Dirichlet Series

Manuel Eberl

March 17, 2025

Abstract

This entry is a formalisation of much of Chapters 2, 3, and 11 of Apostol's "Introduction to Analytic Number Theory" [1]. This includes:

- Definitions and basic properties for several number-theoretic functions (Euler's φ, Möbius μ, Liouville's λ, the divisor function σ, von Mangoldt's Λ)
- Executable code for most of these functions, the most efficient implementations using the factoring algorithm by Thiemann *et al.*
- Dirichlet products and formal Dirichlet series
- Analytic results connecting convergent formal Dirichlet series to complex functions
- Euler product expansions
- Asymptotic estimates of number-theoretic functions including the density of squarefree integers and the average number of divisors of a natural number

These results are useful as a basis for developing more number-theoretic results, such as the Prime Number Theorem.

Contents

1 Miscellaneous auxiliary facts

theory *Dirichlet-Misc*
  imports
    HOL−Number-Theory.Number-Theory
begin

lemma
  fixes $a\ k :: nat$
  assumes $a > 1\ k > 0$
  shows *geometric-sum-nat-aux*: $(a - 1) * (\sum i<k.\ a \mathbin{\char`\^} i) = a \mathbin{\char`\^} k - 1$
    and *geometric-sum-nat-dvd*: $a - 1\ dvd\ a \mathbin{\char`\^} k - 1$
    and *geometric-sum-nat*: $(\sum i<k.\ a \mathbin{\char`\^} i) = (a \mathbin{\char`\^} k - 1)\ div\ (a - 1)$
⟨*proof*⟩

lemma *dvd-div-gt0*: $d\ dvd\ n \Longrightarrow n > 0 \Longrightarrow n\ div\ d > (0::nat)$
  ⟨*proof*⟩

lemma *Set-filter-insert*:
  $Set.filter\ P\ (insert\ x\ A) = (if\ P\ x\ then\ insert\ x\ (Set.filter\ P\ A)\ else\ Set.filter\ P\ A)$
  ⟨*proof*⟩

lemma *Set-filter-union*: $Set.filter\ P\ (A \cup B) = Set.filter\ P\ A \cup Set.filter\ P\ B$
  ⟨*proof*⟩

lemma *Set-filter-empty* [*simp*]: $Set.filter\ P\ \{\} = \{\}$
  ⟨*proof*⟩

lemma *Set-filter-image*: $Set.filter\ P\ (f\ `\ A) = f\ `\ Set.filter\ (P \circ f)\ A$
  ⟨*proof*⟩

lemma *Set-filter-cong* [*cong*]:
    $(\bigwedge x.\ x \in A \Longrightarrow P\ x \longleftrightarrow Q\ x) \Longrightarrow A = B \Longrightarrow Set.filter\ P\ A = Set.filter\ Q\ B$
  ⟨*proof*⟩

lemma *inj-on-insert′*: $(\bigwedge B.\ B \in A \Longrightarrow x \notin B) \Longrightarrow inj\text{-}on\ (insert\ x)\ A$
  ⟨*proof*⟩

lemma
  assumes $finite\ A\ A \neq \{\}$
  shows *card-even-subset-aux*: $card\ \{B.\ B \subseteq A \wedge even\ (card\ B)\} = 2 \mathbin{\char`\^} (card\ A - 1)$
    and *card-odd-subset-aux*: $card\ \{B.\ B \subseteq A \wedge odd\ (card\ B)\} = 2 \mathbin{\char`\^} (card\ A - 1)$
    and *card-even-odd-subset*: $card\ \{B.\ B \subseteq A \wedge even\ (card\ B)\} = card\ \{B.\ B \subseteq A \wedge odd\ (card\ B)\}$
⟨*proof*⟩

lemma *bij-betw-prod-divisors-coprime*:
assumes *coprime a* (*b* :: *nat*)
shows *bij-betw* ($\lambda x.$ *fst x* ∗ *snd x*) ({*d*. *d dvd a*} × {*d*. *d dvd b*}) {*k*. *k dvd a* ∗ *b*}
⟨*proof*⟩

lemma *bij-betw-prime-power-divisors*:
assumes *prime* (*p* :: *nat*)
shows *bij-betw* ((^) *p*) {..*k*} {*d*. *d dvd p* ^ *k*}
⟨*proof*⟩

lemma *sum-divisors-coprime-mult*:
assumes *coprime a* (*b* :: *nat*)
shows ($\sum$ *d* | *d dvd a* ∗ *b*. *f d*) = ($\sum$ *r* | *r dvd a*. $\sum$ *s* | *s dvd b*. *f* (*r* ∗ *s*))
⟨*proof*⟩

end

2 Multiplicative arithmetic functions

theory *Multiplicative-Function*
imports
HOL−Number-Theory.Number-Theory
Dirichlet-Misc
begin

2.1 Definition

locale *multiplicative-function* =
fixes *f* :: *nat* ⇒ *'a* :: *comm-semiring-1*
assumes *zero* [*simp*]: *f 0* = *0*
assumes *one* [*simp*]: *f 1* = *1*
assumes *mult-coprime-aux*: *a* > *1* ⟹ *b* > *1* ⟹ *coprime a b* ⟹ *f* (*a* ∗ *b*) = *f a* ∗ *f b*
begin

lemma *Suc-0* [*simp*]: *f* (*Suc 0*) = *1*
⟨*proof*⟩

lemma *mult-coprime*:
assumes *coprime a b*
shows *f* (*a* ∗ *b*) = *f a* ∗ *f b*
⟨*proof*⟩

lemma *prod-coprime*:
assumes $\bigwedge$*x y*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* ≠ *y* ⟹ *coprime* (*g x*) (*g y*)
shows *f* (*prod g A*) = ($\prod$ *x*∈*A*. *f* (*g x*))
⟨*proof*⟩

lemma *prod-prime-factors*:
  assumes $n > 0$
  shows $f\ n = (\prod p{\in}prime\text{-}factors\ n.\ f\ (p\ \hat{}\ multiplicity\ p\ n))$
$\langle proof \rangle$

lemma *multiplicative-sum-divisors*: $multiplicative\text{-}function\ (\lambda n.\ \sum d\ |\ d\ dvd\ n.\ f\ d)$
$\langle proof \rangle$

end

locale *multiplicative-function'* = *multiplicative-function f* **for** $f :: nat \Rightarrow 'a :: comm\text{-}semiring\text{-}1\ +$
  fixes $f\text{-}prime\text{-}power :: nat \Rightarrow nat \Rightarrow 'a$ **and** $f\text{-}prime :: nat \Rightarrow 'a$
  assumes *prime-power*: $prime\ p \Longrightarrow k > 0 \Longrightarrow f\ (p\ \hat{}\ k) = f\text{-}prime\text{-}power\ p\ k$
  assumes *prime-aux*: $prime\ p \Longrightarrow f\text{-}prime\text{-}power\ p\ 1 = f\text{-}prime\ p$
begin

lemma *prime*: $prime\ p \Longrightarrow f\ p = f\text{-}prime\ p$
  $\langle proof \rangle$

lemma *prod-prime-factors'*:
  assumes $n > 0$
  shows $f\ n = (\prod p{\in}prime\text{-}factors\ n.\ f\text{-}prime\text{-}power\ p\ (multiplicity\ p\ n))$
  $\langle proof \rangle$

lemma *efficient-code-aux*:
  assumes $n > 0$ $set\ ps = (\lambda p.\ (p,\ multiplicity\ p\ n - 1))\ \text{`}\ prime\text{-}factors\ n$ $distinct\ ps$
  shows $f\ n = (\prod (p,d) \leftarrow ps.\ f\text{-}prime\text{-}power\ p\ (Suc\ d))$
$\langle proof \rangle$

lemma *efficient-code*:
  assumes $set\ (ps\ ()) = (\lambda p.\ (p,\ multiplicity\ p\ n - 1))\ \text{`}\ prime\text{-}factors\ n$ $distinct\ (ps\ ())$
  shows $f\ n = (if\ n = 0\ then\ 0\ else\ (\prod (p,d) \leftarrow ps\ ().\ f\text{-}prime\text{-}power\ p\ (Suc\ d)))$
  $\langle proof \rangle$

end

locale *completely-multiplicative-function* =
  fixes $f :: nat \Rightarrow 'a :: comm\text{-}semiring\text{-}1$
  assumes *zero-aux*: $f\ 0 = 0$
  assumes *one-aux*: $f\ (Suc\ 0) = 1$
  assumes *mult-aux*: $a > 1 \Longrightarrow b > 1 \Longrightarrow f\ (a * b) = f\ a * f\ b$
begin

lemma *mult*: $f\ (a * b) = f\ a * f\ b$

⟨*proof*⟩

sublocale *multiplicative-function f*
⟨*proof*⟩

lemma *prod*: $f\ (prod\ g\ A) = (\prod x{\in}A.\ f\ (g\ x))$
⟨*proof*⟩

lemma *power*: $f\ (n\ \widehat{}\ m) = f\ n\ \widehat{}\ m$
⟨*proof*⟩

lemma *prod-prime-factors'*: $n > 0 \Longrightarrow f\ n = (\prod p{\in}prime\text{-}factors\ n.\ f\ p\ \widehat{}\ multiplicity\ p\ n)$
⟨*proof*⟩

end

locale *completely-multiplicative-function'* =
completely-multiplicative-function f **for** $f :: nat \Rightarrow 'a :: comm\text{-}semiring\text{-}1$ +
fixes $f\text{-}prime :: nat \Rightarrow 'a$
assumes *f-prime*: $prime\ p \Longrightarrow f\ p = f\text{-}prime\ p$
begin

lemma *prod-prime-factors''*: $n > 0 \Longrightarrow f\ n = (\prod p{\in}prime\text{-}factors\ n.\ f\text{-}prime\ p\ \widehat{}\ multiplicity\ p\ n)$
⟨*proof*⟩

lemma *efficient-code-aux*:
assumes $n > 0$ $set\ ps = (\lambda p.\ (p,\ multiplicity\ p\ n - 1))$ ‘ $prime\text{-}factors\ n$ $distinct\ ps$
shows $f\ n = (\prod (p,d) \leftarrow ps.\ f\text{-}prime\ p\ \widehat{}\ Suc\ d)$
⟨*proof*⟩

lemma *efficient-code*:
assumes $set\ (ps\ ()) = (\lambda p.\ (p,\ multiplicity\ p\ n - 1))$ ‘ $prime\text{-}factors\ n$ $distinct\ (ps\ ())$
shows $f\ n = (if\ n = 0\ then\ 0\ else\ (\prod (p,d) \leftarrow ps\ ().\ f\text{-}prime\ p\ \widehat{}\ Suc\ d))$
⟨*proof*⟩

end

lemma *multiplicative-function-eqI*:
assumes *multiplicative-function f multiplicative-function g*
assumes $\bigwedge p\ k.\ prime\ p \Longrightarrow k > 0 \Longrightarrow f\ (p\ \widehat{}\ k) = g\ (p\ \widehat{}\ k)$
shows $f\ n = g\ n$
⟨*proof*⟩

lemma *multiplicative-function-of-natI*:
$multiplicative\text{-}function\ f \Longrightarrow multiplicative\text{-}function\ (\lambda n.\ of\text{-}nat\ (f\ n))$

⟨*proof*⟩

lemma *multiplicative-function-of-natD*:
multiplicative-function (λ*n*. *of-nat* (*f n*) :: ′*a* :: {*ring-char-0*, *comm-semiring-1*}) ⟹
multiplicative-function f
⟨*proof*⟩

lemma *multiplicative-function-mult*:
assumes *multiplicative-function f* *multiplicative-function g*
shows *multiplicative-function* (λ*n*. *f n* ∗ *g n*)
⟨*proof*⟩

lemma *multiplicative-function-inverse*:
fixes *f* :: *nat* ⇒ ′*a* :: *field*
assumes *multiplicative-function f*
shows *multiplicative-function* (λ*n*. *inverse* (*f n*))
⟨*proof*⟩

lemma *multiplicative-function-divide*:
fixes *f* :: *nat* ⇒ ′*a* :: *field*
assumes *multiplicative-function f* *multiplicative-function g*
shows *multiplicative-function* (λ*n*. *f n* / *g n*)
⟨*proof*⟩

lemma *completely-multiplicative-function-mult*:
assumes *completely-multiplicative-function f* *completely-multiplicative-function g*
shows *completely-multiplicative-function* (λ*n*. *f n* ∗ *g n*)
⟨*proof*⟩

lemma *completely-multiplicative-function-inverse*:
fixes *f* :: *nat* ⇒ ′*a* :: *field*
assumes *completely-multiplicative-function f*
shows *completely-multiplicative-function* (λ*n*. *inverse* (*f n*))
⟨*proof*⟩

lemma *completely-multiplicative-function-divide*:
fixes *f* :: *nat* ⇒ ′*a* :: *field*
assumes *completely-multiplicative-function f* *completely-multiplicative-function g*
shows *completely-multiplicative-function* (λ*n*. *f n* / *g n*)
⟨*proof*⟩

lemma (**in** *multiplicative-function*) *completely-multiplicativeI*:
assumes ⋀*p k*. *prime p* ⟹ *k* > *0* ⟹ *f* (*p* ^ *k*) = *f p* ^ *k*
shows *completely-multiplicative-function f*
⟨*proof*⟩

2.2 Indicator function

definition *ind* :: $(nat \Rightarrow bool) \Rightarrow nat \Rightarrow 'a :: semiring\text{-}1$ **where**
  $ind\ P\ n = (if\ n > 0 \wedge P\ n\ then\ 1\ else\ 0)$

lemma *ind-0* [*simp*]: $ind\ P\ 0 = 0$ ⟨*proof*⟩

lemma *ind-nonzero*: $n > 0 \Longrightarrow ind\ P\ n = (if\ P\ n\ then\ 1\ else\ 0)$
  ⟨*proof*⟩

lemma *ind-True* [*simp*]: $P\ n \Longrightarrow n > 0 \Longrightarrow ind\ P\ n = 1$
  ⟨*proof*⟩

lemma *ind-False* [*simp*]: $\neg P\ n \Longrightarrow n > 0 \Longrightarrow ind\ P\ n = 0$
  ⟨*proof*⟩

lemma *ind-eq-1-iff*: $ind\ P\ n = 1 \longleftrightarrow n > 0 \wedge P\ n$
  ⟨*proof*⟩

lemma *ind-eq-0-iff*: $ind\ P\ n = 0 \longleftrightarrow n = 0 \vee \neg P\ n$
  ⟨*proof*⟩

lemma *multiplicative-function-ind* [*intro?*]:
  assumes $P\ 1\ \bigwedge a\ b.\ a > 1 \Longrightarrow b > 1 \Longrightarrow coprime\ a\ b \Longrightarrow P\ (a * b) \longleftrightarrow P\ a \wedge P\ b$
  shows *multiplicative-function* (*ind P*)
  ⟨*proof*⟩

end

3 Dirichlet convolution

theory *Dirichlet-Product*
  imports
    Complex-Main
    Multiplicative-Function
begin

lemma *sum-coprime-dvd-cong*:
  $(\sum r \mid r\ dvd\ a.\ \sum s \mid s\ dvd\ b.\ f\ r\ s) = (\sum r \mid r\ dvd\ a.\ \sum s \mid s\ dvd\ b.\ g\ r\ s)$
  if $coprime\ a\ b\ \bigwedge r\ s.\ coprime\ r\ s \Longrightarrow r\ dvd\ a \Longrightarrow s\ dvd\ b \Longrightarrow f\ r\ s = g\ r\ s$
⟨*proof*⟩

definition *dirichlet-prod* :: $(nat \Rightarrow 'a :: semiring\text{-}0) \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a$ **where**
  $dirichlet\text{-}prod\ f\ g = (\lambda n.\ \sum d \mid d\ dvd\ n.\ f\ d * g\ (n\ div\ d))$

lemma *sum-divisors-code*:
  assumes $n > (0::nat)$

shows $(\sum d \mid d\ dvd\ n.\ f\ d) =$
$fold\text{-}atLeastAtMost\text{-}nat\ (\lambda d\ acc.\ if\ d\ dvd\ n\ then\ f\ d + acc\ else\ acc)\ 1\ n\ 0$
⟨*proof*⟩

lemma *dirichlet-prod-code* [*code*]:
$dirichlet\text{-}prod\ f\ g\ n = (if\ n = 0\ then\ 0\ else$
$fold\text{-}atLeastAtMost\text{-}nat\ (\lambda d\ acc.\ if\ d\ dvd\ n\ then\ f\ d * g\ (n\ div\ d) + acc\ else\ acc)\ 1\ n\ 0)$
⟨*proof*⟩

lemma *dirichlet-prod-0* [*simp*]: $dirichlet\text{-}prod\ f\ g\ 0 = 0$
⟨*proof*⟩

lemma *dirichlet-prod-Suc-0* [*simp*]: $dirichlet\text{-}prod\ f\ g\ (Suc\ 0) = f\ (Suc\ 0) * g\ (Suc\ 0)$
⟨*proof*⟩

lemma *dirichlet-prod-cong* [*cong*]:
assumes $(\bigwedge n.\ n > 0 \Longrightarrow f\ n = f'\ n)\ (\bigwedge n.\ n > 0 \Longrightarrow g\ n = g'\ n)$
shows $dirichlet\text{-}prod\ f\ g = dirichlet\text{-}prod\ f'\ g'$
⟨*proof*⟩

lemma *dirichlet-prod-altdef1*:
$dirichlet\text{-}prod\ f\ g = (\lambda n.\ \sum d \mid d\ dvd\ n.\ f\ (n\ div\ d) * g\ d)$
⟨*proof*⟩

lemma *dirichlet-prod-altdef2*:
$dirichlet\text{-}prod\ f\ g = (\lambda n.\ \sum (r,d) \mid r * d = n.\ f\ r * g\ d)$
⟨*proof*⟩

lemma *dirichlet-prod-commutes*:
$dirichlet\text{-}prod\ (f :: nat \Rightarrow 'a :: comm\text{-}semiring\text{-}0)\ g = dirichlet\text{-}prod\ g\ f$
⟨*proof*⟩

lemma *finite-divisors-nat'*: $n > (0 :: nat) \Longrightarrow finite\ \{(a,b).\ a * b = n\}$
⟨*proof*⟩

lemma *dirichlet-prod-assoc-aux1*:
assumes $n > 0$
shows $dirichlet\text{-}prod\ f\ (dirichlet\text{-}prod\ g\ h)\ n =$
$(\sum (a, b, c)\in\{(a, b, c).\ a * b * c = n\}.\ f\ a * g\ b * h\ c)$
⟨*proof*⟩

lemma *dirichlet-prod-assoc-aux2*:
assumes $n > 0$
shows $dirichlet\text{-}prod\ (dirichlet\text{-}prod\ f\ g)\ h\ n =$
$(\sum (a, b, c)\in\{(a, b, c).\ a * b * c = n\}.\ f\ a * g\ b * h\ c)$
⟨*proof*⟩

lemma *dirichlet-prod-assoc*:
  dirichlet-prod (*dirichlet-prod f g*) *h* = *dirichlet-prod f* (*dirichlet-prod g h*)
⟨*proof*⟩

lemma *dirichlet-prod-const-right* [*simp*]:
  assumes $n > 0$
  shows *dirichlet-prod f* (λ*n*. *if n* = *Suc 0 then c else 0*) *n* = *f n* ∗ *c*
⟨*proof*⟩

lemma *dirichlet-prod-const-left* [*simp*]:
  assumes $n > 0$
  shows *dirichlet-prod* (λ*n*. *if n* = *Suc 0 then c else 0*) *g n* = *c* ∗ *g n*
⟨*proof*⟩

fun *dirichlet-inverse* :: (*nat* ⇒ *'a* :: *comm-ring-1*) ⇒ *'a* ⇒ *nat* ⇒ *'a* **where**
  dirichlet-inverse f i n =
    (*if n* = *0 then 0 else if n* = *1 then i*
     else −*i* ∗ (∑ *d* | *d dvd n* ∧ *d* < *n*. *f* (*n div d*) ∗ *dirichlet-inverse f i d*))

lemma *dirichlet-inverse-induct* [*case-names 0 1 gt1*]:
  P 0 ⟹ *P* (*Suc 0*) ⟹ (⋀*n*. *n* > *1* ⟹ (⋀*k*. *k* < *n* ⟹ *P k*) ⟹ *P n*) ⟹ *P n*
  ⟨*proof*⟩

lemma *dirichlet-inverse-0* [*simp*]: *dirichlet-inverse f i 0* = *0*
  ⟨*proof*⟩

lemma *dirichlet-inverse-Suc-0* [*simp*]: *dirichlet-inverse f i* (*Suc 0*) = *i*
  ⟨*proof*⟩

declare *dirichlet-inverse.simps* [*simp del*]

lemma *dirichlet-inverse-gt-1*:
  n > *1* ⟹ *dirichlet-inverse f i n* =
    −*i* ∗ (∑ *d* | *d dvd n* ∧ *d* < *n*. *f* (*n div d*) ∗ *dirichlet-inverse f i d*)
  ⟨*proof*⟩

lemma *dirichlet-inverse-cong* [*cong*]:
  assumes ⋀*n*. *n* > *0* ⟹ *f n* = *f' n* *i* = *i'* *n* = *n'*
  shows *dirichlet-inverse f i n* = *dirichlet-inverse f' i' n'*
⟨*proof*⟩

lemma *dirichlet-inverse-gt-1'*:
  assumes *n* > *1*
  shows *dirichlet-inverse f i n* =
       −*i* ∗ *dirichlet-prod* (λ*n*. *if n* = *1 then 0 else f n*) (*dirichlet-inverse f i*) *n*
⟨*proof*⟩

lemma *of-int-dirichlet-prod*:

of-int (*dirichlet-prod f g n*) = *dirichlet-prod* (λ*n*. *of-int* (*f n*)) (λ*n*. *of-int* (*g n*)) *n*
⟨*proof*⟩

lemma *of-int-dirichlet-inverse*:
of-int (*dirichlet-inverse f i n*) = *dirichlet-inverse* (λ*n*. *of-int* (*f n*)) (*of-int i*) *n*
⟨*proof*⟩

lemma *dirichlet-inverse-code* [*code*]:
dirichlet-inverse f i n = (*if n* = *0 then 0 else if n* = *1 then i else*
−*i* ∗ *fold-atLeastAtMost-nat* (λ*d acc*. *if d dvd n then f* (*n div d*) ∗
dirichlet-inverse f i d + *acc else acc*) *1* (*n* − *1*) *0*)
⟨*proof*⟩

lemma *dirichlet-prod-inverse*:
assumes *f 1* ∗ *i* = *1*
shows *dirichlet-prod f* (*dirichlet-inverse f i*) = (λ*n*. *if n* = *1 then 1 else 0*)
⟨*proof*⟩

lemma *dirichlet-prod-inverse′*:
assumes *f 1* ∗ *i* = *1*
shows *dirichlet-prod* (*dirichlet-inverse f i*) *f* = (λ*n*. *if n* = *1 then 1 else 0*)
⟨*proof*⟩

lemma *dirichlet-inverse-noninvertible*:
assumes *f* (*Suc 0*) = (*0* :: ′*a* :: {*comm-ring-1*}) *i* = *0*
shows *dirichlet-inverse f i n* = *0*
⟨*proof*⟩

lemma *multiplicative-dirichlet-prod*:
assumes *multiplicative-function f*
assumes *multiplicative-function g*
shows *multiplicative-function* (*dirichlet-prod f g*)
⟨*proof*⟩

lemma *multiplicative-dirichlet-prodD1*:
fixes *f g* :: *nat* ⇒ ′*a* :: *comm-semiring-1-cancel*
assumes *multiplicative-function* (*dirichlet-prod f g*)
assumes *multiplicative-function f*
assumes [*simp*]: *g 0* = *0*
shows *multiplicative-function g*
⟨*proof*⟩

lemma *multiplicative-dirichlet-prodD2*:
fixes *f g* :: *nat* ⇒ ′*a* :: *comm-semiring-1-cancel*
assumes *multiplicative-function* (*dirichlet-prod f g*)
assumes *multiplicative-function g*
assumes [*simp*]: *f 0* = *0*
shows *multiplicative-function f*
⟨*proof*⟩

lemma *multiplicative-dirichlet-inverse*:
  assumes *multiplicative-function f*
  shows *multiplicative-function* (*dirichlet-inverse f 1*)
⟨*proof*⟩

lemma *dirichlet-prod-prime-power*:
  assumes *prime p*
  shows *dirichlet-prod f g* (p ^ k) = ($\sum$ $i \le k$. f (p ^ i) $*$ g (p ^ ($k - i$)))
⟨*proof*⟩

lemma *dirichlet-prod-prime*:
  assumes *prime p*
  shows *dirichlet-prod f g p* = *f 1* $*$ *g p* + *f p* $*$ *g 1*
  ⟨*proof*⟩

locale *multiplicative-dirichlet-prod* =
  f: *multiplicative-function f* + *g*: *multiplicative-function g*
  for *f g* :: *nat* ⇒ *'a* :: *comm-semiring-1*
begin

sublocale *multiplicative-function dirichlet-prod f g*
  ⟨*proof*⟩

end

locale *multiplicative-dirichlet-prod'* =
  f: *multiplicative-function' f f-prime-power f-prime* +
  g: *multiplicative-function' g g-prime-power g-prime*
  for *f g* :: *nat* ⇒ *'a* :: *comm-semiring-1* **and** *f-prime-power g-prime-power f-prime g-prime*
begin

sublocale *multiplicative-dirichlet-prod f g* ⟨*proof*⟩

sublocale *multiplicative-function' dirichlet-prod f g*
  λ*p k*. *f-prime-power p k* + *g-prime-power p k* +
    ($\sum$ *i*∈{*0*<..<*k*}. *f-prime-power p i* $*$ *g-prime-power p* (*k* − *i*))
  λ*p*. *f-prime p* + *g-prime p*
⟨*proof*⟩

end

end

4 Formal Dirichlet series

theory *Dirichlet-Series*
imports

Complex-Main
Dirichlet-Product
Multiplicative-Function
HOL−Computational-Algebra.Computational-Algebra
HOL−Number-Theory.Number-Theory
HOL−Library.FuncSet
begin

