Gauss Sums and the Pólya–Vinogradov Inequality

Rodrigo Raya and Manuel Eberl

October 13, 2025

Abstract

This article provides a full formalisation of Chapter 8 of Apostol's *Introduction to Analytic Number Theory* [1]. Subjects that are covered are:

- periodic arithmetic functions and their finite Fourier series
- (generalised) Ramanujan sums
- Gauss sums and separable characters
- induced moduli and primitive characters
- the Pólya–Vinogradov inequality

Contents

1 Auxiliary material

theory *Gauss-Sums-Auxiliary*
imports
  Dirichlet-L.Dirichlet-Characters
  Dirichlet-Series.Moebius-Mu
  Dirichlet-Series.More-Totient
begin

1.1 Various facts

lemma *sum-div-reduce*:
  fixes $d :: nat$ **and** $f :: nat \Rightarrow complex$
  assumes $d\ dvd\ k\ d > 0$
  shows $(\sum n \mid n \in \{1..k\} \wedge d\ dvd\ n.\ f\ n) = (\sum c \in \{1..k\ div\ d\}.\ f\ (c*d))$
   by (*rule sum.reindex-bij-witness*[*of* - $\lambda k.\ k * d\ \lambda k.\ k\ div\ d$])
    (*use assms* **in** ‹*fastforce simp*: *div-le-mono*›)+

lemma *prod-div-sub*:
  fixes $f :: nat \Rightarrow complex$
  assumes $finite\ A\ B \subseteq A\ \forall b \in B.\ f\ b \neq 0$
  shows $(\prod i \in A - B.\ f\ i) = ((\prod i \in A.\ f\ i)\ div\ (\prod i \in B.\ f\ i))$
  using *assms*
proof (*induction card B arbitrary*: *B*)
case *0*
  then show *?case*
    using *infinite-super* **by** *fastforce*
next
  case (*Suc n*)
  then show *?case*
  proof −
    obtain $B'\ x$ **where** *decomp*: $B = B' \cup \{x\} \wedge x \notin B'$
      using *card-eq-SucD*[*OF Suc*(*2*)[*symmetric*]] *insert-is-Un* **by** *auto*
    then have *B′card*: $card\ B' = n$ **using** *Suc*(*2*)
      using *Suc.prems*(*2*) *assms*(*1*) *finite-subset* **by** *fastforce*
    have $prod\ f\ (A - B) = prod\ f\ ((A-B') - \{x\})$
      by (*simp add*: *decomp*,*subst Diff-insert*,*simp*)
    also have $\ldots = (prod\ f\ (A-B'))\ div\ f\ x$
      using *prod-diff1*[*of* $A-B'\ f\ x$] *Suc decomp* **by** *auto*
    also have $\ldots = (prod\ f\ A\ div\ prod\ f\ B')\ div\ f\ x$
      using *Suc*(*1*)[*of B′*] *Suc*(*3*) *B′card decomp*
          Suc.prems(*2*) *Suc.prems*(*3*) **by** *force*
    also have $\ldots = prod\ f\ A\ div\ (prod\ f\ B' * f\ x)$ **by** *auto*
    also have $\ldots = prod\ f\ A\ div\ prod\ f\ B$
      using *decomp Suc.prems*(*2*) *assms*(*1*) *finite-subset* **by** *fastforce*
    finally show *?thesis* **by** *blast*
  qed
qed

lemma *linear-gcd*:

```
  fixes a b c d :: nat
  assumes a > 0 b > 0 c > 0 d > 0
  assumes coprime a c coprime b d
  shows gcd (a∗b) (c∗d) = (gcd a d) ∗ (gcd b c)
  using assms
proof −
  define q1 :: nat where q1 = a div gcd a d
  define q2 :: nat where q2 = c div gcd b c
  define q3 :: nat where q3 = b div gcd b c
  define q4 :: nat where q4 = d div gcd a d

  have coprime q1 q2 coprime q3 q4
    unfolding q1-def q2-def q3-def q4-def
  proof −
    have coprime (a div gcd a d) c
      using ‹coprime a c› coprime-mult-left-iff[of a div gcd a d gcd a d c]
            dvd-mult-div-cancel[OF gcd-dvd1, of a b] by simp
    then show coprime (a div gcd a d) (c div gcd b c)
      using coprime-mult-right-iff[of a div gcd a d gcd b c c div gcd b c]
          dvd-div-mult-self[OF gcd-dvd2[of b c]] by auto
    have coprime (b div gcd b c) d
      using ‹coprime b d› coprime-mult-left-iff[of b div gcd b c gcd b c d]
            dvd-mult-div-cancel[OF gcd-dvd1, of a b] by simp
    then show coprime (b div gcd b c) (d div gcd a d)
      using coprime-mult-right-iff[of b div gcd b c gcd a d d div gcd a d]
          dvd-div-mult-self[OF gcd-dvd2[of b c]] by auto
  qed
  moreover have coprime q1 q4 coprime q3 q2
    unfolding q1-def q2-def q3-def q4-def
    using assms div-gcd-coprime by blast+
  ultimately have 1: coprime (q1∗q3) (q2∗q4)
    by simp
  have gcd (a∗b) (c∗d) = (gcd a d) ∗ (gcd b c) ∗ gcd (q1∗q3) (q2∗q4)
    unfolding q1-def q2-def q3-def q4-def
    by (subst gcd-mult-distrib-nat[of gcd a d ∗ gcd b c],
       simp add: field-simps,
       simp add: mult.left-commute semiring-normalization-rules(18))
  from this 1 show gcd (a∗b) (c∗d) = (gcd a d) ∗ (gcd b c) by auto
qed

lemma reindex-product-bij:
  fixes a b m k :: nat
  defines S ≡ {(d1,d2). d1 dvd gcd a m ∧ d2 dvd gcd k b}
  defines T ≡ {d. d dvd (gcd a m) ∗ (gcd k b)}
  defines f ≡ (λ(d1,d2). d1 ∗ d2)
  assumes coprime a k
  shows bij-betw f S T
  unfolding bij-betw-def
proof
```

```
  show inj: inj-on f S
    unfolding f-def
  proof -
    {fix d1 d2 d1' d2'
    assume (d1,d2) ∈ S (d1',d2') ∈ S
    then have dvd: d1 dvd gcd a m d2 dvd gcd k b
              d1' dvd gcd a m d2' dvd gcd k b
      unfolding S-def by simp+
    assume f (d1,d2) = f (d1',d2')
    then have eq: d1 * d2 = d1' * d2'
      unfolding f-def by simp
    from eq dvd have eq1: d1 = d1'
      by (simp,meson assms coprime-crossproduct-nat coprime-divisors)
    from eq dvd have eq2: d2 = d2'
      using assms(4) eq1 by auto
    from eq1 eq2 have d1 = d1' ∧ d2 = d2' by simp}
   then show inj-on (λ(d1, d2). d1 * d2) S
    using S-def f-def by (intro inj-onI,blast)
  qed
  show surj: f ` S = T
  proof -
    {fix d
      have d dvd (gcd a m) * (gcd k b)
       ⟷ (∃ d1 d2. d = d1*d2 ∧ d1 dvd gcd a m ∧ d2 dvd gcd k b)
        using division-decomp mult-dvd-mono by blast}
      then show ?thesis
        unfolding f-def S-def T-def image-def
        by auto
  qed
qed

lemma p-div-set:
  shows {p. p ∈prime-factors a ∧ ¬ p dvd N} =
        ({p. p ∈prime-factors (a*N)} - {p. p ∈prime-factors N})
  (is ?A = ?B)
proof
  show ?A ⊆ ?B
  proof (simp)
    { fix p
      assume as: p ∈# prime-factorization a ¬ p dvd N
      then have 1: p ∈ prime-factors (a * N)
      proof -
        from in-prime-factors-iff[of p a] as
        have a ≠ 0 p dvd a prime p by simp+
        have N ≠ 0 using ‹¬ p dvd N› by blast
        have a * N ≠ 0 using ‹a ≠ 0› ‹N ≠ 0› by auto
        have p dvd a*N using ‹p dvd a› by simp
        show ?thesis
          using ‹a*N ≠ 0› ‹p dvd a*N› ‹prime p› in-prime-factors-iff by blast
```

```
      qed
      from as have 2: p ∉ prime-factors N by blast
      from 1 2 have p ∈ prime-factors (a ∗ N) − prime-factors N
       by blast
    }
    then show {p. p ∈# prime-factorization a ∧ ¬ p dvd N}
              ⊆ prime-factors (a ∗ N) − prime-factors N by blast
  qed

  show ?B ⊆ ?A
  proof (simp)
    { fix p
      assume as: p ∈ prime-factors (a ∗ N) − prime-factors N
      then have 1: ¬ p dvd N
      proof −
        from as have p ∈ prime-factors (a ∗ N) p ∉ prime-factors N
          using DiffD1 DiffD2 by blast+
        then show ?thesis by (simp add: in-prime-factors-iff)
      qed
      have 2: p ∈# prime-factorization a
      proof −
       have p dvd (a∗N) prime p a∗N ≠ 0 using in-prime-factors-iff as by blast+
         have p dvd a using ‹¬ p dvd N› prime-dvd-multD[OF ‹prime p› ‹p dvd
(a∗N)›] by blast
        have a ≠ 0 using ‹a∗N ≠ 0› by simp
         show ?thesis using in-prime-factors-iff ‹a ≠ 0› ‹p dvd a› ‹prime p› by
blast
      qed
      from 1 2 have p ∈ {p. p ∈# prime-factorization a ∧ ¬ p dvd N} by blast
    }
    then show prime-factors (a ∗ N) − prime-factors N
              ⊆ {p. p ∈# prime-factorization a ∧ ¬ p dvd N} by blast
  qed
qed

lemma coprime-iff-prime-factors-disjoint:
  fixes x y :: 'a :: factorial-semiring
  assumes x ≠ 0 y ≠ 0
  shows coprime x y ⟷ prime-factors x ∩ prime-factors y = {}
proof
  assume coprime x y
  have False if p ∈ prime-factors x p ∈ prime-factors y for p
  proof −
    from that assms have p dvd x p dvd y
      by (auto simp: prime-factors-dvd)
    with ‹coprime x y› have p dvd 1
      using coprime-common-divisor by auto
    with that assms show False by (auto simp: prime-factors-dvd)
  qed
```

thus *prime-factors x* ∩ *prime-factors y* = {} **by** *auto*
next
assume *disjoint*: *prime-factors x* ∩ *prime-factors y* = {}
show *coprime x y*
proof (*rule coprimeI*)
fix *d* **assume** *d*: *d dvd x d dvd y*
show *is-unit d*
proof (*rule ccontr*)
assume ¬*is-unit d*
moreover from *this* **and** *d assms* **have** *d* ≠ *0* **by** *auto*
ultimately obtain *p* **where** *p*: *prime p p dvd d*
using *prime-divisor-exists* **by** *auto*
with *d* **and** *assms* **have** *p* ∈ *prime-factors x* ∩ *prime-factors y*
by (*auto simp*: *prime-factors-dvd*)
with *disjoint* **show** *False* **by** *auto*
qed
qed
qed

lemma *coprime-cong-prime-factors*:
fixes *x y* :: *'a* :: *factorial-semiring-gcd*
assumes *x* ≠ *0* *y* ≠ *0* *x'* ≠ *0* *y'* ≠ *0*
assumes *prime-factors x* = *prime-factors x'*
assumes *prime-factors y* = *prime-factors y'*
shows *coprime x y* ⟷ *coprime x' y'*
using *assms* **by** (*simp add*: *coprime-iff-prime-factors-disjoint*)

lemma *moebius-prod-not-coprime*:
assumes ¬ *coprime N d*
shows *moebius-mu* (*N*∗*d*) = *0*
proof −
from *assms* **obtain** *l* **where** *l-form*: *l dvd N* ∧ *l dvd d* ∧ ¬ *is-unit l*
unfolding *coprime-def* **by** *blast*
then have *l* ∗ *l dvd N* ∗ *d* **using** *mult-dvd-mono* **by** *auto*
then have l^2 *dvd N*∗*d* **by** (*subst power2-eq-square*,*blast*)
then have ¬ *squarefree* (*N*∗*d*)
unfolding *squarefree-def coprime-def* **using** *l-form* **by** *blast*
then show *moebius-mu* (*N*∗*d*) = *0*
using *moebius-mu-def* **by** *auto*
qed

Theorem 2.18

lemma *sum-divisors-moebius-mu-times-multiplicative*:
fixes *f* :: *nat* ⇒ *'a* :: {*comm-ring-1*}
assumes *multiplicative-function f* **and** *n* > *0*
shows (∑ *d* | *d dvd n*. *moebius-mu d* ∗ *f d*) = (∏ *p*∈*prime-factors n*. *1* − *f p*)
proof −
define *g* **where** *g* = (λ*n*. ∑ *d* | *d dvd n*. *moebius-mu d* ∗ *f d*)
define *g'* **where** *g'* = *dirichlet-prod* (λ*n*. *moebius-mu n* ∗ *f n*) (λ*n*. *if n* = *0 then*

```
0 else 1)
  interpret f: multiplicative-function f by fact
  have multiplicative-function (λn. if n = 0 then 0 else 1 :: 'a)
    by standard auto
  interpret multiplicative-function g' unfolding g'-def
    by (intro multiplicative-dirichlet-prod multiplicative-function-mult
          moebius-mu.multiplicative-function-axioms assms) fact+

  have g'-primepow: g' (p ^ k) = 1 − f p if prime p k > 0 for p k
  proof −
    have g' (p ^ k) = (∑i≤k. moebius-mu (p ^ i) * f (p ^ i))
      using that by (simp add: g'-def dirichlet-prod-prime-power)
    also have ... = (∑i∈{0, 1}. moebius-mu (p ^ i) * f (p ^ i))
     using that by (intro sum.mono-neutral-right) (auto simp: moebius-mu-power')
    also have ... = 1 − f p
      using that by (simp add: moebius-mu.prime)
    finally show ?thesis .
  qed

  have g' n = g n
    by (simp add: g-def g'-def dirichlet-prod-def)
  also from assms have g' n = (∏p∈prime-factors n. g' (p ^ multiplicity p n))
   by (intro prod-prime-factors) auto
  also have ... = (∏p∈prime-factors n. 1 − f p)
    by (intro prod.cong) (auto simp: g'-primepow prime-factors-multiplicity)
  finally show ?thesis by (simp add: g-def)
qed

lemma multiplicative-ind-coprime [intro]: multiplicative-function (ind (coprime N))
  by (intro multiplicative-function-ind) auto

lemma sum-divisors-moebius-mu-times-multiplicative-revisited:
  fixes f :: nat ⇒ 'a :: {comm-ring-1}
  assumes multiplicative-function f n > 0 N > 0
  shows   (∑d | d dvd n ∧ coprime N d. moebius-mu d * f d) =
           (∏p∈{p. p ∈ prime-factors n ∧ ¬ (p dvd N)}. 1 − f p)
proof −
  have (∑d | d dvd n ∧ coprime N d. moebius-mu d * f d) =
         (∑d | d dvd n. moebius-mu d * (ind (coprime N) d * f d))
    using assms by (intro sum.mono-neutral-cong-left) (auto simp: ind-def)
  also have ... = (∏p∈prime-factors n. 1 − ind (coprime N) p * f p)
    using assms by (intro sum-divisors-moebius-mu-times-multiplicative)
                  (auto intro: multiplicative-function-mult)
  also from assms have ... = (∏p | p ∈ prime-factors n ∧ ¬(p dvd N). 1 − f p)
    by (intro prod.mono-neutral-cong-right)
     (auto simp: ind-def prime-factors-dvd coprime-commute dest: prime-imp-coprime)
  finally show ?thesis .
qed
```

1.2 Neutral element of the Dirichlet product

definition *dirichlet-prod-neutral* $n = (\textit{if } n = 1 \textit{ then } 1 \textit{ else } 0)$ **for** $n :: nat$

lemma *dirichlet-prod-neutral-intro*:
  fixes $S :: nat \Rightarrow complex$ **and** $f :: nat \Rightarrow nat \Rightarrow complex$
  defines $S \equiv (\lambda(n{::}nat).\ (\sum k \mid k \in \{1..n\} \wedge coprime\ k\ n.\ (f\ k\ n)))$
  shows $S(n) = (\sum k \in \{1..n\}.\ f\ k\ n * dirichlet\text{-}prod\text{-}neutral\ (gcd\ k\ n))$
proof −
  let $?g = \lambda k.\ (f\ k\ n) * (dirichlet\text{-}prod\text{-}neutral\ (gcd\ k\ n))$
  have *zeros*: $\forall k \in \{1..n\} - \{k.\ k \in \{1..n\} \wedge coprime\ k\ n\}.\ ?g\ k = 0$
  proof
    fix k
    assume $k \in \{1..n\} - \{k \in \{1..n\}.\ coprime\ k\ n\}$
    then show $(f\ k\ n) * dirichlet\text{-}prod\text{-}neutral\ (gcd\ k\ n) = 0$
      by (*simp add*: *dirichlet-prod-neutral-def*[*of gcd k n*] *split*: *if-splits*,*presburger*)
  qed

  have $S\ n = (\sum k \mid k \in \{1..n\} \wedge coprime\ k\ n.\ (f\ k\ n))$
    by (*simp add*: *S-def*)
  also have $\ldots = sum\ ?g\ \{k.\ k \in \{1..n\} \wedge coprime\ k\ n\}$
    by (*simp add*: *dirichlet-prod-neutral-def split*: *if-splits*)
  also have $\ldots = sum\ ?g\ \{1..n\}$
    by (*intro sum.mono-neutral-left*, *auto simp add*: *zeros*)
  finally show *?thesis* **by** *blast*
qed

lemma *dirichlet-prod-neutral-right-neutral*:
 $dirichlet\text{-}prod\ f\ dirichlet\text{-}prod\text{-}neutral\ n = f\ n$ **if** $n > 0$ **for** $f :: nat \Rightarrow complex$
and n
proof −
  {fix $d :: nat$
    assume $d\ dvd\ n$
    then have *eq*: $n = d \longleftrightarrow n\ div\ d = 1$
      using *div-self that dvd-mult-div-cancel* **by** *force*
    have $f(d) * dirichlet\text{-}prod\text{-}neutral(n\ div\ d) = (\textit{if } n = d \textit{ then } f(d) \textit{ else } 0)$
      by (*simp add*: *dirichlet-prod-neutral-def eq*)**}**
  note $summand = this$

  have $dirichlet\text{-}prod\ f\ dirichlet\text{-}prod\text{-}neutral\ n =$
        $(\sum d \mid d\ dvd\ n.\ f(d) * dirichlet\text{-}prod\text{-}neutral(n\ div\ d))$
    unfolding *dirichlet-prod-def* **by** *blast*
  also have $\ldots = (\sum d \mid d\ dvd\ n.\ (\textit{if } n = d \textit{ then } f(d) \textit{ else } 0))$
    using *summand* **by** *simp*
  also have $\ldots = (\sum d \mid d = n.\ (\textit{if } n = d \textit{ then } f(d) \textit{ else } 0))$
    using *that* **by** (*intro sum.mono-neutral-right*, *auto*)
  also have $\ldots = f(n)$ **by** *simp*
  finally show *?thesis* **by** *simp*
qed

lemma *dirichlet-prod-neutral-left-neutral*:
dirichlet-prod dirichlet-prod-neutral f n = *f n*
if $n > 0$ **for** *f* :: *nat* ⇒ *complex* **and** *n*
using *dirichlet-prod-neutral-right-neutral*[*OF that*, *of f*]
dirichlet-prod-commutes[*of f dirichlet-prod-neutral*]
by *argo*

corollary *I-right-neutral-0*:
fixes *f* :: *nat* ⇒ *complex*
assumes *f 0* = *0*
shows *dirichlet-prod f dirichlet-prod-neutral n* = *f n*
using *assms dirichlet-prod-neutral-right-neutral* **by** (*cases n*, *simp*, *blast*)

1.3 Multiplicative functions

lemma *mult-id*: *multiplicative-function id*
by (*simp add*: *multiplicative-function-def*)

lemma *mult-moebius*: *multiplicative-function moebius-mu*
using *Moebius-Mu.moebius-mu.multiplicative-function-axioms*
by *simp*

lemma *mult-of-nat*: *multiplicative-function of-nat*
using *multiplicative-function-def of-nat-0 of-nat-1 of-nat-mult* **by** *blast*

lemma *mult-of-nat-c*: *completely-multiplicative-function of-nat*
by (*simp add*: *completely-multiplicative-function-def*)

lemma *completely-multiplicative-nonzero*:
fixes *f* :: *nat* ⇒ *complex*
assumes *completely-multiplicative-function f*
d ≠ *0*
⋀*p*. *prime p* ⟹ *f*(*p*) ≠ *0*
shows *f*(*d*) ≠ *0*
using *assms*(*2*)
proof (*induction d rule*: *nat-less-induct*)
case (*1 n*)
then show *?case*
proof (*cases n* = *1*)
case *True*
then show *?thesis*
using *assms*(*1*)
unfolding *completely-multiplicative-function-def* **by** *simp*
next
case *False*
then obtain *p* **where** *2*:*prime p* ∧ *p dvd n*
using *prime-factor-nat* **by** *blast*
then obtain *a* **where** *3*: *n* = *p* ∗ *a* *a* ≠ *0*
using *1* **by** *auto*

```
    then have 4: f(a) ≠ 0 using 1
      using 2 prime-nat-iff by fastforce
    have 5: f(p) ≠ 0 using assms(3) 2 by simp
    from 3 4 5 show ?thesis
      by (simp add: assms(1) completely-multiplicative-function.mult)
  qed
qed

lemma multipl-div:
  fixes m k d1 d2 :: nat and f :: nat ⇒ complex
  assumes multiplicative-function f d1 dvd m d2 dvd k coprime m k
  shows f ((m*k) div (d1*d2)) = f(m div d1) * f(k div d2)
  using assms
  unfolding multiplicative-function-def
  using assms(1) multiplicative-function.mult-coprime by fastforce

lemma multipl-div-mono:
  fixes m k d :: nat and f :: nat ⇒ complex
  assumes completely-multiplicative-function f
          d dvd k d > 0
          ⋀p. prime p ⟹ f(p) ≠ 0
  shows f (k div d) = f(k) div f(d)
proof −
  have d ≠ 0 using assms(2,3) by auto
  then have nz: f(d) ≠ 0 using assms(1,4) completely-multiplicative-nonzero by
simp

  from assms(2,3) obtain a where div: k = a * d  by fastforce
  have f (k div d) = f ((a*d) div d) using div by simp
  also have ... = f(a) using assms(3) div by simp
  also have ... = f(a)*f(d) div f(d) using nz by auto
  also have ... = f(a*d) div f(d)
    by (simp add: div assms(1) completely-multiplicative-function.mult)
  also have ... = f (k) div f(d) using div by simp
  finally show ?thesis by simp
qed

lemma comp-to-mult: completely-multiplicative-function f ⟹
       multiplicative-function f
  unfolding completely-multiplicative-function-def
            multiplicative-function-def by auto

end
```

2 Periodic arithmetic functions

```
theory Periodic-Arithmetic
imports
  Complex-Main
```

```
  HOL−Number-Theory.Cong
begin

definition
  periodic-arithmetic f k = (∀ n. f (n+k) = f n)
  for n :: int and k :: nat and f :: nat ⇒ complex

lemma const-periodic-arithmetic: periodic-arithmetic (λx. y) k
  unfolding periodic-arithmetic-def by blast

lemma add-periodic-arithmetic:
  fixes f g :: nat ⇒ complex
  assumes periodic-arithmetic f k
  assumes periodic-arithmetic g k
  shows periodic-arithmetic (λn. f n + g n) k
  using assms unfolding periodic-arithmetic-def by simp

lemma mult-periodic-arithmetic:
  fixes f g :: nat ⇒ complex
  assumes periodic-arithmetic f k
  assumes periodic-arithmetic g k
  shows periodic-arithmetic (λn. f n ∗ g n) k
  using assms unfolding periodic-arithmetic-def  by simp

lemma scalar-mult-periodic-arithmetic:
  fixes f :: nat ⇒ complex and a :: complex
  assumes periodic-arithmetic f k
  shows periodic-arithmetic (λn. a ∗ f n) k
 using mult-periodic-arithmetic[OF const-periodic-arithmetic[of a k] assms(1)] by
simp

lemma fin-sum-periodic-arithmetic-set:
  fixes f g :: nat ⇒ complex
  assumes ∀ i∈A. periodic-arithmetic (h i) k
  shows periodic-arithmetic (λn. ∑ i ∈ A. h i n) k
  using assms by (simp add: periodic-arithmetic-def)

lemma mult-period:
  assumes periodic-arithmetic g k
  shows periodic-arithmetic g (k∗q)
  using assms
proof (induction q)
  case 0 then show ?case unfolding periodic-arithmetic-def by simp
next
  case (Suc m)
  then show ?case
    unfolding periodic-arithmetic-def
  proof −
   { fix n
```

```
      have g (n + k * Suc m) = g (n + k +  k * m)
        by (simp add: algebra-simps)
      also have ... = g(n)
        using Suc.IH[OF Suc.prems] assms
        unfolding periodic-arithmetic-def by simp
      finally have g (n + k * Suc m) = g(n) by blast
    }
    then show ∀ n. g (n + k * Suc m) = g n by auto
  qed
qed

lemma unique-periodic-arithmetic-extension:
  assumes k > 0
  assumes ∀ j<k. g j = h j
  assumes periodic-arithmetic g k and periodic-arithmetic h k
  shows g i = h i
proof (cases i < k)
  case True then show ?thesis using assms by simp
next
  case False then show ?thesis
  proof −
    have k * (i div k) + (i mod k) = i ∧ (i mod k) < k
      by (simp add: assms(1) algebra-simps)
    then obtain q r where euclid-div: k*q + r = i ∧ r < k
      using mult.commute by blast
    from assms(3) assms(4)
    have  periodic-arithmetic g (k*q) periodic-arithmetic h (k*q)
      using mult-period by simp+
    have g(k*q+r) = g(r)
      using ‹periodic-arithmetic g (k*q)› unfolding periodic-arithmetic-def
      using add.commute[of k*q r] by presburger
    also have ... = h(r)
      using euclid-div assms(2) by simp
    also have ... = h(k*q+r)
      using ‹periodic-arithmetic h (k*q)› add.commute[of k*q r]
      unfolding periodic-arithmetic-def by presburger
    also have ... = h(i) using euclid-div by simp
    finally show g(i) = h(i) using euclid-div by simp
  qed
qed

lemma periodic-arithmetic-sum-periodic-arithmetic:
  assumes periodic-arithmetic f k
  shows (∑ l ∈ {m..n}. f l) = (∑ l ∈ {m+k..n+k}. f l)
  using periodic-arithmetic-def assms
  by (intro sum.reindex-bij-witness
        [of {m..n} λl. l−k λl. l+k {m+k..n+k} f f])
     auto
```

lemma *mod-periodic-arithmetic*:
  fixes *n m* :: *nat*
  assumes *periodic-arithmetic f k*
  assumes *n mod k* = *m mod k*
  shows *f n* = *f m*
proof −
  obtain *q* **where** *1*: *n* = *q*∗*k*+(*n mod k*)
    using *div-mult-mod-eq*[*of n k*,*symmetric*] **by** *blast*
  obtain *q′* **where** *2*: *m* = *q′*∗*k*+(*m mod k*)
    using *div-mult-mod-eq*[*of m k*,*symmetric*] **by** *blast*
  from *1* **have** *f n* = *f* (*q*∗*k*+(*n mod k*)) **by** *auto*
  also have . . . = *f* (*n mod k*)
   using *mult-period*[*of f k q*] *assms*(*1*) *periodic-arithmetic-def*[*of f k*∗*q*]
   by (*simp add*: *algebra-simps*,*subst add.commute*,*blast*)
  also have . . . = *f* (*m mod k*) **using** *assms*(*2*) **by** *auto*
  also have . . . = *f* (*q′*∗*k*+(*m mod k*))
   using *mult-period*[*of f k q′*] *assms*(*1*) *periodic-arithmetic-def*[*of f k*∗*q′*]
   by (*simp add*: *algebra-simps*,*subst add.commute*,*presburger*)
  also have . . . = *f m* **using** *2* **by** *auto*
  finally show *f n* = *f m* **by** *simp*
qed

lemma *cong-periodic-arithmetic*:
  assumes *periodic-arithmetic f k* [*a* = *b*] (*mod k*)
  shows *f a* = *f b*
  using *assms mod-periodic-arithmetic*[*of f k a b*] **by** (*auto simp*: *cong-def*)

lemma *cong-nat-imp-eq*:
  fixes *m* :: *nat*
  assumes *m* > *0* *x* ∈ {*a*..<*a*+*m*} *y* ∈ {*a*..<*a*+*m*} [*x* = *y*] (*mod m*)
  shows *x* = *y*
  using *assms*
proof (*induction x y rule*: *linorder-wlog*)
  case (*le x y*)
  have [*y* − *x* = *0*] (*mod m*)
   using *cong-diff-iff-cong-0-nat cong-sym le* **by** *blast*
  thus *x* = *y*
   using *le* **by** (*auto simp*: *cong-def*)
qed (*auto simp*: *cong-sym*)

lemma *inj-on-mod-nat*:
  fixes *m* :: *nat*
  assumes *m* > *0*
  shows *inj-on* (λ*x*. *x mod m*) {*a*..<*a*+*m*}
proof
  fix *x y* **assume** *xy*: *x* ∈ {*a*..<*a*+*m*} *y* ∈ {*a*..<*a*+*m*} **and** *eq*: *x mod m* = *y mod*
m
  from ‹*m* > *0*› **and** *xy* **show** *x* = *y*
   by (*rule cong-nat-imp-eq*) (*use eq* **in** ‹*simp-all add*: *cong-def*›)

```
qed

lemma bij-betw-mod-nat-atLeastLessThan:
  fixes k d :: nat
  assumes k > 0
  defines g ≡ (λi. nat ((int i − int d) mod int k) + d)
  shows   bij-betw (λi. i mod k) {d..<d+k} {..<k}
  unfolding bij-betw-def
proof
  show inj: inj-on (λi. i mod k) {d..<d + k}
    by (rule inj-on-mod-nat) fact+
  have (λi. i mod k) ‘ {d..<d + k} ⊆ {..<k}
    by auto
  moreover have card ((λi. i mod k) ‘ {d..<d + k}) = card {..<k}
    using inj by (subst card-image) auto
  ultimately show (λi. i mod k) ‘ {d..<d + k} = {..<k}
    by (intro card-subset-eq) auto
qed

lemma periodic-arithmetic-sum-periodic-arithmetic-shift:
  fixes k d :: nat
  assumes periodic-arithmetic f k k > 0 d > 0
  shows (∑l ∈ {0..k−1}. f l) = (∑l ∈ {d..d+k−1}. f l)
proof −
  have (∑l ∈ {0..k−1}. f l) = (∑l ∈ {0..<k}. f l)
    using assms(2) by (intro sum.cong) auto
  also have ... = (∑l ∈ {d..<d+k}. f (l mod k))
    using assms(2)
    by (simp add: sum.reindex-bij-betw[OF bij-betw-mod-nat-atLeastLessThan[of k
d]]
                  lessThan-atLeast0)
  also have ... = (∑l ∈ {d..<d+k}. f l)
    using mod-periodic-arithmetic[of f k] assms(1) sum.cong
    by (meson mod-mod-trivial)
  also have ... = (∑l ∈ {d..d+k−1}. f l)
    using assms(2,3) by (intro sum.cong) auto
  finally show ?thesis by auto
qed

lemma self-bij-0-k:
  fixes a k :: nat
  assumes coprime a k [a∗i = 1] (mod k) k > 0
  shows bij-betw (λr. r∗a mod k) {0..k−1} {0..k−1}
  unfolding bij-betw-def
proof
  show inj-on (λr. r∗a mod k) {0..k−1}
  proof −
    {fix r1 r2
    assume in-k: r1 ∈ {0..k−1} r2 ∈ {0..k−1}
```

```
    assume as: [r1*a = r2*a] (mod k)
    then have [r1*a*i = r2*a*i] (mod k)
      using cong-scalar-right by blast
    then have [r1 = r2] (mod k)
      using cong-mult-rcancel-nat as assms(1) by simp
    then have r1 = r2 using in-k
      using assms(3) cong-less-modulus-unique-nat by auto}
    note eq = this
    show ?thesis unfolding inj-on-def
      by (safe, simp add: eq cong-def)
  qed
  define f where f = (λr. r * a mod k)
  show f ` {0..k - 1} = {0..k - 1}
    unfolding image-def
  proof (standard)
    show {y. ∃x∈{0..k - 1}. y = f x} ⊆ {0..k - 1}
    proof -
      {fix y
      assume y ∈ {y. ∃x∈{0..k - 1}. y = f x}
      then obtain x where y = f x by blast
      then have y ∈ {0..k-1}
        unfolding f-def
        using Suc-pred assms(3) lessThan-Suc-atMost by fastforce}
      then show ?thesis by blast
    qed
    show {0..k - 1} ⊆ {y. ∃x∈{0..k - 1}. y = f x}
    proof -
      { fix x
        assume ass: x ∈ {0..k-1}
        then have x * i mod k ∈ {0..k-1}
        proof -
          have x * i mod k ∈ {0..<k} by (simp add: assms(3))
          have {0..<k} = {0..k-1} using Suc-diff-1 assms(3) by auto
           show ?thesis using ‹x * i mod k ∈ {0..<k}› ‹{0..<k} = {0..k-1}› by
blast
        qed
        then have f (x * i mod k) = x
        proof -
          have f (x * i mod k) = (x * i mod k) * a mod k
            unfolding f-def by blast
          also have ... = (x*i*a) mod k
            by (simp add: mod-mult-left-eq)
          also have ... = (x*1) mod k
            using assms(2)
            unfolding cong-def
            by (subst mult.assoc, subst (2) mult.commute,
               subst mod-mult-right-eq[symmetric],simp)
          also have ... = x using ass assms(3) by auto
          finally show ?thesis .
```

```
        qed
        then have x ∈ {y. ∃x∈{0..k − 1}. y = f x}
          using ‹x ∗ i mod k ∈ {0..k−1}› by force
      }
      then show ?thesis by blast
    qed
  qed
qed

lemma periodic-arithmetic-homothecy:
  assumes periodic-arithmetic f k
  shows   periodic-arithmetic (λl. f (l∗a)) k
  unfolding periodic-arithmetic-def
proof
  fix n
  have f ((n + k) ∗ a) = f(n∗a+k∗a) by (simp add: algebra-simps)
  also have ... = f(n∗a)
    using mult-period[OF assms] unfolding periodic-arithmetic-def by simp
  finally show f ((n + k) ∗ a) = f (n ∗ a) by simp
qed

theorem periodic-arithmetic-remove-homothecy:
  assumes coprime a k periodic-arithmetic f k k > 0
  shows (∑ l=1..k. f l) = (∑ l=1..k. f (l∗a))
proof −
  obtain i where inv: [a∗i = 1] (mod k)
    using assms(1) coprime-iff-invertible-nat[of a k] by auto
  from this self-bij-0-k assms
  have bij: bij-betw (λr. r ∗ a mod k) {0..k − 1} {0..k − 1} by blast

  have (∑ l = 1..k. f(l)) = (∑ l = 0..k−1. f(l))
   using periodic-arithmetic-sum-periodic-arithmetic-shift[of f k 1] assms by simp
  also have ... = (∑ l = 0..k−1. f(l∗a mod k))
    using sum.reindex-bij-betw[OF bij,symmetric] by blast
  also have ... = (∑ l = 0..k−1. f(l∗a))
   by (intro sum.cong refl) (use mod-periodic-arithmetic[OF assms(2)] mod-mod-trivial
in blast)
  also have ... = (∑ l = 1..k. f(l∗a))
    using periodic-arithmetic-sum-periodic-arithmetic-shift[of (λl. f(l∗a)) k 1]
        periodic-arithmetic-homothecy[OF assms(2)] assms(3) by fastforce
  finally show ?thesis by blast
qed

end

theory Complex-Roots-Of-Unity
imports
  HOL−Analysis.Analysis
  Periodic-Arithmetic
```

