b − c)) = ⌊Arg ((b
− a) / (c − a) ∗ (c − b) / (a − b)) + Arg ((a − c) / (b − c))⌋›
     ‹Arg ((b − a) / (c − a) ∗ (c − b) / (a − b) ∗ (a − c) / (b − c)) = pi› arg-div
h by force
qed

lemma angle-sum-triangle-c:
  assumes h:a ≠ b ∧ b ≠ c ∧ a ≠ c
  shows   ⌊angle-c c a b + angle-c a b c  + angle-c b c a⌋ = pi
  unfolding angle-c-def
  using h by (smt (z3) angle-sum-triangle h)


lemma angle-sum-triangle′:
  assumes h:a ≠ 0 ∧ b ≠ 0 ∧ c ≠ 0
  shows   ⌊∠ (−a) b + ∠ (−b) c  + ∠ (−c) a⌋ = pi
proof −
  have f0:‹a ≠ 0› ‹b≠0› ‹c≠0› using h by auto
  have ‹b/(−a) ∗ a/(−c) ∗ c/(−b) = −1›
```

using *h* **by**(*auto*)
then have *f10*:‹*Arg* (*b*/(−*a*) ∗ *a*/(−*c*) ∗ *c*/(−*b*)) = *pi*›
by (*metis arg-cis cis-pi pi-canonical*)
then have *f1*:‹*Arg* (*b*/(−*a*) ∗ *a*/(−*c*) ∗ *c*/(−*b*))
= ⌊*Arg* (*b*/(−*a*) ∗ *a*/(−*c*)) + *Arg* (*c*/(−*b*))⌋›
using *arg-mult f0*
by (*metis* ‹*b* / − *a* ∗ *a* / − *c* ∗ *c* / − *b* = − *1*› *divide-eq-0-iff divide-eq-minus-1-iff times-divide-eq-right*)
then have *f2*:‹⌊*Arg* (*b*/(−*a*) ∗ *a*/(−*c*)) + *Arg* (*c*/(−*b*))⌋
=⌊*Arg* (*b*/(−*a*)) + *Arg* (*a*/(−*c*)) + *Arg* (*c*/(−*b*))⌋›
by (*metis* (*no-types*, *lifting*) *arg-mult canon-ang-arg canon-ang-sum divide-eq-0-iff f0*(*1*,*2*,*3*)
neg-0-equal-iff-equal no-zero-divisors times-divide-eq-right)
then show *?thesis*
using *f1 f2 arg-div h*
by (*smt* (*verit*) *f10 add.left-commute ang-vec-def neg-equal-0-iff-equal*)
qed

lemma *ang-c-in*:‹*angle-c a b c* ∈ {−*pi*<..*pi*}›
unfolding *angle-c-def* **by**(*auto simp*: *canon-ang*)

lemma *angle-c-aac*:‹*angle-c a a c* = ⌊*Arg* (*c*−*a*)⌋›
by (*simp add*: *Arg-zero angle-c-def*)

lemma *angle-c-caa*:‹*angle-c c a a* = ⌊−*Arg* (*c*−*a*)⌋›
by (*simp add*: *Arg-zero angle-c-def*)

lemma *angle-c-neq0*:‹*angle-c p q r* ≠ *0* ⟹ *p*≠*r* ›
unfolding *angle-c-def*
by *force*

end
theory *Complex-Triangles-Definitions*

imports *Complex-Angles*

begin

3 Complex triangles

In this section we define triangles and derive some useful lemmas on congruent triangles, following the model [1]

locale *congruent-ctriangle* =
fixes *a1 b1 c1* :: *complex* **and** *a2 b2 c2* :: *complex*
assumes *sides'*: *cdist a1 b1* = *cdist a2 b2 cdist a1 c1* = *cdist a2 c2 cdist b1 c1* = *cdist b2 c2*
and *angles'*: *angle-c b1 a1 c1* = *angle-c b2 a2 c2* ∨ *angle-c b1 a1 c1* = − *angle-c b2 a2 c2*

angle-c a1 b1 c1 = *angle-c a2 b2 c2* ∨ *angle-c a1 b1 c1* = − *angle-c a2 b2 c2*
angle-c a1 c1 b1 = *angle-c a2 c2 b2* ∨ *angle-c a1 c1 b1* = − *angle-c a2 c2 b2*
begin

lemma *sides*:
cdist a1 b1 = *cdist a2 b2 cdist a1 c1* = *cdist a2 c2 cdist b1 c1* = *cdist b2 c2*
cdist b1 a1 = *cdist a2 b2 cdist c1 a1* = *cdist a2 c2 cdist c1 b1* = *cdist b2 c2*
cdist a1 b1 = *cdist b2 a2 cdist a1 c1* = *cdist c2 a2 cdist b1 c1* = *cdist c2 b2*
cdist b1 a1 = *cdist b2 a2 cdist c1 a1* = *cdist c2 a2 cdist c1 b1* = *cdist c2 b2*
using *sides′*
by (*auto simp add*: *norm-minus-commute*)

lemma *angles*:
angle-c b1 a1 c1 = *angle-c b2 a2 c2* ∨ *angle-c b1 a1 c1* = − *angle-c b2 a2 c2*
angle-c a1 b1 c1 = *angle-c a2 b2 c2* ∨ *angle-c a1 b1 c1* = − *angle-c a2 b2 c2*
angle-c a1 c1 b1 = *angle-c a2 c2 b2* ∨ *angle-c a1 c1 b1* = − *angle-c a2 c2 b2*
angle-c c1 a1 b1 = *angle-c b2 a2 c2* ∨ *angle-c c1 a1 b1* = − *angle-c b2 a2 c2*
angle-c c1 b1 a1 = *angle-c a2 b2 c2* ∨ *angle-c c1 b1 a1* = − *angle-c a2 b2 c2*
angle-c b1 c1 a1 = *angle-c a2 c2 b2* ∨ *angle-c b1 c1 a1* = − *angle-c a2 c2 b2*
angle-c b1 a1 c1 = *angle-c c2 a2 b2* ∨ *angle-c b1 a1 c1* = − *angle-c c2 a2 b2*
angle-c a1 b1 c1 = *angle-c c2 b2 a2* ∨ *angle-c a1 b1 c1* = − *angle-c c2 b2 a2*
angle-c a1 c1 b1 = *angle-c b2 c2 a2* ∨ *angle-c a1 c1 b1* = − *angle-c b2 c2 a2*
angle-c c1 a1 b1 = *angle-c c2 a2 b2* ∨ *angle-c c1 a1 b1* = − *angle-c c2 a2 b2*
angle-c c1 b1 a1 = *angle-c c2 b2 a2* ∨ *angle-c c1 b1 a1* = − *angle-c c2 b2 a2*
angle-c b1 c1 a1 = *angle-c b2 c2 a2* ∨ *angle-c b1 c1 a1* = − *angle-c b2 c2 a2*
using *angles′ angle-c-commute*
by(*simp-all add*: *angle-c-commute*)(*metis add.inverse-inverse angle-c-commute*)+

end

end
theory *Complex-Trigonometry*

imports *HOL−Analysis.Convex Complex-Angles Complex-Triangles-Definitions*

begin

4 Complex trigonometry

In this section we add some trigonometric results, especially the law of sines

lemma *ang-sin*:
shows‹*Im* ((*b*−*a*)∗*cnj*(*c*−*a*)) = *cmod* (*c*−*a*) ∗*cmod* (*b*−*a*) ∗ *sin* (*angle-c c a b*)›
proof −
have *f0*:‹*sin* (*Arg* (*cnj c* − *cnj a*)) = − *sin* (*Arg* (*c* − *a*))›
by (*metis arg-cnj-not-pi arg-cnj-pi complex-cnj-diff neg-0-equal-iff-equal sin-minus sin-pi*)

have *f1*:‹*cos* (*Arg* (*cnj c* − *cnj a*)) = *cos* (*Arg* (*c* − *a*))›
by (*metis arg-cnj-not-pi arg-cnj-pi complex-cnj-diff cos-minus*)
then have ‹*b*−*a* = *cmod* (*b*−*a*) ∗ *cis* (*Arg* (*b*−*a*)) ∧ *cnj* (*c*−*a*) = *cmod* (*c*−*a*) ∗ *cis* (*Arg* (*cnj* (*c*−*a*)))›
using *rcis-cmod-Arg rcis-def*
by (*metis complex-mod-cnj*)
then have ‹*Im*(*cis* (*Arg* (*b*−*a*)) ∗ *cis* (*Arg* (*cnj*(*c*−*a*)))) = *sin* (*angle-c c a b*)›
unfolding *ang-vec-def angle-c-def* **using** *f1 f0* **by**(*auto simp*:*cis.code sin-diff*)
then have ‹*Im* (*cmod* (*b*−*a*) ∗ *cis* (*Arg* (*b*−*a*)) ∗ *cmod* (*c*−*a*) ∗ *cis* (*Arg* (*cnj* (*c*−*a*)))) =
cmod (*c*−*a*) ∗*cmod* (*b*−*a*) ∗ *sin* (*angle-c c a b*) ›
by (*metis Im-complex-of-real*[*of cmod* (*c* − *a*)] *Im-complex-of-real*[*of cmod* (*b* − *a*)]
Im-mult-real[*of cor* (*cmod* (*c* − *a*))
cis (*Arg* (*cnj* (*c* − *a*))) ∗ (*cor* (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*)))]
Im-mult-real[*of cor* (*cmod* (*b* − *a*)) *cis* (*Arg* (*b* − *a*)) ∗ *cis* (*Arg* (*cnj* (*c* − *a*)))]
Re-complex-of-real[*of cmod* (*c* − *a*)] *Re-complex-of-real*[*of cmod* (*b* − *a*)]
ab-semigroup-mult-class.mult-ac(*1*)[*of cor* (*cmod* (*b* − *a*)) *cis* (*Arg* (*b* − *a*))
cis (*Arg* (*cnj* (*c* − *a*)))]
ab-semigroup-mult-class.mult-ac(*1*)[*of cis* (*Arg* (*cnj* (*c* − *a*)))
cor (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*)) *cor* (*cmod* (*c* − *a*))]
ab-semigroup-mult-class.mult-ac(*1*)[*of cmod* (*c* − *a*) *cmod* (*b* − *a*)
Im (*cis* (*Arg* (*b* − *a*)) ∗ *cis* (*Arg* (*cnj* (*c* − *a*))))]
mult.commute[*of cis* (*Arg* (*cnj* (*c* − *a*))) *cor* (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*))]
mult.commute[*of cis* (*Arg* (*cnj* (*c* − *a*))) ∗ (*cor* (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*)))
cor (*cmod* (*c* − *a*))]
mult.commute[*of cor* (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*)) ∗ *cor* (*cmod* (*c* − *a*))
cis (*Arg* (*cnj* (*c* − *a*)))])
then show ‹*Im* ((*b* − *a*) ∗ *cnj* (*c* − *a*)) = *cmod* (*c* − *a*) ∗ *cmod* (*b* − *a*) ∗ *sin* (*angle-c c a b*)›
by (*metis* ‹*b* − *a* = *cor* (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*)) ∧ *cnj* (*c* − *a*) = *cor* (*cmod* (*c* − *a*)) ∗ *cis* (*Arg* (*cnj* (*c* − *a*)))›
ab-semigroup-mult-class.mult-ac(*1*))
qed

lemma *ang-cos*:
shows‹*Re* ((*b*−*a*)∗*cnj*(*c*−*a*)) = *cmod* (*c*−*a*) ∗*cmod* (*b*−*a*) ∗ *cos* (*angle-c c a b*)›
proof −
have *f0*:‹*sin* (*Arg* (*cnj c* − *cnj a*)) = − *sin* (*Arg* (*c* − *a*))›
by (*metis arg-cnj-not-pi arg-cnj-pi complex-cnj-diff neg-0-equal-iff-equal sin-minus sin-pi*)
have *f1*:‹*cos* (*Arg* (*cnj c* − *cnj a*)) = *cos* (*Arg* (*c* − *a*))›
by (*metis arg-cnj-not-pi arg-cnj-pi complex-cnj-diff cos-minus*)
then have *f2*:‹*b*−*a* = *cmod* (*b*−*a*) ∗ *cis* (*Arg* (*b*−*a*)) ∧ *cnj* (*c*−*a*) = *cmod* (*c*−*a*) ∗ *cis* (*Arg* (*cnj* (*c*−*a*)))›

using *rcis-cmod-Arg rcis-def*
by (*metis complex-mod-cnj*)
then have ‹*Re*(*cis* (*Arg* (*b*−*a*)) ∗ *cis* (*Arg* (*cnj*(*c*−*a*)))) = *cos* (*angle-c c a b*)›
unfolding *ang-vec-def angle-c-def* **using** *f1 f0* **by**(*auto simp*:*cis.code cos-diff*)
then have ‹*Re* (*cmod* (*b*−*a*) ∗ *cmod* (*c*−*a*)∗ *cis* (*Arg* (*b*−*a*)) ∗ *cis* (*Arg* (*cnj* (*c*−*a*)))) =
Re (*cmod* (*c*−*a*) ∗*cmod* (*b*−*a*)) ∗ *cos* (*angle-c c a b*) ›
proof −
have *f1*: *cor* (*cmod* (*b* − *a*) ∗ *cmod* (*c* − *a*)) ∗ *cis* (*Arg* (*b* − *a*)) ∗ *cis* (*Arg* (*cnj* (*c* − *a*)))
= *cor* (*cmod* (*b* − *a*) ∗ *cmod* (*c* − *a*)) ∗ (*cis* (*Arg* (*b* − *a*)) ∗ *cis* (*Arg* (*cnj* (*c* − *a*))))
using *ab-semigroup-mult-class.mult-ac*(*1*) **by** *blast*
have *cmod* (*b* − *a*) ∗ *cmod* (*c* − *a*) = *cmod* (*c* − *a*) ∗ *cmod* (*b* − *a*)
by *argo*
then show *?thesis*
using *f1 Im-complex-of-real Re-mult-real* ‹*Re* (*cis* (*Arg* (*b* − *a*)) ∗ *cis* (*Arg* (*cnj* (*c* − *a*))))
= *cos* (*angle-c c a b*)› **by** *presburger*
qed
then show ‹*Re* ((*b*−*a*)∗*cnj*(*c*−*a*)) = *cmod* (*c*−*a*) ∗*cmod* (*b*−*a*) ∗ *cos* (*angle-c c a b*)›
by (*metis Re-complex-of-real f2 mult.assoc mult.commute of-real-mult*)

qed

lemma *law-of-cosines*′:
assumes *h*:‹*A*≠*C*› ‹*A*≠*B*›
shows ((*cdist B C*)^2 − (*cdist A C*)^2 − (*cdist A B*)^2) / (− *2*∗(*cdist A C*)∗(*cdist A B*)) = (*cos* (∠ (*C*−*A*) (*B*−*A*)))
using *law-of-cosines*[*of B C A*] *h* **by**(*auto simp*:*field-simps*)

lemma *law-of-cosines*′′:
shows (*cdist A C*)^2 = (*cdist B C*)^2 − (*cdist A B*)^2 + *2*∗(*cdist A C*)∗(*cdist A B*)∗(*cos* (∠ (*C*−*A*) (*B*−*A*)))
using *law-of-cosines*[*of B C A*] **by**(*auto simp*:*field-simps*)

lemma *law-of-cosines*′′′:
shows (*cdist A B*)^2 = (*cdist B C*)^2 − (*cdist A C*)^2 + *2*∗(*cdist A C*)∗(*cdist A B*)∗(*cos* (∠ (*C*−*A*) (*B*−*A*)))
using *law-of-cosines*[*of B C A*] **by**(*auto simp*:*field-simps*)

4.1 The law of sines

theorem *law-of-sines*:
assumes *h1*:‹*b* ≠ *a*› ‹*a* ≠ *c*› ‹*b* ≠ *c*›
shows *sin* (*angle-c a b c*) ∗ *cdist b c* = *sin* (*angle-c c a b*) ∗ *cdist a c* (**is** *?A* = *?B*)
proof −

```
{fix a b c ::complex
  have f0:‹sin (Arg (cnj c − cnj a)) = − sin (Arg ( c − a))›
  by (metis arg-cnj-not-pi arg-cnj-pi complex-cnj-diff neg-0-equal-iff-equal sin-minus
     sin-pi)
  have f1:‹cos (Arg (cnj c − cnj a)) = cos (Arg (c − a))›
   by (metis arg-cnj-not-pi arg-cnj-pi complex-cnj-diff cos-minus)
  assume ‹b−a ≠ 0› ‹c−a ≠ 0 ›
  then have f2: ‹b−a = cmod (b−a) ∗ cis (Arg (b−a)) ∧ cnj (c−a) = cmod
(c−a) ∗ cis (Arg (cnj (c−a)))›
   using rcis-cmod-Arg rcis-def
   by (metis complex-mod-cnj)
  then have ‹Im(cis (Arg (b−a)) ∗ cis (Arg (cnj(c−a)))) = sin (angle-c c a b)›
   unfolding ang-vec-def angle-c-def using f1 f0 by(auto simp:cis.code sin-diff)