A formal Dirichlet series

$$A(s) = \sum_{n=1}^{\infty} \frac{a_n}{n^s}$$

is represented its coefficient sequence starting from 1. For simplicity, we represent this in Isabelle with a function of type *nat* ⇒ *'a* whose value for n is the $n+1$-th coefficient.

typedef *'a fds = UNIV :: (nat ⇒ 'a) set*
⟨*proof*⟩

setup-lifting *type-definition-fds*

lift-definition *fds-nth* :: *'a fds ⇒ nat ⇒ 'a :: zero* **is**
λf::nat ⇒ 'a. case-nat 0 f ⟨*proof*⟩

lift-definition *fds* :: *(nat ⇒ 'a) ⇒ 'a fds* **is**
λf. f ∘ Suc ⟨*proof*⟩

lemma *fds-nth-fds*: *fds-nth (fds f) n = (if n = 0 then 0 else f n)*
⟨*proof*⟩

lemma *fds-nth-fds'*: *f 0 = 0 ⟹ fds-nth (fds f) = f*
⟨*proof*⟩

lemma *fds-nth-0* [*simp*]: *fds-nth f 0 = 0*
⟨*proof*⟩

lemma *fds-nth-fds-pos* [*simp*]: *n > 0 ⟹ fds-nth (fds f) n = f n*
⟨*proof*⟩

lemma *fds-fds-nth* [*simp*]: *fds (fds-nth f) = f*
⟨*proof*⟩

lemma *fds-eq-fds-iff*:
fds f = fds g ⟷ (∀ n>0. f n = g n)
⟨*proof*⟩

lemma *fds-eq-fds-iff'*: *f 0 = g 0 ⟹ fds f = fds g ⟷ f = g*
⟨*proof*⟩

lemma *fds-eqI* [*intro?*]:

assumes $(\bigwedge n.\ n > 0 \Longrightarrow \textit{fds-nth } f\ n = \textit{fds-nth } g\ n)$
shows $f = g$
⟨*proof*⟩

lemma *fds-cong* [*cong*]: $(\bigwedge n.\ n > 0 \Longrightarrow f\ n = (g\ n :: 'a :: zero)) \Longrightarrow \textit{fds } f = \textit{fds}$
g
⟨*proof*⟩

lemma *fds-eq-iff*: $f = g \longleftrightarrow (\forall n>0.\ \textit{fds-nth } f\ n = \textit{fds-nth } g\ n)$
⟨*proof*⟩

lemma *dirichlet-prod-fds-nth-fds-left* [*simp*]:
dirichlet-prod (*fds-nth* (*fds f*)) *g* = *dirichlet-prod f g*
⟨*proof*⟩

lemma *dirichlet-prod-fds-nth-fds-right* [*simp*]:
dirichlet-prod f (*fds-nth* (*fds g*)) = *dirichlet-prod f g*
⟨*proof*⟩

definition *fds-const* :: $'a :: zero \Rightarrow 'a$ *fds* **where**
fds-const c = *fds* ($\lambda n.$ *if n* = *1 then c else 0*)

abbreviation *fds-ind* **where** *fds-ind P* ≡ *fds* (*ind P*)

bundle *fds-syntax*
begin

notation *fds-nth* (**infixl** ‹$› *75*)
notation *fds* (**binder** ‹χ› *10*)
notation *dirichlet-prod* (**infixl** ‹⋆› *70*)

end

instantiation *fds* :: (*zero*) *zero*
begin
definition *zero-fds* :: *'a fds* **where** *zero-fds* = *fds* (λ-. *0*)
instance ⟨*proof*⟩
end

instantiation *fds* :: ({*zero*,*one*}) *one*
begin
definition *one-fds* :: *'a fds* **where** *one-fds* = *fds* (λ*n*. *if n* = *1 then 1 else 0*)
instance ⟨*proof*⟩
end

instantiation *fds* :: ({*plus*,*zero*}) *plus*
begin

definition *plus-fds* :: *'a fds* ⇒ *'a fds* ⇒ *'a fds*
  where *plus-fds f g* = *fds* (λ*n*. *fds-nth f n* + *fds-nth g n*)
instance ⟨*proof*⟩
end

instantiation *fds* :: (*semiring-0*) *times*
begin
definition *times-fds* :: *'a fds* ⇒ *'a fds* ⇒ *'a fds*
  where *times-fds f g* = *fds* (*dirichlet-prod* (*fds-nth f*) (*fds-nth g*))
instance ⟨*proof*⟩
end

instantiation *fds* :: ({*uminus*,*zero*}) *uminus*
begin
definition *uminus-fds* :: *'a fds* ⇒ *'a fds*
  where *uminus-fds f* = *fds* (λ*n*. −*fds-nth f n*)
instance ⟨*proof*⟩
end

instantiation *fds* :: ({*minus*,*zero*}) *minus*
begin
definition *minus-fds* :: *'a fds* ⇒ *'a fds* ⇒ *'a fds*
  where *minus-fds f g* = *fds* (λ*n*. *fds-nth f n* − *fds-nth g n*)
instance ⟨*proof*⟩
end

4.1 General properties

lemma *fds-nth-zero* [*simp*]: *fds-nth 0* = (λ-. *0*)
  ⟨*proof*⟩

lemma *fds-nth-one*: *fds-nth 1* = (λ*n*. *if n* = *1 then 1 else 0*)
  ⟨*proof*⟩

lemma *fds-nth-one-Suc-0* [*simp*]: *fds-nth 1* (*Suc 0*) = *1*
  ⟨*proof*⟩

lemma *fds-nth-one-not-Suc-0* [*simp*]: *n* ≠ *Suc 0* ⟹ *fds-nth 1 n* = *0*
  ⟨*proof*⟩

lemma *fds-nth-plus* [*simp*]:
  fds-nth (*f* + *g*) = (λ*n*. *fds-nth f n* + *fds-nth g n* :: *'a* :: *monoid-add*)
  ⟨*proof*⟩

lemma *fds-nth-minus* [*simp*]:
  fds-nth (*f* − *g*) = (λ*n*. *fds-nth f n* − *fds-nth g n* :: *'a* :: {*cancel-comm-monoid-add*})
  ⟨*proof*⟩

lemma *fds-nth-uminus* [*simp*]: *fds-nth* (−*g*) = (λ*n*. − *fds-nth g n* :: *'a* :: *group-add*)

⟨*proof*⟩

lemma *fds-nth-mult*: *fds-nth* (*f* ∗ *g*) = *dirichlet-prod* (*fds-nth f*) (*fds-nth g*)
⟨*proof*⟩

lemma *fds-nth-mult-const-left* [*simp*]: *fds-nth* (*fds-const c* ∗ *f*) *n* = *c* ∗ *fds-nth f n*
⟨*proof*⟩

lemma *fds-nth-mult-const-right* [*simp*]: *fds-nth* (*f* ∗ *fds-const c*) *n* = *fds-nth f n* ∗ *c*
⟨*proof*⟩

instance *fds* :: ({*semigroup-add*, *zero*}) *semigroup-add*
⟨*proof*⟩

instance *fds* :: ({*ab-semigroup-add*, *zero*}) *ab-semigroup-add*
⟨*proof*⟩

instance *fds* :: ({*cancel-semigroup-add*, *zero*}) *cancel-semigroup-add*
⟨*proof*⟩

instance *fds* :: ({*cancel-ab-semigroup-add*, *zero*}) *cancel-ab-semigroup-add*
⟨*proof*⟩

instance *fds* :: (*monoid-add*) *monoid-add*
⟨*proof*⟩

instance *fds* :: (*comm-monoid-add*) *comm-monoid-add*
⟨*proof*⟩

instance *fds* :: (*cancel-comm-monoid-add*) *cancel-comm-monoid-add*
⟨*proof*⟩

instance *fds* :: (*group-add*) *group-add*
⟨*proof*⟩

instance *fds* :: (*ab-group-add*) *ab-group-add*
⟨*proof*⟩

instance *fds* :: (*semiring-0*) *semiring-0*
⟨*proof*⟩

instance *fds* :: (*comm-semiring-0*) *comm-semiring-0*
⟨*proof*⟩

instance *fds* :: (*semiring-0-cancel*) *semiring-0-cancel*
⟨*proof*⟩

instance *fds* :: (*comm-semiring-0-cancel*) *comm-semiring-0-cancel* ⟨*proof*⟩

instance *fds* :: (*semiring-1*) *semiring-1*
⟨*proof*⟩

instance *fds* :: (*comm-semiring-1*) *comm-semiring-1*
⟨*proof*⟩

instance *fds* :: (*semiring-1-cancel*) *semiring-1-cancel* ⟨*proof*⟩
instance *fds* :: (*ring*) *ring* ⟨*proof*⟩
instance *fds* :: (*ring-1*) *ring-1* ⟨*proof*⟩
instance *fds* :: (*comm-ring*) *comm-ring* ⟨*proof*⟩

instance *fds* :: (*semiring-no-zero-divisors*) *semiring-no-zero-divisors*
⟨*proof*⟩

instance *fds* :: (*ring-no-zero-divisors*) *ring-no-zero-divisors* ⟨*proof*⟩
instance *fds* :: (*idom*) *idom* ⟨*proof*⟩

instantiation *fds* :: (*real-vector*) *real-vector*
begin

definition *scaleR-fds* :: *real* ⇒ *'a fds* ⇒ *'a fds* **where**
scaleR-fds c f = *fds* (λ*n*. *c* $*_R$ *fds-nth f n*)

lemma *fds-nth-scaleR* [*simp*]: *fds-nth* (*c* $*_R$ *f*) = (λ*n*. *c* $*_R$ *fds-nth f n*)
⟨*proof*⟩

instance ⟨*proof*⟩

end

instance *fds* :: (*real-algebra*) *real-algebra*
⟨*proof*⟩

instance *fds* :: (*real-algebra-1*) *real-algebra-1* ⟨*proof*⟩

lemma *fds-nth-sum* [*simp*]: *fds-nth* (*sum f A*) *n* = *sum* (λ*x*. *fds-nth* (*f x*) *n*) *A*
⟨*proof*⟩

lemma *sum-fds* [*simp*]: ($\sum$*x*∈*A*. *fds* (*f x*)) = *fds* (λ*n*. $\sum$*x*∈*A*. *f x n*)
⟨*proof*⟩

lemma *fds-nth-const*: *fds-nth* (*fds-const c*) = (λ*n*. *if n* = *1 then c else 0*)
⟨*proof*⟩

lemma *fds-nth-const-Suc-0* [*simp*]: *fds-nth* (*fds-const c*) (*Suc 0*) = *c*

⟨*proof*⟩

lemma *fds-nth-const-not-Suc-0* [*simp*]: $n \neq 1 \Longrightarrow$ *fds-nth* (*fds-const c*) *n* = *0*
⟨*proof*⟩

lemma *fds-const-zero* [*simp*]: *fds-const 0* = *0*
⟨*proof*⟩

lemma *fds-const-one* [*simp*]: *fds-const 1* = *1*
⟨*proof*⟩

lemma *fds-const-add* [*simp*]: *fds-const* (*a* + *b* :: *'a* :: *monoid-add*) = *fds-const a* + *fds-const b*
⟨*proof*⟩

lemma *fds-const-minus* [*simp*]:
fds-const (*a* − *b* :: *'a* :: *cancel-comm-monoid-add*) = *fds-const a* − *fds-const b*
⟨*proof*⟩

lemma *fds-const-uminus* [*simp*]:
fds-const (− *b* :: *'a* :: *ab-group-add*) = − *fds-const b*
⟨*proof*⟩

lemma *fds-const-mult* [*simp*]:
fds-const (*a* ∗ *b* :: *'a* :: *semiring-0*) = *fds-const a* ∗ *fds-const b*
⟨*proof*⟩

lemma *fds-const-of-nat* [*simp*]: *fds-const* (*of-nat c*) = *of-nat c*
⟨*proof*⟩

lemma *fds-const-of-int* [*simp*]: *fds-const* (*of-int c*) = *of-int c*
⟨*proof*⟩

lemma *fds-const-of-real* [*simp*]: *fds-const* (*of-real c*) = *of-real c*
⟨*proof*⟩

instantiation *fds* :: ({*inverse*, *comm-ring-1*}) *inverse*
begin

definition *inverse-fds* :: *'a fds* ⇒ *'a fds* **where**
inverse-fds f = *fds* (λ*n*. *dirichlet-inverse* (*fds-nth f*) (*inverse* (*fds-nth f 1*)) *n*)

definition *divide-fds* :: *'a fds* ⇒ *'a fds* ⇒ *'a fds* **where**
divide-fds f g = *f* ∗ *inverse g*

instance ⟨*proof*⟩

end

lemma *numeral-fds*: *numeral n* = *fds-const* (*numeral n*)
⟨*proof*⟩

lemma *fds-ind-False* [*simp*]: *fds-ind* (λ-. *False*) = *0*
⟨*proof*⟩

lemma *fds-commutes*:
assumes ⋀*m n*. *m* > *0* ⟹ *n* > *0* ⟹ *fds-nth f m* ∗ *fds-nth g n* = *fds-nth g n* ∗ *fds-nth f m*
shows *f* ∗ *g* = *g* ∗ *f*
⟨*proof*⟩

lemma *fds-nth-mult-Suc-0* [*simp*]:
fds-nth (*f* ∗ *g*) (*Suc 0*) = *fds-nth f* (*Suc 0*) ∗ *fds-nth g* (*Suc 0*)
⟨*proof*⟩

lemma *fds-nth-inverse*:
fds-nth (*inverse f*) = *dirichlet-inverse* (*fds-nth f*) (*inverse* (*fds-nth f 1*))
⟨*proof*⟩

lemma *inverse-fds-nonunit*:
fds-nth f 1 = (*0* :: ′*a* :: *field*) ⟹ *inverse f* = *0*
⟨*proof*⟩

lemma *inverse-0-fds* [*simp*]: *inverse* (*0* :: ′*a* :: *field fds*) = *0*
⟨*proof*⟩

lemma *fds-left-inverse*:
fds-nth f 1 ≠ (*0* :: ′*a* :: *field*) ⟹ *inverse f* ∗ *f* = *1*
⟨*proof*⟩

lemma *fds-right-inverse*:
fds-nth f 1 ≠ (*0* :: ′*a* :: *field*) ⟹ *f* ∗ *inverse f* = *1*
⟨*proof*⟩

lemma *fds-left-inverse-unique*:
assumes *f* ∗ *g* = (*1* :: ′*a* :: *field fds*)
shows *f* = *inverse g*
⟨*proof*⟩

lemma *fds-right-inverse-unique*:
assumes *f* ∗ *g* = (*1* :: ′*a* :: *field fds*)
shows *g* = *inverse f*
⟨*proof*⟩

lemma *inverse-1-fds* [*simp*]: *inverse* (*1* :: ′*a* :: *field fds*) = *1*
⟨*proof*⟩

lemma *inverse-const-fds* [*simp*]:
inverse (*fds-const c* :: *'a* :: *field fds*) = *fds-const* (*inverse c*)
⟨*proof*⟩

lemma *inverse-mult-fds*: *inverse* (*f* ∗ *g* :: *'a* :: *field fds*) = *inverse f* ∗ *inverse g*
⟨*proof*⟩

definition *fds-zeta* :: *'a* :: *one fds*
where *fds-zeta* = *fds* (λ-. *1*)

lemma *fds-zeta-altdef*: *fds-zeta* = *fds* (λ*n*. *if n* = *0 then 0 else 1*)
⟨*proof*⟩

lemma *fds-nth-zeta*: *fds-nth fds-zeta* = (λ*n*. *if n* = *0 then 0 else 1*)
⟨*proof*⟩

lemma *fds-nth-zeta-pos* [*simp*]: *n* > *0* ⟹ *fds-nth fds-zeta n* = *1*
⟨*proof*⟩

lemma *fds-zeta-commutes*: *fds-zeta* ∗ (*f* :: *'a* :: *semiring-1 fds*) = *f* ∗ *fds-zeta*
⟨*proof*⟩

lemma *fds-ind-True* [*simp*]: *fds-ind* (λ-. *True*) = *fds-zeta*
⟨*proof*⟩

lemma *finite-extensional-prod-nat*:
assumes *finite A b* > *0*
shows *finite* {*d* ∈ *extensional A*. *prod d A* = (*b* :: *nat*)}
⟨*proof*⟩

The n-th coefficient of a product of Dirichlet series can be determined by summing over all products of k_i-th coefficients of the series such that the product of the k_i is n.

lemma *fds-nth-prod*:
assumes *finite A A* ≠ {} *n* > *0*
shows *fds-nth* (∏*x*∈*A*. *f x*) *n* =
(∑ *d* | *d* ∈ *extensional A* ∧ *prod d A* = *n*. ∏*x*∈*A*. *fds-nth* (*f x*) (*d x*))
⟨*proof*⟩

lemma *fds-nth-power-Suc-0* [*simp*]: *fds-nth* (*f* ^ *n*) (*Suc 0*) = *fds-nth f* (*Suc 0*) ^ *n*
⟨*proof*⟩

lemma *fds-nth-prod-Suc-0* [*simp*]: *fds-nth* (*prod f A*) (*Suc 0*) = (∏*x*∈*A*. *fds-nth* (*f x*) (*Suc 0*))
⟨*proof*⟩

lemma *fds-nth-power-eq-0*:

assumes *n* < *2* ^ *k* *fds-nth f 1* = *0*
shows *fds-nth* (*f* ^ *k*) *n* = *0*
⟨*proof*⟩

4.2 Shifting the argument

class *nat-power* = *semiring-1* +
fixes *nat-power* :: *nat* ⇒ *'a* ⇒ *'a*
assumes *nat-power-0-left* [*simp*]: *x* ≠ *0* ⟹ *nat-power 0 x* = *0*
assumes *nat-power-0-right* [*simp*]: *n* > *0* ⟹ *nat-power n 0* = *1*
assumes *nat-power-1-left* [*simp*]: *nat-power* (*Suc 0*) *x* = *1*
assumes *nat-power-1-right* [*simp*]: *nat-power n 1* = *of-nat n*
assumes *nat-power-add*: *n* > *0* ⟹ *nat-power n* (*a* + *b*) = *nat-power n a* ∗ *nat-power n b*
assumes *nat-power-mult-distrib*:
m > *0* ⟹ *n* > *0* ⟹ *nat-power* (*m* ∗ *n*) *a* = *nat-power m a* ∗ *nat-power n a*
assumes *nat-power-power*:
n > *0* ⟹ *nat-power n* (*a* ∗ *of-nat m*) = *nat-power n a* ^ *m*
begin

lemma *nat-power-of-nat* [*simp*]: *m* > *0* ⟹ *nat-power m* (*of-nat n*) = *of-nat* (*m* ^ *n*)
⟨*proof*⟩

lemma *nat-power-power-left*: *m* > *0* ⟹ *nat-power* (*m* ^ *k*) *n* = *nat-power m n* ^ *k*
⟨*proof*⟩

end

class *nat-power-field* = *nat-power* + *field* +
assumes *nat-power-nonzero* [*simp*]: *n* > *0* ⟹ *nat-power n z* ≠ *0*
begin

lemma *nat-power-diff*: *n* > *0* ⟹ *nat-power n* (*a* − *b*) = *nat-power n a* / *nat-power n b*
⟨*proof*⟩

end

instantiation *nat* :: *nat-power*
begin
definition [*simp*]: *nat-power-nat a b* = (*a* ^ *b* :: *nat*)
instance ⟨*proof*⟩
end

instantiation *real* :: *nat-power-field*
begin
definition [*simp*]: *nat-power-real a b* = (*real a powr b*)

instance ⟨*proof*⟩
end

The following operation corresponds to shifting the argument of a Dirichlet series, i. e. subtracting a constant from it. In effect, this turns the series

$$A(s) = \sum_{n=1}^{\infty} \frac{a_n}{n^s}$$

into the series

$$A(s-c) = \sum_{n=1}^{\infty} \frac{n^c \cdot a_n}{n^s} \ .$$

definition *fds-shift* :: *'a* :: *nat-power* ⇒ *'a fds* ⇒ *'a fds* **where**
fds-shift c f = *fds* (*λn. fds-nth f n* ∗ *nat-power n c*)

lemma *fds-nth-shift* [*simp*]: *fds-nth* (*fds-shift c f*) *n* = *fds-nth f n* ∗ *nat-power n c*
⟨*proof*⟩

lemma *fds-shift-shift* [*simp*]: *fds-shift c* (*fds-shift c' f*) = *fds-shift* (*c'* + *c*) *f*
⟨*proof*⟩

lemma *fds-shift-zero* [*simp*]: *fds-shift c 0* = *0*
⟨*proof*⟩

lemma *fds-shift-1* [*simp*]: *fds-shift a 1* = *1*
⟨*proof*⟩

lemma *fds-shift-const* [*simp*]: *fds-shift a* (*fds-const c*) = *fds-const c*
⟨*proof*⟩

lemma *fds-shift-add* [*simp*]:
fixes *f g* :: *'a* :: {*monoid-add*, *nat-power*} *fds*
shows *fds-shift c* (*f* + *g*) = *fds-shift c f* + *fds-shift c g*
⟨*proof*⟩

lemma *fds-shift-minus* [*simp*]:
fixes *f g* :: *'a* :: {*comm-semiring-1-cancel*, *nat-power*} *fds*
shows *fds-shift c* (*f* − *g*) = *fds-shift c f* − *fds-shift c g*
⟨*proof*⟩

lemma *fds-shift-uminus* [*simp*]:
fixes *f* :: *'a* :: {*ring*, *nat-power*} *fds*
shows *fds-shift c* (−*f*) = −*fds-shift c f*
⟨*proof*⟩

lemma *fds-shift-mult* [*simp*]:
fixes *f g* :: *'a* :: {*comm-semiring*, *nat-power*} *fds*
shows *fds-shift c* (*f* ∗ *g*) = *fds-shift c f* ∗ *fds-shift c g*

