Farey Sequences and Ford Circles

Lawrence C. Paulson

18 May 2025

Abstract

The sequence F_n of *Farey fractions* of order n has the form

$$\frac{0}{1}, \frac{1}{n}, \frac{1}{n-1}, \ldots, \frac{n-1}{n}, \frac{1}{1}$$

where the fractions appear in numerical order and have denominators at most n. The transformation from F_n to F_{n+1} can be effected by combining adjacent elements of the sequence F_n, using an operation called the *mediant*. Adjacent (reduced) fractions $(a/b) < (c/d)$ satisfy the *unimodular* relation $bc - ad = 1$ and their mediant is $\frac{a+c}{b+d}$. A *Ford circle* is specified by a rational number, and interesting consequences follow in the case of Ford circles obtained from some Farey sequence F_n. The formalised material is drawn from Apostol's *Modular Functions and Dirichlet Series in Number Theory* [1].

1 Farey Sequences and Ford Circles

theory *Farey-Ford*
imports *HOL−Analysis.Analysis HOL−Number-Theory.Totient HOL−Library.Sublist*

begin

1.1 Library material

lemma *sum-squared-le-sum-of-squares*:
fixes $f :: 'a \Rightarrow real$
assumes *finite I*
shows $(\sum i{\in}I.\ f\ i)^2 \leq (\sum y{\in}I.\ (f\ y)^2) * card\ I$
proof (*cases finite I* $\wedge$ $I \neq \{\}$)
case *True*
then have $(\sum i{\in}I.\ f\ i\ /\ real\ (card\ I))^2 \leq (\sum i{\in}I.\ (f\ i)^2\ /\ real\ (card\ I))$
using *assms convex-on-sum* [*OF - - convex-power2*, **where** $a = \lambda x.\ 1\ /\ real(card\ I)$ **and** *S=I*]
by *simp*
then show *?thesis*
using *assms*
by (*simp add*: *divide-simps power2-eq-square split*: *if-split-asm flip*: *sum-divide-distrib*)
qed *auto*

lemma *sum-squared-le-sum-of-squares-2*:
$(x+y)/2 \leq sqrt\ ((x^2 + y^2)\ /\ 2)$
proof −
have $(x + y)^2\ /\ 2\hat{}2 \leq (x^2 + y^2)\ /\ 2$
using *sum-squared-le-sum-of-squares* [*of UNIV* $\lambda b.$ *if b then x else y*]
by (*simp add*: *UNIV-bool add.commute*)
then show *?thesis*
by (*metis power-divide real-le-rsqrt*)
qed

lemma *sphere-scale*:
assumes $a \neq 0$
shows $(\lambda x.\ a *_R x)$ ‘ *sphere c r* = *sphere* $(a *_R c :: 'a ::$ *real-normed-vector*$)$ $(|a| * r)$
proof −
have $*$: $(\lambda x.\ a *_R x)$ ‘ *sphere c r* $\subseteq$ *sphere* $(a *_R c)$ $(|a| * r)$ **for** *a r* **and** $c :: 'a$
by (*metis* (*no-types*, *opaque-lifting*) *scaleR-right-diff-distrib dist-norm image-subsetI mem-sphere norm-scaleR*)
have *sphere* $(a *_R c)$ $(|a| * r) = (\lambda x.\ a *_R x)$ ‘ $(\lambda x.$ *inverse a* $*_R x)$ ‘ *sphere* $(a *_R c)$ $(|a| * r)$
unfolding *image-image* **using** *assms* **by** *simp*
also have ... $\subseteq (\lambda x.\ a *_R x)$ ‘ *sphere* $(inverse\ a *_R (a *_R c))$ $(|inverse\ a| * (|a| * r))$
using $*$ **by** *blast*

also have ... = $(\lambda x.\ a *_R x)$ ‘ *sphere c r*
using *assms* **by** (*simp add*: *algebra-simps*)
finally have *sphere* $(a *_R c)$ $(|a| * r) \subseteq (\lambda x.\ a *_R x)$ ‘ *sphere c r* **.**
moreover have $(\lambda x.\ a *_R x)$ ‘ *sphere c r* $\subseteq$ *sphere* $(a *_R c)$ $(|a| * r)$
using $*$ **by** *blast*
ultimately show *?thesis* **by** *blast*
qed

lemma *sphere-cscale*:
assumes $a \neq 0$
shows $(\lambda x.\ a * x)$ ‘ *sphere c r* = *sphere* $(a * c :: complex)$ $(cmod\ a * r)$
proof −
have $*$: $(\lambda x.\ a * x)$ ‘ *sphere c r* $\subseteq$ *sphere* $(a * c)$ $(cmod\ a * r)$ **for** *a r c*
by (*metis* (*no-types*, *lifting*) *dist-complex-def image-subsetI mem-sphere norm-mult right-diff-distrib′*)
have *sphere* $(a * c)$ $(cmod\ a * r)$ = $(\lambda x.\ a * x)$ ‘ $(\lambda x.\ inverse\ a * x)$ ‘ *sphere* $(a * c)$ $(cmod\ a * r)$
by (*simp add*: *image-image inverse-eq-divide*)
also have ... $\subseteq$ $(\lambda x.\ a * x)$ ‘ *sphere* $(inverse\ a * (a * c))$ $(cmod\ (inverse\ a) * (cmod\ a * r))$
using $*$ **by** *blast*
also have ... = $(\lambda x.\ a * x)$ ‘ *sphere c r*
using *assms* **by** (*simp add*: *field-simps flip*: *norm-mult*)
finally have *sphere* $(a * c)$ $(cmod\ a * r) \subseteq (\lambda x.\ a * x)$ ‘ *sphere c r* **.**
moreover have $(\lambda x.\ a * x)$ ‘ *sphere c r* $\subseteq$ *sphere* $(a * c)$ $(cmod\ a * r)$
using $*$ **by** *blast*
ultimately show *?thesis* **by** *blast*
qed

lemma *Complex-divide-complex-of-real*: *Complex x y* / *of-real r* = *Complex* (x/r) (y/r)
by (*metis complex-of-real-mult-Complex divide-inverse mult.commute of-real-inverse*)
lemma *cmod-neg-real*: *cmod* $(Complex\ (-x)\ y)$ = *cmod* $(Complex\ x\ y)$
by (*metis complex-cnj complex-minus complex-mod-cnj norm-minus-cancel*)

1.2 Farey sequences

lemma *sorted-two-sublist*:
fixes *x*:: *′a*::*order*
assumes *sorted*: *sorted-wrt* $(<)$ *l*
shows *sublist* $[x, y]$ *l* $\longleftrightarrow$ $x < y \wedge x \in set\ l \wedge y \in set\ l \wedge (\forall z \in set\ l.\ z \leq x \vee z \geq y)$
proof (*cases* $x < y \wedge x \in set\ l \wedge y \in set\ l$)
case *True*
then obtain *xs us* **where** *us*: *l* = *xs* @ $[x]$ @ *us*
by (*metis append-Cons append-Nil in-set-conv-decomp-first*)
with *True assms* **have** $y \in set\ us$

by (*fastforce simp add*: *sorted-wrt-append*)
then obtain *ys zs* **where** *yz*: $l = xs \mathbin{@} [x] \mathbin{@} ys \mathbin{@} [y] \mathbin{@} zs$
by (*metis split-list us append-Cons append-Nil*)
have *sublist* $[x, y]$ $l \longleftrightarrow ys = []$
using *sorted yz*
apply (*simp add*: *sublist-def sorted-wrt-append*)
by (*metis* (*mono-tags*, *opaque-lifting*) *append-Cons-eq-iff append-Nil assms sorted-wrt.simps*(*2*)
sorted-wrt-append less-irrefl)
also have $\ldots = (\forall z \in set\ l.\ z \leq x \vee z \geq y)$
using *sorted yz*
apply (*simp add*: *sublist-def sorted-wrt-append*)
by (*metis Un-iff empty-iff less-le-not-le list.exhaust list.set*(*1*) *list.set-intros*(*1*))
finally show *?thesis*
using *True* **by** *blast*
next
case *False*
then show *?thesis*
by (*metis list.set-intros*(*1*) *set-subset-Cons sorted-wrt.simps*(*2*) *sorted-wrt-append sublist-def*
set-mono-sublist sorted subset-iff)
qed

lemma *quotient-of-rat-of-int* [*simp*]: *quotient-of* (*rat-of-int i*) = (*i*, *1*)
using *Rat.of-int-def quotient-of-int* **by** *force*

lemma *quotient-of-rat-of-nat* [*simp*]: *quotient-of* (*rat-of-nat i*) = (*int i*, *1*)
by (*metis of-int-of-nat-eq quotient-of-rat-of-int*)

lemma *int-div-le-self*:
‹*x div k* ≤ *x*› **if** ‹*0* < *x*› **for** *x k* :: *int*
by (*metis div-by-1 int-div-less-self less-le-not-le nle-le nonneg1-imp-zdiv-pos-iff order.trans that*)

lemma *transp-add1-int*:
assumes ⋀*n*::*int*. *R* (*f n*) (*f* (*1* + *n*))
and $n < n'$
and *transp R*
shows *R* (*f n*) (*f n'*)
proof −
have *R* (*f n*) (*f* (*1* + *n* + *int k*)) **for** *k*
by (*induction k*) (*use assms* **in** ‹*auto elim*!: *transpE*›)
then show *?thesis*
by (*metis add.commute assms*(*2*) *zle-iff-zadd zless-imp-add1-zle*)
qed

lemma *refl-transp-add1-int*:
assumes $\bigwedge n{::}int.\ R\ (f\ n)\ (f\ (1\ +\ n))$
and $n \leq n'$
and *reflp R transp R*
shows $R\ (f\ n)\ (f\ n')$
by (*metis assms order-le-less reflpE transp-add1-int*)

