

Dirichlet Series

Manuel Eberl

December 7, 2022

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	4
2	Multiplicative arithmetic functions	5
2.1	Definition	5
2.2	Indicator function	9
3	Dirichlet convolution	9
4	Formal Dirichlet series	14
4.1	General properties	16
4.2	Shifting the argument	22
4.3	Scaling the argument	24
4.4	Formal derivative	26
4.5	Formal integral	28
4.6	Formal logarithm	28
4.7	Formal exponential	29
4.8	Subseries	30
4.9	Truncation	31
4.10	Normed series	33
4.11	Lifting a real series to a real algebra	34
4.12	Convergence and connection to concrete functions	34
5	The Möbius μ function	38
6	Euler's ϕ function	42
7	The Liouville λ function	43
8	The divisor functions	45
8.1	The general divisor function	45
8.2	The divisor-counting function	46
8.3	The divisor sum function	47
9	Summatory arithmetic functions	48
9.1	Definition	48
9.2	The Hyperbola method	49
10	Partial summation	50
11	Euler product expansions	51

12 Analytic properties of Dirichlet series	52
12.1 Convergence and absolute convergence	54
12.2 Derivative of a Dirichlet series	60
12.3 Multiplication of two series	62
12.4 Uniqueness	67
12.5 Limit at infinity	68
12.6 Normed series	68
12.7 Logarithms of Dirichlet series	69
12.8 Exponential and logarithm	71
12.9 Euler products	72
12.10 Non-negative Dirichlet series	73
12.11 Convergence of the ζ and Möbius μ series	75
12.12 Application to the Möbius μ function	75
13 Asymptotics of summatory arithmetic functions	76
13.1 Auxiliary bounds	76
13.2 Summatory totient function	76
13.3 Asymptotic distribution of squarefree numbers	77
13.4 The hyperbola method	77
13.5 The asymptotic distribution of coprime pairs	79
13.6 The asymptotics of the number of Farey fractions	79
14 Efficient code for number-theoretic functions	80
14.1 Möbius μ function	81
14.2 Euler's ϕ function	81
14.3 Divisor Functions	82
14.4 Liouville's λ function	82

1 Miscellaneous auxiliary facts

theory *Dirichlet-Misc*

imports

Main

HOL-Number-Theory.Number-Theory

begin

lemma

fixes $a\ k :: \text{nat}$

assumes $a > 1\ k > 0$

shows *geometric-sum-nat-aux*: $(a - 1) * (\sum_{i < k}. a^i) = a^k - 1$

and *geometric-sum-nat-dvd*: $a - 1 \text{ dvd } a^k - 1$

and *geometric-sum-nat*: $(\sum_{i < k}. a^i) = (a^k - 1) \text{ div } (a - 1)$

<proof>

lemma *dvd-div-gt0*: $d \text{ dvd } n \implies n > 0 \implies n \text{ div } d > (0 :: \text{nat})$

<proof>

lemma *Set-filter-insert*:

$\text{Set.filter } P (\text{insert } x\ A) = (\text{if } P\ x \text{ then } \text{insert } x (\text{Set.filter } P\ A) \text{ else } \text{Set.filter } P\ A)$

<proof>

lemma *Set-filter-union*: $\text{Set.filter } P (A \cup B) = \text{Set.filter } P\ A \cup \text{Set.filter } P\ B$

<proof>

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

<proof>

lemma *Set-filter-image*: $\text{Set.filter } P (f\ 'A) = f\ ' \text{Set.filter } (P \circ f)\ A$

<proof>

lemma *Set-filter-cong [cong]*:

$(\bigwedge x. x \in A \implies P\ x \longleftrightarrow Q\ x) \implies A = B \implies \text{Set.filter } P\ A = \text{Set.filter } Q\ B$

<proof>

lemma *finite-Set-filter*: $\text{finite } A \implies \text{finite } (\text{Set.filter } P\ A)$

<proof>

lemma *inj-on-insert'*: $(\bigwedge B. B \in A \implies x \notin B) \implies \text{inj-on } (\text{insert } x)\ A$

<proof>

lemma

assumes $\text{finite } A\ A \neq \{\}$

shows *card-even-subset-aux*: $\text{card } \{B. B \subseteq A \wedge \text{even } (\text{card } B)\} = 2^{\text{card } A - 1}$

and *card-odd-subset-aux*: $\text{card } \{B. B \subseteq A \wedge \text{odd } (\text{card } B)\} = 2^{\text{card } A - 1}$

and *card-even-odd-subset*: $\text{card } \{B. B \subseteq A \wedge \text{even } (\text{card } B)\} = \text{card } \{B. B \subseteq A \wedge \text{odd } (\text{card } B)\}$
 ⟨*proof*⟩

lemma *bij-betw-prod-divisors-coprime*:

assumes *coprime* a ($b :: \text{nat}$)

shows *bij-betw* $(\lambda x. \text{fst } x * \text{snd } x) (\{d. d \text{ dvd } a\} \times \{d. d \text{ dvd } b\}) \{k. k \text{ dvd } a * b\}$
 ⟨*proof*⟩

lemma *bij-betw-prime-power-divisors*:

assumes *prime* ($p :: \text{nat}$)

shows *bij-betw* $((\wedge) p) \{..k\} \{d. d \text{ dvd } p \wedge k\}$

⟨*proof*⟩

lemma *power-diff'*:

assumes $m \geq n$ $x \neq 0$

shows $x \wedge^{(m - n)} = (x \wedge^m \text{div } x \wedge^n :: 'a :: \text{unique-euclidean-semiring})$

⟨*proof*⟩

lemma *sum-divisors-coprime-mult*:

assumes *coprime* a ($b :: \text{nat}$)

shows $(\sum d \mid d \text{ dvd } a * b. f d) = (\sum r \mid r \text{ dvd } a. \sum s \mid s \text{ dvd } b. f (r * s))$

⟨*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 $f :: \text{nat} \Rightarrow 'a :: \text{comm-semiring-1}$

assumes *zero* [*simp*]: $f 0 = 0$

assumes *one* [*simp*]: $f 1 = 1$

assumes *mult-coprime-aux*: $a > 1 \implies b > 1 \implies \text{coprime } a b \implies f (a * b) = f a * f b$

begin

lemma *Suc-0* [*simp*]: $f (\text{Suc } 0) = 1$

⟨*proof*⟩

lemma *mult-coprime*:

assumes *coprime a b*
shows $f (a * b) = f a * f b$
 $\langle proof \rangle$

lemma *prod-coprime:*

assumes $\bigwedge x y. x \in A \implies y \in A \implies x \neq y \implies coprime (g x) (g y)$
shows $f (prod\ g\ A) = (\prod_{x \in A}. f (g x))$
 $\langle proof \rangle$

lemma *prod-prime-factors:*

assumes $n > 0$
shows $f n = (\prod_{p \in prime-factors\ n}. f (p \wedge multiplicity\ p\ n))$
 $\langle proof \rangle$

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

end

locale *multiplicative-function'* = *multiplicative-function f* **for** $f :: nat \Rightarrow 'a :: comm-semiring-1 +$

fixes *f-prime-power* :: $nat \Rightarrow nat \Rightarrow 'a$ **and** *f-prime* :: $nat \Rightarrow 'a$
assumes *prime-power*: $prime\ p \implies k > 0 \implies f (p \wedge k) = f-prime-power\ p\ k$
assumes *prime-aux*: $prime\ p \implies f-prime-power\ p\ 1 = f-prime\ p$

begin

lemma *prime*: $prime\ p \implies f\ p = f-prime\ p$
 $\langle proof \rangle$

lemma *prod-prime-factors'*:

assumes $n > 0$
shows $f n = (\prod_{p \in prime-factors\ n}. f-prime-power\ p (multiplicity\ p\ n))$
 $\langle proof \rangle$

lemma *efficient-code-aux*:

assumes $n > 0$ *set ps =* $(\lambda p. (p, multiplicity\ p\ n - 1))$ ' *prime-factors n distinct*
 ps
shows $f n = (\prod (p,d) \leftarrow ps. f-prime-power\ p (Suc\ d))$
 $\langle proof \rangle$

lemma *efficient-code*:

assumes *set (ps ()) =* $(\lambda p. (p, multiplicity\ p\ n - 1))$ ' *prime-factors n distinct*
 $(ps\ ())$
shows $f n = (if\ n = 0\ then\ 0\ else\ (\prod (p,d) \leftarrow ps\ (). f-prime-power\ p (Suc\ d)))$
 $\langle proof \rangle$

end

```

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

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

sublocale multiplicative-function  $f$ 
  <proof>

lemma prod:  $f\ (\text{prod}\ g\ A) = (\prod_{x \in A}. f\ (g\ x))$ 
  <proof>

lemma power:  $f\ (n \wedge m) = f\ n \wedge m$ 
  <proof>

lemma prod-prime-factors':  $n > 0 \implies f\ n = (\prod_{p \in \text{prime-factors}\ n}. f\ p \wedge \text{multiplicity}\ p\ n)$ 
  <proof>

end

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

lemma prod-prime-factors'':  $n > 0 \implies f\ n = (\prod_{p \in \text{prime-factors}\ n}. f\text{-prime}\ p \wedge \text{multiplicity}\ p\ n)$ 
  <proof>

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

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

end

```

lemma *multiplicative-function-eqI*:

assumes *multiplicative-function f multiplicative-function g*
assumes $\bigwedge p k. \text{prime } p \implies k > 0 \implies f (p \wedge k) = g (p \wedge k)$
shows $f n = g n$

<proof>

lemma *multiplicative-function-of-natI*:

multiplicative-function f \implies *multiplicative-function* $(\lambda n. \text{of-nat } (f n))$

<proof>

lemma *multiplicative-function-of-natD*:

multiplicative-function $(\lambda n. \text{of-nat } (f n)) :: 'a :: \{\text{ring-char-0, comm-semiring-1}\}$

\implies

multiplicative-function f

<proof>

lemma *multiplicative-function-mult*:

assumes *multiplicative-function f multiplicative-function g*
shows *multiplicative-function* $(\lambda n. f n * g n)$

<proof>

lemma *multiplicative-function-inverse*:

fixes $f :: \text{nat} \Rightarrow 'a :: \text{field}$
assumes *multiplicative-function f*
shows *multiplicative-function* $(\lambda n. \text{inverse } (f n))$

<proof>

lemma *multiplicative-function-divide*:

fixes $f :: \text{nat} \Rightarrow 'a :: \text{field}$
assumes *multiplicative-function f multiplicative-function g*
shows *multiplicative-function* $(\lambda n. f n / g n)$

<proof>

lemma *completely-multiplicative-function-mult*:

assumes *completely-multiplicative-function f completely-multiplicative-function g*
shows *completely-multiplicative-function* $(\lambda n. f n * g n)$

<proof>

lemma *completely-multiplicative-function-inverse*:

fixes $f :: \text{nat} \Rightarrow 'a :: \text{field}$
assumes *completely-multiplicative-function f*
shows *completely-multiplicative-function* $(\lambda n. \text{inverse } (f n))$

<proof>

lemma *completely-multiplicative-function-divide*:

fixes $f :: \text{nat} \Rightarrow 'a :: \text{field}$
assumes *completely-multiplicative-function f completely-multiplicative-function*

g

shows *completely-multiplicative-function* $(\lambda n. f n / g n)$
 $\langle proof \rangle$

lemma (*in multiplicative-function*) *completely-multiplicativeI*:
assumes $\bigwedge p k. \text{prime } p \implies k > 0 \implies f (p \wedge k) = f p \wedge k$
shows *completely-multiplicative-function* f
 $\langle proof \rangle$

2.2 Indicator function

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

lemma *ind-0* [*simp*]: $\text{ind } P 0 = 0$ $\langle proof \rangle$

lemma *ind-nonzero*: $n > 0 \implies \text{ind } P n = (\text{if } P n \text{ then } 1 \text{ else } 0)$
 $\langle proof \rangle$

lemma *ind-True* [*simp*]: $P n \implies n > 0 \implies \text{ind } P n = 1$
 $\langle proof \rangle$

lemma *ind-False* [*simp*]: $\neg P n \implies n > 0 \implies \text{ind } P n = 0$
 $\langle proof \rangle$

lemma *ind-eq-1-iff*: $\text{ind } P n = 1 \iff n > 0 \wedge P n$
 $\langle proof \rangle$

lemma *ind-eq-0-iff*: $\text{ind } P n = 0 \iff n = 0 \vee \neg P n$
 $\langle proof \rangle$

lemma *multiplicative-function-ind* [*intro?*]:
assumes $P 1 \wedge a b. a > 1 \implies b > 1 \implies \text{coprime } a b \implies P (a * b) \iff P a$
 $\wedge P b$
shows *multiplicative-function* $(\text{ind } P)$
 $\langle proof \rangle$

end

3 Dirichlet convolution

theory *Dirichlet-Product*

imports

Complex-Main

Multiplicative-Function

begin

lemma *sum-coprime-dvd-cong*:

$(\sum r \mid r \text{ dvd } a. \sum s \mid s \text{ dvd } b. f r s) = (\sum r \mid r \text{ dvd } a. \sum s \mid s \text{ dvd } b. g r s)$
if $\text{coprime } a b \wedge r s. \text{coprime } r s \implies r \text{ dvd } a \implies s \text{ dvd } b \implies f r s = g r s$

<proof>

definition *dirichlet-prod* :: (nat \Rightarrow 'a :: semiring-0) \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a

where

$$\text{dirichlet-prod } f \ g = (\lambda n. \sum d \mid d \text{ dvd } n. f \ d * g \ (n \text{ div } d))$$

lemma *sum-divisors-code*:

assumes $n > (0 :: \text{nat})$

shows $(\sum d \mid d \text{ dvd } n. f \ d) =$

$$\text{fold-atLeastAtMost-nat } (\lambda d \text{ acc. if } d \text{ dvd } n \text{ then } f \ d + \text{acc else acc}) \ 1 \ n \ 0$$

<proof>

lemma *dirichlet-prod-code* [code]:

dirichlet-prod $f \ g \ n = (\text{if } n = 0 \text{ then } 0 \text{ else}$

$\text{fold-atLeastAtMost-nat } (\lambda d \text{ acc. if } d \text{ dvd } n \text{ then } f \ d * g \ (n \text{ div } d) + \text{acc else acc}) \ 1 \ n \ 0)$

<proof>

lemma *dirichlet-prod-0* [simp]: *dirichlet-prod* $f \ g \ 0 = 0$

<proof>

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

<proof>

<proof>

lemma *dirichlet-prod-cong* [cong]:

assumes $(\bigwedge n. n > 0 \implies f \ n = f' \ n) \ (\bigwedge n. n > 0 \implies g \ n = g' \ n)$

shows *dirichlet-prod* $f \ g = \text{dirichlet-prod } f' \ g'$

<proof>

lemma *dirichlet-prod-altdef1*:

$$\text{dirichlet-prod } f \ g = (\lambda n. \sum d \mid d \text{ dvd } n. f \ (n \text{ div } d) * g \ d)$$

<proof>

lemma *dirichlet-prod-altdef2*:

$$\text{dirichlet-prod } f \ g = (\lambda n. \sum (r, d) \mid r * d = n. f \ r * g \ d)$$

<proof>

lemma *dirichlet-prod-commutes*:

$$\text{dirichlet-prod } (f :: \text{nat} \Rightarrow 'a :: \text{comm-semiring-0}) \ g = \text{dirichlet-prod } g \ f$$

<proof>

lemma *finite-divisors-nat'*: $n > (0 :: \text{nat}) \implies \text{finite } \{(a, b). a * b = n\}$

<proof>

lemma *dirichlet-prod-assoc-aux1*:

assumes $n > 0$

shows *dirichlet-prod* $f \ (\text{dirichlet-prod } g \ h) \ n =$

$$(\sum (a, b, c) \in \{(a, b, c). a * b * c = n\}. f \ a * g \ b * h \ c)$$

<proof>

lemma *dirichlet-prod-assoc-aux2*:

assumes $n > 0$

shows $\text{dirichlet-prod } (\text{dirichlet-prod } f \ g) \ h \ n =$

$$\left(\sum (a, b, c) \in \{(a, b, c). a * b * c = n\}. f \ a * g \ b * h \ c \right)$$

<proof>

lemma *dirichlet-prod-assoc*:

$\text{dirichlet-prod } (\text{dirichlet-prod } f \ g) \ h = \text{dirichlet-prod } f \ (\text{dirichlet-prod } g \ h)$

<proof>

lemma *dirichlet-prod-const-right* [simp]:

assumes $n > 0$

shows $\text{dirichlet-prod } f \ (\lambda n. \text{if } n = \text{Suc } 0 \text{ then } c \text{ else } 0) \ n = f \ n * c$

<proof>

lemma *dirichlet-prod-const-left* [simp]:

assumes $n > 0$

shows $\text{dirichlet-prod } (\lambda n. \text{if } n = \text{Suc } 0 \text{ then } c \text{ else } 0) \ g \ n = c * g \ n$

<proof>

fun *dirichlet-inverse* :: $(\text{nat} \Rightarrow 'a :: \text{comm-ring-1}) \Rightarrow 'a \Rightarrow \text{nat} \Rightarrow 'a$ **where**

$\text{dirichlet-inverse } f \ i \ n =$

$(\text{if } n = 0 \text{ then } 0 \text{ else if } n = 1 \text{ then } i$

$\text{else } -i * (\sum d \mid d \ \text{dvd} \ n \wedge d < n. f \ (n \ \text{div} \ d) * \text{dirichlet-inverse } f \ i \ d))$

lemma *dirichlet-inverse-induct* [case-names 0 1 gt1]:

$P \ 0 \Longrightarrow P \ (\text{Suc } 0) \Longrightarrow (\bigwedge n. n > 1 \Longrightarrow (\bigwedge k. k < n \Longrightarrow P \ k) \Longrightarrow P \ n) \Longrightarrow P \ n$

<proof>

lemma *dirichlet-inverse-0* [simp]: $\text{dirichlet-inverse } f \ i \ 0 = 0$

<proof>

lemma *dirichlet-inverse-Suc-0* [simp]: $\text{dirichlet-inverse } f \ i \ (\text{Suc } 0) = i$

<proof>

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

lemma *dirichlet-inverse-gt-1*:

$n > 1 \Longrightarrow \text{dirichlet-inverse } f \ i \ n =$

$-i * (\sum d \mid d \ \text{dvd} \ n \wedge d < n. f \ (n \ \text{div} \ d) * \text{dirichlet-inverse } f \ i \ d)$