⟨*proof*⟩

lemma *fds-shift-power* [*simp*]:
fixes *f* :: *'a* :: {*comm-semiring*, *nat-power*} *fds*
shows *fds-shift c* (*f* ^ *n*) = *fds-shift c f* ^ *n*
⟨*proof*⟩

lemma *fds-shift-by-0* [*simp*]: *fds-shift 0 f* = *f*
⟨*proof*⟩

lemma *fds-shift-inverse* [*simp*]:
fds-shift (*a* :: *'a* :: {*field*, *nat-power*}) (*inverse f*) = *inverse* (*fds-shift a f*)
⟨*proof*⟩

lemma *fds-shift-divide* [*simp*]:
fds-shift (*a* :: *'a* :: {*field*, *nat-power*}) (*f* / *g*) = *fds-shift a f* / *fds-shift a g*
⟨*proof*⟩

lemma *fds-shift-sum* [*simp*]: *fds-shift a* (∑ *x*∈*A*. *f x*) = (∑ *x*∈*A*. *fds-shift a* (*f x*))
⟨*proof*⟩

lemma *fds-shift-prod* [*simp*]: *fds-shift a* (∏ *x*∈*A*. *f x*) = (∏ *x*∈*A*. *fds-shift a* (*f x*))
⟨*proof*⟩

4.3 Scaling the argument

The following operation corresponds to scaling the argument of a Dirichlet series with a natural number, i. e. turning the series

$$A(s) = \sum_{n=1}^{\infty} \frac{a_n}{n^s}$$

into the series

$$A(ks) = \sum_{n=1}^{\infty} \frac{a_n}{(n^k)^2}\ .$$

definition *fds-scale* :: *nat* ⇒ (*'a* :: *zero*) *fds* ⇒ *'a fds* **where**
fds-scale c f =
fds (λ*n*. *if n* > *0* ∧ *is-nth-power c n then fds-nth f* (*nth-root-nat c n*) *else 0*)

lemma *fds-scale-0* [*simp*]: *fds-scale 0 f* = *0*
⟨*proof*⟩

lemma *fds-scale-1* [*simp*]: *fds-scale 1 f* = *f*
⟨*proof*⟩

lemma *fds-nth-scale-power* [*simp*]:
c > *0* ⟹ *fds-nth* (*fds-scale c f*) (*n* ^ *c*) = *fds-nth f n*

⟨*proof*⟩

lemma *fds-nth-scale-nonpower* [*simp*]:
¬*is-nth-power c n* ⟹ *fds-nth* (*fds-scale c f*) *n* = *0*
⟨*proof*⟩

lemma *fds-nth-scale*:
fds-nth (*fds-scale c f*) *n* =
(*if n* > *0* ∧ *is-nth-power c n then fds-nth f* (*nth-root-nat c n*) *else 0*)
⟨*proof*⟩

lemma *fds-scale-const* [*simp*]: *c* > *0* ⟹ *fds-scale c* (*fds-const c*′) = *fds-const c*′
⟨*proof*⟩

lemma *fds-scale-zero* [*simp*]: *fds-scale c 0* = *0*
⟨*proof*⟩

lemma *fds-scale-one* [*simp*]: *c* > *0* ⟹ *fds-scale c 1* = *1*
⟨*proof*⟩

lemma *fds-scale-of-nat* [*simp*]: *c* > *0* ⟹ *fds-scale c* (*of-nat n*) = *of-nat n*
⟨*proof*⟩

lemma *fds-scale-of-int* [*simp*]: *c* > *0* ⟹ *fds-scale c* (*of-int n*) = *of-int n*
⟨*proof*⟩

lemma *fds-scale-numeral* [*simp*]: *c* > *0* ⟹ *fds-scale c* (*numeral n*) = *numeral n*
⟨*proof*⟩

lemma *fds-scale-scale*: *fds-scale c* (*fds-scale c*′ *f*) = *fds-scale* (*c* ∗ *c*′) *f*
⟨*proof*⟩

lemma *fds-scale-add* [*simp*]:
fixes *f g* :: ′*a* :: *monoid-add fds*
shows *fds-scale c* (*f* + *g*) = *fds-scale c f* + *fds-scale c g*
⟨*proof*⟩

lemma *fds-scale-minus* [*simp*]:
fixes *f g* :: ′*a* :: {*cancel-comm-monoid-add*} *fds*
shows *fds-scale c* (*f* − *g*) = *fds-scale c f* − *fds-scale c g*
⟨*proof*⟩

lemma *fds-scale-uminus* [*simp*]:
fixes *f* :: ′*a* :: *group-add fds*
shows *fds-scale c* (−*f*) = −*fds-scale c f*
⟨*proof*⟩

lemma *fds-scale-mult* [*simp*]:
fixes *f g* :: ′*a* :: *semiring-0 fds*

shows *fds-scale c* $(f * g)$ = *fds-scale c f* $*$ *fds-scale c g*
⟨*proof*⟩

lemma *fds-scale-shift*:
fds-shift d (*fds-scale c f*) = *fds-scale c* (*fds-shift* $(c * d)$ *f*)
⟨*proof*⟩

lemma *fds-ind-nth-power*: $k > 0 \Longrightarrow$ *fds-ind* (*is-nth-power k*) = *fds-scale k fds-zeta*
⟨*proof*⟩

4.4 Formal derivative

The formal derivative of a series

$$A(s) = \sum_{n=1}^{\infty} \frac{a_n}{n^s}$$

can easily be seen to be

$$A'(s) = -\sum_{n=1}^{\infty} \frac{\ln n \cdot a_n}{n^s} \ .$$

definition *fds-deriv* :: *'a* :: *real-algebra fds* ⇒ *'a fds* **where**
fds-deriv f = *fds* (λ*n*. − *ln* (*real n*) $*_R$ *fds-nth f n*)

lemma *fds-nth-deriv*: *fds-nth* (*fds-deriv f*) *n* = −*ln* (*real n*) $*_R$ *fds-nth f n*
⟨*proof*⟩

lemma *fds-deriv-const* [*simp*]: *fds-deriv* (*fds-const c*) = *0*
⟨*proof*⟩

lemma *fds-deriv-0* [*simp*]: *fds-deriv 0* = *0*
⟨*proof*⟩

lemma *fds-deriv-1* [*simp*]: *fds-deriv 1* = *0*
⟨*proof*⟩

lemma *fds-deriv-of-nat* [*simp*]: *fds-deriv* (*of-nat n*) = *0*
⟨*proof*⟩

lemma *fds-deriv-of-int* [*simp*]: *fds-deriv* (*of-int n*) = *0*
⟨*proof*⟩

lemma *fds-deriv-of-real* [*simp*]: *fds-deriv* (*of-real n*) = *0*
⟨*proof*⟩

lemma *fds-deriv-uminus* [*simp*]: *fds-deriv* (−*f*) = −*fds-deriv f*
⟨*proof*⟩

lemma *fds-deriv-add* [*simp*]: *fds-deriv* (*f* + *g*) = *fds-deriv f* + *fds-deriv g*
⟨*proof*⟩

lemma *fds-deriv-minus* [*simp*]: *fds-deriv* (*f* − *g*) = *fds-deriv f* − *fds-deriv g*
⟨*proof*⟩

lemma *fds-deriv-times* [*simp*]:
fds-deriv (*f* ∗ *g*) = *fds-deriv f* ∗ *g* + *f* ∗ *fds-deriv g*
⟨*proof*⟩

lemma *fds-deriv-inverse* [*simp*]:
fixes *f* :: *'a* :: {*real-algebra*, *field*} *fds*
assumes *fds-nth f* (*Suc 0*) ≠ *0*
shows *fds-deriv* (*inverse f*) = −*fds-deriv f* / *f* ^ *2*
⟨*proof*⟩

lemma *fds-deriv-shift* [*simp*]: *fds-deriv* (*fds-shift c f*) = *fds-shift c* (*fds-deriv f*)
⟨*proof*⟩

lemma *fds-deriv-scale*: *fds-deriv* (*fds-scale c f*) = *of-nat c* ∗ *fds-scale c* (*fds-deriv f*)
⟨*proof*⟩

lemma *fds-deriv-eq-imp-eq*:
assumes *fds-deriv f* = *fds-deriv g* *fds-nth f* (*Suc 0*) = *fds-nth g* (*Suc 0*)
shows *f* = *g*
⟨*proof*⟩

lemma *completely-multiplicative-fds-deriv*:
assumes *completely-multiplicative-function f*
shows *fds-deriv* (*fds f*) = −*fds* (λ*n*. *f n* ∗ *mangoldt n*) ∗ *fds f*
⟨*proof*⟩

lemma *completely-multiplicative-fds-deriv'*:
completely-multiplicative-function (*fds-nth f*) ⟹
fds-deriv f = − *fds* (λ*n*. *fds-nth f n* ∗ *mangoldt n*) ∗ *f*
⟨*proof*⟩

lemma *fds-deriv-zeta*:
fds-deriv fds-zeta =
−*fds mangoldt* ∗ (*fds-zeta* :: *'a* :: {*comm-semiring-1*,*real-algebra-1*} *fds*)
⟨*proof*⟩

lemma *fds-mangoldt-times-zeta*: *fds mangoldt* ∗ *fds-zeta* = *fds* (λ*x*. *of-real* (*ln* (*real x*)))
⟨*proof*⟩

lemma *fds-deriv-zeta'*: *fds-deriv fds-zeta* =

$-fds\ (\lambda x.\ of\text{-}real\ (ln\ (real\ x)){::}\ 'a :: \{comm\text{-}semiring\text{-}1, real\text{-}algebra\text{-}1\})$
⟨*proof*⟩

4.5 Formal integral

definition *fds-integral* :: $'a \Rightarrow 'a$ :: *real-algebra fds* $\Rightarrow$ $'a$ *fds* **where**
fds-integral c f = *fds* (λn. *if n* = *1 then c else* − *fds-nth f n* $/_R$ *ln* (*real n*))

lemma *fds-integral-0* [*simp*]: *fds-integral a 0* = *fds-const a*
⟨*proof*⟩

lemma *fds-integral-add*: *fds-integral* (*a* + *b*) (*f* + *g*) = *fds-integral a f* + *fds-integral b g*
⟨*proof*⟩

lemma *fds-integral-diff*: *fds-integral* (*a* − *b*) (*f* − *g*) = *fds-integral a f* − *fds-integral b g*
⟨*proof*⟩

lemma *fds-integral-minus*: *fds-integral* (−*a*) (−*f*) = −*fds-integral a f*
⟨*proof*⟩

lemma *fds-shift-integral*: *fds-shift b* (*fds-integral a f*) = *fds-integral a* (*fds-shift b f*)
⟨*proof*⟩

lemma *fds-deriv-fds-integral* [*simp*]:
fds-nth f (*Suc 0*) = *0* $\Longrightarrow$ *fds-deriv* (*fds-integral c f*) = *f*
⟨*proof*⟩

lemma *fds-integral-fds-deriv* [*simp*]: *fds-integral* (*fds-nth f 1*) (*fds-deriv f*) = *f*
⟨*proof*⟩

4.6 Formal logarithm

definition *fds-ln* :: $'a \Rightarrow 'a$ :: {*real-normed-field*} *fds* $\Rightarrow$ $'a$ *fds* **where**
fds-ln l f = *fds-integral l* (*fds-deriv f* / *f*)

lemma *fds-nth-Suc-0-fds-deriv* [*simp*]: *fds-nth* (*fds-deriv f*) (*Suc 0*) = *0*
⟨*proof*⟩

lemma *fds-deriv-fds-ln* [*simp*]: *fds-deriv* (*fds-ln l f*) = *fds-deriv f* / *f*
⟨*proof*⟩

lemma *fds-nth-Suc-0-fds-ln* [*simp*]: *fds-nth* (*fds-ln l f*) (*Suc 0*) = *l*
⟨*proof*⟩

lemma *fds-ln-const* [*simp*]: *fds-ln l* (*fds-const c*) = *fds-const l*
⟨*proof*⟩

lemma *fds-ln-0* [*simp*]: *fds-ln l 0 = fds-const l*
⟨*proof*⟩

lemma *fds-ln-1* [*simp*]: *fds-ln l 1 = fds-const l*
⟨*proof*⟩

lemma *fds-shift-ln* [*simp*]: *fds-shift a (fds-ln l f) = fds-ln l (fds-shift a f)*
⟨*proof*⟩

lemma *fds-ln-mult*:
assumes *fds-nth f 1 ≠ 0 fds-nth g 1 ≠ 0 l′ + l′′ = l*
shows *fds-ln l (f ∗ g) = fds-ln l′ f + fds-ln l′′ g*
⟨*proof*⟩

lemma *fds-ln-power*:
assumes *fds-nth f 1 ≠ 0 l = of-nat n ∗ l′*
shows *fds-ln l (f ^ n) = of-nat n ∗ fds-ln l′ f*
⟨*proof*⟩

lemma *fds-ln-prod*:
assumes $\bigwedge$*x. x ∈ A ⟹ fds-nth (f x) 1 ≠ 0* ($\sum$*x∈A. l′ x) = l*
shows *fds-ln l* ($\prod$*x∈A. f x) =* ($\sum$*x∈A. fds-ln (l′ x) (f x))*
⟨*proof*⟩

4.7 Formal exponential

definition *fds-exp* :: *′a* :: {*real-normed-algebra-1*,*banach*} *fds ⇒ ′a fds* **where**
fds-exp f = (let f′ = fds (λn. if n = 1 then 0 else fds-nth f n)
in fds (λn. exp (fds-nth f 1) ∗ ($\sum$*k. fds-nth (f′ ^ k) n* $/_R$ *fact k)))*

lemma *fds-nth-exp-Suc-0* [*simp*]: *fds-nth (fds-exp f) (Suc 0) = exp (fds-nth f 1)*
⟨*proof*⟩

lemma *fds-exp-times-fds-nth-0*:
fds-const (exp (fds-nth f (Suc 0))) ∗ fds-exp (f − fds-const (fds-nth f (Suc 0)))
= fds-exp f
⟨*proof*⟩

lemma *fds-exp-const* [*simp*]: *fds-exp (fds-const c) = fds-const (exp c)*
⟨*proof*⟩

lemma *fds-exp-numeral* [*simp*]: *fds-exp (numeral n) = fds-const (exp (numeral n))*
⟨*proof*⟩

lemma *fds-exp-0* [*simp*]: *fds-exp 0 = 1*
⟨*proof*⟩

lemma *fds-exp-1* [*simp*]: *fds-exp 1 = fds-const (exp 1)*
⟨*proof*⟩

lemma *fds-nth-Suc-0-exp* [*simp*]: *fds-nth* (*fds-exp f*) (*Suc 0*) = *exp* (*fds-nth f* (*Suc 0*))
⟨*proof*⟩

4.8 Subseries

definition *fds-subseries* :: (*nat* ⇒ *bool*) ⇒ (*'a* :: *semiring-1*) *fds* ⇒ *'a fds* **where**
fds-subseries P f = *fds* (λ*n*. *if P n then fds-nth f n else 0*)

lemma *fds-nth-subseries*:
fds-nth (*fds-subseries P f*) *n* = (*if P n then fds-nth f n else 0*)
⟨*proof*⟩

lemma *fds-subseries-0* [*simp*]: *fds-subseries P 0* = *0*
⟨*proof*⟩

lemma *fds-subseries-1* [*simp*]: *P 1* ⟹ *fds-subseries P 1* = *1*
⟨*proof*⟩

lemma *fds-subseries-const* [*simp*]: *P 1* ⟹ *fds-subseries P* (*fds-const c*) = *fds-const c*
⟨*proof*⟩

lemma *fds-subseries-add* [*simp*]: *fds-subseries P* (*f* + *g*) = *fds-subseries P f* + *fds-subseries P g*
⟨*proof*⟩

lemma *fds-subseries-diff* [*simp*]:
fds-subseries P (*f* − *g* :: *'a* :: *ring-1 fds*) = *fds-subseries P f* − *fds-subseries P g*
⟨*proof*⟩

lemma *fds-subseries-minus* [*simp*]:
fds-subseries P (−*f* :: *'a* :: *ring-1 fds*) = − *fds-subseries P f*
⟨*proof*⟩

lemma *fds-subseries-sum* [*simp*]: *fds-subseries P* (∑ *x*∈*A*. *f x*) = (∑ *x*∈*A*. *fds-subseries P* (*f x*))
⟨*proof*⟩

lemma *fds-subseries-shift* [*simp*]:
fds-subseries P (*fds-shift c f*) = *fds-shift c* (*fds-subseries P f*)
⟨*proof*⟩

lemma *fds-subseries-deriv* [*simp*]:
fds-subseries P (*fds-deriv f*) = *fds-deriv* (*fds-subseries P f*)
⟨*proof*⟩

lemma *fds-subseries-integral* [*simp*]:

P 1 ∨ *c* = *0* ⟹ *fds-subseries P* (*fds-integral c f*) = *fds-integral c* (*fds-subseries P f*)
⟨*proof*⟩

abbreviation *fds-primepow-subseries* :: *nat* ⇒ (*'a* :: *semiring-1*) *fds* ⇒ *'a fds* **where**
fds-primepow-subseries p f ≡ *fds-subseries* (*λn. prime-factors n* ⊆ {*p*}) *f*

lemma *fds-primepow-subseries-mult* [*simp*]:
fixes *p* :: *nat*
defines *P* ≡ (*λn. prime-factors n* ⊆ {*p*})
shows *fds-subseries P* (*f* ∗ *g*) = *fds-subseries P f* ∗ *fds-subseries P g*
⟨*proof*⟩

lemma *fds-primepow-subseries-power* [*simp*]:
fds-primepow-subseries p (*f* ^ *n*) = *fds-primepow-subseries p f* ^ *n*
⟨*proof*⟩

lemma *fds-primepow-subseries-prod* [*simp*]:
fds-primepow-subseries p (∏ *x*∈*A*. *f x*) = (∏ *x*∈*A*. *fds-primepow-subseries p* (*f x*))
⟨*proof*⟩

lemma *completely-multiplicative-function-only-pows*:
assumes *completely-multiplicative-function* (*fds-nth f*)
shows *completely-multiplicative-function* (*fds-nth* (*fds-primepow-subseries p f*))
⟨*proof*⟩

4.9 Truncation

definition *fds-truncate* :: *nat* ⇒ *'a* ::{*zero*} *fds* ⇒ *'a fds* **where**
fds-truncate m f = *fds* (*λn. if n* ≤ *m then fds-nth f n else 0*)

lemma *fds-nth-truncate*: *fds-nth* (*fds-truncate m f*) *n* = (*if n* ≤ *m then fds-nth f n else 0*)
⟨*proof*⟩

lemma *fds-truncate-0* [*simp*]: *fds-truncate 0 f* = *0*
⟨*proof*⟩

lemma *fds-truncate-zero* [*simp*]: *fds-truncate m 0* = *0*
⟨*proof*⟩

lemma *fds-truncate-one* [*simp*]: *m* > *0* ⟹ *fds-truncate m 1* = *1*
⟨*proof*⟩

lemma *fds-truncate-const* [*simp*]: *m* > *0* ⟹ *fds-truncate m* (*fds-const c*) = *fds-const c*
⟨*proof*⟩

lemma *fds-truncate-truncate* [*simp*]: *fds-truncate m* (*fds-truncate n f*) = *fds-truncate* (*min m n*) *f*
⟨*proof*⟩

lemma *fds-truncate-truncate′* [*simp*]: *fds-truncate m* (*fds-truncate m f*) = *fds-truncate m f*
⟨*proof*⟩

lemma *fds-truncate-shift* [*simp*]: *fds-truncate m* (*fds-shift a f*) = *fds-shift a* (*fds-truncate m f*)
⟨*proof*⟩

lemma *fds-truncate-add-strong*:
fds-truncate m (*f* + *g* :: ′*a* :: *monoid-add fds*) = *fds-truncate m f* + *fds-truncate m g*
⟨*proof*⟩

lemma *fds-truncate-add*:
fds-truncate m (*fds-truncate m f* + *fds-truncate m g* :: ′*a* :: *monoid-add fds*) =
fds-truncate m (*f* + *g*)
⟨*proof*⟩

lemma *fds-truncate-mult*:
fds-truncate m (*fds-truncate m f* ∗ *fds-truncate m g*) = *fds-truncate m* (*f* ∗ *g*) (**is** *?A* = *?B*)
⟨*proof*⟩

lemma *fds-truncate-deriv*: *fds-truncate m* (*fds-deriv f*) = *fds-deriv* (*fds-truncate m f*)
⟨*proof*⟩

lemma *fds-truncate-integral*:
m > *0* ∨ *c* = *0* ⟹ *fds-truncate m* (*fds-integral c f*) = *fds-integral c* (*fds-truncate m f*)
⟨*proof*⟩

lemma *fds-truncate-power*: *fds-truncate m* (*fds-truncate m f* ^ *n*) = *fds-truncate m* (*f* ^ *n*)
⟨*proof*⟩

lemma *dirichlet-inverse-cong-simp*:
assumes ⋀*m*. *m* > *0* ⟹ *m* ≤ *n* ⟹ *f m* = *f′ m* *i* = *i′* *n* = *n′*
shows *dirichlet-inverse f i n* = *dirichlet-inverse f′ i′ n′*
⟨*proof*⟩

lemma *fds-truncate-cong*:
(⋀*n*. *m* > *0* ⟹ *n* > *0* ⟹ *n* ≤ *m* ⟹ *fds-nth f n* = *fds-nth f′ n*) ⟹
fds-truncate m f = *fds-truncate m f′*

⟨*proof*⟩

lemma *fds-truncate-inverse*:
fds-truncate m (*inverse* (*fds-truncate m* (*f* :: *'a* :: *field fds*))) = *fds-truncate m* (*inverse f*)
⟨*proof*⟩

lemma *fds-truncate-divide*:
fixes *f g* :: *'a* :: *field fds*
shows *fds-truncate m* (*fds-truncate m f* / *fds-truncate m g*) = *fds-truncate m* (*f* / *g*)
⟨*proof*⟩

lemma *fds-truncate-ln*:
fixes *f* :: *'a* :: *real-normed-field fds*
shows *fds-truncate m* (*fds-ln l* (*fds-truncate m f*)) = *fds-truncate m* (*fds-ln l f*)
⟨*proof*⟩

lemma *fds-truncate-exp*:
shows *fds-truncate m* (*fds-exp* (*fds-truncate m f*)) = *fds-truncate m* (*fds-exp f*)
⟨*proof*⟩

lemma *fds-eqI-truncate*:
assumes $\bigwedge m.\ m > 0 \Longrightarrow$ *fds-truncate m f* = *fds-truncate m g*
shows *f* = *g*
⟨*proof*⟩

4.10 Normed series

definition *fds-norm* :: *'a* :: {*real-normed-div-algebra*} *fds* ⇒ *real fds*
where *fds-norm f* = *fds* (λ*n*. *of-real* (*norm* (*fds-nth f n*)))

lemma *fds-nth-norm* [*simp*]: *fds-nth* (*fds-norm f*) *n* = *norm* (*fds-nth f n*)
⟨*proof*⟩

lemma *fds-norm-1* [*simp*]: *fds-norm 1* = *1*
⟨*proof*⟩

lemma *fds-nth-norm-mult-le*:
shows *norm* (*fds-nth* (*f* ∗ *g*) *n*) ≤ *fds-nth* (*fds-norm f* ∗ *fds-norm g*) *n*
⟨*proof*⟩

lemma *fds-nth-norm-mult-nonneg* [*simp*]: *fds-nth* (*fds-norm f* ∗ *fds-norm g*) *n* ≥ *0*
⟨*proof*⟩