  then have ‹Im (cmod (b−a) ∗ cis (Arg (b−a)) ∗ cmod (c−a) ∗ cis (Arg (cnj
(c−a)))) =
    cmod (c−a) ∗cmod (b−a) ∗ sin (angle-c c a b) ›
   by (metis Im-complex-of-real[of cmod (c − a)] Im-complex-of-real[of cmod (b
− a)]
      Im-mult-real[of cor (cmod (c − a))
       cis (Arg (cnj (c − a))) ∗ (cor (cmod (b − a)) ∗ cis (Arg (b − a)))]
      Im-mult-real[of cor (cmod (b − a)) cis (Arg (b − a)) ∗ cis (Arg (cnj (c −
a)))]
      Re-complex-of-real[of cmod (c − a)] Re-complex-of-real[of cmod (b − a)]
       ab-semigroup-mult-class.mult-ac(1)[of cor (cmod (b − a)) cis (Arg (b −
a))
       cis (Arg (cnj (c − a)))]
      ab-semigroup-mult-class.mult-ac(1)[of cis (Arg (cnj (c − a)))
       cor (cmod (b − a)) ∗ cis (Arg (b − a)) cor (cmod (c − a))]
      ab-semigroup-mult-class.mult-ac(1)[of cmod (c − a) cmod (b − a)
       Im (cis (Arg (b − a)) ∗ cis (Arg (cnj (c − a))))]
      mult.commute[of cis (Arg (cnj (c − a))) cor (cmod (b − a)) ∗ cis (Arg (b
− a))]
      mult.commute[of cis (Arg (cnj (c − a))) ∗ (cor (cmod (b − a)) ∗ cis (Arg
(b − a)))
       cor (cmod (c − a))]
      mult.commute[of cor (cmod (b − a)) ∗ cis (Arg (b − a)) ∗ cor (cmod (c
− a))
       cis (Arg (cnj (c − a)))]])
  then have ‹Im ((b−a)∗cnj(c−a)) = cmod (c−a) ∗cmod (b−a) ∗ sin (angle-c
c a b)›
   using ang-sin by presburger
 }note ang=this
 have i2:‹sin (angle-c c a b) = Im((b−a)∗cnj(c−a))/ (cmod(b−a)∗cmod(c−a))›
  using ang[of b a c] h1 by(auto)
 have ‹sin (angle-c a b c) = Im((c−b)∗cnj(a−b)) / (cmod (c−b)∗cmod (a−b))›
  using ang h1 by(auto)
 then have imp1:‹cmod (c−b) ∗ sin (angle-c a b c) = Im((c−b)∗cnj(a−b)) /
(cmod (a−b)) ›
```

by *auto*
from *i2* **have** *imp2*:‹*cmod* (*c*−*a*) ∗ *sin* (*angle-c c a b*) = *Im*((*b*−*a*)∗*cnj*(*c*−*a*))/ (*cmod*(*b*−*a*)) ›
by *auto*
show *?thesis* **using** *imp1 imp2* **by**(*auto simp*:*field-simps norm-minus-commute*)

qed

lemma *law-of-sines′*: **assumes** *h1*:‹*b* ≠ *a*› ‹*a* ≠ *c*› ‹*b* ≠ *c*›
shows *sin* (*angle-c a b c*) ∗ *cdist b a* = *sin* (*angle-c b c a*) ∗ *cdist a c*
proof −
{**fix** *a b c* ::*complex*
have *f0*:‹*sin* (*Arg* (*cnj c* − *cnj a*)) = − *sin* (*Arg* (*c* − *a*))›
by (*metis arg-cnj-not-pi arg-cnj-pi complex-cnj-diff neg-0-equal-iff-equal sin-minus sin-pi*)
have *f1*:‹*cos* (*Arg* (*cnj c* − *cnj a*)) = *cos* (*Arg* (*c* − *a*))›
by (*metis arg-cnj-not-pi arg-cnj-pi complex-cnj-diff cos-minus*)
assume ‹*b*−*a* ≠ *0*› ‹*c*−*a* ≠ *0* ›
then have *f2*:‹*b*−*a* = *cmod* (*b*−*a*) ∗ *cis* (*Arg* (*b*−*a*)) ∧ *cnj* (*c*−*a*) = *cmod* (*c*−*a*) ∗ *cis* (*Arg* (*cnj* (*c*−*a*)))›
using *rcis-cmod-Arg rcis-def*
by (*metis complex-mod-cnj*)
then have ‹*Im*(*cis* (*Arg* (*b*−*a*)) ∗ *cis* (*Arg* (*cnj*(*c*−*a*)))) = *sin* (*angle-c c a b*)›
unfolding *ang-vec-def angle-c-def* **using** *f0 f1* **by**(*auto simp*:*cis.code sin-diff*)

then have ‹*Im* (*cmod* (*b*−*a*) ∗ *cis* (*Arg* (*b*−*a*)) ∗ *cmod* (*c*−*a*) ∗ *cis* (*Arg* (*cnj* (*c*−*a*)))) =
cmod (*c*−*a*) ∗*cmod* (*b*−*a*) ∗ *sin* (*angle-c c a b*) ›
by (*metis Im-complex-of-real*[*of cmod* (*c* − *a*)] *Im-complex-of-real*[*of cmod* (*b* − *a*)]
Im-mult-real[*of cor* (*cmod* (*c* − *a*))
cis (*Arg* (*cnj* (*c* − *a*))) ∗ (*cor* (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*)))]
Im-mult-real[*of cor* (*cmod* (*b* − *a*)) *cis* (*Arg* (*b* − *a*)) ∗ *cis* (*Arg* (*cnj* (*c* − *a*)))]
Re-complex-of-real[*of cmod* (*c* − *a*)] *Re-complex-of-real*[*of cmod* (*b* − *a*)]
ab-semigroup-mult-class.mult-ac(*1*)[*of cor* (*cmod* (*b* − *a*)) *cis* (*Arg* (*b* − *a*))
cis (*Arg* (*cnj* (*c* − *a*)))]
ab-semigroup-mult-class.mult-ac(*1*)[*of cis* (*Arg* (*cnj* (*c* − *a*)))
cor (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*)) *cor* (*cmod* (*c* − *a*))]
ab-semigroup-mult-class.mult-ac(*1*)[*of cmod* (*c* − *a*) *cmod* (*b* − *a*)
Im (*cis* (*Arg* (*b* − *a*)) ∗ *cis* (*Arg* (*cnj* (*c* − *a*))))]
mult.commute[*of cis* (*Arg* (*cnj* (*c* − *a*))) *cor* (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*))]
mult.commute[*of cis* (*Arg* (*cnj* (*c* − *a*))) ∗ (*cor* (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*)))
cor (*cmod* (*c* − *a*))]
mult.commute[*of cor* (*cmod* (*b* − *a*)) ∗ *cis* (*Arg* (*b* − *a*)) ∗ *cor* (*cmod* (*c* − *a*))

cis (*Arg* (*cnj* (*c* − *a*)))])
then have ‹*Im* ((*b*−*a*)∗*cnj*(*c*−*a*)) = *cmod* (*c*−*a*) ∗*cmod* (*b*−*a*) ∗ *sin* (*angle-c c a b*)›
using *ang-sin* **by** *presburger*
}note *ang*=*this*
have *i2*:‹*sin* (*angle-c b c a*) = *Im*((*a*−*c*)∗*cnj*(*b* − *c*))/ (*cmod*(*b*−*c*)∗*cmod*(*a*−*c*))›
using *ang*[*of a c b*] *h1* **by**(*auto*)
have ‹*sin* (*angle-c a b c*) = *Im*((*c*−*b*)∗*cnj*(*a*−*b*)) / (*cmod* (*c*−*b*)∗*cmod* (*a*−*b*))›
using *ang h1* **by**(*auto*)
then have *imp1*:‹*cmod* (*a*−*b*) ∗ *sin* (*angle-c a b c*) = *Im*((*c*−*b*)∗*cnj*(*a*−*b*)) / (*cmod* (*c*−*b*)) ›
by *auto*
from *i2* **have** *imp2*:‹*cmod* (*a*−*c*) ∗ *sin* (*angle-c b c a*) = *Im*((*a*−*c*)∗*cnj*(*b*−*c*))/ (*cmod*(*b*−*c*)) ›
by *auto*
show *?thesis* **using** *imp1 imp2*
by (*metis cdist-commute h1*(*1*,*2*,*3*) *law-of-sines*)
qed

lemma *ang-pos-pos*:‹*q*≠*p* ⟹ *p*≠ *r* ⟹ *r* ≠ *q* ⟹ *angle-c q r p* ≥ *0* ⟹ *angle-c r p q* ≥*0*›
using *law-of-sines′*[*of r q p*]
by (*smt* (*verit*) *ang-vec-bounded angle-c-def cdist-def mult-neg-pos mult-nonneg-nonneg right-minus-eq sin-ge-zero sin-gt-zero sin-minus zero-less-norm-iff*)

lemma *cmod-pos*:‹*cmod a* ≥ *0*›
by *simp*

lemma *ang-neg-neg*:‹*q*≠*p* ⟹ *p*≠ *r* ⟹ *r* ≠ *q* ⟹ *angle-c q r p* < *0* ⟹ *angle-c r p q* < *0*›
proof −
assume ‹*q*≠*p*› ‹*p*≠ *r*› ‹*r* ≠ *q*› ‹*angle-c q r p* < *0*›
then have ‹*sin* (*angle-c q r p*) < *0*›
using *ang-c-in*
by (*metis ang-vec-def angle-c-def canon-ang*(*1*) *minus-less-iff neg-0-less-iff-less sin-gt-zero sin-minus*)
then have ‹*sin* (*angle-c q r p*) ∗ *cdist r q* < *0*›
using ‹*r* ≠ *q*› *mult-neg-pos* **by** *fastforce*
from *law-of-sines′*[*of r q p*] **have** ‹*sin* (*angle-c r p q*)<*0*›
by (*metis* ‹*p* ≠ *r*› ‹*q* ≠ *p*› ‹*r* ≠ *q*› ‹*sin* (*angle-c q r p*) ∗ *cdist r q* < *0*› *cdist-def linorder-not-less mult-less-0-iff norm-ge-zero*)
then show *?thesis*
using *ang-c-in*[*of r p q*]
by (*metis ang-vec-def angle-c-def canon-ang*(*2*) *linorder-not-less sin-ge-zero*)
qed

lemma *collinear-sin-neq-0*:

‹¬*collinear a2 b2 c2* ⟹ *sin* (*angle-c a2 c2 b2*) ≠ *0*›
unfolding *collinear-def angle-c-def*
by (*metis Im-complex-div-eq-0 ang-sin angle-c-def collinear-def collinear-sym1 collinear-sym2' mult-eq-0-iff*)

lemma *collinear-sin-neq-pi*:
‹¬*collinear a2 b2 c2* ⟹ *sin* (*angle-c a2 c2 b2*) ≠ *pi*›
unfolding *collinear-def angle-c-def*
by (*metis add-cancel-right-left dual-order.antisym dual-order.trans le-add-same-cancel1*

linordered-nonzero-semiring-class.zero-le-one one-add-one one-neq-zero pi-ge-two sin-le-one)

lemma *collinear-iff*:
assumes ‹*a*≠*b* ∧ *b*≠*c* ∧ *c*≠*a*›
shows ‹*collinear a b c* ⟷ (*angle-c a b c* = *pi* ∨ *angle-c a b c* = *0*)›
apply(*rule iffI*)
using *assms* **unfolding** *collinear-def* **using** *collinear-angle collinear-def* **apply**(*fastforce*)
by (*metis collinear-def collinear-sin-neq-0 collinear-sym1 sin-pi sin-zero*)

definition ‹*innerprod a b* ≡ *cnj a* ∗ *b*›

lemma *left-lin-innerprod*:‹*innerprod* (*x* + *y*) *z* = *innerprod x z* + *innerprod y z*›
unfolding *innerprod-def*
by (*simp add*: *mult.commute ring-class.ring-distribs*(*1*))

lemma *right-lin-innerprod*:‹*innerprod x* (*y*+*z*) = *innerprod x y* + *innerprod x z*›
unfolding *innerprod-def*
by (*simp add*: *ring-class.ring-distribs*(*1*))

lemma *leftlin-innerprod*:‹*innerprod x* (*t*∗*y*) = *t* ∗ *innerprod x y*›
unfolding *innerprod-def* **by**(*auto*)

lemma *rightsesqlin-innerprod*:‹*innerprod* (*t*∗*x*) (*y*) = *cnj t* ∗ *innerprod x y*›
unfolding *innerprod-def* **by**(*auto*)

lemma *norm-eq-csqrt-inner*:‹*norm x* = *csqrt* (*innerprod x x*)›
using *complex-mod-sqrt-Re-mult-cnj innerprod-def* **by** *force*

lemma *abs2-eq-inner*:‹*abs* (*innerprod x y*)^*2* = *innerprod x y* ∗ *cnj* (*innerprod x y*)›
unfolding *innerprod-def abs-complex-def* **apply**(*rule complex-eqI*)
by (*metis comp-apply complex-mult-cnj-cmod of-real-power*)+

lemma *complex-add-inner-cnj*:‹*t*∗*innerprod x y* + *cnj* (*t*∗*innerprod x y*) = *2*∗ *Re* (*t*∗*innerprod x y*)›
  using *complex-add-cnj* **by** *blast*

lemma *Re-innerprod-inner*:‹*Re* (*innerprod* (*a*−*b*) (*c*−*b*)) = (*a*−*b*)·(*c*−*b*)›
  unfolding *innerprod-def inner-complex-def* **by**(*auto simp*:*field-simps*)

lemma *angle-c-arccos-pos*:
  assumes *h*:‹*a*≠*b* ∧ *b*≠ *c*› ‹*angle-c a b c* ≥ *0*›
  shows ‹*angle-c a b c* = *arccos* ((*Re* (*innerprod* (*a*−*b*) (*c* − *b*)))/(*cmod*(*a*−*b*)∗*cmod*(*c*−*b*)))›
proof −
  have ‹*Re* ((*a*−*b*)∗*cnj*(*c*−*b*)) = (*Re* (*innerprod* (*a*−*b*) (*c* − *b*)))›
    unfolding *innerprod-def* **by**(*auto simp*:*field-simps*)
  then have ‹((*Re* (*innerprod* (*a*−*b*) (*c* − *b*)))/(*cmod*(*a*−*b*)∗*cmod*(*c*−*b*))) = *cos* (*angle-c a b c*)›
    by (*metis* (*no-types*, *lifting*) *ang-cos angle-c-commute cos-minus divisors-zero h*(*1*) *mult*.*commute nonzero-mult-div-cancel-left norm-eq-zero right-minus-eq*)
  then show *?thesis*
    using *ang-vec-bounded angle-c-def arccos-cos h*(*2*) **by** *presburger*
qed

lemma *angle-c-arccos-neg*:
  assumes *h*:‹*a*≠*b* ∧ *b*≠ *c*› ‹*angle-c a b c* ≤ *0* ›
  shows ‹− *angle-c a b c* = *arccos* ((*Re* (*innerprod* (*a*−*b*) (*c* − *b*)))/(*cmod*(*a*−*b*)∗*cmod*(*c*−*b*)))›
proof −
  have ‹*Re* ((*a*−*b*)∗*cnj*(*c*−*b*)) = (*Re* (*innerprod* (*a*−*b*) (*c* − *b*)))›
    unfolding *innerprod-def* **by**(*auto simp*:*field-simps*)
  then have ‹((*Re* (*innerprod* (*a*−*b*) (*c* − *b*)))/(*cmod*(*a*−*b*)∗*cmod*(*c*−*b*))) = *cos* (*angle-c a b c*)›
    by (*metis* (*no-types*, *lifting*) *ang-cos angle-c-commute cos-minus divisors-zero h*(*1*) *mult*.*commute nonzero-mult-div-cancel-left norm-eq-zero right-minus-eq*)
  then show *?thesis*
    using *ang-vec-bounded angle-c-def arccos-cos h*(*2*)
    by (*metis arccos-cos2 less-le-not-le*)
qed

end
theory *Third-Unity-Root*

imports *Complex-Angles*

begin

5 Third unity root

In this section we prove some basic properties of the third unity root j.

lemma *root-unity-3*: ‹(*z*::*complex*)^*3* − *1* = *0* ⟷ (*z* = *cis* (*2*∗*pi*/*3*) ∨ *z*=*1* ∨ *z*

```
= cis (4*pi/3))›
proof(rule iffI)
  have ‹cis (2 * pi * real xa / 3) ≠ 1 ⟹
        cis (2 * pi * real xa / 3) ≠ cis (2 * pi / 3)
  ⟹ xa < 3 ⟹ cis (2 * pi * real xa / 3) = cis (4 * pi / 3) ›
    for xa
  proof(rule ccontr)
    assume h:‹cis (2 * pi * real xa / 3) ≠ 1›
      ‹cis (2 * pi * real xa / 3) ≠ cis (2 * pi / 3)›
      ‹xa < 3› ‹cis (2 * pi * real xa / 3) ≠ cis (4 * pi / 3)›
    then have ‹xa = 1 ∨ xa = 0›
      using less-Suc-eq numeral-3-eq-3 by fastforce
    then show False
      using h(1) h(2) by auto
  qed
  then have f0:‹(λk. cis (2 * pi * real k / real 3)) ‘ {..<3} = {1, cis(2*pi/3),
cis (4*pi/3)}›
    unfolding image-def  by(auto simp: image-def intro:bexI[where x=0]
        bexI[where x=1] bexI[where x=2])
  have ‹{z. z^3 = 1} = {1, cis(2*pi/3), cis (4*pi/3)}›
    apply(insert bij-betw-roots-unity[of 3, simplified])
    apply(frule bij-betw-imp-surj-on)
    using f0 by auto
  then show ‹z ^ 3 − 1 = 0 ⟹ z = cis (2 * pi / 3) ∨ z = 1 ∨ z = cis (4 * pi
/ 3)›
    unfolding bij-betw-def inj-on-def by(auto)
  show ‹ z = cis (2 * pi / 3) ∨ z = 1 ∨ z = cis (4 * pi / 3) ⟹ z ^ 3 − 1 = 0
›
    using cis-2pi cis-multiple-2pi[of 2] by(auto simp:DeMoivre field-simps)
qed

definition root3 where ‹root3≡cis(2*pi/3)›

lemma root3-eq-0:‹1 + root3 + root3^2 = 0›
proof −
  have ‹(z::complex)^3 − 1 = (z − 1)*(1 + z + z^2)› for z
    by(auto simp:field-simps power2-eq-square power3-eq-cube)
  moreover have ‹root3^3 − 1 = 0›
    using root-unity-3 unfolding root3-def by blast
  moreover have ‹root3−1 ≠ 0›
    by(simp add:cis.code Complex-eq-1 cos-120 root3-def)
  ultimately show ?thesis
    by simp
qed

lemma root-unity-carac:
  assumes h1:‹a1*a2*a3 = j› ‹1−a1*a2 ≠ 0 ∧ 1−a2*a3≠0 ∧ 1−a1*a3 ≠ 0›
‹j=root3›
    and  R : ‹R=(a1 * b2 + b1) / (1 − a1 * a2)› (is ‹R=?R›)
```

```
    and P :‹P=(a2*b3 + b2)/(1−a2*a3)› (is ‹P=?P›)
    and Q:‹Q=(a3*b1 + b3)/(1−a3*a1)› (is ‹Q=?Q›)
  shows ‹(a1^2+a1+1)*b1 + a1^3*(a2^2+a2+1)*b2 +a1^3*a2^3*(a3^2+a3+1)*b3
=
      −j*a1^2*a2*(a1−j)*(a2−j)*(a3−j)*(?R +j*?P +j^2*?Q)›
proof −
  have simp-rules-for-eq:‹1−a1*a2 ≠ 0 ∧ 1−a2*a3≠0 ∧ 1−a1*a3 ≠ 0› ‹a1*a2*a3
= j›
    ‹1 + j +j^2 = 0› ‹j*j*j = 1› ‹(1−a1*a2)*a3 = (a3−j)› ‹(1−a2*a3)*a1 =
(a1−j)› ‹(1−a1*a3)*a2 = (a2−j)›
    using h1 root3-eq-0 root-unity-3 unfolding root3-def
   by(auto simp:power3-eq-cube mult.left-commute  mult.commute right-diff-distrib)

  have y:‹ (− j * a1^2 * a2 * ((1 − a2 * a3) * a1) * ((1 − a1 * a3) * a2) * ((1
− a1 * a2) * a3) *
   (a1 * b2) + − j * a1^2 * a2 * ((1 − a2 * a3) * a1) * ((1 − a1 * a3) * a2) *
((1 − a1 * a2) * a3) * b1) /
    (1 − a1 * a2) = (− j * a1^2 * a2 * ((1 − a3 * a2) * a1) * ((1 − a1 * a3) *
a2) * ( a3) * (a1 * b2)
    + − j * a1^2 * a2 * ((1 − a3 * a2) * a1) * ((1 − a1 * a3) * a2) * (a3) *
b1)›
    apply(subst add-divide-distrib)
    using simp-rules-for-eq(1)
    by (simp add: mult.commute)
  have y2:‹(− j * a1^2 * a2 * ((1 − a2 * a3) * a1) * ((1 − a1 * a3) * a2) * ((1
− a1 * a2) * a3) * (j * (a2 * b3)) +
    − j * a1^2 * a2 * ((1 − a2 * a3) * a1) * ((1 − a1 * a3) * a2) * ((1 − a1 *
a2) * a3) * (j * b2)) /
    (1 − a2 * a3) = (− j * a1^2 * a2 * (a1) * ((1 − a1 * a3) * a2) * ((1 − a1
* a2) * a3) * (j * (a2 * b3)) +
    − j * a1^2 * a2 * (a1) * ((1 − a1 * a3) * a2) * ((1 − a1 * a2) * a3) * (j *
b2))›
    apply(subst add-divide-distrib)
    using simp-rules-for-eq(1) by auto
  have y3:‹(− j * a1^2 * a2 * ((1 − a2 * a3) * a1) * ((1 − a1 * a3) * a2) * ((1
− a1 * a2) * a3) * (j^2 * (a3 * b1)) +
    − j * a1^2 * a2 * ((1 − a2 * a3) * a1) * ((1 − a1 * a3) * a2) * ((1 − a1 *
a2) * a3) * (j^2 * b3)) /
    (1 − a3 * a1) =
(− j * a1^2 * a2 * ((1 − a2 * a3) * a1) * (a2) * ((1 − a1 * a2) * a3) * (j^2 *
(a3 * b1)) +
     − j * a1^2 * a2 * ((1 − a2 * a3) * a1) * (a2) * ((1 − a1 * a2) * a3) * (j^2
* b3))›
    apply(subst add-divide-distrib)
   by (smt (verit, best) mult.commute nonzero-mult-div-cancel-left times-divide-eq-left
simp-rules-for-eq(1))
  have ko:‹a*(b*(c)) = a*b*c› for a b c::complex
    by auto
  have ko':‹a +(b +(c)) = a+b+c› for a b c::complex
```