lemma *transp-Suc*:
assumes $\bigwedge n.\ R\ (f\ n)\ (f\ (Suc\ n))$
and $n < n'$
and *transp R*
shows $R\ (f\ n)\ (f\ n')$
proof −
have $R\ (f\ n)\ (f\ (1\ +\ n\ +\ k))$ **for** k
by (*induction k*) (*use assms* **in** ‹*auto elim*!: *transpE*›)
then show *?thesis*
by (*metis add-Suc-right add-Suc-shift assms*(*2*) *less-natE plus-1-eq-Suc*)
qed

lemma *refl-transp-Suc*:
assumes $\bigwedge n.\ R\ (f\ n)\ (f\ (Suc\ n))$
and $n \leq n'$
and *reflp R transp R*
shows $R\ (f\ n)\ (f\ n')$
by (*metis assms dual-order.order-iff-strict reflpE transp-Suc*)

lemma *sorted-subset-imp-subseq*:
fixes *xs* :: *'a*::*order list*
assumes *set xs* ⊆ *set ys sorted-wrt* (<) *xs sorted-wrt* (≤) *ys*
shows *subseq xs ys*
using *assms*
proof (*induction xs arbitrary*: *ys*)
case *Nil*
then show *?case*
by *auto*
next
case (*Cons x xs*)
then have $x \in$ *set ys*
by *auto*
then obtain *us vs* **where** §: $ys = us\ @\ [x]\ @\ vs$
by (*metis append.left-neutral append-eq-Cons-conv split-list*)
moreover
have *set xs* ⊆ *set vs*
using *Cons.prems* **by** (*fastforce simp*: § *sorted-wrt-append*)
with *Cons* **have** *subseq xs vs*
by (*metis* § *sorted-wrt.simps*(*2*) *sorted-wrt-append*)
ultimately show *?case*

by *auto*
qed

lemma *coprime-unimodular-int*:
fixes *a b*::*int*
assumes *coprime a b a>1 b>1*
obtains *x y* **where** $a*x - b*y = 1$ $0 < x$ $x < b$ $0 < y$ $y < a$
proof −
obtain *u v* **where** *1*: $a * u + b * v = 1$
by (*metis* ‹*coprime a b*› *cong-iff-lin coprime-iff-invertible-int*)
define *k* **where** $k \equiv u \ div \ b$
define *x* **where** $x \equiv u - k*b$
define *y* **where** $y \equiv -(v + k*a)$
show *thesis*
proof
show ∗: $a * x - b * y = 1$
using *1* **by** (*simp add*: *x-def y-def algebra-simps*)
have $u \neq k*b$ *b>0*
using *assms* ∗ **by** (*auto simp*: *k-def x-def y-def zmult-eq-neg1-iff*)
moreover have $u - k*b \geq 0$
by (*simp add*: *k-def* ‹*b>0*› *minus-div-mult-eq-mod*)
ultimately show $x > 0$
by (*fastforce simp*: *x-def*)
show $x < b$
by (*simp add*: ‹*0 < b*› *k-def minus-div-mult-eq-mod x-def*)
have $a*x > 1$
by (*metis* ‹*0 < x*› ‹*a>1*› *int-one-le-iff-zero-less less-1-mult linorder-not-less mult.right-neutral nle-le*)
then have ¬ $b * y \leq 0$
using ∗ **by** *linarith*
then show $y > 0$
by (*simp add*: ‹*0 < b*› *mult-less-0-iff order-le-less*)
show $y < a$
using ∗ ‹*0 < x*› ‹*x < b*›
by (*smt* (*verit*, *best*) ‹*a>1*› *mult.commute mult-less-le-imp-less*)
qed
qed

1.3 Farey Fractions

type-synonym *farey* = *rat*

definition *num-farey* :: *farey* ⇒ *int*
where *num-farey* ≡ λ*x*. *fst* (*quotient-of x*)

definition *denom-farey* :: *farey* ⇒ *int*
where *denom-farey* ≡ λ*x*. *snd* (*quotient-of x*)

definition *farey* :: [*int*,*int*] ⇒ *farey*

where *farey* $\equiv \lambda a\ b.\ max\ 0\ (min\ 1\ (Fract\ a\ b))$

lemma *farey01* [*simp*]: $0 \leq farey\ a\ b\ farey\ a\ b \leq 1$
by (*auto simp*: *min-def max-def farey-def*)

lemma *farey-0* [*simp*]: $farey\ 0\ n = 0$
by (*simp add*: *farey-def rat-number-collapse*)

lemma *farey-1* [*simp*]: $farey\ 1\ 1 = 1$
by (*auto simp*: *farey-def rat-number-expand*)

lemma *num-farey-nonneg*: $x \in \{0..1\} \Longrightarrow num\text{-}farey\ x \geq 0$
by (*cases x*) (*simp add*: *num-farey-def quotient-of-Fract zero-le-Fract-iff*)

lemma *num-farey-le-denom*: $x \in \{0..1\} \Longrightarrow num\text{-}farey\ x \leq denom\text{-}farey\ x$
by (*cases x*) (*simp add*: *Fract-le-one-iff denom-farey-def num-farey-def quotient-of-Fract*)

lemma *denom-farey-pos*: $denom\text{-}farey\ x > 0$
by (*simp add*: *denom-farey-def quotient-of-denom-pos'*)

lemma *coprime-num-denom-farey* [*intro*]: *coprime* (*num-farey x*) (*denom-farey x*)
by (*simp add*: *denom-farey-def num-farey-def quotient-of-coprime*)

lemma *rat-of-farey-conv-num-denom*:
$x = rat\text{-}of\text{-}int\ (num\text{-}farey\ x)\ /\ rat\text{-}of\text{-}int\ (denom\text{-}farey\ x)$
by (*simp add*: *denom-farey-def num-farey-def quotient-of-div*)

lemma *num-denom-farey-eqI*:
assumes $x = of\text{-}int\ a\ /\ of\text{-}int\ b\ b > 0$ *coprime a b*
shows $num\text{-}farey\ x = a\ denom\text{-}farey\ x = b$
using *Fract-of-int-quotient assms quotient-of-Fract*
by (*auto simp*: *num-farey-def denom-farey-def*)

lemma *farey-cases* [*cases type*, *case-names farey*]:
assumes $x \in \{0..1\}$
obtains *a b* **where** $0 \leq a\ a \leq b$ *coprime a b* $x = Fract\ a\ b$
by (*metis Fract-of-int-quotient Rat-cases assms num-denom-farey-eqI num-farey-le-denom num-farey-nonneg*)

lemma *rat-of-farey*: $[\![x = of\text{-}int\ a\ /\ of\text{-}int\ b;\ x \in \{0..1\}]\!] \Longrightarrow x = farey\ a\ b$
by (*simp add*: *Fract-of-int-quotient farey-def max-def*)

lemma *farey-num-denom-eq* [*simp*]: $x \in \{0..1\} \Longrightarrow farey\ (num\text{-}farey\ x)\ (denom\text{-}farey\ x) = x$
using *rat-of-farey rat-of-farey-conv-num-denom* **by** *fastforce*

lemma *farey-eqI*:
assumes $num\text{-}farey\ x = num\text{-}farey\ y\ denom\text{-}farey\ x = denom\text{-}farey\ y$
shows $x = y$

by (*metis Rat.of-int-def assms rat-of-farey-conv-num-denom*)

lemma
assumes *coprime a b* $0 \leq a$ $a < b$
shows *num-farey-eq* [*simp*]: *num-farey* (*farey a b*) = *a*
and *denom-farey-eq* [*simp*]: *denom-farey* (*farey a b*) = *b*
using *Fract-less-one-iff quotient-of-Fract zero-le-Fract-iff*
using *assms num-denom-farey-eqI rat-of-farey* **by** *force+*

lemma
assumes $0 \leq a$ $a \leq b$ $0 < b$
shows *num-farey*: *num-farey* (*farey a b*) = *a div* (*gcd a b*)
and *denom-farey*: *denom-farey* (*farey a b*) = *b div* (*gcd a b*)
proof −
have $0 \leq$ *Fract a b Fract a b* ≤ 1
using *assms* **by** (*auto simp*: *Fract-le-one-iff zero-le-Fract-iff*)
with *assms* **show** *num-farey* (*farey a b*) = *a div* (*gcd a b*) *denom-farey* (*farey a b*) = *b div* (*gcd a b*)
by (*auto simp*: *num-farey-def denom-farey-def farey-def quotient-of-Fract Rat.normalize-def Let-def*)
qed

lemma
assumes *coprime a b* $0 < b$
shows *num-farey-Fract* [*simp*]: *num-farey* (*Fract a b*) = *a*
and *denom-farey-Fract* [*simp*]: *denom-farey* (*Fract a b*) = *b*
using *Fract-of-int-quotient assms num-denom-farey-eqI* **by** *force+*

lemma *num-farey-0* [*simp*]: *num-farey 0* = *0*
and *denom-farey-0* [*simp*]: *denom-farey 0* = *1*
and *num-farey-1* [*simp*]: *num-farey 1* = *1*
and *denom-farey-1* [*simp*]: *denom-farey 1* = *1*
by (*auto simp*: *num-farey-def denom-farey-def*)

lemma *num-farey-neq-denom*: *denom-farey x* $\neq$ *1* $\Longrightarrow$ *num-farey x* $\neq$ *denom-farey x*
by (*metis denom-farey-0 div-0 div-self num-farey-1 rat-of-farey-conv-num-denom*)

lemma *num-farey-0-iff* [*simp*]: *num-farey x* = *0* $\longleftrightarrow$ *x* = *0*
unfolding *num-farey-def*
by (*metis div-0 eq-fst-iff of-int-0 quotient-of-div rat-zero-code*)