4.11 Lifting a real series to a real algebra

definition *fds-of-real* :: *real fds* ⇒ *'a* :: {*real-normed-algebra-1*} *fds* **where**
fds-of-real f = *fds* (λ*n*. *of-real* (*fds-nth f n*))

lemma *fds-nth-of-real* [*simp*]: *fds-nth* (*fds-of-real f*) *n* = *of-real* (*fds-nth f n*)
⟨*proof*⟩

lemma *fds-of-real-0* [*simp*]: *fds-of-real 0* = *0*
and *fds-of-real-1* [*simp*]: *fds-of-real 1* = *1*
and *fds-of-real-const* [*simp*]: *fds-of-real* (*fds-const c*) = *fds-const* (*of-real c*)
and *fds-of-real-minus* [*simp*]: *fds-of-real* (−*f*) = −*fds-of-real f*
and *fds-of-real-add* [*simp*]: *fds-of-real* (*f* + *g*) = *fds-of-real f* + *fds-of-real g*
and *fds-of-real-mult* [*simp*]: *fds-of-real* (*f* ∗ *g*) = *fds-of-real f* ∗ *fds-of-real g*
and *fds-of-real-deriv* [*simp*]: *fds-of-real* (*fds-deriv f*) = *fds-deriv* (*fds-of-real f*)
⟨*proof*⟩

lemma *fds-of-real-higher-deriv* [*simp*]:
(*fds-deriv* ^^ *n*) (*fds-of-real f*) = *fds-of-real* ((*fds-deriv* ^^ *n*) *f*)
⟨*proof*⟩

4.12 Convergence and connection to concrete functions

The following definitions establish a connection of a formal Dirichlet series to the concrete analytic function that it corresponds to. This correspondence is usually partial in the sense that a series may not converge everywhere.

definition *eval-fds* :: (*'a* :: {*nat-power*, *real-normed-field*, *banach*}) *fds* ⇒ *'a* ⇒ *'a*
where
eval-fds f s = ($\sum$ *n*. *fds-nth f n* / *nat-power n s*)

lemma *eval-fds-eqI*:
assumes (λ*n*. *fds-nth f* (*Suc n*) / *nat-power* (*Suc n*) *s*) *sums L*
shows *eval-fds f s* = *L*
⟨*proof*⟩

definition *fds-converges* ::
(*'a* :: {*nat-power*, *real-normed-field*, *banach*}) *fds* ⇒ *'a* ⇒ *bool* **where**
fds-converges f s ⟷ *summable* (λ*n*. *fds-nth f n* / *nat-power n s*)

lemma *fds-converges-iff*:
fds-converges f s ⟷ (λ*n*. *fds-nth f n* / *nat-power n s*) *sums eval-fds f s*
⟨*proof*⟩

definition *fds-abs-converges* ::
(*'a* :: {*nat-power*, *real-normed-field*, *banach*}) *fds* ⇒ *'a* ⇒ *bool* **where**
fds-abs-converges f s ⟷ *summable* (λ*n*. *norm* (*fds-nth f n* / *nat-power n s*))

lemma *fds-abs-converges-imp-converges* [*dest*, *intro*]:
fds-abs-converges f s ⟹ *fds-converges f s*
⟨*proof*⟩

lemma *fds-converges-altdef*:

fds-converges f s ⟷ (λ*n*. *fds-nth f* (*Suc n*) / *nat-power* (*Suc n*) *s*) *sums eval-fds f s*
⟨*proof*⟩

lemma *fds-const-abs-converges* [*simp*]: *fds-abs-converges* (*fds-const c*) *s*
⟨*proof*⟩

lemma *fds-const-converges* [*simp*]: *fds-converges* (*fds-const c*) *s*
⟨*proof*⟩

lemma *eval-fds-const* [*simp*]: *eval-fds* (*fds-const c*) = (λ-. *c*)
⟨*proof*⟩

lemma *fds-zero-abs-converges* [*simp*]: *fds-abs-converges 0 s*
⟨*proof*⟩

lemma *fds-zero-converges* [*simp*]: *fds-converges 0 s*
⟨*proof*⟩

lemma *eval-fds-zero* [*simp*]: *eval-fds 0* = (λ-. *0*)
⟨*proof*⟩

lemma *fds-one-abs-converges* [*simp*]: *fds-abs-converges 1 s*
⟨*proof*⟩

lemma *fds-one-converges* [*simp*]: *fds-converges 1 s*
⟨*proof*⟩

lemma *fds-converges-truncate* [*simp*]: *fds-converges* (*fds-truncate n f*) *s*
⟨*proof*⟩

lemma *fds-abs-converges-truncate* [*simp*]: *fds-abs-converges* (*fds-truncate n f*) *s*
⟨*proof*⟩

lemma *fds-abs-converges-subseries* [*simp*, *intro*]:
assumes *fds-abs-converges f s*
shows *fds-abs-converges* (*fds-subseries P f*) *s*
⟨*proof*⟩

lemma *eval-fds-one* [*simp*]: *eval-fds 1* = (λ-. *1*)
⟨*proof*⟩

lemma *eval-fds-truncate*: *eval-fds* (*fds-truncate n f*) *s* = (∑*k*=*1*..*n*. *fds-nth f k* / *nat-power k s*)
⟨*proof*⟩

lemma *fds-converges-add*:
assumes *fds-converges f s fds-converges g s*

shows *fds-converges* $(f + g)$ *s*
⟨*proof*⟩

lemma *fds-abs-converges-add*:
assumes *fds-abs-converges f s fds-abs-converges g s*
shows *fds-abs-converges* $(f + g)$ *s*
⟨*proof*⟩

lemma *eval-fds-add*:
assumes *fds-converges f s fds-converges g s*
shows *eval-fds* $(f + g)$ $s =$ *eval-fds f s* $+$ *eval-fds g s*
⟨*proof*⟩

lemma *fds-converges-uminus*:
assumes *fds-converges f s*
shows *fds-converges* $(-f)$ *s*
⟨*proof*⟩

lemma *The-cong*: *The* $P =$ *The* Q **if** $\bigwedge x.\ P\ x \longleftrightarrow Q\ x$
⟨*proof*⟩

lemma *fds-abs-converges-uminus*:
assumes *fds-abs-converges f s*
shows *fds-abs-converges* $(-f)$ *s*
⟨*proof*⟩

lemma *eval-fds-uminus*: *fds-converges f s* $\Longrightarrow$ *eval-fds* $(-f)$ $s = -$*eval-fds f s*
⟨*proof*⟩

lemma *fds-converges-diff*:
assumes *fds-converges f s fds-converges g s*
shows *fds-converges* $(f - g)$ *s*
⟨*proof*⟩

lemma *fds-abs-converges-diff*:
assumes *fds-abs-converges f s fds-abs-converges g s*
shows *fds-abs-converges* $(f - g)$ *s*
⟨*proof*⟩

lemma *eval-fds-diff*:
assumes *fds-converges f s fds-converges g s*
shows *eval-fds* $(f - g)$ $s =$ *eval-fds f s* $-$ *eval-fds g s*
⟨*proof*⟩

lemma *eval-fds-at-nat*: *eval-fds f* (*of-nat k*) $= (\sum n.$ *fds-nth f n* / *of-nat n* ^ $k)$
⟨*proof*⟩

lemma *eval-fds-at-numeral*: *eval-fds f* (*numeral k*) = ($\sum$ *n*. *fds-nth f n* / *of-nat n* ^ *numeral k*)
⟨*proof*⟩

lemma *eval-fds-at-1*: *eval-fds f 1* = ($\sum$ *n*. *fds-nth f n* / *of-nat n*)
⟨*proof*⟩

lemma *eval-fds-at-0*: *eval-fds f 0* = ($\sum$ *n*. *fds-nth f n*)
⟨*proof*⟩

lemma *suminf-fds-zeta-aux*:
f 0 = *0* ⟹ ($\sum$ *n*. *fds-nth fds-zeta n* / *f n*) = ($\sum$ *n*. *1* / *f n* :: ′*a* :: *real-normed-field*)
⟨*proof*⟩

lemma *fds-converges-shift* [*simp*]:
fixes *z* :: ′*a* :: {*banach*, *nat-power-field*, *real-normed-field*}
shows *fds-converges* (*fds-shift c f*) *z* ⟷ *fds-converges f* (*z* − *c*)
⟨*proof*⟩

lemma *fds-abs-converges-shift* [*simp*]:
fixes *z* :: ′*a* :: {*banach*, *nat-power-field*, *real-normed-field*}
shows *fds-abs-converges* (*fds-shift c f*) *z* ⟷ *fds-abs-converges f* (*z* − *c*)
⟨*proof*⟩

lemma *fds-eval-shift* [*simp*]:
fixes *z* :: ′*a* :: {*banach*, *nat-power-field*, *real-normed-field*}
shows *eval-fds* (*fds-shift c f*) *z* = *eval-fds f* (*z* − *c*)
⟨*proof*⟩

lemma *fds-converges-scale* [*simp*]:
fixes *z* :: ′*a* :: {*banach*, *nat-power-field*, *real-normed-field*}
assumes *c*: *c* > *0*
shows *fds-converges* (*fds-scale c f*) *z* ⟷ *fds-converges f* (*of-nat c* ∗ *z*)
⟨*proof*⟩

lemma *fds-abs-converges-scale* [*simp*]:
fixes *z* :: ′*a* :: {*banach*, *nat-power-field*, *real-normed-field*}
assumes *c*: *c* > *0*
shows *fds-abs-converges* (*fds-scale c f*) *z* ⟷ *fds-abs-converges f* (*of-nat c* ∗ *z*)
⟨*proof*⟩

lemma *eval-fds-scale* [*simp*]:
fixes *z* :: ′*a* :: {*banach*, *nat-power-field*, *real-normed-field*}
assumes *c*: *c* > *0*
shows *eval-fds* (*fds-scale c f*) *z* = *eval-fds f* (*of-nat c* ∗ *z*)
⟨*proof*⟩

lemma *fds-abs-converges-integral*:
  assumes *fds-abs-converges f s*
  shows *fds-abs-converges (fds-integral c f) s*
  ⟨*proof*⟩

lemma *fds-abs-converges-ln*:
  assumes *fds-abs-converges (fds-deriv f / f) s*
  shows *fds-abs-converges (fds-ln l f) s*
  ⟨*proof*⟩

end

5 The Möbius μ function

theory *Moebius-Mu*
imports
  Main
  HOL−Number-Theory.Number-Theory
  HOL−Computational-Algebra.Squarefree
  Dirichlet-Series
  Dirichlet-Misc
begin

definition *moebius-mu* :: *nat* ⇒ *'a* :: *comm-ring-1* **where**
  moebius-mu n =
    (*if squarefree n then* (−*1*) ^ *card* (*prime-factors n*) *else 0*)

lemma *abs-moebius-mu-le*: *abs* (*moebius-mu n* :: *'a* :: {*linordered-idom*}) ≤ *1*
  ⟨*proof*⟩

lemma *of-int-moebius-mu* [*simp*]: *of-int* (*moebius-mu n*) = *moebius-mu n*
  ⟨*proof*⟩

lemma *minus-1-power-ring-neq-zero* [*simp*]: (− *1* :: *'a* :: *ring-1*) ^ *n* ≠ *0*
  ⟨*proof*⟩

lemma *moebius-mu-0* [*simp*]: *moebius-mu 0* = *0*
  ⟨*proof*⟩

lemma *fds-nth-fds-moebius-mu* [*simp*]: *fds-nth* (*fds moebius-mu*) = *moebius-mu*
  ⟨*proof*⟩

lemma *prime-factors-Suc-0* [*simp*]: *prime-factors* (*Suc 0*) = {}
  ⟨*proof*⟩

lemma *moebius-mu-Suc-0* [*simp*]: *moebius-mu* (*Suc 0*) = *1*
  ⟨*proof*⟩

lemma *moebius-mu-1* [*simp*]: *moebius-mu 1 = 1*
⟨*proof*⟩

lemma *moebius-mu-eq-zero-iff*: *moebius-mu n = 0* ⟷ ¬*squarefree n*
⟨*proof*⟩

lemma *moebius-mu-not-squarefree* [*simp*]: ¬*squarefree n* ⟹ *moebius-mu n = 0*
⟨*proof*⟩

lemma *moebius-mu-power*:
assumes *a > 1 n > 1*
shows *moebius-mu* (*a* ^ *n*) = *0*
⟨*proof*⟩

lemma *moebius-mu-power'*:
moebius-mu (*a* ^ *n*) = (*if a = 1* ∨ *n = 0 then 1 else if n = 1 then moebius-mu a else 0*)
⟨*proof*⟩

lemma *moebius-mu-squarefree-eq*:
squarefree n ⟹ *moebius-mu n* = (−*1*) ^ *card* (*prime-factors n*)
⟨*proof*⟩

lemma *moebius-mu-squarefree-eq'*:
assumes *squarefree n*
shows *moebius-mu n* = (−*1*) ^ *size* (*prime-factorization n*)
⟨*proof*⟩

lemma *sum-moebius-mu-divisors*:
assumes *n > 1*
shows (∑ *d* | *d dvd n. moebius-mu d*) = (*0* :: *'a* :: *comm-ring-1*)
⟨*proof*⟩

lemma *sum-moebius-mu-divisors'*:
(∑ *d* | *d dvd n. moebius-mu d*) = (*if n = 1 then 1 else 0*)
⟨*proof*⟩

lemma *fds-zeta-times-moebius-mu*: *fds-zeta* ∗ *fds moebius-mu = 1*
⟨*proof*⟩

lemma *fds-moebius-inverse-zeta*:
fds moebius-mu = inverse (*fds-zeta* :: *'a* :: *field fds*)
⟨*proof*⟩

lemma *moebius-mu-formula-real*: (*moebius-mu n* :: *real*) = *dirichlet-inverse* (λ-. *1*) *1 n*
⟨*proof*⟩

lemma *moebius-mu-formula-int*: *moebius-mu n = dirichlet-inverse* (λ-. *1* :: *int*) *1*

n
⟨*proof*⟩

lemma *moebius-mu-formula*: *moebius-mu n* = *dirichlet-inverse* (λ-. *1*) *1 n*
⟨*proof*⟩

interpretation *moebius-mu*: *multiplicative-function moebius-mu*
⟨*proof*⟩

interpretation *moebius-mu*:
multiplicative-function′ moebius-mu λ*p k*. *if k* = *1 then* −*1 else 0* λ-. −*1*
⟨*proof*⟩

lemma *moebius-mu-2* [*simp*]: *moebius-mu 2* = −*1*
and *moebius-mu-3* [*simp*]: *moebius-mu 3* = −*1*
⟨*proof*⟩

lemma *moebius-mu-code* [*code*]:
moebius-mu n = *of-int* (*dirichlet-inverse* (λ-. *1* :: *int*) *1 n*)
⟨*proof*⟩

lemma *fds-moebius-inversion*: *f* = *fds moebius-mu* ∗ *g* ⟷ *g* = *f* ∗ *fds-zeta*
⟨*proof*⟩

lemma *moebius-inversion*:
assumes $\bigwedge$*n*. *n* > *0* ⟹ *g n* = (∑ *d* | *d dvd n*. *f d*) *n* > *0*
shows *f n* = *dirichlet-prod moebius-mu g n*
⟨*proof*⟩

lemma *fds-mangoldt*: *fds mangoldt* = *fds moebius-mu* ∗ *fds* (λ*n*. *of-real* (*ln* (*real n*)))
⟨*proof*⟩

lemma *sum-divisors-moebius-mu-times-multiplicative*:
fixes *f* :: *nat* ⇒ ′*a* :: {*comm-ring-1*}
assumes *multiplicative-function f n* > *0*
shows (∑ *d* | *d dvd n*. *moebius-mu d* ∗ *f d*) = (∏ *p*∈*prime-factors n*. *1* − *f p*)
⟨*proof*⟩

lemma *completely-multiplicative-iff-inverse-moebius-mu*:
fixes *f* :: *nat* ⇒ ′*a* :: {*comm-ring-1*, *ring-no-zero-divisors*}
assumes *multiplicative-function f*
defines *g* ≡ *dirichlet-inverse f 1*
shows *completely-multiplicative-function f* ⟷

$(\forall n.\ g\ n = moebius\text{-}mu\ n * f\ n)$
⟨proof⟩

lemma *completely-multiplicative-fds-inverse*:
fixes $f :: nat \Rightarrow 'a :: field$
assumes *completely-multiplicative-function f*
shows $inverse\ (fds\ f) = fds\ (\lambda n.\ moebius\text{-}mu\ n * f\ n)$
⟨proof⟩

lemma *completely-multiplicative-fds-inverse′*:
fixes $f :: 'a :: field\ fds$
assumes *completely-multiplicative-function* (*fds-nth f*)
shows $inverse\ f = fds\ (\lambda n.\ moebius\text{-}mu\ n * fds\text{-}nth\ f\ n)$
⟨proof⟩

context
includes *fds-syntax*
begin

lemma *selberg-aux*:
$(\chi\ n.\ of\text{-}real\ ((ln\ n)^2)) * fds\ moebius\text{-}mu =$
$(fds\ mangoldt)^2 - fds\text{-}deriv\ (fds\ mangoldt :: 'a :: \{comm\text{-}ring\text{-}1,real\text{-}algebra\text{-}1\}$
$fds)$
⟨proof⟩

lemma *selberg-aux′*:
$mangoldt\ n * of\text{-}real\ (ln\ n) + (mangoldt \star mangoldt)\ n =$
$((moebius\text{-}mu \star (\lambda b.\ of\text{-}real\ (ln\ b)\ \hat{}\ 2))\ n$
$:: 'a :: \{comm\text{-}ring\text{-}1,real\text{-}algebra\text{-}1\})$ **if** $n > 0$
⟨proof⟩

end

end

6 Euler's ϕ function

theory *More-Totient*
imports
Moebius-Mu
HOL−Number-Theory.Number-Theory
begin

lemma *fds-totient-times-zeta*:
$fds\ (\lambda n.\ of\text{-}nat\ (totient\ n) :: 'a :: comm\text{-}semiring\text{-}1) * fds\text{-}zeta = fds\ of\text{-}nat$
⟨proof⟩

lemma *fds-totient-times-zeta′*: $fds\ totient * fds\text{-}zeta = fds\ id$

⟨*proof*⟩

lemma *fds-totient*: $fds\ (\lambda n.\ of\text{-}nat\ (totient\ n)) = fds\ of\text{-}nat * fds\ moebius\text{-}mu$
⟨*proof*⟩

lemma *totient-conv-moebius-mu*:
$int\ (totient\ n) = dirichlet\text{-}prod\ moebius\text{-}mu\ int\ n$
⟨*proof*⟩

interpretation *totient*: *multiplicative-function totient*
⟨*proof*⟩

lemma *even-prime-nat*: $prime\ p \Longrightarrow even\ p \Longrightarrow p = (2{::}nat)$
⟨*proof*⟩

lemma *twopow-dvd-totient*:
fixes $n :: nat$
assumes $n > 0$
defines $k \equiv card\ \{p{\in}prime\text{-}factors\ n.\ odd\ p\}$
shows $2\ \widehat{}\ k\ dvd\ totient\ n$
⟨*proof*⟩

lemma *totient-conv-moebius-mu'*:
assumes $n > (0{::}nat)$
shows $real\ (totient\ n) = real\ n * (\sum d \mid d\ dvd\ n.\ moebius\text{-}mu\ d\ /\ real\ d)$
⟨*proof*⟩

lemma *totient-prime-power-Suc*:
assumes *prime p*
shows $totient\ (p\ \widehat{}\ Suc\ n) = p\ \widehat{}\ Suc\ n - p\ \widehat{}\ n$
⟨*proof*⟩

interpretation *totient*: $multiplicative\text{-}function'\ totient\ \lambda p\ k.\ p\ \widehat{}\ k - p\ \widehat{}\ (k - 1)$
$\lambda p.\ p - 1$
⟨*proof*⟩

end

7 The Liouville λ function

theory *Liouville-Lambda*
imports
HOL−Computational-Algebra.Computational-Algebra
HOL−Number-Theory.Number-Theory
Dirichlet-Series
Multiplicative-Function
Moebius-Mu
begin

definition *liouville-lambda* :: *nat* ⇒ *'a* :: *comm-ring-1* **where**
liouville-lambda n = (*if n* = *0 then 0 else* (−*1*) ^ *size* (*prime-factorization n*))

interpretation *liouville-lambda*: *completely-multiplicative-function' liouville-lambda* λ-. −*1*
⟨*proof*⟩

lemma *liouville-lambda-prime* [*simp*]: *prime p* ⟹ *liouville-lambda p* = −*1*
⟨*proof*⟩

lemma *liouville-lambda-prime-power* [*simp*]: *prime p* ⟹ *liouville-lambda* (*p* ^ *k*) = (−*1*) ^ *k*
⟨*proof*⟩

lemma *liouville-lambda-squarefree*: *squarefree n* ⟹ *liouville-lambda n* = *moebius-mu n*
⟨*proof*⟩

lemma *power-neg-one-If*: (−*1*) ^ *n* = (*if even n then 1 else* −*1* :: *'a* :: *ring-1*)
⟨*proof*⟩

lemma *liouville-lambda-power-even*:
n > *0* ⟹ *even m* ⟹ *liouville-lambda* (*n* ^ *m*) = *1*
⟨*proof*⟩

lemma *liouville-lambda-power-odd*:
odd m ⟹ *liouville-lambda* (*n* ^ *m*) = *liouville-lambda n*
⟨*proof*⟩

lemma *liouville-lambda-power*:
liouville-lambda (*n* ^ *m*) =
(*if n* = *0* ∧ *m* > *0 then 0 else if even m then 1 else liouville-lambda n*)
⟨*proof*⟩

interpretation *squarefree*: *multiplicative-function'*
ind squarefree λ*p k*. *if k* > *1 then 0 else 1* λ-. *1*
⟨*proof*⟩

interpretation *is-nth-power*: *multiplicative-function ind* (*is-nth-power n*)
⟨*proof*⟩

interpretation *is-nth-power*: *multiplicative-function'*
ind (*is-nth-power n*) λ*p k*. *if n dvd k then 1 else 0* λ-. *if n* = *1 then 1 else 0*
⟨*proof*⟩

interpretation *is-square*: *multiplicative-function ind is-square*
⟨*proof*⟩

interpretation *is-square*: *multiplicative-function′*
ind is-square λp *k. if even k then 1 else 0* λ*-. 0*
⟨*proof*⟩

lemma *liouville-lambda-divisors-sum*:
($\sum$ *d* | *d dvd n. liouville-lambda d*) = *ind is-square n*
⟨*proof*⟩

lemma *fds-liouville-lambda-times-zeta*: *fds liouville-lambda* ∗ *fds-zeta* = *fds-ind is-square*
⟨*proof*⟩

lemma *fds-liouville-lambda*: *fds liouville-lambda* = *fds-ind is-square* ∗ *fds moebius-mu*
⟨*proof*⟩

lemma *liouville-lambda-altdef*:
liouville-lambda n = ($\sum$ *d* | *d* ^ *2 dvd n. moebius-mu* (*n div d* ^ *2*))
⟨*proof*⟩

lemma *abs-moebius-mu*: *abs* (*moebius-mu n* :: *′a* :: *linordered-idom*) = *ind square-free n*
⟨*proof*⟩

end

8 The divisor functions

theory *Divisor-Count*
imports
Complex-Main
HOL−Number-Theory.Number-Theory
Dirichlet-Series
More-Totient
Moebius-Mu
begin