```
  by auto
 have *:‹a1 * a1 * a1 * a1 * a1 * a2 * a2 * a2 * a2 * a3 * a3 * a3 * a3 * b1
= a1*a1*a2*a3*b1›
  using simp-rules-for-eq by (auto simp add: mult.assoc mult.left-commute)
 have **:‹a1 * a1 * a1 * a1 * a1 * a2 * a2 * a2 * a2 * a2 * a3 * a3 * a3 *
b3 = a1*a1*a2*a2*b3›
  using simp-rules-for-eq by (auto simp add: mult.assoc mult.left-commute)
 have ***:‹a1 * a1 * a1 * a1 * a1 * a1 * a2 * a2 * a2 * a2 * a2 * a3 * a3 *
a3 * a3 * b3
         = a1*a1*a1*a2*a2*a3*b3›
  using simp-rules-for-eq by (auto simp add: mult.assoc mult.left-commute)
 have fin:‹a1 * b1 + a1 * a1 * a1 * a2 * b2 + a1 * a1 * a1 * a1 * a2 * a2 *
a3 * b2 +
  a1 * a1 * a1 * a2 * a2 * a2 * a3 * b3 + a1 * a1 * a1 * a2 * a2 * a3 * a3
* b1 +
  a1 * a1 * a1 * a1 * a1 * a2 * a2 * a2 * a3 * a3 * b2 +a1 * a1 * a1 * a1 *
a2 * a2 * a2 * a2 *
  a3 * a3 * b3 + a1 * a1 * a2 * a2 * b3 + a1 * a1 * a2 * a3 * b1 = a1*b1
+ j*a1*b1 + j^2*a1*b1 +
  a1*a1*a1*a2*b2 + a1*a1*a1*a2*b2*j + a1*a1*a1*a2*b2*j^2 + a1*a1*a2*a2*b3
+ a1*a1*a2*a2*b3*j +
  a1*a1*a2*a2*b3*j^2›
  using simp-rules-for-eq by(auto simp:algebra-simps power2-eq-square power3-eq-cube)
 then have fin':‹... = (a1*b1 + a1*a1*a1*a2*b2 + a1*a1*a2*a2*b3)*(1+j+j^2)›
  by(auto simp:algebra-simps power2-eq-square power3-eq-cube)
 then show graal:‹(a1^2+a1+1)*b1 + a1^3*(a2^2+a2+1)*b2 +a1^3*a2^3*(a3^2+a3+1)*b3
=
   −j*a1^2*a2*(a1−j)*(a2−j)*(a3−j)*(?R +j*?P +j^2*?Q)›
  apply(simp only: simp-rules-for-eq(5−7)[symmetric])
  apply(simp only:y y2 y3 distrib-left distrib-right times-divide-eq-left times-divide-eq-right)

  apply(simp only:simp-rules-for-eq algebra-simps del:mult.assoc mult.commute)

  apply(simp only:ko ko')
  using simp-rules-for-eq apply(auto simp add: power2-eq-square power3-eq-cube
algebra-simps)
   apply(simp only:ko ko') apply(subst *) apply(subst **)  apply(subst ***)
apply(simp)
  apply(simp only:ko ko') apply(subst fin) apply(subst fin')
  using simp-rules-for-eq by fastforce
qed

end
theory Complex-Triangles

imports Complex-Trigonometry Third-Unity-Root

begin
```

```
lemma similar-triangles':
  assumes h:a ≠ 0 ∧ b ≠ 0 ∧ 0 ≠ c ∧ a' ≠ 0 ∧ b'≠0 ∧ c'≠0
    and h1:‹∠ (−a) b = ∠ (−a') b'› ‹∠ (−b') c' = ∠ (−b) c›
  shows ‹∠ (−c) a = ∠ (−c') a'›
proof −
  have ‹⌊∠ (−a) b + ∠ (−b) c + ∠ (−c) a⌋ = pi›
    using angle-sum-triangle' h by auto
  also have ‹⌊∠ (−a') b' + ∠ (−b') c' + ∠ (−c') a'⌋ = pi›
    using angle-sum-triangle' h by auto
  also have ‹⌊∠ (−a') b' + ∠ (−b') c' + ∠ (−c) a⌋ = pi›
    using ‹⌊∠ (− a) b + ∠ (− b) c + ∠ (− c) a⌋ = pi› h1(1,2) by auto
  then have ‹⌊∠ (− c) a⌋ = ⌊∠ (− c') a'⌋›
   by (smt (verit) ‹⌊∠ (− a') b' + ∠ (− b') c' + ∠ (− c') a'⌋ = pi› add.inverse-inverse
ang-vec-bounded
      ang-vec-opposite1 ang-vec-opposite2 ang-vec-opposite-opposite ang-vec-plus-pi1
ang-vec-plus-pi2
        canon-ang-diff canon-ang-id h neg-0-equal-iff-equal)
  ultimately show ?thesis
    by (metis ang-vec-bounded canon-ang-id)
qed


lemma similar-triangles:
  assumes h:a ≠ b ∧ b ≠ c ∧ a ≠ c ∧ a' ≠ b' ∧ b'≠ c' ∧ c'≠a'
    and h1:‹∠ (c−a) (b−a) = ∠ (c'−a') (b'−a')› ‹∠ (a−b) (c−b) = ∠ (a'−b')
(c'−b')›
  shows ‹ ∠ (b−c) (a−c) = ∠ (b'−c') (a'−c')›
proof −
  have ‹a−b≠0› ‹b−c≠0› ‹a−c≠0› ‹a'−b'≠0› ‹b'−c'≠0› ‹c'−a'≠0›
    using h by auto
  then show ?thesis
    by (smt (z3) diff-numeral-special(12) h1(1,2) minus-diff-eq similar-triangles')
qed

lemma similar-triangles-c:
  assumes h:a ≠ b ∧ b ≠ c ∧ a ≠ c ∧ a' ≠ b' ∧ b'≠ c' ∧ c'≠a'
    and h1:‹angle-c c a b = angle-c c' a' b'› ‹angle-c a b c = angle-c a' b' c'›
  shows ‹angle-c b c a = angle-c b' c' a'›
proof −
  have ‹a−b≠0› ‹b−c≠0› ‹a−c≠0› ‹a'−b'≠0› ‹b'−c'≠0› ‹c'−a'≠0›
    using h by auto
  then show ?thesis
    unfolding angle-c-def
    by (metis angle-c-def h h1(1,2) similar-triangles)
qed


lemmas congruent-ctriangleD = congruent-ctriangle.sides congruent-ctriangle.angles
```

lemma *congruent-ctriangles-sss*:
  assumes *h*:*a* ≠ *b* ∧ *b* ≠ *c* ∧ *a* ≠ *c*
   and *h1*:‹*cmod* (*b* − *a*)= *cmod* (*b*′−*a*′)› ‹*cmod* (*b* − *c*) = *cmod* (*b*′ − *c*′)› ‹*cmod* (*c* − *a*) = *cmod* (*c*′ − *a*′)›
  shows ‹*congruent-ctriangle a b c a*′ *b*′ *c*′›
proof −
  {fix *c b a c*′ *b*′ *a*′
   assume *h*:*a* ≠ *b* ∧ *b* ≠ *c* ∧ *a* ≠ *c*
    and *h1*:‹*cmod* (*b* − *a*)= *cmod* (*b*′−*a*′)› ‹*cmod* (*b* − *c*) = *cmod* (*b*′ − *c*′)› ‹*cmod* (*c* − *a*) = *cmod* (*c*′ − *a*′)›
   then have *h*′:‹*a*′≠*b*′ ∧ *b*′≠*c*′ ∧ *c*′≠*a*′›
    by *auto*
   have ‹*cos* (∠ (*c*−*a*) (*b*−*a*)) = ((*cdist b c*)^2 − (*cdist a c*)^2 − (*cdist a b*)^2) / (− *2*∗(*cdist a c*)∗(*cdist a b*)) ›
    using *law-of-cosines*′ *h* **by** *auto*
   moreover have ‹*cos* (∠ (*c*′−*a*′) (*b*′−*a*′)) = ((*cdist b*′ *c*′)^2 − (*cdist a*′ *c*′)^2 − (*cdist a*′ *b*′)^2) / (− *2*∗(*cdist a*′ *c*′)∗(*cdist a*′ *b*′)) ›
    using *law-of-cosines*′ *h h*′ **by** *auto*
   ultimately have *f0*:‹*cos* (∠ (*c*−*a*) (*b*−*a*)) = *cos* (∠ (*c*′−*a*′) (*b*′−*a*′))›
    by (*metis cdist-def h1*(*1*,*2*,*3*) *norm-minus-commute*)
   have ‹∠ (*c*−*a*) (*b*−*a*) ∈ {−*pi* .. *pi*}› ‹∠ (*c*′−*a*′) (*b*′−*a*′) ∈ {−*pi*..*pi*}›
    unfolding *ang-vec-def*
    using *canon-ang*(*1*,*2*) *less-eq-real-def* **by** *auto*
   with *f0* **have** ‹∠ (*c*−*a*) (*b*−*a*) = ∠ (*c*′−*a*′) (*b*′−*a*′) ∨ ∠ (*c*−*a*) (*b*−*a*) = − ∠ (*c*′−*a*′) (*b*′−*a*′)›
    unfolding *ang-vec-def*
   by (*smt* (*verit*) *arccos-cos2 arccos-minus-1 arccos-unique canon-ang*(*1*,*2*))**}note** *dem*=*this*
  then show *?thesis*
   by (*metis* (*mono-tags*, *lifting*) *angle-c-def cdist-def congruent-ctriangle-def h h1*(*1*) *h1*(*2*) *h1*(*3*) *norm-minus-commute*)
qed

lemma *congruent-ctriangleI-sss*:
  assumes *h*:*a* ≠ *b* ∧ *b* ≠ *c* ∧ *a* ≠ *c*
   and *h1*:‹*cdist a b* = *cdist a*′ *b*′› ‹*cdist b c* = *cdist b*′ *c*′› ‹*cdist a c* = *cdist a*′ *c*′›
  shows ‹*congruent-ctriangle a b c a*′ *b*′ *c*′›
  using *cdist-def congruent-ctriangles-sss h1*(*1*,*2*,*3*) *h*
  by (*metis dist-commute dist-complex-def*)

lemmas *congruent-ctriangle-sss* = *congruent-ctriangleD*[*OF congruent-ctriangleI-sss*]

lemma *isosceles-triangles*:
  assumes ‹*cdist a b* = *cdist b c*›
  shows ‹*angle-c b c a* = *angle-c b a c* ∨ *angle-c b c a* = − *angle-c b a c* ›
 by (*metis assms cdist-commute cdist-def congruent-ctriangle-sss*(*14*) *norm-eq-zero right-minus-eq*)

lemma *non-collinear-independant*:¬ *collinear a b c* ⟹ *a* ≠ *b* ∧ *b* ≠ *c* ∧ *a* ≠ *c*
  using *collinear-ex-real* **by** *force*

lemma *congruent-ctriangleI-sas*:
  assumes ‹*a1* ≠ *b1* ∧ *b1* ≠ *c1* ∧ *a1* ≠ *c1*›
  assumes *h1*:*cdist a1 b1* = *cdist a2 b2*
  assumes *h2*:*cdist b1 c1* = *cdist b2 c2*
  assumes *h3*:*angle-c a1 b1 c1* = *angle-c a2 b2 c2* ∨ *angle-c a1 b1 c1* = − *angle-c a2 b2 c2*
  shows *congruent-ctriangle a1 b1 c1 a2 b2 c2*
proof (*rule congruent-ctriangleI-sss*)
  have *h1′*:*cdist a1 b1* = *cdist b2 a2*
    using *cdist-commute h1* **by** *auto*
  show *cdist a1 c1* = *cdist a2 c2*
  proof (*rule power2-eq-imp-eq*)
    show $(cdist\ a1\ c1)^2 = (cdist\ a2\ c2)^2$
      apply(*insert law-of-cosines*[*of a1 c1 b1*] *law-of-cosines*[*of a2 c2 b2*])
      apply(*subst* (*asm*) (*1 2*) *h2*) **apply**(*subst* (*asm*) (*1 2*) *h1′*[*symmetric*])
      using *assms* **unfolding** *angle-c-def*
      by (*smt* (*verit*) *congruent-ctriangle-sss*(*7*) *angle-c-commute angle-c-def cos-minus*)
  qed *simp-all*
  show *a1* ≠ *b1* ∧ *b1* ≠ *c1* ∧ *a1* ≠ *c1*
    cdist a1 b1 = *cdist a2 b2*
    cdist b1 c1 = *cdist b2 c2* **using** *assms cdist-commute* **by** *auto*
qed

lemmas *congruent-ctriangle-sas* = *congruent-ctriangleD*[*OF congruent-ctriangleI-sas*]

lemma *congruent-ctriangleI-aas*:
  assumes *h1*:*angle-c a1 b1 c1* = *angle-c a2 b2 c2*
  assumes *h2*:*angle-c b1 c1 a1* = *angle-c b2 c2 a2*
  assumes *h3*:*cdist a1 b1* = *cdist a2 b2*
  assumes *h4*:¬*collinear a1 b1 c1* ¬*collinear a2 b2 c2*
  shows *congruent-ctriangle a1 b1 c1 a2 b2 c2*
proof (*rule congruent-ctriangleI-sas*)
  from *h4* **have** *neq*: *a1* ≠ *b1* **unfolding** *collinear-def* **by** *auto*
  with *assms*(*3*) **have** *neq′*: *a2* ≠ *b2* **by** *auto*
  have *h0*:‹*a1* ≠ *b1*› ‹*b1* ≠ *c1*› ‹*a1* ≠ *c1*›
    using *assms*(*4*) *non-collinear-independant* **by** *auto*
  have *h0′*:‹*a2* ≠ *b2*› ‹*b2* ≠ *c2*› ‹*a2* ≠ *c2*›
    using *assms*(*5*) *non-collinear-independant* **by** *auto*
  have *A*: *angle-c c1 a1 b1* = *angle-c c2 a2 b2* **using** *neq neq′ assms*
    apply(*insert angle-sum-triangle-c*[*of a1 b1 c1*] *angle-sum-triangle-c*[*of a2 b2*

```
c2])
    by (metis h0 assms(5) h1 h2 non-collinear-independant similar-triangles-c)
  have ‹−∠ (b1 − c1) (a1 − c1) = − ∠ (b2 − c2) (a2 − c2)›
    using h2 unfolding angle-c-def by auto
  then have A1:‹angle-c a1 c1 b1 = angle-c a2 c2 b2›
    using h2  unfolding angle-c-def
    apply(cases ‹angle-c a1 c1 b1 = pi›)
     apply (metis ang-vec-def angle-c-def canon-ang-uminus-pi minus-diff-eq)
    by (metis ang-vec-sym ang-vec-sym-pi)
  have ‹∠ (b1 − a1) (c1 − a1) = ∠ (b2 − a2) (c2 − a2)›
    by (metis ang-vec-sym ang-vec-sym-pi angle-c-def A)
  have sine1:‹ sin (angle-c a1 c1 b1) ∗ cdist c1 b1 = sin (angle-c b1 a1 c1) ∗ cdist
a1 b1›
    by (metis h0 law-of-sines)
  have sine2:‹sin (angle-c a2 c2 b2) ∗ cdist c2 b2 = sin (angle-c b2 a2 c2) ∗ cdist
a2 b2›
    by (metis assms(5) law-of-sines non-collinear-independant)
  have ‹¬collinear a2 b2 c2 ⟹ sin (angle-c a2 c2 b2) ≠ 0›
    unfolding collinear-def angle-c-def
    using angle-c-def collinear-sin-neq-0 h4(2) by presburger
  have h3':‹cdist b1 a1 = cdist b2 a2›
    using h3 cdist-commute by auto
  have A2:‹angle-c b1 a1 c1 = angle-c b2 a2 c2›
   using ‹∠ (b1 − a1) (c1 − a1) = ∠ (b2 − a2) (c2 − a2)› angle-c-def by auto
  have ‹ cdist c1 b1 = sin (angle-c b2 a2 c2) ∗ cdist a2 b2 / sin (angle-c a2 c2
b2) ∧
     cdist c2 b2 = sin (angle-c b2 a2 c2) ∗ cdist a2 b2 / sin (angle-c a2 c2 b2) ›
    apply(cases ‹ sin (angle-c a2 c2 b2) = 0›)
     apply (simp add:
       ‹¬ Elementary-Complex-Geometry.collinear a2 b2 c2 ⟹ sin (angle-c a2 c2
b2) ≠ 0›
       ‹¬ Elementary-Complex-Geometry.collinear a2 b2 c2›)
    by (metis A1 A2 h3 nonzero-mult-div-cancel-left sine1 sine2)
  then show cdist b1 c1 = cdist b2 c2
    using cdist-commute by auto
  show  a1 ≠ b1 ∧ b1 ≠ c1 ∧ a1 ≠ c1
    cdist a1 b1 = cdist a2 b2
    angle-c a1 b1 c1 = angle-c a2 b2 c2 ∨ angle-c a1 b1 c1 = − angle-c a2 b2 c2
    using assms h0 neq by auto
qed


lemmas congruent-ctriangle-aas = congruent-ctriangleD[OF congruent-ctriangleI-aas]

lemma congruent-ctriangleI-asa:
  assumes angle-c a1 b1 c1 = angle-c a2 b2 c2
  assumes cdist a1 b1 = cdist a2 b2
  assumes h0:angle-c b1 a1 c1 = angle-c b2 a2 c2
  assumes h4:¬collinear a1 b1 c1 ¬collinear a2 b2 c2
```

```
  shows   congruent-ctriangle a1 b1 c1 a2 b2 c2
proof (rule congruent-ctriangleI-aas)
  from assms have neq: a1 ≠ b1 a2 ≠ b2 unfolding collinear-def by auto
  show angle-c b1 c1 a1 = angle-c b2 c2 a2
    apply (rule similar-triangles-c)
    using assms(4,5) non-collinear-independant apply auto[1]
    using h0 unfolding angle-c-def
     apply (metis ang-vec-sym ang-vec-sym-pi)
    using angle-c-def assms(1) by auto
qed fact+


lemmas congruent-ctriangle-asa = congruent-ctriangleD[OF congruent-ctriangleI-asa]


lemma orientation-respect-letter-order:
  assumes angle-c a b c = angle-c b a c ¬ collinear a b c
  shows   False
  by (smt (verit, ccfv-threshold) angle-c-commute assms(1)
      assms(2) collinear-sin-neq-0 collinear-sym1 similar-triangles-c sin-pi sin-zero)


lemma isoscele-iff-pr-cis-qr:
  assumes h':‹q≠r›
  shows ‹(cdist q r = cdist p r) ⟷ (p−r) = cis(angle-c q r p)∗(q−r)›
proof (rule iffI)
  assume h:‹(cdist q r = cdist p r)›
  then have f0:‹cmod(p−r) = cmod(q−r)›
    by (simp add: norm-minus-commute)
  have ‹(p−r)/(q−r) = (p−r)∗cnj(q−r)/cmod(q−r)^2›
    using complex-div-cnj by blast
  have f1:‹Re ((p−r)∗cnj(q−r)) = cmod(p−r)∗cmod(q−r)∗cos(angle-c q r p)›
    by (metis (no-types, lifting)  ang-cos cdist-commute cdist-def h(1))
  have f2: ‹Im ((p−r)∗cnj(q−r)) = cmod(p−r)∗cmod(q−r)∗ sin(angle-c q r p)›
    using ang-sin h(1) by auto
 then have ‹ (p−r)∗cnj(q−r)/cmod(q−r)^2 = cmod(p−r)∗cmod(q−r)∗cis(angle-c
q r p)/cmod(q−r)^2›
    apply(intro complex-eqI)
    using f1 f2 by auto
 then have f3:‹(p−r)/(q−r) = cmod(p−r)∗cmod(q−r)∗cis(angle-c q r p)/cmod(q−r)^2›
    by (simp add: ‹(p − r) / (q − r) = (p − r) ∗ cnj (q − r) / cor ((cmod (q −
r))^2)›)
  have ‹(p−r)/(q−r) = cis (angle-c q r p)›
    apply(subst f3) apply(subst f0) by(simp add:h h' power2-eq-square)
  then show ‹p − r = cis (angle-c q r p) ∗ (q − r)›
    using divide-eq-eq h h' by force
next
  assume ‹p − r = cis (angle-c q r p) ∗ (q − r) ›
  have ‹(p−r)/(q−r) = (p−r)∗cnj(q−r)/cmod(q−r)^2›
```