begin

3 Complex roots of unity

definition
  unity-root k n = cis (2 ∗ pi ∗ of-int n / of-nat k)

lemma
  unity-root-k-0 [*simp*]: *unity-root k 0 = 1* **and**
  unity-root-0-n [*simp*]: *unity-root 0 n = 1*
  unfolding *unity-root-def* **by** *simp+*

lemma *unity-root-conv-exp*:
  unity-root k n = exp (of-real (2∗pi∗n/k) ∗ i)
  unfolding *unity-root-def*
  by (*subst cis-conv-exp*,*subst mult.commute*,*blast*)

lemma *unity-root-mod*:
  unity-root k (n mod int k) = unity-root k n
proof (*cases k = 0*)
  case *True* **then show** *?thesis* **by** *simp*
next
  case *False*
  obtain *q* :: *int* **where** *q-def*: *n = q∗k + (n mod k)*
    using *div-mult-mod-eq*[*symmetric*] **by** *blast*
  have *n / k = q + (n mod k) / k*
  proof (*auto simp add*: *divide-simps False*)
    have *real-of-int n = real-of-int (q∗k + (n mod k))*
      using *q-def* **by** *simp*
    also have ... *= real-of-int q ∗ real k + real-of-int (n mod k)*
      using *of-int-add of-int-mult* **by** *simp*
    finally show *real-of-int n = real-of-int q ∗ real k + real-of-int (n mod k)*
      by *blast*
  qed
  then have *(2∗pi∗n/k) = 2∗pi∗q + (2∗pi∗(n mod k)/k)*
    using *False* **by** (*auto simp add*: *field-simps*)
  then have *(2∗pi∗n/k)∗*i *= 2∗pi∗q∗*i *+ (2∗pi∗(n mod k)/k)∗*i (**is** *?l = ?r1 + ?r2*)
    by (*auto simp add*: *algebra-simps*)
  then have *exp ?l = exp ?r2*
    using *exp-plus-2pin* **by** (*simp add*: *exp-add mult.commute*)
  then show *?thesis*
    using *unity-root-def unity-root-conv-exp* **by** *simp*
qed

lemma *unity-root-cong*:
  assumes [*m = n*] (*mod int k*)
  shows *unity-root k m = unity-root k n*
proof −

```
  from assms have m mod int k = n mod int k
    by (auto simp: cong-def)
  hence unity-root k (m mod int k) = unity-root k (n mod int k)
    by simp
  thus ?thesis by (simp add: unity-root-mod)
qed

lemma unity-root-mod-nat:
  unity-root k (nat (n mod int k)) = unity-root k n
proof (cases k)
  case (Suc l)
  then have n mod int k ≥ 0 by auto
  show ?thesis
    unfolding int-nat-eq
    by (simp add: ‹n mod int k ≥ 0› unity-root-mod)
qed auto

lemma unity-root-eqD:
 assumes gr: k > 0
 assumes eq: unity-root k i = unity-root k j
 shows i mod k = j mod k
proof −
  let ?arg1 = (2∗pi∗i/k)∗ i
  let ?arg2 = (2∗pi∗j/k)∗ i
  from eq unity-root-conv-exp have exp ?arg1 = exp ?arg2 by simp
  from this exp-eq
  obtain n :: int where ?arg1 = ?arg2 +(2∗n∗pi)∗i by blast
  then have e1: ?arg1 − ?arg2 = 2∗n∗pi∗i by simp
  have e2: ?arg1 − ?arg2 = 2∗(i−j)∗(1/k)∗pi∗i
    by (auto simp add: algebra-simps)
  from e1 e2 have 2∗n∗pi∗i = 2∗(i−j)∗(1/k)∗pi∗i by simp
  then have 2∗n∗k∗pi∗i = 2∗(i−j)∗pi∗i
    by (simp add: divide-simps ‹k > 0›)(simp add: field-simps)
  then have 2∗n∗k = 2∗(i−j)
   by (meson complex-i-not-zero mult-cancel-right of-int-eq-iff of-real-eq-iff pi-neq-zero)
  then have n∗k = i−j by auto
  then show ?thesis by Groebner-Basis.algebra
qed

lemma unity-root-eq-1-iff:
  fixes k n :: nat
  assumes k > 0
  shows unity-root k n = 1 ⟷ k dvd n
proof −
  have unity-root k n = exp ((2∗pi∗n/k) ∗ i)
    by (simp add: unity-root-conv-exp)
  also have exp ((2∗pi∗n/k)∗ i) = 1 ⟷ k dvd n
    using complex-root-unity-eq-1[of k n] assms
    by (auto simp add: algebra-simps)
```

finally show *?thesis* **by** *simp*
qed

lemma *unity-root-nonzero* [*simp*]: *unity-root n k* $\neq$ *0*
by (*auto simp*: *unity-root-def*)

lemma *unity-root-pow*: *unity-root k n* ^ *m* = *unity-root k* (*n* $*$ *m*)
using *unity-root-def*
by (*simp add*: *Complex.DeMoivre mult.commute algebra-split-simps*(*6*))

lemma *unity-root-add*: *unity-root k* (*m* + *n*) = *unity-root k m* $*$ *unity-root k n*
by (*simp add*: *unity-root-conv-exp add-divide-distrib algebra-simps exp-add*)

lemma *unity-root-uminus*: *unity-root k* (−*m*) = *cnj* (*unity-root k m*)
unfolding *unity-root-conv-exp exp-cnj* **by** *simp*

lemma *inverse-unity-root*: *inverse* (*unity-root k m*) = *cnj* (*unity-root k m*)
unfolding *unity-root-conv-exp exp-cnj* **by** (*simp add*: *field-simps exp-minus*)

lemma *unity-root-diff*: *unity-root k* (*m* − *n*) = *unity-root k m* $*$ *cnj* (*unity-root k n*)
using *unity-root-add*[*of k m* −*n*] **by** (*simp add*: *unity-root-uminus*)

lemma *unity-root-eq-1-iff-int*:
fixes *k* :: *nat* **and** *n* :: *int*
assumes *k* > *0*
shows *unity-root k n* = *1* $\longleftrightarrow$ *k dvd n*
proof (*cases n* $\geq$ *0*)
case *True*
obtain *n'* **where** *n* = *int n'*
using *zero-le-imp-eq-int*[*OF True*] **by** *blast*
then show *?thesis*
using *unity-root-eq-1-iff*[*OF* ‹*k* > *0*›, *of n'*] *of-nat-dvd-iff* **by** *blast*
next
case *False*
then have −*n* $\geq$ *0* **by** *auto*
have *unity-root k n* = *inverse* (*unity-root k* (−*n*))
unfolding *inverse-unity-root* **by** (*simp add*: *unity-root-uminus*)
then have (*unity-root k n* = *1*) = (*unity-root k* (−*n*) = *1*)
by *simp*
also have (*unity-root k* (−*n*) = *1*) = (*k dvd* (−*n*))
using *unity-root-eq-1-iff*[*of k nat* (−*n*),*OF* ‹*k* > *0*›] *False*
int-dvd-int-iff[*of k nat* (−*n*)] *nat-0-le*[*OF* ‹−*n* $\geq$ *0*›] **by** *auto*
finally show *?thesis* **by** *simp*
qed

lemma *unity-root-eq-1* [*simp*]: *int k dvd n* $\Longrightarrow$ *unity-root k n* = *1*
by (*cases k* = *0*) (*auto simp*: *unity-root-eq-1-iff-int*)

lemma *unity-periodic-arithmetic*:
periodic-arithmetic (*unity-root k*) *k*
unfolding *periodic-arithmetic-def*
proof
fix *n*
have *unity-root k* (*n* + *k*) = *unity-root k* ((*n*+*k*) *mod k*)
using *unity-root-mod*[*of k*] *zmod-int* **by** *presburger*
also have *unity-root k* ((*n*+*k*) *mod k*) = *unity-root k n*
using *unity-root-mod zmod-int* **by** *auto*
finally show *unity-root k* (*n* + *k*) = *unity-root k n* **by** *simp*
qed

lemma *unity-periodic-arithmetic-mult*:
periodic-arithmetic (λ*n*. *unity-root k* (*m* ∗ *int n*)) *k*
unfolding *periodic-arithmetic-def*
proof
fix *n*
have *unity-root k* (*m* ∗ *int* (*n* + *k*)) =
unity-root k (*m*∗*n* + *m*∗*k*)
by (*simp add*: *algebra-simps*)
also have ... = *unity-root k* (*m*∗*n*)
using *unity-root-mod*[*of k m* ∗ *int n*] *unity-root-mod*[*of k m* ∗ *int n* + *m* ∗ *int*
k]
mod-mult-self3 **by** *presburger*
finally show *unity-root k* (*m* ∗ *int* (*n* + *k*)) =
unity-root k (*m* ∗ *int n*) **by** *simp*
qed

lemma *unity-root-periodic-arithmetic-mult-minus*:
shows *periodic-arithmetic* (λ*i*. *unity-root k* (−*int i*∗*int m*)) *k*
unfolding *periodic-arithmetic-def*
proof
fix *n*
have *unity-root k* (−(*n* + *k*) ∗ *m*) = *cnj* (*unity-root k* (*n*∗*m*+*k*∗*m*))
by (*simp add*: *ring-distribs unity-root-diff unity-root-add unity-root-uminus*)
also have ... = *cnj* (*unity-root k* (*n*∗*m*))
using *mult-period*[*of unity-root k k m*] *unity-periodic-arithmetic*[*of k*]
unfolding *periodic-arithmetic-def* **by** *presburger*
also have ... = *unity-root k* (−*n*∗*m*)
by (*simp add*: *unity-root-uminus*)
finally show *unity-root k* (−(*n* + *k*) ∗ *m*) = *unity-root k* (−*n*∗*m*)
by *simp*
qed

lemma *unity-div*:
fixes *a* :: *int* **and** *d* :: *nat*
assumes *d dvd k*
shows *unity-root k* (*a*∗*d*) = *unity-root* (*k div d*) *a*
proof −

have *1*: $(2*pi*(a*d)/k) = (2*pi*a)/(k\ div\ d)$
using *Suc-pred assms* **by** (*simp add*: *divide-simps*, *fastforce*)
have *unity-root k* $(a*d) = exp\ ((2*pi*(a*d)/k)*\ \mathrm{i})$
using *unity-root-conv-exp* **by** *simp*
also have ... $= exp\ (((2*pi*a)/(k\ div\ d))*\ \mathrm{i})$
using *1* **by** *simp*
also have ... $=$ *unity-root* $(k\ div\ d)\ a$
using *unity-root-conv-exp* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *unity-div-num*:
assumes $k > 0\ d > 0\ d\ dvd\ k$
shows *unity-root k* $(x * (k\ div\ d)) =$ *unity-root d x*
using *assms dvd-div-mult-self unity-div* **by** *auto*

4 Geometric sums of roots of unity

Apostol calls these 'geometric sums', which is a bit too generic. We therefore decided to refer to them as 'sums of roots of unity'.

definition *unity-root-sum k n* $= (\sum m{<}k.$ *unity-root k* $(n *$ *of-nat m*$))$

lemma *unity-root-sum-0-left* [*simp*]: *unity-root-sum* $0\ n = 0$ **and**
unity-root-sum-0-right [*simp*]: $k > 0 \Longrightarrow$ *unity-root-sum* $k\ 0 = k$
unfolding *unity-root-sum-def* **by** *simp-all*

Theorem 8.1

theorem *unity-root-sum*:
fixes k :: *nat* **and** n :: *int*
assumes *gr*: $k \geq 1$
shows $k\ dvd\ n \Longrightarrow$ *unity-root-sum k n* $= k$
and $\neg k\ dvd\ n \Longrightarrow$ *unity-root-sum k n* $= 0$
proof –
assume *dvd*: *k dvd n*
let $?x =$ *unity-root k n*
have *unit*: $?x = 1$ **using** *dvd gr unity-root-eq-1-iff-int* **by** *auto*
have *exp*: $?x\hat{}m =$ *unity-root k* $(n*m)$ **for** *m* **using** *unity-root-pow* **by** *simp*
have *unity-root-sum k n* $= (\sum m{<}k.$ *unity-root k* $(n*m))$
using *unity-root-sum-def* **by** *simp*
also have ... $= (\sum m{<}k.\ ?x\hat{}m)$ **using** *exp* **by** *auto*
also have ... $= (\sum m{<}k.\ 1)$ **using** *unit* **by** *simp*
also have ... $= k$ **using** *gr* **by** (*induction k*, *auto*)
finally show *unity-root-sum k n* $= k$ **by** *simp*
next
assume *dvd*: $\neg k\ dvd\ n$
let $?x =$ *unity-root k n*
have $?x \neq 1$ **using** *dvd gr unity-root-eq-1-iff-int* **by** *auto*
have $(?x\hat{}k - 1)/(?x - 1) = (\sum m{<}k.\ ?x\hat{}m)$

using *geometric-sum*[*of ?x k*, *OF* ‹*?x* ≠ *1*›] **by** *auto*
then have *sum*: *unity-root-sum k n* = (*?x*^*k* − *1*)/(*?x* − *1*)
using *unity-root-sum-def unity-root-pow* **by** *simp*
have *?x*^*k* = *1*
using *gr unity-root-eq-1-iff-int unity-root-pow* **by** *simp*
then show *unity-root-sum k n* = *0* **using** *sum* **by** *auto*
qed

corollary *unity-root-sum-periodic-arithmetic*:
periodic-arithmetic (*unity-root-sum k*) *k*
unfolding *periodic-arithmetic-def*
proof
fix *n*
show *unity-root-sum k* (*n* + *k*) = *unity-root-sum k n*
by (*cases k* = *0*; *cases k dvd n*) (*auto simp add*: *unity-root-sum*)
qed

lemma *unity-root-sum-nonzero-iff*:
fixes *r* :: *int*
assumes *k* ≥ *1* **and** *r* ∈ {−*k*<..<*k*}
shows *unity-root-sum k r* ≠ *0* ⟷ *r* = *0*
proof
assume *unity-root-sum k r* ≠ *0*
then have *k dvd r* **using** *unity-root-sum assms* **by** *blast*
then show *r* = *0* **using** *assms*(*2*)
using *dvd-imp-le-int* **by** *force*
next
assume *r* = *0*
then have *k dvd r* **by** *auto*
then have *unity-root-sum k r* = *k*
using *assms*(*1*) *unity-root-sum* **by** *blast*
then show *unity-root-sum k r* ≠ *0* **using** *assms*(*1*) **by** *simp*
qed

lemma *cyclotomic-poly-conv-prod-unity-root*:
fixes *n* :: *nat*
assumes *n*: *n* > *0*
defines *w* ≡ (λ*k*. *unity-root n* (*int k*))
shows *Polynomial.monom 1 n* − *1* = (∏ *k*<*n*. [:−*w k*, *1*:]) (**is** *?lhs* = *?rhs*)
proof (*rule ccontr*)
assume *neq*: *?lhs* ≠ *?rhs*
have *?lhs* = *Polynomial.monom 1 n* + (−*1*)
by *simp*
also have *degree* ... = *n*
by (*subst degree-add-eq-left*) (*use n* **in** ‹*auto simp*: *degree-monom-eq*›)
finally have *deg1*: *degree ?lhs* = *n* **.**

have *poly.coeff* (*?lhs* − *?rhs*) *n* = *0*
proof −

have *poly.coeff ?rhs n* = *lead-coeff ?rhs*
by (*simp add*: *degree-prod-sum-eq*)
also have . . . = *1*
by (*subst lead-coeff-prod*) *auto*
finally show *?thesis*
using *n* **by** *simp*
qed
moreover have *?lhs* − *?rhs* ≠ *0*
using *neq* **by** *simp*
ultimately have *degree* (*?lhs* − *?rhs*) ≠ *n*
by (*metis leading-coeff-0-iff*)
moreover have *degree* (*?lhs* − *?rhs*) ≤ *n*
by (*rule degree-diff-le*) (*use deg1* **in** ‹*auto simp*: *degree-prod-sum-eq*›)
ultimately have *degree* (*?lhs* − *?rhs*) < *n*
by *linarith*

have *root1*: *poly ?lhs* (*w k*) = *0* **for** *k*
using ‹*n* > *0*› **by** (*simp add*: *w-def poly-monom unity-root-pow*)
have *root2*: *poly ?rhs* (*w k*) = *0* **if** *k* < *n* **for** *k*
using *that* **by** (*auto simp*: *poly-prod*)

have *inj-on w* {..<*n*}
using *n* **by** (*auto simp*: *inj-on-def w-def dest*!: *unity-root-eqD*)
hence *card* {..<*n*} = *card* (*w* ‘ {..<*n*})
by (*subst card-image*) *auto*
also have *card* (*w* ‘ {..<*n*}) ≤ *card* {*z*. *poly* (*?lhs* − *?rhs*) *z* = *0*}
by (*intro card-mono poly-roots-finite*) (*use neq root1 root2* **in** *auto*)
also have *card* {*z*. *poly* (*?lhs* − *?rhs*) *z* = *0*} ≤ *degree* (*?lhs* − *?rhs*)
by (*rule card-poly-roots-bound*) (*use neq* **in** *auto*)
also have . . . < *n*
by *fact*
finally show *False*
by *simp*
qed

lemma *cyclotomic-poly-conv-prod-unity-root′*:
fixes *n* :: *nat*
assumes *n*: *n* > *0*
defines *w* ≡ (*λk*. *unity-root n* (*int k*))
shows *1* − *Polynomial.monom 1 n* = (∏*k*<*n*. [:*1*, −*w k*:]) (**is** *?lhs* = *?rhs*)
proof −
define *A* **where** *A* = *insert 0* (*w* ‘ {..<*n*})
have *card-A*: *card A* = *Suc n*
proof −
have *card A* = *Suc* (*card* (*w* ‘ {..<*n*}))
unfolding *A-def* **by** (*subst card.insert*) (*auto simp*: *w-def*)
also have *card* (*w* ‘ {..<*n*}) = *n*
by (*subst card-image*) (*use n* **in** ‹*auto simp*: *inj-on-def w-def dest*!: *unity-root-eqD*›)
finally show *?thesis* **.**

qed

show *?thesis*
proof (*rule poly-eqI-degree*)
have *degree* ($1 - $ *Polynomial.monom 1 n* :: *complex poly*) $\leq n$
by (*intro degree-diff-le*) (*auto simp*: *degree-monom-eq*)
thus *degree* ($1 - $ *Polynomial.monom 1 n* :: *complex poly*) $<$ *card A*
by (*simp add*: *card-A*)
next
have *degree* ($\prod k{<}n.$ [:*1*, $-$ *w k*:]) $\leq n$
by (*rule order.trans*[*OF degree-prod-sum-le*]) (*auto simp*: *w-def*)
thus *degree* ($\prod k{<}n.$ [:*1*, $-$ *w k*:]) $<$ *card A*
by (*simp add*: *card-A*)
next
fix *x* **assume** *x*: $x \in A$
have *poly* ($1 - $ *Polynomial.monom 1 n*) (*w k*) $= 0$ **for** *k*
by (*simp add*: *poly-monom w-def unity-root-pow*)
moreover have *poly* ($1 - $ *Polynomial.monom 1 n* :: *complex poly*) $0 = 1$
using *n* **by** (*simp add*: *poly-monom power-0-left*)
moreover have *poly* ($\prod k{<}n.$ [:*1*, $-$*w k*:]) (*w k*) $= 0$ **if** *k*: $k < n$ **for** *k*
proof $-$
define k' **where** $k' = ($*if* $k = 0$ *then 0 else* $n - k)$
have $w\ k * w\ k' = 1$ $k' < n$ **using** *n k*
by (*auto simp*: *k'-def w-def simp flip*: *unity-root-add intro*!: *unity-root-eq-1*)
thus *?thesis*
using *that* **by** (*auto simp*: *poly-prod*)
qed
moreover have *poly* ($\prod k{<}n.$ [:*1*, $-$*w k*:]) $0 = 1$
by (*simp add*: *poly-prod*)
ultimately show *poly* ($1 - $ *Polynomial.monom 1 n*) $x =$ *poly* ($\prod k{<}n.$ [:*1*, $-$ *w k*:]) *x*
using *x* **by** (*auto simp*: *A-def*)
qed
qed

end

5 Finite Fourier series

theory *Finite-Fourier-Series*
imports
Polynomial-Interpolation.Lagrange-Interpolation
Complex-Roots-Of-Unity
begin

5.1 Auxiliary facts

lemma *lagrange-exists*:
assumes *d*: *distinct* (*map fst zs-ws*)

```
  defines e: (p :: complex poly) ≡ lagrange-interpolation-poly zs-ws
  shows degree p ≤ (length zs-ws)−1
        (∀ x y. (x,y) ∈ set zs-ws ⟶ poly p x = y)
proof −
  from e show degree p ≤ (length zs-ws − 1)
    using degree-lagrange-interpolation-poly by auto
  from e d have
    poly p x = y if (x,y) ∈ set zs-ws for x y
    using that lagrange-interpolation-poly by auto
  then show (∀ x y. (x,y) ∈ set zs-ws ⟶ poly p x = y)
    by auto
qed

lemma lagrange-unique:
  assumes o: length zs-ws > 0
  assumes d: distinct (map fst zs-ws)
  assumes 1: degree (p1 :: complex poly) ≤ (length zs-ws)−1 ∧
             (∀ x y. (x,y) ∈ set zs-ws ⟶ poly p1 x = y)
  assumes 2: degree (p2 :: complex poly) ≤ (length zs-ws)−1 ∧
             (∀ x y. (x,y) ∈ set zs-ws ⟶ poly p2 x = y)
  shows p1 = p2
proof (cases p1 − p2 = 0)
  case True then show ?thesis by simp
next
  case False
    have poly (p1−p2) x = 0 if x ∈ set (map fst zs-ws) for x
      using 1 2 that by (auto simp add: field-simps)
    from this d have 3: card {x. poly (p1−p2) x = 0} ≥ length zs-ws
    proof (induction zs-ws)
      case Nil then show ?case by simp
    next
      case (Cons z-w zs-ws)
      from  False poly-roots-finite
      have f: finite {x. poly (p1 − p2) x = 0} by blast
      from Cons have set (map fst (z-w # zs-ws)) ⊆ {x. poly (p1 − p2) x = 0}
        by auto
      then have i: card (set (map fst (z-w # zs-ws))) ≤ card {x. poly (p1 − p2)
x = 0}
        using card-mono f by blast
      have length (z-w # zs-ws) ≤ card (set (map fst (z-w # zs-ws)))
        using Cons.prems(2) distinct-card by fastforce
      from this i show ?case by simp
    qed
    from 1 2 have 4: degree (p1 − p2) ≤ (length zs-ws)−1
      using degree-diff-le by blast

    have p1 − p2 = 0
    proof (rule ccontr)
      assume p1 − p2 ≠ 0
```

```
      then have card {x. poly (p1−p2) x = 0} ≤ degree (p1−p2)
        using poly-roots-degree by blast
      then have card {x. poly (p1−p2) x = 0} ≤ (length zs-ws)−1
        using 4 by auto
      then show False using 3 o by linarith
    qed
    then show ?thesis by simp
qed
```

Theorem 8.2

```
corollary lagrange:
  assumes length zs-ws > 0 distinct (map fst zs-ws)
  shows (∃! (p :: complex poly).
             degree p ≤ length zs-ws − 1 ∧
             (∀ x y. (x, y) ∈ set zs-ws ⟶ poly p x = y))
  using assms lagrange-exists lagrange-unique by blast

lemma poly-altdef':
 assumes gr: k ≥ degree p
 shows poly p (z::complex) = (∑ i≤k. coeff p i ∗ z ^ i)
proof −
  {fix z
  have 1: poly p z = (∑ i≤degree p. coeff p i ∗ z ^ i)
    using poly-altdef[of p z] by simp
  have poly p z = (∑ i≤k. coeff p i ∗ z ^ i)
    using gr
  proof (induction k)
    case 0 then show ?case by (simp add: poly-altdef)
  next
    case (Suc k)
    then show ?case
      using 1 le-degree not-less-eq-eq by fastforce
  qed}
  then show ?thesis using gr by blast
qed
```

5.2 Definition and uniqueness

```
definition finite-fourier-poly :: complex list ⇒ complex poly where
  finite-fourier-poly ws =
    (let k = length ws
     in  poly-of-list [1 / k ∗ (∑ m<k. ws ! m ∗ unity-root k (−n∗m)). n ← [0..<k]])

lemma degree-poly-of-list-le: degree (poly-of-list ws) ≤ length ws − 1
  by (intro degree-le) (auto simp: nth-default-def)

lemma degree-finite-fourier-poly: degree (finite-fourier-poly ws) ≤ length ws − 1
  unfolding finite-fourier-poly-def
proof (subst Let-def)
```

```
  let ?unrolled-list =
      (map (λn. complex-of-real (1 / real (length ws)) *
                (∑ m<length ws.
                    ws ! m *
                    unity-root (length ws) (− int n * int m)))
        [0..<length ws])
  have degree (poly-of-list ?unrolled-list) ≤ length ?unrolled-list − 1
    by (rule degree-poly-of-list-le)
  also have ... = length [0..<length ws] − 1
    using length-map by auto
  also have ... = length ws − 1 by auto
  finally show degree (poly-of-list ?unrolled-list) ≤ length ws − 1 by blast
qed

lemma coeff-finite-fourier-poly:
  assumes n < length ws
  defines k ≡ length ws
  shows coeff (finite-fourier-poly ws) n =
        (1/k) * (∑ m < k. ws ! m * unity-root k (−n*m))
  using assms degree-finite-fourier-poly
  by (auto simp: Let-def nth-default-def finite-fourier-poly-def)

lemma poly-finite-fourier-poly:
  fixes m :: int and ws
  defines k ≡ length ws
  assumes m ∈ {0..<k}
  assumes m < length ws
  shows poly (finite-fourier-poly ws) (unity-root k m) = ws ! (nat m)
proof −
  have k > 0 using assms by auto

  have distr:
   (∑ j<length ws. ws ! j * unity-root k (−i*j))*(unity-root k (m*i)) =
   (∑ j<length ws. ws ! j * unity-root k (−i*j)*(unity-root k (m*i)))
   for i
  using sum-distrib-right[of λj. ws ! j * unity-root k (−i*j)
                          {..<k} (unity-root k (m*i))]
  using k-def by blast

  {fix j i :: nat
   have unity-root k (−i*j)*(unity-root k (m*i)) = unity-root k (−i*j+m*i)
     by (simp add: unity-root-diff unity-root-uminus field-simps)
   also have ... = unity-root k (i*(m−j))
     by (simp add: algebra-simps)
   finally have unity-root k (−i*j)*(unity-root k (m*i)) = unity-root k (i*(m−j))
     by simp
   then have ws ! j * unity-root k (−i*j)*(unity-root k (m*i)) =
             ws ! j * unity-root k (i*(m−j))
     by auto
```

```
} note prod = this

have zeros:
  (unity-root-sum k (m−j) ≠ 0 ⟷ m = j)
    if j ≥ 0 ∧ j < k  for j
  using k-def that assms unity-root-sum-nonzero-iff[of - m−j] by simp
 then have sum-eq:
   (∑j≤k−1. ws ! j ∗ unity-root-sum k (m−j)) =
        (∑j∈{nat m}.  ws ! j ∗ unity-root-sum k (m−j))
   using assms(2) by (intro sum.mono-neutral-right,auto)

 have poly (finite-fourier-poly ws) (unity-root k m) =
       (∑i≤k−1. coeff (finite-fourier-poly ws) i ∗ (unity-root k m) ^ i)
   using degree-finite-fourier-poly[of ws] k-def
         poly-altdef'[of finite-fourier-poly ws k−1 unity-root k m] by blast
 also have ... = (∑i<k. coeff (finite-fourier-poly ws) i ∗ (unity-root k m) ^ i)
   using assms(2) by (intro sum.cong) auto
 also have ... = (∑i<k. 1 / k ∗
   (∑j<k. ws ! j ∗ unity-root k (−i∗j)) ∗ (unity-root k m) ^ i)
   using coeff-finite-fourier-poly[of - ws] k-def by auto
 also have ... = (∑i<k. 1 / k ∗
   (∑j<k. ws ! j ∗ unity-root k (−i∗j))∗(unity-root k (m∗i)))
   using unity-root-pow by auto
 also have ... = (∑i<k. 1 / k ∗
   (∑j<k. ws ! j ∗ unity-root k (−i∗j)∗(unity-root k (m∗i))))
   using distr k-def by simp
 also have ... = (∑i<k. 1 / k ∗
   (∑j<k. ws ! j ∗ unity-root k (i∗(m−j))))
   using prod by presburger
 also have ... = 1 / k ∗ (∑i<k.
   (∑j<k. ws ! j ∗ unity-root k (i∗(m−j))))
   by (simp add: sum-distrib-left)
 also have ... = 1 / k ∗ (∑j<k.
   (∑i<k. ws ! j ∗ unity-root k (i∗(m−j))))
   using sum.swap by fastforce
 also have ... = 1 / k ∗ (∑j<k. ws ! j ∗ (∑i<k. unity-root k (i∗(m−j))))
   by (simp add: vector-space-over-itself.scale-sum-right)
 also have ... = 1 / k ∗ (∑j<k. ws ! j ∗ unity-root-sum k (m−j))
   unfolding unity-root-sum-def by (simp add: algebra-simps)
  also have (∑j<k. ws ! j ∗ unity-root-sum k (m−j)) = (∑j≤k−1. ws ! j ∗
unity-root-sum k (m−j))
   using ‹k > 0› by (intro sum.cong) auto
 also have ... = (∑j∈{nat m}.  ws ! j ∗ unity-root-sum k (m−j))
   using sum-eq .
 also have ... = ws ! (nat m) ∗ k
   using assms(2) by (auto simp: algebra-simps)
 finally have poly (finite-fourier-poly ws) (unity-root k m) = ws ! (nat m)
   using assms(2) by auto
 then show ?thesis by simp
```

qed

Theorem 8.3

```
theorem finite-fourier-poly-unique:
  assumes length ws > 0
  defines k ≡ length ws
  assumes (degree p ≤ k − 1)
  assumes (∀ m ≤ k−1. (ws ! m) = poly p (unity-root k m))
  shows p = finite-fourier-poly ws
proof −
  let ?z = map (λm. unity-root k m) [0..<k]
  have k: k > 0 using assms by auto
  from k have d1: distinct ?z
    unfolding distinct-conv-nth using unity-root-eqD[OF k] by force
  let ?zs-ws = zip ?z ws
  from d1 k-def have d2: distinct (map fst ?zs-ws) by simp
  have l2: length ?zs-ws > 0 using assms(1) k-def by auto
  have l3: length ?zs-ws = k by (simp add: k-def)

  from degree-finite-fourier-poly have degree: degree (finite-fourier-poly ws) ≤ k −
1
    using k-def by simp

  have interp: poly (finite-fourier-poly ws) x = y
    if (x, y) ∈ set ?zs-ws for x y
  proof −
    from that obtain n where
         x = map (unity-root k ∘ int) [0..<k] ! n ∧
         y = ws ! n ∧
         n < length ws
      using in-set-zip[of (x,y) (map (unity-root k) (map int [0..<k])) ws]
      by auto
    then have
         x = unity-root k (int n) ∧
         y = ws ! n ∧
         n < length ws
      using nth-map[of n [0..<k] unity-root k ∘ int ] k-def by simp
    thus poly (finite-fourier-poly ws) x = y
      by (simp add: poly-finite-fourier-poly k-def)
  qed

  have interp-p: poly p x = y if (x,y) ∈ set ?zs-ws for x y
  proof −
    from that obtain n where
         x = map (unity-root k ∘ int) [0..<k] ! n ∧
         y = ws ! n ∧
         n < length ws
      using in-set-zip[of (x,y) (map (unity-root k) (map int [0..<k])) ws]
      by auto
```

```
    then have rw: x = unity-root k (int n) y = ws ! n n < length ws
      using nth-map[of n [0..<k] unity-root k ∘ int ] k-def by simp+
    show poly p x = y
      unfolding rw(1,2) using assms(4) rw(3) k-def by simp
  qed

  from lagrange-unique[of - p finite-fourier-poly ws] d2 l2
  have l:
    degree p ≤ k − 1 ∧
    (∀ x y. (x, y) ∈ set ?zs-ws ⟶ poly p x = y) ⟹
    degree (finite-fourier-poly ws) ≤ k − 1 ∧
    (∀ x y. (x, y) ∈ set ?zs-ws ⟶ poly (finite-fourier-poly ws) x = y) ⟹
    p = (finite-fourier-poly ws)
    using l3 by metis
  from assms degree interp interp-p l3
  show p = (finite-fourier-poly ws) using l by blast
qed
```

The following alternative formulation returns a coefficient

```
definition finite-fourier-poly′ :: (nat ⇒ complex) ⇒ nat ⇒ complex poly where
  finite-fourier-poly′ ws k =
     (poly-of-list [1 / k ∗ (∑ m<k. (ws m) ∗ unity-root k (−n∗m)). n ← [0..<k]])

lemma finite-fourier-poly′-conv-finite-fourier-poly:
  finite-fourier-poly′ ws k = finite-fourier-poly [ws n. n ← [0..<k]]
  unfolding finite-fourier-poly-def finite-fourier-poly′-def by simp

lemma coeff-finite-fourier-poly′:
  assumes n < k
  shows coeff (finite-fourier-poly′ ws k) n =
         (1/k) ∗ (∑ m < k. (ws m) ∗ unity-root k (−n∗m))
proof −
  let ?ws = [ws n. n ← [0..<k]]
  have coeff (finite-fourier-poly′ ws k) n =
         coeff (finite-fourier-poly ?ws) n
    by (simp add: finite-fourier-poly′-conv-finite-fourier-poly)
  also have coeff (finite-fourier-poly ?ws) n =
    1 / k ∗ (∑ m<k. (?ws ! m) ∗ unity-root k (− n∗m))
    using assms by (auto simp: coeff-finite-fourier-poly)
  also have ... = (1/k) ∗ (∑ m < k. (ws m) ∗ unity-root k (−n∗m))
    using assms by simp
  finally show ?thesis by simp
qed

lemma degree-finite-fourier-poly′: degree (finite-fourier-poly′ ws k) ≤ k − 1
  using degree-finite-fourier-poly[of [ws n. n ← [0..<k]]]
  by (auto simp: finite-fourier-poly′-conv-finite-fourier-poly)

lemma poly-finite-fourier-poly′:
```

fixes *m* :: *int* **and** *k*
assumes $m \in \{0..<k\}$
shows *poly* (*finite-fourier-poly′ ws k*) (*unity-root k m*) = *ws* (*nat m*)
using *assms poly-finite-fourier-poly*[*of m* [*ws n. n* ← [*0..<k*]]]
by (*auto simp*: *finite-fourier-poly′-conv-finite-fourier-poly poly-finite-fourier-poly*)

lemma *finite-fourier-poly′-unique*:
assumes $k > 0$
assumes *degree* $p \leq k - 1$
assumes $\forall m \leq k-1.$ *ws m* = *poly p* (*unity-root k m*)
shows *p* = *finite-fourier-poly′ ws k*
proof −
let *?ws* = [*ws n. n* ← [*0..<k*]]
from *finite-fourier-poly-unique* **have** *p* = *finite-fourier-poly ?ws* **using** *assms* **by** *simp*
also have . . . = *finite-fourier-poly′ ws k*
using *finite-fourier-poly′-conv-finite-fourier-poly* **..**
finally show *p* = *finite-fourier-poly′ ws k* **by** *blast*
qed

lemma *fourier-unity-root*:
fixes *k* :: *nat*
assumes $k > 0$
shows *poly* (*finite-fourier-poly′ f k*) (*unity-root k m*) =
($\sum$ *n*<*k*.*1*/*k*∗($\sum$ *m*<*k*.(*f m*)∗*unity-root k* (−*n*∗*m*))∗*unity-root k* (*m*∗*n*))
proof −
have *poly* (*finite-fourier-poly′ f k*) (*unity-root k m*) =
($\sum$ *n*≤*k*−*1*. *coeff* (*finite-fourier-poly′ f k*) *n* ∗(*unity-root k m*)^*n*)
using *poly-altdef′*[*of finite-fourier-poly′ f k k*−*1 unity-root k m*]
degree-finite-fourier-poly′[*of f k*] **by** *simp*
also have . . . = ($\sum$ *n*≤*k*−*1*. *coeff* (*finite-fourier-poly′ f k*) *n* ∗(*unity-root k* (*m*∗*n*)))
using *unity-root-pow* **by** *simp*
also have . . . = ($\sum$ *n*<*k*. *coeff* (*finite-fourier-poly′ f k*) *n* ∗(*unity-root k* (*m*∗*n*)))

using *assms* **by** (*intro sum.cong*) *auto*
also have . . . = ($\sum$ *n*<*k*.(*1*/*k*)∗($\sum$ *m*<*k*.(*f m*)∗*unity-root k* (−*n*∗*m*))∗(*unity-root k* (*m*∗*n*)))
using *coeff-finite-fourier-poly′*[*of - k f*] **by** *simp*
finally show
poly (*finite-fourier-poly′ f k*) (*unity-root k m*) =
($\sum$ *n*<*k*.*1*/*k*∗($\sum$ *m*<*k*.(*f m*)∗*unity-root k* (−*n*∗*m*))∗*unity-root k* (*m*∗*n*))
by *blast*
qed