<proof>

lemma *dirichlet-inverse-cong* [cong]:

assumes $\bigwedge n. n > 0 \Longrightarrow f \ n = f' \ n \ i = i' \ n = n'$

shows $\text{dirichlet-inverse } f \ i \ n = \text{dirichlet-inverse } f' \ i' \ n'$

<proof>

lemma *dirichlet-inverse-gt-1'*:
assumes $n > 1$
shows $\text{dirichlet-inverse } f \ i \ n =$
 $-i * \text{dirichlet-prod } (\lambda n. \text{if } n = 1 \text{ then } 0 \text{ else } f \ n) \ (\text{dirichlet-inverse } f \ i) \ n$
 $\langle \text{proof} \rangle$

lemma *of-int-dirichlet-prod*:
 $\text{of-int } (\text{dirichlet-prod } f \ g \ n) = \text{dirichlet-prod } (\lambda n. \text{of-int } (f \ n)) \ (\lambda n. \text{of-int } (g \ n)) \ n$
 $\langle \text{proof} \rangle$

lemma *of-int-dirichlet-inverse*:
 $\text{of-int } (\text{dirichlet-inverse } f \ i \ n) = \text{dirichlet-inverse } (\lambda n. \text{of-int } (f \ n)) \ (\text{of-int } i) \ n$
 $\langle \text{proof} \rangle$

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

lemma *dirichlet-prod-inverse*:
assumes $f \ 1 * i = 1$
shows $\text{dirichlet-prod } f \ (\text{dirichlet-inverse } f \ i) = (\lambda n. \text{if } n = 1 \text{ then } 1 \text{ else } 0)$
 $\langle \text{proof} \rangle$

lemma *dirichlet-prod-inverse'*:
assumes $f \ 1 * i = 1$
shows $\text{dirichlet-prod } (\text{dirichlet-inverse } f \ i) \ f = (\lambda n. \text{if } n = 1 \text{ then } 1 \text{ else } 0)$
 $\langle \text{proof} \rangle$

lemma *dirichlet-inverse-noninvertible*:
assumes $f \ (\text{Suc } 0) = (0 :: 'a :: \{\text{comm-ring-1}\}) \ i = 0$
shows $\text{dirichlet-inverse } f \ i \ n = 0$
 $\langle \text{proof} \rangle$

lemma *multiplicative-dirichlet-prod*:
assumes *multiplicative-function* f
assumes *multiplicative-function* g
shows *multiplicative-function* $(\text{dirichlet-prod } f \ g)$
 $\langle \text{proof} \rangle$

lemma *multiplicative-dirichlet-prodD1*:
fixes $f \ g :: \text{nat} \Rightarrow 'a :: \text{comm-semiring-1-cancel}$
assumes *multiplicative-function* $(\text{dirichlet-prod } f \ g)$
assumes *multiplicative-function* f
assumes [simp]: $g \ 0 = 0$
shows *multiplicative-function* g
 $\langle \text{proof} \rangle$

```

lemma multiplicative-dirichlet-prodD2:
  fixes  $f\ g :: \text{nat} \Rightarrow 'a :: \text{comm-semiring-1-cancel}$ 
  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  $\text{dirichlet-prod } f\ g\ (p \wedge k) = (\sum_{i \leq k}. f\ (p \wedge i) * g\ (p \wedge (k - i)))$ 
  <proof>

lemma dirichlet-prod-prime:
  assumes prime p
  shows  $\text{dirichlet-prod } f\ g\ p = f\ 1 * g\ p + f\ p * g\ 1$ 
  <proof>

locale multiplicative-dirichlet-prod =
   $f: \text{multiplicative-function } f + g: \text{multiplicative-function } g$ 
  for  $f\ g :: \text{nat} \Rightarrow 'a :: \text{comm-semiring-1}$ 
begin

  sublocale multiplicative-function dirichlet-prod f g
  <proof>

end

locale multiplicative-dirichlet-prod' =
   $f: \text{multiplicative-function}' f\ f\text{-prime-power } f\text{-prime} +$ 
   $g: \text{multiplicative-function}' g\ g\text{-prime-power } g\text{-prime}$ 
  for  $f\ g :: \text{nat} \Rightarrow 'a :: \text{comm-semiring-1}$  and  $f\text{-prime-power } g\text{-prime-power } f\text{-prime}$ 
   $g\text{-prime}$ 
begin

  sublocale multiplicative-dirichlet-prod f g <proof>

  sublocale multiplicative-function' dirichlet-prod f g
   $\lambda p\ k. f\text{-prime-power } p\ k + g\text{-prime-power } p\ k +$ 
   $(\sum_{i \in \{0 <..<k\}}. f\text{-prime-power } p\ i * g\text{-prime-power } p\ (k - i))$ 
   $\lambda p. f\text{-prime } p + g\text{-prime } p$ 
  <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 $\text{nat} \Rightarrow 'a$ whose value for n is the $n + 1$ -th coefficient.

typedef $'a \text{ fds} = \text{UNIV} :: (\text{nat} \Rightarrow 'a) \text{ set}$
<proof>

setup-lifting *type-definition-fds*

lift-definition $\text{fds-nth} :: 'a \text{ fds} \Rightarrow \text{nat} \Rightarrow 'a :: \text{zero is}$
 $\lambda f :: \text{nat} \Rightarrow 'a. \text{case-nat } 0 \ f \ \langle \text{proof} \rangle$

lift-definition $\text{fds} :: (\text{nat} \Rightarrow 'a) \Rightarrow 'a \text{ fds is}$
 $\lambda f. f \circ \text{Suc} \ \langle \text{proof} \rangle$

lemma $\text{fds-nth-fds}: \text{fds-nth} (\text{fds } f) \ n = (\text{if } n = 0 \ \text{then } 0 \ \text{else } f \ n)$
<proof>

lemma $\text{fds-nth-fds}' : f \ 0 = 0 \implies \text{fds-nth} (\text{fds } f) = f$
<proof>

lemma $\text{fds-nth-0} \ [\text{simp}]: \text{fds-nth } f \ 0 = 0$
<proof>

lemma $\text{fds-nth-fds-pos} \ [\text{simp}]: n > 0 \implies \text{fds-nth} (\text{fds } f) \ n = f \ n$
<proof>

lemma $\text{fds-fds-nth} \ [\text{simp}]: \text{fds} (\text{fds-nth } f) = f$
<proof>

lemma *fds-eq-fds-iff*:

$fds\ f = fds\ g \longleftrightarrow (\forall n > 0. f\ n = g\ n)$
<proof>

lemma *fds-eq-fds-iff'*: $f\ 0 = g\ 0 \implies fds\ f = fds\ g \longleftrightarrow f = g$
<proof>

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

assumes $(\bigwedge n. n > 0 \implies fds\text{-}nth\ f\ n = fds\text{-}nth\ g\ n)$
shows $f = g$
<proof>

lemma *fds-cong* [*cong*]: $(\bigwedge n. n > 0 \implies f\ n = (g\ n :: 'a :: zero)) \implies fds\ f = fds\ g$
<proof>

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

lemma *dirichlet-prod-fds-nth-fds-left* [*simp*]:

$dirichlet\text{-}prod\ (fds\text{-}nth\ (fds\ f))\ g = dirichlet\text{-}prod\ f\ g$
<proof>

lemma *dirichlet-prod-fds-nth-fds-right* [*simp*]:

$dirichlet\text{-}prod\ f\ (fds\text{-}nth\ (fds\ g)) = dirichlet\text{-}prod\ f\ g$
<proof>

definition *fds-const* :: $'a :: zero \Rightarrow 'a\ fds$ **where**

$fds\text{-}const\ c = fds\ (\lambda n. \text{if } n = 1 \text{ then } c \text{ else } 0)$

abbreviation *fds-ind* **where** $fds\text{-}ind\ P \equiv fds\ (ind\ P)$

bundle *fds-syntax*

begin

notation *fds-nth* (**infixl** \$ 75)

notation *fds* (**binder** χ 10)

notation *dirichlet-prod* (**infixl** \star 70)

end

instantiation *fds* :: $(zero)\ zero$

begin

definition *zero-fds* :: $'a\ fds$ **where** $zero\text{-}fds = fds\ (\lambda\cdot. 0)$

instance *<proof>*

end

instantiation $fds :: (\{zero,one\})\ one$
begin
definition $one-fds :: 'a\ fds\ \mathbf{where}\ one-fds = fds\ (\lambda n.\ \mathit{if}\ n = 1\ \mathit{then}\ 1\ \mathit{else}\ 0)$
instance $\langle proof \rangle$
end

instantiation $fds :: (\{plus,zero\})\ plus$
begin
definition $plus-fds :: 'a\ fds \Rightarrow 'a\ fds \Rightarrow 'a\ fds$
where $plus-fds\ f\ g = fds\ (\lambda n.\ fds-nth\ f\ n + fds-nth\ g\ n)$
instance $\langle proof \rangle$
end

instantiation $fds :: (semiring-0)\ times$
begin
definition $times-fds :: 'a\ fds \Rightarrow 'a\ fds \Rightarrow 'a\ fds$
where $times-fds\ f\ g = fds\ (dirichlet-prod\ (fds-nth\ f)\ (fds-nth\ g))$
instance $\langle proof \rangle$
end

instantiation $fds :: (\{uminus,zero\})\ uminus$
begin
definition $uminus-fds :: 'a\ fds \Rightarrow 'a\ fds$
where $uminus-fds\ f = fds\ (\lambda n.\ -fds-nth\ f\ n)$
instance $\langle proof \rangle$
end

instantiation $fds :: (\{minus,zero\})\ minus$
begin
definition $minus-fds :: 'a\ fds \Rightarrow 'a\ fds \Rightarrow 'a\ fds$
where $minus-fds\ f\ g = fds\ (\lambda n.\ fds-nth\ f\ n - fds-nth\ g\ n)$
instance $\langle proof \rangle$
end

4.1 General properties

lemma $fds-nth-zero$ [*simp*]: $fds-nth\ 0 = (\lambda -. 0)$
 $\langle proof \rangle$

lemma $fds-nth-one$: $fds-nth\ 1 = (\lambda n.\ \mathit{if}\ n = 1\ \mathit{then}\ 1\ \mathit{else}\ 0)$
 $\langle proof \rangle$

lemma $fds-nth-one-Suc-0$ [*simp*]: $fds-nth\ 1\ (Suc\ 0) = 1$
 $\langle proof \rangle$

lemma $fds-nth-one-not-Suc-0$ [*simp*]: $n \neq Suc\ 0 \Longrightarrow fds-nth\ 1\ n = 0$
 $\langle proof \rangle$

lemma *fds-nth-plus* [*simp*]:
 $fds\text{-}nth\ (f + g) = (\lambda n. fds\text{-}nth\ f\ n + fds\text{-}nth\ g\ n :: 'a :: monoid\text{-}add)$
 $\langle proof \rangle$

lemma *fds-nth-minus* [*simp*]:
 $fds\text{-}nth\ (f - g) = (\lambda n. fds\text{-}nth\ f\ n - fds\text{-}nth\ g\ n :: 'a :: \{cancel\text{-}comm\text{-}monoid\text{-}add\})$
 $\langle proof \rangle$

lemma *fds-nth-uminus* [*simp*]: $fds\text{-}nth\ (-g) = (\lambda n. -\ fds\text{-}nth\ g\ n :: 'a :: group\text{-}add)$
 $\langle proof \rangle$

lemma *fds-nth-mult*: $fds\text{-}nth\ (f * g) = dirichlet\text{-}prod\ (fds\text{-}nth\ f)\ (fds\text{-}nth\ g)$
 $\langle proof \rangle$

lemma *fds-nth-mult-const-left* [*simp*]: $fds\text{-}nth\ (fds\text{-}const\ c * f)\ n = c * fds\text{-}nth\ f\ n$
 $\langle proof \rangle$

lemma *fds-nth-mult-const-right* [*simp*]: $fds\text{-}nth\ (f * fds\text{-}const\ c)\ n = fds\text{-}nth\ f\ n * c$
 $\langle proof \rangle$

instance *fds* :: ($\{semigroup\text{-}add, zero\}$) *semigroup-add*
 $\langle proof \rangle$

instance *fds* :: ($\{ab\text{-}semigroup\text{-}add, zero\}$) *ab-semigroup-add*
 $\langle proof \rangle$

instance *fds* :: ($\{cancel\text{-}semigroup\text{-}add, zero\}$) *cancel-semigroup-add*
 $\langle proof \rangle$

instance *fds* :: ($\{cancel\text{-}ab\text{-}semigroup\text{-}add, zero\}$) *cancel-ab-semigroup-add*
 $\langle proof \rangle$

instance *fds* :: (*monoid-add*) *monoid-add*
 $\langle proof \rangle$

instance *fds* :: (*comm-monoid-add*) *comm-monoid-add*
 $\langle proof \rangle$

instance *fds* :: (*cancel-comm-monoid-add*) *cancel-comm-monoid-add*
 $\langle proof \rangle$

instance *fds* :: (*group-add*) *group-add*
 $\langle proof \rangle$

instance *fds* :: (*ab-group-add*) *ab-group-add*
 $\langle proof \rangle$

```

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 (λn. c *R fds-nth f n)