```
    using complex-div-cnj by blast
  have f1:‹Re ((p−r)∗cnj(q−r)) = cmod(p−r)∗cmod(q−r)∗cos(angle-c q r p)›
    using ang-cos by fastforce
  have f2: ‹Im ((p−r)∗cnj(q−r)) = cmod(p−r)∗cmod(q−r)∗ sin(angle-c q r p)›
    using ang-sin by fastforce
 then have f4:‹ (p−r)∗cnj(q−r)/cmod(q−r)^2 = cmod(p−r)∗cmod(q−r)∗cis(angle-c
q r p)/cmod(q−r)^2›
    apply(intro complex-eqI)
    using f1 f2 by auto
 then have f3:‹(p−r)/(q−r) = cmod(p−r)∗cmod(q−r)∗cis(angle-c q r p)/cmod(q−r)^2›
    by (simp add: ‹(p − r) / (q − r) = (p − r) ∗ cnj (q − r) / cor ((cmod (q −
r))²)›)
  have ‹cmod(p−r)∗cmod(q−r)∗cis(angle-c q r p)/cmod(q−r)^2 = cis (angle-c q
r p)›
    using ‹p − r = cis (angle-c q r p) ∗ (q − r)› ‹q ≠ r› f3 by auto
  then have ‹cmod(p−r)∗cmod(q−r)/cmod(q−r)^2 = 1›
      by (metis f4 ‹q ≠ r› cmod-power2 complex-mod-cnj complex-mod-mult-cnj
complex-mult-cnj
      eq-iff-diff-eq-0 nonzero-norm-divide norm-cis norm-eq-zero norm-mult power-not-zero)
  then show ‹cdist q r = cdist p r›
   by (metis cdist-commute cdist-def eq-divide-eq-1 mult.commute power2-eq-square
real-divide-square-eq)
qed


lemma equilateral-imp-pi3:
  assumes ‹q≠r› cdist q r = cdist p r cdist p r = cdist p q
  shows ⌊(angle-c q r p)⌋ = pi/3 ∨ ⌊(angle-c q r p)⌋ = −pi/3
    (angle-c q r p) = (angle-c p q r) ∧ (angle-c q r p) = (angle-c r p q)
proof −
  have f0:‹q≠r ∧ r≠p ∧ p≠q› using assms by(auto)
  have ‹angle-c q r p ≠ 0 ∧ angle-c q r p ≠ pi›
     by (smt (verit, best) Re-complex-of-real angle-c-commute angle-c-commute-pi
assms(1) assms(2)
         assms(3) cdist-commute cdist-def congruent-ctriangle-sss(20) cor-cmod-real
isoscele-iff-pr-cis-qr
      minus-complex.simps(1) minus-complex.simps(2) minus-diff-eq mult-minus-right
norm-eq-zero right-minus-eq)
  then have f1:‹¬collinear q r p›
    using collinear-angle f0 by blast
  then have f1':‹¬collinear p q r›
    by (simp add: collinear-sym1 collinear-sym2)
  have f12:‹(angle-c q r p) = angle-c r p q ∨ (angle-c q r p) = − angle-c r p q›
    apply(rule congruent-ctriangle-sss)
    using f0 assms by(auto simp:norm-minus-commute)
  have f4:‹(angle-c q r p) = angle-c p q r ∨ (angle-c q r p) = − angle-c p q r›
    apply(rule congruent-ctriangle-sss)
    using f0 assms by(auto simp:norm-minus-commute)
  have f5:‹(angle-c q r p) ≠ pi›
```

using *f1*
by (*metis angle-c-commute canon-ang-sin canon-ang-uminus-pi collinear-sin-neq-0 f1' pi-canonical sin-pi*)
from *f4* **have** ‹ ⌊(*angle-c q r p*) + (*angle-c r p q*) + (*angle-c p q r*)⌋ = *pi*›
using *angle-sum-triangle-c*[*of p q r*] *f0* **by** *auto*
have ‹∀ *x*∈{−*pi*..*0*}. *3*∗*x* ≠ *pi*›
by (*metis atLeastAtMost-iff dual-order.trans mult-eq-0-iff mult-less-cancel-right2 numeral-le-one-iff pi-ge-zero pi-neq-zero semiring-norm*(*70*) *verit-comp-simplify1*(*3*) *verit-la-disequality*)
also have *the-x*:‹∃!*x*∈{−*pi*<..*pi*}. *3*∗*x* = *pi*›
by(*auto intro*:*exI*[**where** *x*=‹*pi*/*3*›])
moreover have ‹(*angle-c x y z*) ∈{−*pi*<..*pi*}› **for** *x y z*
using *ang-c-in* **by** *auto*
have ‹*x*∈{−*3*∗*pi*<..*3*∗*pi*} ∧ *x*−*2*∗*pi* = *pi* ⟹ (*x* = *3*∗*pi*)› **for** *x k*::*real*
by(*auto simp*:)
have ‹*x*∈{−*3*∗*pi*<..*3*∗*pi*} ∧ *x*−*2*∗*k*∗*pi* = *pi* ∧ (*k*≥*2* ∨ *k*≤ − *2*) ⟹ *False*› **for** *x* **and** *k*::*int*
proof −
assume *h*:‹*x*∈{−*3*∗*pi*<..*3*∗*pi*} ∧ *x*−*2*∗*k*∗*pi* = *pi* ∧ (*k*≥*2* ∨ *k*≤ −*2*)›
then have *f0*:‹ *r* = *2* ⟹ *x* − *2*∗ *r*∗*pi* ≤ −*pi*› **for** *r* **by** *auto*
also have *f1*:‹*k* > *2* ⟹ *x* − *2*∗ *k*∗*pi* < *x*−*2*∗*2*∗*pi*›
by(*auto*)
have *f3*:‹*x*−*2*∗*2*∗*pi* ≤ −*pi*›
using *f0*[*of 2*] **by**(*auto simp*:*abs-real-def*)
then have *f2*:‹ *k* > *2* ⟹ *x* − *2*∗ *k*∗*pi* < −*pi*›
using *f1* **by** *argo*
ultimately have *f3*:‹ *k* ≥*2* ⟹ *x*−*2*∗ *k*∗*pi* ≤ −*pi*›
apply(*cases* ‹*abs k*=*2*›) **using** *f0* **by**(*auto*)
have *f0'*:‹*r*=−*2* ⟹ *x* − *2*∗*r*∗*pi* > *pi*› **for** *r* ::*int*
using *h* **by**(*auto*)
have ‹*x*+*2*∗*2*∗*pi* > *pi*› **using** *h* **by** *auto*
also have ‹*k*< −*2* ⟹ *x* −*2*∗*k*∗*pi* > *x* − *2*∗(−*2*)∗*pi*›
using *f0'*[*of 2*]
proof −
assume *a1*: *k* < − *2*
have *f2*: *pi* < *x* − *real-of-int* (− *4*) ∗ *pi*
using *f0'* **by** *force*
have *k* + *k* ≤ − *4*
using *a1* **by** *force*
then show *?thesis*
using *f2* **by** (*metis* (*no-types*) *diff-left-mono h mult.commute mult-2-right of-int-le-iff ordered-comm-semiring-class.comm-mult-left-mono pi-ge-zero verit-comp-simplify1*(*3*))
qed
ultimately have *f4*:‹*k* ≤ −*2* ⟹ *x*−*2*∗ *k*∗*pi* > *pi*›
apply(*cases* ‹*k*=−*2*›) **using** *h*
‹*k* < − *2* ⟹ *x* − *2* ∗ − *2* ∗ *pi* < *x* − *real-of-int* (*2* ∗ *k*) ∗ *pi*› **by** *auto*
show *False*
using *f3 f4 h* **by** *force*

qed
then have *possible-k*:‹$x\in\{-3*pi<..3*pi\} \wedge x-2*k*pi = pi \Longrightarrow k=-1 \vee k=1 \vee k=0$› **for** *k*::*int* **and** *x*
by *force*
have *ang-3-in*:‹$y\in\{-3*pi<..3*pi\} \wedge \lfloor y\rfloor = pi \longleftrightarrow y=pi \vee y = 3*pi \vee y=-pi$› **for** *y*
proof(*rule iffI*)
assume *hyp*:‹$y \in \{- 3 * pi<..3 * pi\} \wedge \lfloor y\rfloor = pi$›
have ‹$\exists k.\ y - 2*k*pi = pi$›
by (*metis add-diff-cancel-left′ diff-add-cancel divide-self-if field-sum-of-halves mult.commute mult-1 mult-2*
pi-neq-zero times-divide-eq-right)
with *hyp* **show** ‹$y = pi \vee y=3*pi \vee y= -pi$›
using *hyp possible-k canon-ang-pi-3pi*[*of y*] *canon-ang-id canon-ang-uminus-pi*

by (*smt* (*verit*, *del-insts*) *canon-ang-minus-3pi-minus-pi greaterThanAtMost-iff*)
next
assume ‹$y = pi \vee y = 3 * pi \vee y = - pi$›
then show ‹$y \in \{- 3 * pi<..3 * pi\} \wedge \lfloor y\rfloor = pi$›
by(*auto simp*:*canon-ang-uminus-pi canon-ang-pi-3pi*)
qed
ultimately have ‹$\lfloor 3*(\text{angle-c } q\ r\ p)\rfloor = pi \Longrightarrow \text{angle-c } q\ r\ p = -pi/3 \vee \text{angle-c } q\ r\ p = pi/3$›
proof −
assume ‹$\lfloor 3*(\text{angle-c } q\ r\ p)\rfloor = pi$›
have ‹$3*(\text{angle-c } q\ r\ p) \in \{-3*pi<..3*pi\}$›
using *ang-c-in* **by** *auto*
then have *f0*:‹$3*(\text{angle-c } q\ r\ p) = -pi \vee 3*(\text{angle-c } q\ r\ p) = pi \vee 3*(\text{angle-c } q\ r\ p) = 3*pi$›
using *ang-3-in* ‹$\lfloor 3*(\text{angle-c } q\ r\ p)\rfloor = pi$› **by** *blast*
then have ‹$3*(\text{angle-c } q\ r\ p) = 3*pi \Longrightarrow False$›
using *collinear-sin-neq-pi* **by** (*simp add*: *f5*)
then show ‹$\text{angle-c } q\ r\ p = -pi/3 \vee \text{angle-c } q\ r\ p = pi/3$›
using *f0* **by** *auto*
qed
have *f2*:‹$\lfloor \text{angle-c } x\ y\ z\ \rfloor = \text{angle-c } x\ y\ z$› **for** *x y z*::*complex*
using *ang-vec-bounded angle-c-def canon-ang-id* **by** *fastforce*
then have ‹$\lfloor 3*(\text{angle-c } q\ r\ p)\rfloor = pi$›
apply (*cases* ‹$(\text{angle-c } q\ r\ p) = - \text{angle-c } r\ p\ q$›)
apply(*simp add*:*f1 f2*)
apply (*metis add.commute add.right-inverse add.right-neutral angle-sum-triangle-c canon-ang-uminus-pi f0 f2 f4 f5*)
by (*smt* (*verit*) ‹$\lfloor \text{angle-c } q\ r\ p + \text{angle-c } r\ p\ q + \text{angle-c } p\ q\ r\rfloor = pi$›
‹$\text{angle-c } q\ r\ p = \text{angle-c } p\ q\ r \vee \text{angle-c } q\ r\ p = - \text{angle-c } p\ q\ r$›
‹$\text{angle-c } q\ r\ p = \text{angle-c } r\ p\ q \vee \text{angle-c } q\ r\ p = - \text{angle-c } r\ p\ q$› *canon-ang-uminus-pi*
f2)
then have ‹$\text{angle-c } q\ r\ p = -pi/3 \vee \text{angle-c } q\ r\ p = pi/3$›
using ‹$\lfloor 3 * \text{angle-c } q\ r\ p\rfloor = pi \Longrightarrow \text{angle-c } q\ r\ p = - pi\ /\ 3 \vee \text{angle-c } q\ r\ p$

= *pi / 3*› **by** *auto*
then show ‹⌊*angle-c q r p*⌋ = *pi / 3* ∨ ⌊*angle-c q r p*⌋ = − *pi / 3*›
using *f2* **by** *auto*
then show ‹*angle-c q r p* = *angle-c p q r* ∧ *angle-c q r p* = *angle-c r p q*›
by (*smt* (*verit*, *del-insts*) ‹⌊*angle-c q r p* + *angle-c r p q* + *angle-c p q r*⌋ = *pi*›

f12 collinear-sin-neq-0 collinear-sym1 f1′ f2 f4 f5 sin-pi)
qed

lemma *isosceles-triangle-converse*:
assumes *angle-c a b c* = *angle-c c a b* ¬*collinear a b c*
shows *dist a c* = *dist b c*
by (*metis assms*(*1*) *assms*(*2*) *cdist-def collinear-sin-neq-0 collinear-sym1 congruent-ctriangle-asa*(*8*)
congruent-ctriangle-asa(*9*) *dist-complex-def law-of-sines mult-cancel-left non-collinear-independant*)

lemma *pi3-imp-equilateral*:
assumes ‹*q*≠*r*› ‹*p*≠*q*› ‹*r*≠*p*›
and ‹(*angle-c q p r*) = *pi*/*3* ∨ (*angle-c q p r*) = −*pi*/*3*›
and ‹(*angle-c q p r*) = (*angle-c r q p*)›
and ‹(*angle-c q p r*) = (*angle-c p r q*)›
shows ‹*cdist p r* = *cdist q r* ∧ *cdist p r* = *cdist p q*›
proof(*safe*)
have *f0*:‹¬ *collinear q p r*›
using *assms*(*1*−*3*) **unfolding** *collinear-def*
by (*smt* (*verit*, *ccfv-SIG*) *arccos-one-half arcsin-one-half arcsin-plus-arccos assms*(*4*)
collinear-angle collinear-def divide-le-eq-1-pos divide-pos-pos minus-divide-left pi-def pi-gt-zero pi-half)
then show ‹*cdist p r* = *cdist q r*›
using *isosceles-triangle-converse*[*of q p r*] *assms*
by (*simp add*: *dist-complex-def norm-minus-commute*)
show ‹*cdist p r* = *cdist p q* ›
using *f0 isosceles-triangle-converse*[*of p r q*] *assms*
by (*metis* ‹*cdist p r* = *cdist q r*› *cdist-def collinear-iff dist-commute dist-norm*)
qed

lemma *pi3-isoscele-imp-equilateral*:
assumes ‹*q*≠*r*› ‹*p*≠*q*› *cdist q r* = *cdist p r*
and ⌊(*angle-c q p r*)⌋ = *pi*/*3* ∨ ⌊(*angle-c q p r*)⌋ = −*pi*/*3*
shows ‹*cdist p q* = *cdist r q*›
proof(−)
have *f*:‹*angle-c p q r* = (*angle-c r p q*)›
by (*smt* (*verit*, *best*) *angle-c-commute angle-c-commute-pi assms*(*2*) *assms*(*3*) *cdist-commute*
collinear-angle isosceles-triangles orientation-respect-letter-order)
have *f0*:‹*r*≠*p*›

using *assms*(*1*) *assms*(*3*) **by** *force*
then have ‹*cdist q p* = *cdist r q*›
by (*smt* (*verit*) *congruent-ctriangle-sss*(*19*) *add-divide-distrib ang-vec-bounded angle-c-commute angle-c-def angle-sum-triangle-c arccos-minus-one-half arccos-one-half arcsin-minus-one-half*
arcsin-one-half arcsin-plus-arccos assms(*2*) *assms*(*3*) *assms*(*4*) *canon-ang-id canon-ang-minus-3pi-minus-pi*
cdist-commute field-sum-of-halves law-of-sines mult-cancel-right pi3-imp-equilateral sin-gt-zero sin-periodic-pi)
then show ‹*cdist p q* = *cdist r q*›
using *cdist-commute* **by** *force*
qed

lemma *pi3-isoscele-imp-equilateral*′:
assumes ‹*q*≠*r*› ‹*p*≠*q*› *cdist q r* = *cdist p r*
and ⌊(*angle-c q p r*)⌋ = *pi*/*3* ∨ ⌊(*angle-c q p r*)⌋ = −*pi*/*3*
shows ‹*cdist p r* = *cdist p q*›
by (*metis assms*(*2*) *assms*(*3*) *assms*(*4*) *cdist-commute pi3-isoscele-imp-equilateral*)

lemma *equilateral-caracterization*:‹*q*≠*r* ⟹ (*cdist q r* = *cdist p r* ∧ *cdist p r* = *cdist p q*)
⟷ ((*p*−*r*) = *cis*(*pi*/*3*)∗(*q*−*r*) ∨ (*p*−*r*) = *cis*(−*pi*/*3*)∗(*q*−*r*))›
proof(*rule iffI*)
assume *h*: ‹*q* ≠ *r*› ‹*cdist q r* = *cdist p r* ∧ *cdist p r* = *cdist p q* ›
then have *f1*:‹*p* − *r* = *cis* (*angle-c q r p*) ∗ (*q* − *r*) ∨ *p* − *r* = *cis* (− *angle-c q r p*) ∗ (*q* − *r*)›
using *isoscele-iff-pr-cis-qr* **by** *blast*
have *f0*:‹*q*≠*r* ∧ *r* ≠ *p* ∧ *p* ≠ *q*› **using** *h* **by** *auto*
have ‹*angle-c q r p* = *pi*/*3* ∨ *angle-c q r p* = −*pi*/*3*›
using *equilateral-imp-pi3*(*1*)[*of q r p*]
by (*metis ang-vec-bounded angle-c-def canon-ang-id h*(*1*) *h*(*2*))
then show ‹*p* − *r* = *cis* (*pi* / *3*) ∗ (*q* − *r*) ∨ *p* − *r* = *cis* (− *pi* / *3*) ∗ (*q* − *r*)›
using *f1*
by (*metis add.inverse-inverse minus-divide-left*)
next
assume *h*:‹*q* ≠ *r*› ‹*p* − *r* = *cis* (*pi* / *3*) ∗ (*q* − *r*) ∨ *p* − *r* = *cis* (− *pi* / *3*) ∗ (*q* − *r*)›
have ‹(*p*−*r*)/(*q*−*r*) = (*p*−*r*)∗*cnj*(*q*−*r*)/*cmod*(*q*−*r*)ˆ*2*›
using *complex-div-cnj* **by** *blast*
have *f1*:‹*Re* ((*p*−*r*)∗*cnj*(*q*−*r*)) = *cmod*(*p*−*r*)∗*cmod*(*q*−*r*)∗*cos*(*angle-c q r p*)›
using *ang-cos* **by** *force*
have *f2*: ‹*Im* ((*p*−*r*)∗*cnj*(*q*−*r*)) = *cmod*(*p*−*r*)∗*cmod*(*q*−*r*)∗ *sin*(*angle-c q r p*)›
using *ang-sin h*(*1*) **by** *auto*
then have ‹ (*p*−*r*)∗*cnj*(*q*−*r*)/*cmod*(*q*−*r*)ˆ*2* = *cmod*(*p*−*r*)∗*cmod*(*q*−*r*)∗*cis*(*angle-c q r p*)/*cmod*(*q*−*r*)ˆ*2*›
apply(*intro complex-eqI*)

using *f1 f2* **by** *auto*
then have *f3*:‹$(p-r)/(q-r) = cmod(p-r)*cmod(q-r)*cis(angle\text{-}c\ q\ r\ p)/cmod(q-r)$^*2*›
by (*simp add*: ‹$(p - r) / (q - r) = (p - r) * cnj (q - r) / cor ((cmod (q - r))^2)$›)
have ‹$(p-r)/(q-r) = cis (angle\text{-}c\ q\ r\ p)$›
apply(*subst f3*)
by (*smt* (*z3*) ‹$(p - r) * cnj (q - r) / cor ((cmod (q - r))^2) = cor (cmod (p - r) * cmod (q - r)) * cis (angle\text{-}c\ q\ r\ p) / cor ((cmod (q - r))^2)$› *cis-divide cis-mult cis-neq-zero cis-zero*
cmod-cor-divide complex-mod-cnj eq-iff-diff-eq-0 f3 h(*1*) *h*(*2*) *nonzero-mult-divide-mult-cancel-left*
nonzero-mult-divide-mult-cancel-right norm-cis norm-mult numeral-One of-real-1 of-real-divide
power2-less-0 times-divide-eq-left)
then have *fpi*:‹$angle\text{-}c\ q\ r\ p = pi/3 \lor angle\text{-}c\ q\ r\ p = -pi/3$›
using *h ang-c-in*
by (*smt* (*verit*, *best*) *add-divide-distrib ang-vec-bounded angle-c-def arccos-one-half*
arcsin-one-half arcsin-plus-arccos cis-inj diff-eq-diff-eq diff-numeral-special(*9*)
divide-nonneg-pos field-sum-of-halves nonzero-divide-eq-eq)
then have *ff*:‹$cdist\ p\ r = cdist\ q\ r$›
by (*metis* ‹$(p - r) / (q - r) = cis (angle\text{-}c\ q\ r\ p)$› *h*(*1*) *isoscele-iff-pr-cis-qr nonzero-divide-eq-eq right-minus-eq*)
then have ‹$angle\text{-}c\ p\ q\ r = angle\text{-}c\ q\ p\ r \lor angle\text{-}c\ p\ q\ r = -angle\text{-}c\ q\ p\ r$›
by (*metis congruent-ctriangle-sss*(*13*) *cdist-commute cdist-def norm-eq-zero right-minus-eq*)
then have *f10*:‹$\lfloor angle\text{-}c\ p\ q\ r + angle\text{-}c\ q\ r\ p + angle\text{-}c\ r\ p\ q\rfloor = pi$›
by (*metis* ‹$angle\text{-}c\ q\ r\ p = pi / 3 \lor angle\text{-}c\ q\ r\ p = - pi / 3$› *angle-c-neq0 angle-sum-triangle-c*
divide-eq-0-iff h(*1*) *neg-equal-0-iff-equal pi-neq-zero zero-neq-numeral*)
then have ‹$\lfloor angle\text{-}c\ q\ p\ r\rfloor = pi / 3 \lor \lfloor angle\text{-}c\ q\ p\ r\rfloor = - pi / 3$›
by (*smt* (*verit*) ‹$angle\text{-}c\ p\ q\ r = angle\text{-}c\ q\ p\ r \lor angle\text{-}c\ p\ q\ r = - angle\text{-}c\ q\ p\ r$›
add-divide-distrib ang-pos-pos ang-vec-bounded ang-vec-sym angle-c-def arccos-minus-one-half
arccos-one-half arcsin-minus-one-half arcsin-one-half arcsin-plus-arccos canon-ang-id
canon-ang-minus-3pi-minus-pi canon-ang-pi-3pi field-sum-of-halves fpi)
show ‹$cdist\ q\ r = cdist\ p\ r \land cdist\ p\ r = cdist\ p\ q$›
apply(*rule conjI*)
using *ff* **apply**(*auto simp*: *norm-minus-commute*)[*1*]
apply(*rule pi3-isoscele-imp-equilateral′*)
using *h*(*1*) **apply** *auto*[*1*]
apply (*metis* ‹$angle\text{-}c\ q\ r\ p = pi / 3 \lor angle\text{-}c\ q\ r\ p = - pi / 3$›
angle-c-neq0 divide-eq-0-iff neg-equal-0-iff-equal pi-neq-zero zero-neq-numeral)
using *ff* **apply** *presburger*
using ‹$\lfloor angle\text{-}c\ q\ p\ r\rfloor = pi / 3 \lor \lfloor angle\text{-}c\ q\ p\ r\rfloor = - pi / 3$› **by** *blast*

qed

lemmas *equilateral-imp-prcispi3* = *equilateral-caracterization*[*THEN iffD1*]

lemmas *prcispi3-imp-equilateral* = *equilateral-caracterization*[*THEN iffD2*]