5.3 Expansion of an arithmetical function

Theorem 8.4

theorem *fourier-expansion-periodic-arithmetic*:

```
  assumes k > 0
  assumes periodic-arithmetic f k
  defines g ≡ (λn. (1 / k) ∗ (∑ m<k. f m ∗ unity-root k (−n ∗ m)))
    shows periodic-arithmetic g k
      and f m = (∑ n<k. g n ∗ unity-root k (m ∗ n))
proof −
 {fix l
  from unity-periodic-arithmetic mult-period
  have period: periodic-arithmetic (λx. unity-root k x) (k∗l) by simp}
  note period = this
 {fix n l
  have unity-root k (−(n+k)∗l) = cnj (unity-root k ((n+k)∗l))
    by (simp add: unity-root-uminus unity-root-diff ring-distribs unity-root-add)
  also have unity-root k ((n+k)∗l) = unity-root k (n∗l)
    by (intro unity-root-cong) (auto simp: cong-def algebra-simps)
  also have cnj ... = unity-root k (−n∗l)
    using unity-root-uminus by simp
  finally have unity-root k (−(n+k)∗l) = unity-root k (−n∗l) by simp}
  note u-period = this

  show 1: periodic-arithmetic g k
    unfolding periodic-arithmetic-def
  proof
    fix n

    have g(n+k) = (1 / k) ∗ (∑ m<k. f(m) ∗ unity-root k (−(n+k)∗m))
      using assms(3) by fastforce
    also have ... = (1 / k) ∗ (∑ m<k. f(m) ∗ unity-root k (−n∗m))
    proof −
      have (∑ m<k. f(m) ∗ unity-root k (−(n+k)∗m)) =
            (∑ m<k. f(m) ∗ unity-root k (−n∗m))
        by (intro sum.cong) (use u-period in auto)
      then show ?thesis by argo
    qed
    also have ... = g(n)
      using assms(3) by fastforce
    finally show g(n+k) = g(n) by simp
  qed

  show f(m) = (∑ n<k. g(n)∗ unity-root k (m ∗ int n))
  proof −
    {
      fix m
      assume range: m ∈ {0..<k}
      have f(m) = (∑ n<k. g(n)∗ unity-root k (m ∗ int n))
      proof −
        have f m = poly (finite-fourier-poly' f k) (unity-root k m)
          using range by (simp add: poly-finite-fourier-poly')
       also have ... = (∑ n<k. (1 / k) ∗ (∑ m<k. f(m) ∗ unity-root k (−n∗m))∗
```

unity-root k $(m{*}n)$)
using *fourier-unity-root assms*(*1*) **by** *blast*
also have ... $= (\sum n{<}k.\ g(n){*}$ *unity-root k* $(m{*}n))$
using *assms* **by** *simp*
finally show *?thesis* **by** *auto*
qed}
note *concentrated* = *this*

have *periodic-arithmetic* $(\lambda m.\ (\sum n{<}k.\ g(n){*}$ *unity-root k* $(m * int\ n)))$ k
proof −
have *periodic-arithmetic* $(\lambda n.\ g(n){*}$ *unity-root k* $(i * int\ n))$ k **for** $i :: int$
using *1 unity-periodic-arithmetic mult-periodic-arithmetic*
unity-periodic-arithmetic-mult **by** *auto*
then have *p-s*: $\forall\, i{<}k.$ *periodic-arithmetic* $(\lambda n.\ g(n){*}$ *unity-root k* $(i * int\ n))$ k
by *simp*
have *periodic-arithmetic* $(\lambda i.\ \sum n{<}k.\ g(n){*}$ *unity-root k* $(i * int\ n))$ k
unfolding *periodic-arithmetic-def*
proof
fix n
show $(\sum na{<}k.\ g\ na *$ *unity-root k* $(int\ (n + k) * int\ na)) =$
$(\sum na{<}k.\ g\ na *$ *unity-root k* $(int\ n * int\ na))$
by (*intro sum.cong refl*, *simp add*: *distrib-right flip*: *of-nat-mult of-nat-add*)
(*insert period*, *unfold periodic-arithmetic-def*, *blast*)
qed
then show *?thesis* **by** *simp*
qed

from *this assms*(*1*−*2*) *concentrated*
unique-periodic-arithmetic-extension[*of k f* $(\lambda i.\ \sum n{<}k.\ g(n){*}$ *unity-root k* $(i$
$* int\ n))$ *m*]
show $f\ m = (\sum n{<}k.\ g\ n *$ *unity-root k* $(int\ m * int\ n))$ **by** *simp*
qed
qed

theorem *fourier-expansion-periodic-arithmetic-unique*:
fixes $f\ g :: nat \Rightarrow complex$
assumes $k > 0$
assumes *periodic-arithmetic f k* **and** *periodic-arithmetic g k*
assumes $\bigwedge m.\ m < k \Longrightarrow f\ m = (\sum n{<}k.\ g\ n *$ *unity-root k* $(int\ (m * n)))$
shows $g\ n = (1\ /\ k) * (\sum m{<}k.\ f\ m *$ *unity-root k* $(-n * m))$
proof −
let $?p$ = *poly-of-list* $[g(n).\ n \leftarrow [0..{<}k]]$
have *d*: *degree* $?p \le k{-}1$
proof −
have *degree* $?p \le$ *length* $[g(n).\ n \leftarrow [0..{<}k]] - 1$
using *degree-poly-of-list-le* **by** *blast*
also have ... = *length* $[0..{<}k] - 1$
using *length-map* **by** *auto*
finally show *?thesis* **by** *simp*

```
qed
have c: coeff ?p i = (if i < k then g(i) else 0) for i
  by (simp add: nth-default-def)
{fix z
have poly ?p z = (∑ n≤k−1. coeff ?p n∗ z^n)
  using poly-altdef'[of ?p k−1] d by blast
also have ... = (∑ n<k. coeff ?p n∗ z^n)
  using ‹k > 0› by (intro sum.cong) auto
also have ... = (∑ n<k. (if n < k then g(n) else 0)∗ z^n)
  using c by simp
also have ... = (∑ n<k. g(n)∗ z^n)
  by (simp split: if-splits)
finally have poly ?p z = (∑ n<k. g n ∗ z ^ n) .}
note eval = this
{fix i
have poly ?p (unity-root k i) = (∑ n<k. g(n)∗ (unity-root k i)^n)
  using eval by blast
then have poly ?p (unity-root k i) = (∑ n<k. g(n)∗ (unity-root k (i∗n)))
  using unity-root-pow by auto}
note interpolation = this

{
  fix m
  assume b: m ≤ k−1
  from d assms(1)
  have f m = (∑ n<k. g(n) ∗ unity-root k (m∗n))
    using assms(4) b by auto
  also have ... = poly ?p (unity-root k m)
    using interpolation by simp
  finally have f m = poly ?p (unity-root k m) by auto
}

from this finite-fourier-poly'-unique[of k - f]
have p-is-fourier: ?p = finite-fourier-poly' f k
  using assms(1) d by blast

{
  fix n
  assume b: n ≤ k−1
  have f-1: coeff ?p n = (1 / k) ∗ (∑ m<k. f(m) ∗ unity-root k (−n∗m))
    using p-is-fourier using assms(1) b by (auto simp: coeff-finite-fourier-poly')
  then have g(n) = (1 / k) ∗ (∑ m<k. f(m) ∗ unity-root k (−n∗m))
    using c b assms(1)
  proof −
    have 1: coeff ?p n = (1 / k) ∗ (∑ m<k. f(m) ∗ unity-root k (−n∗m))
      using f-1 by blast
    have 2: coeff ?p n =  g n
      using c assms(1) b by simp
    show ?thesis using 1 2 by argo
```

```
    qed
  }

  have periodic-arithmetic (λn. (1 / k) * (∑m<k. f(m) * unity-root k (−n*m)))
k
  proof −
    have periodic-arithmetic (λi. unity-root k (−int i*int m)) k for m
      using unity-root-periodic-arithmetic-mult-minus by simp
    then have periodic-arithmetic (λi. f(m) * unity-root k (−i*m)) k for m
      by (simp add: periodic-arithmetic-def)
     then show periodic-arithmetic (λi. (1 / k) * (∑m<k. f m * unity-root k
(−i*m))) k
      by (intro scalar-mult-periodic-arithmetic fin-sum-periodic-arithmetic-set) auto
  qed
  note periodich = this
  let ?h = (λi. (1 / k) *(∑m<k. f m * unity-root k (−i*m)))
  from unique-periodic-arithmetic-extension[of k g ?h n]
       assms(3) assms(1) periodich
  have g n = (1/k) * (∑m<k. f m * unity-root k (−n*m))
    by (simp add: ‹⋀na. na ≤ k − 1 ⟹ g na = complex-of-real (1 / real k) *
(∑m<k. f m * unity-root k (− int na * int m))›)
  then show ?thesis by simp
qed

end
```

6 Ramanujan sums

```
theory Ramanujan-Sums
imports
  Dirichlet-Series.Moebius-Mu
  Gauss-Sums-Auxiliary
  Finite-Fourier-Series
begin
```

6.1 Basic sums

```
definition ramanujan-sum :: nat ⇒ nat ⇒ complex
  where ramanujan-sum k n = (∑m | m ∈ {1..k} ∧ coprime m k. unity-root k
(m*n))

notation ramanujan-sum (‹c›)

lemma ramanujan-sum-0-n [simp]: c 0 n = 0
  unfolding ramanujan-sum-def by simp

lemma sum-coprime-conv-dirichlet-prod-moebius-mu:
  fixes F S :: nat ⇒ complex and f :: nat ⇒ nat ⇒ complex
```

defines $F \equiv (\lambda n.\ (\sum k \in \{1..n\}.\ f\ k\ n))$
defines $S \equiv (\lambda n.\ (\sum k \mid k \in \{1..n\} \wedge coprime\ k\ n\ .\ f\ k\ n))$
assumes $\bigwedge a\ b\ d.\ d\ dvd\ a \Longrightarrow d\ dvd\ b \Longrightarrow f\ (a\ div\ d)\ (b\ div\ d) = f\ a\ b$
shows $S\ n = dirichlet\text{-}prod\ moebius\text{-}mu\ F\ n$
proof $(cases\ n = 0)$
case *True*
then show *?thesis*
using $assms(2)$ **unfolding** *dirichlet-prod-def* **by** *fastforce*
next
case *False*
have $S(n) = (\sum k \mid k \in \{1..n\} \wedge coprime\ k\ n\ .\ (f\ k\ n))$
using *assms* **by** *blast*
also have $\ldots = (\sum k \in \{1..n\}.\ (f\ k\ n) * dirichlet\text{-}prod\text{-}neutral\ (gcd\ k\ n))$
using *dirichlet-prod-neutral-intro* **by** *blast*
also have $\ldots = (\sum k \in \{1..n\}.\ (f\ k\ n) * (\sum d \mid d\ dvd\ (gcd\ k\ n).\ moebius\text{-}mu\ d))$
proof −
{
fix k
have $dirichlet\text{-}prod\text{-}neutral\ (gcd\ k\ n) = (if\ gcd\ k\ n = 1\ then\ 1\ else\ 0)$
using *dirichlet-prod-neutral-def*[*of gcd k n*] **by** *blast*
also have $\ldots = (\sum d \mid d\ dvd\ gcd\ k\ n.\ moebius\text{-}mu\ d)$
using *sum-moebius-mu-divisors'*[*of gcd k n*] **by** *auto*
finally have $dirichlet\text{-}prod\text{-}neutral\ (gcd\ k\ n) = (\sum d \mid d\ dvd\ gcd\ k\ n.\ moebius\text{-}mu\ d)$
by *auto*
} **note** *summand = this*
then show *?thesis* **by** (*simp add*: *summand*)
qed
also have $\ldots = (\sum k = 1..n.\ (\sum d \mid d\ dvd\ gcd\ k\ n.\ (f\ k\ n) * moebius\text{-}mu\ d))$
by (*simp add*: *sum-distrib-left*)
also have $\ldots = (\sum k = 1..n.\ (\sum d \mid d\ dvd\ gcd\ n\ k.\ (f\ k\ n) * moebius\text{-}mu\ d))$
using *gcd.commute*[*of - n*] **by** *simp*
also have $\ldots = (\sum d \mid d\ dvd\ n.\ \sum k \mid k \in \{1..n\} \wedge d\ dvd\ k.\ (f\ k\ n) * moebius\text{-}mu\ d)$
using *sum.swap-restrict*[*of* $\{1..n\}$ $\{d.\ d\ dvd\ n\}$
$\lambda k\ d.\ (f\ k\ n)*moebius\text{-}mu\ d\ \lambda k\ d.\ d\ dvd\ k$] *False* **by** *auto*
also have $\ldots = (\sum d \mid d\ dvd\ n.\ moebius\text{-}mu\ d * (\sum k \mid k \in \{1..n\} \wedge d\ dvd\ k.\ (f\ k\ n)))$
by (*simp add*: *sum-distrib-left mult.commute*)
also have $\ldots = (\sum d \mid d\ dvd\ n.\ moebius\text{-}mu\ d * (\sum q \in \{1..n\ div\ d\}.\ (f\ q\ (n\ div\ d))))$
proof −
have *st*:
$(\sum k \mid k \in \{1..n\} \wedge d\ dvd\ k.\ (f\ k\ n)) =$
$(\sum q \in \{1..n\ div\ d\}.\ (f\ q\ (n\ div\ d)))$
if $d\ dvd\ n\ d > 0$ **for** $d :: nat$
by (*rule sum.reindex-bij-witness*[*of* - $\lambda k.\ k * d\ \lambda k.\ k\ div\ d$])
(*use assms*(*3*) *that* **in** ‹*fastforce simp*: *div-le-mono*›)+
show *?thesis*

```
    by (intro sum.cong) (use st False in fastforce)+
  qed
  also have ... = (∑ d | d dvd n. moebius-mu d * F(n div d))
  proof -
    have F (n div d) = (∑ q ∈ {1..n div d}. (f q (n div d)))
      if d dvd n for d
        by (simp add: F-def real-of-nat-div that)
     then show ?thesis by auto
  qed
  also have ... = dirichlet-prod moebius-mu F n
    by (simp add: dirichlet-prod-def)
  finally show ?thesis by simp
qed

lemma dirichlet-prod-neutral-sum:
  dirichlet-prod-neutral n = (∑ k = 1..n. unity-root n k) for n :: nat
proof (cases n = 0)
  case True then show ?thesis unfolding dirichlet-prod-neutral-def by simp
next
  case False
  have 1: unity-root n 0 = 1 by simp
  have 2: unity-root n n = 1
    using unity-periodic-arithmetic[of n] add.left-neutral
  proof -
    have 1 = unity-root n (int 0)
       using 1 by auto
    also have unity-root n (int 0) = unity-root n (int (0 + n))
      using unity-periodic-arithmetic[of n] periodic-arithmetic-def by algebra
    also have ... = unity-root n (int n) by simp
    finally show ?thesis by auto
  qed
  have (∑ k = 1..n. unity-root n k) = (∑ k = 0..n. unity-root n k) − 1
    by (simp add: sum.atLeast-Suc-atMost sum.atLeast0-atMost-Suc-shift 1)
  also have ... = ((∑ k = 0..n−1. unity-root n k)+1) − 1
    using sum.atLeast0-atMost-Suc[of (λk. unity-root n k) n−1] False
    by (simp add: 2)
  also have ... = (∑ k = 0..n−1. unity-root n k)
    by simp
  also have ... = unity-root-sum n 1
    unfolding unity-root-sum-def using ‹n ≠ 0› by (intro sum.cong) auto
  also have ... = dirichlet-prod-neutral n
    using unity-root-sum[of n 1] False
    by (cases n = 1,auto simp add: False dirichlet-prod-neutral-def)
  finally have 3: dirichlet-prod-neutral n = (∑ k = 1..n. unity-root n k) by auto
  then show ?thesis by blast
qed

lemma moebius-coprime-sum:
  moebius-mu n = (∑ k | k ∈ {1..n} ∧ coprime k n . unity-root n (int k))
```

proof −
  let *?f* = (λ*k n*. *unity-root n k*)
  from *div-dvd-div* **have**
      d dvd a ⟹ *d dvd b* ⟹
      unity-root (*a div d*) (*b div d*) =
      unity-root a b **for** *a b d* :: *nat*
    using *unity-root-def real-of-nat-div* **by** *fastforce*
  then have (∑*k* | *k* ∈ {*1..n*} ∧ *coprime k n*. *?f k n*) =
        dirichlet-prod moebius-mu (λ*n*. ∑*k* = *1..n*. *?f k n*) *n*
    using *sum-coprime-conv-dirichlet-prod-moebius-mu*[*of ?f n*] **by** *blast*
  also have . . . = *dirichlet-prod moebius-mu dirichlet-prod-neutral n*
    by (*simp add*: *dirichlet-prod-neutral-sum*)
  also have . . . = *moebius-mu n*
    by (*cases n* = *0*) (*simp-all add*: *dirichlet-prod-neutral-right-neutral*)
  finally have *moebius-mu n* = (∑*k* | *k* ∈ {*1..n*} ∧ *coprime k n*. *?f k n*)
    by *argo*
  then show *?thesis* **by** *blast*
qed

corollary *ramanujan-sum-1-right* [*simp*]: *c k* (*Suc 0*) = *moebius-mu k*
  unfolding *ramanujan-sum-def* **using** *moebius-coprime-sum*[*of k*] **by** *simp*

lemma *ramanujan-sum-dvd-eq-totient*:
  assumes *k dvd n*
    shows *c k n* = *totient k*
  unfolding *ramanujan-sum-def*
proof −
  have *unity-root k* (*m*∗*n*) = *1* **for** *m*
    using *assms* **by** (*cases k* = *0*) (*auto simp*: *unity-root-eq-1-iff-int*)
  then have (∑*m* | *m* ∈ {*1..k*} ∧ *coprime m k*. *unity-root k* (*m* ∗ *n*)) =
          (∑*m* | *m* ∈ {*1..k*} ∧ *coprime m k*. *1*) **by** *simp*
  also have . . . = *card* {*m*. *m* ∈ {*1..k*} ∧ *coprime m k*} **by** *simp*
  also have . . . = *totient k*
   unfolding *totient-def totatives-def*
  proof −
    have {*1..k*} = {*0*<*..k*} **by** *auto*
    then show *of-nat* (*card* {*m* ∈ {*1..k*}. *coprime m k*}) =
          of-nat (*card* {*ka* ∈ {*0*<*..k*}. *coprime ka k*}) **by** *auto*
  qed
  finally show (∑*m* | *m* ∈ {*1..k*} ∧ *coprime m k*. *unity-root k* (*m* ∗ *n*)) = *totient*
k
    by *auto*
qed

6.2 Generalised sums

definition *gen-ramanujan-sum* :: (*nat* ⇒ *complex*) ⇒ (*nat* ⇒ *complex*) ⇒ *nat* ⇒
nat ⇒ *complex* **where**
  gen-ramanujan-sum f g = (λ*k n*. ∑*d* | *d dvd gcd n k*. *f d* ∗ *g* (*k div d*))

```
notation gen-ramanujan-sum (‹s›)

lemma gen-ramanujan-sum-k-1: s f g k 1 = f 1 ∗ g k
  unfolding gen-ramanujan-sum-def by auto

lemma gen-ramanujan-sum-1-n: s f g 1 n = f 1 ∗ g 1
  unfolding gen-ramanujan-sum-def by simp

lemma gen-ramanujan-sum-periodic: periodic-arithmetic (s f g k) k
  unfolding gen-ramanujan-sum-def periodic-arithmetic-def by simp
```

Theorem 8.5

```
theorem gen-ramanujan-sum-fourier-expansion:
  fixes f g :: nat ⇒ complex and a :: nat ⇒ nat ⇒ complex
  assumes k > 0
  defines a ≡ (λk m. (1/k) ∗ (∑ d| d dvd (gcd m k). g d ∗ f (k div d) ∗ d))
  shows s f g k n = (∑ m≤k−1. a k m ∗ unity-root k (m∗n))
proof −
  let ?g = (λx. 1 / of-nat k ∗ (∑ m<k. s f g k m ∗ unity-root k (−x∗m)))
  {fix m :: nat
  let ?h = λn d. f d ∗ g (k div d) ∗ unity-root k (− m ∗ int n)
  have (∑ l<k. s f g k l ∗ unity-root k (−m∗l)) =
            (∑ l ∈ {0..k−1}. s f g k l ∗ unity-root k (−m∗l))
    using ‹k > 0› by (intro sum.cong) auto
  also have ... = (∑ l ∈ {1..k}. s f g k l ∗ unity-root k (−m∗l))
  proof −
    have periodic-arithmetic (λl. unity-root k (−m∗l)) k
      using unity-periodic-arithmetic-mult by blast
    then have periodic-arithmetic (λl. s f g k l ∗ unity-root k (−m∗l)) k
      using gen-ramanujan-sum-periodic mult-periodic-arithmetic by blast
    from this periodic-arithmetic-sum-periodic-arithmetic-shift[of - k 1 ]
    have sum (λl. s f g k l ∗ unity-root k (−m∗l)) {0..k − 1} =
          sum (λl. s f g k l ∗ unity-root k (−m∗l)) {1..k}
      using assms(1) zero-less-one by simp
    then show ?thesis by argo
  qed
   also have ... = (∑ n∈{1..k}. (∑ d | d dvd (gcd n k). f(d) ∗ g(k div d)) ∗
unity-root k (−m∗n))
    by (simp add: gen-ramanujan-sum-def)
   also have ... = (∑ n∈{1..k}. (∑ d | d dvd (gcd n k). f(d) ∗ g(k div d) ∗
unity-root k (−m∗n)))
    by (simp add: sum-distrib-right)
  also have ... = (∑ d | d dvd k. ∑ n | n ∈ {1..k} ∧ d dvd n. ?h n d)
  proof −
    have (∑ n = 1..k. ∑ d | d dvd gcd n k. ?h n d) =
          (∑ n = 1..k. ∑ d | d dvd k ∧ d dvd n . ?h n d)
      using gcd.commute[of - k] by simp
    also have ... = (∑ d | d dvd k. ∑ n | n ∈ {1..k} ∧ d dvd n. ?h n d)
```

using *sum.swap-restrict*[*of* {*1..k*} {*d. d dvd k*}
- λ*n d. d dvd n*] *assms* **by** *fastforce*
finally have
($\sum$ *n* = *1..k*. $\sum$ *d* | *d dvd gcd n k*. *?h n d*) =
($\sum$ *d* | *d dvd k*. $\sum$ *n* | *n* $\in$ {*1..k*} $\wedge$ *d dvd n*. *?h n d*) **by** *blast*
then show *?thesis* **by** *simp*
qed
also have ... = ($\sum$ *d* | *d dvd k*. *f*(*d*)**g*(*k div d*)*
($\sum$ *n* | *n* $\in$ {*1..k*} $\wedge$ *d dvd n*. *unity-root k* (− *m* * *int n*)))
by (*simp add*: *sum-distrib-left*)
also have ... = ($\sum$ *d* | *d dvd k*. *f*(*d*)**g*(*k div d*)*
($\sum$ *e* $\in$ {*1..k div d*}. *unity-root k* (− *m* * (*e***d*))))
using *assms*(*1*) *sum-div-reduce div-greater-zero-iff dvd-div-gt0* **by** *auto*
also have ... = ($\sum$ *d* | *d dvd k*. *f*(*d*)**g*(*k div d*)*
($\sum$ *e* $\in$ {*1..k div d*}. *unity-root* (*k div d*) (− *m* * *e*)))
proof −
{
fix *d e*
assume *d dvd k*
hence *2* * *pi* * *real-of-int* (− *int m* * *int* (*e* * *d*)) / *real k* =
2 * *pi* * *real-of-int* (− *int m* * *int e*) / *real* (*k div d*) **by** *auto*
hence *unity-root k* (− *m* * (*e* * *d*)) = *unity-root* (*k div d*) (− *m* * *e*)
unfolding *unity-root-def* **by** *simp*
}
then show *?thesis* **by** *simp*
qed
also have ... = *dirichlet-prod* (λ*d*. *f*(*d*)**g*(*k div d*))
(λ*d*. ($\sum$ *e* $\in$ {*1..d*}. *unity-root d* (− *m* * *e*))) *k*
unfolding *dirichlet-prod-def* **by** *blast*
also have ... = *dirichlet-prod* (λ*d*. ($\sum$ *e* $\in$ {*1..d*}. *unity-root d* (− *m* * *e*)))
(λ*d*. *f*(*d*)**g*(*k div d*)) *k*
using *dirichlet-prod-commutes*[*of*
(λ*d*. *f*(*d*)**g*(*k div d*))
(λ*d*. ($\sum$ *e* $\in$ {*1..d*}. *unity-root d* (− *m* * *e*)))] **by** *argo*
also have ... = ($\sum$ *d* | *d dvd k*.
($\sum$ *e* $\in$ {*1..*(*d*::*nat*)}. *unity-root d* (− *m* * *e*))*(*f*(*k div d*)**g*(*k div* (*k div*
d))))
unfolding *dirichlet-prod-def* **by** *blast*
also have ... = ($\sum$ *d* | *d dvd k*. ($\sum$ *e* $\in$ {*1..*(*d*::*nat*)}.
unity-root d (− *m* * *e*))*(*f*(*k div d*)**g*(*d*)))
proof −
{
fix *d* :: *nat*
assume *d dvd k*
then have *k div* (*k div d*) = *d*
by (*simp add*: *assms*(*1*) *div-div-eq-right*)
}
then show *?thesis* **by** *simp*
qed

also have ... $= (\sum (d::nat) \mid d\ dvd\ k \wedge d\ dvd\ m.\ d*(f(k\ div\ d)*g(d)))$
proof −
{
fix *d*
assume *d dvd k*
with *assms* **have** $d > 0$ **by** (*intro Nat.gr0I*) *auto*
have *periodic-arithmetic* ($\lambda x.$ *unity-root d* ($-$ *m* $*$ *int x*)) *d*
using *unity-periodic-arithmetic-mult* **by** *blast*
then have ($\sum e \in \{1..d\}$. *unity-root d* ($-$ *m* $*$ *e*)) =
($\sum e \in \{0..d-1\}$. *unity-root d* ($-$ *m* $*$ *e*))
using *periodic-arithmetic-sum-periodic-arithmetic-shift*[*of* $\lambda e.$ *unity-root d* ($-$ *m* $*$ *e*) *d 1*] *assms* ‹*d dvd k*›
by *fastforce*
also have ... = *unity-root-sum d* ($-m$)
unfolding *unity-root-sum-def* **using** ‹$d > 0$› **by** (*intro sum.cong*) *auto*
finally have
($\sum e \in \{1..d\}$. *unity-root d* ($-$ *m* $*$ *e*)) = *unity-root-sum d* ($-m$)
by *argo*
}
then have
($\sum d \mid d\ dvd\ k$. ($\sum e = 1..d$. *unity-root d* ($-$ *m* $*$ *int e*)) $*$ (*f* (*k div d*) $*$ *g d*)) =
($\sum d \mid d\ dvd\ k$. *unity-root-sum d* ($-m$) $*$ (*f* (*k div d*) $*$ *g d*)) **by** *simp*
also have ... = ($\sum d \mid d\ dvd\ k \wedge d\ dvd\ m$. *unity-root-sum d* ($-m$) $*$ (*f* (*k div d*) $*$ *g d*))
proof (*intro sum.mono-neutral-right*,*simp add*: ‹$k > 0$›,*blast*,*standard*)
fix *i*
assume *as*: $i \in \{d.\ d\ dvd\ k\} - \{d.\ d\ dvd\ k \wedge d\ dvd\ m\}$
then have $i \geq 1$ **using** ‹$k > 0$› **by** *auto*
have $k \geq 1$ **using** ‹$k > 0$› **by** *auto*
have $\neg$ *i dvd* ($-m$) **using** *as* **by** *auto*
thus *unity-root-sum i* ($-$ *int m*) $*$ (*f* (*k div i*) $*$ *g i*) = *0*
using ‹$i \geq 1$› **by** (*subst unity-root-sum*(*2*)) *auto*
qed
also have ... = ($\sum d \mid d\ dvd\ k \wedge d\ dvd\ m.\ d$ $*$ (*f* (*k div d*) $*$ *g d*))
proof −
{**fix** *d* :: *nat*
assume *1*: *d dvd m*
assume *2*: *d dvd k*
then have *unity-root-sum d* ($-m$) = *d*
using *unity-root-sum*[*of d* ($-m$)] *assms*(*1*) *1 2*
by *auto*}
then show *?thesis* **by** *auto*
qed
finally show *?thesis* **by** *argo*
qed
also have ... = ($\sum d \mid d\ dvd\ gcd\ m\ k$. *of-nat d* $*$ (*f* (*k div d*) $*$ *g d*))
by (*simp add*: *gcd.commute*)
also have ... = ($\sum d \mid d\ dvd\ gcd\ m\ k$. *g d* $*$ *f* (*k div d*) $*$ *d*)

by (*simp add*: *algebra-simps sum-distrib-left*)
also have *1 / k* ∗ . . . = *a k m* **using** *a-def* **by** *auto*
finally have *?g m* = *a k m* **by** *simp*}
note *a-eq-g* = *this*
{
fix *m*
from *fourier-expansion-periodic-arithmetic*(*2*)[*of k s f g k*] *gen-ramanujan-sum-periodic assms*(*1*)
have *s f g k m* = (∑ *n*<*k*. *?g n* ∗ *unity-root k* (*int m* ∗ *n*))
by *blast*
also have . . . = (∑ *n*<*k*. *a k n* ∗ *unity-root k* (*int m* ∗ *n*))
using *a-eq-g* **by** *simp*
also have . . . = (∑ *n*≤*k*−*1*. *a k n* ∗ *unity-root k* (*int m* ∗ *n*))
using ‹*k* > *0*› **by** (*intro sum.cong*) *auto*
finally have *s f g k m* =
(∑ *n*≤*k* − *1*. *a k n* ∗ *unity-root k* (*int n* ∗ *int m*))
by (*simp add*: *algebra-simps*)
}
then show *?thesis* **by** *blast*
qed

Theorem 8.6

theorem *ramanujan-sum-dirichlet-form*:
fixes *k n* :: *nat*
assumes *k* > *0*
shows *c k n* = (∑ *d* | *d dvd gcd n k*. *d* ∗ *moebius-mu* (*k div d*))
proof −
define *a* :: *nat* ⇒ *nat* ⇒ *complex*
where *a* = (λ*k m*.
1 / *of-nat k* ∗ (∑ *d* | *d dvd gcd m k*. *moebius-mu d* ∗ *of-nat* (*k div d*) ∗ *of-nat d*))

{**fix** *m*
have *a k m* = (*if gcd m k* = *1 then 1 else 0*)
proof −
have *a k m* = *1* / *of-nat k* ∗ (∑ *d* | *d dvd gcd m k*. *moebius-mu d* ∗ *of-nat* (*k div d*) ∗ *of-nat d*)
unfolding *a-def* **by** *blast*
also have *2*: . . . = *1* / *of-nat k* ∗ (∑ *d* | *d dvd gcd m k*. *moebius-mu d* ∗ *of-nat k*)
proof −
{**fix** *d* :: *nat*
assume *dvd*: *d dvd gcd m k*
have *moebius-mu d* ∗ *of-nat* (*k div d*) ∗ *of-nat d* = *moebius-mu d* ∗ *of-nat k*
proof −
have (*k div d*) ∗ *d* = *k* **using** *dvd* **by** *auto*
then show *moebius-mu d* ∗ *of-nat* (*k div d*) ∗ *of-nat d* = *moebius-mu d* ∗ *of-nat k*
by (*simp add*: *algebra-simps*,*subst of-nat-mult*[*symmetric*],*simp*)

```
    qed} note eq = this
    show ?thesis using sum.cong by (simp add: eq)
  qed

  also have 3: ... = (∑ d | d dvd gcd m k. moebius-mu d)
    by (simp add: sum-distrib-left assms)
  also have 4: ... =  (if gcd m k = 1 then 1 else 0)
    using sum-moebius-mu-divisors' by blast
  finally show a k m  = (if gcd m k = 1 then 1 else 0)
    using coprime-def by blast
qed} note a-expr = this

 let ?f = (λm. (if gcd m k = 1 then 1 else 0) ∗
              unity-root k (int m ∗ n))
 from gen-ramanujan-sum-fourier-expansion[of k id moebius-mu n] assms
 have s (λx. of-nat (id x)) moebius-mu k n =
 (∑ m≤k − 1.
     1 / of-nat k ∗
     (∑ d | d dvd gcd m k.
        moebius-mu d ∗ of-nat (k div d) ∗ of-nat d) ∗
     unity-root k (int m ∗ n)) by simp
 also have ... = (∑ m≤k − 1.
     a k m ∗
     unity-root k (int m ∗ n)) using a-def by blast
 also have ... = (∑ m≤k − 1.
     (if gcd m k = 1 then 1 else 0) ∗
     unity-root k (int m ∗ n)) using a-expr by auto
 also have ... = (∑ m ∈ {1..k}.
     (if gcd m k = 1 then 1 else 0) ∗
     unity-root k (int m ∗ n))
 proof −
   have periodic-arithmetic (λm. (if gcd m k = 1 then 1 else 0) ∗
              unity-root k (int m ∗ n)) k
   proof −
     have periodic-arithmetic (λm. if gcd m k = 1 then 1 else 0) k
       by (simp add: periodic-arithmetic-def)
     moreover have periodic-arithmetic (λm. unity-root k (int m ∗ n)) k
       using unity-periodic-arithmetic-mult[of k n]
       by (subst mult.commute,simp)
     ultimately show periodic-arithmetic ?f k
       using mult-periodic-arithmetic by simp
   qed
   then have sum ?f {0..k − 1} = sum ?f {1..k}
     using periodic-arithmetic-sum-periodic-arithmetic-shift[of ?f k 1] by force
   then show ?thesis by (simp add: atMost-atLeast0)
 qed
 also have ... = (∑ m | m ∈ {1..k} ∧ gcd m k = 1.
                (if gcd m k = 1 then 1 else 0) ∗
                unity-root k (int m ∗ int n))
```

by (*intro sum.mono-neutral-right*,*auto*)
also have ... $= (\sum m \mid m \in \{1..k\} \wedge gcd\ m\ k = 1.$
$unity\text{-}root\ k\ (int\ m * int\ n))$ **by** *simp*
also have ... $= (\sum m \mid m \in \{1..k\} \wedge coprime\ m\ k.$
$unity\text{-}root\ k\ (int\ m * int\ n))$
using *coprime-iff-gcd-eq-1* **by** *presburger*
also have ... $= c\ k\ n$ **unfolding** *ramanujan-sum-def* **by** *simp*
finally show *?thesis* **unfolding** *gen-ramanujan-sum-def* **by** *auto*
qed

corollary *ramanujan-sum-conv-gen-ramanujan-sum*:
$k > 0 \Longrightarrow c\ k\ n = s\ id\ moebius\text{-}mu\ k\ n$
using *ramanujan-sum-dirichlet-form* **unfolding** *gen-ramanujan-sum-def* **by** *simp*