lemma fds-nth-scaleR [simp]: fds-nth (c *R f) = (λn. 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 (λx. fds-nth (f x) n) A

```

<proof>

lemma *sum-fds* [*simp*]: $(\sum x \in A. \text{fds } (f x)) = \text{fds } (\lambda n. \sum x \in A. f x n)$
<proof>

lemma *fds-nth-const*: $\text{fds-nth } (\text{fds-const } c) = (\lambda n. \text{if } n = 1 \text{ then } c \text{ else } 0)$
<proof>

lemma *fds-nth-const-Suc-0* [*simp*]: $\text{fds-nth } (\text{fds-const } c) (\text{Suc } 0) = c$
<proof>

lemma *fds-nth-const-not-Suc-0* [*simp*]: $n \neq 1 \implies \text{fds-nth } (\text{fds-const } c) n = 0$
<proof>

lemma *fds-const-zero* [*simp*]: $\text{fds-const } 0 = 0$
<proof>

lemma *fds-const-one* [*simp*]: $\text{fds-const } 1 = 1$
<proof>

lemma *fds-const-add* [*simp*]: $\text{fds-const } (a + b :: 'a :: \text{monoid-add}) = \text{fds-const } a + \text{fds-const } b$
<proof>

lemma *fds-const-minus* [*simp*]:
 $\text{fds-const } (a - b :: 'a :: \text{cancel-comm-monoid-add}) = \text{fds-const } a - \text{fds-const } b$
<proof>

lemma *fds-const-uminus* [*simp*]:
 $\text{fds-const } (- b :: 'a :: \text{ab-group-add}) = - \text{fds-const } b$
<proof>

lemma *fds-const-mult* [*simp*]:
 $\text{fds-const } (a * b :: 'a :: \text{semiring-0}) = \text{fds-const } a * \text{fds-const } b$
<proof>

lemma *fds-const-of-nat* [*simp*]: $\text{fds-const } (\text{of-nat } c) = \text{of-nat } c$
<proof>

lemma *fds-const-of-int* [*simp*]: $\text{fds-const } (\text{of-int } c) = \text{of-int } c$
<proof>

lemma *fds-const-of-real* [*simp*]: $\text{fds-const } (\text{of-real } c) = \text{of-real } c$
<proof>

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

definition *inverse-fds* :: 'a fds \Rightarrow 'a fds **where**
inverse-fds f = fds (λn . dirichlet-inverse (fds-nth f) (inverse (fds-nth f 1)) n)

definition *divide-fds* :: 'a fds \Rightarrow 'a fds \Rightarrow 'a fds **where**
divide-fds f g = f * inverse g

instance \langle proof \rangle

end

lemma *numeral-fds*: numeral n = fds-const (numeral n)
 \langle proof \rangle

lemma *fds-ind-False* [simp]: fds-ind (λ -. False) = 0
 \langle proof \rangle

lemma *fds-commutes*:

assumes $\bigwedge m n$. $m > 0 \implies n > 0 \implies$ fds-nth f m * fds-nth g n = fds-nth g n
* fds-nth f m
shows f * g = g * f
 \langle proof \rangle

lemma *fds-nth-mult-Suc-0* [simp]:
fds-nth (f * g) (Suc 0) = fds-nth f (Suc 0) * fds-nth g (Suc 0)
 \langle proof \rangle

lemma *fds-nth-inverse*:

fds-nth (inverse f) = dirichlet-inverse (fds-nth f) (inverse (fds-nth f 1))
 \langle proof \rangle

lemma *inverse-fds-nonunit*:

fds-nth f 1 = (0 :: 'a :: field) \implies inverse f = 0
 \langle proof \rangle

lemma *inverse-0-fds* [simp]: inverse (0 :: 'a :: field fds) = 0
 \langle proof \rangle

lemma *fds-left-inverse*:

fds-nth f 1 \neq (0 :: 'a :: field) \implies inverse f * f = 1
 \langle proof \rangle

lemma *fds-right-inverse*:

fds-nth f 1 \neq (0 :: 'a :: field) \implies f * inverse f = 1
 \langle proof \rangle

lemma *fds-left-inverse-unique*:

assumes f * g = (1 :: 'a :: field fds)
shows f = inverse g
 \langle proof \rangle

lemma *fds-right-inverse-unique*:

assumes $f * g = (1 :: 'a :: \text{field } \text{fds})$

shows $g = \text{inverse } f$

<proof>

lemma *inverse-1-fds [simp]*: $\text{inverse } (1 :: 'a :: \text{field } \text{fds}) = 1$

<proof>

lemma *inverse-const-fds [simp]*:

$\text{inverse } (\text{fds-const } c :: 'a :: \text{field } \text{fds}) = \text{fds-const } (\text{inverse } c)$

<proof>

lemma *inverse-mult-fds*: $\text{inverse } (f * g :: 'a :: \text{field } \text{fds}) = \text{inverse } f * \text{inverse } g$

<proof>

definition *fds-zeta* :: $'a :: \text{one } \text{fds}$

where $\text{fds-zeta} = \text{fds } (\lambda-. 1)$

lemma *fds-zeta-altdef*: $\text{fds-zeta} = \text{fds } (\lambda n. \text{if } n = 0 \text{ then } 0 \text{ else } 1)$

<proof>

lemma *fds-nth-zeta*: $\text{fds-nth } \text{fds-zeta} = (\lambda n. \text{if } n = 0 \text{ then } 0 \text{ else } 1)$

<proof>

lemma *fds-nth-zeta-pos [simp]*: $n > 0 \implies \text{fds-nth } \text{fds-zeta } n = 1$

<proof>

lemma *fds-zeta-commutes*: $\text{fds-zeta} * (f :: 'a :: \text{semiring-1 } \text{fds}) = f * \text{fds-zeta}$

<proof>

lemma *fds-ind-True [simp]*: $\text{fds-ind } (\lambda-. \text{True}) = \text{fds-zeta}$

<proof>

lemma *finite-extensional-prod-nat*:

assumes $\text{finite } A \ b > 0$

shows $\text{finite } \{d \in \text{extensional } A. \text{prod } d \ A = (b :: \text{nat})\}$

<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 $\text{finite } A \ A \neq \{\} \ n > 0$

shows $\text{fds-nth } (\prod_{x \in A}. f \ x) \ n =$

$(\sum d \mid d \in \text{extensional } A \wedge \text{prod } d \ A = n. \prod_{x \in A}. \text{fds-nth } (f \ x) \ (d \ x))$

<proof>

lemma *fds-nth-power-Suc-0* [simp]: $\text{fds-nth } (f \wedge n) (\text{Suc } 0) = \text{fds-nth } f (\text{Suc } 0) \wedge n$

<proof>

lemma *fds-nth-prod-Suc-0* [simp]: $\text{fds-nth } (\text{prod } f A) (\text{Suc } 0) = (\prod_{x \in A} \text{fds-nth } (f x) (\text{Suc } 0))$

<proof>

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

assumes $n < 2 \wedge k \text{ fds-nth } f 1 = 0$

shows $\text{fds-nth } (f \wedge k) n = 0$

<proof>

4.2 Shifting the argument

class *nat-power* = *semiring-1* +

fixes *nat-power* :: $\text{nat} \Rightarrow 'a \Rightarrow 'a$

assumes *nat-power-0-left* [simp]: $x \neq 0 \implies \text{nat-power } 0 x = 0$

assumes *nat-power-0-right* [simp]: $n > 0 \implies \text{nat-power } n 0 = 1$

assumes *nat-power-1-left* [simp]: $\text{nat-power } (\text{Suc } 0) x = 1$

assumes *nat-power-1-right* [simp]: $\text{nat-power } n 1 = \text{of-nat } n$

assumes *nat-power-add*: $n > 0 \implies \text{nat-power } n (a + b) = \text{nat-power } n a * \text{nat-power } n b$

assumes *nat-power-mult-distrib*:

$m > 0 \implies n > 0 \implies \text{nat-power } (m * n) a = \text{nat-power } m a * \text{nat-power } n a$

assumes *nat-power-power*:

$n > 0 \implies \text{nat-power } n (a * \text{of-nat } m) = \text{nat-power } n a \wedge m$

begin

lemma *nat-power-of-nat* [simp]: $m > 0 \implies \text{nat-power } m (\text{of-nat } n) = \text{of-nat } (m \wedge n)$

<proof>

lemma *nat-power-power-left*: $m > 0 \implies \text{nat-power } (m \wedge k) n = \text{nat-power } m n \wedge k$

<proof>

end

class *nat-power-field* = *nat-power* + *field* +

assumes *nat-power-nonzero* [simp]: $n > 0 \implies \text{nat-power } n z \neq 0$

begin

lemma *nat-power-diff*: $n > 0 \implies \text{nat-power } n (a - b) = \text{nat-power } n a / \text{nat-power } n b$

<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* (λ*n*. *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 $f :: 'a :: \{\text{ring}, \text{nat-power}\}$ *fds*
shows $\text{fds-shift } c \ (-f) = -\text{fds-shift } c \ f$
 $\langle \text{proof} \rangle$

lemma *fds-shift-mult* [*simp*]:
fixes $f \ g :: 'a :: \{\text{comm-semiring}, \text{nat-power}\}$ *fds*
shows $\text{fds-shift } c \ (f * g) = \text{fds-shift } c \ f * \text{fds-shift } c \ g$
 $\langle \text{proof} \rangle$

lemma *fds-shift-power* [*simp*]:
fixes $f :: 'a :: \{\text{comm-semiring}, \text{nat-power}\}$ *fds*
shows $\text{fds-shift } c \ (f \wedge n) = \text{fds-shift } c \ f \wedge n$
 $\langle \text{proof} \rangle$

lemma *fds-shift-by-0* [*simp*]: $\text{fds-shift } 0 \ f = f$
 $\langle \text{proof} \rangle$

lemma *fds-shift-inverse* [*simp*]:
 $\text{fds-shift } (a :: 'a :: \{\text{field}, \text{nat-power}\}) \ (\text{inverse } f) = \text{inverse } (\text{fds-shift } a \ f)$
 $\langle \text{proof} \rangle$

lemma *fds-shift-divide* [*simp*]:
 $\text{fds-shift } (a :: 'a :: \{\text{field}, \text{nat-power}\}) \ (f / g) = \text{fds-shift } a \ f / \text{fds-shift } a \ g$
 $\langle \text{proof} \rangle$

lemma *fds-shift-sum* [*simp*]: $\text{fds-shift } a \ (\sum x \in A. f \ x) = (\sum x \in A. \text{fds-shift } a \ (f \ x))$
 $\langle \text{proof} \rangle$

lemma *fds-shift-prod* [*simp*]: $\text{fds-shift } a \ (\prod x \in A. f \ x) = (\prod x \in A. \text{fds-shift } a \ (f \ x))$
 $\langle \text{proof} \rangle$

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* :: $\text{nat} \Rightarrow ('a :: \text{zero}) \text{fds} \Rightarrow 'a \text{fds}$ **where**
 $\text{fds-scale } c \ f =$
 $\text{fds } (\lambda n. \text{if } n > 0 \wedge \text{is-nth-power } c \ n \text{ then } \text{fds-nth } f \ (\text{nth-root-nat } c \ n) \text{ else } 0)$

lemma *fds-scale-0* [*simp*]: $\text{fds-scale } 0 \ f = 0$
<proof>

lemma *fds-scale-1* [*simp*]: $\text{fds-scale } 1 \ f = f$
<proof>

lemma *fds-nth-scale-power* [*simp*]:
 $c > 0 \implies \text{fds-nth } (\text{fds-scale } c \ f) \ (n \wedge c) = \text{fds-nth } f \ n$
<proof>

lemma *fds-nth-scale-nonpower* [*simp*]:
 $\neg \text{is-nth-power } c \ n \implies \text{fds-nth } (\text{fds-scale } c \ f) \ n = 0$
<proof>

lemma *fds-nth-scale*:
 $\text{fds-nth } (\text{fds-scale } c \ f) \ n =$
 $(\text{if } n > 0 \wedge \text{is-nth-power } c \ n \text{ then } \text{fds-nth } f \ (\text{nth-root-nat } c \ n) \text{ else } 0)$
<proof>

lemma *fds-scale-const* [*simp*]: $c > 0 \implies \text{fds-scale } c \ (\text{fds-const } c') = \text{fds-const } c'$
<proof>

lemma *fds-scale-zero* [*simp*]: $\text{fds-scale } c \ 0 = 0$
<proof>

lemma *fds-scale-one* [*simp*]: $c > 0 \implies \text{fds-scale } c \ 1 = 1$
<proof>

lemma *fds-scale-of-nat* [*simp*]: $c > 0 \implies \text{fds-scale } c \ (\text{of-nat } n) = \text{of-nat } n$
<proof>

lemma *fds-scale-of-int* [*simp*]: $c > 0 \implies \text{fds-scale } c \ (\text{of-int } n) = \text{of-int } n$
<proof>

lemma *fds-scale-numeral* [*simp*]: $c > 0 \implies \text{fds-scale } c \ (\text{numeral } n) = \text{numeral } n$
<proof>

lemma *fds-scale-scale*: $\text{fds-scale } c \ (\text{fds-scale } c' \ f) = \text{fds-scale } (c * c') \ f$
<proof>

lemma *fds-scale-add* [*simp*]:
fixes $f \ g :: 'a :: \text{monoid-add } \text{fds}$
shows $\text{fds-scale } c \ (f + g) = \text{fds-scale } c \ f + \text{fds-scale } c \ g$
<proof>

lemma *fds-scale-minus* [*simp*]:
fixes $f \ g :: 'a :: \{\text{cancel-comm-monoid-add}\} \text{fds}$
shows $\text{fds-scale } c \ (f - g) = \text{fds-scale } c \ f - \text{fds-scale } c \ g$

<proof>

lemma *fds-scale-uminus* [*simp*]:
fixes $f :: 'a :: \text{group-add } \text{fds}$
shows $\text{fds-scale } c (-f) = -\text{fds-scale } c f$
<proof>

lemma *fds-scale-mult* [*simp*]:
fixes $f g :: 'a :: \text{semiring-0 } \text{fds}$
shows $\text{fds-scale } c (f * g) = \text{fds-scale } c f * \text{fds-scale } c g$
<proof>

lemma *fds-scale-shift*:
 $\text{fds-shift } d (\text{fds-scale } c f) = \text{fds-scale } c (\text{fds-shift } (c * d) f)$
<proof>

lemma *fds-ind-nth-power*: $k > 0 \implies \text{fds-ind } (\text{is-nth-power } k) = \text{fds-scale } k \text{fds-zeta}$
<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* :: $'a :: \text{real-algebra } \text{fds} \implies 'a \text{ fds}$ **where**
 $\text{fds-deriv } f = \text{fds } (\lambda n. - \ln (\text{real } n) *_{\mathbb{R}} \text{fds-nth } f n)$

lemma *fds-nth-deriv*: $\text{fds-nth } (\text{fds-deriv } f) n = -\ln (\text{real } n) *_{\mathbb{R}} \text{fds-nth } f n$
<proof>

lemma *fds-deriv-const* [*simp*]: $\text{fds-deriv } (\text{fds-const } c) = 0$
<proof>

lemma *fds-deriv-0* [*simp*]: $\text{fds-deriv } 0 = 0$
<proof>

lemma *fds-deriv-1* [*simp*]: $\text{fds-deriv } 1 = 0$
<proof>

lemma *fds-deriv-of-nat* [*simp*]: $\text{fds-deriv } (\text{of-nat } n) = 0$
<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 $fds-nth\ f\ (Suc\ 0) \neq 0$
shows $fds-deriv (inverse\ f) = -fds-deriv\ f / f^{\wedge} 2$
<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 $fds-deriv (fds\ f) = -fds (\lambda n. f\ n * mangoldt\ n) * fds\ f$
<proof>

lemma *completely-multiplicative-fds-deriv'*:
completely-multiplicative-function $(fds-nth\ f) \implies$
 $fds-deriv\ f = -fds (\lambda n. fds-nth\ f\ n * mangoldt\ n) * f$
<proof>

lemma *fds-deriv-zeta*:

$fds\text{-deriv } fds\text{-zeta} =$
 $-fds\text{ mangoldt} * (fds\text{-zeta} :: 'a :: \{comm\text{-semiring-1}, real\text{-algebra-1}\} fds)$
 $\langle proof \rangle$

lemma $fds\text{-mangoldt-times-zeta}$: $fds\text{ mangoldt} * fds\text{-zeta} = fds (\lambda x. of\text{-real} (\ln (\text{real } x)))$
 $\langle proof \rangle$

lemma $fds\text{-deriv-zeta}'$: $fds\text{-deriv } fds\text{-zeta} =$
 $-fds (\lambda x. of\text{-real} (\ln (\text{real } x))) :: 'a :: \{comm\text{-semiring-1}, real\text{-algebra-1}\}$
 $\langle proof \rangle$

4.5 Formal integral

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

lemma $fds\text{-integral-0}$ [simp]: $fds\text{-integral } a\ 0 = fds\text{-const } a$
 $\langle proof \rangle$

lemma $fds\text{-integral-add}$: $fds\text{-integral } (a + b)\ (f + g) = fds\text{-integral } a\ f + fds\text{-integral } b\ g$
 $\langle proof \rangle$

lemma $fds\text{-integral-diff}$: $fds\text{-integral } (a - b)\ (f - g) = fds\text{-integral } a\ f - fds\text{-integral } b\ g$
 $\langle proof \rangle$

lemma $fds\text{-integral-minus}$: $fds\text{-integral } (-a)\ (-f) = -fds\text{-integral } a\ f$
 $\langle proof \rangle$

lemma $fds\text{-shift-integral}$: $fds\text{-shift } b\ (fds\text{-integral } a\ f) = fds\text{-integral } a\ (fds\text{-shift } b\ f)$
 $\langle proof \rangle$

lemma $fds\text{-deriv-fds-integral}$ [simp]:
 $fds\text{-nth } f\ (Suc\ 0) = 0 \implies fds\text{-deriv } (fds\text{-integral } c\ f) = f$
 $\langle proof \rangle$

lemma $fds\text{-integral-fds-deriv}$ [simp]: $fds\text{-integral } (fds\text{-nth } f\ 1)\ (fds\text{-deriv } f) = f$
 $\langle proof \rangle$

4.6 Formal logarithm

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

lemma $fds\text{-nth-Suc-0-fds-deriv}$ [simp]: $fds\text{-nth } (fds\text{-deriv } f)\ (Suc\ 0) = 0$
 $\langle proof \rangle$

lemma *fds-deriv-fds-ln* [*simp*]: $\text{fds-deriv } (\text{fds-ln } l \ f) = \text{fds-deriv } f / f$
 ⟨*proof*⟩

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

lemma *fds-ln-const* [*simp*]: $\text{fds-ln } l \ (\text{fds-const } c) = \text{fds-const } l$
 ⟨*proof*⟩

lemma *fds-ln-0* [*simp*]: $\text{fds-ln } l \ 0 = \text{fds-const } l$
 ⟨*proof*⟩

lemma *fds-ln-1* [*simp*]: $\text{fds-ln } l \ 1 = \text{fds-const } l$
 ⟨*proof*⟩

lemma *fds-shift-ln* [*simp*]: $\text{fds-shift } a \ (\text{fds-ln } l \ f) = \text{fds-ln } l \ (\text{fds-shift } a \ f)$
 ⟨*proof*⟩

lemma *fds-ln-mult*:
assumes $\text{fds-nth } f \ 1 \neq 0 \ \text{fds-nth } g \ 1 \neq 0 \ l' + l'' = l$
shows $\text{fds-ln } l \ (f * g) = \text{fds-ln } l' \ f + \text{fds-ln } l'' \ g$
 ⟨*proof*⟩

lemma *fds-ln-power*:
assumes $\text{fds-nth } f \ 1 \neq 0 \ l = \text{of-nat } n * l'$
shows $\text{fds-ln } l \ (f \wedge^n) = \text{of-nat } n * \text{fds-ln } l' \ f$
 ⟨*proof*⟩

lemma *fds-ln-prod*:
assumes $\bigwedge x. x \in A \implies \text{fds-nth } (f \ x) \ 1 \neq 0 \ (\sum x \in A. l' \ x) = l$