8.1 The general divisor function

definition *divisor-sigma* :: *′a* :: *nat-power* ⇒ *nat* ⇒ *′a* **where**
divisor-sigma x n = ($\sum$ *d* | *d dvd n. nat-power d x*)

lemma *divisor-sigma-0* [*simp*]: *divisor-sigma x 0* = *0*
⟨*proof*⟩

lemma *divisor-sigma-Suc-0* [*simp*]: *divisor-sigma x* (*Suc 0*) = *1*
⟨*proof*⟩

lemma *divisor-sigma-1* [*simp*]: *divisor-sigma x 1 = 1*
⟨*proof*⟩

lemma *fds-divisor-sigma*: *fds* (*divisor-sigma x*) = *fds-zeta* ∗ *fds-shift x fds-zeta*
⟨*proof*⟩

interpretation *divisor-sigma*: *multiplicative-function divisor-sigma x*
⟨*proof*⟩

lemma *divisor-sigma-naive* [*code*]:
divisor-sigma x n = (*if n = 0 then 0 else fold-atLeastAtMost-nat*
(λ*d acc. if d dvd n then nat-power d x + acc else acc*) *1 n 0*)
⟨*proof*⟩

lemma *divisor-sigma-of-nat*: *divisor-sigma* (*of-nat x*) *n* = *of-nat* (*divisor-sigma x n*)
⟨*proof*⟩

lemma *divisor-sigma-prime-power-field*:
fixes *x* :: *'a* :: {*field*, *nat-power*}
assumes *prime p*
shows *divisor-sigma x* (*p* ^ *k*) =
(*if nat-power p x = 1 then of-nat* (*k* + *1*) *else*
(*nat-power p x* ^ *Suc k* − *1*) / (*nat-power p x* − *1*))
⟨*proof*⟩

lemma *divisor-sigma-prime-power-nat*:
assumes *prime p*
shows *divisor-sigma x* (*p* ^ *k*) = (*if x = 0 then Suc k else*
(*p* ^ (*x* ∗ *Suc k*) − *1*) *div* (*p* ^ *x* − *1*))
⟨*proof*⟩

interpretation *divisor-sigma-field*:
multiplicative-function' divisor-sigma (*x* :: *'a* :: {*field*, *nat-power*})
λ*p k. if nat-power p x = 1 then of-nat* (*Suc k*) *else*
(*nat-power p x* ^ *Suc k* − *1*) / (*nat-power p x* − *1*)
λ*p. nat-power p x* + *1*
⟨*proof*⟩

interpretation *divisor-sigma-real*:
multiplicative-function' divisor-sigma (*x* :: *real*)
λ*p k. if x = 0 then of-nat* (*Suc k*) *else* ((*real p powr x*) ^ *Suc k* − *1*) / (*real p powr x* − *1*)
λ*p. real p powr x* + *1*
⟨*proof*⟩

interpretation *divisor-sigma-nat*:
multiplicative-function' divisor-sigma (*x* :: *nat*)
λ*p k. if x = 0 then Suc k else* (*p* ^ (*Suc k* ∗ *x*) − *1*) *div* (*p* ^ *x* − *1*)

$\lambda p.\ p$ ^ $x + 1$
⟨proof⟩

lemma *divisor-sigma-prime*:
assumes *prime p*
shows *divisor-sigma x p = nat-power p x + 1*
⟨proof⟩

8.2 The divisor-counting function

definition *divisor-count* :: *nat* ⇒ *nat* **where**
divisor-count n = card {d. d dvd n}

lemma *divisor-count-0* [*simp*]: *divisor-count 0 = 0*
⟨proof⟩

lemma *divisor-count-Suc-0* [*simp*]: *divisor-count* (*Suc 0*) = *1*
⟨proof⟩

lemma *divisor-sigma-0-left-nat*: *divisor-sigma 0 n = divisor-count n*
⟨proof⟩

lemma *divisor-sigma-0-left*: *divisor-sigma 0 n = of-nat* (*divisor-count n*)
⟨proof⟩

lemma *divisor-count-altdef*: *divisor-count n = divisor-sigma 0 n*
⟨proof⟩

lemma *divisor-count-naive* [*code*]:
divisor-count n = (*if n = 0 then 0 else*
fold-atLeastAtMost-nat (λ*d acc. if d dvd n then Suc acc else acc*) *1 n 0*)
⟨proof⟩

interpretation *divisor-count*: *multiplicative-function′ divisor-count* λ*p k. Suc k* λ-. *2*
⟨proof⟩

lemma *divisor-count-dvd-mono*:
assumes *a dvd b b* ≠ *0*
shows *divisor-count a* ≤ *divisor-count b*
⟨proof⟩

8.3 The divisor sum function

definition *divisor-sum* :: *nat* ⇒ *nat* **where**
divisor-sum n = $\sum$ {*d. d dvd n*}

lemma *divisor-sum-0* [*simp*]: *divisor-sum 0 = 0*
⟨proof⟩

lemma *divisor-sum-Suc-0* [*simp*]: *divisor-sum* (*Suc 0*) = *Suc 0*
⟨*proof*⟩

lemma *divisor-sigma-1-left-nat*: *divisor-sigma* (*Suc 0*) *n* = *divisor-sum n*
⟨*proof*⟩

lemma *divisor-sigma-1-left*: *divisor-sigma 1 n* = *of-nat* (*divisor-sum n*)
⟨*proof*⟩

lemma *divisor-sum-altdef*: *divisor-sum n* = *divisor-sigma 1 n*
⟨*proof*⟩

interpretation *divisor-sum*:
multiplicative-function′ divisor-sum λp k. (*p* ^ *Suc k* − *1*) *div* (*p* − *1*) *λp. Suc p*
⟨*proof*⟩

lemma *divisor-sum-dvd-mono*:
assumes *a dvd b b* ≠ *0*
shows *divisor-sum a* ≤ *divisor-sum b*
⟨*proof*⟩

lemma *divisor-sum-naive* [*code*]:
divisor-sum n = (*if n* = *0 then 0 else*
fold-atLeastAtMost-nat (*λd acc. if d dvd n then d* + *acc else acc*) *1 n 0*)
⟨*proof*⟩

lemma *fds-divisor-count*: *fds divisor-count* = *fds-zeta* ^ *2*
⟨*proof*⟩

lemma *fds-shift-zeta-1*: *fds-shift 1 fds-zeta* = *fds of-nat*
⟨*proof*⟩

lemma *fds-shift-zeta-Suc-0*: *fds-shift* (*Suc 0*) *fds-zeta* = *fds id*
⟨*proof*⟩

lemma *fds-divisor-sum*: *fds divisor-sum* = *fds-zeta* ∗ *fds id*
⟨*proof*⟩

lemma *fds-divisor-sum-eq-totient-times-d*: *fds divisor-sum* = *fds totient* ∗ *fds divisor-count*
⟨*proof*⟩

lemma *fds-divisor-sum-times-moebius-mu*:
fds (*divisor-sigma* (*1* :: ′*a* :: {*nat-power*,*comm-ring-1*})) ∗ *fds moebius-mu* = *fds of-nat*
⟨*proof*⟩

lemma *inverse-divisor-sigma*:
fixes *a* :: *'a* :: {*field*, *nat-power*}
shows *inverse* (*fds* (*divisor-sigma a*)) = *fds-shift a* (*fds moebius-mu*) $*$ *fds moebius-mu*
⟨*proof*⟩

end

9 Summatory arithmetic functions

theory *Arithmetic-Summatory*
imports
More-Totient
Moebius-Mu
Liouville-Lambda
Divisor-Count
Dirichlet-Series
begin

9.1 Definition

definition *sum-upto* :: (*nat* ⇒ *'a* :: *comm-monoid-add*) ⇒ *real* ⇒ *'a* **where**
sum-upto f x = ($\sum$ *i* | *0* < *i* ∧ *real i* ≤ *x*. *f i*)

lemma *sum-upto-altdef*: *sum-upto f x* = ($\sum$ *i*∈{*0*<..*nat* ⌊*x*⌋}. *f i*)
⟨*proof*⟩

lemma *sum-upto-0* [*simp*]: *sum-upto f 0* = *0*
⟨*proof*⟩

lemma *sum-upto-cong* [*cong*]:
(⋀*n*. *n* > *0* ⟹ *f n* = *f' n*) ⟹ *n* = *n'* ⟹ *sum-upto f n* = *sum-upto f' n'*
⟨*proof*⟩

lemma *finite-Nats-le-real* [*simp*,*intro*]: *finite* {*n*. *0* < *n* ∧ *real n* ≤ *x*}
⟨*proof*⟩

lemma *sum-upto-ind*: *sum-upto* (*ind P*) *x* = *of-nat* (*card* {*n*. *n* > *0* ∧ *real n* ≤ *x* ∧ *P n*})
⟨*proof*⟩

lemma *sum-upto-sum-divisors*:
sum-upto (λ*n*. $\sum$ *d* | *d dvd n*. *f n d*) *x* = *sum-upto* (λ*k*. *sum-upto* (λ*d*. *f* (*d* $*$ *k*) *k*) (*x* / *k*)) *x*
⟨*proof*⟩

lemma *sum-upto-dirichlet-prod*:
sum-upto (*dirichlet-prod f g*) *x* = *sum-upto* (λ*d*. *f d* $*$ *sum-upto g* (*x* / *real d*)) *x*

⟨proof⟩

lemma *sum-upto-real*:
assumes $x \geq 0$
shows *sum-upto real x = of-int (floor x) ∗ (of-int (floor x) + 1) / 2*
⟨proof⟩

lemma *summable-imp-convergent-sum-upto*:
assumes *summable (f :: nat ⇒ 'a :: real-normed-vector)*
obtains *c* **where** *(sum-upto f ⟶ c) at-top*
⟨proof⟩

9.2 The Hyperbola method

lemma *hyperbola-method-semiring*:
fixes *f g :: nat ⇒ 'a :: comm-semiring-0*
assumes $A \geq 0$ **and** $B \geq 0$ **and** $A * B = x$
shows *sum-upto (dirichlet-prod f g) x + sum-upto f A ∗ sum-upto g B =*
sum-upto (λn. f n ∗ sum-upto g (x / real n)) A +
sum-upto (λn. sum-upto f (x / real n) ∗ g n) B
⟨proof⟩

lemma *hyperbola-method-semiring-sqrt*:
fixes *f g :: nat ⇒ 'a :: comm-semiring-0*
assumes $x \geq 0$
shows *sum-upto (dirichlet-prod f g) x + sum-upto f (sqrt x) ∗ sum-upto g (sqrt x) =*
sum-upto (λn. f n ∗ sum-upto g (x / real n)) (sqrt x) +
sum-upto (λn. sum-upto f (x / real n) ∗ g n) (sqrt x)
⟨proof⟩

lemma *hyperbola-method*:
fixes *f g :: nat ⇒ 'a :: comm-ring*
assumes $A \geq 0$ $B \geq 0$ $A * B = x$
shows *sum-upto (dirichlet-prod f g) x =*
sum-upto (λn. f n ∗ sum-upto g (x / real n)) A +
sum-upto (λn. sum-upto f (x / real n) ∗ g n) B −
sum-upto f A ∗ sum-upto g B
⟨proof⟩

lemma *hyperbola-method-sqrt*:
fixes *f g :: nat ⇒ 'a :: comm-ring*
assumes $x \geq 0$
shows *sum-upto (dirichlet-prod f g) x =*
sum-upto (λn. f n ∗ sum-upto g (x / real n)) (sqrt x) +
sum-upto (λn. sum-upto f (x / real n) ∗ g n) (sqrt x) −
sum-upto f (sqrt x) ∗ sum-upto g (sqrt x)
⟨proof⟩

end

10 Partial summation

theory *Partial-Summation*
  imports
    HOL−Analysis.Analysis
    Arithmetic-Summatory
begin

lemma *finite-vimage-real-of-nat-greaterThanAtMost*: $finite\ (real\ -` \{y<..x\})$
⟨*proof*⟩

context
  fixes $a :: nat \Rightarrow 'a :: \{banach,\ real\text{-}normed\text{-}algebra\}$
  fixes $f\ f' :: real \Rightarrow 'a$
  fixes A
  fixes $X :: real\ set$
  fixes $x\ y :: real$
  defines $A \equiv sum\text{-}upto\ a$
  assumes *fin*: $finite\ X$
  assumes *xy*: $0 \le y\ y < x$
  assumes *deriv*: $\bigwedge z.\ z \in \{y..x\} - X \Longrightarrow (f\ has\text{-}vector\text{-}derivative\ f'\ z)\ (at\ z)$
  assumes *cont-f*: $continuous\text{-}on\ \{y..x\}\ f$
begin

lemma *partial-summation-strong*:
  $((\lambda t.\ A\ t * f'\ t)\ has\text{-}integral$
    $(A\ x * f\ x - A\ y * f\ y - (\sum n \in real\ -` \{y<..x\}.\ a\ n * f\ n)))\ \{y..x\}$
⟨*proof*⟩

lemma *partial-summation-integrable-strong*:
    $(\lambda t.\ A\ t * f'\ t)\ integrable\text{-}on\ \{y..x\}$
  and *partial-summation-strong'*:
    $(\sum n{\in}real\ -` \{y<..x\}.\ a\ n * f\ n) =$
      $A\ x * f\ x - A\ y * f\ y - integral\ \{y..x\}\ (\lambda t.\ A\ t * f'\ t)$
  ⟨*proof*⟩

end

context
  fixes $a :: nat \Rightarrow 'a :: \{banach,\ real\text{-}normed\text{-}algebra\}$
  fixes $f\ f' :: real \Rightarrow 'a$
  fixes A
  fixes $X :: real\ set$
  fixes $x :: real$
  defines $A \equiv sum\text{-}upto\ a$
  assumes *fin*: $finite\ X$

assumes x: $x > 0$
assumes *deriv*: $\bigwedge z.\ z \in \{0..x\} - X \Longrightarrow (f\ \textit{has-vector-derivative}\ f'\ z)\ (\textit{at}\ z)$
assumes *cont-f*: *continuous-on* $\{0..x\}\ f$
begin

lemma *partial-summation-sum-upto-strong*:
$((\lambda t.\ A\ t * f'\ t)\ \textit{has-integral}\ (A\ x * f\ x - \textit{sum-upto}\ (\lambda n.\ a\ n * f\ n)\ x))\ \{0..x\}$
⟨*proof*⟩

lemma *partial-summation-integrable-sum-upto-strong*:
$(\lambda t.\ A\ t * f'\ t)\ \textit{integrable-on}\ \{0..x\}$
and *partial-summation-sum-upto-strong′*:
$\textit{sum-upto}\ (\lambda n.\ a\ n * f\ n)\ x =$
$A\ x * f\ x - \textit{integral}\ \{0..x\}\ (\lambda t.\ A\ t * f'\ t)$
⟨*proof*⟩

end

end

11 Euler product expansions

theory *Euler-Products*
imports
HOL−Analysis.Analysis
Multiplicative-Function
begin

Conflicting notation from *HOL−Analysis.Infinite-Sum*

no-notation *Infinite-Sum.abs-summable-on* (**infixr** ‹*abs′-summable′-on*› *46*)

lemma *prime-factors-power-subset*:
$\textit{prime-factors}\ (x\ \hat{}\ n) \subseteq \textit{prime-factors}\ x$
⟨*proof*⟩

lemma *prime-power-product-in-Pi*:
$(\lambda g.\ \prod p \in \{p.\ p \leq (n::nat) \wedge \textit{prime}\ p\}.\ p\ \hat{}\ g\ p)$
$\in (\{p.\ p \leq n \wedge \textit{prime}\ p\} \rightarrow_E \textit{UNIV}) \rightarrow$
$\{m.\ 0 < m \wedge \textit{prime-factors}\ m \subseteq \{..n\}\}$
⟨*proof*⟩

lemma *inj-prime-power*: $\textit{inj-on}\ (\lambda x.\ \textit{fst}\ x\ \hat{}\ \textit{snd}\ x :: nat)\ (\{a.\ \textit{prime}\ a\} \times \{0{<}..\})$
⟨*proof*⟩

lemma *bij-betw-prime-powers*:
$\textit{bij-betw}\ (\lambda g.\ \prod p \in \{p.\ p \leq n \wedge \textit{prime}\ p\}.\ p\ \hat{}\ g\ p)\ (\{p.\ p \leq n \wedge \textit{prime}\ p\} \rightarrow_E \textit{UNIV})$
$\{m.\ 0 < m \wedge \textit{prime-factors}\ m \subseteq \{..(n::nat)\}\}$
⟨*proof*⟩

lemma
fixes f :: $nat \Rightarrow 'a$ :: $\{real\text{-}normed\text{-}field, banach, second\text{-}countable\text{-}topology\}$
assumes *summable*: *summable* ($\lambda n.$ *norm* (f n))
assumes *multiplicative-function* f
shows *abs-convergent-euler-product*:
abs-convergent-prod ($\lambda p.$ *if prime p then* $\sum n.$ f (p ^ n) *else 1*)
and *euler-product-LIMSEQ*:
($\lambda n.$ ($\prod p{\le}n.$ *if prime p then* $\sum n.$ f (p ^ n) *else 1*)) $\longrightarrow$ ($\sum n.$ f n)
⟨*proof*⟩

lemma
fixes f :: $nat \Rightarrow 'a$ :: $\{real\text{-}normed\text{-}field, banach, second\text{-}countable\text{-}topology\}$
assumes *summable*: *summable* ($\lambda n.$ *norm* (f n))
assumes *completely-multiplicative-function* f
shows *abs-convergent-euler-product'*:
abs-convergent-prod ($\lambda p.$ *if prime p then inverse* ($1 - f\ p$) *else 1*)
and *completely-multiplicative-summable-norm*:
$\bigwedge p.$ *prime p* $\Longrightarrow$ *norm* (f p) < 1
and *euler-product-LIMSEQ'*:
($\lambda n.$ ($\prod p{\le}n.$ *if prime p then inverse* ($1 - f\ p$) *else 1*)) $\longrightarrow$ ($\sum n.$ f
n)
⟨*proof*⟩

end

12 Analytic properties of Dirichlet series

theory *Dirichlet-Series-Analysis*
imports
HOL−Complex-Analysis.Complex-Analysis
HOL−Library.Going-To-Filter
HOL−Real-Asymp.Real-Asymp
Dirichlet-Series
Moebius-Mu
Partial-Summation
Euler-Products
begin

Conflicting notation from *HOL−Analysis.Infinite-Sum*

no-notation *Infinite-Sum.abs-summable-on* (**infixr** ‹*abs'-summable'-on*› *46*)

The following illustrates a concept we will need later on: A property holds for f going to F if we can find e. g. a sequence that tends to F and whose elements eventually satisfy P.

lemma *frequently-going-toI*:
assumes *filterlim* ($\lambda n.$ f (g n)) F G
assumes *eventually* ($\lambda n.$ P (g n)) G

assumes *eventually* $(\lambda n.\ g\ n \in A)\ G$
assumes $G \neq bot$
shows *frequently* P $(f$ *going-to* F *within* $A)$
⟨*proof*⟩

lemma *frequently-filtercomapI*:
assumes *filterlim* $(\lambda n.\ f\ (g\ n))\ F\ G$
assumes *eventually* $(\lambda n.\ P\ (g\ n))\ G$
assumes $G \neq bot$
shows *frequently* P (*filtercomap* f F)
⟨*proof*⟩

lemma *frequently-going-to-at-topE*:
fixes $f :: 'a \Rightarrow real$
assumes *frequently* P $(f$ *going-to at-top*)
obtains g **where** $\bigwedge n.\ P\ (g\ n)$ **and** *filterlim* $(\lambda n.\ f\ (g\ n))$ *at-top sequentially*
⟨*proof*⟩

Apostol often uses statements like '$P(s_k)$ for all k in an infinite sequence s_k such that $\Re(s_k) \longrightarrow \infty$ as $k \to \infty$'.

Instead, we write *frequently* P (*Re going-to at-top*). This lemma shows that our statement is equivalent to his.

lemma *frequently-going-to-at-top-iff*:
frequently P $(f$ *going-to* (*at-top* :: *real filter*)) $\longleftrightarrow$
$(\exists\, g.\ \forall\, n.\ P\ (g\ n) \wedge$ *filterlim* $(\lambda n.\ f\ (g\ n))$ *at-top sequentially*)
⟨*proof*⟩

lemma *surj-bullet-1*: *surj* $(\lambda s::'a::\{$*real-normed-algebra-1*, *real-inner*$\}.\ s \cdot 1)$
⟨*proof*⟩

lemma *bullet-1-going-to-at-top-neq-bot* [*simp*]:
$((\lambda s::'a::\{$*real-normed-algebra-1*, *real-inner*$\}.\ s \cdot 1)$ *going-to at-top*$) \neq bot$
⟨*proof*⟩

lemma *fds-abs-converges-altdef*:
fds-abs-converges f $s \longleftrightarrow (\lambda n.$ *fds-nth* f n / *nat-power* n $s)$ *abs-summable-on* $\{1..\}$
⟨*proof*⟩

lemma *fds-abs-converges-altdef'*:
fds-abs-converges f $s \longleftrightarrow (\lambda n.$ *fds-nth* f n / *nat-power* n $s)$ *abs-summable-on* *UNIV*
⟨*proof*⟩

lemma *eval-fds-altdef*:
assumes *fds-abs-converges* f s
shows *eval-fds* f $s = (\sum_a n.$ *fds-nth* f n / *nat-power* n $s)$

⟨*proof*⟩

lemma *multiplicative-function-divide-nat-power*:
  fixes *f* :: *nat* ⇒ *'a* :: {*nat-power*, *field*}
  assumes *multiplicative-function f*
  shows *multiplicative-function* (*λn. f n* / *nat-power n s*)
⟨*proof*⟩

lemma *completely-multiplicative-function-divide-nat-power*:
  fixes *f* :: *nat* ⇒ *'a* :: {*nat-power*, *field*}
  assumes *completely-multiplicative-function f*
  shows *completely-multiplicative-function* (*λn. f n* / *nat-power n s*)
⟨*proof*⟩