Theorem 8.7

theorem *gen-ramanujan-sum-distrib*:
fixes $f\ g :: nat \Rightarrow complex$
assumes $a > 0\ b > 0\ m > 0\ k > 0$
assumes *coprime a k coprime b m coprime k m*
assumes *multiplicative-function f* **and**
multiplicative-function g
shows $s\ f\ g\ (m*k)\ (a*b) = s\ f\ g\ m\ a * s\ f\ g\ k\ b$
proof −
from *assms*(*1*−*6*) **have** *eq*: $gcd\ (m*k)\ (a*b) = gcd\ a\ m * gcd\ k\ b$
by (*simp add*: *linear-gcd gcd.commute mult.commute*)
have $s\ f\ g\ (m*k)\ (a*b) =$
$(\sum d \mid d\ dvd\ gcd\ (m*k)\ (a*b).\ f(d) * g((m*k)\ div\ d))$
unfolding *gen-ramanujan-sum-def* **by** (*rule sum.cong*, *simp add*: *gcd.commute*,*blast*)

also have ... $=$
$(\sum d \mid d\ dvd\ gcd\ a\ m * gcd\ k\ b.\ f(d) * g((m*k)\ div\ d))$
using *eq* **by** *simp*
also have ... $=$
$(\sum (d1,d2) \mid d1\ dvd\ gcd\ a\ m \wedge d2\ dvd\ gcd\ k\ b.$
$f(d1*d2) * g((m*k)\ div\ (d1*d2)))$
proof −
have *b*: $bij\text{-}betw\ (\lambda(d1,\ d2).\ d1 * d2)$
$\{(d1,\ d2).\ d1\ dvd\ gcd\ a\ m \wedge d2\ dvd\ gcd\ k\ b\}$
$\{d.\ d\ dvd\ gcd\ a\ m * gcd\ k\ b\}$
using *assms*(*5*) *reindex-product-bij* **by** *blast*
have $(\sum (d1,\ d2) \mid d1\ dvd\ gcd\ a\ m \wedge d2\ dvd\ gcd\ k\ b.$
$f\ (d1 * d2) * g\ (m * k\ div\ (d1 * d2))) =$
$(\sum x\in\{(d1,\ d2).\ d1\ dvd\ gcd\ a\ m \wedge d2\ dvd\ gcd\ k\ b\}.$
$f\ (case\ x\ of\ (d1,\ d2) \Rightarrow d1 * d2)*$
$g\ (m * k\ div\ (case\ x\ of\ (d1,\ d2) \Rightarrow d1 * d2)))$
by (*rule sum.cong*,*auto*)
also have ... $= (\sum d \mid d\ dvd\ gcd\ a\ m * gcd\ k\ b.\ f\ d * g\ (m * k\ div\ d))$
using *b* **by** (*rule sum.reindex-bij-betw*[*of* $\lambda(d1,d2).\ d1*d2$])
finally show *?thesis* **by** *argo*

```
    qed
  also have ... = (∑ d1 | d1 dvd gcd a m. ∑ d2 | d2 dvd gcd k b.
                  f (d1*d2) * g ((m*k) div (d1*d2)))
      by (simp add: sum.cartesian-product) (rule sum.cong,auto)
    also have ... = (∑ d1 | d1 dvd gcd a m. ∑ d2 | d2 dvd gcd k b.
                  f d1 * f d2 * g ((m*k) div (d1*d2)))
      using assms(5) assms(8) multiplicative-function.mult-coprime
      by (intro sum.cong refl) fastforce+
    also have ... = (∑ d1 | d1 dvd gcd a m. ∑ d2 | d2 dvd gcd k b.
                  f d1 * f d2* g (m div d1) * g (k div d2))
    proof (intro sum.cong refl, clarify, goal-cases)
      case (1 d1 d2)
      hence g (m * k div (d1 * d2)) = g (m div d1) * g (k div d2)
        using assms(7,9) multipl-div
        by (meson coprime-commute dvd-gcdD1 dvd-gcdD2)
      thus ?case by simp
    qed
    also have ... = (∑ i∈{d1. d1 dvd gcd a m}. ∑ j∈{d2. d2 dvd gcd k b}.
                  f i * g (m div i) * (f j * g (k div j)))
      by (rule sum.cong,blast,rule sum.cong,blast,simp)
    also have ... = (∑ d1 | d1 dvd gcd a m. f d1 * g (m div d1)) *
                  (∑ d2 | d2 dvd gcd k b. f d2 * g (k div d2))
      by (simp add: sum-product)
    also have ... = s f g m a * s f g k b
      unfolding gen-ramanujan-sum-def by (simp add: gcd.commute)
    finally show ?thesis by blast
qed

corollary gen-ramanujan-sum-distrib-right:
 fixes f g :: nat ⇒ complex
 assumes a > 0 and b > 0 and m > 0
 assumes coprime b m
 assumes multiplicative-function f and
        multiplicative-function g
 shows s f g m (a * b) = s f g m a
proof -
  have s f g m (a*b) = s f g m a * s f g 1 b
    using assms gen-ramanujan-sum-distrib[of a b m 1 f g] by simp
  also have ... = s f g m a * f 1 * g 1
    using gen-ramanujan-sum-1-n by auto
  also have ... = s f g m a
    using assms(5−6)
    by (simp add: multiplicative-function-def)
  finally show s f g m (a*b) = s f g m a by blast
qed

corollary gen-ramanujan-sum-distrib-left:
 fixes f g :: nat ⇒ complex
 assumes a > 0 and k > 0 and m > 0
```

assumes *coprime a k* **and** *coprime k m*
assumes *multiplicative-function f* **and**
multiplicative-function g
shows *s f g (m∗k) a = s f g m a ∗ g k*
proof −
have *s f g (m∗k) a = s f g m a ∗ s f g k 1*
using *assms gen-ramanujan-sum-distrib[of a 1 m k f g]* **by** *simp*
also have ... *= s f g m a ∗ f(1) ∗ g(k)*
using *gen-ramanujan-sum-k-1* **by** *auto*
also have ... *= s f g m a ∗ g k*
using *assms(6)*
by *(simp add: multiplicative-function-def)*
finally show *?thesis* **by** *blast*
qed

corollary *ramanujan-sum-distrib*:
assumes *a > 0* **and** *k > 0* **and** *m > 0* **and** *b > 0*
assumes *coprime a k coprime b m coprime m k*
shows *c (m∗k) (a∗b) = c m a ∗ c k b*
proof −
have *c (m∗k) (a∗b) = s id moebius-mu (m∗k) (a∗b)*
using *ramanujan-sum-conv-gen-ramanujan-sum assms(2,3)* **by** *simp*

also have ... *= (s id moebius-mu m a) ∗ (s id moebius-mu k b)*
using *gen-ramanujan-sum-distrib[of a b m k id moebius-mu]*
assms mult-id mult-moebius mult-of-nat
coprime-commute[of m k] **by** *auto*
also have ... *= c m a ∗ c k b* **using** *ramanujan-sum-conv-gen-ramanujan-sum*
assms **by** *simp*
finally show *?thesis* **by** *simp*
qed

corollary *ramanujan-sum-distrib-right*:
assumes *a > 0* **and** *k > 0* **and** *m > 0* **and** *b > 0*
assumes *coprime b m*
shows *c m (a∗b) = c m a*
using *assms ramanujan-sum-conv-gen-ramanujan-sum mult-id mult-moebius*
mult-of-nat gen-ramanujan-sum-distrib-right **by** *auto*

corollary *ramanujan-sum-distrib-left*:
assumes *a > 0 k > 0 m > 0*
assumes *coprime a k coprime m k*
shows *c (m∗k) a = c m a ∗ moebius-mu k*
using *assms*
by *(simp add: ramanujan-sum-conv-gen-ramanujan-sum, subst gen-ramanujan-sum-distrib-left)*
(auto simp: coprime-commute mult-of-nat mult-moebius)

lemma *dirichlet-prod-completely-multiplicative-left*:
fixes *f h :: nat ⇒ complex* **and** *k :: nat*

defines $g \equiv (\lambda k.\ moebius\text{-}mu\ k * h\ k)$
defines $F \equiv dirichlet\text{-}prod\ f\ g$
assumes $k > 0$
assumes *completely-multiplicative-function f*
multiplicative-function h
assumes $\bigwedge p.\ prime\ p \Longrightarrow f(p) \neq 0 \wedge f(p) \neq h(p)$
shows $F\ k = f\ k * (\prod p\in prime\text{-}factors\ k.\ 1 - h\ p\ /\ f\ p)$
proof −
have *1*: $multiplicative\text{-}function\ (\lambda p.\ h(p)\ div\ f(p))$
using *multiplicative-function-divide*
comp-to-mult assms(*4*,*5*) **by** *blast*
have $F\ k = dirichlet\text{-}prod\ g\ f\ k$
unfolding *F-def* **using** *dirichlet-prod-commutes*[*of f g*] **by** *auto*
also have $\ldots = (\sum d \mid d\ dvd\ k.\ moebius\text{-}mu\ d * h\ d * f(k\ div\ d))$
unfolding *g-def dirichlet-prod-def* **by** *blast*
also have $\ldots = (\sum d \mid d\ dvd\ k.\ moebius\text{-}mu\ d * h\ d * (f(k)\ div\ f(d)))$
using *multipl-div-mono*[*of f - k*] *assms*(*4*,*6*)
by (*intro sum.cong*,*auto*,*force*)
also have $\ldots = f\ k * (\sum d \mid d\ dvd\ k.\ moebius\text{-}mu\ d * (h\ d\ div\ f(d)))$
by (*simp add*: *sum-distrib-left algebra-simps*)
also have $\ldots = f\ k * (\prod p\in prime\text{-}factors\ k.\ 1 - (h\ p\ div\ f\ p))$
using *sum-divisors-moebius-mu-times-multiplicative*[*of* $\lambda p.\ h\ p\ div\ f\ p\ k$] *1*
assms(*3*) **by** *simp*
finally show *F-eq*: $F\ k = f\ k * (\prod p\in prime\text{-}factors\ k.\ 1 - (h\ p\ div\ f\ p))$
by *blast*
qed

Theorem 8.8

theorem *gen-ramanujan-sum-dirichlet-expr*:
fixes $f\ h :: nat \Rightarrow complex$ **and** $n\ k :: nat$
defines $g \equiv (\lambda k.\ moebius\text{-}mu\ k * h\ k)$
defines $F \equiv dirichlet\text{-}prod\ f\ g$
defines $N \equiv k\ div\ gcd\ n\ k$
assumes *completely-multiplicative-function f*
multiplicative-function h
assumes $\bigwedge p.\ prime\ p \Longrightarrow f(p) \neq 0 \wedge f(p) \neq h(p)$
assumes $k > 0$ $n > 0$
shows $s\ f\ g\ k\ n = (F(k)*g(N))\ div\ (F(N))$
proof −
define *a* **where** $a \equiv gcd\ n\ k$
have *2*: $k = a*N$ **unfolding** *a-def N-def* **by** *auto*
have *3*: $a > 0$ **using** *a-def assms*(*7*,*8*) **by** *simp*
have *Ngr0*: $N > 0$ **using** *assms*(*7*,*8*) *2 N-def* **by** *fastforce*
have *f-k-not-z*: $f\ k \neq 0$
using *completely-multiplicative-nonzero assms*(*4*,*6*,*7*) **by** *blast*
have *f-N-not-z*: $f\ N \neq 0$
using *completely-multiplicative-nonzero assms*(*4*,*6*) *Ngr0* **by** *blast*
have *bij*: $bij\text{-}betw\ (\lambda d.\ a\ div\ d)\ \{d.\ d\ dvd\ a\}\ \{d.\ d\ dvd\ a\}$
unfolding *bij-betw-def*

proof
  show *inj*: *inj-on* ($\lambda d.$ *a div d*) $\{d.$ *d dvd a*$\}$
    using *inj-on-def 3 dvd-div-eq-2* **by** *blast*
  show *surj*: ($\lambda d.$ *a div d*) ' $\{d.$ *d dvd a*$\}$ = $\{d.$ *d dvd a*$\}$
    unfolding *image-def*
  proof
    show $\{y. \exists x{\in}\{d.\ d\ dvd\ a\}.\ y = a\ div\ x\} \subseteq \{d.\ d\ dvd\ a\}$
      by *auto*
    show $\{d.\ d\ dvd\ a\} \subseteq \{y. \exists x{\in}\{d.\ d\ dvd\ a\}.\ y = a\ div\ x\}$
    proof
      fix *d*
      assume *a*: $d \in \{d.\ d\ dvd\ a\}$
      from *a* **have** *1*: $(a\ div\ d) \in \{d.\ d\ dvd\ a\}$ **by** *auto*
      from *a* **have** *2*: $d = a\ div\ (a\ div\ d)$ **using** *3* **by** *auto*
      from *1 2* **show** $d \in \{y. \exists x{\in}\{d.\ d\ dvd\ a\}.\ y = a\ div\ x\}$ **by** *blast*
    qed
  qed
qed

have $s\ f\ g\ k\ n = (\sum d \mid d\ dvd\ a.\ f(d){*}moebius\text{-}mu(k\ div\ d){*}h(k\ div\ d))$
  unfolding *gen-ramanujan-sum-def g-def a-def* **by** (*simp add*: *mult.assoc*)
also have $\ldots = (\sum d \mid d\ dvd\ a.\ f(d) * moebius\text{-}mu(a{*}N\ div\ d){*}h(a{*}N\ div\ d))$
  using *2* **by** *blast*
also have $\ldots = (\sum d \mid d\ dvd\ a.\ f(a\ div\ d) * moebius\text{-}mu(N{*}d){*}h(N{*}d))$
  (**is** *?a* = *?b*)
proof −
  define *f-aux* **where** $f\text{-}aux \equiv (\lambda d.\ f\ d * moebius\text{-}mu\ (a * N\ div\ d) * h\ (a * N\ div\ d))$
  have *1*: $?a = (\sum d \mid d\ dvd\ a.\ f\text{-}aux\ d)$ **using** *f-aux-def* **by** *blast*
  {fix *d* :: *nat*
  assume *d dvd a*
  then have $N * a\ div\ (a\ div\ d) = N * d$
    using *3* **by** *force***}**
  then have *2*: $?b = (\sum d \mid d\ dvd\ a.\ f\text{-}aux\ (a\ div\ d))$
    unfolding *f-aux-def* **by** (*simp add*: *algebra-simps*)
  show *?a* = *?b*
    using *bij 1 2*
    by (*simp add*: *sum.reindex-bij-betw*[*of* ((*div*) *a*) $\{d.$ *d dvd a*$\}$ $\{d.$ *d dvd a*$\}$])
qed
also have $\ldots = moebius\text{-}mu\ N * h\ N * f\ a * (\sum d \mid d\ dvd\ a \wedge coprime\ N\ d.\ moebius\text{-}mu\ d * (h\ d\ div\ f\ d))$
  (**is** *?a* = *?b*)
proof −
  have $?a = (\sum d \mid d\ dvd\ a \wedge coprime\ N\ d.\ f(a\ div\ d) * moebius\text{-}mu\ (N{*}d) * h\ (N{*}d))$
    by (*rule sum.mono-neutral-right*)(*auto simp add*: *moebius-prod-not-coprime 3*)
  also have $\ldots = (\sum d \mid d\ dvd\ a \wedge coprime\ N\ d.\ moebius\text{-}mu\ N * h\ N * f(a\ div\ d) * moebius\text{-}mu\ d * h\ d)$

proof (*rule sum.cong,simp*)
fix *d*
assume *a*: $d \in \{d.\ d\ dvd\ a \wedge coprime\ N\ d\}$
then have *1*: *moebius-mu* $(N*d)$ = *moebius-mu* N * *moebius-mu* d
using *mult-moebius* **unfolding** *multiplicative-function-def*
by (*simp add*: *moebius-mu.mult-coprime*)
from *a* **have** *2*: $h\ (N*d) = h\ N * h\ d$
using *assms*(*5*) **unfolding** *multiplicative-function-def*
by (*simp add*: *assms*(*5*) *multiplicative-function.mult-coprime*)
show $f\ (a\ div\ d)$ * *moebius-mu* $(N * d) * h\ (N * d)$ =
moebius-mu $N * h\ N * f\ (a\ div\ d)$ * *moebius-mu* $d * h\ d$
by (*simp add*: *divide-simps 1 2*)
qed
also have ... = ($\sum d \mid d\ dvd\ a \wedge coprime\ N\ d.$ *moebius-mu* $N * h\ N * (f\ a\ div\ f\ d)$ * *moebius-mu* $d * h\ d$)
by (*intro sum.cong refl*) (*use multipl-div-mono*[*of f - a*] *assms*(*4*,*6*−*8*) *3* **in** *force*)
also have ... = *moebius-mu* $N * h\ N * f\ a * (\sum d \mid d\ dvd\ a \wedge coprime\ N\ d.$ *moebius-mu* $d * (h\ d\ div\ f\ d))$
by (*simp add*: *sum-distrib-left algebra-simps*)
finally show *?thesis* **by** *blast*
qed
also have ... =
moebius-mu $N * h\ N * f\ a * (\prod p \in \{p.\ p \in$ *prime-factors* $a \wedge \neg\ (p\ dvd\ N)\}.\ 1 - (h\ p\ div\ f\ p))$
proof −
have *multiplicative-function* $(\lambda d.\ h\ d\ div\ f\ d)$
using *multiplicative-function-divide*
comp-to-mult
assms(*4*,*5*) **by** *blast*
then have $(\sum d \mid d\ dvd\ a \wedge coprime\ N\ d.$ *moebius-mu* $d * (h\ d\ div\ f\ d))$ =
$(\prod p \in \{p.\ p \in$ *prime-factors* $a \wedge \neg\ (p\ dvd\ N)\}.\ 1 - (h\ p\ div\ f\ p))$
using *sum-divisors-moebius-mu-times-multiplicative-revisited*[
of $(\lambda d.\ h\ d\ div\ f\ d)\ a\ N$]
assms(*8*) *Ngr0 3* **by** *blast*
then show *?thesis* **by** *argo*
qed
also have ... = $f(a)$ * *moebius-mu*$(N) * h(N)$ *
$((\prod p \in \{p.\ p \in$ *prime-factors* $(a*N)\}.\ 1 - (h\ p\ div\ f\ p))\ div$
$(\prod p \in \{p.\ p \in$ *prime-factors* $N\}.\ 1 - (h\ p\ div\ f\ p)))$
proof −
have $\{p.\ p \in$*prime-factors* $a \wedge \neg\ p\ dvd\ N\}$ =
$(\{p.\ p \in$*prime-factors* $(a*N)\} - \{p.\ p \in$*prime-factors* $N\})$
using *p-div-set*[*of a N*] **by** *blast*
then have *eq2*: $(\prod p \in \{p.\ p \in$*prime-factors* $a \wedge \neg\ p\ dvd\ N\}.\ 1 - h\ p\ /\ f\ p)$ =
prod $(\lambda p.\ 1 - h\ p\ /\ f\ p)\ (\{p.\ p \in$*prime-factors* $(a*N)\} - \{p.\ p \in$*prime-factors* $N\})$
by *auto*
also have *eq*: ... = *prod* $(\lambda p.\ 1 - h\ p\ /\ f\ p)\ \{p.\ p \in$*prime-factors* $(a*N)\}$ *div*

```
                 prod (λp. 1 − h p / f p) {p. p ∈prime-factors N}
  proof (intro prod-div-sub,simp,simp,simp add: 3 Ngr0 dvd-prime-factors,simp,standard)
    fix b
    assume b ∈# prime-factorization N
    then have p-b: prime b using in-prime-factors-iff by blast
    then show f b = 0 ∨ h b ≠ f b using assms(6)[OF p-b] by auto
   qed
   also have ... = (∏ p∈{p. p ∈ prime-factors (a∗N)}. 1 − (h p div f p)) div
    (∏ p∈{p. p ∈ prime-factors N}. 1 − (h p div f p)) by blast
   finally have (∏ p∈{p. p ∈prime-factors a ∧ ¬ p dvd N}. 1 − h p / f p) =
       (∏ p∈{p. p ∈ prime-factors (a∗N)}. 1 − (h p div f p)) div
    (∏ p∈{p. p ∈ prime-factors N}. 1 − (h p div f p))
    using eq eq2 by auto
   then show ?thesis by simp
 qed
 also have ... = f(a) ∗ moebius-mu(N) ∗ h(N) ∗ (F(k) div f(k)) ∗ (f(N) div
F(N))
  (is ?a = ?b)
 proof −
   have F(N) = (f N) ∗(∏ p∈ prime-factors N. 1 − (h p div f p))
    unfolding F-def g-def
     by (intro dirichlet-prod-completely-multiplicative-left) (auto simp add: Ngr0
assms(4−6))
   then have eq-1: (∏ p∈ prime-factors N. 1 − (h p div f p)) =
           F N div f N using 2 f-N-not-z by simp
   have F(k) = (f k) ∗ (∏ p∈ prime-factors k. 1 − (h p div f p))
    unfolding F-def g-def
   by (intro dirichlet-prod-completely-multiplicative-left) (auto simp add: assms(4−7))
   then have eq-2: (∏ p∈ prime-factors k. 1 − (h p div f p)) =
           F k div f k using 2 f-k-not-z by simp

   have ?a = f a ∗ moebius-mu N ∗ h N ∗
        ((∏ p∈ prime-factors k. 1 − (h p div f p)) div
        (∏ p∈ prime-factors N. 1 − (h p div f p)))
    using 2 by (simp add: algebra-simps)
   also have  ... = f a ∗ moebius-mu N ∗ h N ∗ ((F k div f k) div (F N div f N))
    by (simp add: eq-1 eq-2)
   finally show ?thesis by simp
 qed
 also have ... = moebius-mu N ∗ h N ∗ ((F k ∗ f a ∗ f N) div (F N ∗ f k))
   by (simp add: algebra-simps)
 also have ... = moebius-mu N ∗ h N ∗ ((F k ∗ f(a∗N)) div (F N ∗ f k))
 proof −
   have f a ∗ f N = f (a∗N)
   proof (cases a = 1 ∨ N = 1)
    case True
    then show ?thesis
     using assms(4) completely-multiplicative-function-def[of f]
     by auto
```

next
  case *False*
  then show *?thesis*
    using *2 assms(4) completely-multiplicative-function-def[of f]*
      Ngr0 3 **by** *auto*
qed
then show *?thesis* **by** *simp*
qed
also have ... $= moebius\text{-}mu\ N * h\ N * ((F\ k * f(k))\ div\ (F\ N * f\ k))$
  using *2* **by** *blast*
also have ... $= g(N) * (F\ k\ div\ F\ N)$
  using *f-k-not-z g-def* **by** *simp*
also have ... $= (F(k)*g(N))\ div\ (F(N))$ **by** *auto*
finally show *?thesis* **by** *simp*
qed

lemma *totient-conv-moebius-mu-of-nat*:
  of-nat (totient n) = dirichlet-prod moebius-mu of-nat n
proof (*cases n = 0*)
  case *False*
  show *?thesis*
    by (*rule moebius-inversion*)
      (*insert False, simp-all add: of-nat-sum [symmetric] totient-divisor-sum del: of-nat-sum*)
qed *simp-all*

corollary *ramanujan-sum-k-n-dirichlet-expr*:
 fixes *k n :: nat*
 assumes *k > 0 n > 0*
 shows *c k n = of-nat (totient k) **
      moebius-mu (k div gcd n k) div
      of-nat (totient (k div gcd n k))
proof −
  define *f :: nat ⇒ complex*
    where *f ≡ of-nat*
  define *F :: nat ⇒ complex*
    where *F ≡ (λd. dirichlet-prod f moebius-mu d)*
  define *g :: nat ⇒ complex*
    where *g ≡ (λl. moebius-mu l)*
  define *N* **where** *N ≡ k div gcd n k*
  define *h :: nat ⇒ complex*
    where *h ≡ (λx. (if x = 0 then 0 else 1))*

  have *F-is-totient-k*: *F k = totient k*
   by (*simp add: F-def f-def dirichlet-prod-commutes totient-conv-moebius-mu-of-nat[of k]*)
  have *F-is-totient-N*: *F N = totient N*
   by (*simp add: F-def f-def dirichlet-prod-commutes totient-conv-moebius-mu-of-nat[of*

N])

have *c k n* = *s id moebius-mu k n*
using *ramanujan-sum-conv-gen-ramanujan-sum assms* **by** *blast*
also have ... = *s f g k n*
unfolding *f-def g-def* **by** *auto*
also have *g* = (*λk. moebius-mu k* ∗ *h k*)
by (*simp add*: *fun-eq-iff h-def g-def*)
also have *multiplicative-function h*
unfolding *h-def* **by** *standard auto*
hence *s f* (*λk. moebius-mu k* ∗ *h k*) *k n* =
dirichlet-prod of-nat (*λk. moebius-mu k* ∗ *h k*) *k* ∗
(*moebius-mu* (*k div gcd n k*) ∗ *h* (*k div gcd n k*)) /
dirichlet-prod of-nat (*λk. moebius-mu k* ∗ *h k*) (*k div gcd n k*)
unfolding *f-def* **using** *assms mult-of-nat-c*
by (*intro gen-ramanujan-sum-dirichlet-expr*) (*auto simp*: *h-def*)
also have ... = *of-nat* (*totient k*) ∗ *moebius-mu* (*k div gcd n k*) / *of-nat* (*totient* (*k div gcd n k*))
using *F-is-totient-k F-is-totient-N* **by** (*auto simp*: *h-def F-def N-def f-def*)
finally show *?thesis* **.**
qed

no-notation *ramanujan-sum* (‹*c*›)
no-notation *gen-ramanujan-sum* (‹*s*›)

end

theory *Gauss-Sums*
imports
HOL−Algebra.Coset
HOL−Real-Asymp.Real-Asymp
Ramanujan-Sums
begin

7 Gauss sums

bundle *vec-lambda-syntax*
begin
notation *vec-lambda* (**binder** ‹*χ*› *10*)
end

unbundle *no vec-lambda-syntax*

7.1 Definition and basic properties

context *dcharacter*
begin

```
lemma dir-periodic-arithmetic: periodic-arithmetic χ n
  unfolding periodic-arithmetic-def by (simp add: periodic)

definition gauss-sum k = (∑ m = 1..n . χ(m) * unity-root n (m*k))

lemma gauss-sum-periodic:
  periodic-arithmetic (λn. gauss-sum n) n
proof −
  have periodic-arithmetic χ n using dir-periodic-arithmetic by simp
  let ?h = λm k. χ(m) * unity-root n (m*k)
  {fix m :: nat
  have periodic-arithmetic (λk. unity-root n (m*k)) n
    using unity-periodic-arithmetic-mult[of n m] by simp
  have periodic-arithmetic (?h m) n
    using scalar-mult-periodic-arithmetic[OF ‹periodic-arithmetic (λk. unity-root n
(m*k)) n›]
    by blast}
  then have per-all: ∀ m ∈ {1..n}. periodic-arithmetic (?h m) n by blast
  have periodic-arithmetic (λk. (∑ m = 1..n . χ(m) * unity-root n (m*k))) n
    using fin-sum-periodic-arithmetic-set[OF per-all] by blast
  then show ?thesis
    unfolding gauss-sum-def by blast
qed

lemma ramanujan-sum-conv-gauss-sum:
  assumes χ = principal-dchar n
  shows ramanujan-sum n k = gauss-sum k
proof −
  {fix m
  from assms
    have 1: coprime m n ⟹ χ(m) = 1 and
         2: ¬ coprime m n ⟹ χ(m) = 0
      unfolding principal-dchar-def by auto}
  note eq = this

  have gauss-sum k = (∑ m = 1..n . χ(m) * unity-root n (m*k))
    unfolding gauss-sum-def by simp
  also have ... = (∑ m | m ∈ {1..n} ∧ coprime m n . χ(m) * unity-root n (m*k))
    by (rule sum.mono-neutral-right,simp,blast,simp add: eq)
  also have ... = (∑ m | m ∈ {1..n} ∧ coprime m n . unity-root n (m*k))
    by (simp add: eq)
  also have ... = ramanujan-sum n k unfolding ramanujan-sum-def by blast
  finally show ?thesis ..
qed

lemma cnj-mult-self:
  assumes coprime k n
  shows cnj (χ k) * χ k = 1
proof −
```

```
  have cnj (χ k) ∗ χ k = norm (χ k)^2
    by (simp add: mult.commute complex-mult-cnj cmod-def)
  also have ... = 1
    using norm[of k] assms by simp
  finally show ?thesis .
qed
```

Theorem 8.9

```
theorem gauss-sum-reduction:
  assumes coprime k n
  shows gauss-sum k = cnj (χ k) ∗ gauss-sum 1
proof −
  from n have n-pos: n > 0 by simp
  have gauss-sum k = (∑ r = 1..n . χ(r) ∗ unity-root n (r∗k))
    unfolding gauss-sum-def by simp
  also have ... = (∑ r = 1..n . cnj (χ(k)) ∗ χ k ∗ χ r ∗ unity-root n (r∗k))
    using assms by (intro sum.cong) (auto simp: cnj-mult-self)
  also have ... = (∑ r = 1..n . cnj (χ(k)) ∗ χ (k∗r) ∗ unity-root n (r∗k))
    by (intro sum.cong) auto
  also have ... = cnj (χ(k)) ∗ (∑ r = 1..n . χ (k∗r) ∗ unity-root n (r∗k))
    by (simp add: sum-distrib-left algebra-simps)
  also have ...= cnj (χ(k)) ∗ (∑ r = 1..n . χ r ∗ unity-root n r)
  proof −
    have 1: periodic-arithmetic (λr. χ r ∗ unity-root n r) n
     using dir-periodic-arithmetic unity-periodic-arithmetic mult-periodic-arithmetic
by blast
    have (∑ r = 1..n . χ (k∗r) ∗ unity-root n (r∗k)) =
          (∑ r = 1..n . χ (r)∗ unity-root n r)
      using periodic-arithmetic-remove-homothecy[OF assms(1) 1 n-pos]
      by (simp add: algebra-simps n)
    then show ?thesis by argo
  qed
  also have ... = cnj (χ(k)) ∗ gauss-sum 1
    using gauss-sum-def by simp
  finally show ?thesis .
qed
```

The following variant takes an integer argument instead.

```
definition gauss-sum-int k = (∑ m=1..n. χ m ∗ unity-root n (int m∗k))

sublocale gauss-sum-int: periodic-fun-simple gauss-sum-int int n
proof
  fix k
  show gauss-sum-int (k + int n) = gauss-sum-int k
    by (simp add: gauss-sum-int-def ring-distribs unity-root-add)
qed

lemma gauss-sum-int-cong:
  assumes [a = b] (mod int n)
```

shows *gauss-sum-int a* = *gauss-sum-int b*
proof −
from *assms* **obtain** *k* **where** *k*: *b* = *a* + *int n* ∗ *k*
by (*subst* (*asm*) *cong-iff-lin*) *auto*
thus *?thesis*
using *gauss-sum-int.plus-of-int*[*of a k*] **by** (*auto simp*: *algebra-simps*)
qed

lemma *gauss-sum-conv-gauss-sum-int*:
gauss-sum k = *gauss-sum-int* (*int k*)
unfolding *gauss-sum-def gauss-sum-int-def* **by** *auto*

lemma *gauss-sum-int-conv-gauss-sum*:
gauss-sum-int k = *gauss-sum* (*nat* (*k mod n*))
proof −
have *gauss-sum* (*nat* (*k mod n*)) = *gauss-sum-int* (*int* (*nat* (*k mod n*)))
by (*simp add*: *gauss-sum-conv-gauss-sum-int*)
also have . . . = *gauss-sum-int k*
using *n*
by (*intro gauss-sum-int-cong*) (*auto simp*: *cong-def*)
finally show *?thesis* **..**
qed

lemma *gauss-int-periodic*: *periodic-arithmetic gauss-sum-int n*
unfolding *periodic-arithmetic-def gauss-sum-int-conv-gauss-sum* **by** *simp*

proposition *dcharacter-fourier-expansion*:
χ m = (∑*k*=*1*..*n*. *1* / *n* ∗ *gauss-sum-int* (−*k*) ∗ *unity-root n* (*m*∗*k*))
proof −
define *g* **where** *g* = (λ*x*. *1* / *of-nat n* ∗
(∑*m*<*n*. *χ m* ∗ *unity-root n* (− *int x* ∗ *int m*)))
have *per*: *periodic-arithmetic χ n* **using** *dir-periodic-arithmetic* **by** *simp*
have *χ m* = (∑*k*<*n*. *g k* ∗ *unity-root n* (*m* ∗ *int k*))
using *fourier-expansion-periodic-arithmetic*(*2*)[*OF* - *per*, *of m*] *n* **by** (*auto simp*: *g-def*)
also have . . . = (∑*k* = *1*..*n*. *g k* ∗ *unity-root n* (*m* ∗ *int k*))
proof −
have *g-per*: *periodic-arithmetic g n*
using *fourier-expansion-periodic-arithmetic*(*1*)[*OF* - *per*] *n* **by** (*simp add*: *g-def*)
have *fact-per*: *periodic-arithmetic* (λ*k*. *g k* ∗ *unity-root n* (*int m* ∗ *int k*)) *n*
using *mult-periodic-arithmetic*[*OF g-per*] *unity-periodic-arithmetic-mult* **by** *auto*
show *?thesis*
proof −
have (∑*k*<*n*. *g k* ∗ *unity-root n* (*int m* ∗ *int k*)) =
(∑*l* = *0*..*n* − *Suc 0*. *g l* ∗ *unity-root n* (*int m* ∗ *int l*))
using *n* **by** (*intro sum.cong*) *auto*
also have . . . = (∑*l* = *Suc 0*..*n*. *g l* ∗ *unity-root n* (*int m* ∗ *int l*))

using *periodic-arithmetic-sum-periodic-arithmetic-shift*[*OF fact-per*, *of 1*] *n* **by** *auto*
finally show *?thesis* **by** *simp*
qed
qed
also have ... $= (\sum k = 1..n.\ (1\ /\ \textit{of-nat}\ n) * \textit{gauss-sum-int}\ (-k) * \textit{unity-root}\ n\ (m*k))$
proof −
{fix *k* :: *nat*
have *shift*: $(\sum m<n.\ \chi\ m * \textit{unity-root}\ n\ (-\ \textit{int}\ k * \textit{int}\ m)) =$
$(\sum m = 1..n.\ \chi\ m * \textit{unity-root}\ n\ (-\ \textit{int}\ k * \textit{int}\ m))$
proof −
have *per-unit*: *periodic-arithmetic* $(\lambda m.\ \textit{unity-root}\ n\ (-\ \textit{int}\ k * \textit{int}\ m))$ *n*
using *unity-periodic-arithmetic-mult* **by** *blast*
then have *prod-per*: *periodic-arithmetic* $(\lambda m.\ \chi\ m * \textit{unity-root}\ n\ (-\ \textit{int}\ k * \textit{int}\ m))$ *n*
using *per mult-periodic-arithmetic* **by** *blast*
show *?thesis*
proof −
have $(\sum m<n.\ \chi\ m * \textit{unity-root}\ n\ (-\ \textit{int}\ k * \textit{int}\ m)) =$
$(\sum l = 0..n - \textit{Suc}\ 0.\ \chi\ l * \textit{unity-root}\ n\ (-\ \textit{int}\ k * \textit{int}\ l))$
using *n* **by** (*intro sum.cong*) *auto*
also have ... $= (\sum m = 1..n.\ \chi\ m * \textit{unity-root}\ n\ (-\ \textit{int}\ k * \textit{int}\ m))$
using *periodic-arithmetic-sum-periodic-arithmetic-shift*[*OF prod-per*, *of 1*] *n* **by** *auto*
finally show *?thesis* **by** *simp*
qed
qed
have $g\ k = 1\ /\ \textit{of-nat}\ n *$
$(\sum m<n.\ \chi\ m * \textit{unity-root}\ n\ (-\ \textit{int}\ k * \textit{int}\ m))$
using *g-def* **by** *auto*
also have ... $= 1\ /\ \textit{of-nat}\ n *$
$(\sum m = 1..n.\ \chi\ m * \textit{unity-root}\ n\ (-\ \textit{int}\ k * \textit{int}\ m))$
using *shift* **by** *simp*
also have ... $= 1\ /\ \textit{of-nat}\ n * \textit{gauss-sum-int}\ (-k)$
unfolding *gauss-sum-int-def*
by (*simp add*: *algebra-simps*)
finally have $g\ k = 1\ /\ \textit{of-nat}\ n * \textit{gauss-sum-int}\ (-k)$ **by** *simp***}**
note *g-expr* = *this*
show *?thesis*
by (*rule sum.cong*, *simp*, *simp add*: *g-expr*)
qed
finally show *?thesis* **by** *auto*
qed