shows $\text{fds-ln } l \ (\prod x \in A. f \ x) = (\sum x \in A. \text{fds-ln } (l' \ x) \ (f \ x))$
 ⟨*proof*⟩

4.7 Formal exponential

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

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

lemma *fds-exp-times-fds-nth-0*:
 $\text{fds-const } (\text{exp } (\text{fds-nth } f \ (\text{Suc } 0))) * \text{fds-exp } (f - \text{fds-const } (\text{fds-nth } f \ (\text{Suc } 0)))$
 $= \text{fds-exp } f$
 ⟨*proof*⟩

lemma *fds-exp-const* [*simp*]: $\text{fds-exp } (\text{fds-const } c) = \text{fds-const } (\text{exp } c)$
 ⟨*proof*⟩

lemma *fds-exp-numeral* [simp]: $\text{fds-exp } (\text{numeral } n) = \text{fds-const } (\text{exp } (\text{numeral } n))$
 ⟨proof⟩

lemma *fds-exp-0* [simp]: $\text{fds-exp } 0 = 1$
 ⟨proof⟩

lemma *fds-exp-1* [simp]: $\text{fds-exp } 1 = \text{fds-const } (\text{exp } 1)$
 ⟨proof⟩

lemma *fds-nth-Suc-0-exp* [simp]: $\text{fds-nth } (\text{fds-exp } f) (\text{Suc } 0) = \text{exp } (\text{fds-nth } f (\text{Suc } 0))$
 ⟨proof⟩

4.8 Subseries

definition *fds-subseries* :: $(\text{nat} \Rightarrow \text{bool}) \Rightarrow ('a :: \text{semiring-1}) \text{fds} \Rightarrow 'a \text{fds}$ **where**
fds-subseries $P f = \text{fds } (\lambda n. \text{if } P n \text{ then } \text{fds-nth } f n \text{ else } 0)$

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

lemma *fds-subseries-0* [simp]: $\text{fds-subseries } P 0 = 0$
 ⟨proof⟩

lemma *fds-subseries-1* [simp]: $P 1 \Longrightarrow \text{fds-subseries } P 1 = 1$
 ⟨proof⟩

lemma *fds-subseries-const* [simp]: $P 1 \Longrightarrow \text{fds-subseries } P (\text{fds-const } c) = \text{fds-const } c$
 ⟨proof⟩

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

lemma *fds-subseries-diff* [simp]:
 $\text{fds-subseries } P (f - g :: 'a :: \text{ring-1 } \text{fds}) = \text{fds-subseries } P f - \text{fds-subseries } P g$
 ⟨proof⟩

lemma *fds-subseries-minus* [simp]:
 $\text{fds-subseries } P (-f :: 'a :: \text{ring-1 } \text{fds}) = - \text{fds-subseries } P f$
 ⟨proof⟩

lemma *fds-subseries-sum* [simp]: $\text{fds-subseries } P (\sum x \in A. f x) = (\sum x \in A. \text{fds-subseries } P (f x))$
 ⟨proof⟩

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

$$\text{fds-subseries } P \text{ (fds-shift } c \text{ } f) = \text{fds-shift } c \text{ (fds-subseries } P \text{ } f)$$

<proof>

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

$$\text{fds-subseries } P \text{ (fds-deriv } f) = \text{fds-deriv (fds-subseries } P \text{ } f)$$

<proof>

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

$$P \ 1 \vee c = 0 \implies \text{fds-subseries } P \text{ (fds-integral } c \text{ } f) = \text{fds-integral } c \text{ (fds-subseries } P \text{ } f)$$

<proof>

abbreviation *fds-primelow-subseries* :: *nat* \Rightarrow (*'a* :: *semiring-1*) *fds* \Rightarrow *'a* *fds*
where

$$\text{fds-primelow-subseries } p \text{ } f \equiv \text{fds-subseries } (\lambda n. \text{prime-factors } n \subseteq \{p\}) \text{ } f$$

lemma *fds-primelow-subseries-mult* [simp]:

fixes *p* :: *nat*

defines *P* \equiv ($\lambda n. \text{prime-factors } n \subseteq \{p\}$)

shows $\text{fds-subseries } P \text{ (} f * g) = \text{fds-subseries } P \text{ } f * \text{fds-subseries } P \text{ } g$

<proof>

lemma *fds-primelow-subseries-power* [simp]:

$$\text{fds-primelow-subseries } p \text{ (} f \wedge n) = \text{fds-primelow-subseries } p \text{ } f \wedge n$$

<proof>

lemma *fds-primelow-subseries-prod* [simp]:

$$\text{fds-primelow-subseries } p \text{ (} \prod_{x \in A} f \text{ } x) = \prod_{x \in A} \text{fds-primelow-subseries } p \text{ (} f \text{ } x)$$

<proof>

lemma *completely-multiplicative-function-only-pows*:

assumes *completely-multiplicative-function* (*fds-nth* *f*)

shows *completely-multiplicative-function* (*fds-nth* (*fds-primelow-subseries* *p* *f*))

<proof>

4.9 Truncation

definition *fds-truncate* :: *nat* \Rightarrow *'a* :: {*zero*} *fds* \Rightarrow *'a* *fds* **where**

$$\text{fds-truncate } m \text{ } f = \text{fds } (\lambda n. \text{if } n \leq m \text{ then } \text{fds-nth } f \text{ } n \text{ else } 0)$$

lemma *fds-nth-truncate*: $\text{fds-nth (fds-truncate } m \text{ } f) \text{ } n = (\text{if } n \leq m \text{ then } \text{fds-nth } f \text{ } n \text{ else } 0)$

<proof>

lemma *fds-truncate-0* [simp]: $\text{fds-truncate } 0 \text{ } f = 0$

<proof>

lemma *fds-truncate-zero* [*simp*]: $\text{fds-truncate } m \ 0 = 0$
<proof>

lemma *fds-truncate-one* [*simp*]: $m > 0 \implies \text{fds-truncate } m \ 1 = 1$
<proof>

lemma *fds-truncate-const* [*simp*]: $m > 0 \implies \text{fds-truncate } m \ (\text{fds-const } c) = \text{fds-const } c$
<proof>

lemma *fds-truncate-truncate* [*simp*]: $\text{fds-truncate } m \ (\text{fds-truncate } n \ f) = \text{fds-truncate } (\min \ m \ n) \ f$
<proof>

lemma *fds-truncate-truncate'* [*simp*]: $\text{fds-truncate } m \ (\text{fds-truncate } m \ f) = \text{fds-truncate } m \ f$
<proof>

lemma *fds-truncate-shift* [*simp*]: $\text{fds-truncate } m \ (\text{fds-shift } a \ f) = \text{fds-shift } a \ (\text{fds-truncate } m \ f)$
<proof>

lemma *fds-truncate-add-strong*:
 $\text{fds-truncate } m \ (f + g :: 'a :: \text{monoid-add } \text{fds}) = \text{fds-truncate } m \ f + \text{fds-truncate } m \ g$
<proof>

lemma *fds-truncate-add*:
 $\text{fds-truncate } m \ (\text{fds-truncate } m \ f + \text{fds-truncate } m \ g :: 'a :: \text{monoid-add } \text{fds}) = \text{fds-truncate } m \ (f + g)$
<proof>

lemma *fds-truncate-mult*:
 $\text{fds-truncate } m \ (\text{fds-truncate } m \ f * \text{fds-truncate } m \ g) = \text{fds-truncate } m \ (f * g)$ (**is** ?A = ?B)
<proof>

lemma *fds-truncate-deriv*: $\text{fds-truncate } m \ (\text{fds-deriv } f) = \text{fds-deriv } (\text{fds-truncate } m \ f)$
<proof>

lemma *fds-truncate-integral*:
 $m > 0 \vee c = 0 \implies \text{fds-truncate } m \ (\text{fds-integral } c \ f) = \text{fds-integral } c \ (\text{fds-truncate } m \ f)$
<proof>

lemma *fds-truncate-power*: $\text{fds-truncate } m \ (\text{fds-truncate } m \ f \wedge n) = \text{fds-truncate } m \ (f \wedge n)$
<proof>

lemma *dirichlet-inverse-cong-simp*:

assumes $\bigwedge m. m > 0 \implies m \leq n \implies f m = f' m \ i = i' n = n'$

shows $\text{dirichlet-inverse } f \ i \ n = \text{dirichlet-inverse } f' \ i' \ n'$

<proof>

lemma *fds-truncate-cong*:

$(\bigwedge n. m > 0 \implies n > 0 \implies n \leq m \implies \text{fds-nth } f \ n = \text{fds-nth } f' \ n) \implies$

$\text{fds-truncate } m \ f = \text{fds-truncate } m \ f'$

<proof>

lemma *fds-truncate-inverse*:

$\text{fds-truncate } m \ (\text{inverse } (\text{fds-truncate } m \ (f :: 'a :: \text{field } \text{fds}))) = \text{fds-truncate } m$
 $(\text{inverse } f)$

<proof>

lemma *fds-truncate-divide*:

fixes $f \ g :: 'a :: \text{field } \text{fds}$

shows $\text{fds-truncate } m \ (\text{fds-truncate } m \ f / \text{fds-truncate } m \ g) = \text{fds-truncate } m \ (f$
 $/ \ g)$

<proof>

lemma *fds-truncate-ln*:

fixes $f :: 'a :: \text{real-normed-field } \text{fds}$

shows $\text{fds-truncate } m \ (\text{fds-ln } l \ (\text{fds-truncate } m \ f)) = \text{fds-truncate } m \ (\text{fds-ln } l \ f)$

<proof>

lemma *fds-truncate-exp*:

shows $\text{fds-truncate } m \ (\text{fds-exp } (\text{fds-truncate } m \ f)) = \text{fds-truncate } m \ (\text{fds-exp } f)$

<proof>

lemma *fds-eqI-truncate*:

assumes $\bigwedge m. m > 0 \implies \text{fds-truncate } m \ f = \text{fds-truncate } m \ g$

shows $f = g$

<proof>

4.10 Normed series

definition *fds-norm* $:: 'a :: \{\text{real-normed-div-algebra}\} \text{fds} \Rightarrow \text{real } \text{fds}$

where $\text{fds-norm } f = \text{fds } (\lambda n. \text{of-real } (\text{norm } (\text{fds-nth } f \ n)))$

lemma *fds-nth-norm [simp]*: $\text{fds-nth } (\text{fds-norm } f) \ n = \text{norm } (\text{fds-nth } f \ n)$

<proof>

lemma *fds-norm-1 [simp]*: $\text{fds-norm } 1 = 1$

<proof>

lemma *fds-nth-norm-mult-le*:

shows $\text{norm } (\text{fds-nth } (f * g) \ n) \leq \text{fds-nth } (\text{fds-norm } f * \text{fds-norm } g) \ n$

<proof>

lemma *fds-nth-norm-mult-nonneg* [*simp*]: $fds\text{-}nth\ (fds\text{-}norm\ f * fds\text{-}norm\ g)\ n \geq 0$

<proof>

4.11 Lifting a real series to a real algebra

definition *fds-of-real* :: $real\ fds \Rightarrow 'a :: \{real\text{-normed-algebra-1}\}$ *fds* **where**
 $fds\text{-of-real}\ f = fds\ (\lambda n. of\text{-real}\ (fds\text{-}nth\ f\ n))$

lemma *fds-nth-of-real* [*simp*]: $fds\text{-}nth\ (fds\text{-of-real}\ f)\ n = of\text{-real}\ (fds\text{-}nth\ f\ n)$
<proof>

lemma *fds-of-real-0* [*simp*]: $fds\text{-of-real}\ 0 = 0$

and *fds-of-real-1* [*simp*]: $fds\text{-of-real}\ 1 = 1$

and *fds-of-real-const* [*simp*]: $fds\text{-of-real}\ (fds\text{-const}\ c) = fds\text{-const}\ (of\text{-real}\ c)$

and *fds-of-real-minus* [*simp*]: $fds\text{-of-real}\ (-f) = -fds\text{-of-real}\ f$

and *fds-of-real-add* [*simp*]: $fds\text{-of-real}\ (f + g) = fds\text{-of-real}\ f + fds\text{-of-real}\ g$

and *fds-of-real-mult* [*simp*]: $fds\text{-of-real}\ (f * g) = fds\text{-of-real}\ f * fds\text{-of-real}\ g$

and *fds-of-real-deriv* [*simp*]: $fds\text{-of-real}\ (fds\text{-deriv}\ f) = fds\text{-deriv}\ (fds\text{-of-real}\ f)$
<proof>

lemma *fds-of-real-higher-deriv* [*simp*]:

$(fds\text{-deriv}\ \hat{\sim}\ n)\ (fds\text{-of-real}\ f) = fds\text{-of-real}\ ((fds\text{-deriv}\ \hat{\sim}\ n)\ f)$

<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* :: $('a :: \{nat\text{-power}, real\text{-normed-field}, banach\})\ fds \Rightarrow 'a \Rightarrow 'a$
where

$eval\text{-fds}\ f\ s = (\sum n. fds\text{-}nth\ f\ n / nat\text{-power}\ n\ s)$

lemma *eval-fds-eqI*:

assumes $(\lambda n. fds\text{-}nth\ f\ (Suc\ n) / nat\text{-power}\ (Suc\ n)\ s)\ sums\ L$

shows $eval\text{-fds}\ f\ s = L$

<proof>

definition *fds-converges* ::

$('a :: \{nat\text{-power}, real\text{-normed-field}, banach\})\ fds \Rightarrow 'a \Rightarrow bool$ **where**

$fds\text{-converges}\ f\ s \iff summable\ (\lambda n. fds\text{-}nth\ f\ n / nat\text{-power}\ n\ s)$

lemma *fds-converges-iff*:

$fds\text{-converges}\ f\ s \iff (\lambda n. fds\text{-}nth\ f\ n / nat\text{-power}\ n\ s)\ sums\ eval\text{-fds}\ f\ s$

<proof>

definition *fds-abs-converges* ::

(*'a* :: {*nat-power*, *real-normed-field*, *banach*}) *fds* \Rightarrow *'a* \Rightarrow *bool* **where**
fds-abs-converges *f s* \longleftrightarrow *summable* ($\lambda n.$ *norm* (*fds-nth* *f n* / *nat-power* *n s*))

lemma *fds-abs-converges-imp-converges* [*dest*, *intro*]:

fds-abs-converges *f s* \Longrightarrow *fds-converges* *f s*
(*proof*)

lemma *fds-converges-altdef*:

fds-converges *f s* \longleftrightarrow ($\lambda n.$ *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*) = ($\lambda.$ *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* = ($\lambda.$ *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*]: $eval-fds\ 1 = (\lambda-. 1)$
<proof>

lemma *eval-fds-truncate*: $eval-fds\ (fds-truncate\ n\ f)\ s = (\sum\ k=1..n.\ fds-nth\ f\ k\ /\ nat-power\ k\ s)$
<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* $P = \textit{The}\ Q$ **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* $f\ s \implies eval-fds\ (-f)\ s = -eval-fds\ f\ s$
<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*: $eval-fds f (of-nat k) = (\sum n. fds-nth f n / of-nat n \wedge k)$

<proof>

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

<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*:

$f 0 = 0 \implies (\sum n. fds-nth fds-zeta n / f n) = (\sum n. 1 / f n :: 'a :: real-normed-field)$

<proof>

lemma *fds-converges-shift [simp]*:

fixes $z :: 'a :: \{banach, nat-power-field, real-normed-field\}$

shows $fds-converges (fds-shift c f) z \iff fds-converges f (z - c)$

<proof>

lemma *fds-abs-converges-shift [simp]*:

fixes $z :: 'a :: \{banach, nat-power-field, real-normed-field\}$

shows $fds-abs-converges (fds-shift c f) z \iff fds-abs-converges f (z - c)$

<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 $fds-converges (fds-scale c f) z \iff fds-converges f (of-nat c * z)$

<proof>

lemma *fds-abs-converges-scale [simp]*:

fixes $z :: 'a :: \{\text{banach, nat-power-field, real-normed-field}\}$
assumes $c: c > 0$
shows $\text{fds-abs-converges (fds-scale } c \text{ } f) z \longleftrightarrow \text{fds-abs-converges } f \text{ (of-nat } c * z)$
 $\langle \text{proof} \rangle$

lemma *eval-fds-scale [simp]*:
fixes $z :: 'a :: \{\text{banach, nat-power-field, real-normed-field}\}$
assumes $c: c > 0$
shows $\text{eval-fds (fds-scale } c \text{ } f) z = \text{eval-fds } f \text{ (of-nat } c * z)$
 $\langle \text{proof} \rangle$

lemma *fds-abs-converges-integral*:
assumes $\text{fds-abs-converges } f \text{ } s$
shows $\text{fds-abs-converges (fds-integral } c \text{ } f) s$
 $\langle \text{proof} \rangle$

lemma *fds-abs-converges-ln*:
assumes $\text{fds-abs-converges (fds-deriv } f \text{ / } f) s$
shows $\text{fds-abs-converges (fds-ln } l \text{ } f) s$
 $\langle \text{proof} \rangle$

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* $:: \text{nat} \Rightarrow 'a :: \text{comm-ring-1}$ **where**
 $\text{moebius-mu } n =$
(if squarefree } n \text{ then } (-1) \wedge \text{card (prime-factors } n) \text{ else } 0)

lemma *abs-moebius-mu-le*: $\text{abs (moebius-mu } n :: 'a :: \{\text{linordered-idom}\}) \leq 1$
 $\langle \text{proof} \rangle$

lemma *moebius-commute*: $x * \text{moebius-mu } n = \text{moebius-mu } n * x$
 $\langle \text{proof} \rangle$

lemma *dirichlet-prod-moebius-commute*:
 $\text{dirichlet-prod } f \text{ moebius-mu} = \text{dirichlet-prod moebius-mu } f$
 $\langle \text{proof} \rangle$

lemma *fds-moebius-commute*: $x * \text{fds moebius-mu} = \text{fds moebius-mu} * x$

<proof>

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

lemma *minus-1-power-ring-neq-zero* [simp]: $(-1 :: 'a :: \text{ring-1})^n \neq 0$
<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 \longleftrightarrow \neg squarefree n*
<proof>

lemma *moebius-mu-not-squarefree* [simp]: *\neg squarefree n \implies 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 \vee n = 0 then 1 else if n = 1 then moebius-mu a else 0)
<proof>

lemma *moebius-mu-squarefree-eq*:
 squarefree n \implies 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 $(\sum d \mid d \text{ dvd } n. \text{moebius-mu } d) = (0 :: 'a :: \text{comm-ring-1})$
 ⟨proof⟩

lemma *sum-moebius-mu-divisors'*:
 $(\sum d \mid d \text{ dvd } n. \text{moebius-mu } d) = (\text{if } n = 1 \text{ then } 1 \text{ else } 0)$
 ⟨proof⟩

lemma *fds-zeta-times-moebius-mu*: $\text{fds-zeta} * \text{fds moebius-mu} = 1$
 ⟨proof⟩

lemma *fds-moebius-inverse-zeta*:
 $\text{fds moebius-mu} = \text{inverse } (\text{fds-zeta} :: 'a :: \text{field } \text{fds})$
 ⟨proof⟩

lemma *moebius-mu-formula-real*: $(\text{moebius-mu } n :: \text{real}) = \text{dirichlet-inverse } (\lambda-. 1) 1 n$
 ⟨proof⟩