12.1 Convergence and absolute convergence

class *nat-power-normed-field* = *nat-power-field* + *real-normed-field* + *real-inner* + *real-algebra-1* +
  fixes *real-power* :: *real* ⇒ *'a* ⇒ *'a*
  assumes *real-power-nat-power*: *n* > *0* ⟹ *real-power* (*real n*) *c* = *nat-power n c*
  assumes *real-power-1-right-aux*: *d* > *0* ⟹ *real-power d 1* = *d* $*_R$ *1*
  assumes *real-power-add*: *d* > *0* ⟹ *real-power d* (*a* + *b*) = *real-power d a* ∗ *real-power d b*
  assumes *real-power-nonzero* [*simp*]: *d* > *0* ⟹ *real-power d a* ≠ *0*
  assumes *norm-real-power*: *x* > *0* ⟹ *norm* (*real-power x c*) = *x powr* (*c* • *1*)
  assumes *nat-power-of-real-aux*: *nat-power n* (*x* $*_R$ *1*) = ((*real n powr x*) $*_R$ *1*)
  assumes *has-field-derivative-nat-power-aux*:
    ⋀*x*::*'a*. *n* > *0* ⟹ *LIM y inf-class.inf*
      (*Inf* (*principal* ‘ {*S*. *open S* ∧ *x* ∈ *S*})) (*principal* (*UNIV* − {*x*})).
        (*nat-power n y* − *nat-power n x* − *ln* (*real n*) $*_R$ *nat-power n x* ∗ (*y* − *x*)) $/_R$
        norm (*y* − *x*) :> *Inf* (*principal* ‘ {*S*. *open S* ∧ *0* ∈ *S*})
  assumes *has-vector-derivative-real-power-aux*:
    x > *0* ⟹ *filterlim* (*λy*. (*real-power y c* − *real-power x* (*c* :: *'a*) −
      (*y* − *x*) $*_R$ (*c* ∗ *real-power x* (*c* − *1*))) $/_R$
      norm (*y* − *x*)) (*INF S*∈{*S*. *open S* ∧ *0* ∈ *S*}. *principal S*) (*at x*)
  assumes *norm-nat-power*: *n* > *0* ⟹ *norm* (*nat-power n y*) = *real n powr* (*y* • *1*)
begin

lemma *real-power-diff*: *d* > *0* ⟹ *real-power d* (*a* − *b*) = *real-power d a* / *real-power d b*
  ⟨*proof*⟩

end

lemma *real-power-1-right* [*simp*]: *d* > *0* ⟹ *real-power d 1* = *of-real d*
  ⟨*proof*⟩

lemma *has-vector-derivative-real-power* [*derivative-intros*]:
$x > 0 \Longrightarrow ((\lambda y.\ real\text{-}power\ y\ c)\ has\text{-}vector\text{-}derivative\ c * real\text{-}power\ x\ (c - 1))$ $(at\ x\ within\ A)$
⟨*proof*⟩

lemma *has-field-derivative-nat-power* [*derivative-intros*]:
$n > 0 \Longrightarrow ((\lambda y.\ nat\text{-}power\ n\ y)\ has\text{-}field\text{-}derivative\ ln\ (real\ n) *_R nat\text{-}power\ n\ x)$
$(at\ (x :: 'a :: nat\text{-}power\text{-}normed\text{-}field)\ within\ A)$
⟨*proof*⟩

lemma *continuous-on-real-power* [*continuous-intros*]:
$A \subseteq \{0<..\} \Longrightarrow continuous\text{-}on\ A\ (\lambda x.\ real\text{-}power\ x\ s)$
⟨*proof*⟩

instantiation *real* :: *nat-power-normed-field*
begin

definition *real-power-real* :: *real* ⇒ *real* ⇒ *real* **where**
[*simp*]: *real-power-real* = (*powr*)

instance ⟨*proof*⟩

end

instantiation *complex* :: *nat-power-normed-field*
begin

definition *nat-power-complex* :: *nat* ⇒ *complex* ⇒ *complex* **where**
[*simp*]: *nat-power-complex n z* = *of-nat n powr z*

definition *real-power-complex* :: *real* ⇒ *complex* ⇒ *complex* **where**
[*simp*]: *real-power-complex* = (λ*x y. of-real x powr y*)

instance ⟨*proof*⟩

end

lemma *nat-power-of-real* [*simp*]:
nat-power n (*of-real x* :: *'a* :: *nat-power-normed-field*) = *of-real* (*real n powr x*)
⟨*proof*⟩

lemma *fds-abs-converges-of-real* [*simp*]:
fds-abs-converges (*fds-of-real f*)
(*of-real s* :: *'a* :: {*nat-power-normed-field*,*banach*}) ⟷ *fds-abs-converges f s*
⟨*proof*⟩

lemma *eval-fds-of-real* [*simp*]:

assumes *fds-converges f s*
shows *eval-fds* (*fds-of-real f*) (*of-real s* :: *'a* :: {*nat-power-normed-field*,*banach*}) =
of-real (*eval-fds f s*)
⟨*proof*⟩

lemma *fds-abs-summable-zeta-iff* [*simp*]:
fixes *s* :: *'a* :: {*banach*, *nat-power-normed-field*}
shows *fds-abs-converges fds-zeta s* ⟷ *s* · *1* > (*1* :: *real*)
⟨*proof*⟩

lemma *fds-abs-summable-zeta*:
(*s* :: *'a* :: {*banach*, *nat-power-normed-field*}) · *1* > *1* ⟹ *fds-abs-converges fds-zeta s*
⟨*proof*⟩

lemma *fds-abs-converges-moebius-mu*:
fixes *s* :: *'a* :: {*banach*,*nat-power-normed-field*}
assumes *s* · *1* > *1*
shows *fds-abs-converges* (*fds moebius-mu*) *s*
⟨*proof*⟩

definition *conv-abscissa*
:: *'a* :: {*nat-power*,*banach*,*real-normed-field*, *real-inner*} *fds* ⇒ *ereal* **where**
conv-abscissa f = (*INF s*∈{*s*. *fds-converges f s*}. *ereal* (*s* · *1*))

definition *abs-conv-abscissa*
:: *'a* :: {*nat-power*,*banach*,*real-normed-field*, *real-inner*} *fds* ⇒ *ereal* **where**
abs-conv-abscissa f = (*INF s*∈{*s*. *fds-abs-converges f s*}. *ereal* (*s* · *1*))

lemma *conv-abscissa-mono*:
assumes ⋀*s*. *fds-converges g s* ⟹ *fds-converges f s*
shows *conv-abscissa f* ≤ *conv-abscissa g*
⟨*proof*⟩

lemma *abs-conv-abscissa-mono*:
assumes ⋀*s*. *fds-abs-converges g s* ⟹ *fds-abs-converges f s*
shows *abs-conv-abscissa f* ≤ *abs-conv-abscissa g*
⟨*proof*⟩

class *dirichlet-series* = *euclidean-space* + *real-normed-field* + *nat-power-normed-field* +
assumes *one-in-Basis*: *1* ∈ *Basis*

instance *real* :: *dirichlet-series* ⟨*proof*⟩
instance *complex* :: *dirichlet-series* ⟨*proof*⟩

context
  assumes *SORT-CONSTRAINT*(*'a* :: *dirichlet-series*)
begin

lemma *fds-abs-converges-Re-le*:
  fixes *f* :: *'a fds*
  assumes *fds-abs-converges f z z* · *1* ≤ *z'* · *1*
  shows *fds-abs-converges f z'*
  ⟨*proof*⟩

lemma *fds-abs-converges*:
  assumes *s* · *1* > *abs-conv-abscissa* (*f* :: *'a fds*)
  shows *fds-abs-converges f s*
⟨*proof*⟩

lemma *fds-abs-diverges*:
  assumes *s* · *1* < *abs-conv-abscissa* (*f* :: *'a fds*)
  shows ¬*fds-abs-converges f s*
⟨*proof*⟩

lemma *uniformly-Cauchy-eval-fds-aux*:
  fixes *s0* :: *'a* :: *dirichlet-series*
  assumes *bounded*: *Bseq* (*λn.* ∑*k*≤*n. fds-nth f k* / *nat-power k s0*)
  assumes *B*: *compact B* ⋀*z. z* ∈ *B* ⟹ *z* · *1* > *s0* · *1*
  shows *uniformly-Cauchy-on B* (*λN z.* ∑*n*≤*N. fds-nth f n* / *nat-power n z*)
⟨*proof*⟩

lemma *uniformly-convergent-eval-fds-aux*:
  assumes *Bseq* (*λn.* ∑*k*≤*n. fds-nth f k* / *nat-power k* (*s0* :: *'a*))
  assumes *B*: *compact B* ⋀*z. z* ∈ *B* ⟹ *z* · *1* > *s0* · *1*
  shows *uniformly-convergent-on B* (*λN z.* ∑*n*≤*N. fds-nth f n* / *nat-power n z*)
  ⟨*proof*⟩

lemma *uniformly-convergent-eval-fds-aux'*:
  assumes *conv*: *fds-converges f* (*s0* :: *'a*)
  assumes *B*: *compact B* ⋀*z. z* ∈ *B* ⟹ *z* · *1* > *s0* · *1*
  shows *uniformly-convergent-on B* (*λN z.* ∑*n*≤*N. fds-nth f n* / *nat-power n z*)
⟨*proof*⟩

lemma *bounded-partial-sums-imp-fps-converges*:
  fixes *s0* :: *'a* :: *dirichlet-series*
  assumes *Bseq* (*λn.* ∑*k*≤*n. fds-nth f k* / *nat-power k s0*) **and** *s* · *1* > *s0* · *1*
  shows *fds-converges f s*
⟨*proof*⟩

theorem *fds-converges-Re-le*:
  assumes *fds-converges f* (*s0* :: *'a*) *s* · *1* > *s0* · *1*

shows *fds-converges f s*
⟨*proof*⟩

lemma *fds-converges*:
assumes $s \cdot 1 > \textit{conv-abscissa}\ (f :: {}'a\ \textit{fds})$
shows *fds-converges f s*
⟨*proof*⟩

lemma *fds-diverges*:
assumes $s \cdot 1 < \textit{conv-abscissa}\ (f :: {}'a\ \textit{fds})$
shows $\neg$*fds-converges f s*
⟨*proof*⟩

theorem *fds-converges-imp-abs-converges*:
assumes *fds-converges* $(f :: {}'a\ \textit{fds})\ s\ s' \cdot 1 > s \cdot 1 + 1$
shows *fds-abs-converges f s'*
⟨*proof*⟩

lemma *conv-le-abs-conv-abscissa*: *conv-abscissa f* $\leq$ *abs-conv-abscissa f*
⟨*proof*⟩

lemma *conv-abscissa-PInf-iff*: *conv-abscissa f* $= \infty \longleftrightarrow (\forall\ s.\ \neg$*fds-converges f s*$)$
⟨*proof*⟩

lemma *conv-abscissa-PInfI* [*intro*]: $(\bigwedge s.\ \neg$*fds-converges f s*$) \Longrightarrow$ *conv-abscissa f* $= \infty$
⟨*proof*⟩

lemma *conv-abscissa-MInf-iff*: *conv-abscissa* $(f :: {}'a\ \textit{fds}) = -\infty \longleftrightarrow (\forall\ s.$ *fds-converges f s*$)$
⟨*proof*⟩

lemma *conv-abscissa-MInfI* [*intro*]: $(\bigwedge s.$ *fds-converges* $(f::{}'a\ \textit{fds})\ s) \Longrightarrow$ *conv-abscissa f* $= -\infty$
⟨*proof*⟩

lemma *abs-conv-abscissa-PInf-iff*: *abs-conv-abscissa f* $= \infty \longleftrightarrow (\forall\ s.\ \neg$*fds-abs-converges f s*$)$
⟨*proof*⟩

lemma *abs-conv-abscissa-PInfI* [*intro*]: $(\bigwedge s.\ \neg$*fds-converges f s*$) \Longrightarrow$ *abs-conv-abscissa f* $= \infty$
⟨*proof*⟩

lemma *abs-conv-abscissa-MInf-iff*:
abs-conv-abscissa $(f :: {}'a\ \textit{fds}) = -\infty \longleftrightarrow (\forall\ s.$ *fds-abs-converges f s*$)$
⟨*proof*⟩

lemma *abs-conv-abscissa-MInfI* [*intro*]:

$(\bigwedge s.\ \textit{fds-abs-converges}\ (f{::}'a\ \textit{fds})\ s) \Longrightarrow \textit{abs-conv-abscissa}\ f = -\infty$
⟨*proof*⟩

lemma *conv-abscissa-geI*:
assumes $\bigwedge c'.\ \textit{ereal}\ c' < c \Longrightarrow \exists s.\ s \cdot 1 = c' \wedge \neg \textit{fds-converges}\ f\ s$
shows $\textit{conv-abscissa}\ (f :: 'a\ \textit{fds}) \geq c$
⟨*proof*⟩

lemma *conv-abscissa-leI*:
assumes $\bigwedge c'.\ \textit{ereal}\ c' > c \Longrightarrow \exists s.\ s \cdot 1 = c' \wedge \textit{fds-converges}\ f\ s$
shows $\textit{conv-abscissa}\ (f :: 'a\ \textit{fds}) \leq c$
⟨*proof*⟩

lemma *abs-conv-abscissa-geI*:
assumes $\bigwedge c'.\ \textit{ereal}\ c' < c \Longrightarrow \exists s.\ s \cdot 1 = c' \wedge \neg \textit{fds-abs-converges}\ f\ s$
shows $\textit{abs-conv-abscissa}\ (f :: 'a\ \textit{fds}) \geq c$
⟨*proof*⟩

lemma *abs-conv-abscissa-leI*:
assumes $\bigwedge c'.\ \textit{ereal}\ c' > c \Longrightarrow \exists s.\ s \cdot 1 = c' \wedge \textit{fds-abs-converges}\ f\ s$
shows $\textit{abs-conv-abscissa}\ (f :: 'a\ \textit{fds}) \leq c$
⟨*proof*⟩

lemma *conv-abscissa-leI-weak*:
assumes $\bigwedge x.\ \textit{ereal}\ x > d \Longrightarrow \textit{fds-converges}\ f\ (\textit{of-real}\ x)$
shows $\textit{conv-abscissa}\ (f :: 'a\ \textit{fds}) \leq d$
⟨*proof*⟩

lemma *abs-conv-abscissa-leI-weak*:
assumes $\bigwedge x.\ \textit{ereal}\ x > d \Longrightarrow \textit{fds-abs-converges}\ f\ (\textit{of-real}\ x)$
shows $\textit{abs-conv-abscissa}\ (f :: 'a\ \textit{fds}) \leq d$
⟨*proof*⟩

lemma *conv-abscissa-truncate* [*simp*]:
$\textit{conv-abscissa}\ (\textit{fds-truncate}\ m\ (f :: 'a\ \textit{fds})) = -\infty$
⟨*proof*⟩

lemma *abs-conv-abscissa-truncate* [*simp*]:
$\textit{abs-conv-abscissa}\ (\textit{fds-truncate}\ m\ (f :: 'a\ \textit{fds})) = -\infty$
⟨*proof*⟩

theorem *abs-conv-le-conv-abscissa-plus-1*: $\textit{abs-conv-abscissa}\ (f :: 'a\ \textit{fds}) \leq \textit{conv-abscissa}\ f + 1$
⟨*proof*⟩

lemma *uniformly-convergent-eval-fds*:
assumes B: $\textit{compact}\ B\ \bigwedge z.\ z \in B \Longrightarrow z \cdot 1 > \textit{conv-abscissa}\ (f :: 'a\ \textit{fds})$

shows *uniformly-convergent-on B* (λN *z*. $\sum$ *n*≤*N*. *fds-nth f n* / *nat-power n z*)
⟨*proof*⟩

corollary *uniformly-convergent-eval-fds'*:
assumes *B*: *compact B* $\bigwedge$*z*. *z* ∈ *B* ⟹ *z* · *1* > *conv-abscissa* (*f* :: *'a fds*)
shows *uniformly-convergent-on B* (λN *z*. $\sum$ *n*<*N*. *fds-nth f n* / *nat-power n z*)
⟨*proof*⟩

12.2 Derivative of a Dirichlet series

lemma *fds-converges-deriv-aux*:
assumes *conv*: *fds-converges f* (*s0* :: *'a*) **and** *gt*: *s* · *1* > *s0* · *1*
shows *fds-converges* (*fds-deriv f*) *s*
⟨*proof*⟩

theorem
assumes *s* · *1* > *conv-abscissa* (*f* :: *'a fds*)
shows *fds-converges-deriv*: *fds-converges* (*fds-deriv f*) *s*
and *has-field-derivative-eval-fds* [*derivative-intros*]:
(*eval-fds f has-field-derivative eval-fds* (*fds-deriv f*) *s*) (*at s within A*)
⟨*proof*⟩

lemmas *has-field-derivative-eval-fds'* [*derivative-intros*] =
DERIV-chain2[*OF has-field-derivative-eval-fds*]

lemma *continuous-eval-fds* [*continuous-intros*]:
assumes *s* · *1* > *conv-abscissa f*
shows *continuous* (*at s within A*) (*eval-fds* (*f* :: *'a* :: *dirichlet-series fds*))
⟨*proof*⟩

lemma *continuous-eval-fds'* [*continuous-intros*]:
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *continuous* (*at s within A*) *g g s* · *1* > *conv-abscissa f*
shows *continuous* (*at s within A*) (λx. *eval-fds f* (*g x*))
⟨*proof*⟩

lemma *continuous-on-eval-fds* [*continuous-intros*]:
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *A* ⊆ {*s*. *s* · *1* > *conv-abscissa f*}
shows *continuous-on A* (*eval-fds f*)
⟨*proof*⟩

lemma *continuous-on-eval-fds'* [*continuous-intros*]:
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *continuous-on A g g* ‘ *A* ⊆ {*s*. *s* · *1* > *conv-abscissa f*}
shows *continuous-on A* (λx. *eval-fds f* (*g x*))
⟨*proof*⟩

lemma *conv-abscissa-deriv-le*:

fixes *f* :: *'a fds*
shows *conv-abscissa* (*fds-deriv f*) ≤ *conv-abscissa f*
⟨*proof*⟩

lemma *abs-conv-abscissa-integral*:
fixes *f* :: *'a fds*
shows *abs-conv-abscissa* (*fds-integral a f*) = *abs-conv-abscissa f*
⟨*proof*⟩

lemma *abs-conv-abscissa-ln*:
abs-conv-abscissa (*fds-ln l* (*f* :: *'a* :: *dirichlet-series fds*)) =
abs-conv-abscissa (*fds-deriv f* / *f*)
⟨*proof*⟩

lemma *abs-conv-abscissa-deriv*:
fixes *f* :: *'a fds*
shows *abs-conv-abscissa* (*fds-deriv f*) = *abs-conv-abscissa f*
⟨*proof*⟩

lemma *abs-conv-abscissa-higher-deriv*:
abs-conv-abscissa ((*fds-deriv* ^^ *n*) *f*) = *abs-conv-abscissa* (*f* :: *'a* :: *dirichlet-series fds*)
⟨*proof*⟩

lemma *conv-abscissa-higher-deriv-le*:
conv-abscissa ((*fds-deriv* ^^ *n*) *f*) ≤ *conv-abscissa* (*f* :: *'a* :: *dirichlet-series fds*)
⟨*proof*⟩

lemma *abs-conv-abscissa-restrict*:
abs-conv-abscissa (*fds-subseries P f*) ≤ *abs-conv-abscissa f*
⟨*proof*⟩

lemma *eval-fds-deriv*:
fixes *f* :: *'a fds*
assumes *s* · *1* > *conv-abscissa f*
shows *eval-fds* (*fds-deriv f*) *s* = *deriv* (*eval-fds f*) *s*
⟨*proof*⟩

lemma *eval-fds-higher-deriv*:
assumes (*s* :: *'a* :: *dirichlet-series*) · *1* > *conv-abscissa f*
shows *eval-fds* ((*fds-deriv* ^^ *n*) *f*) *s* = (*deriv* ^^ *n*) (*eval-fds f*) *s*
⟨*proof*⟩

end

12.3 Multiplication of two series

lemma
fixes *f g* :: *nat* ⇒ *'a* :: {*banach*, *real-normed-field*, *second-countable-topology*,

nat-power}
  fixes *s* :: *'a*
  assumes [*simp*]: *f 0 = 0 g 0 = 0*
  assumes *summable*: *summable* (*λn. norm* (*f n* / *nat-power n s*))
    summable (*λn. norm* (*g n* / *nat-power n s*))
  shows *summable-dirichlet-prod*: *summable* (*λn. norm* (*dirichlet-prod f g n* / *nat-power n s*))
  and *suminf-dirichlet-prod*:
    ($\sum$ *n. dirichlet-prod f g n* / *nat-power n s*) =
      ($\sum$ *n. f n* / *nat-power n s*) ∗ ($\sum$ *n. g n* / *nat-power n s*)
⟨*proof*⟩

lemma
  fixes *f g* :: *nat* ⇒ *real*
  fixes *s* :: *real*
  assumes *f 0 = 0 g 0 = 0*
  assumes *summable*: *summable* (*λn. norm* (*f n* / *real n powr s*))
    summable (*λn. norm* (*g n* / *real n powr s*))
  shows *summable-dirichlet-prod-real*: *summable* (*λn. norm* (*dirichlet-prod f g n* / *real n powr s*))
  and *suminf-dirichlet-prod-real*:
    ($\sum$ *n. dirichlet-prod f g n* / *real n powr s*) =
      ($\sum$ *n. f n* / *nat-power n s*) ∗ ($\sum$ *n. g n* / *real n powr s*)
  ⟨*proof*⟩

lemma *fds-abs-converges-mult*:
  fixes *s* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*}
  assumes *fds-abs-converges f s fds-abs-converges g s*
  shows *fds-abs-converges* (*f* ∗ *g*) *s*
  ⟨*proof*⟩

lemma *fds-abs-converges-power*:
  fixes *s* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*}
  shows *fds-abs-converges f s* ⟹ *fds-abs-converges* (*f* ^ *n*) *s*
  ⟨*proof*⟩

lemma *fds-abs-converges-prod*:
  fixes *s* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*}
  shows (⋀*x. x* ∈ *A* ⟹ *fds-abs-converges* (*f x*) *s*) ⟹ *fds-abs-converges* (*prod f A*) *s*
  ⟨*proof*⟩

lemma *abs-conv-abscissa-mult-le*:
  abs-conv-abscissa (*f* ∗ *g* :: *'a* :: *dirichlet-series fds*) ≤
    max (*abs-conv-abscissa f*) (*abs-conv-abscissa g*)
⟨*proof*⟩

lemma *abs-conv-abscissa-mult-leI*:
  abs-conv-abscissa (*f* :: *'a* :: *dirichlet-series fds*) ≤ *d* ⟹

abs-conv-abscissa g $\leq$ *d* $\Longrightarrow$ *abs-conv-abscissa* (*f* $*$ *g*) $\leq$ *d*
⟨*proof*⟩

lemma *abs-conv-abscissa-shift* [*simp*]:
abs-conv-abscissa (*fds-shift c f*) = *abs-conv-abscissa* (*f* :: *'a* :: *dirichlet-series fds*) + *c* · *1*
⟨*proof*⟩

lemma *eval-fds-mult*:
fixes *s* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*}
assumes *fds-abs-converges f s fds-abs-converges g s*
shows *eval-fds* (*f* $*$ *g*) *s* = *eval-fds f s* $*$ *eval-fds g s*
⟨*proof*⟩

lemma *eval-fds-power*:
fixes *s* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*}
assumes *fds-abs-converges f s*
shows *eval-fds* (*f* ^ *n*) *s* = *eval-fds f s* ^ *n*
⟨*proof*⟩

lemma *eval-fds-prod*:
fixes *s* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*}
assumes ($\bigwedge$*x*. *x* $\in$ *A* $\Longrightarrow$ *fds-abs-converges* (*f x*) *s*)
shows *eval-fds* (*prod f A*) *s* = ($\prod$*x*$\in$*A*. *eval-fds* (*f x*) *s*) ⟨*proof*⟩

lemma *eval-fds-inverse*:
fixes *s* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*}
assumes *fds-abs-converges f s fds-abs-converges* (*inverse f*) *s fds-nth f 1* $\neq$ *0*
shows *eval-fds* (*inverse f*) *s* = *inverse* (*eval-fds f s*)
⟨*proof*⟩

lemma *eval-fds-integral-has-field-derivative*:
fixes *s* :: *'a* :: *dirichlet-series*
assumes *ereal* (*s* · *1*) > *abs-conv-abscissa f*
assumes *fds-nth f 1* = *0*
shows (*eval-fds* (*fds-integral c f*) *has-field-derivative eval-fds f s*) (*at s*)
⟨*proof*⟩

lemma *holomorphic-fds-eval* [*holomorphic-intros*]:
A $\subseteq$ {*z*. *Re z* > *conv-abscissa f*} $\Longrightarrow$ *eval-fds f holomorphic-on A*
⟨*proof*⟩

lemma *analytic-fds-eval* [*holomorphic-intros*]:
assumes *A* $\subseteq$ {*z*. *Re z* > *conv-abscissa f*}
shows *eval-fds f analytic-on A*
⟨*proof*⟩

lemma *conv-abscissa-0* [*simp*]:
conv-abscissa (*0* :: *'a* :: *dirichlet-series fds*) = $-\infty$