lemma *denom-farey-le1-cases*:
assumes *denom-farey x* $\leq$ *1* *x* $\in \{0..1\}$
shows *x* = *0* $\vee$ *x* = *1*
proof −
consider *num-farey x* = *0* | *num-farey x* = *1 denom-farey x* = *1*
using *assms num-farey-le-denom* [*of x*] *num-farey-nonneg* [*of x*] **by** *linarith*
then show *?thesis*

by (*metis denom-farey-1 farey-eqI num-farey-0-iff num-farey-1*)
qed

definition *mediant* :: *farey* ⇒ *farey* ⇒ *farey* **where**
mediant ≡ λ*x y*. *Fract* (*fst* (*quotient-of x*) + *fst* (*quotient-of y*))
(*snd* (*quotient-of x*) + *snd* (*quotient-of y*))

lemma *mediant-eq-Fract*:
mediant x y = *Fract* (*num-farey x* + *num-farey y*) (*denom-farey x* + *denom-farey y*)
by (*simp add*: *denom-farey-def num-farey-def mediant-def*)

lemma *mediant-eq-farey*:
assumes *x* ∈ {*0..1*} *y* ∈ {*0..1*}
shows *mediant x y* = *farey* (*num-farey x* + *num-farey y*) (*denom-farey x* + *denom-farey y*)
proof −
have *0* ≤ *num-farey x* + *num-farey y*
using *assms num-farey-nonneg* **by** *auto*
moreover have *num-farey x* + *num-farey y* ≤ *denom-farey x* + *denom-farey y*
by (*meson add-mono assms num-farey-le-denom*)
ultimately show *?thesis*
by (*simp add*: *add-pos-pos denom-farey-pos Fract-of-int-quotient rat-of-farey mediant-eq-Fract*)
qed

definition *farey-unimodular* :: *farey* ⇒ *farey* ⇒ *bool* **where**
farey-unimodular x y ⟷
denom-farey x ∗ *num-farey y* − *num-farey x* ∗ *denom-farey y* = *1*

lemma *farey-unimodular-imp-less*:
assumes *farey-unimodular x y*
shows *x* < *y*
using *assms*
by (*auto simp*: *farey-unimodular-def rat-less-code denom-farey-def num-farey-def*)

lemma *denom-mediant*: *denom-farey* (*mediant x y*) ≤ *denom-farey x* + *denom-farey y*
using *quotient-of-denom-pos'* [*of x*] *quotient-of-denom-pos'* [*of y*]
by (*simp add*: *mediant-eq-Fract denom-farey-def num-farey-def quotient-of-Fract normalize-def Let-def int-div-le-self*)

lemma *unimodular-imp-both-coprime*:
fixes *a*:: *'a*::{*algebraic-semidom*,*comm-ring-1*}
assumes *b*∗*c* − *a*∗*d* = *1*
shows *coprime a b coprime c d*
using *mult.commute* **by** (*metis assms coprimeI dvd-diff dvd-mult2*)+

lemma *unimodular-imp-coprime*:
fixes *a*:: *'a*::{*algebraic-semidom*,*comm-ring-1*}
assumes *b*∗*c* − *a*∗*d* = *1*
shows *coprime* (*a*+*c*) (*b*+*d*)
proof (*rule coprimeI*)
fix *k*
assume *k*: *k dvd* (*a*+*c*) *k dvd* (*b*+*d*)
moreover have (*b*+*d*)∗*c* = (*a*+*c*)∗*d* + *1*
using *assms* **by** (*simp add*: *algebra-simps*)
ultimately show *is-unit k*
by (*metis add-diff-cancel-left' dvd-diff dvd-mult2*)
qed

definition *fareys* :: *int* ⇒ *rat list*
where *fareys n* ≡ *sorted-list-of-set* {*x* ∈ {*0..1*}. *denom-farey x* ≤ *n*}

lemma *strict-sorted-fareys*: *sorted-wrt* (<) (*fareys n*)
by (*simp add*: *fareys-def*)

lemma *farey-set-UN-farey*: {*x* ∈ {*0..1*}. *denom-farey x* ≤ *n*} = (⋃ *b* ∈ {*1..n*}.
⋃ *a* ∈ {*0..b*}. {*farey a b*})
proof −
have ∃ *b* ∈ {*1..n*}. ∃ *a* ∈ {*0..b*}. *x* = *farey a b*
if *denom-farey x* ≤ *n x* ∈ {*0..1*} **for** *x* :: *farey*
unfolding *Bex-def*
using *that denom-farey-pos int-one-le-iff-zero-less num-farey-le-denom num-farey-nonneg*
by *fastforce*
moreover have ⋀*b a*::*int*. *0* ≤ *b* ⟹ *b div gcd a b* ≤ *b*
by (*metis div-0 int-div-le-self nless-le*)
ultimately show *?thesis*
by (*auto simp*: *denom-farey*) (*meson order-trans*)
qed

lemma *farey-set-UN-farey'*: {*x* ∈ {*0..1*}. *denom-farey x* ≤ *n*} = (⋃ *b* ∈ {*1..n*}.
⋃ *a* ∈ {*0..b*}. *if coprime a b then* {*farey a b*} *else* {})
proof −
have ∃ *aa ba*. *farey a b* = *farey aa ba* ∧ *0* ≤ *aa* ∧ *aa* ≤ *ba* ∧ *1* ≤ *ba* ∧ *ba* ≤ *n*
∧ *coprime aa ba*
if *1* ≤ *b* **and** *b* ≤ *n* **and** *0* ≤ *a* **and** *a* ≤ *b* **for** *a b*
proof −
let *?a* = *a div gcd a b*
let *?b* = *b div gcd a b*
have *coprime ?a ?b*
by (*metis div-gcd-coprime not-one-le-zero* ‹*b*≥*1*›)
moreover have *farey a b* = *farey ?a ?b*
using *Fract-coprime farey-def* **by** *presburger*
moreover have *?a* ≤ *?b* ∧ *?b* ≤ *n*
by (*smt* (*verit*, *best*) *gcd-pos-int int-div-le-self that zdiv-mono1*)
ultimately show *?thesis*

using *that* **by** (*metis denom-farey denom-farey-pos div-int-pos-iff gcd-ge-0-int int-one-le-iff-zero-less*)
qed
then show *?thesis*
unfolding *farey-set-UN-farey*
by (*fastforce split*: *if-splits*)
qed

lemma *farey-set-UN-Fract*: $\{x \in \{0..1\}.\ \textit{denom-farey}\ x \leq n\} = (\bigcup b \in \{1..n\}.\ \bigcup a \in \{0..b\}.\ \{\textit{Fract}\ a\ b\})$
unfolding *farey-set-UN-farey*
by (*simp add*: *Fract-of-int-quotient farey-def*)

lemma *farey-set-UN-Fract'*: $\{x \in \{0..1\}.\ \textit{denom-farey}\ x \leq n\} = (\bigcup b \in \{1..n\}.\ \bigcup a \in \{0..b\}.\ \textit{if coprime}\ a\ b\ \textit{then}\ \{\textit{Fract}\ a\ b\}\ \textit{else}\ \{\})$
unfolding *farey-set-UN-farey'*
by (*simp add*: *Fract-of-int-quotient farey-def*)

lemma *finite-farey-set*: *finite* $\{x \in \{0..1\}.\ \textit{denom-farey}\ x \leq n\}$
unfolding *farey-set-UN-farey* **by** *blast*

lemma *denom-in-fareys-iff*: $x \in \textit{set}\ (\textit{fareys}\ n) \longleftrightarrow \textit{denom-farey}\ x \leq \textit{int}\ n \wedge x \in \{0..1\}$
using *finite-farey-set* **by** (*auto simp*: *fareys-def*)

lemma *denom-fareys-leI*: $x \in \textit{set}\ (\textit{fareys}\ n) \Longrightarrow \textit{denom-farey}\ x \leq n$
using *finite-farey-set* **by** (*auto simp*: *fareys-def*)

lemma *denom-fareys-leD*: $[\![\textit{denom-farey}\ x \leq \textit{int}\ n; x \in \{0..1\}]\!] \Longrightarrow x \in \textit{set}\ (\textit{fareys}\ n)$
using *denom-in-fareys-iff* **by** *blast*

lemma *fareys-increasing-1*: $\textit{set}\ (\textit{fareys}\ n) \subseteq \textit{set}\ (\textit{fareys}\ (1 + n))$
using *farey-set-UN-farey* **by** (*force simp*: *fareys-def*)

definition *fareys-new* :: $\textit{int} \Rightarrow \textit{rat set}$ **where**
$\textit{fareys-new}\ n \equiv \{\textit{Fract}\ a\ n |\ a.\ \textit{coprime}\ a\ n \wedge a \in \{0..n\}\}$

lemma *fareys-new-0* [*simp*]: $\textit{fareys-new}\ 0 = \{\}$
by (*auto simp*: *fareys-new-def*)