7.2 Separability

definition *separable* $k \longleftrightarrow \textit{gauss-sum}\ k = \textit{cnj}\ (\chi\ k) * \textit{gauss-sum}\ 1$

corollary *gauss-coprime-separable*:
  assumes *coprime k n*
  shows *separable k*
  using *gauss-sum-reduction*[*OF assms*] **unfolding** *separable-def* **by** *simp*

Theorem 8.10

theorem *global-separability-condition*:
  ($\forall$ *n*>*0*. *separable n*) $\longleftrightarrow$ ($\forall$ *k*>*0*. $\neg$*coprime k n* $\longrightarrow$ *gauss-sum k* = *0*)
proof −
  {**fix** *k*
  assume $\neg$ *coprime k n*
  then have χ(*k*) = *0* **by** (*simp add*: *eq-zero*)
  then have *cnj* (χ *k*) = *0* **by** *blast*
  then have *separable k* $\longleftrightarrow$ *gauss-sum k* = *0*
    unfolding *separable-def* **by** *auto*}
  note *not-case* = *this*

  show *?thesis*
    using *gauss-coprime-separable not-case separable-def* **by** *blast*
qed

lemma *of-real-moebius-mu* [*simp*]: *of-real* (*moebius-mu k*) = *moebius-mu k*
  by (*simp add*: *moebius-mu-def*)

corollary *principal-not-totally-separable*:
  assumes χ = *principal-dchar n*
  shows $\neg$($\forall$ *k* > *0*. *separable k*)
proof −
  have *n-pos*: *n* > *0* **using** *n* **by** *simp*
  have *tot-0*: *totient n* $\neq$ *0* **by** (*simp add*: *n-pos*)
  have *moebius-mu* (*n div gcd n n*) $\neq$ *0* **by** (*simp add*: ‹*n* > *0*›)
  then have *moeb-0*: $\exists$ *k*. *moebius-mu* (*n div gcd k n*) $\neq$ *0* **by** *blast*

  have *lem*: *gauss-sum k* = *totient n* ∗ *moebius-mu* (*n div gcd k n*) / *totient* (*n div gcd k n*)
    if *k* > *0* **for** *k*
  proof −
    have *gauss-sum k* = *ramanujan-sum n k*
      using *ramanujan-sum-conv-gauss-sum*[*OF assms*(*1*)] **..**
    also have ... = *totient n* ∗ *moebius-mu* (*n div gcd k n*) / (*totient* (*n div gcd k n*))
      by (*simp add*: *ramanujan-sum-k-n-dirichlet-expr*[*OF n-pos that*])
    finally show *?thesis* **.**
  qed
  have *2*: $\neg$ *coprime n n* **using** *n* **by** *auto*
  have *3*: *gauss-sum n* $\neq$ *0*
    using *lem*[*OF n-pos*] *tot-0 moebius-mu-1* **by** *simp*
  from *n-pos 2 3* **have**
    $\exists$ *k*>*0*. $\neg$*coprime k n* $\wedge$ *gauss-sum k* $\neq$ *0* **by** *blast*

```
  then obtain k where k > 0 ∧ ¬ coprime k n ∧ gauss-sum k ≠ 0 by blast
  note right-not-zero = this

  have cnj (χ k) * gauss-sum 1 = 0 if ¬coprime k n for k
    using that assms by (simp add: principal-dchar-def)
  then show ?thesis
     unfolding separable-def using right-not-zero by auto
qed
```

Theorem 8.11

```
theorem gauss-sum-1-mod-square-eq-k:
  assumes (∀ k. k > 0 ⟶ separable k)
  shows norm (gauss-sum 1) ^ 2 = real n
proof −
  have (norm (gauss-sum 1))^2 = gauss-sum 1 * cnj (gauss-sum 1)
    using complex-norm-square by blast
  also have ... = gauss-sum 1 * (∑ m = 1..n. cnj (χ(m)) * unity-root n (−m))
  proof −
    have cnj (gauss-sum 1) = (∑ m = 1..n. cnj (χ(m)) * unity-root n (−m))
      unfolding gauss-sum-def by (simp add: unity-root-uminus)
    then show ?thesis by argo
  qed
  also have ... = (∑ m = 1..n. gauss-sum 1 * cnj (χ(m)) * unity-root n (−m))
    by (subst sum-distrib-left)(simp add: algebra-simps)
  also have ... = (∑ m = 1..n. gauss-sum m * unity-root n (−m))
  proof (rule sum.cong,simp)
    fix x
    assume as: x ∈ {1..n}
    show gauss-sum 1 * cnj (χ x) * unity-root n (−x) =
          gauss-sum x * unity-root n (−x)
      using assms(1) unfolding separable-def
      by (rule allE[of - x]) (use as in auto)
  qed
  also have ... = (∑ m = 1..n. (∑ r = 1..n. χ r * unity-root n (r*m) * unity-root
n (−m)))
    unfolding gauss-sum-def
    by (rule sum.cong,simp,rule sum-distrib-right)
  also have ... = (∑ m = 1..n. (∑ r = 1..n. χ r * unity-root n (m*(r−1)) ))
   by (intro sum.cong refl) (auto simp: unity-root-diff of-nat-diff unity-root-uminus
field-simps)
  also have ... = (∑ r=1..n. (∑ m=1..n.  χ(r) *unity-root n (m*(r−1))))
    by (rule sum.swap)
  also have ... = (∑ r=1..n. χ(r) *(∑ m=1..n. unity-root n (m*(r−1))))
    by (rule sum.cong, simp, simp add: sum-distrib-left)
  also have ... = (∑ r=1..n. χ(r) * unity-root-sum n (r−1))
  proof (intro sum.cong refl)
    fix x
    assume x ∈ {1..n}
    then have 1: periodic-arithmetic (λm. unity-root n (int (m * (x − 1)))) n
```

using *unity-periodic-arithmetic-mult*[*of n x−1*]
by (*simp add*: *mult.commute*)
have ($\sum m = 1..n.$ *unity-root n* (*int* ($m * (x - 1)$))) =
($\sum m = 0..n-1.$ *unity-root n* (*int* ($m * (x - 1)$)))
using *periodic-arithmetic-sum-periodic-arithmetic-shift*[*OF 1 -, of 1*] *n* **by** *simp*
also have ... = *unity-root-sum n* ($x-1$)
using *n* **unfolding** *unity-root-sum-def* **by** (*intro sum.cong*) (*auto simp*: *mult-ac*)
finally have ($\sum m = 1..n.$ *unity-root n* (*int* ($m * (x - 1)$))) =
unity-root-sum n (*int* ($x - 1$)) **.**
then show $\chi\ x *$ ($\sum m = 1..n.$ *unity-root n* (*int* ($m * (x - 1)$))) =
$\chi\ x *$ *unity-root-sum n* (*int* ($x - 1$)) **by** *argo*
qed
also have ... = ($\sum r \in \{1\}.\ \chi\ r *$ *unity-root-sum n* (*int* ($r - 1$)))
using *n unity-root-sum-nonzero-iff int-ops*(*6*)
by (*intro sum.mono-neutral-right*) *auto*
also have ... = $\chi\ 1 * n$ **using** *n* **by** *simp*
also have ... = *n* **by** *simp*
finally show *?thesis*
using *of-real-eq-iff* **by** *fastforce*
qed

Theorem 8.12

theorem *gauss-sum-nonzero-noncoprime-necessary-condition*:
assumes *gauss-sum k* $\neq$ *0* $\neg$*coprime k n k* > *0*
defines *d* $\equiv$ *n div gcd k n*
assumes *coprime a n* [*a* = *1*] (*mod d*)
shows *d dvd n d* < *n* χ *a* = *1*
proof −
show *d dvd n*
unfolding *d-def* **using** *n* **by** (*subst div-dvd-iff-mult*) *auto*
from *assms*(*2*) **have** *gcd k n* $\neq$ *1* **by** *blast*
then have *gcd k n* > *1* **using** *assms*(*3*,*4*) **by** (*simp add*: *nat-neq-iff*)
with *n* **show** *d* < *n* **by** (*simp add*: *d-def*)

have *periodic-arithmetic* ($\lambda r.\ \chi$ (*r*)$*$ *unity-root n* ($k*r$)) *n*
using *mult-periodic-arithmetic*[*OF dir-periodic-arithmetic unity-periodic-arithmetic-mult*] **by** *auto*
then have *1*: *periodic-arithmetic* ($\lambda r.\ \chi$ (*r*)$*$ *unity-root n* ($r*k$)) *n*
by (*simp add*: *algebra-simps*)

have *gauss-sum k* = ($\sum m = 1..n$. $\chi(m) *$ *unity-root n* ($m*k$))
unfolding *gauss-sum-def* **by** *blast*
also have ... = ($\sum m = 1..n$. $\chi(m*a) *$ *unity-root n* ($m*a*k$))
using *periodic-arithmetic-remove-homothecy*[*OF assms*(*5*) *1*] *n* **by** *auto*
also have ... = ($\sum m = 1..n$. $\chi(m*a) *$ *unity-root n* ($m*k$))
proof (*intro sum.cong refl*)
fix *m*

from *assms*(*6*) **obtain** *b* **where** $a = 1 + b*d$
using ‹$d < n$› *assms*(*5*) *cong-to-1′-nat* **by** *auto*
then have $m*a*k = m*k+m*b*(n\ div\ gcd\ k\ n)*k$
by (*simp add*: *algebra-simps d-def*)
also have $\ldots = m*k+m*b*n*(k\ div\ gcd\ k\ n)$
by (*simp add*: *div-mult-swap dvd-div-mult*)
also obtain *p* **where** $\ldots = m*k+m*b*n*p$ **by** *blast*
finally have $m*a*k = m*k+m*b*p*n$ **by** *simp*
then have *1*: $m*a*k\ mod\ n = m*k\ mod\ n$
using *mod-mult-self1* **by** *simp*
then have *unity-root n* $(m * a * k)$ = *unity-root n* $(m * k)$
proof −
have *unity-root n* $(m * a * k)$ = *unity-root n* $((m * a * k)\ mod\ n)$
using *unity-root-mod*[*of n*] *zmod-int* **by** *simp*
also have $\ldots$ = *unity-root n* $(m * k)$
using *unity-root-mod*[*of n*] *zmod-int 1* **by** *presburger*
finally show *?thesis* **by** *blast*
qed
then show χ $(m * a)$ ∗ *unity-root n* (*int* $(m * a * k)$) =
χ $(m * a)$ ∗ *unity-root n* (*int* $(m * k)$) **by** *auto*
qed
also have $\ldots = (\sum m = 1..n\ .\ \chi(a) * (\chi(m) *$ *unity-root n* $(m*k)))$
by (*rule sum.cong*,*simp*,*subst mult*,*simp*)
also have $\ldots = \chi(a) * (\sum m = 1..n\ .\ \chi(m) *$ *unity-root n* $(m*k))$
by (*simp add*: *sum-distrib-left*[*symmetric*])
also have $\ldots = \chi(a) *$ *gauss-sum k*
unfolding *gauss-sum-def* **by** *blast*
finally have *gauss-sum k* $= \chi(a) *$ *gauss-sum k* **by** *blast*
then show $\chi\ a = 1$
using *assms*(*1*) **by** *simp*
qed

7.3 Induced moduli and primitive characters

definition *induced-modulus d* ⟷ *d dvd n* ∧ (∀ *a*. *coprime a n* ∧ [$a = 1$] (*mod d*) ⟶ $\chi\ a = 1$)

lemma *induced-modulus-dvd*: *induced-modulus d* ⟹ *d dvd n*
unfolding *induced-modulus-def* **by** *blast*

lemma *induced-modulusI* [*intro?*]:
d dvd n ⟹ (⋀*a*. *coprime a n* ⟹ [$a = 1$] (*mod d*) ⟹ $\chi\ a = 1$) ⟹ *induced-modulus d*
unfolding *induced-modulus-def* **by** *auto*

lemma *induced-modulusD*: *induced-modulus d* ⟹ *coprime a n* ⟹ [$a = 1$] (*mod d*) ⟹ $\chi\ a = 1$
unfolding *induced-modulus-def* **by** *blast*

lemma *zero-not-ind-mod*: ¬*induced-modulus 0*
  unfolding *induced-modulus-def* **using** *n* **by** *simp*

lemma *div-gcd-dvd1*: (*a* :: *'a* :: *semiring-gcd*) *div gcd a b dvd a*
  by (*metis dvd-def dvd-div-mult-self gcd-dvd1*)

lemma *div-gcd-dvd2*: (*b* :: *'a* :: *semiring-gcd*) *div gcd a b dvd b*
  by (*metis div-gcd-dvd1 gcd.commute*)

lemma *g-non-zero-ind-mod*:
  assumes *gauss-sum k* ≠ *0* ¬*coprime k n k* > *0*
  shows *induced-modulus* (*n div gcd k n*)
proof
  show *n div gcd k n dvd n*
    by (*metis dvd-div-mult-self dvd-triv-left gcd.commute gcd-dvd1*)
  fix *a* :: *nat*
  assume *coprime a n* [*a* = *1*] (*mod n div gcd k n*)
  thus χ *a* = *1*
    using *assms n gauss-sum-nonzero-noncoprime-necessary-condition*(*3*) **by** *auto*
qed

lemma *induced-modulus-modulus*: *induced-modulus n*
  unfolding *induced-modulus-def*
  by (*metis dvd-refl local.cong mult.one*)

Theorem 8.13

theorem *one-induced-iff-principal*:
 induced-modulus 1 ⟷ χ = *principal-dchar n*
proof
  assume *induced-modulus 1*
  then have (∀ *a*. *coprime a n* ⟶ χ *a* = *1*)
    unfolding *induced-modulus-def* **by** *simp*
  then show χ = *principal-dchar n*
    unfolding *principal-dchar-def* **using** *eq-zero* **by** *auto*
next
  assume *as*: χ = *principal-dchar n*
  {fix *a*
  assume *coprime a n*
  then have χ *a* = *1*
    using *principal-dchar-def as* **by** *simp***}**
  then show *induced-modulus 1*
    unfolding *induced-modulus-def* **by** *auto*
qed

end

locale *primitive-dchar* = *dcharacter* +
  assumes *no-induced-modulus*: ¬(∃ *d*<*n*. *induced-modulus d*)

locale *nonprimitive-dchar* = *dcharacter* +
  assumes *induced-modulus*: $\exists$ *d*<*n*. *induced-modulus d*

lemma (**in** *nonprimitive-dchar*) *nonprimitive*: ¬*primitive-dchar n* χ
proof
  assume *primitive-dchar n* χ
  then interpret *A*: *primitive-dchar n residue-mult-group n* χ
    by *auto*
  from *A.no-induced-modulus induced-modulus* **show** *False* **by** *contradiction*
qed

lemma (**in** *dcharacter*) *primitive-dchar-iff*:
  primitive-dchar n χ $\longleftrightarrow$ ¬($\exists$ *d*<*n*. *induced-modulus d*)
  unfolding *primitive-dchar-def primitive-dchar-axioms-def*
  using *dcharacter-axioms* **by** *metis*

lemma (**in** *residues-nat*) *principal-not-primitive*:
  ¬*primitive-dchar n* (*principal-dchar n*)
  unfolding *principal.primitive-dchar-iff*
  using *principal.one-induced-iff-principal n* **by** *auto*

lemma (**in** *dcharacter*) *not-primitive-imp-nonprimitive*:
  assumes ¬*primitive-dchar n* χ
  shows *nonprimitive-dchar n* χ
  using *assms dcharacter-axioms*
  unfolding *nonprimitive-dchar-def primitive-dchar-def*
      primitive-dchar-axioms-def nonprimitive-dchar-axioms-def **by** *auto*

Theorem 8.14

theorem (**in** *dcharacter*) *prime-nonprincipal-is-primitive*:
  assumes *prime n*
  assumes $\chi \neq$ *principal-dchar n*
  shows *primitive-dchar n* χ
proof −
  {fix *m*
  assume *induced-modulus m*
  then have *m* = *n*
    using *assms prime-nat-iff induced-modulus-def*
      one-induced-iff-principal **by** *blast***}**
  then show *?thesis* **using** *primitive-dchar-iff* **by** *blast*
qed

Theorem 8.15

corollary (**in** *primitive-dchar*) *primitive-encoding*:
  $\forall$ *k*>*0*. ¬*coprime k n* $\longrightarrow$ *gauss-sum k* = *0*
  $\forall$ *k*>*0*. *separable k*
  norm (*gauss-sum 1*) ^ *2* = *n*
proof *safe*

show *1*: *gauss-sum k* = *0* **if** *k* > *0* **and** ¬*coprime k n* **for** *k*
proof (*rule ccontr*)
assume *gauss-sum k* ≠ *0*
hence *induced-modulus* (*n div gcd k n*)
using *that* **by** (*intro g-non-zero-ind-mod*) *auto*
moreover have *n div gcd k n* < *n*
using *n that*
by (*meson coprime-iff-gcd-eq-1 div-eq-dividend-iff le-less-trans linorder-neqE-nat nat-dvd-not-less principal.div-gcd-dvd2 zero-le-one*)
ultimately show *False* **using** *no-induced-modulus* **by** *blast*
qed

have (∀ *n*>*0*. *separable n*)
unfolding *global-separability-condition* **by** (*auto intro*!: *1*)
thus *separable n* **if** *n* > *0* **for** *n*
using *that* **by** *blast*
thus *norm* (*gauss-sum 1*) ^ *2* = *n*
using *gauss-sum-1-mod-square-eq-k* **by** *blast*
qed

Theorem 8.16

lemma (**in** *dcharacter*) *induced-modulus-altdef1*:
induced-modulus d ⟷
d dvd n ∧ (∀ *a b*. *coprime a n* ∧ *coprime b n* ∧ [*a* = *b*] (*mod d*) ⟶ χ *a* = χ *b*)
proof
assume *1*: *induced-modulus d*
with *n* **have** *d*: *d dvd n d* > *0*
by (*auto simp*: *induced-modulus-def intro*: *Nat.gr0I*)
show *d dvd n* ∧ (∀ *a b*. *coprime a n* ∧ *coprime b n* ∧ [*a* = *b*] (*mod d*) ⟶ χ(*a*) = χ(*b*))
proof *safe*
fix *a b*
assume *2*: *coprime a n coprime b n* [*a* = *b*] (*mod d*)
show χ(*a*) = χ(*b*)
proof −
from *2*(*1*) **obtain** *a'* **where** *eq*: [*a*∗*a'* = *1*] (*mod n*)
using *cong-solve* **by** *blast*
from *this d* **have** [*a*∗*a'* = *1*] (*mod d*)
using *cong-dvd-modulus-nat* **by** *blast*
from *this 1* **have** χ(*a*∗*a'*) = *1*
unfolding *induced-modulus-def*
by (*meson 2*(*2*) *eq cong-imp-coprime cong-sym coprime-divisors gcd-nat.refl one-dvd*)
then have *3*: χ(*a*)∗χ(*a'*) = *1*
by *simp*

from *2*(*3*) **have** [*a*∗*a'* = *b*∗*a'*] (*mod d*)
by (*simp add*: *cong-scalar-right*)
moreover have *4*: [*b*∗*a'* = *1*] (*mod d*)

using ‹$[a * a' = 1]$ $(mod\ d)$› *calculation cong-sym cong-trans* **by** *blast*
have $\chi(b{*}a') = 1$
proof −
have *coprime* $(b{*}a')$ n
using $2(2)$ *cong-imp-coprime*[*OF cong-sym*[*OF eq*]] **by** *simp*
then show *?thesis* **using** *4 induced-modulus-def 1* **by** *blast*
qed
then have 4: $\chi(b){*}\chi(a') = 1$
by *simp*
from *3 4* **show** $\chi(a) = \chi(b)$
using *mult-cancel-left*
by (*cases* $\chi(a') = 0$) (*fastforce simp add*: *field-simps*)+
qed
qed *fact*+
next
assume ∗: $d\ dvd\ n \wedge (\forall\ a\ b.\ coprime\ a\ n \wedge coprime\ b\ n \wedge [a = b]\ (mod\ d) \longrightarrow \chi\ a = \chi\ b)$
then have $\forall\ a\ .\ coprime\ a\ n \wedge coprime\ 1\ n \wedge [a = 1]\ (mod\ d) \longrightarrow \chi\ a = \chi\ 1$
by *blast*
then have $\forall\ a\ .\ coprime\ a\ n \wedge [a = 1]\ (mod\ d) \longrightarrow \chi\ a = 1$
using *coprime-1-left* **by** *simp*
then show *induced-modulus d*
unfolding *induced-modulus-def* **using** ∗ **by** *blast*
qed

Exercise 8.4

lemma *induced-modulus-altdef2-lemma*:
fixes *n a d q* :: *nat*
defines $q \equiv (\prod\ p \mid prime\ p \wedge p\ dvd\ n \wedge \neg\ (p\ dvd\ a).\ p)$
defines $m \equiv a + q * d$
assumes $n > 0$ *coprime a d*
shows $[m = a]$ $(mod\ d)$ **and** *coprime m n*
proof (*simp add*: *assms*(2) *cong-add-lcancel-0-nat cong-mult-self-right*)
have *fin*: *finite* $\{p.\ prime\ p \wedge p\ dvd\ n \wedge \neg\ (p\ dvd\ a)\}$ **by** (*simp add*: *assms*)
{ fix *p*
assume 4: *prime p p dvd m p dvd n*
have $p = 1$
proof (*cases p dvd a*)
case *True*
from *this assms* 4(2) **have** $p\ dvd\ q{*}d$
by (*simp add*: *dvd-add-right-iff*)
then have *a1*: $p\ dvd\ q \vee p\ dvd\ d$
using 4(1) *prime-dvd-mult-iff* **by** *blast*

have *a2*: $\neg\ (p\ dvd\ q)$
proof (*rule ccontr*,*simp*)
assume *p dvd q*
then have $p\ dvd\ (\prod\ p \mid prime\ p \wedge p\ dvd\ n \wedge \neg\ (p\ dvd\ a).\ p)$
unfolding *assms* **by** *simp*

```
      then have ∃x∈{p. prime p ∧ p dvd n ∧ ¬ p dvd a}. p dvd x
        using prime-dvd-prod-iff[OF fin 4(1)] by simp
      then obtain x where c: p dvd x ∧ prime x ∧ ¬ x dvd a by blast
      then have p = x using 4(1) by (simp add: primes-dvd-imp-eq)
      then show False using True c by auto
     qed
     have a3: ¬ (p dvd d)
       using True assms 4(1) coprime-def not-prime-unit by auto

     from a1 a2 a3 show ?thesis by simp
   next
     case False
     then have p dvd q
     proof −
      have in-s: p ∈ {p. prime p ∧ p dvd n ∧ ¬ p dvd a}
        using False 4(3) 4(1) by simp
      show p dvd q
        unfolding assms using dvd-prodI[OF fin in-s ] by fast
     qed
     then have p dvd q∗d by simp
     then have p dvd a using 4(2) assms
       by (simp add: dvd-add-left-iff)
     then show ?thesis using False by auto
   qed
 }
 note lem = this
 show coprime m n
 proof (subst coprime-iff-gcd-eq-1)
   {fix a
    assume a dvd m a dvd n a ≠ 1
    {fix p
     assume prime p p dvd a
     then have p dvd m p dvd n
      using ‹a dvd m› ‹a dvd n› by auto
     from lem have p = a
      using not-prime-1 ‹prime p› ‹p dvd m› ‹p dvd n› by blast}
     then have prime a
      using prime-prime-factor[of a] ‹a ≠ 1› by blast
     then have a = 1 using lem ‹a dvd m› ‹a dvd n› by blast
     then have False using ‹a = 1› ‹a ≠ 1› by blast
   }
   then show gcd m n = 1 by blast
 qed
qed
```

Theorem 8.17

The case $d = 1$ is exactly the case described in *dcharacter ?n ?χ* $\Longrightarrow$ *dcharacter.induced-modulus ?n ?χ 1* $=$ (*?χ* $=$ *principal-dchar ?n*).

theorem (**in** *dcharacter*) *induced-modulus-altdef2*:
  assumes *d dvd n d* $\neq$ *1*
  defines $\chi_1 \equiv$ *principal-dchar n*
  shows *induced-modulus d* $\longleftrightarrow$ ($\exists\,\Phi$. *dcharacter d* Φ $\wedge$ ($\forall\,k$. χ *k* = Φ *k* $*$ χ_1 *k*))
proof
  from *n* **have** *n-pos*: *n* > *0* **by** *simp*
  assume *as-im*: *induced-modulus d*
  define *f* **where**
    f ≡ (λ*k*. *k* +
      (*if k* = *1 then*
         0
       else (*prod id* {*p*. *prime p* ∧ *p dvd n* ∧ ¬ (*p dvd k*)})∗*d*)
)
  have [*simp*]: *f* (*Suc 0*) = *1* **unfolding** *f-def* **by** *simp*
  {
    fix *k*
    assume *as*: *coprime k d*
    hence [*f k* = *k*] (*mod d*) *coprime* (*f k*) *n*
      using *induced-modulus-altdef2-lemma*[*OF n-pos as*] **by** (*simp-all add*: *f-def*)
  }
  note *m-prop* = *this*

  define Φ **where**
   Φ ≡ (λ*n*. (*if* ¬ *coprime n d then 0 else* χ(*f n*)))

  have Φ-*1*: Φ *1* = *1*
    unfolding Φ-*def* **by** *simp*

  from *assms*(*1*,*2*) *n* **have** *d* > *0* **by** (*intro Nat.gr0I*) *auto*
  from *induced-modulus-altdef1 assms*(*1*) ‹*d* > *0*› *as-im*
    have *b*: (∀ *a b*. *coprime a n* ∧ *coprime b n* ∧
                  [*a* = *b*] (*mod d*) ⟶ χ *a* = χ *b*) **by** *blast*

  have Φ-*periodic*:   ∀ *a*. Φ (*a* + *d*) = Φ *a*
  proof
    fix *a*
    have *gcd* (*a*+*d*) *d* = *gcd a d* **by** *auto*
    then have *cop*: *coprime* (*a*+*d*) *d* = *coprime a d*
      using *coprime-iff-gcd-eq-1* **by** *presburger*
    show Φ (*a* + *d*) = Φ *a*
    proof (*cases coprime a d*)
      case *True*
      from *True cop* **have** *cop-ad*: *coprime* (*a*+*d*) *d* **by** *blast*
      have *p1*: [*f* (*a*+*d*) = *f a*] (*mod d*)
        using *m-prop*(*1*)[*of a*+*d*, *OF cop-ad*]
          m-prop(*1*)[*of a*,*OF True*] **by** (*simp add*: *unique-euclidean-semiring-class.cong-def*)
      have *p2*: *coprime* (*f* (*a*+*d*)) *n coprime* (*f a*) *n*
        using *m-prop*(*2*)[*of a*+*d*, *OF cop-ad*]
            m-prop(*2*)[*of a*, *OF True*] **by** *blast*+

```
      from b p1 p2 have eq: χ (f (a + d)) = χ (f a) by blast
      show ?thesis
        unfolding Φ-def
        by (subst cop,simp,safe, simp add: eq)
    next
      case False
      then show ?thesis unfolding Φ-def by (subst cop,simp)
    qed
  qed

  have Φ-mult: ∀ a b. a ∈ totatives d ⟶
          b ∈ totatives d ⟶ Φ (a * b) = Φ a * Φ b
  proof (safe)
    fix a b
    assume a ∈ totatives d b ∈ totatives d
    consider (ab) coprime a d ∧ coprime b d |
             (a) coprime a d ∧ ¬ coprime b d |
             (b) coprime b d ∧ ¬ coprime a d |
             (n) ¬ coprime a d ∧ ¬ coprime b d by blast
    then show Φ (a * b) = Φ a * Φ b
    proof cases
      case ab
      then have c-ab:
        coprime (a*b) d coprime a d coprime b d by simp+
      then have p1: [f (a * b) = a * b] (mod d) coprime (f (a * b)) n
        using m-prop[of a*b, OF c-ab(1)] by simp+
      moreover have p2: [f a = a] (mod d) coprime (f a) n
                  [f b = b] (mod d) coprime (f b) n
        using m-prop[of a,OF c-ab(2)]
            m-prop[of b,OF c-ab(3) ] by simp+
      have p1s: [f (a * b) = (f a) * (f b)] (mod d)
      proof −
        have [f (a * b) = a * b] (mod d)
          using p1(1) by blast
        moreover have [a * b = f(a) * f(b)] (mod d)
          using p2(1) p2(3) by (simp add: cong-mult cong-sym)
        ultimately show ?thesis using cong-trans by blast
      qed
      have p2s: coprime (f a*f b) n
        using p2(2) p2(4) by simp
      have χ (f (a * b)) = χ (f a * f b)
        using p1s p2s p1(2) b by blast
      then show ?thesis
        unfolding Φ-def by (simp add: c-ab)
    qed (simp-all add: Φ-def)
  qed
  have d-gr-1: d > 1 using assms(1,2)
    using ‹0 < d› by linarith
  show ∃Φ. dcharacter d Φ ∧ (∀n. χ n = Φ n * χ₁ n)
```

```
  proof (standard,rule conjI)
    show dcharacter d Φ
      unfolding dcharacter-def residues-nat-def dcharacter-axioms-def
      using d-gr-1 Φ-def f-def Φ-mult Φ-1 Φ-periodic by simp
    show ∀ n. χ n = Φ n * χ₁ n
    proof
      fix k
      show χ k = Φ k * χ₁ k
      proof (cases coprime k n)
        case True
        then have coprime k d using assms(1) by auto
        then have Φ(k) = χ(f k) using Φ-def by simp
        moreover have [f k = k] (mod d)
          using m-prop[OF ‹coprime k d›] by simp
        moreover have χ₁ k = 1
          using assms(3) principal-dchar-def ‹coprime k n› by auto
        ultimately show χ(k) = Φ(k) * χ₁(k)
        proof −
          assume Φ k = χ (f k) [f k = k] (mod d) χ₁ k = 1
          then have χ k = χ (f k)
            using ‹local.induced-modulus d› induced-modulus-altdef1 assms(1) ‹d >
0›
                True ‹coprime k d› m-prop(2) by auto
          also have ... = Φ k by (simp add: ‹Φ k = χ (f k)›)
          also have ... = Φ k * χ₁ k by (simp add: ‹χ₁ k = 1›)
          finally show ?thesis by simp
        qed
      next
        case False
        hence χ k = 0
          using eq-zero-iff by blast
        moreover have χ₁ k = 0
          using False assms(3) principal-dchar-def by simp
        ultimately show ?thesis by simp
      qed
    qed
  qed
next
  assume (∃ Φ. dcharacter d Φ ∧ (∀ k. χ k = Φ k * χ₁ k))
  then obtain Φ where 1: dcharacter d Φ (∀ k. χ k = Φ k * χ₁ k) by blast
  show induced-modulus d
    unfolding induced-modulus-def
  proof (rule conjI,fact,safe)
    fix k
    assume 2: coprime k n [k = 1] (mod d)
    then have χ₁ k = 1
      by (simp add: χ₁-def)
    moreover have Φ k = 1
      by (metis 1(1) 2(2) One-nat-def dcharacter.Suc-0 dcharacter.cong)
```

ultimately show χ *k* = *1* **using** *1*(*2*) **by** *simp*
qed
qed

7.4 The conductor of a character

context *dcharacter*
begin

definition *conductor* = *Min* {*d*. *induced-modulus d*}

lemma *conductor-fin*: *finite* {*d*. *induced-modulus d*}
proof −
let *?A* = {*d*. *induced-modulus d*}
have *?A* ⊆ {*d*. *d dvd n*}
unfolding *induced-modulus-def* **by** *blast*
moreover have *finite* {*d*. *d dvd n*} **using** *n* **by** *simp*
ultimately show *finite ?A* **using** *finite-subset* **by** *auto*
qed

lemma *conductor-induced*: *induced-modulus conductor*
proof −
have {*d*. *induced-modulus d*} ≠ {} **using** *induced-modulus-modulus* **by** *blast*
then show *induced-modulus conductor*
using *Min-in*[*OF conductor-fin*] *conductor-def* **by** *auto*
qed

lemma *conductor-le-iff*: *conductor* ≤ *a* ⟷ (∃ *d*≤*a*. *induced-modulus d*)
unfolding *conductor-def* **using** *conductor-fin induced-modulus-modulus* **by** (*subst Min-le-iff*) *auto*

lemma *conductor-ge-iff*: *conductor* ≥ *a* ⟷ (∀ *d*. *induced-modulus d* ⟶ *d* ≥ *a*)
unfolding *conductor-def* **using** *conductor-fin induced-modulus-modulus* **by** (*subst Min-ge-iff*) *auto*

lemma *conductor-leI*: *induced-modulus d* ⟹ *conductor* ≤ *d*
by (*subst conductor-le-iff*) *auto*

lemma *conductor-geI*: (⋀*d*. *induced-modulus d* ⟹ *d* ≥ *a*) ⟹ *conductor* ≥ *a*
by (*subst conductor-ge-iff*) *auto*

lemma *conductor-dvd*: *conductor dvd n*
using *conductor-induced* **unfolding** *induced-modulus-def* **by** *blast*

lemma *conductor-le-modulus*: *conductor* ≤ *n*
using *conductor-dvd* **by** (*rule dvd-imp-le*) (*use n* **in** *auto*)

lemma *conductor-gr-0*: *conductor* > *0*
unfolding *conductor-def* **using** *zero-not-ind-mod*

using *conductor-def conductor-induced neq0-conv* **by** *fastforce*

lemma *conductor-eq-1-iff-principal*: *conductor* = *1* $\longleftrightarrow$ χ = *principal-dchar n*
proof
assume *conductor* = *1*
then have *induced-modulus 1*
using *conductor-induced* **by** *auto*
then show χ = *principal-dchar n*
using *one-induced-iff-principal* **by** *blast*
next
assume χ = *principal-dchar n*
then have *im-1*: *induced-modulus 1* **using** *one-induced-iff-principal* **by** *auto*
show *conductor* = *1*
proof −
have *conductor* $\leq$ *1*
using *conductor-fin Min-le*[*OF conductor-fin*,*simplified*,*OF im-1*]
by (*simp add*: *conductor-def*[*symmetric*])
then show *?thesis* **using** *conductor-gr-0* **by** *auto*
qed
qed

lemma *conductor-principal* [*simp*]: χ = *principal-dchar n* $\Longrightarrow$ *conductor* = *1*
by (*subst conductor-eq-1-iff-principal*)

lemma *nonprimitive-imp-conductor-less*:
assumes $\neg$*primitive-dchar n* χ
shows *conductor* < *n*
proof −
obtain *d* **where** *d*: *induced-modulus d d* < *n*
using *primitive-dchar-iff assms* **by** *blast*
from *d*(*1*) **have** *conductor* $\leq$ *d*
by (*rule conductor-leI*)
also have . . . < *n* **by** *fact*
finally show *?thesis* **.**
qed

lemma (**in** *nonprimitive-dchar*) *conductor-less-modulus*: *conductor* < *n*
using *nonprimitive-imp-conductor-less nonprimitive* **by** *metis*