```
lemma equilateral-caracterization-neg:
  fixes p q r::complex
  assumes h1:‹p≠r›
  shows ‹(cdist p r = cdist p q ∧ cdist p q = cdist q r ∧ angle-c q r p = −pi/3)
⟷ p + root3 ∗ q + root3 ^ 2 ∗ r = 0›
proof(rule iffI)
  assume h:‹cdist p r = cdist p q ∧ cdist p q = cdist q r ∧ angle-c q r p = −pi/3›
  then have ‹p−r = cis(−pi/3)∗(q−r)›
    using h1 isoscele-iff-pr-cis-qr[of q r p]
    by force
  with h have ‹p−r+cis(2∗pi/3)∗(q−r) = 0›
    apply(simp add:cis.code)
    apply(subst sin-120)
    apply(intro complex-eqI)
    using cos-120 cos-60 sin-120 sin-60
    by(auto simp add:field-simps)
  then have ‹p + root3∗q − (root3+1)∗r = 0› by(auto simp:field-simps root3-def)
  then show ‹p + root3 ∗ q + root3^2 ∗ r = 0›
    by (metis (no-types, lifting) add.commute eq-iff-diff-eq-0 left-diff-distrib
         ring-class.ring-distribs(2) root3-eq-0)
next
  assume h:‹p + root3 ∗ q + root3^2 ∗ r = 0›
  then have ‹p + root3∗q − (root3+1)∗r = 0›
   by (metis add.commute add-diff-cancel-left diff-numeral-special(9) left-diff-distrib
ring-class.ring-distribs(2)
        root3-eq-0)
  then have ‹p−r+root3∗(q−r) = 0›
    by(auto simp:field-simps)
  have ‹−root3 = cis(−pi/3)›
    using cos-120 cos-60 sin-120 sin-60
    by (auto intro:complex-eqI simp: root3-def)
  then have f1':‹p − r = cis(−pi/3)∗(q−r)›
    by (metis ‹p − r + root3 ∗ (q − r) = 0› eq-neg-iff-add-eq-0 mult-minus-left)
  then have f1:‹p − r = cis(pi/3)∗(q−r) ∨ p − r = cis(−pi/3)∗(q−r)›
    by auto
  then have f2:‹cmod (p−r) = cmod (q−r)›
    by(auto simp:norm-mult)
  then have f3:‹dist p r = dist q r›
    by (simp add: dist-complex-def)
  have ang-dem:‹angle-c q r p = −pi/3›
  proof −
    have ∗:‹q≠r›
      using f2 h1 by auto
```

then have ∗∗:‹(*p* − *r*) / (*q*−*r*) = *cis*(−*pi*/*3*)›
using *f1′* **by** *auto*
have ∗∗∗:‹ *angle-c q r p* = *Arg* ((*p* − *r*) /(*q*−*r*))›
by (*metis ang-vec-def angle-c-def arg-div f1′ h1 mult-zero-right right-minus-eq*)
then have ‹*Arg* ((*p* − *r*) /(*q*−*r*)) = ⌊−*pi*/*3*⌋›
by (*simp add*:*arg-cis* ∗∗)
then show *?thesis* **using** ∗ ∗∗ ∗∗∗
using *canon-ang-id pi-ge-two* **by** *force*
qed
have ‹*cmod* (*q* − *p*) = *cmod* (*r* − *q*)›
using *f2*
by (*metis cdist-def dist-eq-0-iff equilateral-caracterization f1′ f3*)
then show ‹*cdist p r* = *cdist p q* ∧ *cdist p q* = *cdist q r* ∧ *angle-c q r p* = −*pi*/*3*›

using *equilateral-caracterization*[*THEN iffD2*, *of q r p*, *OF* - *f1*]
using *f2 ang-dem* **by**(*auto simp*: *f2 field-simps intro*!:)
qed

end
theory *Complex-Axial-Symmetry*

imports *Complex-Angles Complex-Triangles*

begin

6 Axial symmetry in complex field

In the following we define the axial symmetry and prove basics properties.

context
fixes *z1 z2* :: *complex* **and** *α β* :: *complex*
assumes *neq0*:‹*z1*≠*z2*›
defines ‹*α* ≡ (*z1*−*z2*)/(*cnj z1* − *cnj z2*)›
defines ‹*β* ≡ (*z2*∗*cnj z1* − *z1*∗*cnj z2*)/(*cnj z1* − *cnj z2*)›
begin

definition *axial-symmetry*::‹*complex* ⇒ *complex*› **where**
‹*axial-symmetry z* ≡ *cnj z*∗(*z1* − *z2*)/(*cnj z1* − *cnj z2*) + (*z2*∗*cnj z1* − *z1*∗*cnj z2*)/(*cnj z1* − *cnj z2*)›

lemma *norm-α-eq-1*:‹*cmod* (*α*) = *1*›
by (*auto simp*: *neq0 α-def β-def*)

lemma *z1-inv*:‹*axial-symmetry z1* = *z1*›
unfolding *axial-symmetry-def*
by (*metis* (*mono-tags*, *lifting*) *add-diff-eq add-divide-distrib complex-cnj-cancel-iff*

diff-add-cancel eq-iff-diff-eq-0 mult.commute neq0 nonzero-eq-divide-eq right-diff-distrib′)

lemma *z2-inv*:‹*axial-symmetry z2* = *z2*›
unfolding *axial-symmetry-def*
by (*smt* (*z3*) *add-diff-cancel-left add-diff-cancel-left′ add-divide-distrib axial-symmetry-def*

complex-cnj-cancel-iff diff-add-cancel divide-divide-eq-right eq-divide-imp mult-commute-abs right-minus-eq z1-inv)

lemma *cmod-axial*:‹*cmod* (*axial-symmetry z* − *axial-symmetry z′*) = *cmod* (α∗(*cnj z* − *cnj z′*))›
unfolding *axial-symmetry-def*
by (*auto simp*: α*-def* β*-def*) (*simp add*: *diff-divide-distrib mult.commute right-diff-distrib′*)

lemma *cmod-axial-inv*:‹*cmod* (*axial-symmetry z* − *axial-symmetry z′*) = *cmod* (*z* − *z′*)›
by (*metis cmod-axial complex-cnj-diff complex-mod-cnj mult.commute mult.right-neutral norm-*α*-eq-1 norm-mult*)

lemma *axial-symmetry-dist1*:‹*cdist z1 z* = *cdist z1* (*axial-symmetry z*)›
by (*metis cdist-def cmod-axial-inv z1-inv*)

lemma *axial-symmetry-dist2*:‹*cdist z2 z* = *dist z2* (*axial-symmetry z*)›
by (*metis cdist-def cmod-axial-inv dist-commute dist-norm z2-inv*)

lemma αβ: ‹α∗*cnj* β + β = *0*›
by (*simp add*: *add-divide-eq-iff* α*-def* β*-def*)

lemma *involution-symmetry*:‹*axial-symmetry* (*axial-symmetry z*) = *z*›
proof −
have ‹α∗*cnj*(α∗*cnj z* + β) + β = α∗*cnj* α∗*z* + α∗*cnj* β +β›
by (*simp add*: *ring-class.ring-distribs*(*1*))
then show *?thesis* **unfolding** *axial-symmetry-def*
by (*smt* (*verit*, *best*) αβ *add.right-neutral cmod-eq-one group-cancel.add1 mult.commute mult-cancel-right2 norm-*α*-eq-1 times-divide-eq-left* α*-def* β*-def*)
qed

lemma *arg-*α:‹*Arg* α = ⌊*2*∗*Arg* (*z1*−*z2*)⌋›
unfolding α*-def* β*-def*
by (*smt* (*verit*, *del-insts*) *arg-cnj-not-pi arg-div arg-pi-iff canon-ang-id canon-ang-pi-3pi*

complex-cnj-cancel-iff complex-cnj-diff eq-cnj-iff-real eq-iff-diff-eq-0 neq0 pi-ge-two)

lemma *Arg-invol*:‹*Arg* (*axial-symmetry* (*axial-symmetry z*)) = *Arg z*›
by (*simp add*: *involution-symmetry*)

lemma *angle-sum-symmetry*:‹*z*≠*z1* ⟹ ⌊*angle-c z z1 z2* + *angle-c z2 z1* (*axial-symmetry*

z)⌋ = angle-c z z1 (axial-symmetry z)›
proof −
assume ‹z≠z1›
have ‹z2−z1 ≠ 0› **using** angle-c-sum neq0 **by** auto
have ‹z1 − (axial-symmetry z) ≠ 0›
by (metis ‹z ≠ z1› eq-iff-diff-eq-0 involution-symmetry z1-inv)
then show ?thesis
using angle-c-sum
by (metis ‹z2 − z1 ≠ 0› right-minus-eq z1-inv)
qed

lemma angle-symmetry-eq-imp:
assumes h:‹z1≠z› ‹z2≠z›
shows‹angle-c z z1 z2 = − angle-c (axial-symmetry z) z1 z2 ∨ angle-c z z1 z2 = angle-c (axial-symmetry z) z1 z2›
by (metis (mono-tags, lifting) axial-symmetry-dist1 cdist-def cmod-axial-inv congruent-ctriangle-sss(22) h(1) neq0 z2-inv)

lemma angle-symmetry:
assumes h:‹z1≠z› ‹z2≠z›
and ‹angle-c z z1 z2 = angle-c (axial-symmetry z) z1 z2›
shows ‹z = axial-symmetry z›
proof −
have ‹cmod (z − z1) = cmod (axial-symmetry z − z1)›
using axial-symmetry-dist1 **by** auto
have ‹cmod (z − z2) = cmod (axial-symmetry z − z2)›
by (metis axial-symmetry-dist2 cdist-def dist-commute dist-complex-def)
have ‹z−z1 = axial-symmetry z−z1›
by (metis ‹cmod (z − z1) = cmod (local.axial-symmetry z − z1)› ang-cos ang-sin assms(3)
complex-cnj-cnj complex-eq-iff eq-iff-diff-eq-0 mult-cancel-left neq0)
then show ?thesis
by simp
qed

lemma line-is-inv:‹z∈ line z1 z2 ∧ z≠z2 ∧ z≠z1 ⟹ z = axial-symmetry z›
proof −
assume ‹z∈ line z1 z2 ∧ z≠z2 ∧ z≠z1›
then have ‹angle-c z z1 z2 = 0 ∨ angle-c z z1 z2 = pi›
unfolding line-def **using** neq0 collinear-angle
using collinear-sym1 collinear-sym2′ **by** blast
then show ?thesis
by (smt (verit) ‹z ∈ line z1 z2 ∧ z ≠ z2 ∧ z ≠ z1›
angle-c-commute angle-c-commute-pi angle-symmetry angle-symmetry-eq-imp)
qed

lemma dist-inv:‹cdist a b = cdist (axial-symmetry a) (axial-symmetry b)›
by (simp add: cmod-axial-inv)

lemma *collinear-inv*: **assumes** ‹*collinear a b c*› **and** ‹$a \neq b \wedge b \neq c \wedge c \neq a$›
shows ‹*collinear* (*axial-symmetry a*) (*axial-symmetry b*) (*axial-symmetry c*)›
proof −
have ‹*angle-c a b c* = *pi* ∨ *angle-c a b c* = *0*›
using *assms*(*1*) *assms*(*2*) *collinear-angle* **by** *blast*
then have ‹*angle-c* (*axial-symmetry a*) (*axial-symmetry b*) (*axial-symmetry c*) = *angle-c a b c*
∨ *angle-c* (*axial-symmetry a*) (*axial-symmetry b*) (*axial-symmetry c*) = − *angle-c a b c*›
by (*metis congruent-ctriangle-sss*(*24*) *assms*(*2*) *dist-inv minus-equation-iff*)
then show *?thesis*
using ‹*angle-c a b c* = *pi* ∨ *angle-c a b c* = *0*› *collinear-sin-neq-0 collinear-sym1* **by** *fastforce*
qed

lemma *axial-symmetry-eq-line*:‹$z \neq z1 \wedge z \neq z2 \Longrightarrow z =$ *axial-symmetry z* $\Longrightarrow z \in$ *line z1 z2*›
proof −
assume ‹$z \neq z1 \wedge z \neq z2$› ‹*z* = *axial-symmetry z*›
then have *g0*:‹$z = \alpha * cnj\ z + \beta$› **unfolding** *axial-symmetry-def*
by (*simp add*: *mult.commute* α*-def* β*-def*)
then have *g1*:‹$cnj\ z = (z-\beta)*cnj\ \alpha$ ›
by (*smt* (*verit*, *ccfv-threshold*) *add-diff-cancel complex-cnj-cnj complex-cnj-diff* α*-def* β*-def*
complex-cnj-divide mult.commute neq0 nonzero-eq-divide-eq right-minus-eq times-divide-eq-left)
also have *g2*:‹$z = (cnj\ z - cnj\ \beta)*\alpha$›
by (*metis calculation complex-cnj-cnj complex-cnj-diff complex-cnj-mult*)
have ‹$\alpha/(z1-z2) = 1\ /(cnj\ z1 - cnj\ z2)$›
by (*auto simp*: α*-def* β*-def*)
have ‹$Im\ w = (w-cnj\ w)/(2*\mathrm{i})$› **for** *w*
using *Im-express-cnj* **by** *blast*
then have ‹$Im\ (z2*cnj\ z1) = (z2*cnj\ z1 - cnj\ (z2*cnj\ z1))/(2*\mathrm{i})$›
by *presburger*
then have ‹$Im\ (z2*cnj\ z1)*2*\mathrm{i} = (z2*cnj\ z1 - cnj\ z2*\ z1)$›
by (*metis complex-cnj-cnj complex-cnj-mult complex-diff-cnj mult.commute*)
have *f0*:‹$(cnj\ z1 - cnj\ z2)*(z1-z2) = cmod\ z1\hat{}2 + cmod\ z2\hat{}2 - cnj\ z1 * z2 - cnj\ z2 * z1$›
by (*smt* (*verit*) *cancel-ab-semigroup-add-class.diff-right-commute complex-mult-cnj-cmod*

diff-diff-eq2 left-diff-distrib mult.assoc mult.commute norm-minus-commute norm-mult of-real-add of-real-eq-iff)
have *f1*:‹$cmod\ z1\hat{}2 + cmod\ z2\hat{}2 - cnj\ z1 * z2 - cnj\ z2 * z1 = cmod\ z1\hat{}2 + cmod\ z2\hat{}2 - 2*Re\ z1*Re\ z2 - 2*Im\ z1*Im\ z2$›
by(*auto simp*:*field-simps intro*:*complex-eqI*)
have ‹$\beta = 2*\mathrm{i}*Im\ (z2*cnj\ z1)\ /\ (cnj\ z1-cnj\ z2)$›
using ‹$cor\ (Im\ (z2 * cnj\ z1)) = (z2 * cnj\ z1 - cnj\ (z2 * cnj\ z1))\ /\ (2 * \mathrm{i})$›
unfolding α*-def* β*-def* **by** *auto*
then have *f2*:‹$\beta/(z1-z2) = 2*\mathrm{i}*Im\ (z2*cnj\ z1)\ /\ ((cnj\ z1-cnj\ z2)*(z1-z2))$›

by *simp*
then have ‹*β*/(*z1*−*z2*) = *2*∗i∗*Im* (*z2*∗*cnj z1*) /(*cmod z1*^*2* + *cmod z2*^*2* − *2*∗*Re z1*∗*Re z2* − *2*∗*Im z1*∗*Im z2*)›
using *f0 f1* **by** *presburger*
then have ‹*is-real* ((*z* − *z1*)/(*z1*−*z2*))›
unfolding *axial-symmetry-def* **using** *g2*
by (*smt* (*verit*, *del-insts*) *add-diff-cancel-right axial-symmetry-def complex-cnj-cnj complex-cnj-diff complex-cnj-divide diff-divide-distrib divide-divide-eq-right eq-cnj-iff-real g0 g2 mult.commute times-divide-eq-right z1-inv z2-inv α-def β-def*)
then show ‹*z* ∈ *line z1 z2*›
unfolding *line-def collinear-def*
by (*metis* (*mono-tags*, *lifting*) *Im-i-times Re-i-times cnj.simps*(*2*) *complex-i-mult-minus eq-cnj-iff-real mem-Collect-eq minus-diff-eq minus-divide-right*)
qed

lemma *angle-symmetry-eq*:
assumes *h*:‹*z1*≠*z*› ‹*z2*≠*z*› ‹*z*∉*line z1 z2*›
shows‹*angle-c z z1 z2* = − *angle-c* (*axial-symmetry z*) *z1 z2*›
proof −
have *f0*:‹*angle-c z z1 z2* = − *angle-c* (*axial-symmetry z*) *z1 z2* ∨ *angle-c z z1 z2* = *angle-c* (*axial-symmetry z*) *z1 z2*›
using *angle-symmetry-eq-imp h*(*1*) *h*(*2*) **by** *blast*
have *f1*:‹*angle-c z z2 z1* = − *angle-c* (*axial-symmetry z*) *z2 z1* ∨ *angle-c z z2 z1* = *angle-c* (*axial-symmetry z*) *z2 z1*›
by (*metis axial-symmetry-dist1 congruent-ctriangle-sss*(*24*) *dist-inv h*(*1*) *neq0 z2-inv*)
show *?thesis*
using *angle-symmetry axial-symmetry-eq-line f0 h*(*1*) *h*(*2*) *h*(*3*) **by** *presburger*
qed

end
end
theory *Morley*

imports *Complex-Axial-Symmetry*

begin

7 Rotations

locale *complex-rotation* =
fixes *A*::*complex* **and** *ϑ*::*real*
begin

definition ‹*r z* = *A* + (*z*−*A*)∗*cis*(*ϑ*)›