lemma *fareys-new-1* [*simp*]: $\textit{fareys-new}\ 1 = \{0,1\}$
proof −
have $\textit{Fract}\ a\ 1 = 1$
if *a*: $\textit{Fract}\ a\ 1 \neq 0\ 0 \leq a\ a \leq 1$ **for** *a*
by (*metis One-rat-def int-one-le-iff-zero-less nless-le order-antisym-conv rat-number-collapse*(*1*) *that*)
moreover have $\exists a.\ \textit{Fract}\ a\ 1 = 0 \wedge 0 \leq a \wedge a \leq 1$
using *rat-number-expand*(*1*) **by** *auto*

```
  moreover have ∃ a. Fract a 1 = 1 ∧ 0 ≤ a ∧ a ≤ 1
    using One-rat-def by fastforce
  ultimately show ?thesis
    by (auto simp: fareys-new-def)
qed

lemma fareys-new-not01:
  assumes n>1
  shows 0 ∉ (fareys-new n) 1 ∉ (fareys-new n)
  using assms by (simp-all add: Fract-of-int-quotient fareys-new-def)

lemma inj-num-farey: inj-on num-farey (fareys-new n)
proof (cases n=1)
  case True
  then show ?thesis
    using fareys-new-1 by auto
next
  case False
  then show ?thesis
  proof −
    have Fract a n = Fract a' n
      if coprime a n 0 ≤ a a ≤ n
        and coprime a' n 0 ≤ a' a' ≤ n
        and num-farey (Fract a n) = num-farey (Fract a' n)
      for a a'
    proof −
      from that
      obtain a < n a' < n
        using False by force+
      with that show ?thesis
        by auto
    qed
    with False show ?thesis
      by (auto simp: inj-on-def fareys-new-def)
  qed
qed

lemma finite-fareys-new [simp]: finite (fareys-new n)
  by (auto simp: fareys-new-def)

lemma card-fareys-new:
  assumes n > 1
  shows card (fareys-new (int n)) = totient n
proof −
  have bij-betw num-farey (fareys-new n) (int ‘ totatives n)
  proof −
    have ∃ a>0. a ≤ n ∧ coprime a n ∧ num-farey x = int a
      if x: x ∈ fareys-new n for x
    proof −
```

```
    obtain a where a: x = Fract a (int n) coprime a (int n) 0 ≤ a a ≤ int n
      using x by (auto simp: fareys-new-def)
    then have a > 0
      using assms less-le-not-le by fastforce
    moreover have coprime (nat a) n
      by (metis a(2,3) coprime-int-iff nat-0-le)
    ultimately have num-farey x = int (nat a)
      using a num-farey by auto
    then show ?thesis
      using ‹0 < a› ‹coprime (nat a) n› a(4) nat-le-iff zero-less-nat-eq by blast
  qed
  moreover have ∃x∈fareys-new n. int a = num-farey x
    if 0 < a a ≤ n coprime a n for a
  proof −
    have §: coprime (int a) (int n) 0 ≤ (int a) (int a) ≤ int n
      using that by auto
    then have Fract (int a) (int n) = Fract (int a) (int n)
      using Fract-of-int-quotient assms rat-of-farey by auto
    with § have Fract (int a) (int n) ∈ fareys-new n
      by (auto simp: fareys-new-def)
    then have int a = num-farey (Fract (int a) n)
      using ‹coprime (int a) (int n)› assms by auto
    then show ?thesis
      using ‹Fract (int a) (int n) ∈ fareys-new (int n)› by blast
  qed
  ultimately show ?thesis
      by (auto simp add: totatives-def bij-betw-def inj-num-farey comp-inj-on
image-iff)
 qed
 then show ?thesis
   unfolding totient-def by (metis bij-betw-same-card bij-betw-of-nat)
qed

lemma disjoint-fareys-plus1:
 assumes n > 0
 shows disjnt (set (fareys n)) (fareys-new (1 + n))
proof −
 have False
   if §: 0 ≤ a a ≤ 1 + n coprime a (1 + n)
     1 ≤ d d ≤ n Fract a (1 + n) = Fract c d 0 ≤ c c ≤ d coprime c d
   for a c d
 proof (cases c<d)
   case True
   have alen: a ≤ n
     using nle-le that by fastforce
   have d = denom-farey (Fract c d)
     using that by force
   also have ... = 1 + n
     using denom-farey-Fract that by fastforce
```

```
    finally show False
      using that(5) by fastforce
  next
    case False
    with ‹c ≤ d› have c=d by auto
    with that have d=1 by force
    with that have 1 + n = 1
        by (metis One-rat-def ‹c = d› assms denom-farey-1 denom-farey-Fract
le-imp-0-less
         order-less-le)
    then show ?thesis
      using ‹c = d› § assms by auto
  qed
  then show ?thesis
  unfolding fareys-def farey-set-UN-Fract' fareys-new-def disjnt-iff
    by auto
qed


lemma set-fareys-plus1: set (fareys (1 + n)) = set (fareys n) ∪ fareys-new (1 +
n)
proof −
  have ∃ b≥1. b ≤ n ∧ (∃ a≥0. a ≤ b ∧ coprime a b ∧ Fract c d = Fract a b)
    if Fract c d ∉ fareys-new (1 + n)
      and coprime c d 1 ≤ d d ≤ 1 + n 0 ≤ c c ≤ d
    for c d
  proof (cases d = 1 + n)
    case True
    with that show ?thesis
      by (auto simp: fareys-new-def)
  qed (use that in auto)
  moreover have ∃ d≥1. d ≤ 1 + n ∧ (∃ c≥0. c ≤ d ∧ coprime c d ∧ x = Fract
c d)
    if x ∈ fareys-new (1 + n) for x
    using that nle-le by (fastforce simp add: fareys-new-def)
  ultimately show ?thesis
    unfolding fareys-def farey-set-UN-Fract' by fastforce
qed

lemma length-fareys-Suc:
  assumes n>0
  shows length (fareys (1 + int n)) = length (fareys n) + totient (Suc n)
proof −
  have length (fareys (1 + int n)) = card (set (fareys (1 + int n)))
   by (metis fareys-def finite-farey-set sorted-list-of-set.sorted-key-list-of-set-unique)
  also have ... = card (set (fareys n)) + card (fareys-new(1 + int n))
   using disjoint-fareys-plus1 assms by (simp add: set-fareys-plus1 card-Un-disjnt)
  also have ... = card (set (fareys n)) + totient (Suc n)
    using assms card-fareys-new by force
```

```
  also have ... = length (fareys n) + totient (Suc n)
    using fareys-def finite-farey-set by auto
  finally show ?thesis .
qed


lemma fareys-0 [simp]: fareys 0 = []
  unfolding fareys-def farey-set-UN-farey
  by simp

lemma fareys-1 [simp]: fareys 1 = [0, 1]
proof −
  have {x ∈ {0..1}. denom-farey x ≤ 1} = {0,1}
    using denom-farey-le1-cases by auto
  then show ?thesis
    by (simp add: fareys-def)
qed

lemma fareys-2 [simp]: fareys 2 = [0, farey 1 2, 1]
proof −
  have §: denom-farey x ≤ 2 ⟷ denom-farey x = 1 ∨ denom-farey x = 2 for x
    using denom-farey-pos [of x] by auto
  have {x ∈ {0..1}. denom-farey x ≤ 2} = {farey 0 1, farey 1 2, farey 1 1}
  proof −
    have x = farey 1 1
      if x ≠ farey 0 1 x ∈ {0..1} denom-farey x = 1 for x
      using that denom-farey-le1-cases order.eq-iff rat-of-farey by auto
    moreover have False
      if x ≠ farey 0 1 x ≠ farey 1 2 denom-farey x = 2 x ∈ {0..1} for x
      using that num-farey-neq-denom
    by (smt (verit) farey-0 farey-num-denom-eq num-farey-le-denom num-farey-nonneg)
    moreover have denom-farey (farey 1 1) = 1
      by (simp add: Fract-of-int-quotient farey-def)
    ultimately show ?thesis
      by (auto simp: farey-set-UN-farey §)
  qed
  also have ... = {0, 1/2, 1::rat}
    by (simp add: farey-def Fract-of-int-quotient)
  finally show ?thesis
    by (simp add: fareys-def farey-def Fract-of-int-quotient)
qed

lemma length-fareys-1:
  shows length (fareys 1) = 1 + totient 1
  by simp

lemma length-fareys: n>0 ⟹ length (fareys n) = 1 + (∑ k=1..n. totient k)
proof (induction n)
  case (Suc n)
```

then show *?case*
by (*cases n=0*) (*auto simp add*: *length-fareys-Suc*)
qed *auto*

lemma *subseq-fareys-1*: *subseq* (*fareys n*) (*fareys* (*1 + n*))
by (*metis fareys-increasing-1 strict-sorted-fareys sorted-subset-imp-subseq strict-sorted-imp-sorted*)

lemma *monotone-fareys*: *monotone* (≤) *subseq fareys*
using *refl-transp-add1-int* [*OF - - subseq-order.reflp-on-le subseq-order.transp-on-le*]
by (*metis monotoneI subseq-fareys-1*)

lemma *farey-unimodular-0-1* [*simp*, *intro*]: *farey-unimodular 0 1*
by (*auto simp*: *farey-unimodular-def*)

Apostol's Theorem 5.2 for integers

lemma *mediant-lies-betw-int*:
fixes *a b c d*::*int*
assumes *rat-of-int a / of-int b < of-int c / of-int d b>0 d>0*
shows *rat-of-int a / of-int b < (of-int a + of-int c) / (of-int b + of-int d)*
(rat-of-int a + of-int c) / (of-int b + of-int d) < of-int c / of-int d
using *assms* **by** (*simp-all add*: *field-split-simps*)

Apostol's Theorem 5.2

theorem *mediant-inbetween*:
fixes *x y*::*farey*
assumes *x < y*
shows *x < mediant x y mediant x y < y*
using *assms mediant-lies-betw-int Fract-of-int-quotient*
by (*metis denom-farey-pos mediant-eq-Fract of-int-add rat-of-farey-conv-num-denom*)+

lemma *sublist-fareysD*:
assumes *sublist* [*x*,*y*] (*fareys n*)
obtains *x* ∈ *set* (*fareys n*) *y* ∈ *set* (*fareys n*)
by (*meson assms list.set-intros set-mono-sublist subsetD*)