⟨*proof*⟩

lemma *abs-conv-abscissa-0* [*simp*]:
abs-conv-abscissa (*0* :: *'a* :: *dirichlet-series fds*) = −∞
⟨*proof*⟩

lemma *conv-abscissa-1* [*simp*]:
conv-abscissa (*1* :: *'a* :: *dirichlet-series fds*) = −∞
⟨*proof*⟩

lemma *abs-conv-abscissa-1* [*simp*]:
abs-conv-abscissa (*1* :: *'a* :: *dirichlet-series fds*) = −∞
⟨*proof*⟩

lemma *conv-abscissa-const* [*simp*]:
conv-abscissa (*fds-const* (*c* :: *'a* :: *dirichlet-series*)) = −∞
⟨*proof*⟩

lemma *abs-conv-abscissa-const* [*simp*]:
abs-conv-abscissa (*fds-const* (*c* :: *'a* :: *dirichlet-series*)) = −∞
⟨*proof*⟩

lemma *conv-abscissa-numeral* [*simp*]:
conv-abscissa (*numeral n* :: *'a* :: *dirichlet-series fds*) = −∞
⟨*proof*⟩

lemma *abs-conv-abscissa-numeral* [*simp*]:
abs-conv-abscissa (*numeral n* :: *'a* :: *dirichlet-series fds*) = −∞
⟨*proof*⟩

lemma *abs-conv-abscissa-power-le*:
abs-conv-abscissa (*f* ^ *n* :: *'a* :: *dirichlet-series fds*) ≤ *abs-conv-abscissa f*
⟨*proof*⟩

lemma *abs-conv-abscissa-power-leI*:
abs-conv-abscissa (*f* :: *'a* :: *dirichlet-series fds*) ≤ *d* ⟹ *abs-conv-abscissa* (*f* ^ *n*) ≤ *d*
⟨*proof*⟩

lemma *abs-conv-abscissa-prod-le*:
assumes ⋀*x*. *x* ∈ *A* ⟹ *abs-conv-abscissa* (*f x* :: *'a* :: *dirichlet-series fds*) ≤ *d*
shows *abs-conv-abscissa* (*prod f A*) ≤ *d* ⟨*proof*⟩

lemma *conv-abscissa-add-le*:
conv-abscissa (*f* + *g* :: *'a* :: *dirichlet-series fds*) ≤ *max* (*conv-abscissa f*) (*conv-abscissa g*)
⟨*proof*⟩

lemma *conv-abscissa-add-leI*:

conv-abscissa (*f* :: *'a* :: *dirichlet-series fds*) ≤ *d* ⟹ *conv-abscissa g* ≤ *d* ⟹
conv-abscissa (*f* + *g*) ≤ *d*
⟨*proof*⟩

lemma *conv-abscissa-sum-leI*:
assumes ⋀*x*. *x* ∈ *A* ⟹ *conv-abscissa* (*f x* :: *'a* :: *dirichlet-series fds*) ≤ *d*
shows *conv-abscissa* (*sum f A*) ≤ *d* ⟨*proof*⟩

lemma *abs-conv-abscissa-add-le*:
abs-conv-abscissa (*f* + *g* :: *'a* :: *dirichlet-series fds*) ≤ *max* (*abs-conv-abscissa f*) (*abs-conv-abscissa g*)
⟨*proof*⟩

lemma *abs-conv-abscissa-add-leI*:
abs-conv-abscissa (*f* :: *'a* :: *dirichlet-series fds*) ≤ *d* ⟹ *abs-conv-abscissa g* ≤ *d* ⟹
abs-conv-abscissa (*f* + *g*) ≤ *d*
⟨*proof*⟩

lemma *abs-conv-abscissa-sum-leI*:
assumes ⋀*x*. *x* ∈ *A* ⟹ *abs-conv-abscissa* (*f x* :: *'a* :: *dirichlet-series fds*) ≤ *d*
shows *abs-conv-abscissa* (*sum f A*) ≤ *d* ⟨*proof*⟩

lemma *fds-converges-cmult-left* [*intro*]:
assumes *fds-converges f s*
shows *fds-converges* (*fds-const c* ∗ *f*) *s*
⟨*proof*⟩

lemma *fds-converges-cmult-right* [*intro*]:
assumes *fds-converges f s*
shows *fds-converges* (*f* ∗ *fds-const c*) *s*
⟨*proof*⟩

lemma *conv-abscissa-cmult-left* [*simp*]:
fixes *c* :: *'a* :: *dirichlet-series* **assumes** *c* ≠ *0*
shows *conv-abscissa* (*fds-const c* ∗ *f*) = *conv-abscissa f*
⟨*proof*⟩

lemma *conv-abscissa-cmult-right* [*simp*]:
fixes *c* :: *'a* :: *dirichlet-series* **assumes** *c* ≠ *0*
shows *conv-abscissa* (*f* ∗ *fds-const c*) = *conv-abscissa f*
⟨*proof*⟩

lemma *abs-conv-abscissa-cmult*:
fixes *c* :: *'a* :: *dirichlet-series* **assumes** *c* ≠ *0*
shows *abs-conv-abscissa* (*fds-const c* ∗ *f*) = *abs-conv-abscissa f*
⟨*proof*⟩

lemma *conv-abscissa-minus* [*simp*]:

fixes *f* :: *'a* :: *dirichlet-series fds*
shows *conv-abscissa* (−*f*) = *conv-abscissa f*
⟨*proof*⟩

lemma *abs-conv-abscissa-minus* [*simp*]:
fixes *f* :: *'a* :: *dirichlet-series fds*
shows *abs-conv-abscissa* (−*f*) = *abs-conv-abscissa f*
⟨*proof*⟩

lemma *conv-abscissa-diff-le*:
conv-abscissa (*f* − *g* :: *'a* :: *dirichlet-series fds*) ≤ *max* (*conv-abscissa f*) (*conv-abscissa g*)
⟨*proof*⟩

lemma *conv-abscissa-diff-leI*:
conv-abscissa (*f* :: *'a* :: *dirichlet-series fds*) ≤ *d* ⟹ *conv-abscissa g* ≤ *d* ⟹
conv-abscissa (*f* − *g*) ≤ *d*
⟨*proof*⟩

lemma *abs-conv-abscissa-diff-le*:
abs-conv-abscissa (*f* − *g* :: *'a* :: *dirichlet-series fds*) ≤
max (*abs-conv-abscissa f*) (*abs-conv-abscissa g*)
⟨*proof*⟩

lemma *abs-conv-abscissa-diff-leI*:
abs-conv-abscissa (*f* :: *'a* :: *dirichlet-series fds*) ≤ *d* ⟹ *abs-conv-abscissa g* ≤ *d* ⟹
abs-conv-abscissa (*f* − *g*) ≤ *d*
⟨*proof*⟩

lemmas *eval-fds-integral-has-field-derivative'* [*derivative-intros*] =
DERIV-chain'[*OF - eval-fds-integral-has-field-derivative*]

lemma *abs-conv-abscissa-completely-multiplicative-log-deriv*:
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *completely-multiplicative-function* (*fds-nth f*) *fds-nth f 1* ≠ *0*
shows *abs-conv-abscissa* (*fds-deriv f* / *f*) ≤ *abs-conv-abscissa f*
⟨*proof*⟩

12.4 Uniqueness

context
assumes *SORT-CONSTRAINT* (*'a* :: *dirichlet-series*)
begin

lemma *norm-dirichlet-series-cutoff-le*:
assumes *fds-abs-converges f* (*s0* :: *'a*) *N* > *0 s* · *1* ≥ *c c* ≥ *s0* · *1*
shows *summable* (*λn. fds-nth f* (*n* + *N*) / *nat-power* (*n* + *N*) *s*)
summable (*λn. norm* (*fds-nth f* (*n* + *N*)) / *nat-power* (*n* + *N*) *c*)

and *norm* ($\sum$ *n. fds-nth f* (*n* + *N*) / *nat-power* (*n* + *N*) *s*) ≤
($\sum$ *n. norm* (*fds-nth f* (*n* + *N*)) / *nat-power* (*n* + *N*) *c*) / *nat-power* *N* (*s* • *1* − *c*)
⟨*proof*⟩

lemma *eval-fds-zeroD-aux*:
fixes *h* :: ′*a fds*
assumes *conv*: *fds-abs-converges h* (*s0* :: ′*a*)
assumes *freq*: *frequently* (λ*s. eval-fds h s* = *0*) ((λ*s. s* • *1*) *going-to at-top*)
shows *h* = *0*
⟨*proof*⟩

lemma *eval-fds-zeroD*:
fixes *h* :: ′*a fds*
assumes *conv*: *conv-abscissa h* < ∞
assumes *freq*: *frequently* (λ*s. eval-fds h s* = *0*) ((λ*s. s* • *1*) *going-to at-top*)
shows *h* = *0*
⟨*proof*⟩

lemma *eval-fds-eqD*:
fixes *f g* :: ′*a fds*
assumes *conv*: *conv-abscissa f* < ∞ *conv-abscissa g* < ∞
assumes *eq*: *frequently* (λ*s. eval-fds f s* = *eval-fds g s*) ((λ*s. s* • *1*) *going-to at-top*)
shows *f* = *g*
⟨*proof*⟩

end

12.5 Limit at infinity

lemma *eval-fds-at-top-tail-bound*:
fixes *f* :: ′*a* :: *dirichlet-series fds*
assumes *c*: *ereal c* > *abs-conv-abscissa f*
defines *B* ≡ ($\sum$ *n. norm* (*fds-nth f* (*n*+*2*)) / *real* (*n*+*2*) *powr c*) ∗ *2 powr c*
assumes *s*: *s* • *1* ≥ *c*
shows *norm* (*eval-fds f s* − *fds-nth f 1*) ≤ *B* / *2 powr* (*s* • *1*)
⟨*proof*⟩

lemma *tendsto-eval-fds-Re-at-top*:
assumes *conv-abscissa* (*f* :: ′*a* :: *dirichlet-series fds*) ≠ ∞
assumes *lim*: *filterlim* (λ*x. S x* • *1*) *at-top F*
shows ((λ*x. eval-fds f* (*S x*)) ⟶ *fds-nth f 1*) *F*
⟨*proof*⟩

lemma *tendsto-eval-fds-Re-at-top*′:
assumes *conv-abscissa* (*f* :: *complex fds*) ≠ ∞
shows *uniform-limit UNIV* (λσ *t. eval-fds f* (*of-real* σ + *of-real t* ∗ i)
) (λ- .*fds-nth f 1*) *at-top*

⟨*proof*⟩

lemma *tendsto-eval-fds-Re-going-to-at-top*:
assumes *conv-abscissa* (*f* :: *'a* :: *dirichlet-series fds*) ≠ ∞
shows ((λ*s*. *eval-fds f s*) ⟶ *fds-nth f 1*) ((λ*s*. *s* • *1*) *going-to at-top*)
⟨*proof*⟩

lemma *tendsto-eval-fds-Re-going-to-at-top'*:
assumes *conv-abscissa* (*f* :: *complex fds*) ≠ ∞
shows ((λ*s*. *eval-fds f s*) ⟶ *fds-nth f 1*) (*Re going-to at-top*)
⟨*proof*⟩

Any Dirichlet series that is not identically zero and does not diverge everywhere has a half-plane in which it converges and is non-zero.

theorem *fds-nonzero-halfplane-exists*:
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *conv-abscissa f* < ∞ *f* ≠ *0*
shows *eventually* (λ*s*. *fds-converges f s* ∧ *eval-fds f s* ≠ *0*) ((λ*s*. *s* • *1*) *going-to at-top*)
⟨*proof*⟩

12.6 Normed series

lemma *fds-converges-norm-iff* [*simp*]:
fixes *s* :: *'a* :: {*nat-power-normed-field*,*banach*}
shows *fds-converges* (*fds-norm f*) (*s* • *1*) ⟷ *fds-abs-converges f s*
⟨*proof*⟩

lemma *fds-abs-converges-norm-iff* [*simp*]:
fixes *s* :: *'a* :: {*nat-power-normed-field*,*banach*}
shows *fds-abs-converges* (*fds-norm f*) (*s* • *1*) ⟷ *fds-abs-converges f s*
⟨*proof*⟩

lemma *fds-converges-norm-iff'*:
fixes *f* :: *'a* :: {*nat-power-normed-field*,*banach*} *fds*
shows *fds-converges* (*fds-norm f*) *s* ⟷ *fds-abs-converges f* (*of-real s*)
⟨*proof*⟩

lemma *fds-abs-converges-norm-iff'*:
fixes *f* :: *'a* :: {*nat-power-normed-field*,*banach*} *fds*
shows *fds-abs-converges* (*fds-norm f*) *s* ⟷ *fds-abs-converges f* (*of-real s*)
⟨*proof*⟩

lemma *abs-conv-abscissa-norm* [*simp*]:
fixes *f* :: *'a* :: *dirichlet-series fds*
shows *abs-conv-abscissa* (*fds-norm f*) = *abs-conv-abscissa f*
⟨*proof*⟩

lemma *conv-abscissa-norm* [*simp*]:

fixes *f* :: *'a* :: *dirichlet-series fds*
shows *conv-abscissa* (*fds-norm f*) = *abs-conv-abscissa f*
⟨*proof*⟩

lemma
fixes *f g* :: *'a* :: *dirichlet-series fds*
assumes *fds-abs-converges* (*fds-norm f*) *s fds-abs-converges* (*fds-norm g*) *s*
shows *fds-abs-converges-norm-mult*: *fds-abs-converges* (*fds-norm* (*f* ∗ *g*)) *s*
and *eval-fds-norm-mult-le*:
eval-fds (*fds-norm* (*f* ∗ *g*)) *s* ≤ *eval-fds* (*fds-norm f*) *s* ∗ *eval-fds* (*fds-norm g*) *s*
⟨*proof*⟩

lemma *eval-fds-norm-nonneg*:
assumes *fds-abs-converges* (*fds-norm f*) *s*
shows *eval-fds* (*fds-norm f*) *s* ≥ *0*
⟨*proof*⟩

lemma
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *fds-abs-converges* (*fds-norm f*) *s*
shows *fds-abs-converges-norm-power*: *fds-abs-converges* (*fds-norm* (*f* ^ *n*)) *s*
and *eval-fds-norm-power-le*:
eval-fds (*fds-norm* (*f* ^ *n*)) *s* ≤ *eval-fds* (*fds-norm f*) *s* ^ *n*
⟨*proof*⟩

12.7 Logarithms of Dirichlet series

lemma *eventually-gt-ereal-at-top*: *c* ≠ ∞ ⟹ *eventually* (λ*x*. *ereal x* > *c*) *at-top*
⟨*proof*⟩

lemma *eval-fds-log-deriv*:
fixes *s* :: *'a* :: *dirichlet-series*
assumes *fds-nth f 1* ≠ *0 s* · *1* > *abs-conv-abscissa f*
s · *1* > *abs-conv-abscissa* (*fds-deriv f* / *f*)
assumes *eval-fds f s* ≠ *0*
shows *eval-fds* (*fds-deriv f* / *f*) *s* = *eval-fds* (*fds-deriv f*) *s* / *eval-fds f s*
⟨*proof*⟩

Given a sufficiently nice absolutely convergent Dirichlet series that converges to some function $f(s)$ and a holomorphic branch of $\ln f(s)$, we can construct a Dirichlet series that absolutely converges to that logarithm.

lemma *eval-fds-ln*:
fixes *s0* :: *ereal*
assumes *nz*: ⋀*s*. *Re s* > *s0* ⟹ *eval-fds f s* ≠ *0 fds-nth f 1* ≠ *0*
assumes *l*: *exp l* = *fds-nth f 1* ((*g* ∘ *of-real*) ⟶ *l*) *at-top*
assumes *g*: ⋀*s*. *Re s* > *s0* ⟹ *exp* (*g s*) = *eval-fds f s*
assumes *holo-g*: *g holomorphic-on* {*s*. *Re s* > *s0*}
assumes *ereal* (*Re s*) > *s0*

assumes *s0* ≥ *abs-conv-abscissa f* **and** *s0* ≥ *abs-conv-abscissa* (*fds-deriv f* / *f*)
shows *eval-fds* (*fds-ln l f*) *s* = *g s*
⟨*proof*⟩

Less explicitly: For a sufficiently nice absolutely convergent Dirichlet series converging to a function $f(s)$, the formal logarithm absolutely converges to some logarithm of $f(s)$.

lemma *eval-fds-ln′*:
fixes *s0* :: *ereal*
assumes *ereal* (*Re s*) > *s0*
assumes *s0* ≥ *abs-conv-abscissa f* **and** *s0* ≥ *abs-conv-abscissa* (*fds-deriv f* / *f*)
and *nz*: ⋀*s*. *Re s* > *s0* ⟹ *eval-fds f s* ≠ *0 fds-nth f 1* ≠ *0*
assumes *l*: *exp l* = *fds-nth f 1*
shows *exp* (*eval-fds* (*fds-ln l f*) *s*) = *eval-fds f s*
⟨*proof*⟩

lemma *fds-ln-completely-multiplicative*:
fixes *f* :: ′*a* :: *dirichlet-series fds*
assumes *completely-multiplicative-function* (*fds-nth f*)
assumes *fds-nth f 1* ≠ *0*
shows *fds-ln l f* = *fds* (λ*n*. *if n* = *1 then l else fds-nth f n* ∗ *mangoldt n* $/_R$ *ln n*)
⟨*proof*⟩

lemma *eval-fds-ln-completely-multiplicative-strong*:
fixes *s* :: ′*a* :: *dirichlet-series* **and** *l* :: ′*a* **and** *f* :: ′*a fds* **and** *g* :: *nat* ⇒ ′*a*
defines *h* ≡ *fds* (λ*n*. *fds-nth* (*fds-ln l f*) *n* ∗ *g n*)
assumes *fds-abs-converges h s*
assumes *completely-multiplicative-function* (*fds-nth f*) **and** *fds-nth f 1* ≠ *0*
shows (λ(*p*,*k*). (*fds-nth f p* / *nat-power p s*) ^ *Suc k* ∗ *g* (*p* ^ *Suc k*) / *of-nat* (*Suc k*))
abs-summable-on ({*p*. *prime p*} × *UNIV*) (**is** *?th1*)
and *eval-fds h s* = *l* ∗ *g 1* + ($\sum_a$(*p*, *k*)∈{*p*. *prime p*}×*UNIV*.
(*fds-nth f p* / *nat-power p s*) ^ *Suc k* ∗ *g* (*p* ^ *Suc k*) / *of-nat* (*Suc k*))
(**is** *?th2*)
⟨*proof*⟩

lemma *eval-fds-ln-completely-multiplicative*:
fixes *s* :: ′*a* :: *dirichlet-series* **and** *l* :: ′*a* **and** *f* :: ′*a fds*
assumes *completely-multiplicative-function* (*fds-nth f*) **and** *fds-nth f 1* ≠ *0*
assumes *s* · *1* > *abs-conv-abscissa* (*fds-deriv f* / *f*)
shows (λ(*p*,*k*). (*fds-nth f p* / *nat-power p s*) ^ *Suc k* / *of-nat* (*Suc k*))
abs-summable-on ({*p*. *prime p*} × *UNIV*) (**is** *?th1*)
and *eval-fds* (*fds-ln l f*) *s* =
l + ($\sum_a$(*p*, *k*)∈{*p*. *prime p*}×*UNIV*.
(*fds-nth f p* / *nat-power p s*) ^ *Suc k* / *of-nat* (*Suc k*)) (**is** *?th2*)
⟨*proof*⟩

12.8 Exponential and logarithm

lemma *summable-fds-exp-aux*:
assumes *fds-nth f' 1* = (*0* :: *'a* :: *real-normed-algebra-1*)
shows *summable* (λ*k*. *fds-nth* (*f'* ^ *k*) *n* $/_R$ *fact k*)
⟨*proof*⟩

lemma
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *fds-abs-converges f s*
shows *fds-abs-converges-exp*: *fds-abs-converges* (*fds-exp f*) *s*
and *eval-fds-exp*: *eval-fds* (*fds-exp f*) *s* = *exp* (*eval-fds f s*)
⟨*proof*⟩

lemma *fds-exp-add*:
fixes *f* :: *'a* :: *dirichlet-series fds*
shows *fds-exp* (*f* + *g*) = *fds-exp f* ∗ *fds-exp g*
⟨*proof*⟩

lemma *fds-exp-minus*:
fixes *f* :: *'a* :: *dirichlet-series fds*
shows *fds-exp* (−*f*) = *inverse* (*fds-exp f*)
⟨*proof*⟩

lemma *abs-conv-abscissa-exp*:
fixes *f* :: *'a* :: *dirichlet-series fds*
shows *abs-conv-abscissa* (*fds-exp f*) ≤ *abs-conv-abscissa f*
⟨*proof*⟩

lemma *fds-deriv-exp* [*simp*]:
fixes *f* :: *'a* :: *dirichlet-series fds*
shows *fds-deriv* (*fds-exp f*) = *fds-exp f* ∗ *fds-deriv f*
⟨*proof*⟩

lemma *fds-exp-ln-strong*:
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *fds-nth f* (*Suc 0*) ≠ *0*
shows *fds-exp* (*fds-ln l f*) = *fds-const* (*exp l* / *fds-nth f* (*Suc 0*)) ∗ *f*
⟨*proof*⟩

lemma *fds-exp-ln* [*simp*]:
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *exp l* = *fds-nth f* (*Suc 0*)
shows *fds-exp* (*fds-ln l f*) = *f*
⟨*proof*⟩

lemma *fds-ln-exp* [*simp*]:
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *l* = *fds-nth f* (*Suc 0*)
shows *fds-ln l* (*fds-exp f*) = *f*

⟨*proof*⟩

12.9 Euler products

lemma *fds-euler-product-LIMSEQ*:
fixes *f* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*} *fds*
assumes *multiplicative-function* (*fds-nth f*) **and** *fds-abs-converges f s*
shows $(\lambda n.\ \prod p \le n.\ \textit{if prime p then}\ \sum i.\ \textit{fds-nth f}\ (p\ \hat{}\ i)\ /\ \textit{nat-power}\ (p\ \hat{}\ i)\ s\ \textit{else}\ 1) \longrightarrow$ *eval-fds f s*
⟨*proof*⟩

lemma *fds-euler-product-LIMSEQ'*:
fixes *f* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*} *fds*
assumes *completely-multiplicative-function* (*fds-nth f*) **and** *fds-abs-converges f s*
shows $(\lambda n.\ \prod p \le n.\ \textit{if prime p then inverse}\ (1 - \textit{fds-nth f p}\ /\ \textit{nat-power p s})\ \textit{else}\ 1) \longrightarrow$ *eval-fds f s*
⟨*proof*⟩

lemma *fds-abs-convergent-euler-product*:
fixes *f* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*} *fds*
assumes *multiplicative-function* (*fds-nth f*) **and** *fds-abs-converges f s*
shows *abs-convergent-prod*
$(\lambda p.\ \textit{if prime p then}\ \sum i.\ \textit{fds-nth f}\ (p\ \hat{}\ i)\ /\ \textit{nat-power}\ (p\ \hat{}\ i)\ s\ \textit{else}\ 1)$
⟨*proof*⟩

lemma *fds-abs-convergent-euler-product'*:
fixes *f* :: *'a* :: {*nat-power*, *real-normed-field*, *banach*, *second-countable-topology*} *fds*
assumes *completely-multiplicative-function* (*fds-nth f*) **and** *fds-abs-converges f s*
shows *abs-convergent-prod*
$(\lambda p.\ \textit{if prime p then inverse}\ (1 - \textit{fds-nth f p}\ /\ \textit{nat-power p s})\ \textit{else}\ 1)$
⟨*proof*⟩

lemma *fds-abs-convergent-zero-iff*:
fixes *f* :: *'a* :: {*nat-power-field*, *real-normed-field*, *banach*, *second-countable-topology*} *fds*
assumes *completely-multiplicative-function* (*fds-nth f*)
assumes *fds-abs-converges f s*
shows $\textit{eval-fds f s} = 0 \longleftrightarrow (\exists p.\ \textit{prime p} \wedge \textit{fds-nth f p} = \textit{nat-power p s})$
⟨*proof*⟩

lemma
fixes *s* :: *'a* :: {*nat-power-normed-field*,*banach*,*euclidean-space*}
assumes $s \cdot 1 > 1$

shows *euler-product-fds-zeta*:
$(\lambda n.\ \prod p{\le}n.\ \textit{if prime p then inverse } (1 - 1 \ / \ \textit{nat-power p s}) \textit{ else } 1)$
$\longrightarrow$ *eval-fds fds-zeta s* (**is** *?th1*)
and *eval-fds-zeta-nonzero*: *eval-fds fds-zeta s* $\neq 0$
⟨*proof*⟩

lemma *fds-primepow-subseries-euler-product-cm*:
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *completely-multiplicative-function* (*fds-nth f*) *prime p*
assumes *s* · *1* > *abs-conv-abscissa f*
shows *eval-fds* (*fds-primepow-subseries p f*) *s* = *1* / (*1* − *fds-nth f p* / *nat-power p s*)
⟨*proof*⟩