```
lemma cmod-inv-rotation:‹cmod (z−A) = cmod (r z − A)›
  unfolding r-def
  by (simp add: norm-mult)

lemma inner-ang:‹cos (∠ z1 z2)∗(cmod z1 ∗cmod z2) = Re (innerprod z1 z2)›
proof −
  have ‹Re (innerprod z1 z2) = Re (scalprod z1 z2)›
    unfolding innerprod-def by(auto)
  then show ?thesis
    by (metis cos-cmod-scalprod mult.commute)
qed

lemma ang-eq-cos-theta:‹z≠A ⟹ cos (angle-c z A (r z)) = cos (ϑ)›
proof −
  assume ‹z≠A›
  then have ‹innerprod (z−A) ((r z − A)) = (z−A)∗cis(ϑ)∗cnj(z − A)›
    unfolding innerprod-def r-def by auto
  then have f0:‹innerprod (z−A) ((r z − A)) = cmod (z−A)^2∗cis(ϑ)›
   by (metis ab-semigroup-mult-class.mult-ac(1) complex-mult-cnj-cmod mult.commute)
  then have ‹cos (angle-c z A (r z))∗cmod (z−A)∗cmod(z − A) = Re (innerprod
(z−A) ((r z − A)))›
    unfolding angle-c-def
    by (metis inner-ang mult.assoc cmod-inv-rotation)
  then have ‹cos (angle-c z A (r z)) = Re (cis ϑ)›
    by (simp add: ‹z ≠ A› f0 power2-eq-square)
  then show ?thesis
    by(auto simp:cis.code)
qed

lemma cdist-dist:‹cdist = dist›
  using cdist-commute dist-complex-def by fastforce

lemma ang-eq-theta:assumes h:‹z≠A› shows ‹angle-c z A (r z) = ⌊ϑ⌋›
proof(cases ‹angle-c z A (r z) = ⌊− ϑ⌋›)
  case True
  then have ‹r z = A + (z−A)∗cis(−ϑ)›
   by (metis add-diff-cancel-left′ arg-cis cdist-def cis-cmod cis-neq-zero cmod-inv-rotation

          isoscele-iff-pr-cis-qr mult.left-commute mult.right-neutral mult-cancel-left
norm-cis
        norm-minus-commute of-real-1 r-def right-minus-eq)
  then show ?thesis
   by (smt (verit, del-insts) True add-diff-cancel-left′ angle-c-neq0 arg-cis arg-mult-eq
          canon-ang-cos canon-ang-sin cis.code cis-cnj diff-add-cancel divisors-zero
r-def)
next
  case False
  have ‹cos (angle-c z A (r z)) = cos (ϑ)›
    using ang-eq-cos-theta h by auto
```

then show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *cmod-inv-rotation r-def add-diff-cancel-left′ angle-c-neq0*
angle-c-sum arg-cis canon-ang-sin cdist-def cis.code cis-neq-zero divisors-zero eq-iff-diff-eq-0 h
isoscele-iff-pr-cis-qr nonzero-mult-div-cancel-left nonzero-mult-div-cancel-right norm-minus-commute)
qed

lemma *inj-r*:‹*inj r*›
unfolding *inj-on-def* **by**(*auto simp*:*r-def*)

lemma *img-eqI*:‹*cdist A z1* = *cdist A z2* ∧ *angle-c z1 A z2* = *ϑ* ⟹ *z2* = *r z1*›
apply(*cases* ‹*z1* = *A* ∨ *z2* = *A*›)
using *r-def add-minus-cancel isoscele-iff-pr-cis-qr* **apply** *force*
unfolding *r-def*
by (*metis add.commute cdist-commute diff-add-cancel isoscele-iff-pr-cis-qr mult.commute*)

lemma *r-id-iff*:‹⌊*ϑ*⌋ = *0* ⟷ *r* = *id*›
proof −
obtain *cc* :: (*complex* ⇒ *complex*) ⇒ *complex*
and *cca* :: (*complex* ⇒ *complex*) ⇒ *complex* **where**
f1: ∀ *f*. (*id* = *f* ∨ *f* (*cc f*) ≠ *cc f*) ∧ ((∀ *c*. *f c* = *c*) ∨ *id* ≠ *f*)
by (*metis* (*no-types*) *eq-id-iff*)
have *f2*: ∀ *c ca ra*. *ca* + (*c* − *ca*) ∗ *Complex* (*cos ra*) (*sin ra*) = *complex-rotation.r ca ra c*
by (*simp add*: *complex-rotation.r-def cis.code*)
then have ∀ *c ca*. *complex-rotation.r c 0 ca* = *ca*
by (*metis* (*no-types*) *add-diff-cancel-left′ cos-zero diff-diff-eq2*
lambda-one mult.commute one-complex.code sin-zero)
then show *?thesis*
using *f2 f1*
by (*metis* (*lifting*, *full-types*) *ang-eq-theta angle-c-neq0*
canon-ang-cos canon-ang-sin complex-i-not-zero)
qed

end

lemma *axial-symmetry-eq*:‹*axial-symmetry B C P* = *axial-symmetry C B P*› **if** ‹*C*≠*B*› **for** *C B P*
unfolding *axial-symmetry-def*[*OF that*] *axial-symmetry-def*[*OF that*[*symmetric*]]

by (*metis* (*no-types*, *lifting*) *complex-cnj-cancel-iff eq-iff-diff-eq-0*
minus-diff-eq nonzero-minus-divide-divide times-divide-eq-right)

lemma *img-r-sym*:
assumes *h*:‹*z1* ≠ *z2*› ‹*z* ∉ *line z1 z2*›
shows ‹*axial-symmetry z1 z2 z* = *complex-rotation.r z1* (⌊*2*∗*angle-c z z1 z2*⌋) *z*›

proof −
  interpret *complex-rotation z1* ⌊*2*∗*angle-c z z1 z2*⌋ **.**
  let *?z* = ‹*axial-symmetry z1 z2 z*›
  from *h* **have** ‹*z1*≠*z*› ‹*z2* ≠ *z*›
    unfolding *line-def Elementary-Complex-Geometry.collinear-def* **by**(*auto*)
  then have ‹*angle-c ?z z1 z2* = −*angle-c z z1 z2*›
    using *angle-symmetry-eq h*(*1*) *h*(*2*) **by** *force*
  then have ‹*angle-c z z1 ?z* = ⌊*2*∗*angle-c z z1 z2*⌋›
    by (*metis* ‹*z1* ≠ *z*› *add.inverse-inverse angle-c-commute angle-c-commute-pi*
      angle-sum-symmetry h(*1*) *mult-2-right mult-commute-abs*)
  have ‹ *cdist z1 z* = *cdist z1* (*axial-symmetry z1 z2 z*)›
    using *axial-symmetry-dist1 h*(*1*) **by** *blast*
  then show *?thesis*
    apply(*intro img-eqI*)
    by (*metis* ‹*angle-c z z1* (*axial-symmetry z1 z2 z*) = ⌊*2* ∗ *angle-c z z1 z2*⌋›)
qed

lemma *img-r-sym′*:
  assumes *h*:‹*z1* ≠ *z2*› ‹*z*∉*line z1 z2*›
  shows ‹*axial-symmetry z1 z2 z* = *complex-rotation.r z1* (⌊−*2*∗*angle-c z2 z1 z*⌋)
z›
  by (*metis angle-c-commute angle-c-neq0 axial-symmetry-eq-line*
      cancel-comm-monoid-add-class.diff-cancel complex-rotation.img-eqI
      complex-rotation.r-def h(*1*,*2*) *img-r-sym mult-eq-0-iff mult-minus-left*
      mult-minus-right pi-neq-zero two-pi-canonical)

lemma *equality-for-pqr*:
  assumes *1*:‹(*a2*::*complex*)∗*a3*≠*1*› **and** *2*:‹⋀*z*. *h z* = *a3*∗*z* + *b3*› **and** *3*:‹⋀*z*. *g*
z = *a2*∗*z* + *b2*› **and** *4*:‹*g* (*h z*) = *z*›
  shows ‹*z* = (*a2*∗*b3* + *b2*)/(*1*−*a2*∗*a3*)›
proof −
  have *f21*:‹*g* (*h z*) = *a2*∗*a3*∗*z* + *a2*∗*b3* +*b2*›
    using *assms* **by**(*auto simp*:*2 3field-simps*)
  then have ‹ *g* (*h z*) = *a2*∗*a3*∗*z* + *a2*∗*b3* +*b2* ⟷ *z*∗(*1*−*a2*∗*a3*) = *a2*∗*b3* +
b2›
    by(*auto simp*:*field-simps 4*)
  then have ‹*a2*∗*a3* ≠*1* ⟹ *z* = (*a2*∗*b3* + *b2*)/(*1*−*a2*∗*a3*)›
    using *f21* **by**(*auto simp*:*field-simps*)
  then show *?thesis* **using** *1* **by** *auto*
qed

lemma *equality-for-comp*:
  assumes *2*:‹⋀*z*. *h z* = (*a3*::*complex*)∗*z* + *b3*› **and** *3*:‹⋀*z*. *g z* = *a2*∗*z* + *b2*›
    and *4*:‹⋀*z*. *f z* = *a1*∗*z* +*b1*›
   shows ‹((*f*∘*f*∘*f*)∘(*g*∘*g*∘*g*)∘(*h*∘*h*∘*h*)) *z* = (*a1*∗*a2*∗*a3*)ˆ*3*∗*z* +(*a1*ˆ*2*+*a1*+*1*)∗*b1*
+*a1*ˆ*3*∗(*a2*ˆ*2*+*a2*+*1*)∗*b2*
+*a1*ˆ*3*∗*a2*ˆ*3*∗(*a3*ˆ*2*+*a3*+*1*)∗*b3* ›
  using *assms* **unfolding** *comp-def* **by**(*auto simp*:*fun-eq-iff power2-eq-square power3-eq-cube*
field-simps)

lemma *eq-translation-id*:
  assumes ‹h = *complex-rotation.r* A *0*› ‹h B = B›
  shows ‹h = *id*›
  using *assms*(*1*) *complex-rotation.r-id-iff* **by** *auto*

lemma *r-eqI*:
  assumes ‹$A = B$› ‹$\vartheta 1 = \vartheta 2$›
  shows ‹$r\ A\ \vartheta 1 = r\ B\ \vartheta 2$›
  using *assms*(*1*) *assms*(*2*) **by** *force*

lemma *r-eqI′*:
  assumes ‹$A = B$› ‹$\vartheta 1 = \vartheta 2$›
  shows ‹$r\ A\ \vartheta 1\ z = r\ B\ \vartheta 2\ z$›
  using *assms*(*1*) *assms*(*2*) **by** *force*

lemma *composed-rotations-same-center*:
 shows ‹(*complex-rotation.r* A $\vartheta 1$ ∘ *complex-rotation.r* A $\vartheta 2$) = *complex-rotation.r* A ($\vartheta 1 + \vartheta 2$)›
  unfolding *complex-rotation.r-def* **by** (*auto simp*: *fun-eq-iff cis-mult add-ac*)

lemma *composed-rotations*:
  assumes h:‹$\lfloor\vartheta 1 + \vartheta 2\rfloor \neq 0$›
  shows ‹(*complex-rotation.r* A $\vartheta 1$ ∘ *complex-rotation.r* B $\vartheta 2$) =
      complex-rotation.r $((A*(1-cis\ \vartheta 1) + B*cis\ \vartheta 1*(1-cis\ \vartheta 2))/(1-cis\ (\vartheta 1+\vartheta 2)))\ (\vartheta 1 + \vartheta 2)$›
proof −
  have ‹$cis\ (\vartheta 1 + \vartheta 2) \neq 1$›
    by (*metis arg-cis assms cis-zero zero-canonical*)
  with h **have** ‹(*complex-rotation.r* A $\vartheta 1$ ∘ *complex-rotation.r* B $\vartheta 2$)
        $((A*(1-cis\ \vartheta 1) + B*cis\ \vartheta 1*(1-cis\ \vartheta 2))/(1-cis\ (\vartheta 1+\vartheta 2)))$
      $= (A*(1-cis\ \vartheta 1) + B*cis\ \vartheta 1*(1-cis\ \vartheta 2))/(1-cis\ (\vartheta 1+\vartheta 2))$ ›
    unfolding *complex-rotation.r-def* **using** *assms*
    by(*auto simp*:*cis-mult field-simps intro*!:)
  with h **show** *?thesis*
    apply(*cases* ‹$\vartheta 1 = 0 \vee \vartheta 2 = 0$›)
    unfolding *complex-rotation.r-def* **using** *assms*
    by(*auto simp*:*cis-mult field-simps fun-eq-iff diff-divide-distrib add-divide-distrib intro*!:)
qed

lemma *composed-rotation-is-trans*:
  assumes ‹$\lfloor\vartheta 1 + \vartheta 2\rfloor = 0$›
  shows ‹(*complex-rotation.r* A $\vartheta 1$ ∘ *complex-rotation.r* B $\vartheta 2$) $z = z + (B - A)*(cis(\vartheta 1) - 1)$›
  using *assms*
 by (*auto simp*:*complex-rotation.r-def add-divide-distrib diff-divide-distrib field-simps*)

(*metis add.commute canon-ang-cos canon-ang-sin cis.code cis-mult cis-zero lambda-one mult.commute*)

8 Morley's theorem

We begin by proving the Morley's theorem in the case where angles are positives then using the congruence between two triangles with the same angles only not of the same sign we prove Morley's theorem when angles are negatives.

We then proceed to conclude because in a triangle either angles are all negatives or all the angles are positives depending on orientation.

```
theorem Morley-pos:
  assumes‹¬collinear A B C›
    ‹angle-c A B R = angle-c A B C / 3› (is ‹?abr = ?abc›)
    angle-c B A R = angle-c B A C / 3 (is ‹?bar = ?α›)
    angle-c B C P = angle-c B C A / 3 (is ‹?bcp = ?bca›)
    angle-c C B P = angle-c C B A / 3 (is ‹?cbp = ?β›)
    angle-c C A Q = angle-c C A B / 3 (is ‹?caq = ?cab›)
    angle-c A C Q = angle-c A C B / 3 (is ‹?acq = ?γ›)
   and hhh:‹⌊angle-c B A C / 3+angle-c C B A / 3+angle-c A C B / 3⌋ = pi/3›
  shows ‹cdist R P = cdist P Q ∧ cdist Q R = cdist P Q›
proof −
  have bundle-line:‹A∉ line B C› ‹B∉ line A C› ‹C ∉ line A B› ‹A≠B› ‹B≠C›
‹C≠A›
   using assms(1) non-collinear-independant by (auto simp:collinear-sym1 collinear-sym2
line-def)
  {fix A B C γ
    assume ABC-nline:‹A ∉ line B C›
      and eq-3c:‹angle-c A C B = 3∗γ›
    then have neq-PI:‹abs γ < pi/3›
    proof −
      have ‹angle-c A C B ≠ pi›
        using ABC-nline(1) unfolding line-def
       by (metis angle-c-commute-pi collinear-iff mem-Collect-eq non-collinear-independant)
      then have ‹angle-c A C B ∈ {−pi<..<pi}›
        using ang-c-in less-eq-real-def by auto
      then show ‹abs γ < pi/3›
        using eq-3c by force
    qed}note ang-inf-pi3=this
  have ‹⌊angle-c B A C + angle-c C B A  + angle-c A C B⌋ = pi›
    by (metis collinear-def add.commute
        angle-sum-triangle-c assms(1) collinear-sym1 collinear-sym2')
  have eq-pi:‹⌊3∗?α + 3∗?β + 3∗?γ⌋ = pi›
    using ‹⌊angle-c B A C + angle-c C B A + angle-c A C B⌋ = pi› by force
  then have neq-pi:‹⌊?β⌋ ≠ pi ∧ ⌊?γ⌋ ≠ pi ∧ ⌊?α⌋ ≠ pi›
    by (smt (verit) ang-neg-neg angle-c-commute angle-c-neq0 assms
```

canon-ang-sin collinear-angle collinear-sin-neq-0 divide-eq-0-iff divide-nonneg-pos

minus-divide-left pi-neq-zero sin-pi sin-pi-minus zero-canonical)
moreover have ‹$3*?\alpha \neq 0 \wedge 3*?\beta \neq 0 \wedge 3*?\gamma \neq 0$›
using *bundle-line collinear-sin-neq-0 line-def angle-c-commute assms*(*1*) *bundle-line*(*4*)
bundle-line(*5*) *bundle-line*(*6*) *collinear-iff assms*(*1*) *collinear-sin-neq-0*
by (*metis collinear-sym2′ divide-eq-eq-numeral1* (*1*) *mem-Collect-eq mult.commute zero-neq-numeral*)
ultimately have *neq-0*:‹$\lfloor ?\beta \rceil \neq 0 \wedge \lfloor ?\gamma \rceil \neq 0 \wedge \lfloor ?\alpha \rceil \neq 0$›
by (*metis ang-vec-def angle-c-def arg-cis assms*(*3*) *assms*(*5*) *assms*(*7*) *canon-ang-arg mult-zero-right*)

interpret *rot1*: *complex-rotation A 2∗?α* **.**
interpret *rot2*: *complex-rotation B 2∗?β* **.**
interpret *rot3*: *complex-rotation C 2∗?γ* **.**

let *?f*=‹*rot1.r*›
have *f0*:‹*rot1.r A* = *A*›
unfolding *rot1.r-def* **by** *auto*
have *f1*:‹*rot2.r B* = *B*›
unfolding *rot2.r-def* **by** *auto*
have *f2*:‹*rot3.r C* = *C*›
unfolding *rot3.r-def* **by** *auto*

have ‹*cmod* (*rot3.r z* − *C*) = *cmod* (*z*−*C*)› **for** *z*
using *rot3.cmod-inv-rotation* **by** *presburger*
have *f2*:‹*B*≠*C*›
by (*metis collinear-def assms*(*1*) *collinear-sym1 collinear-sym2*)
have *f5*:‹*angle-c C B P* = *?β*› ‹*angle-c P B C* = −*?β*›
by (*auto simp add*: *assms*(*5*) *angle-c-commute*)
(*metis angle-c-commute angle-c-commute-pi assms*(*5*) *neq-pi pi-canonical*)
then have *f3*:‹*P* ∉ *line C B*›
by (*smt* (*verit*, *ccfv-SIG*) *neq-0 neq-pi angle-c-commute angle-c-neq0 assms*(*4*)
collinear-angle divide-eq-0-iff line-def mem-Collect-eq pi-canonical zero-canonical)
then have *f3′*:‹*P* ∉ *line B C*›
using *collinear-sym2′ line-def* **by** *blast*
then have *f4*:‹*P*≠*C* ∧ *P*≠*B*›
by (*metis collinear-def collinear-sym1 collinear-sym2 line-def mem-Collect-eq*)
then have ‹*angle-c P B C* = − *angle-c* (*axial-symmetry B C P*) *B C*›
using *angle-symmetry-eq*[*OF f2 - - f3′*] **by** *auto*
then have ‹*angle-c* (*axial-symmetry B C P*) *B C* = *?β*›
using *f5*(*2*) **by** *fastforce*

have *f13*:‹*angle-c P C B* = *?γ*›
by (*smt* (*verit*, *ccfv-threshold*) *angle-c-commute angle-c-commute-pi assms*(*1*) *assms*(*4*) *assms*(*7*)
collinear-sin-neq-0 nonzero-minus-divide-divide nonzero-minus-divide-right sin-pi)

let *?P'*= ‹(*axial-symmetry B C P*)›

have ‹*angle-c* (*axial-symmetry B C P*) *B P* = ⌊*2*∗*?β*⌋›
by (*metis* ‹*P* ≠ *C* ∧ *P* ≠ *B*› ‹*angle-c* (*axial-symmetry B C P*) *B C* = *angle-c C B A* / *3*›
angle-sum-symmetry f2 f5(*1*) *involution-symmetry mult.commute mult-2-right z1-inv*)
have ‹*cdist B* (*P*) = *cdist B ?P'*›
by(*auto*)(*metis cmod-axial-inv f2 z1-inv*)
then have ‹*cdist B* (*rot2.r P*) = *cdist B ?P'*›
unfolding *cdist-def*
using *rot2.cmod-inv-rotation* **by** *presburger*
have *f16*:‹*rot2.r ?P'* = *P*›
by (*metis* ‹*angle-c* (*axial-symmetry B C P*) *B P* = ⌊*2* ∗ (*angle-c C B A* / *3*)⌋›
‹*cdist B P* = *cdist B* (*axial-symmetry B C P*)› *canon-ang-cos*
canon-ang-sin cis.code complex-rotation.img-eqI complex-rotation.r-def)
have *f10*:‹*angle-c P C B* = ⌊*?γ*⌋›
by (*metis* ‹*angle-c P C B* = *angle-c A C B* / *3*› *ang-vec-def angle-c-def arg-cis canon-ang-arg*)

from *f10* **have** *f11*:‹*angle-c B C ?P'* = ⌊*?γ*⌋›
by (*metis axial-symmetry-eq* ‹*P* ≠ *C* ∧ *P* ≠ *B*› ‹*angle-c P C B* = *angle-c A C B* / *3*› *angle-c-commute*
angle-symmetry angle-symmetry-eq-imp axial-symmetry-eq-line canon-ang-uminus-pi f2 f3 pi-canonical)
then have *f12*:‹*angle-c P C ?P'* = ⌊*2*∗*?γ*⌋›
by (*metis axial-symmetry-eq f13 angle-sum-symmetry f10 f2 f4 mult.commute mult-2-right*)
then have *f15*:‹*rot3.r P* = *?P'*›
by (*metis axial-symmetry-eq canon-ang-cos canon-ang-sin cis.code f13 f2 f3 img-r-sym complex-rotation.r-def*)
have *P-inv*:‹*rot2.r* (*rot3.r P*) = *P*›
using *f16 f15* **by** *presburger*
let *?Q'*=‹*axial-symmetry A C Q*›
have ‹*angle-c C A Q* = −*?α*›
by (*metis angle-c-commute assms*(*1*) *assms*(*6*) *collinear-sin-neq-0*
collinear-sym1 collinear-sym2' minus-divide-left sin-pi)
then have ‹*angle-c Q A C* = *?α*›
by (*metis add.inverse-inverse angle-c-commute canon-ang-uminus-pi neq-pi pi-canonical*)
have ‹*A*≠*C* ∧ *Q*∉*line A C*›
by (*metis* ‹*angle-c Q A C* = *angle-c B A C* / *3*› *angle-c-neq0 assms*(*7*) *collinear-angle div-0*
f10 f13 line-def mem-Collect-eq neq-0 neq-pi zero-canonical)
then have *f17*:‹*rot1.r Q* = *?Q'*›
using *img-r-sym*[*of A C Q*]
using ‹*angle-c Q A C* = *angle-c B A C* / *3*› *canon-ang-cos canon-ang-sin cis.code complex-rotation.r-def*
by *presburger*

then have ‹*angle-c ?Q′ C A = ?γ*›
by (*smt* (*verit, ccfv-SIG*) ‹*A ≠ C ∧ Q ∉ line A C*› ‹*angle-c Q A C = angle-c B A C / 3*›
complex-rotation.ang-eq-theta angle-c-commute angle-symmetry angle-symmetry-eq-imp assms(*7*)
axial-symmetry-eq axial-symmetry-eq-line f10 f13 complex-rotation.img-eqI neq-0 neq-pi)
then have ‹*angle-c A C Q = ?γ*›
using *assms*(*7*) **by** *blast*
then have ‹*angle-c ?Q′ C Q =*⌊*2∗?γ*⌋›
by (*metis* ‹*A ≠ C ∧ Q ∉ line A C*› ‹*angle-c* (*axial-symmetry A C Q*) *C A = angle-c A C B / 3*›
‹*angle-c Q A C = angle-c B A C / 3*› *angle-c-neq0 angle-sum-symmetry angle-symmetry-eq*
axial-symmetry-eq involution-symmetry mult-2-right mult-commute-abs neg-equal-0-iff-equal
neq-0 zero-canonical)
then have *f18*:‹*rot3.r ?Q′ = Q*›
using *img-r-sym*[*of C A ?Q′*]
by (*metis* ‹*A ≠ C ∧ Q ∉ line A C*› *axial-symmetry-dist1*
axial-symmetry-eq canon-ang-cos canon-ang-sin cis.code
complex-rotation.img-eqI complex-rotation.r-def)
have *Q-inv*:‹*rot3.r* (*rot1.r Q*) *= Q*›
using *f17 f18* **by** *presburger*
let *?R′*=‹*axial-symmetry A B R*›
have ‹*angle-c B A R = ?α*›
by (*simp add*: *assms*(*3*))
then have ‹*angle-c R A B = −?α*›
by (*metis angle-c-commute canon-ang-uminus-pi neq-pi pi-canonical*)
have ‹*angle-c R B A = ?β*›
by (*metis* ‹*angle-c* (*axial-symmetry B C P*) *B C = angle-c C B A / 3*› *angle-c-commute*
angle-c-commute-pi assms(*1*) *assms*(*2*) *collinear-sin-neq-0 collinear-sym1 minus-divide-left sin-pi*)
have ‹*A≠B ∧ R∉line A B*›
by (*metis* (*no-types, opaque-lifting*) ‹*angle-c Q A C = angle-c B A C / 3*› ‹*angle-c R A B = −*
(*angle-c B A C / 3*)› *angle-c-commute angle-c-commute-pi angle-c-neq0 assms*(*2*) *collinear-angle*
collinear-sym2′ diff-zero divide-eq-0-iff line-def mem-Collect-eq minus-diff-eq neg-equal-zero
neq-0 zero-canonical)
then have *f17*:‹*rot2.r R = ?R′*›
using *img-r-sym*[*of B A R*]
using ‹*angle-c R B A = angle-c C B A / 3*› *axial-symmetry-eq*
cis.code collinear-sym2 line-def complex-rotation.r-def **by** *auto*
then have ‹*angle-c ?R′ A B = ?α*›
by (*metis* ‹*A ≠ B ∧ R ∉ line A B*› ‹*angle-c R A B = −* (*angle-c B A C / 3*)›
‹*angle-c R B A = angle-c C B A / 3*› *add.inverse-inverse add.inverse-neutral angle-c-neq0*