Adding the denominators of two consecutive Farey fractions

lemma *sublist-fareys-add-denoms*:
fixes *a b c d*::*int*
defines *x* ≡ *Fract a b*
defines *y* ≡ *Fract c d*
assumes *sub*: *sublist* [*x*,*y*] (*fareys* (*int n*)) **and** *b>0 d>0 coprime a b coprime c d*
shows *b + d > n*
proof (*rule ccontr*)
have §: *x < y* ∀*z* ∈ *set* (*fareys n*). *z* ≤ *x* ∨ *z* ≥ *y*
using *sorted-two-sublist strict-sorted-fareys sub* **by** *blast*+
assume ¬ *int n < b + d*
with *assms* **have** *denom-farey* (*mediant x y*) ≤ *int n*
by (*metis denom-farey-Fract denom-mediant dual-order.trans leI*)

then have *mediant x y ∈ set (fareys n)*
by (*metis sub atLeastAtMost-iff denom-in-fareys-iff farey01 mediant-eq-farey sublist-fareysD*)
moreover have *x < mediant x y mediant x y < y*
by (*simp-all add*: *mediant-inbetween ‹x < y›*)
ultimately show *False*
using § *x-def y-def* **by** *fastforce*
qed

1.4 Apostol's Theorems 5.3–5.5

theorem *consec-subset-fareys*:
fixes *a b c d*::*int*
assumes *abcd*: *0 ≤ Fract a b Fract a b < Fract c d Fract c d ≤ 1*
and *consec*: *b∗c − a∗d = 1*
and *max*: *max b d ≤ n n < b+d*
and *b>0*
shows *sublist* [*Fract a b*, *Fract c d*] (*fareys n*)
proof (*rule ccontr*)
assume *con*: *¬ ?thesis*
have *d > 0*
using *max* **by** *force*
have *coprime a b coprime c d*
using *consec unimodular-imp-both-coprime* **by** *blast+*
with *‹b > 0› ‹d > 0›* **have** *denom-farey* (*Fract a b*) = *b denom-farey* (*Fract c d*) = *d*
by *auto*
moreover have *b≤n d≤n*
using *max* **by** *auto*
ultimately have *ab*: *Fract a b ∈ set* (*fareys n*) **and** *cd*: *Fract c d ∈ set* (*fareys n*)
using *abcd finite-farey-set* **by** (*auto simp*: *fareys-def*)
then obtain *xs us* **where** *us*: *fareys n = xs @* [*Fract a b*] *@ us*
using *abcd* **by** (*metis append-Cons append-Nil split-list*)
have *Fract c d ∈ set us*
using *abcd cd strict-sorted-fareys* [*of n*]
by (*fastforce simp add*: *us sorted-wrt-append*)
then obtain *ys zs* **where** *yz*: *fareys n = xs @* [*Fract a b*] *@ ys @* [*Fract c d*] *@ zs*
by (*metis split-list us append-Cons append-Nil*)
with *con* **have** *ys ≠* []
by (*metis Cons-eq-append-conv sublist-appendI*)
then obtain *h k* **where** *hk*: *coprime h k Fract h k ∈ set ys k>0*
by (*metis Rat-cases list.set-sel*(*1*))
then have *hk-fareys*: *Fract h k ∈ set* (*fareys n*)
by (*auto simp*: *yz*)
have *less*: *Fract a b < Fract h k Fract h k < Fract c d*
using *hk strict-sorted-fareys* [*of n*] **by** (*auto simp add*: *yz sorted-wrt-append*)
with *‹b > 0› ‹d > 0› hk* **have** ∗: *k∗a < h∗b d∗h < c∗k*

by (*simp-all add*: *Fract-of-int-quotient mult.commute divide-simps flip*: *of-int-mult*)
have $k \leq n$
using *hk* **by** (*metis hk-fareys denom-fareys-leI denom-farey-Fract*)
have $k = (b*c - a*d)*k$
by (*simp add*: *consec*)
also have $\ldots = b*(c*k - d*h) + d*(b*h - a*k)$
by (*simp add*: *algebra-simps*)
finally have *k*: $k = b * (c * k - d * h) + d * (b * h - a * k)$ **.**
moreover have $c*k - d*h \geq 1$ $b*h - a*k \geq 1$
using ‹$b > 0$› ‹$d > 0$› ∗ **by** (*auto simp*: *mult.commute*)
ultimately have $b * (c * k - d * h) + d * (b * h - a * k) \geq b+d$
by (*metis* ‹$b > 0$› ‹$d > 0$› *add-mono mult.right-neutral mult-left-mono order-le-less*)
then show *False*
using ‹$k \leq n$› *max k* **by** *force*
qed

lemma *farey-unimodular-mediant*:
assumes *farey-unimodular x y*
shows *farey-unimodular x* (*mediant x y*) *farey-unimodular* (*mediant x y*) *y*
using *assms quotient-of-denom-pos'* [*of x*] *quotient-of-denom-pos'* [*of y*]
unfolding *farey-unimodular-def*
by (*auto simp*: *mediant-eq-Fract denom-farey-def num-farey-def quotient-of-Fract unimodular-imp-coprime algebra-simps*)

Apostol's Theorem 5.4

theorem *mediant-unimodular*:
fixes *a b c d*::*int*
assumes *abcd*: $0 \leq$ *Fract a b Fract a b* $<$ *Fract c d Fract c d* ≤ 1
and *consec*: $b*c - a*d = 1$
and *0*: $b>0$ $d>0$
defines $h \equiv a+c$
defines $k \equiv b+d$
obtains *Fract a b* $<$ *Fract h k Fract h k* $<$ *Fract c d coprime h k*
$b*h - a*k = 1$ $c*k - d*h = 1$
proof
show *Fract a b* $<$ *Fract h k Fract h k* $<$ *Fract c d*
using *abcd 0*
by (*simp-all add*: *Fract-of-int-quotient h-def k-def distrib-left distrib-right divide-simps*)
show *coprime h k*
by (*simp add*: *consec unimodular-imp-coprime h-def k-def*)
show $b * h - a * k = 1$
by (*simp add*: *consec distrib-left h-def k-def*)
show $c * k - d * h = 1$
by (*simp add*: *consec h-def distrib-left k-def mult.commute*)
qed

Apostol's Theorem 5.5, first part: "Each fraction in F $(n + 1)$ which is not in F n is the mediant of a pair of consecutive fractions in F n

lemma *get-consecutive-parents*:

```
  fixes m n::int
  assumes coprime m n 0<m m<n
  obtains a b c d where m = a+c n = b+d b*c - a*d = 1 a≥0 b>0 c>0 d>0
a<b c≤d
proof (cases m=1)
  case True
  show ?thesis
  proof
    show m = 0 + 1 n = 1 + (n-1)
      by (auto simp: True)
  qed (use True ‹m<n› in auto)
next
  case False
  then obtain d c where *: n*c - m*d = 1 0 < d d < n 0 < c c < m
    using coprime-unimodular-int
 [of n m] coprime-commute assms by (smt (verit) coprime-commute)
  then have **: n * (c - d) + (n - m) * d = 1
    by (metis mult-diff-mult)
  show ?thesis
  proof
    show c≤d
      using * ** ‹m<n› by (smt (verit) mult-le-0-iff)
    show (n-d) * c - (m-c) * d = 1
      using * by (simp add: algebra-simps)
    with * ‹m<n› show m-c < n-d
      by (smt (verit, best) mult-mono)
  qed (use * in auto)
qed

theorem fareys-new-eq-mediant:
  assumes x ∈ fareys-new n n>1
  obtains a b c d where
    sublist [Fract a b, Fract c d] (fareys (n-1))
    x = mediant (Fract a b) (Fract c d) coprime a b coprime c d a≥0 b>0 c>0
d>0
proof -
  obtain m where m: coprime m n 0 ≤ m m ≤ n x = Fract m n
    using assms nless-le zero-less-imp-eq-int by (force simp: fareys-new-def)
  moreover
  have x ≠ 0 x ≠ 1
    using assms fareys-new-not01 by auto
  with m have 0<m m<n
    using ‹n>1› of-nat-le-0-iff by fastforce+
  ultimately
  obtain a b c d where
    abcd: m = a+c n = b+d b*c - a*d = 1 a≥0 b>0 c>0 d>0 a<b c≤d
    by (metis get-consecutive-parents)
  show thesis
  proof
```

```
    have Fract a b < Fract c d
      using abcd mult.commute[of b c] by force
    with consec-subset-fareys
    show sublist [Fract a b, Fract c d] (fareys (n−1))
      using Fract-le-one-iff abcd zero-le-Fract-iff by auto
    show x = mediant (Fract a b) (Fract c d)
      using abcd ‹x = Fract m n› mediant-eq-Fract unimodular-imp-both-coprime
by fastforce
    show coprime a b coprime c d
      using ‹b * c − a * d = 1› unimodular-imp-both-coprime by blast+
  qed (use abcd in auto)
qed
```