lemma *moebius-mu-formula-int*: $\text{moebius-mu } n = \text{dirichlet-inverse } (\lambda-. 1 :: \text{int}) 1 n$
 ⟨proof⟩

lemma *moebius-mu-formula*: $\text{moebius-mu } n = \text{dirichlet-inverse } (\lambda-. 1) 1 n$
 ⟨proof⟩

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

interpretation *moebius-mu*:
multiplicative-function' moebius-mu $\lambda p k. \text{if } k = 1 \text{ then } -1 \text{ else } 0$ $\lambda-. -1$
 ⟨proof⟩

lemma *moebius-mu-2 [simp]*: $\text{moebius-mu } 2 = -1$
and *moebius-mu-3 [simp]*: $\text{moebius-mu } 3 = -1$
 ⟨proof⟩

lemma *moebius-mu-code [code]*:
 $\text{moebius-mu } n = \text{of-int } (\text{dirichlet-inverse } (\lambda-. 1 :: \text{int}) 1 n)$
 ⟨proof⟩

lemma *fds-moebius-inversion*: $f = \text{fds moebius-mu} * g \iff g = f * \text{fds-zeta}$
 ⟨proof⟩

lemma *moebius-inversion*:
assumes $\bigwedge n. n > 0 \implies g n = (\sum d \mid d \text{ dvd } n. f d) n > 0$
shows $f n = \text{dirichlet-prod moebius-mu } g n$

<proof>

lemma *fds-mangoldt*: $fds\ mangoldt = fds\ moebius\text{-}mu * fds\ (\lambda n. of\text{-}real\ (ln\ (real\ n)))$

<proof>

lemma *sum-divisors-moebius-mu-times-multiplicative*:

fixes $f :: nat \Rightarrow 'a :: \{comm\text{-}ring\text{-}1\}$

assumes *multiplicative-function* $f\ n > 0$

shows $(\sum d \mid d\ dvd\ n. moebius\text{-}mu\ d * f\ d) = (\prod_{p \in prime\text{-}factors\ n} 1 - f\ p)$

<proof>

lemma *completely-multiplicative-iff-inverse-moebius-mu*:

fixes $f :: nat \Rightarrow 'a :: \{comm\text{-}ring\text{-}1, ring\text{-}no\text{-}zero\text{-}divisors\}$

assumes *multiplicative-function* f

defines $g \equiv dirichlet\text{-}inverse\ f\ 1$

shows *completely-multiplicative-function* $f \longleftrightarrow$

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

<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* $(fds\text{-}nth\ f)$

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'*:

$mangoldt\ n * of\text{-}real\ (ln\ n) + (mangoldt\ \star\ mangoldt)\ n =$

$((moebius\text{-}mu\ \star\ (\lambda b. of\text{-}real\ (ln\ b)^2))\ n$

$\langle proof \rangle$
 $:: 'a :: \{comm-ring-1, real-algebra-1\}$ if $n > 0$

end

end

6 Euler's ϕ function

theory *More-Totient*

imports

Moebius-Mu

HOL-Number-Theory.Number-Theory

begin

lemma *fds-totient-times-zeta*:

$fds (\lambda n. of-nat (totient n)) :: 'a :: comm-semiring-1 * fds-zeta = fds of-nat$
 $\langle proof \rangle$

lemma *fds-totient-times-zeta'*: $fds totient * fds-zeta = fds id$

$\langle proof \rangle$

lemma *fds-totient*: $fds (\lambda n. of-nat (totient n)) = fds of-nat * fds moebius-mu$

$\langle proof \rangle$

lemma *totient-conv-moebius-mu*:

$int (totient n) = dirichlet-prod moebius-mu int n$

$\langle proof \rangle$

interpretation *totient*: *multiplicative-function totient*

$\langle proof \rangle$

lemma *even-prime-nat*: $prime p \implies even p \implies p = (2::nat)$

$\langle proof \rangle$

lemma *twopow-dvd-totient*:

fixes $n :: nat$

assumes $n > 0$

defines $k \equiv card \{p \in prime-factors n. odd p\}$

shows $2^k dvd totient n$

$\langle proof \rangle$

lemma *totient-conv-moebius-mu'*:

assumes $n > (0::nat)$

shows $real (totient n) = real n * (\sum d \mid d dvd n. moebius-mu d / real d)$
 $\langle proof \rangle$

lemma *totient-prime-power-Suc*:

assumes *prime p*

shows $\text{totient } (p \wedge \text{Suc } n) = p \wedge \text{Suc } n - p \wedge n$
(proof)

interpretation *totient: multiplicative-function'* $\text{totient } \lambda p k. p \wedge k - p \wedge (k - 1)$
 $\lambda p. p - 1$
(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* :: $\text{nat} \Rightarrow 'a :: \text{comm-ring-1}$ **where**
 $\text{liouville-lambda } n = (\text{if } n = 0 \text{ then } 0 \text{ else } (-1) \wedge \text{size } (\text{prime-factorization } n))$

interpretation *liouville-lambda: completely-multiplicative-function'* *liouville-lambda*
 $\lambda-. -1$
(proof)

lemma *liouville-lambda-prime [simp]: prime* $p \implies \text{liouville-lambda } p = -1$
(proof)

lemma *liouville-lambda-prime-power [simp]: prime* $p \implies \text{liouville-lambda } (p \wedge k)$
 $= (-1) \wedge k$
(proof)

lemma *liouville-lambda-squarefree: squarefree* $n \implies \text{liouville-lambda } n = \text{moebius-mu } n$
(proof)

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

lemma *liouville-lambda-power-even:*
 $n > 0 \implies \text{even } m \implies \text{liouville-lambda } (n \wedge m) = 1$
(proof)

lemma *liouville-lambda-power-odd:*
 $\text{odd } m \implies \text{liouville-lambda } (n \wedge m) = \text{liouville-lambda } n$
(proof)

lemma *liouville-lambda-power:*

liouville-lambda ($n \wedge m$) =

(if $n = 0 \wedge m > 0$ then 0 else if even m then 1 else *liouville-lambda* n)

<proof>

interpretation *squarefree: multiplicative-function'*

ind squarefree $\lambda p k$. 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) $\lambda p k$. 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 $\lambda p k$. if even k then 1 else 0 λ -. 0

<proof>

lemma *liouville-lambda-divisors-sum:*

$(\sum d \mid d \text{ dvd } n. \text{liouville-lambda } d) = \text{ind is-square } n$

<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 $n = (\sum d \mid d \wedge 2 \text{ dvd } n. \text{moebius-mu } (n \text{ div } d \wedge 2))$

<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 \Rightarrow nat \Rightarrow 'a **where**
divisor-sigma x n = $(\sum d \mid d \text{ dvd } n. \text{ nat-power } d \ x)$

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*
($\lambda d \text{ acc. if } d \text{ dvd } n \text{ then nat-power } d \ x + \text{acc else acc}$) 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 ($p \wedge k$) =
(if *nat-power* p x = 1 then *of-nat* (k + 1) else
($\text{nat-power } p \ x \wedge \text{Suc } k - 1$) / ($\text{nat-power } p \ x - 1$))
<proof>

lemma *divisor-sigma-prime-power-nat*:
assumes *prime* p

shows $\text{divisor-sigma } x (p \wedge k) = (\text{if } x = 0 \text{ then } \text{Suc } k \text{ else } (p \wedge (x * \text{Suc } k) - 1) \text{ div } (p \wedge x - 1))$

$\langle \text{proof} \rangle$

interpretation *divisor-sigma-field*:

multiplicative-function' $\text{divisor-sigma } (x :: 'a :: \{\text{field}, \text{nat-power}\})$

$\lambda p k. \text{if } \text{nat-power } p \ x = 1 \text{ then } \text{of-nat } (\text{Suc } k) \text{ else } (\text{nat-power } p \ x \wedge \text{Suc } k - 1) / (\text{nat-power } p \ x - 1)$

$\lambda p. \text{nat-power } p \ x + 1$

$\langle \text{proof} \rangle$

interpretation *divisor-sigma-real*:

multiplicative-function' $\text{divisor-sigma } (x :: \text{real})$

$\lambda p k. \text{if } x = 0 \text{ then } \text{of-nat } (\text{Suc } k) \text{ else } ((\text{real } p \ \text{powr } x) \wedge \text{Suc } k - 1) / (\text{real } p \ \text{powr } x - 1)$

$\lambda p. \text{real } p \ \text{powr } x + 1$

$\langle \text{proof} \rangle$

interpretation *divisor-sigma-nat*:

multiplicative-function' $\text{divisor-sigma } (x :: \text{nat})$

$\lambda p k. \text{if } x = 0 \text{ then } \text{Suc } k \text{ else } (p \wedge (\text{Suc } k * x) - 1) \text{ div } (p \wedge x - 1)$

$\lambda p. p \wedge x + 1$

$\langle \text{proof} \rangle$

lemma *divisor-sigma-prime*:

assumes *prime* p

shows $\text{divisor-sigma } x \ p = \text{nat-power } p \ x + 1$

$\langle \text{proof} \rangle$

8.2 The divisor-counting function

definition *divisor-count* $:: \text{nat} \Rightarrow \text{nat}$ **where**

$\text{divisor-count } n = \text{card } \{d. d \ \text{dvd } n\}$

lemma *divisor-count-0* [*simp*]: $\text{divisor-count } 0 = 0$

$\langle \text{proof} \rangle$

lemma *divisor-count-Suc-0* [*simp*]: $\text{divisor-count } (\text{Suc } 0) = 1$

$\langle \text{proof} \rangle$

lemma *divisor-sigma-0-left-nat*: $\text{divisor-sigma } 0 \ n = \text{divisor-count } n$

$\langle \text{proof} \rangle$

lemma *divisor-sigma-0-left*: $\text{divisor-sigma } 0 \ n = \text{of-nat } (\text{divisor-count } n)$

$\langle \text{proof} \rangle$

lemma *divisor-count-altdef*: $\text{divisor-count } n = \text{divisor-sigma } 0 \ n$

$\langle \text{proof} \rangle$

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

interpretation *divisor-count*: *multiplicative-function'* *divisor-count* $\lambda p k. \text{Suc } k$
 $\lambda-. 2$
 ⟨proof⟩

lemma *divisor-count-dvd-mono*:
assumes $a \text{ dvd } b \ b \neq 0$
shows $\text{divisor-count } a \leq \text{divisor-count } b$
 ⟨proof⟩

8.3 The divisor sum function

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

lemma *divisor-sum-0* [simp]: $\text{divisor-sum } 0 = 0$
 ⟨proof⟩

lemma *divisor-sum-Suc-0* [simp]: $\text{divisor-sum } (\text{Suc } 0) = \text{Suc } 0$
 ⟨proof⟩

lemma *divisor-sigma-1-left-nat*: $\text{divisor-sigma } (\text{Suc } 0) n = \text{divisor-sum } n$
 ⟨proof⟩

lemma *divisor-sigma-1-left*: $\text{divisor-sigma } 1 n = \text{of-nat } (\text{divisor-sum } n)$
 ⟨proof⟩

lemma *divisor-sum-altdef*: $\text{divisor-sum } n = \text{divisor-sigma } 1 n$
 ⟨proof⟩

interpretation *divisor-sum*:
multiplicative-function' *divisor-sum* $\lambda p k. (p \wedge \text{Suc } k - 1) \text{div } (p - 1) \lambda p. \text{Suc } p$
 ⟨proof⟩

lemma *divisor-sum-dvd-mono*:
assumes $a \text{ dvd } b \ b \neq 0$
shows $\text{divisor-sum } a \leq \text{divisor-sum } b$
 ⟨proof⟩

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

lemma *fds-divisor-count*: $\text{fds divisor-count} = \text{fds-zeta} \wedge 2$
 ⟨proof⟩

lemma *fds-shift-zeta-1*: $\text{fds-shift } 1 \text{ fds-zeta} = \text{fds of-nat}$
 ⟨proof⟩

lemma *fds-shift-zeta-Suc-0*: $\text{fds-shift } (\text{Suc } 0) \text{ fds-zeta} = \text{fds id}$
 ⟨proof⟩

lemma *fds-divisor-sum*: $\text{fds divisor-sum} = \text{fds-zeta} * \text{fds id}$
 ⟨proof⟩

lemma *fds-divisor-sum-eq-totient-times-d*: $\text{fds divisor-sum} = \text{fds totient} * \text{fds divisor-count}$
 ⟨proof⟩

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

lemma *inverse-divisor-sigma*:
fixes $a :: 'a :: \{\text{field, nat-power}\}$
shows $\text{inverse } (\text{fds } (\text{divisor-sigma } a)) = \text{fds-shift } a \text{ (fds moebius-mu)} * \text{fds moebius-mu}$
 ⟨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* :: $(\text{nat} \Rightarrow 'a :: \text{comm-monoid-add}) \Rightarrow \text{real} \Rightarrow 'a$ **where**
 $\text{sum-upto } f \ x = (\sum i \mid 0 < i \wedge \text{real } i \leq x. f \ i)$

lemma *sum-upto-altdef*: $\text{sum-upto } f \ x = (\sum i \in \{0 < .. \text{nat } \lfloor x \rfloor\}. f \ i)$
 ⟨proof⟩

lemma *sum-upto-0* [*simp*]: $\text{sum-upto } f \ 0 = 0$

<proof>

lemma *sum-upto-cong* [*cong*]:

$(\bigwedge n. n > 0 \implies f \ n = f' \ n) \implies n = n' \implies \text{sum-upto } f \ n = \text{sum-upto } f' \ n'$

<proof>

lemma *finite-Nats-le-real* [*simp,intro*]: $\text{finite } \{n. 0 < n \wedge \text{real } n \leq x\}$

<proof>

lemma *sum-upto-ind*: $\text{sum-upto } (\text{ind } P) \ x = \text{of-nat } (\text{card } \{n. n > 0 \wedge \text{real } n \leq x \wedge P \ n\})$

<proof>

lemma *sum-upto-sum-divisors*:

$\text{sum-upto } (\lambda n. \sum d \mid d \ \text{dvd} \ n. f \ n \ d) \ x = \text{sum-upto } (\lambda k. \text{sum-upto } (\lambda d. f \ (d * k)) \ (x / k)) \ x$

<proof>

lemma *sum-upto-dirichlet-prod*:

$\text{sum-upto } (\text{dirichlet-prod } f \ g) \ x = \text{sum-upto } (\lambda d. f \ d * \text{sum-upto } g \ (x / \text{real } d)) \ x$

<proof>

lemma *sum-upto-real*:

assumes $x \geq 0$

shows $\text{sum-upto } \text{real } x = \text{of-int } (\text{floor } x) * (\text{of-int } (\text{floor } x) + 1) / 2$

<proof>

lemma *summable-imp-convergent-sum-upto*:

assumes $\text{summable } (f :: \text{nat} \Rightarrow 'a :: \text{real-normed-vector})$

obtains c **where** $(\text{sum-upto } f \longrightarrow c)$ *at-top*

<proof>

9.2 The Hyperbola method

lemma *hyperbola-method-semiring*:

fixes $f \ g :: \text{nat} \Rightarrow 'a :: \text{comm-semiring-0}$

assumes $A \geq 0$ **and** $B \geq 0$ **and** $A * B = x$

shows $\text{sum-upto } (\text{dirichlet-prod } f \ g) \ x + \text{sum-upto } f \ A * \text{sum-upto } g \ B =$
 $\text{sum-upto } (\lambda n. f \ n * \text{sum-upto } g \ (x / \text{real } n)) \ A +$
 $\text{sum-upto } (\lambda n. \text{sum-upto } f \ (x / \text{real } n) * g \ n) \ B$

<proof>

lemma *hyperbola-method-semiring-sqrt*:

fixes $f \ g :: \text{nat} \Rightarrow 'a :: \text{comm-semiring-0}$

assumes $x \geq 0$

shows $\text{sum-upto } (\text{dirichlet-prod } f \ g) \ x + \text{sum-upto } f \ (\text{sqrt } x) * \text{sum-upto } g \ (\text{sqrt } x) =$

$sum-upto (\lambda n. f n * sum-upto g (x / real n)) (sqrt x) +$
 $sum-upto (\lambda n. sum-upto f (x / real n) * g n) (sqrt x)$
 <proof>

lemma *hyperbola-method*:

fixes $f g :: nat \Rightarrow 'a :: comm-ring$
assumes $A \geq 0 B \geq 0 A * B = x$
shows $sum-upto (dirichlet-prod f g) x =$
 $sum-upto (\lambda n. f n * sum-upto g (x / real n)) A +$
 $sum-upto (\lambda n. sum-upto f (x / real n) * g n) B -$
 $sum-upto f A * sum-upto g B$
 <proof>

lemma *hyperbola-method-sqrt*:

fixes $f g :: nat \Rightarrow 'a :: comm-ring$
assumes $x \geq 0$
shows $sum-upto (dirichlet-prod f g) x =$
 $sum-upto (\lambda n. f n * sum-upto g (x / real n)) (sqrt x) +$
 $sum-upto (\lambda n. sum-upto f (x / real n) * g n) (sqrt x) -$
 $sum-upto f (sqrt x) * sum-upto g (sqrt x)$
 <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 $a :: nat \Rightarrow 'a :: \{banach, real-normed-algebra\}$
fixes $f f' :: real \Rightarrow 'a$
fixes A
fixes $X :: real\ set$
fixes $x y :: real$
defines $A \equiv sum-upto a$
assumes $fin: finite X$
assumes $xy: 0 \leq y y < x$
assumes $deriv: \bigwedge z. z \in \{y..x\} - X \implies (f\ has-vector-derivative\ f'\ z)\ (at\ z)$
assumes $cont-f: continuous-on\ \{y..x\}\ f$

begin

lemma *partial-summation-strong*:

$((\lambda t. A t * f' t) \text{ has-integral } (A x * f x - A y * f y - (\sum n \in \text{real} - \{y < ..x\}. a n * f n))) \{y..x\}$
 <proof>

lemma *partial-summation-integrable-strong*:

$(\lambda t. A t * f' t) \text{ integrable-on } \{y..x\}$

and *partial-summation-strong'*:

$(\sum n \in \text{real} - \{y < ..x\}. a n * f n) =$

$A x * f x - A y * f y - \text{integral } \{y..x\} (\lambda t. A t * f' t)$

<proof>

end

context

fixes $a :: \text{nat} \Rightarrow 'a :: \{\text{banach, real-normed-algebra}\}$

fixes $f f' :: \text{real} \Rightarrow 'a$

fixes A

fixes $X :: \text{real set}$

fixes $x :: \text{real}$

defines $A \equiv \text{sum-upto } a$

assumes $\text{fin}: \text{finite } X$

assumes $x: x > 0$

assumes $\text{deriv}: \bigwedge z. z \in \{0..x\} - X \implies (f \text{ has-vector-derivative } f' z) \text{ (at } z)$

assumes $\text{cont-f}: \text{continuous-on } \{0..x\} f$

begin

lemma *partial-summation-sum-upto-strong*:

$((\lambda t. A t * f' t) \text{ has-integral } (A x * f x - \text{sum-upto } (\lambda n. a n * f n) x)) \{0..x\}$

<proof>

lemma *partial-summation-integrable-sum-upto-strong*:

$(\lambda t. A t * f' t) \text{ integrable-on } \{0..x\}$

and *partial-summation-sum-upto-strong'*:

$\text{sum-upto } (\lambda n. a n * f n) x =$

$A x * f x - \text{integral } \{0..x\} (\lambda t. A t * f' t)$

<proof>

end

end

11 Euler product expansions

theory *Euler-Products*

imports

HOL-Analysis.Analysis

Multiplicative-Function

begin

lemma *prime-factors-power-subset*:

prime-factors $(x \wedge n) \subseteq \text{prime-factors } x$
<proof>

lemma *prime-power-product-in-Pi*:

$(\lambda g. \prod_{p \in \{p. p \leq (n::nat) \wedge \text{prime } p\}. p \wedge g p}$
 $\in (\{p. p \leq n \wedge \text{prime } p\} \rightarrow_E UNIV) \rightarrow$
 $\{m. 0 < m \wedge \text{prime-factors } m \subseteq \{..n\}\})$
<proof>

lemma *inj-prime-power*: *inj-on* $(\lambda x. \text{fst } x \wedge \text{snd } x :: nat)$ $(\{a. \text{prime } a\} \times \{0<..\})$
<proof>

lemma *bij-betw-prime-powers*:

bij-betw $(\lambda g. \prod_{p \in \{p. p \leq n \wedge \text{prime } p\}. p \wedge g p}$ $(\{p. p \leq n \wedge \text{prime } p\} \rightarrow_E$
UNIV)
 $\{m. 0 < m \wedge \text{prime-factors } m \subseteq \{..(n::nat)\}\})$
<proof>

lemma

fixes $f :: nat \Rightarrow 'a :: \{\text{real-normed-field}, \text{banach}, \text{second-countable-topology}\}$

assumes *summable*: *summable* $(\lambda n. \text{norm } (f n))$

assumes *multiplicative-function* f

shows *abs-convergent-euler-product*:

abs-convergent-prod $(\lambda p. \text{if prime } p \text{ then } \sum n. f (p \wedge n) \text{ else } 1)$

and *euler-product-LIMSEQ*:

$(\lambda n. (\prod_{p \leq n. \text{if prime } p \text{ then } \sum n. f (p \wedge n) \text{ else } 1)) \longrightarrow (\sum n. f n))$

<proof>

lemma

fixes $f :: nat \Rightarrow 'a :: \{\text{real-normed-field}, \text{banach}, \text{second-countable-topology}\}$

assumes *summable*: *summable* $(\lambda n. \text{norm } (f n))$

assumes *completely-multiplicative-function* f

shows *abs-convergent-euler-product'*:

abs-convergent-prod $(\lambda p. \text{if prime } p \text{ then } \text{inverse } (1 - f p) \text{ else } 1)$

and *completely-multiplicative-summable-norm*:

$\bigwedge p. \text{prime } p \implies \text{norm } (f p) < 1$

and *euler-product-LIMSEQ'*:

$(\lambda n. (\prod_{p \leq n. \text{if prime } p \text{ then } \text{inverse } (1 - f p) \text{ else } 1)) \longrightarrow (\sum n. f$

$n)$

<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

lemma *frequently-eventually-frequently*:

frequently P $F \implies$ *eventually* Q $F \implies$ *frequently* $(\lambda x. P\ x \wedge Q\ x)$ F
 ⟨*proof*⟩

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 $G \neq \text{bot}$
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 $G \neq \text{bot}$
shows *frequently* P (*filtercomap* f F)
 ⟨*proof*⟩

lemma *frequently-going-to-at-topE*:

fixes $f :: 'a \Rightarrow \text{real}$
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 $k \rightarrow \infty$ ’.

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* $:: \text{real filter}$)) \iff
 $(\exists g. \forall n. P\ (g\ n) \wedge \text{filterlim}\ (\lambda n. f\ (g\ n))\ \text{at-top sequentially})$
 ⟨*proof*⟩

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

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

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

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

lemma *eval-fds-altdef*:
assumes *fds-abs-converges* f s
shows *eval-fds* f $s = (\sum a \ n. \text{fds-nth } f \ n / \text{nat-power } n \ s)$
 ⟨*proof*⟩

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

lemma *completely-multiplicative-function-divide-nat-power*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{nat-power}, \text{field}\}$
assumes *completely-multiplicative-function* f
shows *completely-multiplicative-function* $(\lambda n. f \ n / \text{nat-power } n \ s)$
 ⟨*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* $:: \text{real} \Rightarrow 'a \Rightarrow 'a$
assumes *real-power-nat-power*: $n > 0 \implies \text{real-power } (\text{real } n) \ c = \text{nat-power } n \ c$
assumes *real-power-1-right-aux*: $d > 0 \implies \text{real-power } d \ 1 = d *_{\mathbb{R}} 1$
assumes *real-power-add*: $d > 0 \implies \text{real-power } d \ (a + b) = \text{real-power } d \ a * \text{real-power } d \ b$
assumes *real-power-nonzero* [*simp*]: $d > 0 \implies \text{real-power } d \ a \neq 0$
assumes *norm-real-power*: $x > 0 \implies \text{norm } (\text{real-power } x \ c) = x \ \text{powr } (c \cdot 1)$
assumes *nat-power-of-real-aux*: $\text{nat-power } n \ (x *_{\mathbb{R}} 1) = ((\text{real } n \ \text{powr } x) *_{\mathbb{R}} 1)$
assumes *has-field-derivative-nat-power-aux*:
 $\bigwedge x::'a. n > 0 \implies \text{LIM } y \ \text{inf-class.inf}$

$(\text{Inf } (\text{principal } \{S. \text{ open } S \wedge x \in S\})) (\text{principal } (\text{UNIV} - \{x\})).$
 $(\text{nat-power } n \ y - \text{nat-power } n \ x - \ln (\text{real } n) *_R \text{nat-power } n \ x * (y$
 $- x)) /_R$
 $\text{norm } (y - x) \text{:> Inf } (\text{principal } \{S. \text{ open } S \wedge 0 \in S\})$
assumes *has-vector-derivative-real-power-aux*:
 $x > 0 \implies \text{filterlim } (\lambda y. (\text{real-power } y \ c - \text{real-power } x \ (c :: 'a) -$
 $(y - x) *_R (c * \text{real-power } x \ (c - 1))) /_R$
 $\text{norm } (y - x)) (\text{INF } S \in \{S. \text{ open } S \wedge 0 \in S\}. \text{principal } S) (\text{at } x)$
assumes *norm-nat-power*: $n > 0 \implies \text{norm } (\text{nat-power } n \ y) = \text{real } n \ \text{powr } (y \cdot$
 $1)$
begin

lemma *real-power-diff*: $d > 0 \implies \text{real-power } d \ (a - b) = \text{real-power } d \ a /$
 $\text{real-power } d \ b$
 $\langle \text{proof} \rangle$

end

lemma *real-power-1-right* [*simp*]: $d > 0 \implies \text{real-power } d \ 1 = \text{of-real } d$
 $\langle \text{proof} \rangle$

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

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

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

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

definition *real-power-real* :: *real* \Rightarrow *real* \Rightarrow *real* **where**
 $[\text{simp}]: \text{real-power-real} = (\text{powr})$

instance $\langle \text{proof} \rangle$

end

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

definition *nat-power-complex* :: *nat* \Rightarrow *complex* \Rightarrow *complex* **where**

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

definition *real-power-complex* :: *real* \Rightarrow *complex* \Rightarrow *complex* **where**

[*simp*]: *real-power-complex* = ($\lambda x y.$ *of-real x powr y*)

instance \langle *proof* \rangle

end

lemma *nat-power-of-real* [*simp*]:

nat-power n (of-real x :: 'a :: nat-power-normed-field) = *of-real (real n powr x)*

\langle *proof* \rangle

lemma *fds-abs-converges-of-real* [*simp*]:

fds-abs-converges (fds-of-real f)

(*of-real s :: 'a :: {nat-power-normed-field,banach}*) \longleftrightarrow *fds-abs-converges f s*

\langle *proof* \rangle

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)

\langle *proof* \rangle

lemma *fds-abs-summable-zeta-iff* [*simp*]:

fixes *s :: 'a :: {banach, nat-power-normed-field}*

shows *fds-abs-converges fds-zeta s* \longleftrightarrow *s \cdot 1 > (1 :: real)*

\langle *proof* \rangle

lemma *fds-abs-summable-zeta*:

(*s :: 'a :: {banach, nat-power-normed-field}*) \cdot 1 > 1 \implies *fds-abs-converges fds-zeta*

s

\langle *proof* \rangle

lemma *fds-abs-converges-moebius-mu*:

fixes *s :: 'a :: {banach, nat-power-normed-field}*

assumes *s \cdot 1 > 1*

shows *fds-abs-converges (fds moebius-mu) s*

\langle *proof* \rangle

definition *conv-abscissa*

:: *'a :: {nat-power, banach, real-normed-field, real-inner}* *fds* \Rightarrow *ereal* **where**

conv-abscissa f = (*INF s* \in {*s. fds-converges f s*}. *ereal (s \cdot 1)*)

definition *abs-conv-abscissa*

$:: 'a :: \{\text{nat-power}, \text{banach}, \text{real-normed-field}, \text{real-inner}\} \text{fds} \Rightarrow \text{ereal}$ **where**
 $\text{abs-conv-abscissa } f = (\text{INF } s \in \{s. \text{fds-abs-converges } f\ s\}. \text{ereal } (s \cdot 1))$

lemma *conv-abscissa-mono*:

assumes $\bigwedge s. \text{fds-converges } g\ s \implies \text{fds-converges } f\ s$
shows $\text{conv-abscissa } f \leq \text{conv-abscissa } g$
<proof>

lemma *abs-conv-abscissa-mono*:

assumes $\bigwedge s. \text{fds-abs-converges } g\ s \implies \text{fds-abs-converges } f\ s$
shows $\text{abs-conv-abscissa } f \leq \text{abs-conv-abscissa } g$
<proof>

class *dirichlet-series* = *euclidean-space* + *real-normed-field* + *nat-power-normed-field*

+

assumes *one-in-Basis*: $1 \in \text{Basis}$

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 $f :: 'a \text{fds}$
assumes $\text{fds-abs-converges } f\ z\ z \cdot 1 \leq z' \cdot 1$
shows $\text{fds-abs-converges } f\ z'$
<proof>

lemma *fds-abs-converges*:

assumes $s \cdot 1 > \text{abs-conv-abscissa } (f :: 'a \text{fds})$
shows $\text{fds-abs-converges } f\ s$
<proof>

lemma *fds-abs-diverges*:

assumes $s \cdot 1 < \text{abs-conv-abscissa } (f :: 'a \text{fds})$
shows $\neg \text{fds-abs-converges } f\ s$
<proof>

lemma *uniformly-Cauchy-eval-fds-aux*:

fixes $s0 :: 'a :: \text{dirichlet-series}$
assumes *bounded*: $B \text{seq } (\lambda n. \sum_{k \leq n}. \text{fds-nth } f\ k / \text{nat-power } k\ s0)$
assumes *B*: *compact* $B \bigwedge z. z \in B \implies z \cdot 1 > s0 \cdot 1$
shows *uniformly-Cauchy-on* $B (\lambda N z. \sum_{n \leq N}. \text{fds-nth } f\ n / \text{nat-power } n\ z)$
<proof>

lemma *uniformly-convergent-eval-fds-aux*:

assumes $Bseq (\lambda n. \sum_{k \leq n}. fds_nth\ f\ k / nat_power\ k\ (s0 :: 'a))$

assumes $B: compact\ B \wedge z. z \in B \implies z \cdot 1 > s0 \cdot 1$

shows $uniformly_convergent_on\ B (\lambda N\ z. \sum_{n \leq N}. fds_nth\ f\ n / nat_power\ n\ z)$

<proof>

lemma *uniformly-convergent-eval-fds-aux'*:

assumes $conv: fds_converges\ f\ (s0 :: 'a)$

assumes $B: compact\ B \wedge z. z \in B \implies z \cdot 1 > s0 \cdot 1$

shows $uniformly_convergent_on\ B (\lambda N\ z. \sum_{n \leq N}. fds_nth\ f\ n / nat_power\ n\ z)$

<proof>

lemma *bounded-partial-sums-imp-fps-converges*:

fixes $s0 :: 'a :: dirichlet_series$

assumes $Bseq (\lambda n. \sum_{k \leq n}. fds_nth\ f\ k / nat_power\ k\ s0)$ **and** $s \cdot 1 > s0 \cdot 1$

shows $fds_converges\ f\ s$

<proof>

theorem *fds-converges-Re-le*:

assumes $fds_converges\ f\ (s0 :: 'a)\ s \cdot 1 > s0 \cdot 1$

shows $fds_converges\ f\ s$

<proof>

lemma *fds-converges*:

assumes $s \cdot 1 > conv_abscissa\ (f :: 'a\ fds)$

shows $fds_converges\ f\ s$

<proof>

lemma *fds-diverges*:

assumes $s \cdot 1 < conv_abscissa\ (f :: 'a\ fds)$

shows $\neg fds_converges\ f\ s$

<proof>

theorem *fds-converges-imp-abs-converges*:

assumes $fds_converges\ (f :: 'a\ fds)\ s\ s' \cdot 1 > s \cdot 1 + 1$

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*: $conv_abscissa\ f = \infty \iff (\forall s. \neg fds_converges\ f\ s)$

<proof>

lemma *conv-abscissa-PInfI [intro]*: $(\bigwedge s. \neg fds_converges\ f\ s) \implies conv_abscissa\ f = \infty$

<proof>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

theorem *abs-conv-le-conv-abscissa-plus-1*: $\text{abs-conv-abscissa } (f :: 'a \text{ fds}) \leq \text{conv-abscissa } f + 1$
 $\langle \text{proof} \rangle$

lemma *uniformly-convergent-eval-fds*:
assumes $B: \text{compact } B \bigwedge z. z \in B \implies z \cdot 1 > \text{conv-abscissa } (f :: 'a \text{ fds})$
shows $\text{uniformly-convergent-on } B (\lambda N z. \sum n \leq N. \text{fds-nth } f n / \text{nat-power } n z)$
 $\langle \text{proof} \rangle$

corollary *uniformly-convergent-eval-fds'*:
assumes $B: \text{compact } B \bigwedge z. z \in B \implies z \cdot 1 > \text{conv-abscissa } (f :: 'a \text{ fds})$
shows $\text{uniformly-convergent-on } B (\lambda N z. \sum n < N. \text{fds-nth } f n / \text{nat-power } n z)$
 $\langle \text{proof} \rangle$

12.2 Derivative of a Dirichlet series

lemma *fds-converges-deriv-aux*:
assumes $\text{conv: fds-converges } f (s0 :: 'a)$ and $\text{gt: } s \cdot 1 > s0 \cdot 1$
shows $\text{fds-converges } (\text{fds-deriv } f) s$
 $\langle \text{proof} \rangle$

theorem
assumes $s \cdot 1 > \text{conv-abscissa } (f :: 'a \text{ fds})$
shows $\text{fds-converges-deriv: fds-converges } (\text{fds-deriv } f) s$
and $\text{has-field-derivative-eval-fds}$ [*derivative-intros*]:
 $(\text{eval-fds } f \text{ has-field-derivative } \text{eval-fds } (\text{fds-deriv } f) s) \text{ (at } s \text{ within } A)$
 $\langle \text{proof} \rangle$

lemmas $\text{has-field-derivative-eval-fds'}$ [*derivative-intros*] =
 $\text{DERIV-chain2}[OF \text{ has-field-derivative-eval-fds}]$

lemma *continuous-eval-fds* [*continuous-intros*]:
assumes $s \cdot 1 > \text{conv-abscissa } f$
shows $\text{continuous (at } s \text{ within } A) (\text{eval-fds } (f :: 'a :: \text{dirichlet-series } \text{fds}))$

<proof>

lemma *continuous-eval-fds'* [*continuous-intros*]:

fixes $f :: 'a :: \text{dirichlet-series fds}$

assumes *continuous (at s within A) g* $g \cdot 1 > \text{conv-abscissa } f$

shows *continuous (at s within A) (λx. eval-fds f (g x))*

<proof>

lemma *continuous-on-eval-fds* [*continuous-intros*]:

fixes $f :: 'a :: \text{dirichlet-series fds}$

assumes $A \subseteq \{s. s \cdot 1 > \text{conv-abscissa } f\}$

shows *continuous-on A (eval-fds f)*

<proof>

lemma *continuous-on-eval-fds'* [*continuous-intros*]:

fixes $f :: 'a :: \text{dirichlet-series fds}$

assumes *continuous-on A g* $g \cdot A \subseteq \{s. s \cdot 1 > \text{conv-abscissa } f\}$

shows *continuous-on A (λx. eval-fds f (g x))*

<proof>

lemma *conv-abscissa-deriv-le*:

fixes $f :: 'a \text{ fds}$

shows $\text{conv-abscissa } (\text{fds-deriv } f) \leq \text{conv-abscissa } f$

<proof>

lemma *abs-conv-abscissa-integral*:

fixes $f :: 'a \text{ fds}$

shows $\text{abs-conv-abscissa } (\text{fds-integral } a \ f) = \text{abs-conv-abscissa } f$

<proof>

lemma *abs-conv-abscissa-ln*:

$\text{abs-conv-abscissa } (\text{fds-ln } l \ (f :: 'a :: \text{dirichlet-series fds})) =$

$\text{abs-conv-abscissa } (\text{fds-deriv } f / f)$

<proof>

lemma *abs-conv-abscissa-deriv*:

fixes $f :: 'a \text{ fds}$

shows $\text{abs-conv-abscissa } (\text{fds-deriv } f) = \text{abs-conv-abscissa } f$

<proof>

lemma *abs-conv-abscissa-higher-deriv*:

$\text{abs-conv-abscissa } ((\text{fds-deriv } \sim^n) \ f) = \text{abs-conv-abscissa } (f :: 'a :: \text{dirichlet-series fds})$

<proof>

lemma *conv-abscissa-higher-deriv-le*:

$\text{conv-abscissa } ((\text{fds-deriv } \sim^n) \ f) \leq \text{conv-abscissa } (f :: 'a :: \text{dirichlet-series fds})$

<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 $s \cdot 1 > \text{conv-abscissa } f$
shows $\text{eval-fds (fds-deriv } f) s = \text{deriv (eval-fds } f) s$
 ⟨proof⟩

lemma *eval-fds-higher-deriv*:

assumes $(s :: 'a :: \text{dirichlet-series}) \cdot 1 > \text{conv-abscissa } f$
shows $\text{eval-fds ((fds-deriv } \hat{\hat{n}}) f) s = (\text{deriv } \hat{\hat{n}}) (\text{eval-fds } f) s$