```
        angle-symmetry-eq neq-0 zero-canonical)
  then have f18:‹rot1.r ?R′ = R›
    using img-r-sym[of A B ?R′]
   by (metis ‹A ≠ B ∧ R ∉ line A B› axial-symmetry-eq canon-ang-cos canon-ang-sin
cis.code
        involution-symmetry line-is-inv complex-rotation.r-def z2-inv)
  have R-inv:‹rot1.r (rot2.r R) = R›
    using f17 f18 by presburger
 let ?a1 = ‹cis(2∗?α)› let ?b1=‹A∗(1−?a1)› let ?a2=‹cis(2∗?β)› let ?b2=‹B∗(1−?a2)›
  let ?a3=‹cis(2∗?γ)› let ?b3=‹C∗(1−?a3)›
  have possi-abc:‹⌊?α+?β+?γ⌋ =pi/3 ∨ ⌊?α+?β+?γ⌋ = −pi/3 ∨ ⌊?α+?β+?γ⌋ =
pi›
  proof −
    have ‹⌊?α+?β+?γ⌋∈{−pi<..pi}›
      by (simp add: canon-ang(1) canon-ang(2))
    then have ‹3∗⌊?α+?β+?γ⌋ ∈ {−3∗pi<..3∗pi}›
      by(auto)
   then have ‹⌊⌊?α+?β+?γ⌋+⌊?α+?β+?γ⌋+⌊?α+?β+?γ⌋⌋ = ⌊?α+?β+?γ+?α+?β+?γ+?α+?β+?γ⌋›
      by (smt (verit, ccfv-SIG) add.commute arg-cis canon-ang-arg
          canon-ang-sum distrib-left is-num-normalize(1) mult-2)
    then have ‹⌊3∗⌊?α+?β+?γ⌋⌋=pi›
      using eq-pi by argo
   then have ‹3∗⌊?α+?β+?γ⌋ = 3∗pi ∨ 3∗⌊?α+?β+?γ⌋ = −pi ∨  3∗⌊?α+?β+?γ⌋=pi›
      by (smt (verit, del-insts) canon-ang(1) canon-ang(2) canon-ang-id
          canon-ang-minus-3pi-minus-pi canon-ang-pi-3pi)
    then show ?thesis
      by force
  qed
  have jj:‹B ∉ line C A› ‹Q ∉ line A C› ‹R ∉ line A B›
    using assms f3′ ‹A ≠ C ∧ Q ∉ line A C› ‹A ≠ B ∧ R ∉ line A B›
    by (simp-all add: collinear-sym1 collinear-sym2 line-def)
  then have inf-pi3:‹abs ?α < pi/3› ‹abs ?β < pi/3› ‹abs ?γ < pi/3›
    using ang-inf-pi3[of B C A ?α] ang-inf-pi3[of C A B ?β] ang-inf-pi3[of A B C
?γ]
    by (auto simp add:collinear-sym2 collinear-sym1 assms line-def)
  then have ‹abs (?α+?β+?γ) < pi›
    by argo
  have j1:‹⌊?α+?β+?γ⌋ = pi/3 ⟹ ?a1∗?a2∗?a3 = root3›
  proof −
    assume hh:‹⌊?α+?β+?γ⌋ = pi/3›
    have f20:‹cis (2 ∗ (angle-c B A C / 3)) ∗ cis (2 ∗ (angle-c C B A / 3)) ∗ cis
(2 ∗ (angle-c A C B / 3))
  = cis (2∗(?α+?β+?γ))›
      by (simp add: cis-mult)
    then have f21:‹cis (2∗(⌊?α+?β+?γ⌋)) = cis (2∗(?α+?β+?γ))›
       by (smt (verit, ccfv-threshold) canon-ang-cos canon-ang-sin canon-ang-sum
cis.code)
    with hh show ?thesis
      by (metis (no-types, opaque-lifting) f21  f20 times-divide-eq-right root3-def)
```

qed
have ‹*rot1.r*∘*rot1.r* = *complex-rotation.r A* (*4*∗*?α*)›
using *composed-rotations-same-center* **by** *auto*
have *g10*:‹((*rot1.r*∘*rot1.r*∘*rot1.r*)∘(*rot2.r*∘*rot2.r*∘*rot2.r*)∘(*rot3.r* ∘*rot3.r*∘*rot3.r*)) =
complex-rotation.r A (*6*∗*?α*) ∘ *complex-rotation.r B* (*6*∗*?β*) ∘ *complex-rotation.r C* (*6*∗*?γ*)›
using *composed-rotations-same-center* **by** *auto*
have *f26*:‹⌊*6*∗*?α* + *6*∗*?β* + *6*∗*?γ*⌋ = *0*›
by (*smt* (*verit*, *ccfv-SIG*) *canon-ang-sum eq-pi two-pi-canonical*)
also have ‹⌊*6*∗*?γ*⌋ ≠ *0*›
proof (*rule ccontr*)
assume ‹¬ ⌊*6* ∗ (*angle-c A C B* / *3*)⌋ ≠ *0*›
then obtain *k*::*int* **where** ‹*6* ∗ ⌊(*angle-c A C B* / *3*)⌋ = *2*∗*k*∗*pi*›
by (*metis add.commute add.inverse-neutral add-0 canon-ang*(*3*)
diff-conv-add-uminus f10 f13 of-int-mult of-int-numeral)
then have ‹*3*∗⌊(*angle-c A C B* / *3*)⌋ = *k*∗*pi*›
by *force*
then show *False*
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*1*) *collinear-sin-neq-0 divide-eq-eq-numeral1*(*1*)
f10 f13 mult-1 sin-kpi times-divide-eq-left zero-neq-numeral)
qed
then have *f24*:‹⌊*6*∗*?α* + *6*∗*?β*⌋ ≠ *0*›
by (*metis add.commute add.right-neutral arg-cis calculation*
canon-ang-cos canon-ang-sin canon-ang-sum cis.code)
let *?A1*=‹(*A*∗(*1*−*cis* (*6*∗*?α*)) + *B*∗*cis* (*6*∗*?α*)∗(*1*−*cis* (*6*∗*?β*)))/(*1*−*cis* (*6*∗*?β*+*6*∗*?α*))›
have *g11*:‹*complex-rotation.r A* (*6*∗*?α*) ∘ *complex-rotation.r B* (*6*∗*?β*) ∘ *complex-rotation.r C* (*6*∗*?γ*) =
complex-rotation.r ?A1 (*6*∗*?α* + *6*∗*?β*)∘ *complex-rotation.r C* (*6*∗*?γ*) ›
using *composed-rotations*[*of* ‹*6*∗*?α*› ‹*6*∗*?β*› *A B*, *OF f24*]
by *argo*
then have *f27*:‹*complex-rotation.r ?A1* (*6*∗*?α* + *6*∗*?β*)∘ *complex-rotation.r C* (*6*∗*?γ*) = (*λz. z* + (*C* −
(*A* ∗ (*1* − *cis* (*6* ∗ (*angle-c B A C* / *3*))) + *B* ∗ *cis* (*6* ∗ (*angle-c B A C* / *3*)) ∗ (*1* − *cis* (*6* ∗ (*angle-c C B A* / *3*)))) /
(*1* − *cis* (*6* ∗ (*angle-c C B A* / *3*) + *6* ∗ (*angle-c B A C* / *3*)))) ∗
(*cis* (*6* ∗ (*angle-c B A C* / *3*) + *6* ∗ (*angle-c C B A* / *3*)) − *1*))› (**is** ‹*?l* = (*λz. z* + *?A2*)›)
using *composed-rotation-is-trans*[*of* ‹*6*∗*?α* + *6*∗*?β*› ‹*6*∗*?γ*› *?A1 C*, *OF f26*]
by *auto*
have *f30*:‹*2*∗*angle-c A C B* = *6*∗*?γ*›
by *auto*
have ‹*axial-symmetry C B A* = *complex-rotation.r C* (⌊*2*∗*angle-c A C B*⌋) *A*›
using *img-r-sym collinear-sym1 f2 jj*(*1*) *line-def* **by** *auto*
then have *first*:‹*complex-rotation.r C* (*6*∗*?γ*) *A* = *axial-symmetry C B A*› (**is** ‹*?rr* = *?axA*›)
using *canon-ang-cos canon-ang-sin cis.code f30 complex-rotation.r-def* **by** *presburger*
have *f31*:‹*axial-symmetry B C ?axA* = *complex-rotation.r B* (⌊*2*∗*angle-c ?axA B*

C⌋) ?axA›
  by (*metis ‹A ≠ C ∧ Q ∉ line A C› ‹⌊6 ∗ (angle-c A C B / 3)⌋ ≠ 0›*
    ‹complex-rotation.r C (6 ∗ (angle-c A C B / 3)) A = axial-symmetry C B A›
    complex-rotation.ang-eq-theta angle-c-neq0
    axial-symmetry-eq f2 img-r-sym involution-symmetry line-is-inv z1-inv)
 have *f32*:*‹angle-c C B A = 3∗?β›*
  by *simp*
 then have *f33*:*‹angle-c ?axA B C = 3∗?β›*
  by (*smt* (*verit*) *‹A ≠ B ∧ R ∉ line A B› ‹A ≠ C ∧ Q ∉ line A C› ‹⌊6 ∗ (angle-c A C B / 3)⌋ ≠ 0›*
    ‹complex-rotation.r C (6 ∗ (angle-c A C B / 3)) A = axial-symmetry C B A›
    complex-rotation.ang-eq-theta angle-c-commute angle-c-neq0 angle-symmetry-eq assms(*1*)
    axial-symmetry-eq collinear-sin-neq-0 collinear-sym1 f2 line-is-inv sin-pi)
 then have *snd*:*‹complex-rotation.r B (6∗?β) ((axial-symmetry B C A)) = A›*
  by (*smt* (*verit*, *ccfv-SIG*) *‹A ≠ C ∧ Q ∉ line A C› ‹⌊6 ∗ (angle-c A C B / 3)⌋ ≠ 0›*
    ‹complex-rotation.r C (6 ∗ (angle-c A C B / 3)) A = axial-symmetry C B A›
    complex-rotation.ang-eq-theta angle-c-neq0 axial-symmetry-eq canon-ang-cos canon-ang-sin
    cis.code f2 img-r-sym involution-symmetry line-is-inv complex-rotation.r-def z1-inv)
 then have *thrd*:*‹complex-rotation.r A (6∗?α) A = A›*
  unfolding *complex-rotation.r-def* **by** *auto*
 then have *‹(complex-rotation.r A (6∗?α) ∘ complex-rotation.r B (6∗?β) ∘ complex-rotation.r C (6∗?γ)) A = A›*
  apply(*simp*)
  by (*smt* (*verit*, *best*) *axial-symmetry-eq f30 f32 first snd*)
 then have *g21*:*‹((rot1.r∘rot1.r∘rot1.r)∘(rot2.r∘rot2.r∘rot2.r)∘(rot3.r ∘rot3.r∘rot3.r)) = (λz. z + ?A2)›*
  apply(*subst g10*) **apply**(*subst g11*) **apply**(*subst f27*) **by**(*simp add:fun-eq-iff*)
 then have *‹?A2 = 0›*
  by (*metis ‹(complex-rotation.r A (6 ∗ (angle-c B A C / 3)) ∘*
      complex-rotation.r B (6 ∗ (angle-c C B A / 3)) ∘
      complex-rotation.r C (6 ∗ (angle-c A C B / 3))) A = A›
    add-diff-cancel-left′ cancel-comm-monoid-add-class.diff-cancel g10)
 then have *g22*:*‹((rot1.r∘rot1.r∘rot1.r)∘(rot2.r∘rot2.r∘rot2.r)∘(rot3.r ∘rot3.r∘rot3.r)) = id›*
  using *g21* **by**(*auto simp:fun-eq-iff*)
 have *g20*:*‹rot1.r z = ?a1∗z +?b1› ‹rot2.r z =?a2∗z + ?b2› ‹rot3.r z = ?a3 ∗ z + ?b3›* **for** *z*
  by(*auto simp:field-simps complex-rotation.r-def*)
 then have *‹((rot1.r∘rot1.r∘rot1.r)∘(rot2.r∘rot2.r∘rot2.r)∘(rot3.r ∘rot3.r∘rot3.r)) z = (cis (2 ∗ (angle-c B A C / 3)) ∗ cis (2 ∗ (angle-c C B A / 3)) ∗ cis (2 ∗ (angle-c A C B / 3))) ^ 3 ∗ z +*
  ((cis (2 ∗ (angle-c B A C / 3)))^2 *+ cis (2 ∗ (angle-c B A C / 3)) + 1) ∗ (A*

```
∗ (1 − cis (2 ∗ (angle-c B A C / 3)))) +
    cis (2 ∗ (angle-c B A C / 3)) ^ 3 ∗ ((cis (2 ∗ (angle-c C B A / 3)))^2 + cis (2
∗ (angle-c C B A / 3)) + 1) ∗
    (B ∗ (1 − cis (2 ∗ (angle-c C B A / 3)))) +
    cis (2 ∗ (angle-c B A C / 3)) ^ 3 ∗ cis (2 ∗ (angle-c C B A / 3)) ^ 3 ∗
    ((cis (2 ∗ (angle-c A C B / 3)))^2 + cis (2 ∗ (angle-c A C B / 3)) + 1) ∗
    (C ∗ (1 − cis (2 ∗ (angle-c A C B / 3)))))› (is ‹?l z = ?A3 z›) for z
     using equality-for-comp[of rot3.r ?a3 ?b3 rot2.r ?a2 ?b2 rot1.r ?a1 ?b1, OF
g20(3) g20(2) g20(1)]
    by blast
  then have ‹z = ?A3 z› for z
    by (metis g22 id-apply)
  then have very-imp:‹((cis (2 ∗ (angle-c B A C / 3)))^2 + cis (2 ∗ (angle-c B A
C / 3)) + 1) ∗ (A ∗ (1 − cis (2 ∗ (angle-c B A C / 3)))) +
    cis (2 ∗ (angle-c B A C / 3)) ^ 3 ∗ ((cis (2 ∗ (angle-c C B A / 3)))^2 + cis (2
∗ (angle-c C B A / 3)) + 1) ∗
    (B ∗ (1 − cis (2 ∗ (angle-c C B A / 3)))) +
    cis (2 ∗ (angle-c B A C / 3)) ^ 3 ∗ cis (2 ∗ (angle-c C B A / 3)) ^ 3 ∗
    ((cis (2 ∗ (angle-c A C B / 3)))^2 + cis (2 ∗ (angle-c A C B / 3)) + 1) ∗
    (C ∗ (1 − cis (2 ∗ (angle-c A C B / 3)))) = 0›
    by (metis comm-monoid-add-class.add-0 mult-zero-class.mult-zero-right)
  {assume hhh:‹⌊?α+?β+?γ⌋ = pi/3›
    have j5:‹?a1∗?a2 ≠1›
    proof(rule ccontr)
      assume ‹¬ cis (2 ∗ (angle-c B A C / 3)) ∗ cis (2 ∗ (angle-c C B A / 3)) ≠
1›
      then have ‹?a1∗?a2 = cis 0›
        using cis-zero by presburger
      then have ‹⌊2∗(?α+?β)⌋ = 0›
        by (metis arg-cis cis-mult distrib-left zero-canonical)
      then have t:‹⌊?α+?β⌋ = pi ∨ ⌊?α+?β⌋ = 0›
      by (smt (verit, del-insts) canon-ang(1) canon-ang-id canon-ang-sum canon-ang-uminus)
      then have ‹⌊?α+?β⌋ = 0 ⟹ False›
        by (metis add-0 assms(1) canon-ang-sum collinear-sin-neq-0
            divide-cancel-right f10 f13 hhh sin-pi zero-neq-numeral)
      then have ‹⌊1/3∗(3∗?α+3∗?β)⌋=pi› using t by(auto simp:algebra-simps)
      then have ‹⌊(pi−⌊3∗?γ⌋)⌋ = ⌊(3∗?α+3∗?β)⌋›
        by (smt (verit, ccfv-SIG) canon-ang-diff eq-pi)
      then have ‹⌊(pi/3−⌊?γ⌋)⌋ = pi›
         by (smt (verit, best) ‹⌊angle-c B A C / 3 + angle-c C B A / 3⌋ = 0 ⟹
False›
            canon-ang-diff hhh t)
      then have ‹⌊?γ⌋ ∈ {0<..<pi/3}›
        by (smt (verit, ccfv-SIG) add-divide-distrib ang-vec-bounded angle-c-def
          arccos-minus-one-half arccos-one-half arcsin-minus-one-half arcsin-one-half

             arcsin-plus-arccos canon-ang-id canon-ang-pi-3pi divide-cancel-right f13
field-sum-of-halves)
      then show False
```