Apostol's Theorem 5.5, second part: "Moreover, if $a / b < c / d$ are consecutive in any $F\ n$, then they satisfy the unimodular relation $bc - ad = 1$.

```
theorem consec-imp-unimodular:
  assumes sublist [Fract a b, Fract c d] (fareys (int n)) b>0 d>0 coprime a b
coprime c d
  shows  b*c − a*d = 1
  using assms
proof (induction n arbitrary: a b c d)
  case 0
  then show ?case
    by auto
next
  case (Suc n)
  show ?case
  proof (cases n=0)
    case True
    with Suc.prems have Fract a b = 0 Fract c d = 1
      by (auto simp add: sublist-Cons-right)
    with Suc.prems obtain a=0 b=1 c=1 d=1
      by (auto simp: Fract-of-int-quotient)
    then show ?thesis
      by auto
  next
    case False
    have Fract a b < Fract c d
      and ab: Fract a b ∈ set (fareys (1 + int n)) and cd: Fract c d ∈ set (fareys
(1 + int n))
      using strict-sorted-fareys [of Suc n] Suc.prems
      by (auto simp add: sublist-def sorted-wrt-append)
    have con: z ≤ Fract a b ∨ Fract c d ≤ z if z ∈ set (fareys (int n)) for z
    proof −
      have z ∈ set (fareys (1 + int n))
        using fareys-increasing-1 that by blast
      with Suc.prems strict-sorted-fareys [of 1 + int n] show ?thesis
        by (fastforce simp add: sublist-def sorted-wrt-append)
```

```
  qed
  show ?thesis
  proof (cases Fract a b ∈ set (fareys n) ∧ Fract c d ∈ set (fareys n))
    case True
    then have sublist [Fract a b, Fract c d] (fareys n)
      using con ‹Fract a b < Fract c d› sorted-two-sublist strict-sorted-fareys by
blast
    then show ?thesis
      by (simp add: Suc)
  next
    case False
   have notboth: False if §: Fract a b ∈ fareys-new (1+n) Fract c d ∈ fareys-new
(1+n)
    proof −
      obtain a' b' c' d' where eq':
        sublist [Fract a' b', Fract c' d'] (fareys n) Fract a b = mediant (Fract a'
b') (Fract c' d')
        by (smt (verit) ‹n≠0› § fareys-new-eq-mediant of-nat-Suc of-nat-le-0-iff
plus-1-eq-Suc)
      then have abcd': Fract a' b' ∈ set (fareys n) Fract c' d' ∈ set (fareys n)
        by (auto simp: sublist-def)
      have con': z ≤ Fract a' b' ∨ Fract c' d' ≤ z if z ∈ set (fareys n) for z
        by (metis eq'(1) sorted-two-sublist strict-sorted-fareys that)
      have Fract a' b' < Fract c' d'
        using eq'(1) sorted-two-sublist [OF strict-sorted-fareys] by blast
      then obtain A: Fract a' b' < Fract a b Fract a b < Fract c' d'
        using eq'(2) mediant-inbetween by presburger
        obtain a'' b'' c'' d'' where eq'':
        sublist [Fract a'' b'', Fract c'' d''] (fareys n) Fract c d = mediant (Fract
a'' b'') (Fract c'' d'')
        by (smt (verit) § ‹n≠0› fareys-new-eq-mediant of-nat-Suc of-nat-le-0-iff
plus-1-eq-Suc)
      then have abcd'': Fract a'' b'' ∈ set (fareys n) Fract c'' d'' ∈ set (fareys n)
        by (auto simp: sublist-def)
      then have Fract c'' d'' ∈ set (fareys (1 + int n))
        using fareys-increasing-1 by blast
      have con'': z ≤ Fract a'' b'' ∨ Fract c'' d'' ≤ z if z ∈ set (fareys n) for z
        using sorted-two-sublist [OF strict-sorted-fareys] eq''(1) that by blast
      then obtain Fract a'' b'' < Fract c'' d'' Fract a'' b'' < Fract c d Fract c d
< Fract c'' d''
      by (metis eq'' sorted-two-sublist [OF strict-sorted-fareys] mediant-inbetween)
      with A show False
        using con' con'' abcd' abcd'' con ‹Fract a b < Fract c d›
        by (metis eq'(2) eq''(2) dual-order.strict-trans1 not-less-iff-gr-or-eq)
    qed
    consider Fract a b ∈ fareys-new (1+n) | Fract c d ∈ fareys-new (1+n)
      using False set-fareys-plus1 [of n]
      by (metis (mono-tags, opaque-lifting) Suc.prems(1) Suc-eq-plus1-left UnE
list.set-intros(1) of-nat-Suc set-mono-sublist
```

```
        set-subset-Cons subset-iff)
    then show ?thesis
    proof cases
      case 1
      then obtain a' b' c' d' where eq:
        sublist [Fract a' b', Fract c' d'] (fareys n)
       Fract a b = mediant (Fract a' b') (Fract c' d') coprime a' b' coprime c' d'
b'>0 d'>0
          by (smt (verit) ‹n≠0› fareys-new-eq-mediant of-nat-Suc of-nat-le-0-iff
plus-1-eq-Suc)
      then have abcd': Fract a' b' ∈ set (fareys n) Fract c' d' ∈ set (fareys n)
        by (auto simp: sublist-def)
      have con': z ≤ Fract a' b' ∨ Fract c' d' ≤ z if z ∈ set (fareys n) for z
        using eq(1) sorted-two-sublist strict-sorted-fareys that by blast
      obtain Fract a' b' < Fract c' d' Fract a b < Fract c' d'
     using eq by (simp add: mediant-inbetween(2) sorted-two-sublist strict-sorted-fareys)
      then have Fract c d ≤ Fract c' d'
        using con abcd' linorder-not-less by blast
      moreover have Fract c' d' ≤ Fract c d
        if Fract c d ∈ set (fareys n)
        by (metis con' ‹Fract a b < Fract c d› ‹Fract a' b' < Fract c' d'› eq(2)
order.trans linorder-not-less mediant-inbetween(1)
            nless-le that)
      ultimately have Fract c' d' = Fract c d
        using notboth 1 cd set-fareys-plus1 by auto
      with Suc.prems obtain c' = c d' = d
        by (metis ‹0 < d'› ‹coprime c' d'› denom-farey-Fract num-farey-Fract)
      then have uni: b'*c − a'*d = 1
        using Suc eq by blast
      then obtain a = a' + c b = b' + d
        using eq Suc.prems apply (simp add: mediant-eq-Fract)
     by (metis ‹c' = c› ‹d' = d› denom-farey-Fract num-farey-Fract pos-add-strict
            unimodular-imp-coprime)
      with uni show ?thesis
        by (auto simp: algebra-simps)
    next
      case 2
      then obtain a' b' c' d' where eq:
        sublist [Fract a' b', Fract c' d'] (fareys n)
         Fract c d = mediant (Fract a' b') (Fract c' d') coprime a' b' coprime c'
d' b'>0 d'>0
          by (smt (verit) ‹n≠0› fareys-new-eq-mediant of-nat-Suc of-nat-le-0-iff
plus-1-eq-Suc)
      then have abcd': Fract a' b' ∈ set (fareys n) Fract c' d' ∈ set (fareys n)
        by (auto simp: sublist-def)
      have con': z ≤ Fract a' b' ∨ Fract c' d' ≤ z if z ∈ set (fareys n) for z
        using eq(1) sorted-two-sublist strict-sorted-fareys that by blast
      obtain Fract a' b' < Fract c' d' Fract a' b' < Fract c d
        using eq mediant-inbetween
```

by (*metis sorted-two-sublist strict-sorted-fareys*)
then have *Fract a′ b′* ≤ *Fract a b*
using *con abcd′ linorder-not-less* **by** *blast*
moreover have *Fract a b* ≤ *Fract a′ b′*
if *Fract a b* ∈ *set* (*fareys n*)
by (*metis* ‹*Fract a b* < *Fract c d*› ‹*Fract a′ b′* < *Fract c′ d′*› *con′ order.strict-trans2 eq*(*2*) *mediant-inbetween*(*2*) *not-less-iff-gr-or-eq that*)
ultimately have *Fract a′ b′* = *Fract a b*
using *notboth 2 ab set-fareys-plus1* **by** *auto*
with *Suc.prems* **obtain** *a′* = *a b′* = *b*
by (*metis* ‹*0* < *b′*› ‹*coprime a′ b′*› *denom-farey-Fract num-farey-Fract*)
then have *uni*: *b*∗*c′* − *a*∗*d′* = *1*
using *Suc.IH Suc.prems eq* **by** *blast*
then obtain *c* = *a* + *c′ d* = *b* + *d′*
using *eq Suc.prems* **apply** (*simp add*: *mediant-eq-Fract*)
by (*metis* ‹*a′* = *a*› ‹*b′* = *b*› *denom-farey-Fract num-farey-Fract pos-add-strict unimodular-imp-coprime*)
with *uni* **show** *?thesis*
by (*auto simp*: *algebra-simps*)
qed
qed
qed
qed

1.5 Ford circles

definition *Ford-center* :: *rat* ⇒ *complex* **where**
Ford-center r ≡ (λ(*h*,*k*). *Complex* (*h*/*k*) (*1*/(*2* ∗ *k*^*2*))) (*quotient-of r*)

definition *Ford-radius* :: *rat* ⇒ *real* **where**
Ford-radius r ≡ (λ(*h*,*k*). *1*/(*2* ∗ *k*^*2*)) (*quotient-of r*)

definition *Ford-tan* :: [*rat*,*rat*] ⇒ *bool* **where**
Ford-tan r s ≡ *dist* (*Ford-center r*) (*Ford-center s*) = *Ford-radius r* + *Ford-radius s*

definition *Ford-circle* :: *rat* ⇒ *complex set* **where**
Ford-circle r ≡ *sphere* (*Ford-center r*) (*Ford-radius r*)

lemma *Im-Ford-center* [*simp*]: *Im* (*Ford-center r*) = *Ford-radius r*
by (*auto simp*: *Ford-center-def Ford-radius-def split*: *prod.splits*)

lemma *Ford-radius-nonneg*: *Ford-radius r* ≥ *0*
by (*simp add*: *Ford-radius-def split*: *prod.splits*)

lemma *two-Ford-tangent*:
assumes *r*: (*a*,*b*) = *quotient-of r* **and** *s*: (*c*,*d*) = *quotient-of s*
shows (*dist* (*Ford-center r*) (*Ford-center s*))^*2* − (*Ford-radius r* + *Ford-radius*

```
s) ^2
      = ((a*d - b*c) ^2 - 1) / (b*d) ^2
proof -
  obtain 0: b > 0 d > 0
    by (metis assms quotient-of-denom-pos)
  have 1: dist (Ford-center r) (Ford-center s) ^ 2 = (a/b - c/d) ^2 + (1/(2*b^2)
- 1/(2*d^2)) ^ 2
    using assms by (force simp: Ford-center-def dist-norm complex-norm complex-diff
split: prod.splits)
  have 2: (Ford-radius r + Ford-radius s) ^ 2 = (1/(2*b^2) + 1/(2*d^2)) ^ 2
    using assms by (force simp: Ford-radius-def split: prod.splits)
  show ?thesis
    using 0 unfolding 1 2 by (simp add: field-simps eval-nat-numeral)
qed
```