 ⟨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*: $\text{summable } (\lambda n. \text{norm } (f\ n / \text{nat-power } n\ s))$
 $\text{summable } (\lambda n. \text{norm } (g\ n / \text{nat-power } n\ s))$
shows *summable-dirichlet-prod*: $\text{summable } (\lambda n. \text{norm } (\text{dirichlet-prod } f\ g\ n / \text{nat-power } n\ s))$
and *suminf-dirichlet-prod*:
 $(\sum n. \text{dirichlet-prod } f\ g\ n / \text{nat-power } n\ s) =$
 $(\sum n. f\ n / \text{nat-power } n\ s) * (\sum n. g\ n / \text{nat-power } n\ s)$
 ⟨proof⟩

lemma

fixes *f g* :: *nat* ⇒ *real*
fixes *s* :: *real*
assumes $f\ 0 = 0 \ g\ 0 = 0$
assumes *summable*: $\text{summable } (\lambda n. \text{norm } (f\ n / \text{real } n\ \text{powr } s))$
 $\text{summable } (\lambda n. \text{norm } (g\ n / \text{real } n\ \text{powr } s))$
shows *summable-dirichlet-prod-real*: $\text{summable } (\lambda n. \text{norm } (\text{dirichlet-prod } f\ g\ n / \text{real } n\ \text{powr } s))$
and *suminf-dirichlet-prod-real*:
 $(\sum n. \text{dirichlet-prod } f\ g\ n / \text{real } n\ \text{powr } s) =$
 $(\sum n. f\ n / \text{real } n\ \text{powr } s) * (\sum n. g\ n / \text{real } n\ \text{powr } s)$
 ⟨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 $\text{fds-abs-converges } (f * g) s$
 $\langle \text{proof} \rangle$

lemma $\text{fds-abs-converges-power}$:

fixes $s :: 'a :: \{\text{nat-power, real-normed-field, banach, second-countable-topology}\}$
shows $\text{fds-abs-converges } f s \implies \text{fds-abs-converges } (f \wedge n) s$
 $\langle \text{proof} \rangle$

lemma $\text{fds-abs-converges-prod}$:

fixes $s :: 'a :: \{\text{nat-power, real-normed-field, banach, second-countable-topology}\}$
shows $(\bigwedge x. x \in A \implies \text{fds-abs-converges } (f x) s) \implies \text{fds-abs-converges } (\text{prod } f A) s$
 $\langle \text{proof} \rangle$

lemma $\text{abs-conv-abcissa-mult-le}$:

$\text{abs-conv-abcissa } (f * g :: 'a :: \text{dirichlet-series fds}) \leq$
 $\text{max } (\text{abs-conv-abcissa } f) (\text{abs-conv-abcissa } g)$
 $\langle \text{proof} \rangle$

lemma $\text{abs-conv-abcissa-mult-leI}$:

$\text{abs-conv-abcissa } (f :: 'a :: \text{dirichlet-series fds}) \leq d \implies$
 $\text{abs-conv-abcissa } g \leq d \implies \text{abs-conv-abcissa } (f * g) \leq d$
 $\langle \text{proof} \rangle$

lemma $\text{abs-conv-abcissa-shift [simp]}$:

$\text{abs-conv-abcissa } (\text{fds-shift } c f) = \text{abs-conv-abcissa } (f :: 'a :: \text{dirichlet-series fds})$
 $+ c \cdot 1$
 $\langle \text{proof} \rangle$

lemma eval-fds-mult :

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

lemma eval-fds-power :

fixes $s :: 'a :: \{\text{nat-power, real-normed-field, banach, second-countable-topology}\}$
assumes $\text{fds-abs-converges } f s$
shows $\text{eval-fds } (f \wedge n) s = \text{eval-fds } f s \wedge n$
 $\langle \text{proof} \rangle$

lemma eval-fds-prod :

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

lemma eval-fds-inverse :

fixes $s :: 'a :: \{\text{nat-power, real-normed-field, banach, second-countable-topology}\}$
assumes $\text{fds-abs-converges } f s \text{ fds-abs-converges } (\text{inverse } f) s \text{ fds-nth } f 1 \neq 0$

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 $ereal (s \cdot 1) > abs-conv-abscissa f$

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]*:

$A \subseteq \{z. Re z > conv-abscissa f\} \implies eval-fds f \text{ holomorphic-on } A$

<proof>

lemma *analytic-fds-eval [holomorphic-intros]*:

assumes $A \subseteq \{z. Re z > conv-abscissa f\}$

shows $eval-fds f \text{ analytic-on } A$
<proof>

lemma *conv-abscissa-0 [simp]*:

$conv-abscissa (0 :: 'a :: dirichlet-series fds) = -\infty$

<proof>

lemma *abs-conv-abscissa-0 [simp]*:

$abs-conv-abscissa (0 :: 'a :: dirichlet-series fds) = -\infty$

<proof>

lemma *conv-abscissa-1 [simp]*:

$conv-abscissa (1 :: 'a :: dirichlet-series fds) = -\infty$

<proof>

lemma *abs-conv-abscissa-1 [simp]*:

$abs-conv-abscissa (1 :: 'a :: dirichlet-series fds) = -\infty$

<proof>

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

$conv-abscissa (fds-const (c :: 'a :: dirichlet-series)) = -\infty$

<proof>

lemma *abs-conv-abscissa-const [simp]*:

$abs-conv-abscissa (fds-const (c :: 'a :: dirichlet-series)) = -\infty$

<proof>

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

$conv-abscissa (numeral n :: 'a :: dirichlet-series fds) = -\infty$

<proof>

lemma *abs-conv-abscissa-numeral [simp]*:

$abs-conv-abscissa (numeral n :: 'a :: dirichlet-series fds) = -\infty$

<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 $\bigwedge x. x \in A \Rightarrow \text{abs-conv-abscissa } (f\ x :: 'a :: \text{dirichlet-series fds}) \leq d$

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 $\bigwedge x. x \in A \Rightarrow \text{conv-abscissa } (f\ x :: 'a :: \text{dirichlet-series fds}) \leq d$

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 $\bigwedge x. x \in A \Rightarrow \text{abs-conv-abscissa } (f\ x :: 'a :: \text{dirichlet-series fds}) \leq d$

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*
 \Rightarrow
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 \neq 0

shows *abs-conv-abscissa* (*fds-deriv* *f* / *f*) \leq *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*) $N > 0$ $s \cdot 1 \geq c$ $c \geq s0 \cdot 1$

shows *summable* ($\lambda n. \text{fds-nth } f (n + N) / \text{nat-power } (n + N) s$)

summable ($\lambda n. \text{norm } (\text{fds-nth } f (n + N)) / \text{nat-power } (n + N) c$)

and *norm* ($\sum n. \text{fds-nth } f (n + N) / \text{nat-power } (n + N) s$) \leq

$(\sum n. \text{norm } (\text{fds-nth } f (n + N)) / \text{nat-power } (n + N) c) / \text{nat-power}$

$N (s \cdot 1 - c)$

<proof>

lemma *eval-fds-zeroD-aux*:

fixes *h* :: '*a* *fds*

assumes *conv: fds-abs-converges* *h* (*s0* :: '*a*)

assumes *freq: frequently* ($\lambda s. \text{eval-fds } h s = 0$) ($(\lambda s. s \cdot 1)$ *going-to at-top*)

shows *h* = 0

<proof>

lemma *eval-fds-zeroD*:

fixes *h* :: '*a* *fds*

assumes *conv: conv-abscissa* *h* $< \infty$

assumes *freq: frequently* ($\lambda s. \text{eval-fds } h s = 0$) ($(\lambda s. s \cdot 1)$ *going-to at-top*)

shows *h* = 0

<proof>

lemma *eval-fds-eqD*:

fixes *f g* :: '*a* *fds*

assumes *conv: conv-abscissa* *f* $< \infty$ *conv-abscissa* *g* $< \infty$

assumes *eq: frequently* ($\lambda s. \text{eval-fds } f s = \text{eval-fds } g s$) ($(\lambda s. s \cdot 1)$ *going-to at-top*)

shows *f* = *g*

<proof>

end

12.5 Limit at infinity

lemma *eval-fds-at-top-tail-bound*:

fixes $f :: 'a :: \text{dirichlet-series fds}$

assumes $c: \text{ereal } c > \text{abs-conv-abscissa } f$

defines $B \equiv (\sum n. \text{norm } (\text{fds-nth } f (n+2)) / \text{real } (n+2) \text{ powr } c) * 2 \text{ powr } c$

assumes $s: s \cdot 1 \geq c$

shows $\text{norm } (\text{eval-fds } f s - \text{fds-nth } f 1) \leq B / 2 \text{ powr } (s \cdot 1)$

<proof>

lemma *tendsto-eval-fds-Re-at-top*:

assumes $\text{conv-abscissa } (f :: 'a :: \text{dirichlet-series fds}) \neq \infty$

assumes $\text{lim: filterlim } (\lambda x. S x \cdot 1) \text{ at-top } F$

shows $((\lambda x. \text{eval-fds } f (S x)) \longrightarrow \text{fds-nth } f 1) F$

<proof>

lemma *tendsto-eval-fds-Re-at-top'*:

assumes $\text{conv-abscissa } (f :: \text{complex fds}) \neq \infty$

shows $\text{uniform-limit UNIV } (\lambda \sigma t. \text{eval-fds } f (\text{of-real } \sigma + \text{of-real } t * i)) (\lambda \cdot \text{fds-nth } f 1) \text{ at-top}$

<proof>

lemma *tendsto-eval-fds-Re-going-to-at-top*:

assumes $\text{conv-abscissa } (f :: 'a :: \text{dirichlet-series fds}) \neq \infty$

shows $((\lambda s. \text{eval-fds } f s) \longrightarrow \text{fds-nth } f 1) ((\lambda s. s \cdot 1) \text{ going-to at-top})$

<proof>

lemma *tendsto-eval-fds-Re-going-to-at-top'*:

assumes $\text{conv-abscissa } (f :: \text{complex fds}) \neq \infty$

shows $((\lambda s. \text{eval-fds } f s) \longrightarrow \text{fds-nth } f 1) (\text{Re going-to at-top})$

<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 $f :: 'a :: \text{dirichlet-series fds}$

assumes $\text{conv-abscissa } f < \infty \wedge f \neq 0$

shows $\text{eventually } (\lambda s. \text{fds-converges } f s \wedge \text{eval-fds } f s \neq 0) ((\lambda s. s \cdot 1) \text{ going-to at-top})$

<proof>

12.6 Normed series

lemma *fds-converges-norm-iff [simp]*:

fixes $s :: 'a :: \{\text{nat-power-normed-field}, \text{banach}\}$

shows $\text{fds-converges } (\text{fds-norm } f) (s \cdot 1) \longleftrightarrow \text{fds-abs-converges } f s$

<proof>

lemma *fds-abs-converges-norm-iff [simp]*:

fixes $s :: 'a :: \{\text{nat-power-normed-field}, \text{banach}\}$

shows $\text{fds-abs-converges } (\text{fds-norm } f) (s \cdot 1) \longleftrightarrow \text{fds-abs-converges } f s$
 ⟨proof⟩

lemma *fds-converges-norm-iff'*:

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

lemma *fds-abs-converges-norm-iff'*:

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

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

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

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

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

lemma

fixes $f g :: 'a :: \text{dirichlet-series}$ *fds*
assumes $\text{fds-abs-converges } (\text{fds-norm } f) s$ $\text{fds-abs-converges } (\text{fds-norm } g) s$
shows $\text{fds-abs-converges-norm-mult: } \text{fds-abs-converges } (\text{fds-norm } (f * g)) s$
and $\text{eval-fds-norm-mult-le:}$
 $\text{eval-fds } (\text{fds-norm } (f * g)) s \leq \text{eval-fds } (\text{fds-norm } f) s * \text{eval-fds } (\text{fds-norm } g) s$
 ⟨proof⟩

lemma *eval-fds-norm-nonneg*:

assumes $\text{fds-abs-converges } (\text{fds-norm } f) s$
shows $\text{eval-fds } (\text{fds-norm } f) s \geq 0$
 ⟨proof⟩

lemma

fixes $f :: 'a :: \text{dirichlet-series}$ *fds*
assumes $\text{fds-abs-converges } (\text{fds-norm } f) s$
shows $\text{fds-abs-converges-norm-power: } \text{fds-abs-converges } (\text{fds-norm } (f \wedge n)) s$
and $\text{eval-fds-norm-power-le:}$
 $\text{eval-fds } (\text{fds-norm } (f \wedge n)) s \leq \text{eval-fds } (\text{fds-norm } f) s \wedge n$
 ⟨proof⟩

12.7 Logarithms of Dirichlet series

lemma *eventually-gt-ereal-at-top*: $c \neq \infty \implies \text{eventually } (\lambda x. \text{ereal } x > c) \text{ at-top}$
 ⟨proof⟩

lemma *eval-fds-log-deriv*:

fixes $s :: 'a :: \text{dirichlet-series}$

assumes $\text{fds-nth } f \ 1 \neq 0$ $s \cdot 1 > \text{abs-conv-abscissa } f$

$s \cdot 1 > \text{abs-conv-abscissa } (\text{fds-deriv } f / f)$

assumes $\text{eval-fds } f \ s \neq 0$

shows $\text{eval-fds } (\text{fds-deriv } f / f) \ s = \text{eval-fds } (\text{fds-deriv } f) \ s / \text{eval-fds } f \ s$

<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 $s0 :: \text{ereal}$

assumes $\text{nz}: \bigwedge s. \text{Re } s > s0 \implies \text{eval-fds } f \ s \neq 0$ $\text{fds-nth } f \ 1 \neq 0$

assumes $l: \text{exp } l = \text{fds-nth } f \ 1 \ ((g \circ \text{of-real}) \longrightarrow l) \text{ at-top}$

assumes $g: \bigwedge s. \text{Re } s > s0 \implies \text{exp } (g \ s) = \text{eval-fds } f \ s$

assumes *holo-g*: $g \text{ holomorphic-on } \{s. \text{Re } s > s0\}$

assumes $\text{ereal } (\text{Re } s) > s0$

assumes $s0 \geq \text{abs-conv-abscissa } f$ **and** $s0 \geq \text{abs-conv-abscissa } (\text{fds-deriv } f / f)$

shows $\text{eval-fds } (\text{fds-ln } l \ f) \ s = g \ s$

<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 $s0 :: \text{ereal}$

assumes $\text{ereal } (\text{Re } s) > s0$

assumes $s0 \geq \text{abs-conv-abscissa } f$ **and** $s0 \geq \text{abs-conv-abscissa } (\text{fds-deriv } f / f)$

and $\text{nz}: \bigwedge s. \text{Re } s > s0 \implies \text{eval-fds } f \ s \neq 0$ $\text{fds-nth } f \ 1 \neq 0$

assumes $l: \text{exp } l = \text{fds-nth } f \ 1$

shows $\text{exp } (\text{eval-fds } (\text{fds-ln } l \ f) \ s) = \text{eval-fds } f \ s$

<proof>

lemma *fds-ln-completely-multiplicative*:

fixes $f :: 'a :: \text{dirichlet-series}$ fds

assumes *completely-multiplicative-function* $(\text{fds-nth } f)$

assumes $\text{fds-nth } f \ 1 \neq 0$

shows $\text{fds-ln } l \ f = \text{fds } (\lambda n. \text{if } n = 1 \text{ then } l \text{ else } \text{fds-nth } f \ n * \text{mangoldt } n /_R \ln n)$

<proof>

lemma *eval-fds-ln-completely-multiplicative-strong*:

fixes $s :: 'a :: \text{dirichlet-series}$ **and** $l :: 'a$ **and** $f :: 'a \ \text{fds}$ **and** $g :: \text{nat} \Rightarrow 'a$

defines $h \equiv \text{fds } (\lambda n. \text{fds-nth } (\text{fds-ln } l \ f) \ n * g \ n)$

assumes *fds-abs-converges* $h \ s$

assumes *completely-multiplicative-function* $(\text{fds-nth } f)$ **and** $\text{fds-nth } f \ 1 \neq 0$

shows $(\lambda(p,k). (fds\text{-}nth\ f\ p\ /\ nat\text{-}power\ p\ s) \wedge Suc\ k * g\ (p \wedge Suc\ k) /\ of\text{-}nat\ (Suc\ k))$
abs-summable-on $(\{p.\ prime\ p\} \times UNIV)$ **(is ?th1)**
and $eval\text{-}fds\ h\ s = l * g\ 1 + (\sum_a(p, k) \in \{p.\ prime\ p\} \times UNIV. (fds\text{-}nth\ f\ p\ /\ nat\text{-}power\ p\ s) \wedge Suc\ k * g\ (p \wedge Suc\ k) /\ of\text{-}nat\ (Suc\ k))$
(is ?th2)
 $\langle proof \rangle$

lemma *eval-fds-ln-completely-multiplicative:*

fixes $s :: 'a :: dirichlet\text{-}series$ **and** $l :: 'a$ **and** $f :: 'a\ fds$
assumes *completely-multiplicative-function* $(fds\text{-}nth\ f)$ **and** $fds\text{-}nth\ f\ 1 \neq 0$
assumes $s \cdot 1 > abs\text{-}conv\text{-}abscissa\ (fds\text{-}deriv\ f\ /\ f)$
shows $(\lambda(p,k). (fds\text{-}nth\ f\ p\ /\ nat\text{-}power\ p\ s) \wedge Suc\ k /\ of\text{-}nat\ (Suc\ k))$
abs-summable-on $(\{p.\ prime\ p\} \times UNIV)$ **(is ?th1)**
and $eval\text{-}fds\ (fds\text{-}ln\ l\ f)\ s =$
 $l + (\sum_a(p, k) \in \{p.\ prime\ p\} \times UNIV. (fds\text{-}nth\ f\ p\ /\ nat\text{-}power\ p\ s) \wedge Suc\ k /\ of\text{-}nat\ (Suc\ k))$ **(is ?th2)**
 $\langle proof \rangle$

12.8 Exponential and logarithm

lemma *summable-fds-exp-aux:*

assumes $fds\text{-}nth\ f'\ 1 = (0 :: 'a :: real\text{-}normed\text{-}algebra\text{-}1)$
shows $summable\ (\lambda k. fds\text{-}nth\ (f' \wedge k)\ n\ /_R\ fact\ k)$
 $\langle proof \rangle$

lemma

fixes $f :: 'a :: dirichlet\text{-}series\ fds$
assumes *fds-abs-converges* $f\ s$
shows *fds-abs-converges-exp:* *fds-abs-converges* $(fds\text{-}exp\ f)\ s$
and *eval-fds-exp:* $eval\text{-}fds\ (fds\text{-}exp\ f)\ s = exp\ (eval\text{-}fds\ f\ s)$
 $\langle proof \rangle$

lemma *fds-exp-add:*

fixes $f :: 'a :: dirichlet\text{-}series\ fds$
shows $fds\text{-}exp\ (f + g) = fds\text{-}exp\ f * fds\text{-}exp\ g$
 $\langle proof \rangle$

lemma *fds-exp-minus:*

fixes $f :: 'a :: dirichlet\text{-}series\ fds$
shows $fds\text{-}exp\ (-f) = inverse\ (fds\text{-}exp\ f)$
 $\langle proof \rangle$

lemma *abs-conv-abscissa-exp:*

fixes $f :: 'a :: dirichlet\text{-}series\ fds$