```
      using ‹⌊pi / 3 − ⌊angle-c A C B / 3⌋⌋ = pi› add-divide-distrib canon-ang-id
by auto
    qed
    have r-eq-all:‹rot1.r z = ?a1∗z + ?b1› ‹rot2.r z = ?a2∗z + ?b2› ‹rot3.r z =
?a3∗z+?b3› for z
      by(auto simp:field-simps complex-rotation.r-def cis.code)
    have f21:‹rot2.r (rot3.r P) = ?a2∗?a3∗P + ?a2∗?b3 +?b2›
      apply(subst r-eq-all(2)) apply(subst r-eq-all(3)) by(auto simp:field-simps)

   then have ‹ rot2.r (rot3.r P) = ?a2∗?a3∗P + ?a2∗?b3 +?b2 ⟷ P∗(1−?a2∗?a3)
= ?a2∗?b3 + ?b2›
      by (smt (verit) add.commute add-diff-cancel-left' add-diff-eq f15
f16 mult.commute mult.right-neutral right-diff-distrib')
    then have j2:‹?a2∗?a3 ≠1 ⟹  P = (?a2∗?b3 + ?b2)/(1−?a2∗?a3)›
      using f21 by(auto simp:field-simps)
    have j4:‹?a1∗?a3 ≠ 1›
    proof(rule ccontr)
      assume ‹¬ cis (2 ∗ (angle-c B A C / 3)) ∗ cis (2 ∗ (angle-c A C B / 3)) ≠
1›
      then have ‹?a1∗?a3 = cis 0›
        using cis-zero by presburger
      then have ‹⌊2∗(?α+?γ)⌋ = 0›
        by (metis arg-cis cis-mult distrib-left zero-canonical)
      then have t:‹⌊?α+?γ⌋ = pi ∨ ⌊?α+?γ⌋ = 0›
      by (smt (verit, del-insts) canon-ang(1) canon-ang-id canon-ang-sum canon-ang-uminus)
      then have k0:‹⌊?α+?γ⌋ = 0 ⟹ False›
        by (smt (verit, ccfv-SIG) ‹A ≠ B ∧ R ∉ line A B› ‹A ≠ C ∧ Q ∉ line A
C›
            ‹|angle-c B A C / 3 + angle-c C B A / 3 + angle-c A C B / 3| < pi›
            ‹angle-c (axial-symmetry A B R) A B = angle-c B A C / 3› ‹angle-c R
A B = − (angle-c B A C / 3)›
           ‹angle-c R B A = angle-c C B A / 3› ang-pos-pos assms(3) canon-ang-id
f2 f30 f32 neq-0 z1-inv)

      then have ‹⌊1/3∗(3∗?α+3∗?γ)⌋=pi› using t by(auto simp:algebra-simps)
      then have ‹⌊(pi−⌊3∗?β⌋)⌋ = ⌊(3∗?α+3∗?γ)⌋›
        by (smt (verit, ccfv-SIG) canon-ang-diff eq-pi)
      then have ‹⌊(pi/3−⌊?β⌋)⌋ = pi›
        by (smt (verit, best) k0
            canon-ang-diff hhh t)
      then have k1:‹⌊?β⌋ ∈ {0<..<pi/3}›
        by (smt (verit, del-insts) arccos-one-half arcsin-one-half arcsin-plus-arccos
            canon-ang-id divide-nonneg-pos field-sum-of-halves inf-pi3(2) pi-ge-zero)
      then show False
        using k1 add-divide-distrib canon-ang-id ‹⌊pi / 3 − ⌊angle-c C B A / 3⌋⌋
= pi› by auto
    qed
    have j3:‹?a2∗?a3 ≠ 1›
    proof(rule ccontr)
```

assume ‹¬ *cis* (*2* ∗ (*angle-c C B A* / *3*)) ∗ *cis* (*2* ∗ (*angle-c A C B* / *3*)) ≠ *1*›
then have ‹*?a2*∗*?a3* = *cis 0*›
using *cis-zero* **by** *presburger*
then have ‹⌊*2*∗(*?β*+*?γ*)⌋ = *0*›
by (*metis arg-cis cis-mult distrib-left zero-canonical*)
then have *t*:‹⌊*?β*+*?γ*⌋ = *pi* ∨ ⌊*?β*+*?γ*⌋ = *0*›
by (*smt* (*verit*, *del-insts*) *canon-ang*(*1*) *canon-ang-id canon-ang-sum canon-ang-uminus*)
then have *k0*:‹⌊*?β*+*?γ*⌋ = *0* ⟹ *False*›
by (*smt* (*verit*, *ccfv-SIG*) ‹*A* ≠ *B* ∧ *R* ∉ *line A B*› ‹*A* ≠ *C* ∧ *Q* ∉ *line A C*›
‹|*angle-c B A C* / *3* + *angle-c C B A* / *3* + *angle-c A C B* / *3*| < *pi*›
‹*angle-c* (*axial-symmetry A B R*) *A B* = *angle-c B A C* / *3*› ‹*angle-c R A B* = − (*angle-c B A C* / *3*)›
‹*angle-c R B A* = *angle-c C B A* / *3*› *ang-pos-pos assms*(*3*) *canon-ang-id f2 f30 f32 neq-0 z1-inv*)

then have ‹⌊*1*/*3*∗(*3*∗*?β*+*3*∗*?γ*)⌋=*pi*› **using** *t* **by**(*auto simp*:*algebra-simps*)
then have ‹⌊(*pi*−⌊*3*∗*?α*⌋)⌋ = ⌊(*3*∗*?β*+*3*∗*?γ*)⌋›
by (*smt* (*verit*, *ccfv-SIG*) *canon-ang-diff eq-pi*)
then have ‹⌊(*pi*/*3*−⌊*?α*⌋)⌋ = *pi*›
by (*smt* (*verit*, *best*) *k0*
canon-ang-diff hhh t)
then have *k1*:‹⌊*?α*⌋ ∈ {*0*<..<*pi*/*3*}›
by (*smt* (*verit*, *del-insts*) *arccos-one-half arcsin-one-half arcsin-plus-arccos canon-ang-id divide-nonneg-pos field-sum-of-halves inf-pi3*(*1*) *pi-ge-zero*)
then show *False*
using *k1 add-divide-distrib canon-ang-id* ‹⌊(*pi*/*3*−⌊*?α*⌋)⌋ = *pi*› **by** *auto*
qed
have *f21′*:‹*rot3*.*r* (*rot1*.*r Q*) = *?a3*∗*?a1*∗*Q* + *?a3*∗*?b1* +*?b3*›
apply(*subst r-eq-all*(*1*)) **apply**(*subst r-eq-all*(*3*)) **by**(*auto simp*:*field-simps*)
have *f21″*:‹*rot1*.*r* (*rot2*.*r R*) = *?a1*∗*?a2*∗*R* + *?a1*∗*?b2* +*?b1*›
apply(*subst r-eq-all*(*1*)) **apply**(*subst r-eq-all*(*2*)) **by**(*auto simp*:*field-simps*)
have *jjj*:‹*?a1* ≠ *0* ∧ *?a2* ≠ *0* ∧ *?a3* ≠ *0*›
using *cis-neq-zero* **by** *blast*
then have *eq-j*:‹*?a1*∗*?a2*∗*?a3* = *root3*›
using *j1 hhh* **by** *blast*
then have *P-def*:‹*P* = (*?a2*∗*?b3* + *?b2*)/(*1*−*?a2*∗*?a3*)›(**is** ‹-=*?P*›)
using *j2 j3* **by** *blast*
have *n1*: ‹*1*−*?a2*∗*?a3* = (*?a1* − *root3*)/*?a1*›
by (*smt* (*verit*, *ccfv-threshold*) *cis-divide cis-times-cis-opposite cis-zero diff-divide-distrib eq-j minus-real-def mult-1 times-divide-eq-left*)
then have *P-last*:‹*P* = *?a1*∗(*?a2*∗*?b3* + *?b2*)/(*?a1*−*root3*)›
using *P-def jjj* **by** (*simp add*:)
then have *Q-def*:‹*Q* = (*?a3*∗*?b1* + *?b3*)/(*1*−*?a3*∗*?a1*)› (**is** ‹-=*?Q*›)
using *f21′ j4 Q-inv* **by**(*auto simp*:*field-simps*)
have *n2*: ‹*1*−*?a3*∗*?a1* = (*?a2* − *root3*)/*?a2*›
by (*smt* (*verit*, *ccfv-threshold*) *cis-divide cis-times-cis-opposite cis-zero diff-divide-distrib eq-j minus-real-def mult-1 times-divide-eq-left*)

then have *Q-last*:‹Q = ?a2∗(?a3∗?b1 + ?b3)/(?a2−root3)›
using *Q-def jjj* **by** *simp*
then have *R-def*:‹R = (?a1∗?b2 + ?b1)/(1−?a1∗?a2)› (**is** ‹-=?R›)
using *f21′′ j5 R-inv* **by**(*auto simp:field-simps*)
have *n3*:‹1−?a1∗?a2 = (?a3 − root3)/?a3›
by (*smt* (*verit, ccfv-threshold*) *cis-divide cis-times-cis-opposite cis-zero diff-divide-distrib eq-j minus-real-def mult-1 times-divide-eq-left*)
then have *R-last*:‹R = ?a3∗(?a1∗?b2 + ?b1)/(?a3−root3)›
using *R-def jjj* **by** *simp*
have ‹?a1 − root3 ≠ 0 ∧ ?a2−root3≠0 ∧ ?a3−root3 ≠ 0›
using *eq-j j3 j4 j5* **by** *auto*
have *rule-s*:‹c ≠ 0 =⇒ a/c + b/c = (a+b)/c› **for** *a b c::real*
by *argo*
define *j′* **where**
defs: ‹j′≡root3›
have *simp-rules-for-eq*:‹P = (?a2∗?b3 + ?b2)/(1−?a2∗?a3)› ‹R = (?a1∗?b2 + ?b1)/(1−?a1∗?a2)›
‹Q = (?a3∗?b1 + ?b3)/(1−?a1∗?a3)› ‹1−?a1∗?a2 ≠ 0 ∧ 1−?a2∗?a3≠0 ∧ 1−?a1∗?a3 ≠ 0›
‹?a1∗?a2∗?a3 = j′ ›
‹1 + j′ +j′^2 = 0› ‹((1 − ?a1 ∗ ?a3) ∗ ((1 − ?a1 ∗ ?a2) ∗ (1 − ?a2 ∗ ?a3)))≠0›
‹j′^3 = 1› ‹j′∗j′∗j′ = 1› ‹?a1≠0› ‹?a2≠0› ‹?a3≠0›
‹(1−?a1∗?a2)∗?a3 = (?a3−j′)› ‹(1−?a2∗?a3)∗?a1 = (?a1−j′)› ‹(1−?a1∗?a3)∗?a2 = (?a2−j′)›
using *Q-def P-def R-def eq-j j3 jjj j4 j5 root3-eq-0 root-unity-3 n1 n2 n3*
by(*auto simp add:mult.commute power3-eq-cube defs root3-def*)
have *graal*:‹(?a1^2+?a1+1)∗?b1 + ?a1^3∗(?a2^2+?a2+1)∗?b2 +?a1^3∗?a2^3∗(?a3^2+?a3+1)∗?b3 =
−j′∗?a1^2∗?a2∗(?a1−j′)∗(?a2−j′)∗(?a3−j′)∗(?R +j′∗?P +j′^2∗?Q)›
using *root-unity-carac*[*of ?a1 ?a2 ?a3 j′ ?R ?b2 ?b1 ?P ?b3 ?Q*]
using *simp-rules-for-eq defs* **by**(*auto simp*:)
then have ‹(?a1^2+?a1+1)∗?b1 + ?a1^3∗(?a2^2+?a2+1)∗?b2 +?a1^3∗?a2^3∗(?a3^2+?a3+1)∗?b3 = 0›
unfolding *defs* **using** *very-imp* **by** *auto*
then have ‹(?R +j′∗?P +j′^2∗?Q) = 0›
using *graal simp-rules-for-eq*(*13*) *simp-rules-for-eq*(*14*) *simp-rules-for-eq*(*15*) *simp-rules-for-eq*(*11*) *simp-rules-for-eq*(*12*) *simp-rules-for-eq*(*4*) **by** *force*
then have *impp*:‹R +j′∗P +j′^2∗Q = 0›
using *R-def P-def Q-def*
by (*metis simp-rules-for-eq*(*1*) *simp-rules-for-eq*(*2*))
have *neq-all*:‹A≠B ∧ B≠C ∧ C≠A›
using ‹A ≠ B ∧ R ∉ line A B› ‹A ≠ C ∧ Q ∉ line A C› *f2* **by** *argo*
have ‹R ≠ Q›
proof (*rule ccontr*)
assume ‹¬ R ≠ Q›
then have *q1*:‹angle-c A B R = angle-c A B Q›
‹angle-c B A R = angle-c B A Q› ‹angle-c C A Q = angle-c C A R›

‹angle-c A C Q = angle-c A C R›
using assms by auto
then have ‹angle-c B A R = ?α› using assms by auto
then have ‹angle-c B A Q = ?α›
using q1 by auto
then have ‹angle-c A B Q = angle-c A B R + angle-c R B Q›
using ‹¬ R ≠ Q› angle-c-neq0 by auto
have ‹angle-c C A B = − 3∗?α› using assms
using ‹angle-c C A Q = − (angle-c B A C / 3)› by argo
then have ‹angle-c (axial-symmetry B A R) A B = ?α›
by (metis ‹angle-c (axial-symmetry A B R) A B = angle-c B A C / 3› axial-symmetry-eq)
have ‹C − A ≠ 0 ∧ Q − A ≠ 0 ∧A − R ≠ 0 ∧ A − B ≠ 0›
using ‹A ≠ C ∧ Q ∉ line A C› ‹angle-c A B Q = angle-c A B R + angle-c R B Q›
‹angle-c R B A = angle-c C B A / 3› neq-0 q1(1) neq-all by fastforce
then have ‹angle-c C A B = angle-c C A Q + angle-c Q A R + angle-c R A B›
using angle-c-sum[of C A Q R] angle-c-sum[of C A R B]
by (smt (verit) ‹¬ R ≠ Q› ‹angle-c C A B = − 3 ∗ (angle-c B A C / 3)›
‹angle-c C A Q = − (angle-c B A C / 3)› ‹angle-c R A B = − (angle-c B A C / 3)›
angle-c-neq0 canon-ang(1) canon-ang(2) canon-ang-id)
then show False
using ‹¬ R ≠ Q›
by (smt (verit, best) ‹angle-c C A B = − 3 ∗ (angle-c B A C / 3)›
‹angle-c C A Q = − (angle-c B A C / 3)› ‹angle-c R A B = − (angle-c B A C / 3)›
angle-c-neq0 neq-0 zero-canonical)
qed
then have ‹cdist R P = cdist R Q ∧ cdist R Q = cdist Q P›
using equilateral-caracterization-neg[of R Q P] impp unfolding defs
by (metis cdist-commute)
then have ?thesis
using cdist-commute by auto
}note case-pi3=this
then show ?thesis using hhh by auto
qed

theorem Morley-neg:
assumes‹¬collinear A B C›
‹angle-c A B R = angle-c A B C / 3› (is ‹?abr = ?abc›)
angle-c B A R = angle-c B A C / 3 (is ‹?bar = ?α›)
angle-c B C P = angle-c B C A / 3 (is ‹?bcp = ?bca›)
angle-c C B P = angle-c C B A / 3 (is ‹?cbp = ?β›)
angle-c C A Q = angle-c C A B / 3 (is ‹?caq = ?cab›)
angle-c A C Q = angle-c A C B / 3 (is ‹?acq = ?γ›)
and hhh:‹⌊angle-c B A C / 3+angle-c C B A / 3+angle-c A C B / 3⌉ = −pi/3›

```
  shows  ‹cdist R P = cdist P Q ∧ cdist Q R = cdist P Q›
proof –
  have ‹⌊−angle-c B A C / 3+−angle-c C B A / 3+−angle-c A C B / 3⌋ = pi/3›
    using hhh
   by (metis (no-types, opaque-lifting) canon-ang(1) canon-ang-uminus less-eq-real-def

        linorder-not-le minus-add-distrib minus-divide-left mult-le-0-iff
      nonzero-mult-div-cancel-left not-numeral-le-zero pi-gt-zero times-divide-eq-right
verit-minus-simplify(4) zero-canonical)
  then show ?thesis using Morley-pos[of C B A P Q R]
    by (smt (verit, best) Morley-pos angle-c-commute assms(1) assms(2) assms(3)
assms(4) assms(5)
      assms(6) assms(7) cdist-commute collinear-sin-neq-0 collinear-sym1 collinear-sym2
sin-pi)
qed

theorem Morley:
  assumes‹¬collinear A B C›
    ‹angle-c A B R = angle-c A B C / 3› (is ‹?abr = ?abc›)
    angle-c B A R = angle-c B A C / 3 (is ‹?bar = ?α›)
    angle-c B C P = angle-c B C A / 3 (is ‹?bcp = ?bca›)
    angle-c C B P = angle-c C B A / 3 (is ‹?cbp = ?β›)
    angle-c C A Q = angle-c C A B / 3 (is ‹?caq = ?cab›)
    angle-c A C Q = angle-c A C B / 3 (is ‹?acq = ?γ›)
  shows  ‹cdist R P = cdist P Q ∧ cdist Q R = cdist P Q›
proof –
  {fix A B C γ
    assume ABC-nline:‹A ∉ line B C›
      and eq-3c:‹angle-c A C B = 3∗γ›
    then have neq-PI:‹abs γ < pi/3›
    proof –
      have ‹angle-c A C B ≠ pi› using ABC-nline(1) unfolding line-def
      by (metis angle-c-commute-pi collinear-iff mem-Collect-eq non-collinear-independant)
      then have ‹angle-c A C B ∈ {−pi<..<pi}›
        using ang-c-in less-eq-real-def by auto
      then show ‹abs γ < pi/3›
        using eq-3c by force
    qed}note ang-inf-pi3=this
  have ‹⌊angle-c B A C + angle-c C B A  + angle-c A C B⌋ = pi›
    by (metis Elementary-Complex-Geometry.collinear-def add.commute
        angle-sum-triangle-c assms(1) collinear-sym1 collinear-sym2')
  have eq-pi:‹⌊3∗?α + 3∗?β + 3∗?γ⌋ = pi›
    using ‹⌊angle-c B A C + angle-c C B A + angle-c A C B⌋ = pi› by force
  have ‹A∉line B C ∧ B∉ line A C ∧ C ∉ line A B›
   using assms(1) unfolding line-def using collinear-sym1 collinear-sym2' by(auto)

  then have f2:‹abs ?α < pi/3 ∧ abs ?β < pi/3 ∧ abs ?γ < pi/3›
    using ang-inf-pi3
   by (smt (verit, ccfv-SIG) ang-vec-bounded angle-c-def assms(1) collinear-sin-neq-0
```

```
      collinear-sym1 collinear-sym2 divide-less-cancel minus-divide-left sin-pi)
  have possi-abc:‹⌊?α+?β+?γ⌋ =pi/3 ∨ ⌊?α+?β+?γ⌋ = −pi/3›
  proof −
    have ‹?α+?β+?γ ≤ pi›
      using f2 by(auto simp:field-simps)
    then have ‹⌊?α+?β+?γ⌋ = ?α+?β+?γ›
      using canon-ang-id f2 by fastforce
    then have ‹⌊3∗⌊?α+?β+?γ⌋⌋ = pi›
      using eq-pi by force
    then show ?thesis
       by (smt (verit) add-divide-distrib arccos-minus-one-half arccos-one-half arc-
sin-minus-one-half
 arcsin-one-half arcsin-plus-arccos canon-ang-id canon-ang-minus-3pi-minus-pi canon-ang-pi-3pi

 eq-pi f2 field-sum-of-halves)
  qed
  then show ?thesis
    using Morley-neg Morley-pos assms(1) assms(2) assms(3) assms(4) assms(5)
assms(6) assms(7)
    by meson
qed

end
```

References

[1] Manuel Eberl *Basic Geometric Properties of Triangles*, Archive of Formal Proofs, December 2015 https://isa-afp.org/entries/Triangle.html

[2] Morley theorem http://denisevellachemla.eu/Alain-Connes-Theoreme-Morley.pdf.

[3] Wiedijk's catalogue "Formalizing 100 Theorems" https://www.cs.ru.nl/~freek/100/,Itappearsatposition121....