Apostol's Theorem 5.6

```
lemma two-Ford-tangent-iff:
  assumes r: (a,b) = quotient-of r and s: (c,d) = quotient-of s
  shows Ford-tan r s ⟷ |b * c - a * d| = 1
proof -
  obtain 0: b > 0 d > 0
    by (metis assms quotient-of-denom-pos)
  have Ford-tan r s ⟷ dist (Ford-center r) (Ford-center s) ^ 2 = (Ford-radius
r + Ford-radius s) ^ 2
    using Ford-radius-nonneg by (simp add: Ford-tan-def)
  also have ... ⟷ ((a*d - b*c) ^2 - 1) / (b*d) ^2 = 0
    using two-Ford-tangent [OF assms] by (simp add: diff-eq-eq)
  also have ... ⟷ |b * c - a * d| = 1
     using 0 by (simp add: abs-square-eq-1 abs-minus-commute flip: of-int-mult
of-int-diff)
  finally show ?thesis .
qed
```

Also Apostol's Theorem 5.6: Distinct Ford circles do not overlap

```
lemma Ford-no-overlap:
  assumes r ≠ s
  shows dist (Ford-center r) (Ford-center s) ≥ Ford-radius r + Ford-radius s
proof -
  obtain a b c d where r: (a,b) = quotient-of r and s: (c,d) = quotient-of s
                  and b>0 d>0
    by (metis quotient-of-denom-pos surj-pair)
  moreover have a ≠ c ∨ b ≠ d
    using assms r s quotient-of-inject by force
  ultimately have a * d ≠ c * b
    by (metis Fract-of-int-quotient assms eq-rat(1) less-irrefl quotient-of-div)
  then have (a*d - b*c) ^2 ≥ (1::int)
    by (simp add: mult.commute int-one-le-iff-zero-less)
  then have ((a*d - b*c) ^2 - 1) / (b*d) ^2 ≥ (0::real)
    by (simp add: divide-simps mult-less-0-iff flip: of-int-mult of-int-power)
```

then show *?thesis*
using *two-Ford-tangent* [*OF r s*]
by (*metis* (*no-types*, *lifting*) *ge-iff-diff-ge-0 of-int-1 of-int-diff of-int-mult of-int-power power2-le-imp-le zero-le-dist*)
qed

lemma *Ford-aux1*:
assumes *a≠0*
shows *cmod* (*Complex* (*b* / (*a* * (*a*^{2} + *b*^{2}))) (*1* / (*2* * *a*^{2}) − *inverse* (*a*^{2} + *b*^{2}))) = *1* / (*2* * *a*^{2})
(**is** *cmod ?z* = *?r*)
proof −
have (*2* * *a*^{2}) * *cmod ?z* = *cmod* ((*2* * *a*^{2}) * *?z*)
by (*simp add*: *norm-mult power2-eq-square*)
also have ... = *cmod* (*Complex* (*2**a*b* / (*a*^{2} + *b*^{2})) (*1* − (*2* * *a*^{2}) / (*a*^{2} + *b*^{2})))
unfolding *complex-of-real-mult-Complex inverse-eq-divide*
using ‹*a≠0*› **by** (*simp add*: *power2-eq-square mult.assoc right-diff-distrib*)
also have ... = *cmod* (*Complex* (*2**a*b*) ((*a*^{2} + *b*^{2}) − (*2* * *a*^{2})) / (*a*^{2} + *b*^{2}))
unfolding *Complex-divide-complex-of-real diff-divide-distrib*
using *assms* **by** *force*
also have ... = *cmod* (*Complex* (*2**a*b*) ((*a*^{2} + *b*^{2}) − (*2* * *a*^{2}))) / (*a*^{2} + *b*^{2})
by (*smt* (*verit*) *norm-divide norm-of-real not-sum-power2-lt-zero*)
also have ... = *sqrt* ((*a*^{2} + *b*^{2}) ^ *2*) / (*a*^{2} + *b*^{2})
unfolding *power2-eq-square complex-norm*
by (*simp add*: *algebra-simps*)
also have ... = *1*
using *assms* **by** *auto*
finally show *?thesis*
by (*metis inverse-eq-divide inverse-unique*)
qed

lemma *Ford-aux2*:
assumes *a≠0*
shows *cmod* (*Complex* (*a* / (*b* * (*b*^{2} + *a*^{2})) − *1* / (*a* * *b*)) (*1* / (*2* * *a*^{2}) − *inverse* (*b*^{2} + *a*^{2}))) = *1* / (*2* * *a*^{2})
(**is** *cmod ?z* = *?r*)
proof −
have *a* / (*b* * (*b*^{2} + *a*^{2})) − *1* / (*a* * *b*) = −*b* / (*a* * (*b*^{2} + *a*^{2}))
by (*simp add*: *divide-simps power2-eq-square*)
then have *cmod ?z* = *cmod* (*Complex* (*b* / (*a* * (*a*^{2} + *b*^{2}))) (*1* / (*2* * *a*^{2}) − *inverse* (*a*^{2} + *b*^{2})))
by (*simp add*: *cmod-neg-real add.commute*)
also have ... = *1* / (*2* * *a*^{2})
using *Ford-aux1 assms* **by** *simp*
finally show *?thesis* **.**
qed

definition *Radem-trans* :: *rat* ⇒ *complex* ⇒ *complex* **where**

Radem-trans ≡ λ*r* τ. *let* (*h*,*k*) = *quotient-of r in* − i ∗ *of-int k* ^ *2* ∗ (τ − *of-rat r*)

Theorem 5.8 first part

lemma *Radem-trans-image*: *Radem-trans r* ‘ *Ford-circle r* = *sphere* (*of-rat* (*1*/*2*)) (*1*/*2*)
proof −
obtain *h k* **where** *r*: *quotient-of r* = (*h*,*k*) **and** *k*>*0* **and** *req*: *r* = *of-int h* / *of-int k*
using *quotient-of-denom-pos quotient-of-div* **by** *fastforce*
have *Radem-trans r* ‘ *Ford-circle r* = ((∗) (−i ∗ *of-int k* ^ *2*)) ‘ (λτ. τ − *of-rat r*) ‘ *Ford-circle r*
by (*simp add*: *Radem-trans-def r image-comp*)
also have ... = ((∗) (−i ∗ *of-int k* ^ *2*)) ‘ *sphere* (*Ford-center r* − *of-rat r*) (*Ford-radius r*)
by (*simp add*: *Ford-circle-def flip*: *sphere-translation-subtract*)
also have ... = *sphere* (− i ∗ (*of-int k*)^2 ∗ (*Ford-center r* − *r*)) (*cmod* (− i ∗ (*of-int k*)^2) ∗ *Ford-radius r*)
using ‹*k*>*0*› **by** (*intro sphere-cscale*) *auto*
also have ... = *sphere* (*of-rat* (*1*/*2*)) (*1*/*2*)
proof −
have (− i ∗ (*of-int k*)^2 ∗ (*Ford-center r* − *r*)) = *1*/*2*
using ‹*k*>*0*›
apply (*simp add*: *Ford-center-def r algebra-simps Complex-eq*)
by (*simp add*: *of-rat-divide req*)
moreover
have (*cmod* (− i ∗ (*of-int k*)^2) ∗ *Ford-radius r*) = *1*/*2*
using ‹*k*>*0*›
by (*simp add*: *norm-mult norm-power Ford-radius-def r*)
ultimately show *?thesis*
by (*metis of-rat-1 of-rat-divide of-rat-numeral-eq*)
qed
finally show *?thesis* .
qed

locale *three-Ford* =
fixes *N*::*nat*
fixes *h1 k1 h k h2 k2*::*int*
assumes *sub1*: *sublist* [*Fract h1 k1*, *Fract h k*] (*fareys* (*int N*))
assumes *sub2*: *sublist* [*Fract h k*, *Fract h2 k2*] (*fareys* (*int N*))
assumes *coprime*: *coprime h1 k1 coprime h k coprime h2 k2*
assumes *k-pos*: *k1* > *0 k* > *0 k2* > *0*

begin

definition *r1* ≡ *Fract h1 k1*
definition *r* ≡ *Fract h k*
definition *r2* ≡ *Fract h2 k2*

lemma *N-pos*: *N>0*
using *sub1 gr0I* **by** *force*

lemma *r-eq-quotient*:
(*h1*,*k1*) = *quotient-of r1* (*h*,*k*) = *quotient-of r* (*h2*,*k2*) = *quotient-of r2*
by (*simp-all add*: *coprime k-pos quotient-of-Fract r1-def r-def r2-def*)

lemma *r-eq-divide*:
r1 = *of-int h1* / *of-int k1 r* = *of-int h* / *of-int k r2* = *of-int h2*/ *of-int k2*
by (*simp-all add*: *Fract-of-int-quotient of-rat-divide r1-def r2-def r-def*)

lemma *collapse-r*:
real-of-int h1 / *of-int k1* = *of-rat r1*
real-of-int h / *of-int k* = *of-rat r real-of-int h2*/ *of-int k2* = *of-rat r2*
by (*simp-all add*: *of-rat-divide r-eq-divide*)