Theorem 8.18

theorem *primitive-principal-form*:
defines $\chi_1 \equiv$ *principal-dchar n*
assumes $\chi \neq$ *principal-dchar n*
shows $\exists\,\Phi$. *primitive-dchar conductor* $\Phi \wedge (\forall\, n.\ \chi(n) = \Phi(n) * \chi_1(n))$
proof −

from *n* **have** *n-pos*: *n* > *0* **by** *simp*
define *d* **where** *d* = *conductor*
have *induced*: *induced-modulus d*

```
    unfolding d-def using conductor-induced by blast
  then have d-not-1: d ≠ 1
    using one-induced-iff-principal assms by auto
  hence d-gt-1: d > 1 using conductor-gr-0 by (auto simp: d-def)

  from induced obtain Φ where Φ-def: dcharacter d Φ ∧ (∀ n. χ n = Φ n ∗ χ1
n)
    using d-not-1
      by (subst (asm) induced-modulus-altdef2) (auto simp: d-def conductor-dvd
χ1-def)
  have phi-dchars: Φ ∈ dcharacters d using Φ-def dcharacters-def by auto

  interpret Φ: dcharacter d residue-mult-group d Φ
    using Φ-def by auto

  have Φ-prim: primitive-dchar d Φ
  proof (rule ccontr)
    assume ¬ primitive-dchar d Φ
    then obtain q where
      1: q dvd d ∧ q < d ∧ Φ.induced-modulus q
      unfolding Φ.induced-modulus-def Φ.primitive-dchar-iff by blast
    then have 2: induced-modulus q
    proof −
      {fix k
      assume mod-1: [k = 1] (mod q)
      assume cop: coprime k n
      have χ(k) = Φ(k)∗χ1(k) using Φ-def by auto
      also have . . . = Φ(k)
        using cop by (simp add: assms principal-dchar-def)
      also have . . . = 1
          using 1 mod-1 Φ.induced-modulus-def
                ‹induced-modulus d› cop induced-modulus-def by auto
      finally have χ(k) = 1 by blast}

      then show ?thesis
        using induced-modulus-def 1 ‹induced-modulus d› by auto
    qed

    from 1 have q < d by simp
    moreover have d ≤ q unfolding d-def
      by (intro conductor-leI) fact
    ultimately show False by linarith
  qed

  from Φ-def Φ-prim d-def phi-dchars show ?thesis by blast
qed

definition primitive-extension :: nat ⇒ complex where
  primitive-extension =
```

(*SOME* Φ. *primitive-dchar conductor* Φ ∧ (∀ *k*. χ *k* = Φ *k* ∗ *principal-dchar n k*))

lemma
assumes *nonprincipal*: χ ≠ *principal-dchar n*
shows *primitive-primitive-extension*: *primitive-dchar conductor primitive-extension*
and *principal-decomposition*: χ *k* = *primitive-extension k* ∗ *principal-dchar n k*
proof −
note ∗ = *someI-ex*[*OF primitive-principal-form*[*OF nonprincipal*], *folded primitive-extension-def*]
from ∗ **show** *primitive-dchar conductor primitive-extension* **by** *blast*
from ∗ **show** χ *k* = *primitive-extension k* ∗ *principal-dchar n k* **by** *blast*
qed

end

7.5 The connection between primitivity and separability

lemma *residue-mult-group-coset*:
fixes *m n m1 m2* :: *nat* **and** *f* :: *nat* ⇒ *nat* **and** *G H*
defines *G* ≡ *residue-mult-group n*
defines *H* ≡ *residue-mult-group m*
defines *f* ≡ (λ*k*. *k mod m*)
assumes *b* ∈ (*rcosets*$_G$ *kernel G H f*)
assumes *m1* ∈ *b m2* ∈ *b*
assumes *n* > *1 m dvd n*
shows *m1 mod m* = *m2 mod m*
proof −
have *h-1*: $\mathbf{1}_H$ = *1*
using *assms*(*2*) **unfolding** *residue-mult-group-def totatives-def* **by** *simp*

from *assms*(*4*)
obtain *a* :: *nat* **where** *cos-expr*: *b* = (*kernel G H f*) #>$_G$ *a* ∧ *a* ∈ *carrier G*
using *RCOSETS-def*[*of G kernel G H f*] **by** *blast*
then have *cop*: *coprime a n*
using *assms*(*1*) **unfolding** *residue-mult-group-def totatives-def* **by** *auto*

obtain *a'* **where** [*a* ∗ *a'* = *1*] (*mod n*)
using *cong-solve-coprime-nat*[*OF cop*] **by** *auto*
then have *a-inv*: (*a*∗*a'*) *mod n* = *1*
using *unique-euclidean-semiring-class.cong-def*[*of a*∗*a' 1 n*] *assms*(*7*) **by** *simp*

have *m1* ∈ (⋃*h*∈*kernel G H f*. {*h* ⊗$_G$ *a*})
m2 ∈ (⋃*h*∈*kernel G H f*. {*h* ⊗$_G$ *a*})
using *r-coset-def*[*of G kernel G H f a*] *cos-expr assms*(*5*,*6*) **by** *blast+*
then have *m1* ∈ (⋃*h*∈*kernel G H f*. {(*h* ∗ *a*) *mod n*})
m2 ∈ (⋃*h*∈*kernel G H f*. {(*h* ∗ *a*) *mod n*})
using *assms*(*1*) **unfolding** *residue-mult-group-def*[*of n*] **by** *auto*

```
  then obtain m1' m2' where
    m-expr: m1 = (m1'* a) mod n ∧ m1' ∈ kernel G H f
            m2 = (m2'* a) mod n ∧ m2' ∈ kernel G H f
    by blast

  have eq-1: m1 mod m = a mod m
  proof -
    have m1 mod m = ((m1'* a) mod n) mod m using m-expr by blast
    also have ... = (m1' * a) mod m
      using euclidean-semiring-cancel-class.mod-mod-cancel assms(8) by blast
    also have ... = (m1' mod m) * (a mod m) mod m
      by (simp add: mod-mult-eq)
    also have ... = (a mod m) mod m
      using m-expr(1) h-1 unfolding kernel-def assms(3) by simp
    also have ... = a mod m by auto
    finally show ?thesis by simp
  qed

  have eq-2: m2 mod m = a mod m
  proof -
    have m2 mod m = ((m2'* a) mod n) mod m using m-expr by blast
    also have ... = (m2' * a) mod m
      using euclidean-semiring-cancel-class.mod-mod-cancel assms(8) by blast
    also have ... = (m2' mod m) * (a mod m) mod m
      by (simp add: mod-mult-eq)
    also have ... = (a mod m) mod m
      using m-expr(2) h-1 unfolding kernel-def assms(3) by simp
    also have ... = a mod m by auto
    finally show ?thesis by simp
  qed

  from eq-1 eq-2 show ?thesis by argo
qed

lemma residue-mult-group-kernel-partition:
  fixes m n :: nat and f :: nat ⇒ nat and G H
  defines G ≡ residue-mult-group n
  defines H ≡ residue-mult-group m
  defines f ≡ (λk. k mod m)
  assumes m > 1 n > 0 m dvd n
  shows partition (carrier G) (rcosets_G kernel G H f)
      and card (rcosets_G kernel G H f) = totient m
      and card (kernel G H f) = totient n div totient m
      and b ∈(rcosets_G kernel G H f) ⟹ b ≠ {}
      and b ∈(rcosets_G kernel G H f) ⟹ card (kernel G H f) = card b
      and bij-betw (λb. (the-elem (f ‘ b))) (rcosets_G kernel G H f) (carrier H)
proof -
  have 1 < m by fact
  also have m ≤ n using ‹n > 0› ‹m dvd n› by (intro dvd-imp-le) auto
```

```
  finally have n > 1 .
  note mn = ‹m > 1› ‹n > 1› ‹m dvd n› ‹m ≤ n›

  interpret n: residues-nat n G
    using mn by unfold-locales (auto simp: assms)
  interpret m: residues-nat m H
    using mn by unfold-locales (auto simp: assms)

  from mn have subset: f ` carrier G ⊆ carrier H
    by (auto simp: assms(1−3) residue-mult-group-def totatives-def
          dest: coprime-common-divisor-nat intro!: Nat.gr0I)
  moreover have super-set: carrier H ⊆ f ` carrier G
  proof safe
    fix k assume k ∈ carrier H
    hence k: k > 0 k ≤ m coprime k m
      by (auto simp: assms(2) residue-mult-group-def totatives-def)
    from mn ‹k ∈ carrier H› have k < m
      by (simp add: totatives-less assms(2) residue-mult-group-def)
    define P where P = {p ∈ prime-factors n. ¬(p dvd m)}
    define a where a = ∏ P
    have [simp]: a ≠ 0 by (auto simp: P-def a-def intro!: Nat.gr0I)
    have [simp]: prime-factors a = P
    proof −
      have prime-factors a = set-mset (sum prime-factorization P)
        unfolding a-def using mn
        by (subst prime-factorization-prod)
           (auto simp: P-def prime-factors-dvd prime-gt-0-nat)
      also have sum prime-factorization P = (∑ p∈P. {#p#})
        using mn by (intro sum.cong) (auto simp: P-def prime-factorization-prime
prime-factors-dvd)
      finally show ?thesis by (simp add: P-def)
    qed

    from mn have coprime m a
      by (subst coprime-iff-prime-factors-disjoint) (auto simp: P-def)
    hence ∃x. [x = k] (mod m) ∧ [x = 1] (mod a)
      by (intro binary-chinese-remainder-nat)
    then obtain x where x: [x = k] (mod m) [x = 1] (mod a)
      by auto
    from x(1) mn k have [simp]: x ≠ 0
      by (meson ‹k < m› cong-0-iff cong-sym-eq nat-dvd-not-less)

    from x(2) have coprime x a
      using cong-imp-coprime cong-sym by force
    hence coprime x (a ∗ m)
      using k cong-imp-coprime[OF cong-sym[OF x(1)]] by auto
    also have ?this ←→ coprime x n using mn
      by (intro coprime-cong-prime-factors)
         (auto simp: prime-factors-product P-def in-prime-factors-iff)
```

```
    finally have x mod n ∈ totatives n
      using mn by (auto simp: totatives-def intro!: Nat.gr0I)

    moreover have f (x mod n) = k
     using x(1) k mn ‹k < m› by (auto simp: assms(3) unique-euclidean-semiring-class.cong-def
mod-mod-cancel)
    ultimately show k ∈ f ‘ carrier G
      by (auto simp: assms(1) residue-mult-group-def)
  qed

  ultimately have image-eq: f ‘ carrier G = carrier H by blast

  have [simp]: f (k ⊗G l) = f k ⊗H f l if k ∈ carrier G l ∈ carrier G for k l
   using that mn by (auto simp: assms(1−3) residue-mult-group-def totatives-def
                              mod-mod-cancel mod-mult-eq)
  interpret f: group-hom G H f
    using subset by unfold-locales (auto simp: hom-def)

  show bij-betw (λb. (the-elem (f ‘ b))) (rcosetsG kernel G H f) (carrier H)
    unfolding bij-betw-def
  proof
    show inj-on (λb. (the-elem (f ‘ b))) (rcosetsG kernel G H f)
      using f.FactGroup-inj-on unfolding FactGroup-def by auto
    have eq: f ‘ carrier G = carrier H
      using subset super-set by blast
    show (λb. the-elem (f ‘ b)) ‘ (rcosetsG kernel G H f) = carrier H
      using f.FactGroup-onto[OF eq] unfolding FactGroup-def by simp
  qed

  show partition (carrier G) (rcosetsG kernel G H f)
  proof
    show ⋀a. a ∈ carrier G =⇒
        ∃!b. b ∈ rcosetsG kernel G H f ∧ a ∈ b
    proof −
      fix a
      assume a-in: a ∈ carrier G
      show ∃!b. b ∈ rcosetsG kernel G H f ∧ a ∈ b
      proof −

        have ∃ b. b ∈ rcosetsG kernel G H f ∧ a ∈ b
          using a-in n.rcosets-part-G[OF f.subgroup-kernel]
          by blast
        then show ?thesis
          using group.rcos-disjoint[OF n.is-group f.subgroup-kernel]
          by (auto simp: disjoint-def)
      qed
    qed
  next
    show ⋀b. b ∈ rcosetsG kernel G H f =⇒ b ⊆ carrier G
```

```
      using n.rcosets-part-G f.subgroup-kernel by auto
  qed


  have lagr: card (carrier G) = card (rcosets_G kernel G H f) * card (kernel G H
f)
     using group.lagrange-finite[OF n.is-group n.fin f.subgroup-kernel] Coset.order-def[of
G] by argo
  have k-size: card (kernel G H f) > 0
    using f.subgroup-kernel finite-subset n.subgroupE(1) n.subgroupE(2) by fast-
force
  have G-size: card (carrier G) = totient n
    using n.order Coset.order-def[of G] by simp
  have H-size: totient m = card (carrier H)
    using n.order Coset.order-def[of H] by simp
  also have ... = card (carrier (G Mod kernel G H f))
   using f.FactGroup-iso[OF image-eq] card-image f.FactGroup-inj-on f.FactGroup-onto
image-eq by fastforce
  also have ... = card (carrier G) div card (kernel G H f)
  proof -
    have card (carrier (G Mod kernel G H f)) =
          card (rcosets_G kernel G H f)
      unfolding FactGroup-def by simp
    also have ... = card (carrier G) div card (kernel G H f)
      by (simp add: lagr k-size)
    finally show ?thesis by blast
  qed
  also have ... = totient n div card (kernel G H f)
    using G-size by argo
  finally have eq: totient m = totient n div card (kernel G H f) by simp
  show card (kernel G H f) = totient n div totient m
  proof -
    have totient m ≠ 0
      using totient-0-iff[of m] assms(4) by blast
    have card (kernel G H f) dvd totient n
      using lagr ‹card (carrier G) = totient n› by auto
    have totient m * card (kernel G H f) = totient n
      unfolding eq using ‹card (kernel G H f) dvd totient n› by auto
    have totient n div totient m = totient m * card (kernel G H f) div totient m
      using ‹totient m * card (kernel G H f) = totient n› by auto
    also have ... = card (kernel G H f)
      using nonzero-mult-div-cancel-left[OF ‹totient m ≠ 0›] by blast
    finally show ?thesis by auto
  qed

  show card (rcosets_G kernel G H f) = totient m
  proof -
    have H-size: totient m = card (carrier H)
      using n.order Coset.order-def[of H] by simp
```

also have ... = *card* (*carrier* (*G Mod kernel G H f*))
using *f.FactGroup-iso*[*OF image-eq*] *card-image f.FactGroup-inj-on f.FactGroup-onto image-eq* **by** *fastforce*
also have *card* (*carrier* (*G Mod kernel G H f*)) =
card ($rcosets_G$ *kernel G H f*)
unfolding *FactGroup-def* **by** *simp*
finally show *card* ($rcosets_G$ *kernel G H f*) = *totient m*
by *argo*
qed

assume $b \in rcosets_G$ *kernel G H f*
then show $b \neq \{\}$
proof −
have *card b* = *card* (*kernel G H f*)
using ‹$b \in rcosets_G$ *kernel G H f*› *f.subgroup-kernel n.card-rcosets-equal n.subgroupE*(*1*) **by** *auto*
then have *card b* > *0*
by (*simp add*: *k-size*)
then show *?thesis* **by** *auto*
qed

assume *b-cos*: $b \in rcosets_G$ *kernel G H f*
show *card* (*kernel G H f*) = *card b*
using *group.card-rcosets-equal*[*OF n.is-group b-cos*]
f.subgroup-kernel subgroup.subset **by** *blast*
qed

lemma *primitive-iff-separable-lemma*:
assumes *prod*: ($\forall n.\ \chi\ n = \Phi\ n * \chi_1\ n$) ∧ *primitive-dchar d* Φ
assumes ‹*d* > *1*› ‹*0* < *k*› ‹*d dvd k*› ‹*k* > *1*›
shows ($\sum m \mid m \in \{1..k\} \wedge$ *coprime m k*. $\Phi(m) *$ *unity-root d m*) =
(*totient k div totient d*) ∗ ($\sum m \mid m \in \{1..d\} \wedge$ *coprime m d*. $\Phi(m) *$ *unity-root d m*)
proof −
from *assms* **interpret** Φ: *primitive-dchar d residue-mult-group d* Φ
by *auto*
define *G* **where** *G* = *residue-mult-group k*
define *H* **where** *H* = *residue-mult-group d*
define *f* **where** *f* = ($\lambda t.\ t$ *mod d*)

from *residue-mult-group-kernel-partition*(*2*)[*OF* ‹*d* > *1*› ‹*0* < *k*› ‹*d dvd k*›]
have *fin-cosets*: *finite* ($rcosets_G$ *kernel G H f*)
using ‹*1* < *d*› *card.infinite* **by** (*fastforce simp*: *G-def H-def f-def*)

have *fin-G*: *finite* (*carrier G*)
unfolding *G-def residue-mult-group-def* **by** *simp*

have *eq*: ($\sum m \mid m \in \{1..k\} \wedge$ *coprime m k*. $\Phi(m) *$ *unity-root d m*) =

($\sum m \mid m \in$ *carrier G* . $\Phi(m) *$ *unity-root d m*)
unfolding *residue-mult-group-def totatives-def G-def*
by (*rule sum.cong,auto*)
also have ... = *sum* (λm. $\Phi(m) *$ *unity-root d m*) (*carrier G*) **by** *simp*
also have *eq'*: ... = *sum* (*sum* (λm. Φ *m* $*$ *unity-root d* (*int m*))) (*rcosets*$_G$ *kernel G H f*)
by (*rule disjoint-sum* [*symmetric*])
(*use fin-G fin-cosets residue-mult-group-kernel-partition*(*1*)[*OF* ‹*d* > *1*› ‹*k* > *0*› ‹*d dvd k*›] **in**
‹*auto simp*: *G-def H-def f-def*›)
also have ... =
($\sum b \in$ (*rcosets*$_G$ *kernel G H f*) . ($\sum m \in b$. Φ *m* $*$ *unity-root d* (*int m*))) **by** *simp*
finally have *1*: ($\sum m \mid m \in \{1..k\} \wedge$ *coprime m k*. $\Phi(m) *$ *unity-root d m*) =
($\sum b \in$ (*rcosets*$_G$ *kernel G H f*) . ($\sum m \in b$. Φ *m* $*$ *unity-root d* (*int m*)))
using *eq eq'* **by** *auto*
have *eq'''*: ... =
($\sum b \in$ (*rcosets*$_G$ *kernel G H f*) . (*totient k div totient d*) $*$ (Φ (*the-elem* (*f* ‘ *b*)) $*$ *unity-root d* (*int* (*the-elem* (*f* ‘ *b*)))))
proof (*rule sum.cong,simp*)
fix *b*
assume *b-in*: $b \in$ (*rcosets*$_G$ *kernel G H f*)
note *b-not-empty* = *residue-mult-group-kernel-partition*(*4*)
[*OF* ‹*d* > *1*› ‹*0* < *k*› ‹*d dvd k*› *b-in*[*unfolded G-def H-def f-def*]]

{
fix *m1 m2*
assume *m-in*: $m1 \in b$ $m2 \in b$
have *m-mod*: *m1 mod d* = *m2 mod d*
using *residue-mult-group-coset*[*OF b-in*[*unfolded G-def H-def f-def*] *m-in* ‹*k* > *1*› ‹*d dvd k*›]
by *blast*
} **note** *m-mod* = *this*
{
fix *m1 m2*
assume *m-in*: $m1 \in b$ $m2 \in b$
have Φ *m1* $*$ *unity-root d* (*int m1*) = Φ *m2* $*$ *unity-root d* (*int m2*)
proof −
have Φ-*periodic*: *periodic-arithmetic* Φ *d* **using** Φ.*dir-periodic-arithmetic* **by** *blast*
have *1*: Φ *m1* = Φ *m2*
using *mod-periodic-arithmetic*[*OF* ‹*periodic-arithmetic* Φ *d*› *m-mod*[*OF m-in*]] **by** *simp*
have *2*: *unity-root d m1* = *unity-root d m2*
using *m-mod*[*OF m-in*] **by** (*intro unity-root-cong*) (*auto simp*: *unique-euclidean-semiring-class.cong-def simp flip*: *zmod-int*)
from *1 2* **show** *?thesis* **by** *simp*

qed
} **note** *all-eq-in-coset* = *this*

from *all-eq-in-coset b-not-empty*
obtain *l* **where** *l-prop*: $l \in b \wedge (\forall y \in b.\ \Phi\ y * \textit{unity-root}\ d\ (\textit{int}\ y) = \Phi\ l * \textit{unity-root}\ d\ (\textit{int}\ l))$ **by** *blast*

have $(\sum m \in b.\ \Phi\ m * \textit{unity-root}\ d\ (\textit{int}\ m)) = ((\textit{totient}\ k\ \textit{div}\ \textit{totient}\ d) * (\Phi\ l * \textit{unity-root}\ d\ (\textit{int}\ l)))$
proof −
have $(\sum m \in b.\ \Phi\ m * \textit{unity-root}\ d\ (\textit{int}\ m)) = (\sum m \in b.\ \Phi\ l * \textit{unity-root}\ d\ (\textit{int}\ l))$
by (*rule sum.cong,simp*) (*use all-eq-in-coset l-prop* **in** *blast*)
also have ... = *card* $b * \Phi\ l * \textit{unity-root}\ d\ (\textit{int}\ l)$
by *simp*
also have ... = $(\textit{totient}\ k\ \textit{div}\ \textit{totient}\ d) * \Phi\ l * \textit{unity-root}\ d\ (\textit{int}\ l)$
using *residue-mult-group-kernel-partition*(*3*)[*OF* ‹*d* > *1*› ‹*0* < *k*› ‹*d dvd k*›]

residue-mult-group-kernel-partition(*5*)
[*OF* ‹*d* > *1*› ‹*0* < *k*› ‹*d dvd k*› *b-in* [*unfolded G-def H-def f-def*]]
by *argo*
finally have *2*:
$(\sum m \in b.\ \Phi\ m * \textit{unity-root}\ d\ (\textit{int}\ m)) = (\textit{totient}\ k\ \textit{div}\ \textit{totient}\ d) * \Phi\ l * \textit{unity-root}\ d\ (\textit{int}\ l)$
by *blast*
from *b-not-empty 2* **show** *?thesis* **by** *auto*
qed
also have ... = $((\textit{totient}\ k\ \textit{div}\ \textit{totient}\ d) * (\Phi\ (\textit{the-elem}\ (f\ \text{‘}\ b)) * \textit{unity-root}\ d\ (\textit{int}\ (\textit{the-elem}\ (f\ \text{‘}\ b)))))$
proof −
have *foral*: $(\bigwedge y.\ y \in b \Longrightarrow f\ y = f\ l)$
using *m-mod l-prop* **unfolding** *f-def* **by** *blast*
have *eq*: *the-elem* $(f\ \text{‘}\ b) = f\ l$
by (*simp add*: *b-not-empty foral the-elem-image-unique*)
have *per*: *periodic-arithmetic* Φ *d* **using** *prod* Φ.*dir-periodic-arithmetic* **by** *blast*
show *?thesis*
unfolding *eq* **using** *mod-periodic-arithmetic*[*OF per*, *of l mod d l*]
by (*auto simp*: *f-def unity-root-mod zmod-int*)
qed
finally show $(\sum m \in b.\ \Phi\ m * \textit{unity-root}\ d\ (\textit{int}\ m)) = ((\textit{totient}\ k\ \textit{div}\ \textit{totient}\ d) * (\Phi\ (\textit{the-elem}\ (f\ \text{‘}\ b)) * \textit{unity-root}\ d\ (\textit{int}\ (\textit{the-elem}\ (f\ \text{‘}\ b)))))$
by *blast*
qed
have ... =
$(\sum b \in (\textit{rcosets}_G\ \textit{kernel}\ G\ H\ f)\ .\ (\textit{totient}\ k\ \textit{div}\ \textit{totient}\ d) * (\Phi\ (\textit{the-elem}\ (f\ \text{‘}\ b)) * \textit{unity-root}\ d\ (\textit{int}\ (\textit{the-elem}\ (f\ \text{‘}\ b)))))$
by *blast*

also have *eq''*:
  ... = ($\sum$*h* ∈ *carrier H* . (*totient k div totient d*) ∗ (Φ (*h*) ∗ *unity-root d* (*int* (*h*))))
  unfolding *H-def G-def f-def*
  by (*rule sum.reindex-bij-betw*[*OF residue-mult-group-kernel-partition*(*6*)[*OF* ‹*d* > *1*› ‹*0* < *k*› ‹*d dvd k*›]])
 finally have *2*: ($\sum$*m* | *m* ∈ {*1..k*} ∧ *coprime m k*. Φ(*m*) ∗ *unity-root d m*) =
   (*totient k div totient d*)∗($\sum$*h* ∈ *carrier H* . (Φ (*h*) ∗ *unity-root d* (*int* (*h*))))
  using *1* **by** (*simp add*: *eq'' eq''' sum-distrib-left*)
 also have ... = (*totient k div totient d*)∗($\sum$*m* | *m* ∈ {*1..d*} ∧ *coprime m d* . (Φ (*m*) ∗ *unity-root d* (*int* (*m*))))
  unfolding *H-def residue-mult-group-def* **by** (*simp add*: *totatives-def Suc-le-eq*)
 finally show *?thesis* **by** *simp*
qed

Theorem 8.19

theorem (**in** *dcharacter*) *primitive-iff-separable*:
 primitive-dchar n χ ⟷ (∀ *k*>*0*. *separable k*)
proof (*cases χ* = *principal-dchar n*)
 case *True*
 thus *?thesis*
  using *principal-not-primitive principal-not-totally-separable* **by** *auto*
next
 case *False*
 note *nonprincipal* = *this*
 show *?thesis*
 proof
  assume *primitive-dchar n χ*
  then interpret *A*: *primitive-dchar n residue-mult-group n χ* **by** *auto*
  show (∀ *k*. *k* > *0* ⟶ *separable k*)
   using *n A.primitive-encoding*(*2*) **by** *blast*
 next
  assume *tot-separable*: ∀ *k*>*0*. *separable k*
  {
   assume *as*: ¬ *primitive-dchar n χ*
   have ∃ *r*. *r* ≠ *0* ∧ ¬ *coprime r n* ∧ *gauss-sum r* ≠ *0*
   proof −
    from *n* **have** *n* > *0* **by** *simp*
    define *d* **where** *d* = *conductor*
    have *d* > *0* **unfolding** *d-def* **using** *conductor-gr-0* **.**
    then have *d* > *1* **using** *nonprincipal d-def conductor-eq-1-iff-principal* **by** *auto*
    have *d* < *n* **unfolding** *d-def* **using** *nonprimitive-imp-conductor-less*[*OF as*] **.**
    have *d dvd n* **unfolding** *d-def* **using** *conductor-dvd* **by** *blast*
    define *r* **where** *r* = *n div d*
    have *0*: *r* ≠ *0* **unfolding** *r-def*
     using ‹*0* < *n*› ‹*d dvd n*› *dvd-div-gt0* **by** *auto*

have *gcd r n* > *1*
unfolding *r-def*
proof −
have *n div d* > *1* **using** ‹*1* < *n*› ‹*d* < *n*› ‹*d dvd n*› **by** *auto*
have *n div d dvd n* **using** ‹*d dvd n*› **by** *force*
have *gcd* (*n div d*) *n* = *n div d* **using** *gcd-nat.absorb1*[*OF* ‹*n div d dvd n*›] **by** *blast*
then show *1* < *gcd* (*n div d*) *n* **using** ‹*n div d* > *1*› **by** *argo*
qed
then have *1*: ¬ *coprime r n* **by** *auto*
define χ_1 **where** χ_1 = *principal-dchar n*
from *primitive-principal-form*[*OF nonprincipal*]
obtain Φ **where**
prod: (∀ *k*. χ(*k*) = Φ(*k*)∗χ_1(*k*)) ∧ *primitive-dchar d* Φ
using *d-def* **unfolding** χ_1*-def* **by** *blast*
then have *prod1*: (∀ *k*. χ(*k*) = Φ(*k*)∗χ_1(*k*)) *primitive-dchar d* Φ **by** *blast*+
then interpret Φ: *primitive-dchar d residue-mult-group d* Φ
by *auto*

have *gauss-sum r* = (∑ *m* = *1*..*n* . χ(*m*) ∗ *unity-root n* (*m*∗*r*))
unfolding *gauss-sum-def* **by** *blast*
also have . . . = (∑ *m* = *1*..*n* . Φ(*m*)∗χ_1(*m*) ∗ *unity-root n* (*m*∗*r*))
by (*rule sum.cong*,*auto simp add*: *prod*)
also have . . . = (∑ *m* | *m* ∈ {*1*..*n*} ∧ *coprime m n*. Φ(*m*)∗χ_1(*m*) ∗ *unity-root n* (*m*∗*r*))
by (*intro sum.mono-neutral-right*) (*auto simp*: χ_1*-def principal-dchar-def*)
also have . . . = (∑ *m* | *m* ∈ {*1*..*n*} ∧ *coprime m n*. Φ(*m*)∗χ_1(*m*) ∗ *unity-root d m*)
proof (*rule sum.cong*,*simp*)
fix *x*
assume *x* ∈ {*m* ∈ {*1*..*n*}. *coprime m n*}
have *unity-root n* (*int* (*x* ∗ *r*)) = *unity-root d* (*int x*)
using *unity-div-num*[*OF* ‹*n* > *0*› ‹*d* > *0*› ‹*d dvd n*›]
by (*simp add*: *algebra-simps r-def*)
then show Φ *x* ∗ χ_1 *x* ∗ *unity-root n* (*int* (*x* ∗ *r*)) =
Φ *x* ∗ χ_1 *x* ∗ *unity-root d* (*int x*) **by** *auto*
qed
also have . . . = (∑ *m* | *m* ∈ {*1*..*n*} ∧ *coprime m n*. Φ(*m*) ∗ *unity-root d m*)
by (*rule sum.cong*,*auto simp add*: χ_1*-def principal-dchar-def*)
also have . . . = (*totient n div totient d*) ∗ (∑ *m* | *m* ∈ {*1*..*d*} ∧ *coprime m d*. Φ(*m*) ∗ *unity-root d m*)
using *primitive-iff-separable-lemma*[*OF prod* ‹*d* > *1*› ‹*n* > *0*› ‹*d dvd n*› ‹*n* > *1*›] **by** *blast*
also have . . . = (*totient n div totient d*) ∗ Φ.*gauss-sum 1*
proof −
have Φ.*gauss-sum 1* = (∑ *m* = *1*..*d* . Φ *m* ∗ *unity-root d* (*int* (*m*)))
by (*simp add*: Φ.*gauss-sum-def*)
also have . . . = (∑ *m* | *m* ∈ {*1*..*d*} . Φ *m* ∗ *unity-root d* (*int m*))

by (*rule sum.cong,auto*)
also have ... $=$ ($\sum$ *m* | *m* $\in$ {*1..d*} $\wedge$ *coprime m d*. $\Phi(m) *$ *unity-root d m*)
by (*rule sum.mono-neutral-right*) (*use* Φ.*eq-zero* **in** *auto*)
finally have Φ.*gauss-sum 1* $=$ ($\sum$ *m* | *m* $\in$ {*1..d*} $\wedge$ *coprime m d*. $\Phi(m)$ $*$ *unity-root d m*)
by *blast*
then show *?thesis* **by** *metis*
qed
finally have *g-expr*: *gauss-sum r* $=$ (*totient n div totient d*) $*$ Φ.*gauss-sum 1*
by *blast*
have *t-non-0*: *totient n div totient d* $\neq$ *0*
by (*simp add*: ‹*0* $<$ *n*› ‹*d dvd n*› *dvd-div-gt0 totient-dvd*)
have (*norm* (Φ.*gauss-sum 1*))^2 $=$ *d*
using Φ.*primitive-encoding*(*3*) **by** *simp*
then have Φ.*gauss-sum 1* $\neq$ *0*
using ‹*0* $<$ *d*› **by** *auto*
then have *2*: *gauss-sum r* $\neq$ *0*
using *g-expr t-non-0* **by** *auto*
from *0 1 2* **show** $\exists$ *r*. *r* $\neq$ *0* $\wedge$ $\neg$ *coprime r n* $\wedge$ *gauss-sum r* $\neq$ *0*
by *blast*
qed
}
note *contr* $=$ *this*

show *primitive-dchar n* χ
proof (*rule ccontr*)
assume $\neg$ *primitive-dchar n* χ
then obtain *r* **where** *1*: *r* $\neq$ *0* $\wedge$ $\neg$ *coprime r n* $\wedge$ *gauss-sum r* $\neq$ *0*
using *contr* **by** *blast*
from *global-separability-condition tot-separable*
have *2*: ($\forall$ *k*>*0*. $\neg$ *coprime k n* $\longrightarrow$ *gauss-sum k* $=$ *0*)
by *blast*
from *1 2* **show** *False* **by** *blast*
qed
qed
qed