12.10 Non-negative Dirichlet series

lemma *nonneg-Reals-sum*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{R}_{\ge 0}) \Longrightarrow \textit{sum}\ f\ A \in \mathbb{R}_{\ge 0}$
⟨*proof*⟩

locale *nonneg-dirichlet-series* =
fixes *f* :: *'a* :: *dirichlet-series fds*
assumes *nonneg-coeffs-aux*: $n > 0 \Longrightarrow \textit{fds-nth}\ f\ n \in \mathbb{R}_{\ge 0}$
begin

lemma *nonneg-coeffs*: *fds-nth f n* $\in \mathbb{R}_{\ge 0}$
⟨*proof*⟩

end

lemma *nonneg-dirichlet-series-0* [*simp*,*intro*]: *nonneg-dirichlet-series 0*
⟨*proof*⟩

lemma *nonneg-dirichlet-series-1* [*simp*,*intro*]: *nonneg-dirichlet-series 1*
⟨*proof*⟩

lemma *nonneg-dirichlet-series-const* [*simp*,*intro*]:
$c \in \mathbb{R}_{\ge 0} \Longrightarrow$ *nonneg-dirichlet-series* (*fds-const c*)
⟨*proof*⟩

lemma *nonneg-dirichlet-series-add* [*intro*]:
assumes *nonneg-dirichlet-series f nonneg-dirichlet-series g*
shows *nonneg-dirichlet-series* (*f* + *g*)
⟨*proof*⟩

lemma *nonneg-dirichlet-series-mult* [*intro*]:
assumes *nonneg-dirichlet-series f nonneg-dirichlet-series g*
shows *nonneg-dirichlet-series* (*f* ∗ *g*)
⟨*proof*⟩

lemma *nonneg-dirichlet-series-power* [*intro*]:
  assumes *nonneg-dirichlet-series f*
  shows *nonneg-dirichlet-series* (*f* ^ *n*)
  ⟨*proof*⟩

context *nonneg-dirichlet-series*
begin

lemma *nonneg-exp* [*intro*]: *nonneg-dirichlet-series* (*fds-exp f*)
⟨*proof*⟩

end

lemma *nonneg-dirichlet-series-lnD*:
  assumes *nonneg-dirichlet-series* (*fds-ln l f*) *exp l* = *fds-nth f* (*Suc 0*)
  shows *nonneg-dirichlet-series f*
⟨*proof*⟩

context *nonneg-dirichlet-series*
begin

lemma *fds-of-real-norm*: *fds-of-real* (*fds-norm f*) = *f*
⟨*proof*⟩

end

lemma *pringsheim-landau-aux*:
  fixes *c* :: *real* **and** *f* :: *complex fds*
  assumes *nonneg-dirichlet-series f*
  assumes *abscissa*: *c* ≥ *abs-conv-abscissa f*
  assumes *g*: ⋀*s*. *s* ∈ *A* ⟹ *Re s* > *c* ⟹ *g s* = *eval-fds f s*
  assumes *g holomorphic-on A open A c* ∈ *A*
  shows ∃ *x*. *x* < *c* ∧ *fds-abs-converges f* (*of-real x*)
⟨*proof*⟩

theorem *pringsheim-landau*:
  fixes *c* :: *real* **and** *f* :: *complex fds*
  assumes *nonneg-dirichlet-series f*
  assumes *abscissa*: *abs-conv-abscissa f* = *c*
  assumes *g*: ⋀*s*. *s* ∈ *A* ⟹ *Re s* > *c* ⟹ *g s* = *eval-fds f s*
  assumes *g holomorphic-on A open A c* ∈ *A*
  shows *False*
⟨*proof*⟩

corollary *entire-continuation-imp-abs-conv-abscissa-MInfty*:
  assumes *nonneg-dirichlet-series f*
  assumes *c*: *c* ≥ *abs-conv-abscissa f*
  assumes *g*: ⋀*s*. *Re s* > *c* ⟹ *g s* = *eval-fds f s*

assumes *holo*: *g holomorphic-on UNIV*
shows *abs-conv-abscissa f* = $-\infty$
⟨*proof*⟩

12.11 Convergence of the ζ and Möbius μ series

lemma *fds-abs-summable-zeta-real-iff* [*simp*]:
fds-abs-converges fds-zeta s $\longleftrightarrow$ *s* > (*1* :: *real*)
⟨*proof*⟩

lemma *fds-abs-summable-zeta-real*: *s* > (*1* :: *real*) $\Longrightarrow$ *fds-abs-converges fds-zeta s*
⟨*proof*⟩

lemma *fds-abs-converges-moebius-mu-real*:
assumes *s* > (*1* :: *real*)
shows *fds-abs-converges* (*fds moebius-mu*) *s*
⟨*proof*⟩

12.12 Application to the Möbius μ function

lemma *inverse-squares-sums'*: (λ*n*. *1* / *real n* ^ *2*) *sums* (*pi* ^ *2* / *6*)
⟨*proof*⟩

lemma *norm-summable-moebius-over-square*:
summable (λ*n*. *norm* (*moebius-mu n* / *real n* ^ *2*))
⟨*proof*⟩

lemma *summable-moebius-over-square*:
summable (λ*n*. *moebius-mu n* / *real n* ^ *2*)
⟨*proof*⟩

lemma *moebius-over-square-sums*: (λ*n*. *moebius-mu n* / n^2) *sums* (*6* / pi^2)
⟨*proof*⟩

end

13 Asymptotics of summatory arithmetic functions

theory *Arithmetic-Summatory-Asymptotics*
imports
Euler-MacLaurin.Euler-MacLaurin-Landau
Arithmetic-Summatory
Dirichlet-Series-Analysis
Landau-Symbols.Landau-More
begin

13.1 Auxiliary bounds

lemma *sum-inverse-squares-tail-bound*:
assumes $d > 0$
shows $summable\ (\lambda n.\ 1\ /\ (real\ (Suc\ n) + d)\ \hat{}\ 2)$
$(\sum n.\ 1\ /\ (real\ (Suc\ n) + d)\ \hat{}\ 2) \leq 1\ /\ d$
⟨*proof*⟩

lemma *moebius-sum-tail-bound*:
assumes $d > 0$
shows $abs\ (\sum n.\ moebius\text{-}mu\ (Suc\ n + d)\ /\ real\ (Suc\ n + d)\hat{}2) \leq 1\ /\ d$ (**is** $abs\ ?S \leq$ -)
⟨*proof*⟩

lemma *sum-upto-inverse-bound*:
$sum\text{-}upto\ (\lambda i.\ 1\ /\ real\ i)\ x \geq 0$
$eventually\ (\lambda x.\ sum\text{-}upto\ (\lambda i.\ 1\ /\ real\ i)\ x \leq ln\ x + 13\ /\ 22)\ at\text{-}top$
⟨*proof*⟩

lemma *sum-upto-inverse-bigo*: $sum\text{-}upto\ (\lambda i.\ 1\ /\ real\ i) \in O(\lambda x.\ ln\ x)$
⟨*proof*⟩

lemma
defines $G \equiv (\lambda x{::}real.\ (\sum n.\ moebius\text{-}mu\ (n + Suc\ (nat\ \lfloor x \rfloor))\ /\ (n + Suc\ (nat\ \lfloor x \rfloor))\hat{}2)\ ::\ real)$
shows *moebius-sum-tail-bound'*: $\bigwedge t.\ t \geq 2 \Longrightarrow |G\ t| \leq 1\ /\ (t - 1)$
and *moebius-sum-tail-bigo*: $G \in O(\lambda t.\ 1\ /\ t)$
⟨*proof*⟩

13.2 Summatory totient function

theorem *summatory-totient-asymptotics*:
$(\lambda x.\ sum\text{-}upto\ (\lambda n.\ real\ (totient\ n))\ x - 3\ /\ pi^2 * x^2) \in O(\lambda x.\ x * ln\ x)$
⟨*proof*⟩

theorem *summatory-totient-asymptotics'*:
$(\lambda x.\ sum\text{-}upto\ (\lambda n.\ real\ (totient\ n))\ x) =o\ (\lambda x.\ 3\ /\ pi^2 * x^2) +o\ O(\lambda x.\ x * ln\ x)$
⟨*proof*⟩

theorem *summatory-totient-asymptotics''*:
$sum\text{-}upto\ (\lambda n.\ real\ (totient\ n)) \sim[at\text{-}top]\ (\lambda x.\ 3\ /\ pi^2 * x^2)$
⟨*proof*⟩

13.3 Asymptotic distribution of squarefree numbers

lemma *le-sqrt-iff*: $x \geq 0 \Longrightarrow x \leq sqrt\ y \longleftrightarrow x\hat{}2 \leq y$
⟨*proof*⟩

theorem *squarefree-asymptotics*: $(\lambda x.\ card\ \{n.\ real\ n \leq x \wedge squarefree\ n\} - 6\ /\ pi^2 * x) \in O(sqrt)$

⟨*proof*⟩

theorem *squarefree-asymptotics′*:
$(\lambda x.\ card\ \{n.\ real\ n \le x \wedge squarefree\ n\}) =o\ (\lambda x.\ 6\ /\ pi^2 * x) +o\ O(\lambda x.\ sqrt\ x)$
⟨*proof*⟩

theorem *squarefree-asymptotics′′*:
$(\lambda x.\ card\ \{n.\ real\ n \le x \wedge squarefree\ n\}) \sim[at\text{-}top]\ (\lambda x.\ 6\ /\ pi^2 * x)$
⟨*proof*⟩

13.4 The hyperbola method

lemma *hyperbola-method-bigo*:
fixes $f\ g :: nat \Rightarrow 'a :: real\text{-}normed\text{-}field$
assumes $(\lambda x.\ sum\text{-}upto\ (\lambda n.\ f\ n * sum\text{-}upto\ g\ (x\ /\ real\ n))\ (sqrt\ x) - R\ x) \in O(b)$
assumes $(\lambda x.\ sum\text{-}upto\ (\lambda n.\ sum\text{-}upto\ f\ (x\ /\ real\ n) * g\ n)\ (sqrt\ x) - S\ x) \in O(b)$
assumes $(\lambda x.\ sum\text{-}upto\ f\ (sqrt\ x) * sum\text{-}upto\ g\ (sqrt\ x) - T\ x) \in O(b)$
shows $(\lambda x.\ sum\text{-}upto\ (dirichlet\text{-}prod\ f\ g)\ x - (R\ x + S\ x - T\ x)) \in O(b)$
⟨*proof*⟩

lemma *frac-le-1*: $frac\ x \le 1$
⟨*proof*⟩

lemma *ln-minus-ln-floor-bound*:
assumes $x \ge 2$
shows $ln\ x - ln\ (floor\ x) \in \{0..<1\ /\ (x - 1)\}$
⟨*proof*⟩

lemma *ln-minus-ln-floor-bigo*:
$(\lambda x::real.\ ln\ x - ln\ (floor\ x)) \in O(\lambda x.\ 1\ /\ x)$
⟨*proof*⟩

lemma *divisor-count-asymptotics-aux*:
$(\lambda x.\ sum\text{-}upto\ (\lambda n.\ sum\text{-}upto\ (\lambda\text{-}.\ 1)\ (x\ /\ real\ n))\ (sqrt\ x) - (x * ln\ x\ /\ 2 + euler\text{-}mascheroni * x)) \in O(sqrt)$
⟨*proof*⟩

lemma *sum-upto-sqrt-bound*:
assumes x: $x \ge (0 :: real)$
shows $norm\ ((sum\text{-}upto\ (\lambda\text{-}.\ 1)\ (sqrt\ x))^2 - x) \le 2 * norm\ (sqrt\ x)$
⟨*proof*⟩

lemma *summatory-divisor-count-asymptotics*:
$(\lambda x.\ sum\text{-}upto\ (\lambda n.\ real\ (divisor\text{-}count\ n))\ x - (x * ln\ x + (2 * euler\text{-}mascheroni - 1) * x)) \in O(sqrt)$
⟨*proof*⟩

theorem *summatory-divisor-count-asymptotics′*:
  $(\lambda x.\ sum\text{-}upto\ (\lambda n.\ real\ (divisor\text{-}count\ n))\ x) =o$
    $(\lambda x.\ x * ln\ x + (2 * euler\text{-}mascheroni - 1) * x) +o\ O(\lambda x.\ sqrt\ x)$
  ⟨*proof*⟩

theorem *summatory-divisor-count-asymptotics′′*:
  $sum\text{-}upto\ (\lambda n.\ real\ (divisor\text{-}count\ n)) \sim[at\text{-}top]\ (\lambda x.\ x * ln\ x)$
⟨*proof*⟩

lemma *summatory-divisor-eq*:
  $sum\text{-}upto\ (\lambda n.\ real\ (divisor\text{-}count\ n))\ (real\ m) = card\ \{(n,d).\ n \in \{0<..m\} \wedge d\ dvd\ n\}$
⟨*proof*⟩

context
  fixes $M :: nat \Rightarrow real$
  defines $M \equiv \lambda m.\ card\ \{(n,d).\ n \in \{0<..m\} \wedge d\ dvd\ n\}\ /\ card\ \{0<..m\}$
begin

lemma *mean-divisor-count-asymptotics*:
  $(\lambda m.\ M\ m - (ln\ m + 2 * euler\text{-}mascheroni - 1)) \in O(\lambda m.\ 1\ /\ sqrt\ m)$
⟨*proof*⟩

theorem *mean-divisor-count-asymptotics′*:
  $M =o\ (\lambda x.\ ln\ x + 2 * euler\text{-}mascheroni - 1) +o\ O(\lambda x.\ 1\ /\ sqrt\ x)$
  ⟨*proof*⟩

theorem *mean-divisor-count-asymptotics′′*: $M \sim[at\text{-}top]\ ln$
⟨*proof*⟩

end

13.5 The asymptotic ditribution of coprime pairs

context
  fixes $A :: nat \Rightarrow (nat \times nat)\ set$
  defines $A \equiv (\lambda N.\ \{(m,n) \in \{1..N\} \times \{1..N\}.\ coprime\ m\ n\})$
begin

lemma *coprime-pairs-asymptotics*:
  $(\lambda N.\ real\ (card\ (A\ N)) - 6\ /\ pi^2 * (real\ N)^2) \in O(\lambda N.\ real\ N * ln\ (real\ N))$
⟨*proof*⟩

theorem *coprime-pairs-asymptotics′*:
  $(\lambda N.\ real\ (card\ (A\ N))) =o\ (\lambda N.\ 6\ /\ pi^2 * (real\ N)^2) +o\ O(\lambda N.\ real\ N * ln\ (real\ N))$
  ⟨*proof*⟩

theorem *coprime-pairs-asymptotics′′*:

$(\lambda N.\ real\ (card\ (A\ N)))\ \sim[at\text{-}top]\ (\lambda N.\ 6\ /\ pi^2 * (real\ N)^2)$
⟨*proof*⟩

theorem *coprime-probability-tendsto*:
$(\lambda N.\ card\ (A\ N)\ /\ card\ (\{1..N\} \times \{1..N\})) \longrightarrow 6\ /\ pi^2$
⟨*proof*⟩

end

13.6 The asymptotics of the number of Farey fractions

definition *farey-fractions* :: *nat* ⇒ *rat set* **where**
$farey\text{-}fractions\ N = \{q :: rat \in \{0{<}..1\}.\ snd\ (quotient\text{-}of\ q) \leq int\ N\}$

lemma *Fract-eq-coprime*:
assumes $Rat.Fract\ a\ b = Rat.Fract\ c\ d\ \ b > 0\ \ d > 0\ \ coprime\ a\ b\ \ coprime\ c\ d$
shows $a = c\ \ b = d$
⟨*proof*⟩

lemma *quotient-of-split*:
$P\ (quotient\text{-}of\ q) = (\forall a\ b.\ b > 0 \longrightarrow coprime\ a\ b \longrightarrow q = Rat.Fract\ a\ b \longrightarrow P\ (a,\ b))$
⟨*proof*⟩

lemma *quotient-of-split-asm*:
$P\ (Rat.quotient\text{-}of\ q) = (\neg(\exists a\ b.\ b > 0 \wedge coprime\ a\ b \wedge q = Rat.Fract\ a\ b \wedge \neg P\ (a,\ b)))$
⟨*proof*⟩

lemma *farey-fractions-bij*:
$bij\text{-}betw\ (\lambda(a,b).\ Rat.Fract\ (int\ a)\ (int\ b))$
$\{(a,b)|a\ b.\ 0 < a \wedge a \leq b \wedge b \leq N \wedge coprime\ a\ b\}\ (farey\text{-}fractions\ N)$
⟨*proof*⟩

lemma *card-farey-fractions*: $card\ (farey\text{-}fractions\ N) = sum\ totient\ \{0{<}..N\}$
⟨*proof*⟩

lemma *card-farey-fractions-asymptotics*:
$(\lambda N.\ real\ (card\ (farey\text{-}fractions\ N)) - 3\ /\ pi^2 * (real\ N)^2) \in O(\lambda N.\ real\ N * ln\ (real\ N))$
⟨*proof*⟩

theorem *card-farey-fractions-asymptotics′*:
$(\lambda N.\ card\ (farey\text{-}fractions\ N)) =o\ (\lambda N.\ 3\ /\ pi^2 * N\hat{}2) +o\ O(\lambda N.\ N * ln\ N)$
⟨*proof*⟩

theorem *card-farey-fractions-asymptotics″*:
$(\lambda N.\ real\ (card\ (farey\text{-}fractions\ N)))\ \sim[at\text{-}top]\ (\lambda N.\ 3\ /\ pi^2 * (real\ N)^2)$
⟨*proof*⟩

end

14 Efficient code for number-theoretic functions

theory *Dirichlet-Efficient-Code*
imports
  Main
  Moebius-Mu
  More-Totient
  Divisor-Count
  Liouville-Lambda
  HOL−Library.Code-Target-Numeral
  Polynomial-Factorization.Prime-Factorization
begin

definition *prime-factorization-nat′* :: *nat* ⇒ (*nat* × *nat*) *list* **where**
  prime-factorization-nat′ n = (
    let ps = *prime-factorization-nat n*
    in map (λ*p*. (*p*, *length* (*filter* ((=) *p*) *ps*) − *1*)) (*remdups-adj* (*sort ps*)))

lemma *set-prime-factorization-nat′*:
  set (*prime-factorization-nat′ n*) = (λ*p*. (*p*, *multiplicity p n* − *1*)) ‘ *prime-factors n*
⟨*proof*⟩

lemma *distinct-prime-factorization-nat′* [*simp*]: *distinct* (*prime-factorization-nat′ n*)
  ⟨*proof*⟩

lemmas (**in** *multiplicative-function′*) *efficient-code′* =
  efficient-code [*of* λ-. *prime-factorization-nat′ n n* **for** *n*,
    OF set-prime-factorization-nat′ distinct-prime-factorization-nat′]

14.1 Möbius μ function

definition *moebius-mu-aux* :: *nat* ⇒ (*unit* ⇒ *nat list*) ⇒ *int* **where**
  moebius-mu-aux n ps =
    (*if n* ≠ *0* ∧ ¬*4 dvd n* ∧ ¬*9 dvd n then*
      (*let ps* = *ps* () *in if distinct ps then if even* (*length ps*) *then 1 else* −*1 else 0*) *else 0*)

lemma *moebius-mu-conv-moebius-mu-aux*:
  fixes *qs* :: *unit* ⇒ *nat list*
  defines *ps* ≡ *qs* ()
  assumes *mset ps* = *prime-factorization n*
  shows *moebius-mu n* = *of-int* (*moebius-mu-aux n qs*)
⟨*proof*⟩

lemma *moebius-mu-code* [*code*]:
moebius-mu n = *of-int* (*moebius-mu-aux n* (λ-. *prime-factorization-nat n*))
⟨*proof*⟩

value *moebius-mu 12578972695257* :: *int*

14.2 Euler's ϕ function

primrec *totient-aux1* :: *nat* ⇒ *nat list* ⇒ *nat* **where**
totient-aux1 n [] = *n*
| *totient-aux1 n* (*p* # *ps*) = *totient-aux1* (*n* − *n div p*) *ps*

lemma *of-nat-totient-aux1*:
assumes ⋀*p*. *p* ∈ *set ps* ⟹ *prime p* ⋀*p*. *p* ∈ *set ps* ⟹ *p dvd n distinct ps*
shows *real* (*totient-aux1 n ps*) = *real n* ∗ (∏ *p*∈*set ps*. *1* − *1* / *real p*)
⟨*proof*⟩

lemma *totient-conv-totient-aux1*:
assumes *set ps* = *prime-factors n distinct ps*
shows *totient n* = *totient-aux1 n ps*
⟨*proof*⟩

definition *prime-factors-nat* :: *nat* ⇒ *nat list* **where**
prime-factors-nat n = *remdups-adj* (*sort* (*prime-factorization-nat n*))

lemma *set-prime-factors-nat* [*simp*]: *set* (*prime-factors-nat n*) = *prime-factors n*
⟨*proof*⟩

lemma *distinct-prime-factors-nat* [*simp*]: *distinct* (*prime-factors-nat n*)
⟨*proof*⟩

definition *totient-aux2* :: (*nat* × *nat*) *list* ⇒ *nat* **where**
totient-aux2 xs = (∏ (*p*,*k*)←*xs*. *p* ^ *k* ∗ (*p* − *1*))

lemma *totient-conv-totient-aux2*:
assumes *n* ≠ *0*
assumes *set xs* = (λ*p*. (*p*, *multiplicity p n* − *1*)) ` *prime-factors n*
assumes *distinct xs*
shows *totient n* = *totient-aux2 xs*
⟨*proof*⟩

lemma *totient-code1*: *totient n* = *totient-aux1 n* (*prime-factors-nat n*)
⟨*proof*⟩

lemma *totient-code2*: *totient n* = (*if n* = *0 then 0 else totient-aux2* (*prime-factorization-nat′ n*))
⟨*proof*⟩

declare *totient-code-naive* [*code del*]

lemmas [*code*] = *totient-code2*

value *totient 125789726827482323235784*

14.3 Divisor Functions

lemmas [*code del*] = *divisor-count-naive divisor-sum-naive*
lemmas [*code*] = *divisor-count.efficient-code′ divisor-sum.efficient-code′*

value *int* (*divisor-count 378568418621*)
value *int* (*divisor-sum 378568418621*)

14.4 Liouville's λ function

lemma [*code*]: *liouville-lambda n* =
(*if n* = *0 then 0 else if even* (*length* (*prime-factorization-nat n*)) *then 1 else* −*1*)
⟨*proof*⟩

value *liouville-lambda 1264785343674* :: *int*

end

References

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