shows $abs\text{-}conv\text{-}abscissa\ (fds\text{-}exp\ f) \leq abs\text{-}conv\text{-}abscissa\ f$
 $\langle proof \rangle$

lemma *fds-deriv-exp [simp]:*

fixes $f :: 'a :: \text{dirichlet-series fds}$
shows $\text{fds-deriv} (\text{fds-exp } f) = \text{fds-exp } f * \text{fds-deriv } f$
 $\langle \text{proof} \rangle$

lemma fds-exp-ln-strong :
fixes $f :: 'a :: \text{dirichlet-series fds}$
assumes $\text{fds-nth } f (\text{Suc } 0) \neq 0$
shows $\text{fds-exp} (\text{fds-ln } l f) = \text{fds-const} (\text{exp } l / \text{fds-nth } f (\text{Suc } 0)) * f$
 $\langle \text{proof} \rangle$

lemma fds-exp-ln [simp] :
fixes $f :: 'a :: \text{dirichlet-series fds}$
assumes $\text{exp } l = \text{fds-nth } f (\text{Suc } 0)$
shows $\text{fds-exp} (\text{fds-ln } l f) = f$
 $\langle \text{proof} \rangle$

lemma fds-ln-exp [simp] :
fixes $f :: 'a :: \text{dirichlet-series fds}$
assumes $l = \text{fds-nth } f (\text{Suc } 0)$
shows $\text{fds-ln } l (\text{fds-exp } f) = f$
 $\langle \text{proof} \rangle$

12.9 Euler products

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

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

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

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

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

lemma
fixes $s :: 'a :: \{\text{nat-power-normed-field, banach, euclidean-space}\}$
assumes $s \cdot 1 > 1$
shows *euler-product-fds-zeta*:
 $(\lambda n. \prod_{p \leq n} \text{if prime } p \text{ then inverse } (1 - 1 / \text{nat-power } p \text{ } s) \text{ else } 1)$
 $\longrightarrow \text{eval-fds } \text{fds-zeta } s \text{ (is ?th1)}$
and *eval-fds-zeta-nonzero*: $\text{eval-fds } \text{fds-zeta } s \neq 0$
 $\langle \text{proof} \rangle$

lemma *fds-primew-subseries-euler-product-cm*:
fixes $f :: 'a :: \text{dirichlet-series } \text{fds}$
assumes *completely-multiplicative-function* (*fds-nth* f) *prime* p
assumes $s \cdot 1 > \text{abs-conv-abscissa } f$
shows $\text{eval-fds } (\text{fds-primew-subseries } p \text{ } f) \text{ } s = 1 / (1 - \text{fds-nth } f \text{ } p / \text{nat-power } p \text{ } s)$
 $\langle \text{proof} \rangle$

12.10 Non-negative Dirichlet series

lemma *nonneg-Reals-sum*: $(\bigwedge x. x \in A \implies f \text{ } x \in \mathbf{R}_{\geq 0}) \implies \text{sum } f \text{ } A \in \mathbf{R}_{\geq 0}$
 $\langle \text{proof} \rangle$

locale *nonneg-dirichlet-series* =
fixes $f :: 'a :: \text{dirichlet-series } \text{fds}$
assumes *nonneg-coeffs-aux*: $n > 0 \implies \text{fds-nth } f \text{ } n \in \mathbf{R}_{\geq 0}$
begin

lemma *nonneg-coeffs*: $\text{fds-nth } f \text{ } n \in \mathbf{R}_{\geq 0}$
 $\langle \text{proof} \rangle$

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*]:
 $c \in \mathbb{R}_{\geq 0} \implies \text{nonneg-dirichlet-series } (\text{fds-const } c)$
⟨*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*: $c \geq \text{abs-conv-abscissa } f$
assumes $g: \bigwedge s. s \in A \implies \text{Re } s > c \implies g \ s = \text{eval-fds } f \ s$
assumes g *holomorphic-on* A *open* A $c \in A$
shows $\exists x. x < c \wedge \text{fds-abs-converges } f \ (\text{of-real } x)$
 $\langle \text{proof} \rangle$

theorem *pringsheim-landau*:
fixes $c :: \text{real}$ **and** $f :: \text{complex fds}$
assumes *nonneg-dirichlet-series* f
assumes *abscissa*: $\text{abs-conv-abscissa } f = c$
assumes $g: \bigwedge s. s \in A \implies \text{Re } s > c \implies g \ s = \text{eval-fds } f \ s$
assumes g *holomorphic-on* A *open* A $c \in A$
shows *False*
 $\langle \text{proof} \rangle$

corollary *entire-continuation-imp-abs-conv-abscissa-MInfty*:
assumes *nonneg-dirichlet-series* f
assumes $c: c \geq \text{abs-conv-abscissa } f$
assumes $g: \bigwedge s. \text{Re } s > c \implies g \ s = \text{eval-fds } f \ s$
assumes *holo*: g *holomorphic-on* *UNIV*
shows $\text{abs-conv-abscissa } f = -\infty$
 $\langle \text{proof} \rangle$

12.11 Convergence of the ζ and Möbius μ series

lemma *fds-abs-summable-zeta-real-iff* [*simp*]:
 $\text{fds-abs-converges } \text{fds-zeta } s \longleftrightarrow s > (1 :: \text{real})$
 $\langle \text{proof} \rangle$

lemma *fds-abs-summable-zeta-real*: $s > (1 :: \text{real}) \implies \text{fds-abs-converges } \text{fds-zeta } s$
 $\langle \text{proof} \rangle$

lemma *fds-abs-converges-moebius-mu-real*:
assumes $s > (1 :: \text{real})$
shows $\text{fds-abs-converges } (\text{fds moebius-mu}) \ s$
 $\langle \text{proof} \rangle$

12.12 Application to the Möbius μ function

lemma *inverse-squares-sums'*: $(\lambda n. 1 / \text{real } n \wedge 2) \text{ sums } (\text{pi} \wedge 2 / 6)$
 $\langle \text{proof} \rangle$

lemma *norm-summable-moebius-over-square*:
 $\text{summable } (\lambda n. \text{norm } (\text{moebius-mu } n / \text{real } n \wedge 2))$
 $\langle \text{proof} \rangle$

lemma *summable-moebius-over-square*:
 $\text{summable } (\lambda n. \text{moebius-mu } n / \text{real } n \wedge 2)$

<proof>

lemma *moebius-over-square-sums*: $(\lambda n. \text{moebius-mu } n / n^2) \text{ sums } (6 / \pi^2)$
<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* $(\lambda n. 1 / (\text{real } (\text{Suc } n) + d) ^ 2)$

$(\sum n. 1 / (\text{real } (\text{Suc } n) + d) ^ 2) \leq 1 / d$

<proof>

lemma *moebius-sum-tail-bound*:

assumes $d > 0$

shows $\text{abs } (\sum n. \text{moebius-mu } (\text{Suc } n + d) / \text{real } (\text{Suc } n + d) ^ 2) \leq 1 / d$ (**is**
 $\text{abs } ?S \leq -$)

<proof>

lemma *sum-upto-inverse-bound*:

sum-upto $(\lambda i. 1 / \text{real } i) x \geq 0$

eventually $(\lambda x. \text{sum-upto } (\lambda i. 1 / \text{real } i) x \leq \ln x + 13 / 22)$ *at-top*

<proof>

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

<proof>

lemma

defines $G \equiv (\lambda x::\text{real}. (\sum n. \text{moebius-mu } (n + \text{Suc } (\text{nat } \lfloor x \rfloor))) / (n + \text{Suc } (\text{nat } \lfloor x \rfloor))) ^ 2 :: \text{real}$

shows *moebius-sum-tail-bound'*: $\bigwedge t. t \geq 2 \implies |G t| \leq 1 / (t - 1)$

and *moebius-sum-tail-bigo*: $G \in O(\lambda t. 1 / t)$

<proof>

13.2 Summatory totient function

theorem *summatory-totient-asymptotics*:

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

theorem *summatory-totient-asymptotics'*:

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

theorem *summatory-totient-asymptotics''*:

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

13.3 Asymptotic distribution of squarefree numbers

lemma *le-sqrt-iff*: $x \geq 0 \implies x \leq \text{sqrt } y \iff x^2 \leq y$
 ⟨proof⟩

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

theorem *squarefree-asymptotics'*:

($\lambda x. \text{card } \{n. \text{real } n \leq x \wedge \text{squarefree } n\} = o(\lambda x. 6 / \text{pi}^2 * x) + o O(\lambda x. \text{sqrt } x)$)
 ⟨proof⟩

theorem *squarefree-asymptotics''*:

($\lambda x. \text{card } \{n. \text{real } n \leq x \wedge \text{squarefree } n\} \sim_{[\text{at-top}]} (\lambda x. 6 / \text{pi}^2 * x)$)
 ⟨proof⟩

13.4 The hyperbola method

lemma *hyperbola-method-bigo*:

fixes $f g :: \text{nat} \Rightarrow 'a :: \text{real-normed-field}$

assumes ($\lambda x. \text{sum-upto } (\lambda n. f n * \text{sum-upto } g (x / \text{real } n)) (\text{sqrt } x) - R x \in O(b)$)

assumes ($\lambda x. \text{sum-upto } (\lambda n. \text{sum-upto } f (x / \text{real } n) * g n) (\text{sqrt } x) - S x \in O(b)$)

assumes ($\lambda x. \text{sum-upto } f (\text{sqrt } x) * \text{sum-upto } g (\text{sqrt } x) - T x \in O(b)$)

shows ($\lambda x. \text{sum-upto } (\text{dirichlet-prod } f g) x - (R x + S x - T x) \in O(b)$)

⟨proof⟩

lemma *frac-le-1*: $\text{frac } x \leq 1$

⟨proof⟩

lemma *ln-minus-ln-floor-bound*:

assumes $x \geq 2$

shows $\ln x - \ln (\text{floor } x) \in \{0..<1 / (x - 1)\}$

⟨proof⟩

lemma *ln-minus-ln-floor-bigo*:

($\lambda x::\text{real}. \ln x - \ln (\text{floor } x) \in O(\lambda x. 1 / x)$)

<proof>

lemma *divisor-count-asymptotics-aux:*

$(\lambda x. \text{sum-upto } (\lambda n. \text{sum-upto } (\lambda-. 1) (x / \text{real } n)) (\text{sqrt } x) - (x * \ln x / 2 + \text{euler-mascheroni} * x)) \in O(\text{sqrt})$

<proof>

lemma *sum-upto-sqrt-bound:*

assumes $x: x \geq (0 :: \text{real})$

shows $\text{norm } ((\text{sum-upto } (\lambda-. 1) (\text{sqrt } x))^2 - x) \leq 2 * \text{norm } (\text{sqrt } x)$

<proof>

lemma *summatory-divisor-count-asymptotics:*

$(\lambda x. \text{sum-upto } (\lambda n. \text{real } (\text{divisor-count } n)) x - (x * \ln x + (2 * \text{euler-mascheroni} - 1) * x)) \in O(\text{sqrt})$

<proof>

theorem *summatory-divisor-count-asymptotics':*

$(\lambda x. \text{sum-upto } (\lambda n. \text{real } (\text{divisor-count } n)) x) = o$
 $(\lambda x. x * \ln x + (2 * \text{euler-mascheroni} - 1) * x) + o O(\lambda x. \text{sqrt } x)$

<proof>

theorem *summatory-divisor-count-asymptotics'':*

$\text{sum-upto } (\lambda n. \text{real } (\text{divisor-count } n)) \sim[\text{at-top}] (\lambda x. x * \ln x)$

<proof>

lemma *summatory-divisor-eq:*

$\text{sum-upto } (\lambda n. \text{real } (\text{divisor-count } n)) (\text{real } m) = \text{card } \{(n,d). n \in \{0 <.. m\} \wedge d \text{ dvd } n\}$

<proof>

context

fixes $M :: \text{nat} \Rightarrow \text{real}$

defines $M \equiv \lambda m. \text{card } \{(n,d). n \in \{0 <.. m\} \wedge d \text{ dvd } n\} / \text{card } \{0 <.. m\}$

begin

lemma *mean-divisor-count-asymptotics:*

$(\lambda m. M m - (\ln m + 2 * \text{euler-mascheroni} - 1)) \in O(\lambda m. 1 / \text{sqrt } m)$

<proof>

theorem *mean-divisor-count-asymptotics':*

$M = o (\lambda x. \ln x + 2 * \text{euler-mascheroni} - 1) + o O(\lambda x. 1 / \text{sqrt } x)$

<proof>

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

<proof>

end

13.5 The asymptotic distribution of coprime pairs

context

fixes $A :: \text{nat} \Rightarrow (\text{nat} \times \text{nat}) \text{ set}$

defines $A \equiv (\lambda N. \{(m,n) \in \{1..N\} \times \{1..N\}. \text{coprime } m \ n\})$

begin

lemma *coprime-pairs-asymptotics*:

$(\lambda N. \text{real } (\text{card } (A \ N)) - 6 / \text{pi}^2 * (\text{real } N)^2) \in O(\lambda N. \text{real } N * \ln (\text{real } N))$
 $\langle \text{proof} \rangle$

theorem *coprime-pairs-asymptotics'*:

$(\lambda N. \text{real } (\text{card } (A \ N))) =_o (\lambda N. 6 / \text{pi}^2 * (\text{real } N)^2) +_o O(\lambda N. \text{real } N * \ln (\text{real } N))$
 $\langle \text{proof} \rangle$

theorem *coprime-pairs-asymptotics''*:

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

theorem *coprime-probability-tendsto*:

$(\lambda N. \text{card } (A \ N) / \text{card } (\{1..N\} \times \{1..N\})) \longrightarrow 6 / \text{pi}^2$
 $\langle \text{proof} \rangle$

end

13.6 The asymptotics of the number of Farey fractions

definition *farey-fractions* :: $\text{nat} \Rightarrow \text{rat set}$ **where**

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

lemma *Fract-eq-coprime*:

assumes $\text{Rat.Fract } a \ b = \text{Rat.Fract } c \ d \ b > 0 \ d > 0 \ \text{coprime } a \ b \ \text{coprime } c \ d$

shows $a = c \ b = d$

$\langle \text{proof} \rangle$

lemma *quotient-of-split*:

$P (\text{quotient-of } q) = (\forall a \ b. b > 0 \longrightarrow \text{coprime } a \ b \longrightarrow q = \text{Rat.Fract } a \ b \longrightarrow P (a, b))$

$\langle \text{proof} \rangle$

lemma *quotient-of-split-asm*:

$P (\text{Rat.quotient-of } q) = (\neg(\exists a \ b. b > 0 \wedge \text{coprime } a \ b \wedge q = \text{Rat.Fract } a \ b \wedge \neg P (a, b)))$

$\langle \text{proof} \rangle$

lemma *farey-fractions-bij*:

bij-betw $(\lambda(a,b). \text{Rat.Fract } (\text{int } a) (\text{int } b))$

$\{(a,b) | a \ b. 0 < a \wedge a \leq b \wedge b \leq N \wedge \text{coprime } a \ b\} (\text{farey-fractions } N)$

$\langle \text{proof} \rangle$

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

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

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

theorem *card-farey-fractions-asymptotics''*:
 $(\lambda N. \text{real} (\text{card} (\text{farey-fractions } N))) \sim[\text{at-top}] (\lambda N. 3 / \pi^2 * (\text{real } N)^2)$
 ⟨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'* :: $\text{nat} \Rightarrow (\text{nat} \times \text{nat}) \text{ list}$ **where**
 $\text{prime-factorization-nat}' n =$
 $\text{let } ps = \text{prime-factorization-nat } n$
 $\text{in } \text{map } (\lambda p. (p, \text{length} (\text{filter } ((=) p) ps) - 1)) (\text{remdups-adj } (\text{sort } ps))$

lemma *set-prime-factorization-nat'*:
 $\text{set} (\text{prime-factorization-nat}' n) = (\lambda p. (p, \text{multiplicity } p n - 1)) \text{ ' prime-factors } n$
 ⟨proof⟩

lemma *distinct-prime-factorization-nat' [simp]*: $\text{distinct} (\text{prime-factorization-nat}' n)$
 ⟨proof⟩

lemmas (**in** *multiplicative-function'*) *efficient-code'* =
 $\text{efficient-code} [\text{of } \lambda-. \text{prime-factorization-nat}' n n \text{ for } n,$
 $\text{OF } \text{set-prime-factorization-nat}' \text{ distinct-prime-factorization-nat}']$

14.1 Möbius μ function

definition *moebius-mu-aux* :: $\text{nat} \Rightarrow (\text{unit} \Rightarrow \text{nat list}) \Rightarrow \text{int}$ **where**
moebius-mu-aux *n ps* =
 (if $n \neq 0 \wedge \neg 4 \text{ dvd } n \wedge \neg 9 \text{ dvd } n$ 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* :: $\text{unit} \Rightarrow \text{nat list}$
defines *ps* \equiv *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 (\lambda-. prime-factorization-nat n))*
 ⟨*proof*⟩

value *moebius-mu 12578972695257* :: *int*

14.2 Euler's ϕ function

primrec *totient-aux1* :: $\text{nat} \Rightarrow \text{nat list} \Rightarrow \text{nat}$ **where**
totient-aux1 *n []* = *n*
 | *totient-aux1 n (p # ps)* = *totient-aux1 (n - n div p) ps*

lemma *of-nat-totient-aux1*:
assumes $\bigwedge p. p \in \text{set } ps \implies \text{prime } p \wedge p \in \text{set } ps \implies p \text{ dvd } n \text{ distinct } ps$
shows $\text{real } (\text{totient-aux1 } n \text{ } ps) = \text{real } n * (\prod_{p \in \text{set } ps} 1 - 1 / \text{real } p)$
 ⟨*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* :: $\text{nat} \Rightarrow \text{nat list}$ **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* :: $(\text{nat} \times \text{nat}) \text{ list} \Rightarrow \text{nat}$ **where**
totient-aux2 xs = $(\prod_{(p,k) \leftarrow xs} p \wedge k * (p - 1))$

```

lemma totient-conv-totient-aux2:
  assumes  $n \neq 0$ 
  assumes set xs = ( $\lambda p. (p, \text{multiplicity } p \ n - 1)$ ) ' prime-factors n
  assumes distinct xs
  shows  $\text{totient } n = \text{totient-aux2 } xs$ 
  <proof>

lemma totient-code1:  $\text{totient } n = \text{totient-aux1 } n \ (\text{prime-factors-nat } n)$ 
  <proof>

lemma totient-code2:  $\text{totient } n = (\text{if } n = 0 \text{ then } 0 \text{ else } \text{totient-aux2 } (\text{prime-factorization-nat}' n))$ 
  <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.