Theorem 8.20

theorem (**in** *primitive-dchar*) *fourier-primitive*:
includes *no vec-lambda-syntax*
fixes τ :: *complex*
defines $\tau \equiv$ *gauss-sum 1* / *sqrt n*
shows χ *m* $=$ τ / *sqrt n* $*$ ($\sum$ *k*=*1..n*. *cnj* (χ *k*) $*$ *unity-root n* ($-m*k$))
and *norm* τ $=$ *1*
proof $-$
have *chi-not-principal*: $\chi \neq$ *principal-dchar n*
using *principal-not-totally-separable primitive-encoding*(*2*) **by** *blast*

then have *case-0*: $(\sum k{=}1..n.\ \chi\ k) = 0$
proof −
have *sum* χ $\{0..n{-}1\}$ = *sum* χ $\{1..n\}$
using *periodic-arithmetic-sum-periodic-arithmetic-shift*[*OF dir-periodic-arithmetic*, *of 1*] *n*
by *auto*
also have $\{0..n{-}1\} = \{..{<}n\}$
using *n* **by** *auto*
finally show $(\sum n = 1..n\ .\ \chi\ n) = 0$
using *sum-dcharacter-block chi-not-principal* **by** *simp*
qed

have χ *m* =
$(\sum k = 1..n.\ 1\ /\ \textit{of-nat}\ n * \textit{gauss-sum-int}\ (-\ \textit{int}\ k) *$
unity-root n (*int* (*m* ∗ *k*)))
using *dcharacter-fourier-expansion*[*of m*] **by** *auto*
also have ... = $(\sum k = 1..n.\ 1\ /\ \textit{of-nat}\ n * \textit{gauss-sum}\ (\textit{nat}\ ((-\ k)\ \textit{mod}\ n)) *$
unity-root n (*int* (*m* ∗ *k*)))
by (*auto simp*: *gauss-sum-int-conv-gauss-sum*)
also have ... = $(\sum k = 1..n.\ 1\ /\ \textit{of-nat}\ n * \textit{cnj}\ (\chi\ (\textit{nat}\ ((-\ k)\ \textit{mod}\ n))) *$ *gauss-sum 1* ∗ *unity-root n* (*int* (*m* ∗ *k*)))
proof (*rule sum.cong*,*simp*)
fix *k*
assume $k \in \{1..n\}$
have *gauss-sum* (*nat* (− *int k mod int n*)) =
cnj (χ (*nat* (− *int k mod int n*))) ∗ *gauss-sum 1*
proof (*cases nat* ((− *k*) *mod n*) > *0*)
case *True*
then show *?thesis*
using *mp*[*OF spec*[*OF primitive-encoding*(*2*)] *True*]
unfolding *separable-def* **by** *auto*
next
case *False*
then have *nat-0*: *nat* ((− *k*) *mod n*) = *0* **by** *blast*
show *?thesis*
proof −
have *gauss-sum* (*nat* (− *int k mod int n*)) = *gauss-sum 0*
using *nat-0* **by** *argo*
also have ... = $(\sum m = 1..n.\ \chi\ m)$
unfolding *gauss-sum-def* **by** (*rule sum.cong*) *auto*
also have ... = *0* **using** *case-0* **by** *blast*
finally have *1*: *gauss-sum* (*nat* (− *int k mod int n*)) = *0*
by *blast*

have *2*: *cnj* (χ (*nat* (− *int k mod int n*))) = *0*
using *nat-0 zero-eq-0* **by** *simp*
show *?thesis* **using** *1 2* **by** *simp*
qed

qed
then show *1 / of-nat n ∗ gauss-sum (nat (− int k mod int n)) ∗ unity-root n (int (m ∗ k)) =*
1 / of-nat n ∗ cnj (χ (nat (− int k mod int n))) ∗ gauss-sum 1 ∗ unity-root n (int (m ∗ k))
by *auto*
qed
also have ... = $(\sum k = 1..n.$ *1 / of-nat n ∗ cnj (χ (nat (− int k mod int n))) ∗*
gauss-sum 1 ∗ unity-root n (int (m ∗ (nat (int k mod int n)))))
proof *(rule sum.cong,simp)*
fix *x*
assume *x ∈ {1..n}*
have *unity-root n (m ∗ x) = unity-root n (m ∗ x mod n)*
using *unity-root-mod-nat[of n m∗x]* **by** *(simp add: nat-mod-as-int)*
also have ... = *unity-root n (m ∗ (x mod n))*
by *(metis mod-mult-right-eq nat-mod-as-int unity-root-mod-nat)*
finally have *unity-root n (m ∗ x) = unity-root n (m ∗ (x mod n))* **by** *blast*
then show *1 / of-nat n ∗ cnj (χ (nat (− int x mod int n))) ∗*
gauss-sum 1 ∗ unity-root n (int (m ∗ x)) =
1 / of-nat n ∗ cnj (χ (nat (− int x mod int n))) ∗ gauss-sum 1 ∗
unity-root n (int (m ∗ nat (int x mod int n)))
by *(simp add: nat-mod-as-int)*
qed
also have ... = $(\sum k = 0..n-1.$ *1 / of-nat n ∗ cnj (χ k) ∗ gauss-sum 1 ∗ unity-root n (− int (m ∗ k)))*
proof −
have *b: bij-betw (λk. nat((−k) mod n)) {1..n} {0..n−1}*
unfolding *bij-betw-def*
proof
show *inj-on (λk. nat (− int k mod int n)) {1..n}*
unfolding *inj-on-def*
proof *(safe)*
fix *x y*
assume *a1: x ∈ {1..n} y ∈ {1..n}*
assume *a2: nat (− x mod n) = nat (− y mod n)*
then have *(− x) mod n = − y mod n*
using *n eq-nat-nat-iff* **by** *auto*
then have *[−int x = − int y] (mod n)*
using *unique-euclidean-semiring-class.cong-def* **by** *blast*
then have *[x = y] (mod n)*
by *(simp add: cong-int-iff cong-minus-minus-iff)*
then have *cong: x mod n = y mod n* **using** *unique-euclidean-semiring-class.cong-def* **by** *blast*
then show *x = y*
proof *(cases x = n)*
case *True* **then show** *?thesis* **using** *cong a1(2)* **by** *auto*
next
case *False*

then have $x\ mod\ n = x$ **using** $a1(1)$ **by** *auto*
then have $y \neq n$ **using** $a1(1)$ *local.cong* **by** *fastforce*
then have $y\ mod\ n = y$ **using** $a1(2)$ **by** *auto*
then show *?thesis* **using** ‹$x\ mod\ n = x$› *cong* **by** *linarith*
qed
qed
show $(\lambda k.\ nat\ (-\ int\ k\ mod\ int\ n))$ ‘ $\{1..n\} = \{0..n - 1\}$
unfolding *image-def*
proof
let $?A = \{y.\ \exists x\in\{1..n\}.\ y = nat\ (-\ int\ x\ mod\ int\ n)\}$
let $?B = \{0..n - 1\}$
show $?A \subseteq ?B$
proof
fix y
assume $y \in \{y.\ \exists x\in\{1..n\}.\ y = nat\ (-\ int\ x\ mod\ int\ n)\}$
then obtain x **where** $x\in\{1..n\} \wedge y = nat\ (-\ int\ x\ mod\ int\ n)$ **by** *blast*
then show $y \in \{0..n - 1\}$ **by** (*simp add*: *nat-le-iff of-nat-diff*)
qed
show $?A \supseteq ?B$
proof
fix x
assume 1: $x \in \{0..n-1\}$
then have $n - x \in \{1..n\}$
using n **by** *auto*
have $x = nat\ (-\ int\ (n-x)\ mod\ int\ n)$
proof −
have $nat\ (-\ int\ (n-x)\ mod\ int\ n) = nat\ (int\ x)\ mod\ int\ n$
apply(*simp add*: *int-ops*(*6*),*rule conjI*)
using ‹$n - x \in \{1..n\}$› **by** *force+*
also have $\ldots = x$
using $1\ n$ **by** *auto*
finally show *?thesis* **by** *presburger*
qed
then show $x \in \{y.\ \exists x\in\{1..n\}.\ y = nat\ (-\ int\ x\ mod\ int\ n)\}$
using ‹$n - x \in \{1..n\}$› **by** *blast*
qed
qed
qed
show *?thesis*
proof −
have 1: $(\sum k = 1..n.\ 1\ /\ of\text{-}nat\ n * cnj\ (\chi\ (nat\ (-\ int\ k\ mod\ int\ n))) *$
$gauss\text{-}sum\ 1 * unity\text{-}root\ n\ (int\ (m * nat\ (int\ k\ mod\ int\ n)))) =$
$(\sum x = 1..n.\ 1\ /\ of\text{-}nat\ n * cnj\ (\chi\ (nat\ (-\ int\ x\ mod\ int\ n))) *$
$gauss\text{-}sum\ 1 * unity\text{-}root\ n\ (-\ int\ (m * nat\ (-\ int\ x\ mod\ int\ n))))$
proof (*rule sum.cong*,*simp*)
fix x
have $(int\ m * (int\ x\ mod\ int\ n))\ mod\ n = (m*x)\ mod\ n$
by (*simp add*: *mod-mult-right-eq zmod-int*)
also have $\ldots = (-\ ((-\ int\ (m*x)\ mod\ n)))\ mod\ n$

by (*simp add*: *mod-minus-eq of-nat-mod*)
have (*int m* ∗ (*int x mod int n*)) *mod n* = (− (*int m* ∗ (− *int x mod int n*))) *mod n*
apply(*subst mod-mult-right-eq*,*subst add.inverse-inverse*[*symmetric*],*subst* (*5*) *add.inverse-inverse*[*symmetric*])
by (*subst minus-mult-minus*,*subst mod-mult-right-eq*[*symmetric*],*auto*)
then have *unity-root n* (*int m* ∗ (*int x mod int n*)) =
unity-root n (− (*int m* ∗ (− *int x mod int n*)))
using *unity-root-mod*[*of n int m* ∗ (*int x mod int n*)]
unity-root-mod[*of n* − (*int m* ∗ (− *int x mod int n*))] **by** *argo*
then show *1* / *of-nat n* ∗ *cnj* (χ (*nat* (− *int x mod int n*))) ∗
gauss-sum 1 ∗
unity-root n (*int* (*m* ∗ *nat* (*int x mod int n*))) =
1 / *of-nat n* ∗ *cnj* (χ (*nat* (− *int x mod int n*))) ∗
gauss-sum 1 ∗
unity-root n (− *int* (*m* ∗ *nat* (− *int x mod int n*)))
by *clarsimp*
qed
also have *2*: (∑*x* = *1*..*n*. *1* / *of-nat n* ∗ *cnj* (χ (*nat* (− *int x mod int n*))) ∗
gauss-sum 1 ∗ *unity-root n* (− *int* (*m* ∗ *nat* (− *int x mod int n*)))) =
(∑*md* = *0*..*n* − *1*. *1* / *of-nat n* ∗ *cnj* (χ *md*) ∗ *gauss-sum 1* ∗
unity-root n (− *int* (*m* ∗ *md*)))
using *sum.reindex-bij-betw*[*OF b*, *of* λ*md*. *1* / *of-nat n* ∗ *cnj* (χ *md*) ∗ *gauss-sum 1* ∗ *unity-root n* (− *int* (*m* ∗ *md*))]
by *blast*
also have *3*: . . . = (∑*k* = *0*..*n* − *1*.
1 / *of-nat n* ∗ *cnj* (χ *k*) ∗ *gauss-sum 1* ∗
unity-root n (− *int* (*m* ∗ *k*))) **by** *blast*
finally have (∑*k* = *1*..*n*. *1* / *of-nat n* ∗ *cnj* (χ (*nat* (− *int k mod int n*))) ∗
gauss-sum 1 ∗ *unity-root n* (*int* (*m* ∗ *nat* (*int k mod int n*)))) =
(∑*k* = *0*..*n* − *1*.
1 / *of-nat n* ∗ *cnj* (χ *k*) ∗ *gauss-sum 1* ∗
unity-root n (− *int* (*m* ∗ *k*))) **using** *1 2 3* **by** *argo*
then show *?thesis* **by** *blast*
qed
qed
also have . . . = (∑*k* = *1*..*n*.
1 / *of-nat n* ∗ *cnj* (χ *k*) ∗ *gauss-sum 1* ∗
unity-root n (− *int* (*m* ∗ *k*)))
proof −
let *?f* = (λ*k*. *1* / *of-nat n* ∗ *cnj* (χ *k*) ∗ *gauss-sum 1* ∗ *unity-root n* (− *int* (*m* ∗ *k*)))
have *?f 0* = *0*
using *zero-eq-0* **by** *auto*
have *?f n* = *0*
using *zero-eq-0 mod-periodic-arithmetic*[*OF dir-periodic-arithmetic*, *of n 0*]
by *simp*
have (∑*n* = *0*..*n* − *1*. *?f n*) = (∑*n* = *1*..*n* − *1*. *?f n*)
using *sum-shift-lb-Suc0-0*[*of ?f*, *OF* ‹*?f 0* = *0*›]

by *auto*
also have ... = ($\sum$ *n* = *1..n. ?f n*)
proof (*rule sum.mono-neutral-left*,*simp*,*simp*,*safe*)
fix *i*
assume *i* ∈ {*1..n*} *i* ∉ {*1..n* − *1*}
then have *i* = *n* **using** *n* **by** *auto*
then show *1* / *of-nat n* ∗ *cnj* (χ *i*) ∗ *gauss-sum 1* ∗ *unity-root n* (− *int* (*m* ∗ *i*)) = *0*
using ‹*?f n* = *0*› **by** *blast*
qed
finally show *?thesis* **by** *blast*
qed
also have ... = ($\sum$ *k* = *1..n.* (τ / *sqrt n*) ∗ *cnj* (χ *k*) ∗ *unity-root n* (− *int* (*m* ∗ *k*)))
proof (*rule sum.cong*,*simp*)
fix *x*
assume *x* ∈ {*1..n*}
have τ / *sqrt* (*real n*) = *1* / *of-nat n* ∗ *gauss-sum 1*
proof −
have τ / *sqrt* (*real n*) = *gauss-sum 1* / *sqrt n* / *sqrt n*
using *assms* **by** *auto*
also have ... = *gauss-sum 1* / (*sqrt n* ∗ *sqrt n*)
by (*subst divide-divide-eq-left*,*subst of-real-mult*,*blast*)
also have ... = *gauss-sum 1* / *n*
using *real-sqrt-mult-self* **by** *simp*
finally show *?thesis* **by** *simp*
qed
then show
1 / *of-nat n* ∗ *cnj* (χ *x*) ∗ *gauss-sum 1* ∗ *unity-root n* (− *int* (*m* ∗ *x*)) =
(τ / *sqrt n*) ∗ *cnj* (χ *x*) ∗ *unity-root n* (− *int* (*m* ∗ *x*)) **by** *simp*
qed
also have ... = τ / *sqrt* (*real n*) ∗
($\sum$ *k* = *1..n. cnj* (χ *k*) ∗ *unity-root n* (− *int* (*m* ∗ *k*)))
proof −
have ($\sum$ *k* = *1..n.* τ / *sqrt* (*real n*) ∗ *cnj* (χ *k*) ∗ *unity-root n* (− *int* (*m* ∗ *k*))) =
($\sum$ *k* = *1..n.* τ / *sqrt* (*real n*) ∗ (*cnj* (χ *k*) ∗ *unity-root n* (− *int* (*m* ∗ *k*))))
by (*rule sum.cong*,*simp*, *simp add*: *algebra-simps*)
also have ... = τ / *sqrt* (*real n*) ∗ ($\sum$ *k* = *1..n. cnj* (χ *k*) ∗ *unity-root n* (− *int* (*m* ∗ *k*)))
by (*rule sum-distrib-left*[*symmetric*])
finally show *?thesis* **by** *blast*
qed

finally show χ *m* = (τ / *sqrt* (*real n*)) ∗
($\sum$ *k*=*1..n. cnj* (χ *k*) ∗ *unity-root n* (− *int m* ∗ *int k*)) **by** *simp*

have *1*: *norm* (*gauss-sum 1*) = *sqrt n*

using *gauss-sum-1-mod-square-eq-k*[*OF primitive-encoding*(*2*)]
by (*simp add*: *cmod-def*)
from *assms* **have** *2*: *norm* τ = *norm* (*gauss-sum 1*) / |*sqrt n*|
by (*simp add*: *norm-divide*)
show *norm* τ = *1* **using** *1 2 n* **by** *simp*
qed

unbundle *vec-lambda-syntax*

end

8 The Pólya–Vinogradov Inequality

theory *Polya-Vinogradov*
imports
Gauss-Sums
Dirichlet-Series.Divisor-Count
begin

unbundle *no vec-lambda-syntax*

8.1 The case of primitive characters

We first prove a stronger variant of the Pólya–Vinogradov inequality for primitive characters. The fully general variant will then simply be a corollary of this. First, we need some bounds on logarithms, exponentials, and the harmonic numbers:

lemma *exp-1-less-powr*:
assumes $x > (0::real)$
shows *exp 1* < (*1* + *1* / *x*) *powr* (*x*+*1*)
proof −
have *1* < (*x* + *1*) ∗ *ln* ((*x* + *1*) / *x*) (**is** - < *?f x*)
proof (*rule DERIV-neg-imp-decreasing-at-top*[**where** *?f* = *?f*])
fix *t* **assume** *t*: $x \leq t$
have (*?f has-field-derivative* (*ln* (*1* + *1* / *t*) − *1* / *t*)) (*at t*)
using *t assms* **by** (*auto intro*!: *derivative-eq-intros simp*:*divide-simps*)
moreover have *ln* (*1* + *1* / *t*) − *1* / *t* < *0*
using *ln-add-one-self-less-self*[*of 1* / *t*] *t assms* **by** *auto*
ultimately show ∃ *y*. ((λ*t*. (*t* + *1*) ∗ *ln* ((*t* + *1*) / *t*)) *has-real-derivative y*) (*at t*) ∧ *y* < *0*
by *blast*
qed *real-asymp*
thus *exp 1* < (*1* + *1* / *x*) *powr* (*x* + *1*)
using *assms* **by** (*simp add*: *powr-def field-simps*)
qed

lemma *harm-aux-ineq-1*:
fixes *k* :: *real*

```
  assumes k > 1
  shows 1 / k < ln (1 + 1 / (k − 1))
proof −
  have k−1 > 0 ‹k > 0› using assms by simp+
  from exp-1-less-powr[OF ‹k−1 > 0›]
  have eless: exp 1 < (1 + 1 / (k − 1)) powr k by simp
  then have n-z: (1 + 1 / (k − 1)) powr k > 0
      using assms not-exp-less-zero by auto

  have (1::real) = ln (exp(1)) using ln-exp by auto
  also have ... < ln ((1 + 1 / (k − 1)) powr k)
    by (meson eless dual-order.strict-trans exp-gt-zero ln-less-cancel-iff)
  also have ... = k ∗ ln (1 + 1 / (k − 1))
    using ln-powr n-z by simp
  finally have 1 < k ∗ ln (1 + 1 / (k − 1))
    by blast
  then show ?thesis using assms by (simp add: field-simps)
qed

lemma harm-aux-ineq-2-lemma:
  assumes x ≥ (0::real)
  shows  1 < (x + 1) ∗ ln (1 + 2 / (2 ∗ x + 1))
proof −
  have 0 < ln (1+2/(2∗x+1)) − 1 / (x + 1) (is - < ?f x)
  proof (rule DERIV-neg-imp-decreasing-at-top[where ?f = ?f])
    fix t assume t: x ≤ t
    from assms t have 3 + 8 ∗ t + 4 ∗ tˆ2 > 0
      by (intro add-pos-nonneg) auto
    hence ∗: 3 + 8 ∗ t + 4 ∗ tˆ2 ≠ 0
      by auto
    have (?f has-field-derivative (−1 / ((1 + t)ˆ2 ∗ (3 + 8 ∗ t + 4 ∗ t ˆ 2))))
(at t)
    apply (insert assms t ∗, (rule derivative-eq-intros refl | simp add: add-pos-pos)+)
      apply (auto simp: divide-simps)
      apply (auto simp: algebra-simps power2-eq-square)
      done
    moreover have −1 / ((1 + t)ˆ2 ∗ (3 + 8 ∗ t + 4 ∗ tˆ2)) < 0
      using t assms by (intro divide-neg-pos mult-pos-pos add-pos-nonneg) auto
    ultimately show ∃ y. (?f has-real-derivative y) (at t) ∧ y < 0
      by blast
  qed real-asymp
  thus 1 < (x + 1) ∗ ln (1+2/(2∗x+1))
    using assms by (simp add: field-simps)
qed

lemma harm-aux-ineq-2:
  fixes k :: real
  assumes k ≥ 1
  shows  1 / (k + 1) < ln (1 + 2 / (2 ∗ k + 1))
```

```
proof -
  have k > 0 using assms by auto
  have 1 < (k + 1) * ln (1 + 2 / (2 * k + 1))
    using harm-aux-ineq-2-lemma assms by simp
  then show ?thesis
    by (simp add: ‹0 < k› add-pos-pos mult.commute mult-imp-div-pos-less)
qed

lemma nat-0-1-induct [case-names 0 1 step]:
  assumes P 0 P 1 ⋀n. n ≥ 1 ⟹ P n ⟹ P (Suc n)
  shows   P n
proof (induction n rule: less-induct)
  case (less n)
  show ?case
    using assms(3)[OF - less.IH[of n − 1]]
    by (cases n ≤ 1)
      (insert assms(1−2),auto simp: eval-nat-numeral le-Suc-eq)
qed

lemma harm-less-ln:
  fixes m :: nat
  assumes m > 0
  shows   harm m < ln (2 * m + 1)
  using assms
proof (induct m rule: nat-0-1-induct)
  case 0
  then show ?case by blast
next
  case 1
  have harm 1 = (1::real) unfolding harm-def by simp
  have harm 1 < ln (3::real)
    by (subst ‹harm 1 = 1›,subst ln3-gt-1,simp)
  then show ?case by simp
next
  case (step n)
  have harm (n+1) = harm n + 1/(n+1)
   by ((subst Suc-eq-plus1[symmetric])+,subst harm-Suc,subst inverse-eq-divide,blast)
  also have ... < ln (real (2 * n + 1)) + 1/(n+1)
    using step(1−2) by auto
  also have ... < ln (real (2 * n + 1)) + ln (1+2/(2*n+1))
  proof -
    from step(1) have real n ≥ 1 by simp
    have 1 / real (n + 1) < ln (1 + 2 / real (2 * n + 1))
      using harm-aux-ineq-2[OF ‹1 ≤ (real n)›]  by (simp add: add.commute)
    then show ?thesis by auto
  qed
  also have ... = ln ((2 * n + 1) * (1+2/(2*n+1)))
    by (auto simp add: ln-div divide-simps)
  also have ... = ln (2*(n+1)+1)
```

proof −
have $(2 * n + 1) * (1+2/(2*n+1)) = 2*(n+1)+1$
by (*simp add*: *field-simps*)
then show *?thesis* **by** *presburger*
qed
finally show *?case* **by** *simp*
qed

Theorem 8.21

theorem (**in** *primitive-dchar*) *polya-vinogradov-inequality-primitive*:
fixes x :: *nat*
shows *norm* $(\sum m=1..x.\ \chi\ m) < sqrt\ n * ln\ n$
proof −
define τ :: *complex* **where** τ = *gauss-sum 1 div sqrt n*
have τ-*mod*: *norm* $\tau = 1$ **using** *fourier-primitive*(*2*)
by (*simp add*: τ-*def*)
{
fix m
have $\chi\ m = (\tau\ div\ sqrt\ n) * (\sum k = 1..n.\ (cnj\ (\chi\ k)) * unity\text{-}root\ n\ (-m*k))$
using *fourier-primitive*(*1*)[*of m*] τ-*def* **by** *blast***}**
note *chi-expr* = *this*
have $(\sum m = 1..x.\ \chi(m)) = (\sum m = 1..x.\ (\tau\ div\ sqrt\ n) * (\sum k = 1..n.\ (cnj\ (\chi\ k)) * unity\text{-}root\ n\ (-m*k)))$
by(*rule sum.cong*[*OF refl*]) (*use chi-expr* **in** *blast*)
also have $\ldots = (\sum m = 1..x.\ (\sum k = 1..n.\ (\tau\ div\ sqrt\ n) * ((cnj\ (\chi\ k)) * unity\text{-}root\ n\ (-m*k))))$
by (*rule sum.cong*,*simp*,*simp add*: *sum-distrib-left*)
also have $\ldots = (\sum k = 1..n.\ (\sum m = 1..x.\ (\tau\ div\ sqrt\ n) * ((cnj\ (\chi\ k)) * unity\text{-}root\ n\ (-m*k))))$
by (*rule sum.swap*)
also have $\ldots = (\sum k = 1..n.\ (\tau\ div\ sqrt\ n) * (cnj\ (\chi\ k) * (\sum m = 1..x.\ unity\text{-}root\ n\ (-m*k))))$
by (*rule sum.cong*,*simp*,*simp add*: *sum-distrib-left*)
also have $\ldots = (\sum k = 1..{<}n.\ (\tau\ div\ sqrt\ n) * (cnj\ (\chi\ k) * (\sum m = 1..x.\ unity\text{-}root\ n\ (-m*k))))$
using n **by** (*intro sum.mono-neutral-right*) (*auto intro*: *eq-zero*)
also have $\ldots = (\tau\ div\ sqrt\ n) * (\sum k = 1..{<}n.\ (cnj\ (\chi\ k) * (\sum m = 1..x.\ unity\text{-}root\ n\ (-m*k))))$
by (*simp add*: *sum-distrib-left*)
finally have $(\sum m = 1..x.\ \chi(m)) = (\tau\ div\ sqrt\ n) * (\sum k = 1..{<}n.\ (cnj\ (\chi\ k) * (\sum m = 1..x.\ unity\text{-}root\ n\ (-m*k))))$
by *blast*
hence *eq*: $sqrt\ n * (\sum m=1..x.\ \chi(m)) = \tau * (\sum k=1..{<}n.\ (cnj\ (\chi\ k) * (\sum m=1..x.\ unity\text{-}root\ n\ (-m*k))))$
by *auto*
define f **where** $f = (\lambda k.\ (\sum m = 1..x.\ unity\text{-}root\ n\ (-m*k)))$

hence $(sqrt\ n) * norm(\sum m = 1..x.\ \chi(m)) = norm(\tau * (\sum k=1..{<}n.\ (cnj\ (\chi\ k) * (\sum m = 1..x.\ unity\text{-}root\ n\ (-m*k)))))$

proof −
have $norm(sqrt\ n * (\sum m{=}1..x.\ \chi(m))) = norm\ (sqrt\ n) * norm((\sum m = 1..x.\ \chi(m)))$
by (*simp add*: *norm-mult*)
also have $\ldots = (sqrt\ n) * norm((\sum m = 1..x.\ \chi(m)))$
by *simp*
finally have *1*: $norm((sqrt\ n) * (\sum m = 1..x.\ \chi(m))) = (sqrt\ n) * norm((\sum m = 1..x.\ \chi(m)))$
by *blast*
then show *?thesis* **using** *eq* **by** *algebra*
qed
also have $\ldots = norm\ (\sum k = 1..{<}n.\ (cnj\ (\chi\ k) * (\sum m = 1..x.\ unity\text{-}root\ n\ ({-}m{*}k))))$
by (*simp add*: *norm-mult* *τ-mod*)
also have $\ldots \le (\sum k = 1..{<}n.\ norm\ (cnj\ (\chi\ k) * (\sum m = 1..x.\ unity\text{-}root\ n\ ({-}m{*}k))))$
using *norm-sum* **by** *blast*
also have $\ldots = (\sum k = 1..{<}n.\ norm\ (cnj\ (\chi\ k)) * norm((\sum m = 1..x.\ unity\text{-}root\ n\ ({-}m{*}k))))$
by (*rule sum.cong*,*simp*, *simp add*: *norm-mult*)
also have $\ldots \le (\sum k = 1..{<}n.\ norm((\sum m = 1..x.\ unity\text{-}root\ n\ ({-}m{*}k))))$
proof −
show *?thesis*
proof (*rule sum-mono*)
fix *k*
assume $k \in \{1..{<}n\}$
define *sum-aux* :: *real* **where** $sum\text{-}aux = norm\ (\sum m{=}1..x.\ unity\text{-}root\ n\ ({-}\ int\ m * int\ k))$
have $sum\text{-}aux \ge 0$ **unfolding** *sum-aux-def* **by** *auto*
have $norm\ (cnj\ (\chi\ k)) \le 1$ **using** *norm-le-1*[*of k*] **by** *simp*
then have $norm\ (cnj\ (\chi\ k)) * sum\text{-}aux \le 1 * sum\text{-}aux$
using ‹$sum\text{-}aux \ge 0$› **by** (*simp add*: *mult-left-le-one-le*)
then show $norm\ (cnj\ (\chi\ k)) *$
$norm\ (\sum m = 1..x.\ unity\text{-}root\ n\ ({-}\ int\ m * int\ k))$
$\le norm\ (\sum m = 1..x.\ unity\text{-}root\ n\ ({-}\ int\ m * int\ k))$
unfolding *sum-aux-def* **by** *argo*
qed
qed
also have $\ldots = (\sum k = 1..{<}n.\ norm(f\ k))$
using *f-def* **by** *blast*
finally have *24*: $(sqrt\ n) * norm(\sum m = 1..x.\ \chi(m)) \le (\sum k = 1..{<}n.\ norm(f\ k))$
by *blast*

{
fix *k* :: *int*
have $f(n{-}k) = cnj(f(k))$
proof −
have $f(n{-}k) = (\sum m = 1..x.\ unity\text{-}root\ n\ ({-}m{*}(n{-}k)))$

```
      unfolding f-def by blast
    also have ... = (∑ m = 1..x. unity-root n (m∗k))
    proof (rule sum.cong,simp)
      fix xa
      assume xa ∈ {1..x}
      have (k ∗ int xa − int n ∗ int xa) mod int n = (k ∗ int xa − 0) mod int n
        by (intro mod-diff-cong) auto
      thus unity-root n (−int xa ∗ (int n − k)) = unity-root n (int xa ∗ k)
      by (metis left-diff-distrib diff-zero minus-diff-eq mult.commute unity-root-mod)
    qed
    also have ... = cnj(f(k))
    proof −
      have cnj(f(k)) = cnj (∑ m = 1..x. unity-root n (− int m ∗ k))
        unfolding f-def by blast
      also have cnj (∑ m = 1..x. unity-root n (− int m ∗ k)) =
           (∑ m = 1..x. cnj(unity-root n (− int m ∗ k)))
        by (rule cnj-sum)
      also have ... = (∑ m = 1..x. unity-root n (int m ∗ k))
        by (intro sum.cong) (auto simp: unity-root-uminus)
      finally show ?thesis by auto
    qed
    finally show f(n−k) = cnj(f(k)) by blast
  qed
  hence norm(f(n−k)) = norm(cnj(f(k))) by simp
  hence norm(f(n−k)) = norm(f(k)) by auto
}
note eq = this
have 25:
  odd n ⟹ (∑ k = 1..n − 1. norm (f (int k))) ≤
             2 ∗ (∑ k = 1..(n−1) div 2. norm (f (int k)))
  even n ⟹ (∑ k = 1..n − 1. norm (f (int k))) ≤
             2 ∗ (∑ k = 1..(n−2) div 2. norm (f (int k))) + norm(f(n div 2))
proof −
  assume odd n
  define g where g = (λk. norm (f k))
  have (n−1) div 2 = n div 2 using ‹odd n› n
    using div-mult-self1-is-m[OF pos2,of n−1]
         odd-two-times-div-two-nat[OF ‹odd n›] by linarith
  have (∑ i=1..n−1. g i) = (∑ i∈{1..n div 2}∪{n div 2<..n−1}. g i)
    using n by (intro sum.cong,auto)
  also have ... = (∑ i∈{1..n div 2}. g i) + (∑ i∈{n div 2<..n−1}. g i)
    by (subst sum.union-disjoint,auto)
  also have (∑ i∈{n div 2<..n−1}. g i) = (∑ i∈{1..n − (n div 2 + 1)}. g (n
− i))
    by (rule sum.reindex-bij-witness[of - λi. n − i λi. n − i],auto)
  also have ... ≤ (∑ i∈{1..n div 2}. g (n − i))
    by (intro sum-mono2,simp,auto simp add: g-def)
  finally have 1: (∑ i=1..n−1. g i) ≤ (∑ i=1..n div 2. g i + g (n − i))
    by (simp add: sum.distrib)
```

have ($\sum i{=}1..n$ *div* $2.$ $g\ i + g\ (n - i)) = (\sum i{=}1..n$ *div* $2.$ $2 * g\ i)$
unfolding *g-def*
apply(*rule sum.cong*,*simp*)
using *eq int-ops*(*6*) **by** *force*
also have $\ldots = 2 * (\sum i{=}1..n$ *div* $2.$ $g\ i)$
by (*rule sum-distrib-left*[*symmetric*])
finally have *2*: $(\sum i{=}1..n$ *div* $2.$ $g\ i + g\ (n - i)) = 2 * (\sum i{=}1..n$ *div* $2.$ g
$i)$
by *blast*
from *1 2* **have** $(\sum i{=}1..n{-}1.\ g\ i) \leq 2 * (\sum i{=}1..n$ *div* $2.$ $g\ i)$ **by** *algebra*
then show $(\sum n = 1..n - 1.$ *norm* $(f\ ($*int* $n))) \leq 2 * (\sum n = 1..(n{-}1)$ *div*
$2.$ *norm* $(f\ ($*int* $n)))$
unfolding *g-def* ‹$(n{-}1)$ *div* $2 = n$ *div* 2› **by** *blast*
next
assume *even n*
define g **where** $g = (\lambda n.$ *norm* $(f\ (n)))$
have $(n{-}2)$ *div* $2 = n$ *div* $2 - 1$ **using** ‹*even n*› n **by** *simp*
have $(\sum i{=}1..n{-}1.\ g\ i) = (\sum i{\in}\{1..{<}n$ *div* $2\}{\cup}\ \{n$ *div* $2\} \cup \{n$ *div* $2{<}..n{-}1\}.$
$g\ i)$
using n **by** (*intro sum.cong*,*auto*)
also have $\ldots = (\sum i{\in}\{1..{<}n$ *div* $2\}.\ g\ i) + (\sum i{\in}\{n$ *div* $2{<}..n{-}1\}.\ g\ i) +$
$g(n$ *div* $2)$
by (*subst sum.union-disjoint*,*auto*)
also have $(\sum i{\in}\{n$ *div* $2{<}..n{-}1\}.\ g\ i) = (\sum i{\in}\{1..n - (n$ *div* $2{+}1)\}.\ g\ (n$
$- i))$
by (*rule sum.reindex-bij-witness*[*of* - $\lambda i.\ n - i\ \lambda i.\ n - i$],*auto*)
also have $\ldots \leq (\sum i{\in}\{1..{<}n$ *div* $2\}.\ g\ (n - i))$
proof (*intro sum-mono2*,*simp*)
have $n - n$ *div* $2 = n$ *div* 2 **using** ‹*even n*› n **by** *auto*
then have $n - (n$ *div* $2 + 1) < n$ *div* 2
using n **by** (*simp add*: *divide-simps*)
then show $\{1..n - (n$ *div* $2 + 1)\} \subseteq \{1..{<}n$ *div* $2\}$ **by** *fastforce*
qed *auto*
finally have *1*: $(\sum i{=}1..n{-}1.\ g\ i) \leq (\sum i{=}1..{<}n$ *div* $2.$ $g\ i + g\ (n - i)) +$
$g(n$ *div* $2)$
by (*simp add*: *sum.distrib*)
have $(\sum i{=}1..{<}n$ *div* $2.$ $g\ i + g\ (n - i)) = (\sum i{=}1..{<}n$ *div* $2.$ $2 * g\ i)$
unfolding *g-def*
apply(*rule sum.cong*,*simp*)
using *eq int-ops*(*6*) **by** *force*
also have $\ldots = 2 * (\sum i{=}1..{<}n$ *div* $2.$ $g\ i)$
by (*rule sum-distrib-left*[*symmetric*])
finally have *2*: $(\sum i{=}1..{<}n$ *div* $2.$ $g\ i + g\ (n - i)) = 2 * (\sum i{=}1..{<}n$ *div*
$2.$ $g\ i)$
by *blast*
from *1 2* **have** *3*: $(\sum i{=}1..n{-}1.\ g\ i) \leq 2 * (\sum i{=}1..{<}n$ *div* $2.$ $g\ i) + g(n$
div $2)$ **by** *algebra*
then have $(\sum i{=}1..n{-}1.\ g\ i) \leq 2 * (\sum i{=}1..(n{-}2)$ *div* $2.$ $g\ i) + g(n$ *div*
$2)$

proof −
have $\{1..<n\ div\ 2\} = \{1..(n-2)\ div\ 2\}$ **by** *auto*
then have $(\sum i=1..<n\ div\ 2.\ g\ i) = (\sum i=1..(n-2)\ div\ 2.\ g\ i)$
by (*rule sum.cong*,*simp*)
then show *?thesis* **using** *3* **by** *presburger*
qed
then show $(\sum k = 1..n - 1.\ norm\ (f\ (int\ k))) \leq 2 * (\sum n = 1..(n-2)\ div\ 2.\ norm\ (f\ (int\ n))) + g(n\ div\ 2)$
unfolding *g-def* **by** *blast*
qed