lemma *unimod1*: *k1∗h* − *h1∗k* = *1*
and *unimod2*: *k∗h2* − *h∗k2* = *1*
using *consec-imp-unimodular coprime k-pos sub1 sub2* **by** *blast+*

lemma *r-less*: *r1* < *r r* < *r2*
using *r1-def r-def r2-def sub1 sub2 sorted-two-sublist* [*OF strict-sorted-fareys*] **by** *auto*

lemma *r01*:
obtains *r1* ∈ {*0..1*} *r* ∈ {*0..1*} *r2* ∈ {*0..1*}
by (*metis denom-in-fareys-iff r1-def r2-def r-def sub1 sub2 sublist-fareysD*)

lemma *atMost-N*:
obtains *k1* ≤ *N k* ≤ *N k2* ≤ *N*
by (*metis denom-farey-def denom-in-fareys-iff prod.sel*(*2*) *r1-def r2-def r-def r-eq-quotient sub1 sub2*
sublist-fareysD)

lemma *greaterN1*: *k1* + *k* > *N*
using *sublist-fareys-add-denoms coprime k-pos sub1* **by** *blast*

lemma *greaterN2*: *k* + *k2* > *N*
using *sublist-fareys-add-denoms coprime k-pos sub2* **by** *blast*

definition *alpha1* ≡ *Complex* (*h*/*k* − *k1* / *of-int*(*k* ∗ (*k*^{2} + *k1*^{2}))) (*inverse* (*of-int* (*k*^{2} + *k1*^{2})))
definition *alpha2* ≡ *Complex* (*h*/*k* + *k2* / *of-int*(*k* ∗ (*k*^{2} + *k2*^{2}))) (*inverse* (*of-int* (*k*^{2} + *k2*^{2})))

definition *zed1* ≡ *Complex* (*k*^{2}) (*k∗k1*) / ((*k*^{2} + *k1*^{2}))
definition *zed2* ≡ *Complex* (*k*^{2}) (− *k∗k2*) / ((*k*^{2} + *k2*^{2}))

Apostol's Theorem 5.7

lemma *three-Ford-tangent*:
obtains *alpha1* ∈ *Ford-circle r alpha1* ∈ *Ford-circle r1*
alpha2 ∈ *Ford-circle r alpha2* ∈ *Ford-circle r2*
proof
show *alpha1* ∈ *Ford-circle r*
using *k-pos Ford-aux1 r-eq-quotient*
by (*force simp*: *alpha1-def Ford-circle-def Ford-center-def dist-norm complex-diff*

Ford-radius-def split: *prod.splits*)
have *1*: *real-of-int h1 / real-of-int k1 = real-of-int h / real-of-int k − 1 / (k1∗k)*
using *unimod1 k-pos*
by (*simp add*: *divide-simps*) (*simp add*: *algebra-simps flip*: *of-int-mult of-int-diff*)
show *alpha1* ∈ *Ford-circle r1*
using *k-pos Ford-aux2 r-eq-quotient*
by (*force simp*: *alpha1-def Ford-circle-def Ford-center-def dist-norm complex-diff*
1 Ford-radius-def split: *prod.splits*)
show *alpha2* ∈ *Ford-circle r*
using *k-pos Ford-aux1* [*of k k2*] *cmod-neg-real r-eq-quotient*
by (*force simp add*: *alpha2-def Ford-circle-def Ford-center-def dist-norm complex-diff*
Ford-radius-def split: *prod.splits*)
have *2*: *real-of-int h / real-of-int k = real-of-int h2 / real-of-int k2 − 1 / (k∗k2)*
using *unimod2 k-pos*
by (*simp add*: *divide-simps*) (*simp add*: *algebra-simps flip*: *of-int-mult of-int-diff*)
show *alpha2* ∈ *Ford-circle r2*
using *k-pos Ford-aux2* [*of k2 k*] *cmod-neg-real r-eq-quotient*
apply (*simp add*: *alpha2-def Ford-circle-def Ford-center-def dist-norm complex-diff*
2 Ford-radius-def split: *prod.splits*)
by (*smt* (*verit*) *mult.commute prod.sel*)
qed

Theorem 5.8 second part, for alpha1

lemma *Radem-trans-alpha1*: *Radem-trans r alpha1 = zed1*
proof −
have *Radem-trans r alpha1* = ((∗) (−i ∗ *of-int k* ^ *2*)) ((λτ. τ − *of-rat r*) *alpha1*)
by (*metis Radem-trans-def prod.simps*(*2*) *r-eq-quotient*(*2*))
also have ... = ((∗) (−i ∗ *of-int k* ^ *2*)) (*Complex* (− *k1* / *of-int*(*k* ∗ (k^2 + $k1^2$))) (*inverse* (*of-int* (k^2 + $k1^2$))))
using *k-pos* **by** (*simp add*: *alpha1-def r-def of-rat-rat Complex-eq*)
also have ... = *zed1*
unfolding *complex-eq-iff* **by** (*simp add*: *zed1-def inverse-eq-divide power2-eq-square*)
finally show *?thesis* **.**
qed

Theorem 5.8 second part, for alpha2

lemma *Radem-trans-alpha2*: *Radem-trans r alpha2 = zed2*
proof −
have *Radem-trans r alpha2* = ((∗) (−i ∗ *of-int k* ^ *2*)) ((λτ. τ − *of-rat r*) *alpha2*)
by (*metis Radem-trans-def prod.simps*(*2*) *r-eq-quotient*(*2*))
also have ... = ((∗) (−i ∗ *of-int k* ^ *2*)) (*Complex* (*k2* / *of-int*(*k* ∗ (k^2 + $k2^2$))) (*inverse* (*of-int* (k^2 + $k2^2$))))

using *k-pos* **by** (*simp add*: *alpha2-def r-def of-rat-rat Complex-eq*)
also have ... = *zed2*
unfolding *complex-eq-iff* **by** (*simp add*: *zed2-def inverse-eq-divide power2-eq-square*)
finally show *?thesis* **.**
qed

Theorem 5.9, for zed1

lemma *cmod-zed1*: *cmod zed1* = *k* / *sqrt* (k^2 + $k1^2$)
proof −
have *cmod zed1* ^ 2 = (*k*^4 + k^2 ∗ $k1^2$) / (k^2 + $k1^2$)^2
by (*simp add*: *zed1-def cmod-def divide-simps*)
also have ... = (*of-int k*) ^ 2 / (k^2 + $k1^2$)
by (*simp add*: *eval-nat-numeral divide-simps*) *argo*
finally have *cmod zed1* ^ 2 = (*of-int k*) ^ 2 / (k^2 + $k1^2$) **.**
with *k-pos real-sqrt-divide* **show** *?thesis*
unfolding *cmod-def* **by** *force*
qed

Theorem 5.9, for zed2

lemma *cmod-zed2*: *cmod zed2* = *k* / *sqrt* (k^2 + $k2^2$)
proof −
have *cmod zed2* ^ 2 = (*k*^4 + k^2 ∗ $k2^2$) / (k^2 + $k2^2$)^2
by (*simp add*: *zed2-def cmod-def divide-simps*)
also have ... = (*of-int k*) ^ 2 / (k^2 + $k2^2$)
by (*simp add*: *eval-nat-numeral divide-simps*) *argo*
finally have *cmod zed2* ^ 2 = (*of-int k*) ^ 2 / (k^2 + $k2^2$) **.**
with *k-pos real-sqrt-divide* **show** *?thesis*
unfolding *cmod-def* **by** *force*
qed

The last part of theorem 5.9

lemma *RMS-calc*:
assumes *k′* > 0 *k′* + *k* > *int N*
shows *k* / *sqrt* (k^2 + k'^2) < *sqrt* 2 ∗ *k*/*N*
proof −
have §: (*k* + *k′*)/2 ≤ *sqrt* ((k^2 + k'^2) / 2)
using *sum-squared-le-sum-of-squares-2* **by** *simp*
have *N* / *sqrt* 2 < (*N*+1) / *sqrt* 2
by (*simp add*: *divide-strict-right-mono*)
also have ... ≤ (*k* + *k′*) / *sqrt* 2
using *assms* **by** (*simp add*: *divide-simps*)
also have ... = (*k* + *k′*)/2 ∗ *sqrt* 2
by (*metis nonzero-divide-eq-eq real-div-sqrt times-divide-eq-right zero-le-numeral zero-neq-numeral*)
also have ... ≤ *sqrt* (k^2 + k'^2)
using § **by** (*simp add*: *le-divide-eq real-sqrt-divide*)
finally have *1*: *real N* / *sqrt* 2 < *sqrt* (*real-of-int* (k^2 + k'^2)) **.**
with *N-pos k-pos not-sum-power2-lt-zero* **show** *?thesis*
by (*force simp add*: *cmod-zed1 mult.commute divide-simps*)

qed

lemma *on-chord-bounded-cmod*:
assumes $z \in$ *closed-segment zed1 zed2*
shows *cmod z* $<$ *sqrt 2* $*$ k/N
proof $-$
have *cmod z* $\leq$ *max* (*cmod zed1*) (*cmod zed2*)
using *segment-furthest-le* [*OF assms*, *of 0*] **by** *auto*
moreover
obtain *cmod zed1* $<$ *sqrt 2* $*$ k/N *cmod zed2* $<$ *sqrt 2* $*$ k/N
using *RMS-calc cmod-zed1 cmod-zed2 greaterN1 greaterN2 k-pos* **by** *force*
ultimately show *?thesis*
using *assms cmod-zed1 cmod-zed2* **by** *linarith*
qed

end

end

Acknowledgements Manual Eberl set up the initial Farey development.

References

[1] T. M. Apostol. *Modular Functions and Dirichlet Series in Number Theory.* Springer, 1990.