{**fix** k :: *int*
assume $1 \leq k$ $k \leq n\ div\ 2$
have $k \leq n - 1$
using ‹$k \leq n\ div\ 2$› n **by** *linarith*
define y **where** $y = unity\text{-}root\ n\ (-k)$
define z **where** $z = exp\ (-(pi*k/n)*\ \mathrm{i})$
have $z\hat{}2 = exp\ (2*(-(pi*k/n)*\ \mathrm{i}))$
unfolding *z-def* **using** *exp-double*[*symmetric*] **by** *blast*
also have $\ldots = y$
unfolding *y-def unity-root-conv-exp* **by** (*simp add*: *algebra-simps*)
finally have *z-eq*: $y = z\hat{}2$ **by** *blast*
have *z-not-0*: $z \neq 0$
using *z-eq* **by** (*simp add*: *z-def*)

then have $y \neq 1$
using *unity-root-eq-1-iff-int* ‹$1 \leq k$› ‹$k \leq n - 1$› *not-less*
unity-root-eq-1-iff-int y-def zdvd-not-zless **by** *auto*

have $f(k) = (\sum m = 1..x\ .\ y\hat{}m)$
unfolding *f-def y-def*
by (*subst unity-root-pow*,*rule sum.cong*,*simp*,*simp add*: *algebra-simps*)
also have *sum*: $\ldots = (\sum m = 1..<x+1\ .\ y\hat{}m)$
by (*rule sum.cong*,*fastforce*,*simp*)
also have $\ldots = (\sum m = 0..<x+1\ .\ y\hat{}m) - 1$
by (*subst* (*2*) *sum.atLeast-Suc-lessThan*) *auto*
also have $\ldots = (y\hat{}(x+1) - 1)\ div\ (y - 1) - 1$
using *geometric-sum*[*OF* ‹$y \neq 1$›, *of* $x+1$] **by** (*simp add*: *atLeast0LessThan*)

also have $\ldots = (y\hat{}(x+1) - 1 - (y-1))\ div\ (y - 1)$
proof −
have $y - 1 \neq 0$ **using** ‹$y \neq 1$› **by** *simp*
show *?thesis*
using *divide-diff-eq-iff*[*OF* ‹$y - 1 \neq 0$›, *of* $(y\hat{}(x+1) - 1)$ *1*] **by** *auto*
qed
also have $\ldots = (y\hat{}(x+1) - y)\ div\ (y - 1)$
by (*simp add*: *algebra-simps*)
also have $\ldots = y * (y\hat{}x - 1)\ div\ (y - 1)$

by (*simp add*: *algebra-simps*)
also have ... = $z\hat{}2 * ((z\hat{}2)\hat{}x - 1)$ *div* $(z\hat{}2 - 1)$
unfolding *z-eq* **by** *blast*
also have ... = $z\hat{}2 * (z\hat{}(2*x) - 1)$ *div* $(z\hat{}2 - 1)$
by (*subst power-mult*[*symmetric*, *of z 2 x*],*blast*)
also have ... = $z\hat{}(x+1)*((z\ \hat{}x - inverse(z\hat{}x)))\ /\ (z - inverse(z))$
proof −
have $z\hat{}x \neq 0$ **using** *z-not-0* **by** *auto*
have *1*: $z\ \hat{}\ (2 * x) - 1 = z\hat{}x*(z\ \hat{}x - inverse(z\hat{}x))$
by (*simp add*: *semiring-normalization-rules*(*36*) *right-inverse*[*OF* ‹$z\hat{}x \neq 0$›] *right-diff-distrib′*)
have *2*: $z^2 - 1 = z*(z - inverse(z))$
by (*simp add*: *right-diff-distrib′ semiring-normalization-rules*(*29*) *right-inverse*[*OF* ‹$z \neq 0$›])

have *3*: $z^2 * (z\hat{}x\ /\ z) = z\hat{}(x+1)$
proof −
have $z^2 * (z\hat{}x\ /\ z) = z^2 * (z\hat{}x * inverse\ z)$
by (*simp add*: *inverse-eq-divide*)
also have ... = $z\hat{}(x+1)$
by (*simp add*: *algebra-simps power2-eq-square right-inverse*[*OF* ‹$z \neq 0$›])
finally show *?thesis* **by** *blast*
qed
have $z^2 * (z\ \hat{}\ (2 * x) - 1)\ /\ (z^2 - 1) =$
$z^2 * (z\hat{}x*(z\ \hat{}x - inverse(z\hat{}x)))\ /\ (z*(z - inverse(z)))$
by (*subst 1*, *subst 2*,*blast*)
also have ... = $(z^2 * (z\hat{}x\ /\ z)) * ((z\ \hat{}x - inverse(z\hat{}x)))\ /\ (z - inverse(z))$
by *simp*
also have ... = $z\hat{}(x+1) *((z\ \hat{}x - inverse(z\hat{}x)))\ /\ (z - inverse(z))$
by (*subst 3*,*simp*)
finally show *?thesis* **by** *simp*
qed
finally have $f(k) = z\hat{}(x+1) *((z\ \hat{}x - inverse(z\hat{}x)))\ /\ (z - inverse(z))$ **by** *blast*

then have $norm(f(k)) = norm(z\hat{}(x+1) * (((z\ \hat{}x - inverse(z\hat{}x)))\ /\ (z - inverse(z))))$ **by** *auto*
also have ... = $norm(z\hat{}(x+1)) * norm(((z\ \hat{}x - inverse(z\hat{}x)))\ /\ (z - inverse(z)))$
using *norm-mult* **by** *blast*
also have ... = $norm(((z\ \hat{}x - inverse(z\hat{}x)))\ /\ (z - inverse(z)))$
proof −
have $norm(z) = 1$
unfolding *z-def* **by** *auto*
have $norm(z\hat{}(x+1)) = 1$
by (*subst norm-power*,*simp add*: ‹$norm(z) = 1$›)
then show *?thesis* **by** *simp*
qed

also have ... = *norm*((*exp* (−(*x*∗*pi*∗*k*/*n*)∗ i) − *exp* ((*x*∗*pi*∗*k*/*n*)∗ i)) *div*
(*exp* (−(*pi*∗*k*/*n*)∗ i) − *exp* ((*pi*∗*k*/*n*)∗ i)))
proof −
have *1*: *z* ^ *x* = *exp* (−(*x*∗*pi*∗*k*/*n*)∗ i)
unfolding *z-def*
by (*subst exp-of-nat-mult*[*symmetric*],*simp add*: *algebra-simps*)
have *inverse* (*z* ^ *x*) = *inverse* (*exp* (−(*x*∗*pi*∗*k*/*n*)∗ i))
using ‹*z* ^ *x* = *exp* (−(*x*∗*pi*∗*k*/*n*)∗ i)› **by** *auto*
also have ... = (*exp* ((*x*∗*pi*∗*k*/*n*)∗ i))
by (*simp add*: *exp-minus*)
finally have *2*: *inverse*(*z*^*x*) = *exp* ((*x*∗*pi*∗*k*/*n*)∗ i) **by** *simp*
have *3*: *inverse z* = *exp* ((*pi*∗*k*/*n*)∗ i)
by (*simp add*: *exp-minus z-def*)
show *?thesis* **using** *1 2 3 z-def* **by** *simp*
qed
also have ... = *norm*((*sin* (*x*∗*pi*∗*k*/*n*)) *div* (*sin* (*pi*∗*k*/*n*)))
proof −
have *num*: (*exp* (−(*x*∗*pi*∗*k*/*n*)∗ i) − *exp* ((*x*∗*pi*∗*k*/*n*)∗ i)) = (−*2*∗i∗ *sin*((*x*∗*pi*∗*k*/*n*)))

proof −
have *1*: *exp* (−(*x*∗*pi*∗*k*/*n*)∗ i) = *cos*(−(*x*∗*pi*∗*k*/*n*)) + i ∗ *sin*(−(*x*∗*pi*∗*k*/*n*))
exp ((*x*∗*pi*∗*k*/*n*)∗ i) = *cos*((*x*∗*pi*∗*k*/*n*)) + i ∗ *sin*((*x*∗*pi*∗*k*/*n*))
using *Euler Im-complex-of-real Im-divide-of-nat Im-i-times Re-complex-of-real*
complex-Re-of-int complex-i-mult-minus exp-zero mult.assoc mult.commute
by *force*+
have (*exp* (−(*x*∗*pi*∗*k*/*n*)∗ i) − *exp* ((*x*∗*pi*∗*k*/*n*)∗ i)) =
(*cos*(−(*x*∗*pi*∗*k*/*n*)) + i ∗ *sin*(−(*x*∗*pi*∗*k*/*n*))) −
(*cos*((*x*∗*pi*∗*k*/*n*)) + i ∗ *sin*((*x*∗*pi*∗*k*/*n*)))
using *1* **by** *argo*
also have ... = −*2*∗i∗ *sin*((*x*∗*pi*∗*k*/*n*)) **by** *simp*
finally show *?thesis* **by** *blast*
qed

have *den*: (*exp* (−(*pi*∗*k*/*n*)∗ i) − *exp* ((*pi*∗*k*/*n*)∗ i)) = −*2*∗i∗ *sin*((*pi*∗*k*/*n*))
proof −
have *1*: *exp* (−(*pi*∗*k*/*n*)∗ i) = *cos*(−(*pi*∗*k*/*n*)) + i ∗ *sin*(−(*pi*∗*k*/*n*))
exp ((*pi*∗*k*/*n*)∗ i) = *cos*((*pi*∗*k*/*n*)) + i ∗ *sin*((*pi*∗*k*/*n*))
using *Euler Im-complex-of-real Im-divide-of-nat Im-i-times Re-complex-of-real*

complex-Re-of-int complex-i-mult-minus exp-zero mult.assoc mult.commute
by *force*+
have (*exp* (−(*pi*∗*k*/*n*)∗ i) − *exp* ((*pi*∗*k*/*n*)∗ i)) =
(*cos*(−(*pi*∗*k*/*n*)) + i ∗ *sin*(−(*pi*∗*k*/*n*))) −
(*cos*((*pi*∗*k*/*n*)) + i ∗ *sin*((*pi*∗*k*/*n*)))
using *1* **by** *argo*
also have ... = −*2*∗i∗ *sin*((*pi*∗*k*/*n*)) **by** *simp*
finally show *?thesis* **by** *blast*
qed

have $norm((exp\ (-(x*pi*k/n)*\ \mathrm{i})\ -\ exp\ ((x*pi*k/n)*\ \mathrm{i}))\ div$
$(exp\ (-(pi*k/n)*\ \mathrm{i})\ -\ exp\ ((pi*k/n)*\ \mathrm{i}))) =$
$norm((-2*\mathrm{i}*\ sin((x*pi*k/n)))\ div\ (-2*\mathrm{i}*\ sin((pi*k/n))))$
using *num den* **by** *presburger*
also have $\ldots = norm(sin((x*pi*k/n))\ div\ sin((pi*k/n)))$
by (*simp add*: *norm-divide*)
finally show *?thesis* **by** *blast*
qed
also have $\ldots = norm((sin\ (x*pi*k/n)))\ div\ norm((sin\ (pi*k/n)))$
by (*simp add*: *norm-divide*)
also have $\ldots \leq 1\ div\ norm((sin\ (pi*k/n)))$
proof −
have $norm((sin\ (pi*k/n))) \geq 0$ **by** *simp*
have $norm\ (sin\ (x*pi*k/n)) \leq 1$ **by** *simp*
then show *?thesis*
using *divide-right-mono*[*OF* ‹$norm\ (sin\ (x*pi*k/n)) \leq 1$› ‹$norm((sin\ (pi*k/n))) \geq 0$›]
by *blast*
qed
finally have *26*: $norm(f(k)) \leq 1\ div\ norm((sin\ (pi*k/n)))$
by *blast*

{
fix *t*
assume $t \geq 0\ t \leq pi\ div\ 2$
then have $t \in \{0..pi\ div\ 2\}$ **by** *auto*
have *convex-on* $\{0..pi/2\}\ (\lambda x.\ -sin\ x)$
by (*rule convex-on-realI*[**where** $f' = \lambda x.\ -\ cos\ x$])
(*auto intro*!: *derivative-eq-intros simp*: *cos-monotone-0-pi-le*)
from *convex-onD-Icc'*[*OF this* ‹$t \in \{0..pi\ div\ 2\}$›] **have** $sin(t) \geq (2\ div\ pi)*t$ **by** *simp*
}
note *sin-ineq* = *this*

have *sin-ineq-inst*: $sin\ ((pi*k)\ /\ n) \geq (2 * k)\ /\ n$
proof −
have $pi\ /\ n \geq 0$ **by** *simp*
have *1*: $(pi*k)\ /\ n \geq 0$ **using** ‹$1 \leq k$› **by** *auto*
have $(pi*k)/n = (pi\ /\ n) * k$ **by** *simp*
also have $\ldots \leq (pi\ /\ n) * (n\ /\ 2)$
using *mult-left-mono*[*of k n / 2 pi / n*]
‹$k \leq n\ div\ 2$› ‹$0 \leq pi\ /\ real\ n$› **by** *linarith*
also have $\ldots \leq pi\ /\ 2$
by (*simp add*: *divide-simps*)
finally have *2*: $(pi*k)/n \leq pi\ /\ 2$ **by** *auto*

have $(2\ /\ pi) * (pi * k\ /\ n) \leq sin((pi * k)\ /\ n)$
using *sin-ineq*[*OF 1 2*] **by** *blast*

```
      then show sin((pi ∗ k) / n) ≥ (2∗k) / n
        by auto
    qed

    from 26 have norm(f(k)) ≤ 1 div abs((sin (pi∗k/n))) by simp
    also have ... ≤ 1 / abs((2∗k) / n)
    proof −
      have sin (pi∗k/n) ≥ (2∗k) / n using sin-ineq-inst by simp
      moreover have (2∗k) / n > 0 using n ‹1 ≤ k› by auto
      ultimately have abs((sin (pi∗k/n))) ≥ abs((2∗k)/n) by auto
      have abs((2∗k)/n) > 0 using ‹(2∗k)/n > 0› by linarith
      then show 1 div abs((sin (pi∗k/n))) ≤ 1 / abs(((2∗k)/n))
        using ‹abs((2∗k)/n) > 0› ‹abs((sin (pi∗k/n))) ≥ abs(((2∗k)/n))›
        by (intro frac-le) auto
    qed
    also have ... = n / (2∗k) using ‹k ≥ 1› by simp
    finally have norm(f(k)) ≤ n / (2∗k) by blast
  }
  note ineq = this


  have sqrt n ∗ norm (sum χ {1..x}) < n ∗ ln n
  proof (cases even n)
    case True
    have norm (f(n div 2)) ≤ 1
    proof −
      have int (n div 2) ≥ 1 using n ‹even n› by auto
      show ?thesis
        using ineq[OF ‹int (n div 2) ≥ 1›] True n by force
    qed
    from 24 have sqrt n ∗ norm (sum χ {1..x})
           ≤ (∑ k = 1..<n. norm (f (int k))) by blast
    also have ... = (∑ k = 1..n−1. norm (f (int k)))
      by (intro sum.cong) auto
    also have ... ≤ 2 ∗ (∑ k = 1..(n − 2) div 2. norm (f (int k))) + norm(f(n
div 2))
      using 25(2)[OF True] by blast
    also have ... ≤  real n ∗ (∑ k = 1..(n − 2) div 2. 1 / k) + norm(f(n div 2))
    proof −
      have (∑ k = 1..(n − 2) div 2. norm (f (int k))) ≤ (∑ k = 1..(n − 2) div
2. real n div (2∗k))
      proof (rule sum-mono)
        fix k
        assume k ∈ {1..(n − 2) div 2}
        then have 1 ≤ int k int k ≤ n div 2 by auto
        show norm (f (int k)) ≤ real n / (2∗k)
          using ineq[OF ‹1 ≤ int k› ‹int k ≤ n div 2›] by auto
      qed
      also have ... = (∑ k = 1..(n − 2) div 2. (real n div 2) ∗ (1 / k))
```

by (*rule sum.cong,auto*)
also have ... = (*real n div 2*) ∗ (∑ *k* = *1*..(*n* − *2*) *div 2*. *1* / *k*)
using *sum-distrib-left*[*symmetric*] by *fast*
finally have (∑ *k* = *1*..(*n* − *2*) *div 2*. *norm* (*f* (*int k*))) ≤
(*real n div 2*) ∗ (∑ *k* = *1*..(*n* − *2*) *div 2*. *1* / *k*)
by *blast*
then show *?thesis* by *argo*
qed
also have ... = *real n* ∗ *harm* ((*n* − *2*) *div 2*) + *norm*(*f*(*n div 2*))
unfolding *harm-def inverse-eq-divide* by *simp*
also have ... < *n* ∗ *ln n*
proof (*cases n* = *2*)
case *True*
have *real n* ∗ *harm* ((*n* − *2*) *div 2*) + *norm* (*f* (*int* (*n div 2*))) ≤ *1*
using ‹*n* = *2*› ‹*norm* (*f* (*int* (*n div 2*))) ≤ *1*›
unfolding *harm-def* by *simp*
moreover have *real n* ∗ *ln* (*real n*) ≥ *4* / *3*
using ‹*n* = *2*› *ln2-ge-two-thirds* by *auto*
ultimately show *?thesis* by *argo*
next
case *False*
have *n* > *3* using *n* ‹*n* ≠ *2*› ‹*even n*› by *auto*
then have (*n*−*2*) *div 2* > *0* by *simp*
then have *harm* ((*n* − *2*) *div 2*) < *ln* (*real* (*2* ∗ ((*n* − *2*) *div 2*) + *1*))
using *harm-less-ln* by *blast*
also have ... = *ln* (*real* (*n* − *1*))
using ‹*even n*› ‹*n* > *3*› by *simp*
finally have *1*: *harm* ((*n* − *2*) *div 2*) < *ln* (*real* (*n* − *1*))
by *blast*
then have *real n* ∗ *harm* ((*n* − *2*) *div 2*) < *real n* ∗ *ln* (*real* (*n* − *1*))
using *n* by *simp*
then have *real n* ∗ *harm* ((*n* − *2*) *div 2*) + *norm* (*f* (*int* (*n div 2*)))
< *real n* ∗ *ln* (*real* (*n* − *1*)) + *1*
using ‹*norm* (*f* (*int* (*n div 2*))) ≤ *1*› by *argo*
also have ... = *real n* ∗ *ln* (*real* (*n* − *1*)) + *real n* ∗ *1* / *real n*
using *n* by *auto*
also have ... < *real n* ∗ *ln* (*real* (*n* − *1*)) + *real n* ∗ *ln* (*1* + *1* / (*real n* −
1))
proof −
have *real n* > *1* *real n* > *0* using *n* by *simp*+
then have *real n* ∗ (*1* / *real n*) < *real n* ∗ *ln* (*1* + *1* / (*real n* − *1*))
by (*intro mult-strict-left-mono harm-aux-ineq-1*) *auto*
then show *?thesis* by *auto*
qed
also have ... = *real n* ∗ (*ln* (*real* (*n* − *1*)) + *ln* (*1* + *1* / (*real n* − *1*)))
by *argo*
also have ... = *real n* ∗ (*ln* (*real* (*n* − *1*) ∗ (*1* + *1* / (*real n* − *1*))))
proof −
have *real* (*n* − *1*) > *0* *1* + *1* / (*real n* − *1*) > *0*

using n **by** (*auto simp add*: *add-pos-nonneg*)
then show *?thesis*
by (*simp add*: *ln-mult*)
qed
also have $\ldots = real\ n * ln\ n$
using n **by** (*auto simp add*: *divide-simps*)
finally show *?thesis* **by** *blast*
qed
finally show *?thesis* **by** *blast*
next
case *False*
from *24* **have** $sqrt\ n * norm\ (sum\ \chi\ \{1..x\}) \leq (\sum k{=}\ 1..{<}n.\ norm\ (f\ (int\ k)))$
by *blast*
also have $\ldots = (\sum k{=}\ 1..n{-}1.\ norm\ (f\ (int\ k)))$
by (*intro sum.cong*) *auto*
also have $\ldots \leq 2 * (\sum k = 1..(n - 1)\ div\ 2.\ norm\ (f\ (int\ k)))$
using *25(1)*[*OF False*] **by** *blast*
also have $\ldots \leq real\ n * (\sum k = 1..(n - 1)\ div\ 2.\ 1\ /\ k)$
proof −
have $(\sum k = 1..(n - 1)\ div\ 2.\ norm\ (f\ (int\ k))) \leq (\sum k = 1..(n - 1)\ div\ 2.\ real\ n\ div\ (2{*}k))$
proof (*rule sum-mono*)
fix k
assume $k \in \{1..(n - 1)\ div\ 2\}$
then have $1 \leq int\ k\ int\ k \leq n\ div\ 2$ **by** *auto*
show $norm\ (f\ (int\ k)) \leq real\ n\ /\ (2{*}k)$
using *ineq*[*OF* ‹$1 \leq int\ k$› ‹$int\ k \leq n\ div\ 2$›] **by** *auto*
qed
also have $\ldots = (\sum k = 1..(n - 1)\ div\ 2.\ (n\ /\ 2) * (1\ /\ k))$
by (*rule sum.cong*,*auto*)
also have $\ldots = (n\ /\ 2) * (\sum k = 1..(n - 1)\ div\ 2.\ 1\ /\ k)$
using *sum-distrib-left*[*symmetric*] **by** *fast*
finally have $(\sum k = 1..(n - 1)\ div\ 2.\ norm\ (f\ (int\ k))) \leq$
$(real\ n\ div\ 2) * (\sum k = 1..(n - 1)\ div\ 2.\ 1\ /\ k)$
by *blast*
then show *?thesis* **by** *argo*
qed
also have $\ldots = real\ n * harm\ ((n - 1)\ div\ 2)$
unfolding *harm-def inverse-eq-divide* **by** *simp*
also have $\ldots < n * ln\ n$
proof −
have $n > 2$ **using** n ‹*odd n*› **by** *presburger*
then have $(n{-}1)\ div\ 2 > 0$ **by** *auto*
then have $harm\ ((n - 1)\ div\ 2) < ln\ (real\ (2 * ((n - 1)\ div\ 2) + 1))$
using *harm-less-ln* **by** *blast*
also have $\ldots = ln\ (real\ n)$ **using** ‹*odd n*› **by** *simp*
finally show *?thesis* **using** n **by** *simp*
qed

finally show *?thesis* **by** *blast*
qed

then have *1*: *sqrt n* $*$ *norm* (*sum* χ {*1..x*}) < *n* $*$ *ln n*
by *blast*
show *norm* (*sum* χ {*1..x*}) < *sqrt n* $*$ *ln n*
proof −
have *2*: *norm* (*sum* χ {*1..x*}) $*$ *sqrt n* < *n* $*$ *ln n*
using *1* **by** *argo*
have *sqrt n* > *0* **using** *n* **by** *simp*
have *3*: (*n* $*$ *ln n*) / *sqrt n* = *sqrt n* $*$ *ln n*
using *n* **by** (*simp add*: *field-simps*)
show *norm* (*sum* χ {*1..x*}) < *sqrt n* $*$ *ln n*
using *mult-imp-less-div-pos*[*OF* ‹*sqrt n* > *0*› *2*] *3* **by** *argo*
qed
qed

8.2 General case

We now first prove the inequality for the general case in terms of the divisor function:

theorem (**in** *dcharacter*) *polya-vinogradov-inequality-explicit*:
assumes *nonprincipal*: $\chi \neq$ *principal-dchar n*
shows *norm* (*sum* χ {*1..x*}) < *sqrt conductor* $*$ *ln conductor* $*$ *divisor-count* (*n div conductor*)
proof −
write *primitive-extension* (‹Φ›)
write *conductor* (‹*c*›)
interpret Φ: *primitive-dchar c residue-mult-group c primitive-extension*
using *primitive-primitive-extension nonprincipal* **by** *metis*

have $*$: *k* $\leq$ *x div b* $\longleftrightarrow$ *b* $*$ *k* $\leq$ *x* **if** *b* > *0* **for** *b k*
by (*metis that antisym-conv div-le-mono div-mult-self1-is-m less-imp-le not-less times-div-less-eq-dividend*)
have $**$: *a* > *0* **if** *a dvd n* **for** *a*
using *n that* **by** (*auto intro*!: *Nat.gr0I*)

from *nonprincipal* **have** ($\sum$ *m=1..x*. χ *m*) = ($\sum$ *m* | *m* $\in$ {*1..x*} $\wedge$ *coprime m n*. Φ *m*)
by (*intro sum.mono-neutral-cong-right*) (*auto simp*: *eq-zero-iff principal-decomposition*)
also have ... = ($\sum$ *m=1..x*. Φ *m* $*$ ($\sum$ *d* | *d dvd gcd m n*. *moebius-mu d*))
by (*subst sum-moebius-mu-divisors'*, *intro sum.mono-neutral-cong-left*)
(*auto simp*: *coprime-iff-gcd-eq-1 simp del*: *coprime-imp-gcd-eq-1*)
also have ... = ($\sum$ *m=1..x*. $\sum$ *d* | *d dvd gcd m n*. Φ *m* $*$ *moebius-mu d*)
by (*simp add*: *sum-distrib-left*)
also have ... = ($\sum$ *m=1..x*. $\sum$ *d* | *d dvd m* $\wedge$ *d dvd n*. Φ *m* $*$ *moebius-mu d*)
by (*intro sum.cong*) *auto*
also have ... = ($\sum$(*m*, *d*)$\in$(*SIGMA m*:{*1..x*}. {*d*. *d dvd m* $\wedge$ *d dvd n*}). Φ *m* $*$ *moebius-mu d*)

using *n* **by** (*subst sum.Sigma*) *auto*
also have ... = (∑(*d*, *q*)∈(*SIGMA d*:{*d*. *d dvd n*}. {*1..x div d*}). *moebius-mu d* ∗ Φ (*d* ∗ *q*))
by (*intro sum.reindex-bij-witness*[*of* - λ(*d*,*q*). (*d* ∗ *q*, *d*) λ(*m*,*d*). (*d*, *m div d*)])
(*auto simp*: ∗ ∗∗ *Suc-le-eq*)
also have ... = (∑*d* | *d dvd n*. *moebius-mu d* ∗ Φ *d* ∗ (∑*q*=*1..x div d*. Φ *q*))
using *n* **by** (*subst sum.Sigma* [*symmetric*]) (*auto simp*: *sum-distrib-left mult.assoc*)
finally have *eq*: (∑*m*=*1..x*. χ *m*) = ...

have *norm* (∑*m*=*1..x*. χ *m*) ≤
(∑*d* | *d dvd n*. *norm* (*moebius-mu d* ∗ Φ *d*) ∗ *norm* (∑*q*=*1..x div d*. Φ *q*))
unfolding *eq* **by** (*intro sum-norm-le*) (*simp add*: *norm-mult*)
also have ... < (∑*d* | *d dvd n*. *norm* (*moebius-mu d* ∗ Φ *d*) ∗ (*sqrt c* ∗ *ln c*))
(**is** *sum ?lhs* - < *sum ?rhs* -)
proof (*rule sum-strict-mono-ex1*)
show ∀ *d*∈{*d*. *d dvd n*}. *?lhs d* ≤ *?rhs d*
by (*intro ballI mult-left-mono less-imp-le*[*OF* Φ.*polya-vinogradov-inequality-primitive*]) *auto*
show ∃ *d*∈{*d*. *d dvd n*}. *?lhs d* < *?rhs d*
by (*intro bexI*[*of* - *1*] *mult-strict-left-mono* Φ.*polya-vinogradov-inequality-primitive*) *auto*
qed (*use n* **in** *auto*)
also have ... = *sqrt c* ∗ *ln c* ∗ (∑*d* | *d dvd n*. *norm* (*moebius-mu d* ∗ Φ *d*))
by (*simp add*: *sum-distrib-left sum-distrib-right mult-ac*)
also have (∑*d* | *d dvd n*. *norm* (*moebius-mu d* ∗ Φ *d*)) =
(∑*d* | *d dvd n* ∧ *squarefree d* ∧ *coprime d c*. *1*)
using *n* **by** (*intro sum.mono-neutral-cong-right*)
(*auto simp*: *moebius-mu-def* Φ.*eq-zero-iff norm-mult norm-power* Φ.*norm*)
also have ... = *card* {*d*. *d dvd n* ∧ *squarefree d* ∧ *coprime d c*}
by *simp*
also have *card* {*d*. *d dvd n* ∧ *squarefree d* ∧ *coprime d c*} ≤ *card* {*d*. *d dvd* (*n div c*)}
proof (*intro card-mono*; *safe?*)
show *finite* {*d*. *d dvd* (*n div c*)}
using *dvd-div-eq-0-iff*[*of c n*] *n conductor-dvd* **by** (*intro finite-divisors-nat*) *auto*
next
fix *d* **assume** *d*: *d dvd n squarefree d coprime d c*
hence *d* > *0* **by** (*intro Nat.gr0I*) *auto*
show *d dvd* (*n div c*)
proof (*rule multiplicity-le-imp-dvd*)
fix *p* :: *nat* **assume** *p*: *prime p*
show *multiplicity p d* ≤ *multiplicity p* (*n div c*)
proof (*cases p dvd d*)
assume *p dvd d*
with *d* ‹*d* > *0*› *p* **have** *multiplicity p d* = *1*
by (*auto simp*: *squarefree-factorial-semiring′ in-prime-factors-iff*)

moreover have *p dvd* (*n div c*)
proof −
have *p dvd c* ∗ (*n div c*)
using ‹*p dvd d*› ‹*d dvd n*› *conductor-dvd* **by** *auto*
moreover have ¬(*p dvd c*)
using *d p* ‹*p dvd d*› *coprime-common-divisor not-prime-unit* **by** *blast*
ultimately show *p dvd* (*n div c*)
using *p prime-dvd-mult-iff* **by** *blast*
qed
hence *multiplicity p* (*n div c*) ≥ *1*
using *n p conductor-dvd dvd-div-eq-0-iff*[*of c n*]
by (*intro multiplicity-geI*) (*auto intro*: *Nat.gr0I*)
ultimately show *?thesis* **by** *simp*
qed (*auto simp*: *not-dvd-imp-multiplicity-0*)
qed (*use* ‹*d* > *0*› **in** *simp-all*)
qed
also have *card* {*d. d dvd* (*n div c*)} = *divisor-count* (*n div c*)
by (*simp add*: *divisor-count-def*)
finally show *norm* (*sum* χ {*1..x*}) < *sqrt c* ∗ *ln c* ∗ *divisor-count* (*n div c*)
using *conductor-gr-0* **by** (*simp add*: *mult-left-mono*)
qed

Next, we obtain a suitable upper bound on the number of divisors of *n*:

lemma *divisor-count-upper-bound-aux*:
fixes *n* :: *nat*
shows *divisor-count n* ≤ *2* ∗ *card* {*d. d dvd n* ∧ *d* ≤ *sqrt n*}
proof (*cases n* = *0*)
case *False*
hence *n*: *n* > *0* **by** *simp*
have ∗: *x* > *0* **if** *x dvd n* **for** *x*
using *that n* **by** (*auto intro*!: *Nat.gr0I*)
have ∗∗: *real n* = *sqrt* (*real n*) ∗ *sqrt* (*real n*)
by *simp*
have ∗∗∗: *n* < *x* ∗ *sqrt n* ⟷ *sqrt n* < *x x* ∗ *sqrt n* < *n* ⟷ *x* < *sqrt n* **for** *x*
by (*metis* ∗∗ *n of-nat-0-less-iff mult-less-cancel-right-pos real-sqrt-gt-0-iff*)+

have *divisor-count n* = *card* {*d. d dvd n*}
by (*simp add*: *divisor-count-def*)
also have {*d. d dvd n*} = {*d. d dvd n* ∧ *d* ≤ *sqrt n*} ∪ {*d. d dvd n* ∧ *d* > *sqrt n*}
by *auto*
also have *card* ... = *card* {*d. d dvd n* ∧ *d* ≤ *sqrt n*} + *card* {*d. d dvd n* ∧ *d* > *sqrt n*}
using *n* **by** (*subst card-Un-disjoint*) *auto*
also have *bij-betw* (λ*d. n div d*) {*d. d dvd n* ∧ *d* > *sqrt n*} {*d. d dvd n* ∧ *d* < *sqrt n*}
using *n* **by** (*intro bij-betwI*[*of - - - λd. n div d*])
(*auto simp*: *Real.real-of-nat-div real-sqrt-divide field-simps* ∗ ∗∗∗)
hence *card* {*d. d dvd n* ∧ *d* > *sqrt n*} = *card* {*d. d dvd n* ∧ *d* < *sqrt n*}

by (*rule bij-betw-same-card*)
also have ... $\leq$ *card* {*d. d dvd n* $\wedge$ *d* $\leq$ *sqrt n*}
using *n* **by** (*intro card-mono*) *auto*
finally show *divisor-count n* $\leq$ *2* $*$ ... **by** *simp*
qed *auto*

lemma *divisor-count-upper-bound*:
fixes *n* :: *nat*
shows *divisor-count n* $\leq$ *2* $*$ *nat* $\lfloor$*sqrt n*$\rfloor$
proof (*cases n* = *0*)
case *False*
have *divisor-count n* $\leq$ *2* $*$ *card* {*d. d dvd n* $\wedge$ *d* $\leq$ *sqrt n*}
by (*rule divisor-count-upper-bound-aux*)
also have *card* {*d. d dvd n* $\wedge$ *d* $\leq$ *sqrt n*} $\leq$ *card* {*1..nat* $\lfloor$*sqrt n*$\rfloor$}
using *False* **by** (*intro card-mono*) (*auto simp*: *le-nat-iff le-floor-iff Suc-le-eq intro*!: *Nat.gr0I*)
also have ... = *nat* $\lfloor$*sqrt n*$\rfloor$ **by** *simp*
finally show *?thesis* **by** *simp*
qed *auto*

lemma *divisor-count-upper-bound'*:
fixes *n* :: *nat*
shows *real* (*divisor-count n*) $\leq$ *2* $*$ *sqrt n*
proof −
have *real* (*divisor-count n*) $\leq$ *2* $*$ *real* (*nat* $\lfloor$*sqrt n*$\rfloor$)
using *divisor-count-upper-bound*[*of n*] **by** *linarith*
also have ... $\leq$ *2* $*$ *sqrt n*
by *simp*
finally show *?thesis* **.**
qed

We are now ready to prove the 'regular' Pólya–Vinogradov inequality.

Apostol formulates it in the following way (Theorem 13.15, notation adapted): 'If χ is any nonprincipal character mod n, then for all $x \geq 2$ we have $\sum_{m\leq x} \chi(m) = O(\sqrt{n}\log n)$.'

The precondition $x \geq 2$ here is completely unnecessary. The 'Big-O' notation is somewhat problematic since it does not make explicit in what way the variables are quantified (in particular the x and the χ). The statement of the theorem in this way (for a fixed character χ) seems to suggest that n is fixed here, which would make the use of 'Big-O' completely vacuous, since it is an asymptotic statement about n.

We therefore decided to formulate the inequality in the following more explicit way, even giving an explicit constant factor:

theorem (**in** *dcharacter*) *polya-vinogradov-inequality*:
assumes *nonprincipal*: $\chi \neq$ *principal-dchar n*
shows *norm* ($\sum$ *m*=*1..x.* χ *m*) < *2* $*$ *sqrt n* $*$ *ln n*
proof −

have *n div conductor* > *0*
using *n conductor-dvd dvd-div-eq-0-iff* [*of conductor n*] **by** *auto*
have *norm* ($\sum$ *m=1..x.* χ *m*) < *sqrt conductor* ∗ *ln conductor* ∗ *divisor-count* (*n div conductor*)
using *nonprincipal* **by** (*rule polya-vinogradov-inequality-explicit*)
also have ... ≤ *sqrt conductor* ∗ *ln conductor* ∗ (*2* ∗ *sqrt* (*n div conductor*))
using *conductor-gr-0* ‹*n div conductor* > *0*›
by (*intro mult-left-mono divisor-count-upper-bound′*) (*auto simp*: *Suc-le-eq*)
also have *sqrt* (*n div conductor*) = *sqrt n* / *sqrt conductor*
using *conductor-dvd* **by** (*simp add*: *Real.real-of-nat-div real-sqrt-divide*)
also have *sqrt conductor* ∗ *ln conductor* ∗ (*2* ∗ (*sqrt n* / *sqrt conductor*)) = *2* ∗ *sqrt n* ∗ *ln conductor*
using *conductor-gr-0 n* **by** (*simp add*: *algebra-simps*)
also have ... ≤ *2* ∗ *sqrt n* ∗ *ln n*
using *conductor-le-modulus conductor-gr-0* **by** (*intro mult-left-mono*) *auto*
finally show *?thesis* **.**
qed

unbundle *vec-lambda-syntax*

end

References

[1] T. M. Apostol. *Introduction to Analytic Number Theory.* Undergraduate Texts in Mathematics. Springer-Verlag, 1976.